URCHIN: PARAMETRIZING ATTENTION FOR LONG CONTEXT RECALL

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformers remain the dominant architecture for recall as they cache every key-value pair from context; however, the memory requirements for this cache limit scalability to ultra-long sequences. This has motivated work into finite-state relaxations of attention, such as linear attention and state-space models (SSMs). The test-time regression perspective offers insight into such layers by viewing attention as an online nonparametric regressor from keys to values, and proposing linear key-value regressors as parametric alternatives. Linear-state methods, such as Mamba-2, Gated DeltaNet, and Kimi Delta Attention, have seen widespread success; however, while competitive with attention on language modeling, they fall short on long-context recall. Recent dual-state SSMs such as Raven address this by introducing nonlinearity into the retrieval process, yet they sit outside the test-time regression framework, leaving it unclear how to extend or improve these methods in a principled manner. In this work, we view softmax attention as a nonparametric Gaussian mixture model over the key-value space, with a mixture component centered at each observed key-value pair. By parameterizing this density estimate with a fixed number of mixture components, this framework yields a design space of fixed-state models with the same softmax nonlinearity as attention. We unify past dual-state SSMs as particular instantiations of this framework, and use our framework to derive Urchin, a new sequence layer for long-context recall and language modeling. Urchin demonstrates strong recall across tasks and reaches near-perfect passkey retrieval on 1M-token contexts, a 500x extrapolation beyond its training length.

1 INTRODUCTION

In pursuit of lifelong in-context learning, where models must accumulate experience and act over millions or billions of tokens, we need to develop scalable alternatives to attention. While Transformers have ushered in a new era of language modeling, attention retains every past key-value pair in a cache, whose size scales with the context length. This limits attention's scalability to truly long-context applications. Recent progress in linear attention (Katharopoulos et al., 2020) and state-space models (SSMs) (Gu et al., 2022; Gu & Dao, 2024) aim to replace this ever-growing cache with a fixed-size state. While these methods have nearly closed the gap to attention on language modeling, they still fall behind attention on in-context learning over long sequences. In particular, they fall short on recall, the simplest subproblem of in-context learning (Arora et al., 2024a). In this work, we focus on developing a class of models with a fixed-size state that can nevertheless perform recall over arbitrarily long contexts. .

Developing fixed-state alternatives requires building a theoretical framework explaining attention's role as a strong sequence modeling primitive. Test-time regression (Wang et al., 2025) offers one such perspective, by viewing attention as an online regressor that learns to map keys to their associated values at test-time. This perspective naturally promotes linear regressors as fixed-state alternatives to attention, unifying past linear attention mechanisms (Dao & Gu, 2024; Yang et al., 2025) and laying the foundation for stronger methods (Kimi Team, 2025)[1]. However, linear-state methods like linear attention and SSMs lack the nonlinearity required to prevent keys from competing with each

[1] See Appendix E for related work.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

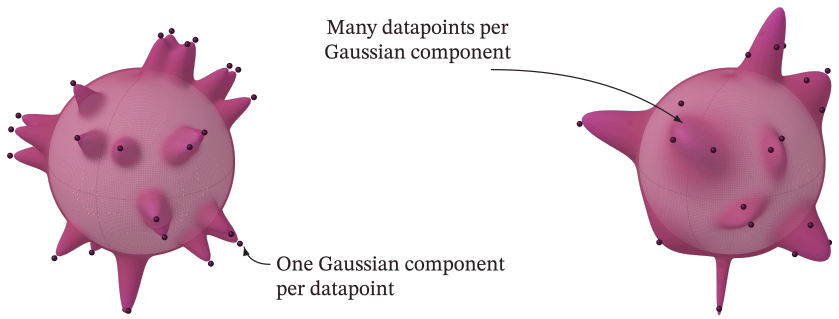


Figure 1: (Left) Attention can be viewed as modeling the probability density of key-value pairs (points) with a Gaussian mixture component centered at each point. (Right) We introduce a class of fixed-state models which parametrically model the density with a fixed number of components.

other. This undermines their ability to retrieve a specific key-value pair from the context (McDermott et al., 2026b). By contrast, attention resembles a Nadaraya-Watson kernel regressor (Nadaraya, 1964; Watson, 1964; Goel & Bartlett, 2024), wherein the exponential kernel improves recall precision by up-weighting highly similar query-key interactions. Recent methods such as Raven (Afzal et al., 2026) and Gated Slot Attention (GSA) (Zhang et al., 2024) keep the exponential weighting of softmax attention but with a fixed-state. Nonetheless, these layers are designed mechanistically, sitting outside of theoretical frameworks like test-time regression; this makes it unclear how to improve upon or generalize these layers.

In this work, we view attention from a probabilistic perspective by exploring the connection between Nadaraya-Watson estimators and nonparametric Gaussian mixture models (GMMs) (Bishop, 2006). We interpret attention as implicitly learning a nonparametric density estimate of the key-value space with a GMM. This places a separate Gaussian component at the location of each past key-value pair. We use this perspective to derive a family of parametric, fixed-state alternatives to attention. Rather than let the number of components grow freely with the context length, we parametrize the density estimator by instead using a fixed number of components. At test-time, the mean of each Gaussian component is fit to the set of key-value pairs in a streaming manner. We design a recall-focused learning objective around three principles: 1) fit to incoming data, 2) retain previous information from context, and 3) encourage parameter plasticity over time. Different implementations of these principles lead to different sequence modeling layers; this perspective therefore gives us a framework by which to design sequence layers. For example, we show that our framework unifies previous "dual-state" SSMs like ABC (Peng et al., 2022), GSA, and Raven, as particular instantiations.

With our framework, we derive Urchin, our novel fixed-state attention alternative.[2] Urchin is derived from maximum a posteriori (MAP) estimation using the conditional likelihood of the values given the keys. This contrasts with previous layer instantiations such as ABC, GSA, and Raven, which we show implicitly perform maximum likelihood estimation over the joint key-value pair. By providing an additional prior over the parameter space, we re-introduce gating mechanisms in this framework. Overall, Urchin demonstrates strong performance across synthetic and real-world recall tasks, while remaining competitive with SSMs and other fixed-state models at language modeling. Urchin achieves near-perfect accuracy on RULER's S-NIAH-1 task (Hsieh et al., 2024) at 1 million context length across model sizes—a **500x** extrapolation beyond its training length—and $> 90\%$ accuracy even at **10 million**-token contexts. Furthermore, we show that Urchin layers can be used in combination with state-of-the-art SSMs such as KDA to achieve strong performance on both language modeling and long-context recall. Finally, we dive deeper into understanding how these language models perform recall, via further analysis and ablations of RULER-style Needle-in-a-Haystack evaluations.

[2]Urchin is named after its ability to store and recall needles in a haystack. (See Figure 1 for instance.)

2 Viewing Attention as a Mixture of Gaussian Memories

In this section, we show that attention can be interpreted as a nonparametric Gaussian mixture model, with one component per key-value pairs (Bishop, 2006). Softmax attention (Vaswani et al., 2017) operation processes a sequence of input tokens $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t \in \mathbb{R}^d$. Each token is projected to a query, a key and a value:

$$\boldsymbol{q}_t = \mathbf{W}_q \boldsymbol{x}_t \in \mathbb{R}^{d_k}, \qquad \boldsymbol{k}_t = \mathbf{W}_k \boldsymbol{x}_t \in \mathbb{R}^{d_k}, \qquad \boldsymbol{v}_t = \mathbf{W}_v \boldsymbol{x}_t \in \mathbb{R}^{d_v}, \tag{1}$$

with trainable matrices $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$. Softmax attention compares the current query to all past keys and computes the output $\boldsymbol{y}_t$ as a weighted sum of the past values,

$$\boldsymbol{y}_t = \sum_{i=1}^{t} \frac{\exp\left(\boldsymbol{q}_t^\top \boldsymbol{k}_i / \sqrt{d_k}\right)}{\sum_{j=1}^{t} \exp\left(\boldsymbol{q}_t^\top \boldsymbol{k}_j / \sqrt{d_k}\right)} \boldsymbol{v}_i \in \mathbb{R}^{d_v}. \tag{2}$$

This requires attention layers to maintain an ever-growing cache of t-many key-value pairs.

In this work, we study attention through the lens of probabilistic modeling. We treat keys and values as jointly random variables K and V. Each key-value pair $(\boldsymbol{k}_i, \boldsymbol{v}_i)$ is a low-dimensional observation at time i that gives partial information about the context. Attention uses these observations to predict the value associated with a given query. Specifically, we show that attention can be viewed as a nonparametric Gaussian mixture model over key-value pairs, and that its output is the expected value under this model,

$$\boldsymbol{y}_t = \mathbb{E}\big[\mathsf{V} \mid \mathsf{K} = \boldsymbol{q}_t\big] \in \mathbb{R}^{d_v}. \tag{3}$$

Here, the query is an observation at the current time step that contains only the key coordinates. In the associative memory literature, the query is often seen as a corrupted memory (i.e., a noisy or sparsified memory) that needs to be recovered (Hopfield, 1982). To compute this expectation, we need to model (K, V). Softmax attention models (K, V) nonparametrically by storing each observation $(\boldsymbol{k}_i, \boldsymbol{v}_i)$ as its own component, or "memory", of a Gaussian mixture model. The observation is used directly as the mean of the component,

$$\mathcal{N}\big([\boldsymbol{k}, \boldsymbol{v}];\, [\boldsymbol{k}_i, \boldsymbol{v}_i],\, \sigma^2 \mathbf{I}_{d_k + d_v}\big), \tag{4}$$

where $\mathcal{N}(\cdot\,; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the Gaussian density with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. Each component of a Gaussian mixture model also carries a mixture weight, which sets how much of the density it contributes. We introduce the random variable $c_t \in \{1, \ldots, t\}$ for the index of the memory that a pair is drawn from, so that the mixture weights are its prior, $P(c_t = j) = \pi_j / \sum_{i=1}^{t} \pi_i$ for weights $\pi_j > 0$. Overall, softmax attention uses the probabilistic model

$$P(\boldsymbol{k}, \boldsymbol{v}) = \sum_{j=1}^{t} P(c_t = j)\, P(\boldsymbol{k}, \boldsymbol{v} \mid c_t = j), \qquad P(\boldsymbol{k}, \boldsymbol{v} \mid c_t = j) = \mathcal{N}\big([\boldsymbol{k}, \boldsymbol{v}];\, [\boldsymbol{k}_j, \boldsymbol{v}_j],\, \sigma^2 \mathbf{I}_{d_k + d_v}\big). \tag{5}$$

Under this model, the expectation in Eq. 3 is softmax attention,

$$\mathbb{E}\big[\mathsf{V} \mid \mathsf{K} = \boldsymbol{q}_t\big] = \sum_{j=1}^{t} \frac{\pi_j \exp\left(\boldsymbol{q}_t^\top \boldsymbol{k}_j / \sigma^2 - \|\boldsymbol{k}_j\|_2^2 / 2\sigma^2\right)}{\sum_{i=1}^{t} \pi_i \exp\left(\boldsymbol{q}_t^\top \boldsymbol{k}_i / \sigma^2 - \|\boldsymbol{k}_i\|_2^2 / 2\sigma^2\right)} \boldsymbol{v}_j \tag{6}$$

with $\pi_j = \exp(\|\boldsymbol{k}_j\|_2^2 / 2\sigma^2)$ and $\sigma^2 = \sqrt{d_k}$ (Appendix H). Eq. 6 also explains the role of key norms in attention. A key with a large norm is a memory with a large amplitude in the mixture. Normalizing keys and queries (QK-Norm) is the choice of uniform mixture weights, $\pi_i = \pi_j$. Scaling the query scales every logit by the same factor, so once the key norms are absorbed into π_j, the query norm acts as the variance σ^2: a larger query makes the mixture sharper.

3 Learning Parametric Memories

In this section we introduce our formal learning framework for fixed-state sequence modeling (Section 3.1), introduce our new instantiation Urchin as a sequence model layer (Section 3.2), and discuss how the framework generalizes prior methods (Section 3.3).

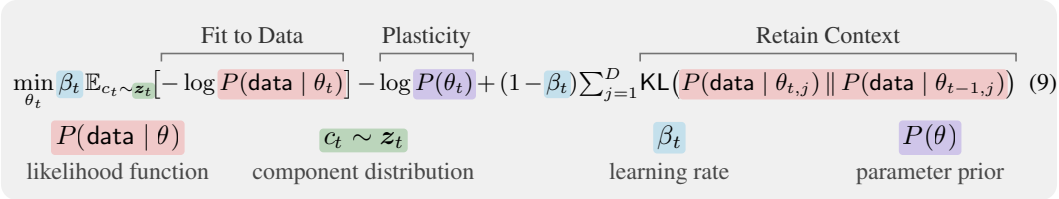


Figure 2: In our framework, fixed-state layers are a choice of four things, based on the three desiderata of fitting to data, maintaining plasticity, and retaining context.

3.1 A Framework for Parametric Attention Methods

Softmax attention maintains strong recall because it gives each observation its own memory or Gaussian component in the mixture. For long-context inference this is infeasible, so we must use a state of fixed size. Under this framework, we parametrize this probability density estimate by using constant D-many components in the mixture. At each time step, we fit the means of each component $\theta_t = \{\theta_{t,j}\}_{j \le D}$ with $\theta_{t,j} \in \mathbb{R}^{d_k + d_v}$ to the key-value observations. The mixture is defined similarly as above, except over a constant number of components instead of evergrowing t as in

$$P(\boldsymbol{k}, \boldsymbol{v} \mid \theta_t) = \sum_{j=1}^{D} P(c_t = j)\, \mathcal{N}\big([\boldsymbol{k}, \boldsymbol{v}];\, \theta_{t,j},\, \sigma^2 \mathbf{I}_{d_k + d_v}\big), \tag{7}$$

Here, $c_t \in \{1, \ldots, D\}$ represents the index for the component in a mixture. With mixture weights $P(c_t = j) \propto \exp(\|\theta_{t,j}^{(1)}\|_2^2 / 2\sigma^2)$, the same choice that recovered softmax attention in Eq. 6, the output is

$$\boldsymbol{y}_t = \mathbb{E}\big[\mathsf{V} \mid \mathsf{K} = \boldsymbol{q}_t\big] = \sum_{j=1}^{D} P(c_t = j \mid \mathsf{K} = \boldsymbol{q}_t, \theta_t^{(1)})\; \theta_{t,j}^{(2)} = \theta_t^{(2)}\, \mathrm{Softmax}\big(\theta_t^{(1)} \boldsymbol{q}_t / \sigma^2\big), \tag{8}$$

where $\theta_t^{(1)} \in \mathbb{R}^{D \times d_k}$ and $\theta_t^{(2)} \in \mathbb{R}^{d_v \times D}$ denote all key-side and value-side parameters at time t, respectively. This parametric Gaussian Mixture Model resembles the same readout as sliding window attention, as well as, dual-state SSMs like ABC, GSA, and Raven. Compared to linear-state methods like linear attention variants or traditional SSMs, these perform a nonlinear function of the query. This allows for sharper retrieval for highly similar keys, a crucial element in recall.

Our main interest lies in how to best optimize the parameters θ_t at each time step. In our general objective, we use three desiderata to learn from context: 1) fit to the incoming key-value observation, 2) remember past observations, and 3) encourage plasticity with a prior. Formally, we define the overall objective as Eq. 9 in Figure 2. We describe each of these in the following section.

Maximizing the likelihood of new observations. To learn from the new observation, we use maximum likelihood estimation. This raises the question: *what are we maximizing the likelihood of?* Either we can maximize the likelihood of the joint key-value pair $P(\mathsf{data} \mid \theta) = P(\boldsymbol{v}_t, \boldsymbol{k}_t \mid c_t, \theta)$ or the conditional likelihood $P(\boldsymbol{v}_t, c_t, \mid \boldsymbol{k}_t, \theta)$.

Since the true component distribution c_t is unknown, in both cases we must perform expectation-maximization. First, in the E-step, we predict a component distribution denoted $\boldsymbol{z}_t \in \mathbb{R}^D$ for $P(c_t = j) = z_{t,j}$, which we leave as a design choice. Then, in the M-step, we maximize the log likelihood in expectation over c_t, as in

$$\mathbb{E}_{c_t \sim \boldsymbol{z}_t}[\log P(\boldsymbol{v}_t, \boldsymbol{k}_t \mid c_t, \theta)] = \mathbb{E}[\log P(\boldsymbol{v}_t \mid c_t, \theta^{(2)})] + \mathbb{E}[\log P(\boldsymbol{k}_t \mid c_t, \theta^{(1)})] \text{ (Joint)} \tag{10}$$

$$\mathbb{E}_{c_t \sim \boldsymbol{z}_t}[\log P(\boldsymbol{v}_t, c_t \mid \boldsymbol{k}_t, \theta)] = \mathbb{E}[\log P(\boldsymbol{v}_t \mid c_t, \theta^{(2)})] + \mathbb{E}[\log P(c_t \mid \boldsymbol{k}_t, \theta^{(1)})] \text{ (Conditional)} \tag{11}$$

In both settings, the key- and value-side parameters $\theta^{(1)}$ and $\theta^{(2)}$ decompose into separate terms as they are independent conditioned on a component. Given a component distribution $\boldsymbol{z}_t$, we can solve for each state in parallel.

Furthermore, this decomposition illustrates how the joint and conditional likelihood agree for $\theta^{(2)}$ but encourage different roles for $\theta^{(1)}$. In the conditional, $\theta^{(1)}$ learns to map keys to the estimated

component distribution $c_t \sim \boldsymbol{z}_t$. This is preferable over the joint, as we compute $P(c_t \mid \mathsf{K} = \boldsymbol{q}_t)$ when computing the output expectation, $\mathbb{E}[\mathsf{V} \mid \mathsf{K} = \boldsymbol{q}]$.

Encouraging plasticity. A natural next step to maximum likelihood estimation is maximum a posteriori, which leverages a prior of the parameter space. While previous methods (Peng et al., 2022; Zhang et al., 2024; Afzal et al., 2026) implicitly use a uniform prior, we choose a zero-mean Gaussian prior over θ with data-dependent isotropic variance. This recovers the same gating mechanisms in SSMs (Dao & Gu, 2024) and linear attention variants (Yang et al., 2025) by penalizing parameter choices with large Frobenius norm (Appendix I). Furthermore, one can choose a non-zero mean Gaussian, which would decay the state parameters towards a pre-determined state, useful in settings where θ at $t = 0$ is initialized as some nonzero state.

While gating is traditionally viewed as a form of "forgetting" (Gers et al., 2000). we argue this is more akin to plasticity. Shrinking a vector does not change the direction; however, it changes how easily a future update will move this vector. Higher magnitude vectors do not rotate or change direction easily from additive updates, acting as rigid memories. A Gaussian prior slowly decays parameter magnitudes over time, increasing the plasticity of each memory.

Remembering past observations. Catastrophic forgetting is a critical issue when performing online learning on streaming data (Kirkpatrick et al., 2017). Unlike attention, we do not have access to past key-value pairs; we cannot leverage an unbounded cache. Instead, the previous parameter iteration, θ_{t-1}, remains our best statistic of the past key-value pairs $\{(\boldsymbol{k}_i, \boldsymbol{v}_i)\}_{i \le t}$. To retain past information, we use update regularization to prevent parameters from drifting too far from their previous iteration (Behrouz et al., 2026b). Concretely, we choose the reverse KL divergence in our objective to measure how "far" each memory is from the previous time step θ_{t-1}, summed over memories as in Eq. 9. This divergence compares the distribution of data that memory j generates before and after the update, $P(\mathsf{data} \mid \theta_{t-1,j})$ and $P(\mathsf{data} \mid \theta_{t,j})$. For the joint likelihood, we compute the divergence of memory j over the data (K, V), which is equivalent to the squared Euclidean distance between its new and previous mean, $\|\theta_{t,j} - \theta_{t-1,j}\|_2^2 / 2\sigma^2$ (Appendix I). For the conditional likelihood, we compute the divergence over (V, c_t) given the observed key $\mathsf{K} = \boldsymbol{k}_t$. The value-side parameters $\theta^{(2)}$ have the same Euclidean divergence. However, the key-side parameters minimize the divergence over the component probabilities given the key,

$$\mathrm{KL}\big(P(c_t \mid \boldsymbol{k}_t, \theta_t^{(1)}) \,\|\, P(c_t \mid \boldsymbol{k}_t, \theta_{t-1}^{(1)})\big), \qquad P(c_t = j \mid \boldsymbol{k}_t, \theta^{(1)}) = \mathrm{Softmax}_j\big(\theta^{(1)} \boldsymbol{k}_t / \sigma^2\big). \quad (12)$$

This term encourages $\theta_t^{(1)}$ to send $\boldsymbol{k}_t$ to the components that previous parameters $\theta_{t-1}^{(1)}$ would have routed to (Appendix I).

3.2 URCHIN

We combine these principles to design *Urchin*, our recall-focused, fixed-state sequence layer. Urchin performs maximum a posteriori with the conditional likelihood and the Gaussian parameter prior. It retains past context by minimizing the reverse KL divergence with respect to the previous parameter iteration. Inspired by Raven, Urchin also uses a sparse E-step (Neal & Hinton, 1998), updating only the most relevant memories with respect to the current key-value pair.

We choose to compute the component distribution $\boldsymbol{z}_t$ for Urchin as the posterior over memories under the previous parameters. The posterior measures the key- and value-similarity of the pair with each memory, together with a data-dependent mixture weight $\pi_{t,j} > 0$ that acts as a prior over components. The logits of the component distribution are

$$\boldsymbol{l}_{t,j} = \log \pi_{t,j} + \boldsymbol{k}_t^\top \theta_{t-1,j}^{(1)} + \boldsymbol{v}_t^\top \theta_{t-1,j}^{(2)}, \quad \text{with} \quad \boldsymbol{z}_t = \mathrm{Softmax}_{\text{top-16}}(\boldsymbol{l}_t). \quad (13)$$

Here, the softmax is taken over the top-16 largest logits and $\boldsymbol{z}_{t,j} = 0$ for the rest. This sparse E-step keeps unlikely pairs from making small adjustments to a memory, so a memory that holds a rare pair keeps it until a similar pair arrives. For parallelism during prefill, we precompute $\boldsymbol{z}_t$ over a chunk with stale parameters θ_{old} from the start of the chunk; otherwise, decoding uses θ_{t-1} directly. Given $\boldsymbol{z}_t$, $\boldsymbol{k}_t$ and $\boldsymbol{v}_t$, Urchin updates the relevant memories as minimizers of Eq. 9. For each memory j

with $\boldsymbol{z}_{t,j} > 0$,

$$\theta^{(2)}_{t,j} = \frac{(1-\beta_t)}{(1-\beta_t)+\beta_t \boldsymbol{z}_{t,j}}\,\alpha_t \theta^{(2)}_{t-1,j} + \frac{\beta_t \boldsymbol{z}_{t,j}}{(1-\beta_t)+\beta_t \boldsymbol{z}_{t,j}}\,\boldsymbol{v}_t, \tag{14}$$

$$\theta^{(1)}_{t,j} = \alpha_t \theta^{(1)}_{t-1,j} + \beta_t\big(\boldsymbol{l}_{t,j} - \boldsymbol{k}_t^\top \theta^{(1)}_{t-1,j}\big)\,\boldsymbol{k}_t. \tag{15}$$

Otherwise, it keeps the previous parameter iteration. In the update rule, α_t acts as the decay term induced by the Gaussian prior. The value-side state $\theta^{(2)}_{t,j}$ blends between retaining previous (decayed) context $\alpha_t \theta^{(2)}_{t-1,j}$ and the observation $\boldsymbol{v}_t$, depending on the component distribution $\boldsymbol{z}_{t,j}$. The key-side state $\theta^{(1)}_t$ is updated in the logit-space to match the component distribution of $\boldsymbol{z}_t$. With this update rule, Urchin can sharply route a later query resembling $\boldsymbol{k}_t$ to the components $(\boldsymbol{k}_t, \boldsymbol{v}_t)$ is stored in.

We follow GDN (Yang et al., 2025) for the block design, the short convolution, and the parametrization of the decay α_t as a Mamba-2 gate. The learning rate β_t is a softplus of $\boldsymbol{x}_t$ clamped at one, and the prior $\log \pi_t \in \mathbb{R}^D$ is a low-rank linear projection of $\boldsymbol{x}_t$. Both update rules are gated linear recurrences, allowing us to leverage fast triton kernels (Yang & Zhang, 2024) over chunks of 256 tokens for implementation. The remaining training details are in Appendix F and the full derivation is in Appendix J.

3.3 Previous Layer Designs

Beyond Urchin, our framework encompasses previous dual-state SSMs such as ABC, GSA, and Raven. These layers can be viewed as minimizing Eq. 9 with the joint likelihood and a uniform prior, yielding the update rule

$$\theta_{t,j} = \frac{(1-\beta_t)}{(1-\beta_t)+\beta_t \boldsymbol{z}_{t,j}}\,\theta_{t-1,j} + \frac{\beta_t \boldsymbol{z}_{t,j}}{(1-\beta_t)+\beta_t \boldsymbol{z}_{t,j}}\,[\boldsymbol{k}_t, \boldsymbol{v}_t]. \tag{16}$$

Under the perspective of Gaussian mixture modeling, these layers compute component distributions $\boldsymbol{z}_t$ in the E-step that only depend on the current token $\boldsymbol{x}_t$. Each of these layers makes a different choice for computing the component distribution $\boldsymbol{z}_t$ and learning rate β_t. ABC accumulates probability masses over time to decrease the learning rate for memories that model many key-value observations. GSA computes each memory's contribution independently with a sigmoid. Raven restricts $\boldsymbol{z}_t$ over the most relevant memories, preventing unrelated key-value observations from corrupting important memories. We adopt this choice in Urchin by only computing the M-step when $\boldsymbol{z}_t$ is nonzero. Lastly, Urchin differs from all three by modeling the conditional likelihood. We dive deeper into understanding these previous layers in Appendix K

4 Experiments

We empirically validate that Urchin is a promising sequence modeling layer with strong long-context retrieval capabilities.

- In Section 4.1, we show that Urchin achieves *near-perfect* recall accuracy on the RULER S-NIAH-1 benchmark (Hsieh et al., 2024) at extreme context lengths up to **1 million** and beyond—well past the lengths at which other models break down.
- In Section 4.2, we show that Urchin models, and hybrid Urchin variants, achieve strong performance on the S-NIAH-2 and 3 benchmarks—matching or exceeding the performance of other models at 850M scale on S-NIAH-2—as well as maintaining competitive accuracy on real-world recall tasks and standard language modeling benchmarks.
- In Section 4.3, we study RULER as a measure of recall performance more in-depth. We show how minor prompting differences can elicit large improvements across model types, and construct a suite of 5 additional RULER-like tasks, as a companion to the existing SNIAH1-3, to more thoroughly understand and evaluate the recall limits of models.

We pretrain models on the SlimPajama dataset (Soboleva et al., 2023) at three model sizes: $\sim$120M, $\sim$400M, and $\sim$850M; exact parameter counts within each size class are chosen so as to approximately flop-match across model types. 120M and 400M models are trained on 15B tokens, with

Table 1: **Long Context Recall** on RULER S-NIAH-1 at extreme context lengths. We report the average of three training seeds for 120M and 400M models. Accuracies are averaged over 500 samples up to 128K, 250 at 256K, 100 at 384K/512K and 50 at 768K/1M.

		S-NIAH-1 Across Context Lengths												
Model	**Params**	1K	2K	4K	8K	16K	32K	64K	128K	256K	384K	512K	768K	1M
~120M params, 15B tokens														
Mamba-2	121M	99.3 ±0	69.2 ±20	25.9 ±11	11.6 ±6	6.3 ±2	3.3 ±2	1.7 ±1	0.9 ±0	0.2 ±0	0.0 ±0	0.0 ±0	0.0 ±0	0.0 ±0
GLA	121M	43.8 ±5	15.6 ±1	4.1 ±1	0.9 ±1	0.9 ±1	0.4 ±0	0.2 ±0	0.0 ±0	0.0 ±0	0.0 ±0	0.0 ±0	0.0 ±0	0.0 ±0
GSA	121M	96.7 ±0	95.8 ±1	90.2 ±5	76.2 ±21	60.0 ±39	36.9 ±30	18.7 ±15	10.7 ±9	5.2 ±5	3.5 ±4	0.5 ±1	2.0 ±3	0.0 ±0
Gated DeltaNet	121M	97.9 ±4	96.0 ±6	91.3 ±13	75.4 ±23	35.3 ±16	17.9 ±8	10.3 ±4	4.9 ±3	2.3 ±1	1.7 ±1	0.0 ±0	0.0 ±0	0.0 ±0
KDA	127M	**99.9** ±0	99.7 ±0	94.0 ±6	73.9 ±23	46.1 ±31	18.6 ±12	4.6 ±2	1.3 ±1	0.8 ±1	0.0 ±0	0.3 ±1	0.0 ±0	0.0 ±0
Raven	127M	97.7 ±2	96.9 ±3	92.7 ±11	84.8 ±22	72.9 ±35	59.7 ±33	45.9 ±39	30.1 ±32	13.9 ±17	9.0 ±8	5.0 ±6	2.7 ±3	1.3 ±1
Urchin	124M	99.7 ±1	99.7 ±1	98.1 ±3	96.3 ±5	89.9 ±10	**84.8** ±14	**80.8** ±17	**80.9** ±18	**80.3** ±19	**77.3** ±22	**75.7** ±22	**78.0** ±20	**82.0** ±18
Urchin-KDA	126M	**99.9** ±0	**99.9** ±0	**99.8** ±0	**98.5** ±2	**92.2** ±11	77.1 ±28	54.6 ±38	43.3 ±46	34.9 ±47	33.3 ±47	28.7 ±48	22.7 ±38	26.7 ±43
~400M params, 15B tokens														
Mamba-2	400M	89.1 ±8	51.3 ±19	10.4 ±8	3.0 ±3	1.9 ±2	0.5 ±1	0.5 ±1	0.2 ±0	0.3 ±0	0.0 ±0	0.0 ±0	0.0 ±0	–
GLA	400M	84.6 ±16	30.3 ±13	9.9 ±3	3.4 ±1	2.3 ±2	0.4 ±0	0.3 ±0	0.0 ±0	0.0 ±0	0.0 ±0	0.0 ±0	0.0 ±0	0.0 ±0
GSA	399M	99.1 ±1	98.1 ±2	94.3 ±4	74.3 ±6	39.4 ±4	17.6 ±3	9.5 ±3	5.7 ±1	1.9 ±0	1.7 ±1	0.0 ±0	0.0 ±0	0.0 ±0
Gated DeltaNet	400M	**100.0** ±0	**100.0** ±0	98.2 ±2	82.7 ±15	55.9 ±34	34.5 ±27	19.4 ±15	10.5 ±11	5.1 ±6	5.3 ±5	0.7 ±1	1.3 ±2	1.3 ±2
KDA	400M	**100.0** ±0	99.9 ±0	97.0 ±5	85.1 ±19	62.3 ±30	40.7 ±31	27.0 ±28	17.0 ±22	8.8 ±12	5.3 ±7	2.0 ±4	2.0 ±4	2.7 ±5
Raven	424M	99.2 ±1	99.1 ±1	98.7 ±1	97.9 ±1	93.7 ±1	83.0 ±7	57.9 ±28	42.1 ±39	33.2 ±36	21.7 ±27	13.3 ±14	6.7 ±6	5.3 ±8
Urchin	406M	**100.0** ±0	**100.0** ±0	**100.0** ±0	**99.9** ±0	**99.3** ±1	**98.3** ±2	**96.8** ±3	**96.4** ±4	**95.9** ±4	**95.7** ±5	**97.3** ±2	**94.7** ±5	**97.3** ±3
Urchin-KDA	402M	**100.0** ±0	**100.0** ±0	**100.0** ±0	99.5 ±1	94.5 ±10	82.9 ±30	74.1 ±44	67.6 ±49	49.5 ±42	40.0 ±40	28.7 ±33	16.7 ±19	11.3 ±18
~850M params, 30B tokens														
Mamba-2	835M	**100.0**	**100.0**	99.0	52.6	19.4	8.0	2.6	0.4	0.4	–	–	–	–
GLA	836M	**100.0**	95.8	40.6	9.0	2.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
GSA	834M	**100.0**	**100.0**	99.6	98.0	88.0	38.6	10.6	6.0	1.6	0.0	0.0	0.0	0.0
Gated DeltaNet	835M	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.4	93.8	63.6	41.0	30.0	16.0	18.0
KDA	816M	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.8	96.2	66.8	28.0	13.0	7.0	4.0	4.0
Raven	891M	99.6	99.6	99.6	99.6	**100.0**	84.0	91.8	90.8	87.6	84.0	75.0	74.0	64.0
Urchin	845M	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Urchin-KDA	826M	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	95.2	58.8	32.8	21.0	11.0	10.0	6.0

Table 2: **Beyond 1 Million Token Haystacks**. Accuracies are averaged over 50 samples.

		S-NIAH-1 Across Context Lengths								
Model	**Params**	1.25M	1.5M	1.75M	2M	2.5M	3M	5M	7.5M	10M
Urchin	406M	94.0 ±3.5	92.0 ±4.0	95.3 ±2.3	93.3 ±1.2	93.3 ±4.6	91.3 ±3.1	93.3 ±4.6	94.0 ±4.0	93.3 ±6.4

a training context length of 2048; at these sizes, we train three runs with different seeds. 850M models are trained on 30B tokens with a training context length of 4096. These settings follow the methodology of Afzal et al. (2026). Further training and evaluation details, as well as additional ablation and interpretability experiments and full result tables, are in Appendix F.

4.1 Extreme-context recall

We first measure recall with RULER S-NIAH-1 in Table 1. In this task, each prompt stores a 7-digit number under a key inside a repeated haystack, and the model is prompted for the needle at the end. We evaluate all models from 1K up to 1M tokens (with no long-context fine-tuning). Urchin reaches 82.0%, 97.3% and 100.0% accuracy at 1M tokens at the three model sizes respectively, which is a 500× training length extrapolation (250× for the largest model). Furthermore, Table 1 demonstrates that a hybrid model alternating Urchin and KDA layers at a 1:2 ratio also achieves strong recall; i.e., adding Urchin layers can boost the length extrapolation capabilities of KDA models. In Table 2, we show that Urchin's memory retention can extend even further with small degradation, achieving 93.3% accuracy on S-NIAH-1 at **10 million** tokens.

4.2 Recall in natural text and language modeling

In Table 3, we study needle-in-a-haystack recall with natural language (essay) haystacks. S-NIAH-2 uses the same 7-digit number as the needle; S-NIAH-3 makes this task more difficult by using a 32-digit UUID. Additionally, as discussed in Section 4.3 we found that appending a colon (':') to the prompt significantly boosted recall accuracy for all models, so we report both the default prompt and the colon-added prompt results for S-NIAH-2 and S-NIAH-3. The full results for all models are located in Appendix C.

Aside from RULER, we also benchmark these fixed-state models on real-world retrieval tasks such as SWDE (Hao et al., 2011), FDA (Arora et al., 2023), and SQuAD (Rajpurkar et al., 2018), which

Table 3: **Language-Based Retrieval and Common Sense Reasoning.** For S-NIAH-2/3, we report two scores per context: the top score uses the default prompt, and the bottom score appends a single ":" to the prompt (see Sec. 4.3 for discussion). For Recall, we average scores on SWDE, FDA, SQuAD. For LM acc, we average accuracies on LAMBADA, PIQA, HellaSwag, WinoGrande, ARC-e, ARC-c. Urchin-KDA uses a 1:2 hybrid ratio. 400M-scale results are averaged across 3 training seeds. Full results are in the Appendix .

Model	SNIAH2				SNIAH3				Real-World	Language
Params	1K	2K	4K	8K	1K	2K	4K	8K	Recall	Modeling
Mamba-2	92.0 ±9	35.1 ±24	17.1 ±11	14.5 ±3	73.1 ±15	23.9 ±24	11.4 ±6	8.9 ±4	21.9 ±0.9	40.2 ±0.2
400M	99.9 ±0	96.3 ±1	36.0 ±2	15.5 ±3	81.7 ±10	54.6 ±3	18.2 ±3	8.8 ±4		
GLA	87.5 ±14	58.9 ±34	21.0 ±9	3.7 ±2	38.0 ±13	9.6 ±4	1.3 ±1	0.7 ±0	19.0 ±1.7	39.4 ±0.4
400M	96.0 ±5	82.2 ±14	24.7 ±6	4.4 ±2	43.3 ±8	12.7 ±2	2.1 ±0	0.5 ±0		
GSA	98.6 ±1	97.2 ±1	47.7 ±7	8.8 ±1	46.5 ±33	26.2 ±19	6.6 ±5	2.7 ±2	19.0 ±0.9	40.1 ±0.3
399M	98.6 ±1	97.6 ±1	74.1 ±4	13.7 ±1	65.9 ±2	35.5 ±4	12.0 ±2	4.4 ±2		
GDN	99.9 ±0	89.3 ±13	55.7 ±21	19.3 ±11	78.5 ±10	41.0 ±35	27.0 ±12	**17.9** ±6	24.4 ±0.9	40.7 ±0.3
400M	**100.0** ±0	**100.0** ±0	76.0 ±24	28.7 ±8	86.3 ±9	65.9 ±5	40.9 ±12	**18.9** ±7		
KDA	99.9 ±0	97.0 ±5	24.1 ±10	12.3 ±5	81.6 ±13	**63.4** ±36	14.8 ±12	6.6 ±8	29.5 ±1.0	**41.6** ±0.3
400M	**100.0** ±0	**100.0** ±0	55.9 ±26	20.3 ±4	**89.5** ±8	**80.6** ±14	38.2 ±13	14.5 ±2		
Raven	98.7 ±2	96.6 ±3	**71.7** ±28	**33.9** ±33	63.7 ±12	38.7 ±13	9.0 ±3	2.0 ±2	27.1 ±2.0	39.5 ±0.4
424M	98.7 ±2	97.3 ±3	89.2 ±8	**38.3** ±29	64.5 ±11	39.0 ±13	9.7 ±4	2.1 ±2		
Urchin	92.1 ±12	74.8 ±18	47.3 ±24	4.6 ±3	68.9 ±32	48.5 ±35	**30.0** ±24	3.1 ±4	28.7 ±1.2	39.8 ±0.4
406M	**100.0** ±0	**100.0** ±0	95.2 ±3	33.8 ±11	87.6 ±4	68.7 ±6	47.5 ±4	9.0 ±3		
Urchin-KDA	**100.0** ±0	**99.9** ±0	38.0 ±20	8.1 ±8	**83.7** ±8	63.3 ±10	5.4 ±5	3.7 ±3	**29.6** ±0.5	41.2 ±0.2
402M	**100.0** ±0	**100.0** ±0	**95.9** ±3	30.3 ±17	87.5 ±9	79.0 ±15	**47.7** ±16	8.8 ±7		
Mamba-2	**100.0**	99.6	83.8	22.8	71.6	77.6	3.2	11.2	40.0	44.2
835M	**100.0**	99.8	86.8	32.2	98.0	91.8	50.8	11.0		
GLA	99.2	99.0	89.0	17.2	89.2	75.8	33.6	3.0	33.1	43.3
836M	99.8	99.2	94.6	25.6	90.2	75.6	33.8	3.6		
GSA	**100.0**	99.6	57.8	24.4	94.8	85.4	30.0	9.8	31.8	42.8
834M	**100.0**	99.6	76.6	27.2	95.6	85.6	36.2	10.2		
GDN	**100.0**	**100.0**	86.0	14.8	96.4	75.2	60.4	32.6	37.1	45.1
835M	**100.0**	**100.0**	98.6	67.8	96.4	75.4	61.0	32.2		
KDA	**100.0**	**100.0**	98.0	36.4	**99.8**	**99.8**	**84.6**	**34.6**	**44.7**	**45.6**
816M	**100.0**	**100.0**	**99.6**	53.0	**99.8**	**99.8**	**86.6**	**35.6**		
Raven	**100.0**	99.6	44.4	**44.2**	89.8	74.4	17.0	3.4	32.8	41.7
891M	**100.0**	99.8	96.6	48.2	89.4	73.2	29.4	4.8		
Urchin	**100.0**	99.6	81.8	39.4	94.0	87.4	65.4	27.4	38.2	43.5
845M	**100.0**	**100.0**	99.0	**87.6**	94.0	87.2	67.4	26.6		
Urchin-KDA	**100.0**	**100.0**	**98.8**	22.2	99.4	96.6	46.4	11.8	42.8	45.5
826M	**100.0**	**100.0**	**99.6**	64.8	99.6	98.2	72.6	33.8		

test web-based and document-level extraction. We report the full results for all models in Appendix B.

Next, we evaluate language modeling performance across a suite of common sense reasoning tasks (Lambada (Paperno et al., 2016), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), and ARC-easy/challenge (Clark et al., 2018)). We report these results in Table 4 for 850M parameter models, but leave all architecture sizes in Appendix A. Urchin demonstrates comparable language modeling performance to most other fixed-state models, although KDA is generally the strongest language model. Finally, hybrid Urchin-KDA models achieve strong results on these evaluations; coupled with the results from Table 1, this shows that hybrid models can combine the language modeling prowess of KDA with Urchin's improved long-context recall behavior.

4.3 Understanding the Difficulties of Recall

We found that small base pretrained models often struggle to follow the RULER prompt instructions to return the needle. In other words, it is often not the case that they cannot perform recall—many

Table 4: **Language Modeling** across 850M parameter models.

Model	Params	LAMBADA		PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	Avg
		ppl↓	*acc*↑	*acc*↑	*acc*↑	*acc*↑	*acc*↑	*acc*↑	*acc*↑
∼850M params, 30B tokens									
Mamba-2	835M	22.83	38.7	68.7	34.7	51.3	51.5	20.5	44.2
GLA	836M	25.22	37.5	67.2	33.4	52.4	49.2	19.9	43.3
GSA	834M	28.93	33.5	67.4	34.4	50.7	49.4	21.6	42.8
Gated DeltaNet	835M	18.99	40.5	68.8	35.1	53.0	52.5	20.9	45.1
KDA	816M	**18.38**	41.0	**69.2**	**35.9**	**54.1**	51.6	**21.9**	**45.6**
Raven	891M	30.20	35.5	65.8	32.2	51.1	46.4	19.5	41.7
Urchin	845M	26.97	35.9	68.2	34.7	52.0	50.0	20.2	43.5
Urchin-KDA	826M	18.40	**42.2**	68.3	35.5	52.2	**53.3**	21.3	45.5

Table 5: **NIAH Variations** across haystack types, needle types, and needle lengths at 4K context. SN1-3* are colon-added versions of the corresponding RULER tasks; SN4-8* are our own created evaluations (also with colon at the end of prompt). Results are averaged across 3 training seeds.

Model	Params *haystack* *needle*	SN1* *repeat* 7-num	SN2* *essay* 7-num	SN3* *essay* 32-uuid	SN4* *repeat* 32-uuid	SN5* *essay* 7-uuid	SN6* *essay* 32-num	SN7* *repeat* 7-uuid	SN8* *repeat* 32-num
Mamba-2	400M	21.8 ±14.1	36.0 ±2.5	18.2 ±3.3	29.3 ±12.1	26.1 ±9.8	9.3 ±2.4	39.1 ±14.9	5.2 ±4.5
GLA	400M	13.2 ±4.3	24.7 ±6.3	2.1 ±0.5	2.1 ±0.4	13.0 ±5.8	0.0 ±0.0	7.1 ±4.0	0.1 ±0.1
GSA	399M	95.7 ±2.2	74.1 ±4.0	12.0 ±2.0	52.1 ±10.3	50.7 ±8.5	1.2 ±0.7	87.3 ±15.2	7.1 ±5.1
Gated DeltaNet	400M	99.3 ±1.2	76.0 ±23.7	40.9 ±12.0	87.5 ±6.0	48.3 ±6.4	35.4 ±27.2	93.8 ±10.2	63.4 ±34.1
KDA	400M	99.1 ±1.5	55.9 ±26.0	38.2 ±12.7	80.8 ±12.8	48.0 ±17.7	21.1 ±11.7	85.8 ±12.1	43.4 ±7.6
Raven	424M	98.6 ±1.1	89.2 ±8.0	9.7 ±4.4	82.7 ±3.7	63.2 ±11.5	0.3 ±0.3	99.7 ±0.4	4.4 ±3.8
Urchin	406M	**100.0** ±0.0	95.2 ±2.6	47.5 ±4.0	**95.1** ±1.1	62.1 ±9.6	**66.7** ±13.8	**99.8** ±0.2	85.8 ±8.7
Urchin-KDA	402M	**100.0** ±0.0	**95.9** ±2.9	**47.7** ±16.1	92.5 ±4.1	**69.6** ±11.5	55.3 ±27.7	99.0 ±0.2	**85.9** ±7.2

apparent recall failures are due instead simply to prompt adherence failure. By appending a single colon to the RULER prompt, we observed a significant boost in prompt adherence, and thereby recall accuracy, for all models; this allows us to further disentangle pure recall capabilities from other confounders. We detail this further in Appendix D.

In Table 5, we dive further into understanding what makes recall so difficult. In RULER, S-NIAH-1/2/3 all are specific choices of haystack type (repeated words vs. essay), needle type (numbers vs. UUIDs), and needle length (7 vs. 32 characters). We enumerate the remaining 5 possible combinations of these settings to create our own evaluations, denoted as S-NIAH-4-8. Based on our findings, we also use the colon-added prompt for these evaluations.

Across these 8 tasks, Urchin and Urchin-KDA shows results relatively robust to task configuration (haystack, needle type, and needle length). We found that even generally strong models struggled with longer needles; for instance, Raven achieves near-zero accuracy with 32-digit numbers in the essay haystack setting. While Gated DeltaNet and KDA are somewhat more robust to needle length, they still underperform Urchin.

5 Conclusion

In this paper, we introduce a framework for deriving fixed-state alternatives to softmax attention, based on the interpretation of attention as a nonparametric GMM. Our framework leads to both theoretical insights and practical improvements. We use our framework to derive Urchin, a new sequence modeling layer, and show that this layer achieves strong gains on recall even at extreme context lengths, while still remaining competitive on language modeling.

AI USE STATEMENT

We have used generative AI tools to help implement efficient methods, help formulate mathematical claims / assist in proofs, design figures, and for early drafting. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on research methodology or experiments, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis, or interpret results. All AI-generated code was verified. We have reviewed all AI-assisted work through unit-testing correctness for code speed-up, manually checking evaluations, writing mathematical claims by hand, and rewriting by hand. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

In Appendix F, we provide neccesary training details for reproducing the pretraining runs. Custom evaluations, such as colon-added prompting and SNIAH variants, are clearly explained to be reproduced. We will also be releasing our code implementation shortly.

REFERENCES

David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

Arshia Afzal, Aviv Bick, Eric P. Xing, Volkan Cevher, and Albert Gu. Raven: High-recall sequence modeling with sparse memory routing. *arXiv preprint arXiv:2607.25357*, 2026.

Nick Alonso, Tomas Figliolia, and Beren Millidge. Online vector quantized attention, 2026. URL https://arxiv.org/abs/2602.03922.

Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *Proceedings of the VLDB Endowment*, 17(2):92–105, 2023.

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

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 1763–1840. PMLR, 2024b.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. ATLAS: Learning to optimally memorize the context at test time. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a.

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. It's all connected: A journey through test-time memorization, attentional bias, retention, and online optimization. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology, 2024. URL https://arxiv.org/abs/2409.20325.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, NY, 2006. ISBN 978-0-387-31073-2.

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

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.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071. PMLR, 2024.

Sai Surya Duvvuri, Nirmal Patel, Nilesh Gupta, and Inderjit S. Dhillon. LUCID: Attention with preconditioned representations. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 10 2000. ISSN 0899-7667. doi: 10.1162/089976600300015015. URL https://doi.org/10.1162/089976600300015015.

Gautam Goel and Peter Bartlett. Can a transformer represent a Kalman filter? In Alessandro Abate, Mark Cannon, Kostas Margellos, and Antonis Papachristodoulou (eds.), *Proceedings of the 6th Annual Learning for Dynamics & Control Conference*, volume 242 of *Proceedings of Machine Learning Research*, pp. 1502–1512. PMLR, 15–17 Jul 2024. URL https://proceedings.mlr.press/v242/goel24a.html.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. From one tree to a forest: A unified solution for structured web data extraction. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 775–784, 2011.

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, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. https://kellerjordan.github.io/posts/muon/, 2024.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020.

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

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114. URL https://www.pnas.org/doi/abs/10.1073/pnas.1611835114.

Dmitry Krotov and John J. Hopfield. Dense associative memory for pattern recognition. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Aakash Lahoti, Kevin Y. Li, Berlin Chen, Caitlin Wang, Aviv Bick, J. Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. In *The Fourteenth International Conference on Learning Representations*, 2026.

Xia Li, Zhisheng Zhong, Jianlong Wu, Yibo Yang, Zhouchen Lin, and Hong Liu. Expectation-maximization attention networks for semantic segmentation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9166–9175. IEEE, 2019.

Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and Qiang Liu. Longhorn: State space models are amortized online learners. In *The Thirteenth International Conference on Learning Representations*, 2025.

Luke McDermott, Jr. Robert W. Heath, and Rahul Parhi. Lifelong in-context learning with transformers requires parametric forms of attention. *arXiv preprint arXiv:2606.25342*, 2026a.

Luke McDermott, Robert W. Heath, and Rahul Parhi. LoLA: Low-rank linear attention with sparse caching. *Transactions on Machine Learning Research*, 2026b. ISSN 2835-8856. URL https://openreview.net/forum?id=3KhDA252y3.

E. A. Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.

Radford M. Neal and Geoffrey E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In Michael I. Jordan (ed.), *Learning in Graphical Models*, pp. 355–368. Springer, 1998.

Tam Minh Nguyen, Tan Minh Nguyen, Dung D. D. Le, Duy Khuong Nguyen, Viet-Anh Tran, Richard Baraniuk, Nhat Ho, and Stanley Osher. Improving transformers with probabilistic attention keys. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16595–16621. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/nguyen22c.html.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Hao Peng, Jungo Kasai, Nikolaos Pappas, Dani Yogatama, Zhaofeng Wu, Lingpeng Kong, Roy Schwartz, and Noah A. Smith. ABC: Attention with bounded-memory control. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7469–7483. Association for Computational Linguistics, 2022.

Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. HGRN2: Gated linear RNNs with state expansion. In *First Conference on Language Modeling*, 2024.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, 2018.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Thomas Adler, David Kreil, Michael K. Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. In *International Conference on Learning Representations*, 2021.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366. PMLR, 2021.

Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. Deltaproduct: Improving state-tracking in linear RNNs via householder products. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

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. `https://huggingface.co/datasets/cerebras/SlimPajama-627B`, June 2023.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57503–57522. PMLR, 2025.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, Rif A. Saurous, Guillaume Lajoie, Charlotte Frenkel, Razvan Pascanu, Blaise Agüera y Arcas, and João Sacramento. MesaNet: Sequence modeling by locally optimal test-time training. In *The Fourteenth International Conference on Learning Representations*, 2026.

Ke Alexander Wang, Jiaxin Shi, and Emily B. Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *arXiv preprint arXiv:2501.12352*, 2025.

Geoffrey S. Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, 26(4):359–372, 1964.

Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. In *IRE WESCON Convention Record, Part 4*, pp. 96–104, 1960.

Songlin Yang and Yu Zhang. FLA: A Triton-based library for hardware-efficient implementations of linear attention mechanism. `https://github.com/fla-org/flash-linear-attention`, January 2024.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 56501–56523. PMLR, 2024a.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, volume 37, 2024b.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

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. In *The Fourteenth International Conference on Learning Representations*, 2026.

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

Yu Zhang, Songlin Yang, Ruijie Zhu, Yue Zhang, Leyang Cui, Yiqiao Wang, Bolun Wang, Freda Shi, Bailin Wang, Wei Bi, Peng Zhou, and Guohong Fu. Gated slot attention for efficient linear-time sequence modeling. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Yifei Zuo, Dhruv Pai, Zhichen Zeng, Alec Dewulf, Shuming Hu, and Zhaoran Wang. Parallax: Parameterized local linear attention for language modeling. *arXiv preprint arXiv:2605.29157*, 2026.

Table 6: **Extended Language Modeling Results**

Model	Params	LAMBADA		PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	Avg
		ppl↓	*acc*↑	*acc*↑	*acc*↑	*acc*↑	*acc*↑	*acc*↑	*acc*↑
∼120M params, 15B tokens									
Mamba-2	121M	112.25 ±5.57	22.3 ±0.8	61.5 ±0.7	28.4 ±0.0	**52.2** ±0.1	40.6 ±1.3	16.8 ±0.4	36.9 ±0.3
GLA	121M	108.48 ±10.89	22.2 ±0.9	60.8 ±0.6	28.1 ±0.3	51.3 ±1.9	39.3 ±0.4	17.8 ±1.0	36.6 ±0.5
GSA	121M	117.36 ±6.64	21.4 ±0.4	60.6 ±0.5	28.3 ±0.0	51.1 ±0.2	40.0 ±1.0	17.7 ±0.1	36.5 ±0.0
Gated DeltaNet	121M	94.56 ±5.78	23.8 ±0.6	61.5 ±0.9	28.5 ±0.0	51.7 ±0.8	41.1 ±0.4	17.1 ±0.7	37.3 ±0.3
KDA	127M	**70.87** ±2.20	**27.2** ±0.2	61.7 ±0.3	**28.9** ±0.2	51.2 ±1.6	**41.4** ±0.4	17.5 ±0.8	**38.0** ±0.4
Raven	127M	115.44 ±15.15	23.7 ±1.9	59.7 ±0.7	27.9 ±0.1	51.5 ±0.6	38.2 ±0.7	17.7 ±1.2	36.4 ±0.5
Urchin	124M	125.89 ±0.48	21.1 ±0.4	60.9 ±0.8	28.3 ±0.1	51.4 ±2.3	40.6 ±0.3	17.3 ±0.4	36.6 ±0.5
Urchin-KDA (1:1)	126M	80.16 ±1.15	26.4 ±0.3	**61.9** ±0.3	28.7 ±0.2	50.7 ±0.6	41.0 ±0.6	**18.3** ±0.9	37.8 ±0.1
Urchin-KDA (1:2)	126M	74.95 ±4.45	26.8 ±0.5	**61.9** ±0.2	28.6 ±0.1	50.2 ±1.1	40.8 ±0.7	17.9 ±0.6	37.7 ±0.1
Urchin-KDA (1:3)	127M	78.07 ±1.75	26.6 ±1.0	61.4 ±0.5	28.8 ±0.2	51.7 ±0.8	41.3 ±0.7	17.9 ±1.0	37.9 ±0.3
∼400M params, 15B tokens									
Mamba-2	400M	48.74 ±2.23	29.3 ±0.9	64.6 ±1.0	30.6 ±0.1	51.6 ±1.2	45.4 ±0.8	19.5 ±0.4	40.2 ±0.2
GLA	400M	50.12 ±3.62	28.9 ±1.3	63.7 ±0.2	29.8 ±0.2	51.2 ±1.6	44.5 ±0.9	18.3 ±0.4	39.4 ±0.4
GSA	399M	46.56 ±2.49	28.9 ±0.9	64.3 ±0.3	30.6 ±0.1	**51.7** ±0.5	45.7 ±1.0	**19.7** ±0.6	40.1 ±0.3
Gated DeltaNet	400M	38.43 ±1.40	31.7 ±0.4	65.3 ±0.1	31.0 ±0.1	50.8 ±1.3	46.2 ±0.7	19.0 ±0.8	40.7 ±0.3
KDA	400M	**30.62** ±1.11	**35.4** ±0.5	65.2 ±0.6	**31.4** ±0.1	51.0 ±0.8	46.8 ±0.9	19.6 ±0.1	**41.6** ±0.3
Raven	424M	45.78 ±4.07	31.1 ±1.1	62.9 ±0.8	29.5 ±0.1	50.6 ±1.4	43.8 ±0.8	19.2 ±0.7	39.5 ±0.4
Urchin	406M	49.91 ±2.96	28.4 ±1.0	64.9 ±0.4	30.5 ±0.1	50.6 ±0.4	46.1 ±0.5	18.3 ±1.0	39.8 ±0.4
Urchin-KDA (1:1)	403M	34.02 ±1.05	34.2 ±0.5	65.0 ±0.6	**31.4** ±0.3	50.6 ±0.8	**47.1** ±1.1	19.3 ±1.2	41.3 ±0.2
Urchin-KDA (1:2)	402M	33.36 ±2.51	33.9 ±1.1	65.0 ±0.7	**31.4** ±0.1	**51.7** ±0.6	46.1 ±0.3	19.0 ±0.7	41.2 ±0.2
Urchin-KDA (1:3)	401M	33.38 ±1.61	33.9 ±0.8	**65.4** ±0.1	31.3 ±0.2	**51.7** ±0.7	46.6 ±1.0	18.7 ±0.2	41.3 ±0.0
∼850M params, 30B tokens									
Mamba-2	835M	22.83	38.7	68.7	34.7	51.3	51.5	20.5	44.2
GLA	836M	25.22	37.5	67.2	33.4	52.4	49.2	19.9	43.3
GSA	834M	28.93	33.5	67.4	34.4	50.7	49.4	21.6	42.8
Gated DeltaNet	835M	18.99	40.5	68.8	35.1	53.0	52.5	20.9	45.1
KDA	816M	18.38	41.0	**69.2**	**35.9**	**54.1**	51.6	**21.9**	**45.6**
Raven	891M	30.20	35.5	65.8	32.2	51.1	46.4	19.5	41.7
Urchin	845M	26.97	35.9	68.2	34.7	52.0	50.0	20.2	43.5
Urchin-KDA (1:1)	831M	19.30	40.6	69.1	35.5	51.9	51.6	20.3	44.8
Urchin-KDA (1:2)	826M	18.40	42.2	68.3	35.5	52.2	**53.3**	21.3	45.5
Urchin-KDA (1:3)	824M	**17.75**	**42.8**	69.1	35.3	52.6	52.4	20.6	45.5

A Language Modeling Results

Table 6 reports the extended, per-task results behind the language modeling averages of Table 3, for all three tiers and all three hybrid ratios. Urchin demonstrates comparable language modeling results to other fixed-state methods, though KDA is incredibly strong in this domain. Each hybrid nearly recovers the language modeling prowess of full KDA, while demonstrating strong recall as in Table 1.

B Real-World Recall Results

Table 7 reports the extended, per-task recall results behind Table 3. Urchin models achieve the strongest performance in the 120M and 400M regimes, but fall short of KDA at 850M

C Extended RULER Results

Tables 8, 9 and 10 report S-NIAH-2 and S-NIAH-3 across architecture sizes and context lengths from 1K to 16K. We included both the default and colon-added prompting strategy for all models. Additionally we report the performance of Urchin-KDA hybrids across all 3 ratios.

In Table 2, we report the effect of colon-added prompts on S-NIAH-1 at 512K context length. While Urchin already highly performs here, we observe a substantial increase in other models such as the hybrids and Raven (discussed more in Appendix D).

Table 7: **Extended Real-World Recall.**

Model	Params	SWDE	FDA	SQuAD	Avg
∼120M params, 15B tokens					
Mamba-2	121M	12.9 ±0.4	8.7 ±0.3	21.8 ±1.2	14.5 ±0.3
GLA	121M	11.5 ±0.8	2.5 ±0.4	20.5 ±1.4	11.5 ±0.8
GSA	121M	11.6 ±2.4	3.7 ±1.0	19.9 ±0.9	11.7 ±0.8
Gated DeltaNet	121M	16.1 ±0.5	11.6 ±1.6	24.5 ±1.1	17.4 ±0.9
KDA	127M	18.8 ±0.5	10.4 ±0.5	25.8 ±1.6	18.3 ±0.6
Raven	127M	17.2 ±1.4	7.9 ±2.0	27.0 ±0.8	17.4 ±0.5
Urchin	124M	18.8 ±1.0	**15.6** ±0.7	24.2 ±1.1	19.5 ±0.5
Urchin-KDA (1:1)	126M	18.5 ±2.0	10.3 ±1.2	**27.1** ±0.6	18.6 ±1.0
Urchin-KDA (1:2)	126M	**20.8** ±0.3	15.3 ±1.8	26.3 ±0.4	**20.8** ±0.5
Urchin-KDA (1:3)	127M	20.1 ±3.1	11.1 ±2.5	25.1 ±0.7	18.8 ±1.1
∼400M params, 15B tokens					
Mamba-2	400M	23.3 ±2.7	11.7 ±3.1	30.7 ±2.3	21.9 ±0.9
GLA	400M	21.3 ±3.8	7.7 ±0.5	27.9 ±4.1	19.0 ±1.7
GSA	399M	21.2 ±1.7	8.3 ±1.6	27.5 ±1.4	19.0 ±0.9
Gated DeltaNet	400M	29.2 ±0.8	13.5 ±1.5	30.4 ±0.9	24.4 ±0.9
KDA	400M	34.3 ±2.1	20.9 ±1.7	33.2 ±1.4	29.5 ±1.0
Raven	424M	29.4 ±2.2	18.7 ±1.5	**33.3** ±2.5	27.1 ±2.0
Urchin	406M	31.3 ±1.6	**24.6** ±5.1	30.3 ±2.0	28.7 ±1.2
Urchin-KDA (1:1)	403M	34.7 ±0.4	22.1 ±1.9	**33.3** ±2.4	**30.0** ±1.1
Urchin-KDA (1:2)	402M	**36.0** ±1.8	20.4 ±2.2	32.5 ±1.4	29.6 ±0.5
Urchin-KDA (1:3)	401M	35.1 ±2.3	20.7 ±1.9	31.8 ±0.3	29.2 ±0.6
∼850M params, 30B tokens					
Mamba-2	835M	47.2	34.6	38.1	39.9
GLA	836M	39.7	23.9	35.7	33.1
GSA	834M	38.9	22.5	34.1	31.8
Gated DeltaNet	835M	46.9	27.7	36.7	37.1
KDA	816M	52.8	**40.9**	**40.3**	**44.7**
Raven	891M	37.3	21.9	39.1	32.7
Urchin	845M	44.7	33.7	36.3	38.2
Urchin-KDA (1:1)	831M	52.3	38.6	40.0	43.6
Urchin-KDA (1:2)	826M	50.2	38.1	40.1	42.8
Urchin-KDA (1:3)	824M	**52.9**	38.7	39.1	43.6

D NIAH VARIATIONS

Task Variations As described in Section 4.3, the RULER NIAH benchmarks have two different choices for haystack type (repeating text vs. essay), needle type (numbers vs. alphanumeric UUIDs), and needle length (7 or 32 characters). SNIAH1-3 represent specific instantiations of these choices, but there are 8 total possible tasks that can be constructed via every combination of these choices. These can be implemented via a simple extension of the RULER repository; we plan to release our evaluation code after the deadline.

Colon As shown in Section 4, appending a colon to the prompt improves models' performance on NIAH tasks, sometimes dramatically. We hypothesize that this is because in RULER, the needle is introduced in the text preceded by "is:" (e.g., `One of the special magic numbers for <key> is: 12345`), and the prompt ends with "is" (`The special magic number for <key> mentioned in the provided text is`). The substring "is:" appears nowhere else in the text. Appending a colon is thus a clearer signal that the model should reply with the needle that followed the colon. Without this intervention, common failure modes we observed across models include: simply continuing to generating English text continuations in response to the prompt; generating underscores (as though explicitly generating a blank for "fill-in-the-blank"). In practice, this intervention involves a very simple change to the base RULER evaluation workflow: it involves simply appending a colon to the `gen_prefix` field in the RULER task configuration.

Table 8: **120M-Scale Models on S-NIAH-2 and S-NIAH-3.**
Top-row uses default prompting, bottom-row adds a ":" before generation.

		S-NIAH-2					S-NIAH-3				
Model	**Params**	1K	2K	4K	8K	16K	1K	2K	4K	8K	16K
Mamba-2	121M	82.4 ±15.7	11.5 ±9.9	27.1 ±14.9	5.7 ±0.6	2.9 ±1.9	29.1 ±8.1	3.3 ±5.6	10.1 ±8.2	1.9 ±1.0	1.1 ±0.4
→		99.9 ±0.1	91.9 ±5.3	41.3 ±6.4	9.3 ±5.7	6.9 ±1.0	60.3 ±4.5	33.5 ±6.7	15.1 ±7.0	3.8 ±2.3	2.3 ±0.4
GLA	121M	70.3 ±30.7	15.3 ±21.6	6.5 ±7.7	0.4 ±0.5	0.2 ±0.3	5.1 ±5.2	0.7 ±1.3	0.1 ±0.2	0.0 ±0.0	0.0 ±0.0
→		91.3 ±5.1	54.6 ±15.9	5.9 ±8.0	0.2 ±0.3	0.5 ±0.5	7.9 ±2.3	1.1 ±0.9	0.1 ±0.2	0.0 ±0.0	0.0 ±0.0
GSA	121M	90.5 ±2.9	81.2 ±6.9	31.7 ±12.9	10.9 ±4.3	2.7 ±1.0	6.5 ±1.2	2.5 ±1.2	0.6 ±0.9	0.3 ±0.2	0.1 ±0.2
→		91.3 ±2.9	81.7 ±6.5	36.1 ±10.2	12.7 ±2.2	3.3 ±1.2	6.9 ±0.8	2.5 ±1.2	0.6 ±0.9	0.3 ±0.2	0.1 ±0.2
Gated DeltaNet	121M	93.7 ±6.4	60.5 ±45.6	30.7 ±19.5	6.7 ±7.3	1.5 ±1.2	22.4 ±27.5	8.2 ±14.2	6.9 ±6.0	**2.4** ±4.0	0.4 ±0.4
→		**100.0** ±0.0	99.9 ±0.1	77.3 ±15.7	21.0 ±3.6	**9.6** ±1.1	73.3 ±16.6	49.1 ±16.0	19.6 ±7.5	8.1 ±2.6	**3.0** ±0.9
KDA	127M	92.1 ±10.5	59.0 ±41.6	10.7 ±5.9	7.8 ±6.4	1.9 ±0.3	38.9 ±37.2	18.7 ±23.2	1.1 ±0.9	0.4 ±0.7	0.5 ±0.6
→		**100.0** ±0.0	**100.0** ±0.0	52.5 ±24.9	14.5 ±8.7	4.1 ±2.9	**87.3** ±5.0	**69.1** ±10.7	13.7 ±9.8	8.1 ±2.3	1.7 ±1.5
Raven	127M	93.5 ±6.0	91.7 ±6.6	50.1 ±22.5	15.2 ±8.7	3.2 ±1.9	34.7 ±3.4	13.5 ±1.5	1.3 ±0.1	0.4 ±0.2	0.3 ±0.1
→		93.4 ±6.3	91.6 ±6.8	59.9 ±27.2	21.5 ±16.4	3.4 ±2.3	34.7 ±3.8	13.7 ±1.5	2.0 ±0.4	0.7 ±0.1	0.4 ±0.0
Urchin	124M	80.6 ±18.1	59.0 ±27.2	46.0 ±19.7	7.5 ±8.7	1.7 ±1.9	24.9 ±27.1	8.1 ±6.8	4.2 ±3.7	0.0 ±0.0	0.0 ±0.0
→		**100.0** ±0.0	99.9 ±0.1	**96.7** ±1.2	27.6 ±19.1	4.5 ±2.7	76.8 ±10.0	55.4 ±5.0	**35.9** ±7.0	0.3 ±0.6	0.0 ±0.0
Urchin-KDA (1:1)	126M	**99.4** ±0.9	**97.6** ±2.3	43.7 ±18.5	6.7 ±4.5	2.1 ±1.3	**50.8** ±32.8	**37.1** ±26.9	**15.1** ±11.8	1.9 ±3.2	**1.4** ±1.3
→		99.9 ±0.1	**100.0** ±0.0	82.4 ±14.8	22.5 ±2.3	8.0 ±3.1	70.0 ±4.4	49.5 ±6.6	21.1 ±5.2	**9.9** ±3.4	2.8 ±0.9
Urchin-KDA (1:2)	126M	98.1 ±0.3	62.9 ±48.0	10.9 ±0.8	10.3 ±4.2	**3.9** ±0.6	45.6 ±21.7	16.4 ±21.8	4.6 ±2.8	1.0 ±0.9	1.0 ±1.0
→		99.9 ±0.2	99.9 ±0.2	79.1 ±15.9	**32.1** ±12.3	**9.6** ±1.4	76.4 ±6.1	53.1 ±11.4	17.3 ±2.2	4.9 ±2.4	1.5 ±1.3
Urchin-KDA (1:3)	127M	70.0 ±50.8	57.3 ±49.3	23.4 ±23.2	5.6 ±1.2	1.7 ±0.5	23.9 ±29.7	9.8 ±13.8	8.3 ±8.1	1.8 ±3.1	0.2 ±0.3
→		**100.0** ±0.0	99.7 ±0.3	67.8 ±21.4	20.9 ±7.0	8.1 ±6.3	73.5 ±13.8	44.6 ±16.1	20.0 ±7.1	8.7 ±5.8	2.1 ±2.3

Table 9: **400M-Scale Models on S-NIAH-2 and S-NIAH-3.**
Top-row uses default prompting, bottom-row adds a ":" before generation.

		S-NIAH-2					S-NIAH-3				
Model	**Params**	1K	2K	4K	8K	16K	1K	2K	4K	8K	16K
Mamba-2	400M	92.0 ±9.1	35.1 ±24.4	17.1 ±10.8	14.5 ±3.4	3.1 ±1.0	73.1 ±14.7	23.9 ±24.2	11.4 ±5.9	8.9 ±3.9	2.5 ±1.4
→		99.9 ±0.1	96.3 ±0.5	36.0 ±2.5	15.5 ±2.8	4.3 ±0.8	81.7 ±9.7	54.6 ±2.9	18.2 ±3.3	8.8 ±3.7	2.8 ±1.6
GLA	400M	87.5 ±13.9	58.9 ±33.7	21.0 ±9.4	3.7 ±1.8	2.1 ±0.4	38.0 ±13.2	9.6 ±3.6	1.3 ±1.3	0.7 ±0.3	0.3 ±0.3
→		96.0 ±5.4	82.2 ±13.9	24.7 ±6.3	4.4 ±1.9	2.3 ±0.4	43.3 ±8.5	12.7 ±1.9	2.1 ±0.5	0.5 ±0.4	0.4 ±0.5
GSA	399M	98.6 ±0.9	97.2 ±0.9	47.7 ±7.4	8.8 ±0.5	2.3 ±0.9	46.5 ±33.1	26.2 ±18.9	6.6 ±5.2	2.7 ±1.7	0.8 ±0.5
→		98.6 ±0.9	97.6 ±1.0	74.1 ±4.0	13.7 ±1.1	4.6 ±0.7	65.9 ±2.3	35.5 ±3.8	12.0 ±2.0	4.4 ±2.0	1.5 ±0.5
Gated DeltaNet	400M	99.9 ±0.1	89.3 ±12.9	55.7 ±21.4	19.3 ±10.6	4.1 ±2.7	78.5 ±10.5	41.0 ±35.0	27.0 ±12.1	**17.9** ±5.8	**3.9** ±1.9
→		**100.0** ±0.0	**100.0** ±0.0	76.0 ±23.7	28.7 ±7.9	13.4 ±5.9	86.3 ±8.8	65.9 ±5.3	40.9 ±12.0	**18.9** ±7.2	**5.7** ±2.8
KDA	400M	99.9 ±0.1	97.0 ±4.5	24.1 ±10.3	12.3 ±5.1	3.2 ±1.3	81.6 ±13.1	63.4 ±36.4	14.8 ±11.8	6.6 ±7.9	1.5 ±1.4
→		**100.0** ±0.0	**100.0** ±0.0	55.9 ±26.0	20.3 ±4.3	7.5 ±2.1	89.5 ±8.3	80.6 ±13.8	38.2 ±12.7	14.5 ±1.5	5.3 ±2.9
Raven	424M	98.7 ±1.8	96.6 ±3.0	71.7 ±28.2	**33.9** ±33.1	**7.1** ±7.9	63.7 ±11.6	38.7 ±12.7	9.0 ±3.3	2.0 ±2.5	1.2 ±1.0
→		98.7 ±1.6	97.3 ±3.2	89.2 ±8.0	38.3 ±28.8	9.9 ±10.5	64.5 ±10.8	39.0 ±13.2	9.7 ±4.4	2.1 ±2.4	1.3 ±0.7
Urchin	406M	92.1 ±12.1	74.8 ±18.5	47.3 ±23.7	4.6 ±3.2	1.1 ±0.8	68.9 ±32.0	48.5 ±34.7	30.0 ±24.0	3.1 ±4.1	0.3 ±0.6
→		**100.0** ±0.0	**100.0** ±0.0	95.2 ±2.6	33.8 ±10.7	**13.9** ±6.0	87.6 ±4.1	68.7 ±6.3	47.5 ±4.0	9.0 ±3.1	1.1 ±1.0
Urchin-KDA (1:1)	403M	99.9 ±0.1	89.7 ±14.7	64.7 ±24.7	23.8 ±22.6	4.8 ±3.1	**95.6** ±0.3	**83.4** ±11.5	**49.7** ±0.8	8.0 ±6.2	3.5 ±4.2
→		**100.0** ±0.0	**100.0** ±0.0	94.7 ±2.5	**41.9** ±8.4	13.4 ±7.3	**97.5** ±1.5	**87.1** ±5.4	**52.1** ±1.5	18.5 ±4.4	5.0 ±4.2
Urchin-KDA (1:2)	402M	**100.0** ±0.0	**99.9** ±0.1	38.0 ±19.7	8.1 ±7.5	2.7 ±2.0	83.7 ±8.0	63.3 ±10.2	5.4 ±4.6	3.7 ±3.4	1.3 ±1.2
→		**100.0** ±0.0	**100.0** ±0.0	**95.9** ±2.9	30.3 ±16.7	9.1 ±5.4	87.5 ±9.3	79.0 ±15.0	47.7 ±16.1	8.8 ±6.7	2.7 ±3.0
Urchin-KDA (1:3)	401M	99.7 ±0.6	92.2 ±12.5	63.1 ±20.4	19.0 ±6.9	2.5 ±1.0	40.9 ±45.6	46.2 ±39.4	10.8 ±14.6	2.0 ±1.7	0.3 ±0.3
→		**100.0** ±0.0	99.8 ±0.3	85.7 ±3.9	30.2 ±12.0	11.9 ±11.2	95.1 ±1.8	85.2 ±6.0	33.9 ±2.5	10.9 ±5.9	4.1 ±3.6

In Table 13, we additionally compare three different prompting strategies: the default, a colon followed by a space, and a bare colon. We found that bare-colon prompts were robust across task type, particularly due to the affects of spacing and tokenization. In Tables 13 and 14, we see that adding a space in the prompt can change how the needle tokenizes in the haystack versus at generation, leading to recall issues in small models. Furthermore, in Table 15, we observed only a small colon-prompt effect for the (harder) multi-needle tasks (MK1, MQ, MV). We hypothesize is that this is because these tasks contain multiple needle demonstrations in the context, and models therefore have an easier time knowing that they need to return the needle but simply fail to actually retrieve the correct value.

Discussion We show results for all models at 4K context length in Table 12, for all 8 SNIAH variants with and without a colon. We observe that all three axes of task variation (haystack type, needle type, and needle length) can have a large effect on performance.

We also observe that adding the colon maintains or improves accuracy in every setting (up to noise), often substantially. Additionally, it often makes results much more stable across seeds: without the

Table 10: **850M-Scale Models on S-NIAH-2 and S-NIAH-3.**
Top-row uses default prompting, bottom-row adds a ":" before generation.

		S-NIAH-2					S-NIAH-3				
Model	**Params**	1K	2K	4K	8K	16K	1K	2K	4K	8K	16K
Mamba-2	835M	**100.0**	99.6	83.8	22.8	9.2	71.6	77.6	3.2	11.2	2.8
→		**100.0**	99.8	86.8	32.2	10.0	98.0	91.8	50.8	11.0	3.6
GLA	836M	99.2	99.0	89.0	17.2	2.2	89.2	75.8	33.6	3.0	2.0
→		99.8	99.2	94.6	25.6	2.2	90.2	75.6	33.8	3.6	2.0
GSA	834M	**100.0**	99.6	57.8	24.4	5.2	94.8	85.4	30.0	9.8	1.2
→		**100.0**	99.6	76.6	27.2	8.2	95.6	85.6	36.2	10.2	2.4
Gated DeltaNet	835M	**100.0**	**100.0**	86.0	14.8	4.6	96.4	75.2	60.4	32.6	**9.2**
→		**100.0**	**100.0**	98.6	67.8	20.8	96.4	75.4	61.0	32.2	9.6
KDA	816M	**100.0**	**100.0**	98.0	36.4	14.4	**99.8**	**99.8**	**84.6**	**34.6**	2.6
→		**100.0**	**100.0**	**99.6**	53.0	16.6	**99.8**	**99.8**	**86.6**	**35.6**	10.2
Raven	891M	**100.0**	99.6	44.4	44.2	4.0	89.8	74.4	17.0	3.4	0.0
→		**100.0**	99.8	96.6	48.2	8.2	89.4	73.2	29.4	4.8	0.4
Urchin	845M	**100.0**	99.6	81.8	39.4	**17.2**	94.0	87.4	65.4	27.4	4.6
→		**100.0**	**100.0**	99.0	87.6	**78.0**	94.0	87.2	67.4	26.6	**12.0**
Urchin-KDA (1:1)	831M	**100.0**	**100.0**	95.6	**55.4**	8.4	96.8	94.4	66.2	19.0	3.0
→		**100.0**	**100.0**	98.6	**88.8**	13.4	97.0	95.2	67.4	27.2	3.6
Urchin-KDA (1:2)	826M	**100.0**	**100.0**	**98.8**	22.2	13.6	99.4	96.6	46.4	11.8	3.4
→		**100.0**	**100.0**	**99.6**	64.8	18.0	99.6	98.2	72.6	33.8	6.0
Urchin-KDA (1:3)	824M	**100.0**	**100.0**	97.4	28.4	12.2	94.8	67.4	4.6	3.8	1.6
→		**100.0**	**100.0**	99.0	49.4	15.6	99.0	92.8	68.8	19.2	5.2

Table 11: **SNIAH1 at 512K context with and without colon prompt.** The default prompt ends "in the provided text" vs. colon-added prompt ends "in the provided text:".

Model	**Params**	**SNIAH1**	**SNIAH1 + colon**
GLA	400M	0.0 ±0.0	0.0 ±0.0
GSA	399M	0.0 ±0.0	0.0 ±0.0
Gated DeltaNet	400M	0.7 ±1.2	7.7 ±13.3
KDA	400M	2.0 ±3.5	4.7 ±8.1
Raven	424M	13.3 ±13.7	28.0 ±19.1
Urchin	406M	97.3 ±2.3	97.0 ±2.6
Urchin-KDA (1:1)	403M	7.3 ±4.0	66.3 ±19.3
Urchin-KDA (1:2)	402M	28.7 ±33.3	61.7 ±31.5
Urchin-KDA (1:3)	401M	52.3 ±6.7	78.0 ±7.8

colon, we observe that some (model, evaluation) pairs exhibit fairly bimodal performance, where certain seeds of the same settings will perform reasonably well while others will nearly completely fail. With the colon, this phenomenon is generally less severe.

We hope that our modified and extended versions of the NIAH benchmarks will prove useful to future practitioners who want to assess specific recall capabilities of models.

E Related Work

Linear Methods and Test-Time Regression. Test-time regression (Wang et al., 2025) views attention and many fixed-state alternatives as online, key-to-value regressors. Specifically, softmax attention is a Nadaraya–Watson regressor (Nadaraya, 1964; Watson, 1964) with the Gaussian kernel when queries and keys are unit-norm. This framework leads to many different sequence layers, depending on the type of regressor used (McDermott et al., 2026a). In this section we discuss examples of linear regressors, nonlinear (parametric) regressors, and nonparametric regressors.

Table 12: **NIAH Variations** across haystack types, needle types, and needle lengths at 4K context, with and without a colon at the end of the prompt. For each model, the first row uses the default prompt and the + *colon* row ends the prompt with a bare colon ("...is:"). SN1–3 are RULER tasks; SN4–8 are our own evaluations. Within each model, both rows are scored on identical samples. Results are averaged across 3 training seeds. Bold/underline mark the best/second-best score within each prompt setting. (This is an expanded version of Table 5.)

Model	**Params** *haystack* *needle*	SN1 *repeat* 7-num	SN2 *essay* 7-num	SN3 *essay* 32-uuid	SN4 *repeat* 32-uuid	SN5 *essay* 7-uuid	SN6 *essay* 32-num	SN7 *repeat* 7-uuid	SN8 *repeat* 32-num
Mamba-2	400M	10.4 ±8.3	17.1 ±10.8	11.4 ±5.9	28.7 ±11.6	12.3 ±8.3	5.3 ±3.8	37.1 ±14.9	3.3 ±2.9
+ colon		21.8 ±14.1	36.0 ±2.5	18.2 ±3.3	29.3 ±12.1	26.1 ±9.8	9.3 ±2.4	39.1 ±14.9	5.2 ±4.5
GLA	400M	9.9 ±3.1	21.0 ±9.4	1.3 ±1.3	1.7 ±0.2	8.3 ±9.6	0.0 ±0.0	5.9 ±1.6	0.1 ±0.1
+ colon		13.2 ±4.3	24.7 ±6.3	2.1 ±0.5	2.1 ±0.4	13.0 ±5.8	0.0 ±0.0	7.1 ±4.0	0.1 ±0.1
GSA	399M	94.3 ±3.5	47.7 ±7.4	6.6 ±5.2	45.9 ±12.1	14.3 ±17.9	1.3 ±0.8	78.1 ±16.8	6.5 ±4.1
+ colon		95.7 ±2.2	74.1 ±4.0	12.0 ±2.0	52.1 ±10.3	50.7 ±8.5	1.2 ±0.7	87.3 ±15.2	7.1 ±5.1
Gated DeltaNet	400M	98.2 ±1.7	55.7 ±21.4	27.0 ±12.1	87.5 ±5.8	30.8 ±14.8	**30.0** ±23.2	93.9 ±10.1	62.5 ±33.6
+ colon		99.3 ±1.2	76.0 ±23.7	40.9 ±12.0	87.5 ±6.0	48.3 ±6.4	35.4 ±27.2	93.8 ±10.2	63.4 ±34.1
KDA	400M	97.0 ±5.2	24.1 ±10.3	14.8 ±11.8	78.9 ±15.1	14.4 ±11.6	12.7 ±9.8	84.3 ±13.2	41.4 ±8.9
+ colon		99.1 ±1.5	55.9 ±26.0	38.2 ±12.7	80.8 ±12.8	48.0 ±17.7	21.1 ±11.7	85.8 ±12.1	43.4 ±7.6
Raven	424M	98.7 ±1.3	**71.7** ±28.2	9.0 ±3.3	83.3 ±3.5	**55.2** ±6.9	0.2 ±0.3	99.5 ±0.5	3.9 ±3.5
+ colon		98.6 ±1.1	89.2 ±8.0	9.7 ±4.4	82.7 ±3.7	63.2 ±11.5	0.3 ±0.3	99.7 ±0.4	4.4 ±3.8
Urchin	406M	**100.0** ±0.0	47.3 ±23.7	**30.0** ±24.0	**95.0** ±1.1	38.8 ±33.8	28.2 ±20.6	**99.8** ±0.2	**84.6** ±9.7
+ colon		**100.0** ±0.0	95.2 ±2.6	47.5 ±4.0	**95.1** ±1.1	62.1 ±9.6	**66.7** ±13.8	**99.8** ±0.2	85.8 ±8.7
Urchin-KDA	402M	**100.0** ±0.0	38.0 ±19.7	5.4 ±4.6	92.1 ±4.1	6.7 ±4.8	24.4 ±15.5	98.8 ±0.9	83.6 ±7.1
+ colon		**100.0** ±0.0	**95.9** ±2.9	**47.7** ±16.1	92.5 ±4.1	**69.6** ±11.5	55.3 ±27.7	99.0 ±0.2	**85.9** ±7.2

Table 13: **400M Models on S-NIAH-2 and S-NIAH-3: answer-cue ablation.**
The top row for each model represents the base evaluations, followed by results when adding a ": ", then a ":" to the prompt before generation. We found appending a bare-colon, no trailing space, was the most robust, likely due to tokenization of letter-first UUIDs.

		S-NIAH-2					**S-NIAH-3**				
Model	**Params**	1K	2K	4K	8K	16K	1K	2K	4K	8K	16K
Mamba-2	400M	92.0 ±9.1	35.1 ±24.4	17.1 ±10.8	14.5 ±3.4	3.1 ±1.0	73.1 ±14.7	23.9 ±24.2	11.4 ±5.9	8.9 ±3.9	2.5 ±1.4
→ ": "		99.9 ±0.1	96.3 ±0.6	36.1 ±2.2	15.5 ±2.9	4.1 ±0.6	58.1 ±0.6	47.5 ±6.1	18.0 ±2.1	8.5 ±2.8	2.7 ±1.3
→ ":"		99.9 ±0.1	96.3 ±0.5	36.0 ±2.5	15.5 ±2.8	4.3 ±0.8	81.7 ±9.7	54.6 ±2.9	18.2 ±3.3	8.8 ±3.7	2.8 ±1.6
Gated DeltaNet	400M	99.9 ±0.1	89.3 ±12.9	55.7 ±21.4	19.3 ±10.6	4.1 ±2.7	78.5 ±10.5	41.0 ±35.0	27.0 ±12.1	**17.9** ±5.8	**3.9** ±1.9
→ ": "		**100.0** ±0.0	**100.0** ±0.0	76.6 ±24.8	29.1 ±8.3	13.8 ±6.6	59.7 ±1.4	58.2 ±3.1	41.1 ±15.4	**16.9** ±6.1	**6.1** ±3.9
→ ":"		**100.0** ±0.0	**100.0** ±0.0	76.0 ±23.7	28.7 ±7.9	13.4 ±5.9	86.3 ±8.8	65.9 ±5.3	40.9 ±12.0	**18.9** ±7.2	**5.7** ±2.8
KDA	400M	99.9 ±0.1	97.0 ±4.5	24.1 ±10.3	12.3 ±5.1	3.2 ±1.3	81.6 ±13.1	63.4 ±36.4	14.8 ±11.8	6.6 ±7.9	1.5 ±1.4
→ ": "		**100.0** ±0.0	**100.0** ±0.0	55.6 ±26.5	20.1 ±4.4	7.5 ±2.1	61.3 ±2.5	**58.7** ±1.1	31.6 ±7.0	13.5 ±1.3	4.6 ±2.0
→ ":"		**100.0** ±0.0	**100.0** ±0.0	55.9 ±26.0	20.3 ±4.3	7.5 ±2.1	89.5 ±8.3	80.6 ±13.8	38.2 ±12.7	14.5 ±1.5	5.3 ±2.9
Raven	424M	98.7 ±1.8	96.6 ±3.0	71.7 ±28.2	**33.9** ±33.1	**7.1** ±7.9	63.7 ±11.6	38.7 ±12.7	9.0 ±3.3	2.0 ±2.5	1.2 ±1.0
→ ": "		98.7 ±1.6	97.3 ±3.6	89.7 ±7.2	47.0 ±21.7	10.7 ±9.1	40.4 ±5.6	28.5 ±7.6	7.5 ±2.8	2.1 ±2.1	1.0 ±0.6
→ ":"		98.7 ±1.6	97.3 ±3.2	89.2 ±8.0	38.3 ±28.8	9.9 ±10.5	64.5 ±10.8	39.0 ±13.2	9.7 ±4.4	2.1 ±2.4	1.3 ±0.7
Urchin	406M	92.1 ±12.1	74.8 ±18.5	47.3 ±23.7	4.6 ±3.2	1.1 ±0.8	68.9 ±32.0	48.5 ±34.7	30.0 ±24.0	3.1 ±4.1	0.3 ±0.6
→ ": "		**100.0** ±0.0	**100.0** ±0.0	95.5 ±2.2	33.7 ±10.8	15.2 ±6.0	59.5 ±0.9	57.4 ±1.8	43.1 ±3.9	10.5 ±0.4	1.5 ±0.4
→ ":"		**100.0** ±0.0	**100.0** ±0.0	95.2 ±2.6	33.8 ±10.7	**13.9** ±6.0	87.6 ±4.1	68.7 ±6.3	47.5 ±4.0	9.0 ±3.1	1.1 ±1.0
Urchin-KDA (1:1)	403M	99.9 ±0.1	89.7 ±14.7	64.7 ±24.7	23.8 ±22.6	4.8 ±3.1	**95.6** ±0.3	**83.4** ±11.5	**49.7** ±0.8	8.0 ±6.2	3.5 ±4.2
→ ": "		**100.0** ±0.0	**100.0** ±0.0	95.9 ±2.7	**49.4** ±21.8	**19.3** ±6.6	**61.5** ±1.1	58.4 ±1.2	**43.3** ±2.4	15.1 ±6.3	4.5 ±3.8
→ ":"		**100.0** ±0.0	**100.0** ±0.0	94.7 ±2.5	**41.9** ±8.4	13.4 ±7.3	**97.5** ±1.5	**87.1** ±5.4	**52.1** ±1.5	18.5 ±4.4	5.0 ±4.2
Urchin-KDA (1:2)	402M	**100.0** ±0.0	**99.9** ±0.1	38.0 ±19.7	8.1 ±7.5	2.7 ±2.0	83.7 ±8.0	63.3 ±10.2	5.4 ±4.6	3.7 ±3.4	1.3 ±1.2
→ ": "		**100.0** ±0.0	**100.0** ±0.0	**97.9** ±0.6	47.0 ±31.5	17.2 ±12.3	59.1 ±1.0	56.3 ±1.7	39.1 ±7.0	8.9 ±6.4	2.7 ±2.9
→ ":"		**100.0** ±0.0	**100.0** ±0.0	**95.9** ±2.9	30.3 ±16.7	9.1 ±5.4	87.5 ±9.3	79.0 ±15.0	47.7 ±16.1	8.8 ±6.7	2.7 ±3.0
Urchin-KDA (1:3)	401M	99.7 ±0.6	92.2 ±12.5	63.1 ±20.4	19.0 ±6.9	2.5 ±1.0	40.9 ±45.6	46.2 ±39.4	10.8 ±14.6	2.0 ±1.7	0.3 ±0.3
→ ": "		**100.0** ±0.0	99.8 ±0.3	86.1 ±4.0	32.5 ±12.4	11.9 ±11.2	58.1 ±1.3	54.7 ±2.1	32.9 ±7.8	9.9 ±3.0	3.4 ±2.9
→ ":"		**100.0** ±0.0	99.8 ±0.3	85.7 ±3.9	30.2 ±12.0	11.9 ±11.2	95.1 ±1.8	85.2 ±6.0	33.9 ±2.5	10.9 ±5.9	4.1 ±3.6

Online linear regressors are the most common due to their parallel updates and readouts, but they differ in what online objective they optimize and how they optimize it. Models from the DeltaNet family (Schlag et al., 2021; Yang et al., 2024b) minimize the Euclidean distance between the predicted value (given the key) and the true value. Performing gradient descent over a single key-value pair yields update rules of the form

$$\theta_t = \alpha_t \theta_{t-1} + \beta_t (\boldsymbol{v}_t - \theta_{t-1} \boldsymbol{k}_t) \boldsymbol{k}_t^\top \in \mathbb{R}^{d_v \times d_k}. \tag{17}$$

This gives the error-correcting behavior of the *delta rule* (Widrow & Hoff, 1960), which adds the residual $\boldsymbol{v}_t - \theta_{t-1}\boldsymbol{k}_t$ to the state at each step. Gated DeltaNet (Yang et al., 2025) introduces the

Table 14: **Exploring UUID Effects in SNIAH3.** Default prompt ("...is") vs. colon followed by a space ("...is: ") vs. bare colon ("...is:"). Results are averaged across 500 samples, of which 198 UUIDs begin with a letter. With the trailing space, a letter-first uuid can no longer begin with its usual space-merged token (e.g. "␣ed"), so the model frequently fails to copy it.

Prompt	Digit-first	Letter-first
Default	76.8 $\pm$35.3	57.1 $\pm$28.8
Colon + space	97.7 $\pm$1.3	1.2 $\pm$0.8
Bare colon	97.5 $\pm$1.7	72.6 $\pm$12.7

Table 15: **Colon Effects in MK1, MQ and MV at 4K context** The default prompt ends "...is" / "...are" vs. colon-added prompt ends "...is:" / "...are:". Mean $\pm$ s.d. over 3 seeds, 500 samples each.

		MK1		MQ		MV	
Model	**Params**	default	+ colon	default	+ colon	default	+ colon
Gated DeltaNet	400M	**21.9** $\pm$5.7	**22.6** $\pm$4.7	12.8 $\pm$2.3	17.8 $\pm$7.0	15.5 $\pm$8.0	15.6 $\pm$8.0
Raven	424M	20.2 $\pm$4.2	20.7 $\pm$4.6	**17.4** $\pm$2.8	18.7 $\pm$3.2	**16.9** $\pm$3.1	16.9 $\pm$2.9
Urchin	406M	17.9 $\pm$7.1	**22.6** $\pm$1.7	14.1 $\pm$9.9	**22.4** $\pm$1.3	16.3 $\pm$6.1	**20.4** $\pm$0.9

gating mechanism of Mamba-2 (Dao & Gu, 2024), which can be viewed as adding weight decay to the objective. Kimi Delta Attention (Kimi Team, 2025) takes this further with a diagonal decay matrix, decaying each channel of the state at its own rate. Models that use the *Hebbian rule* instead update the state with the value itself, with no residual, and can be seen as maximizing the dot-product alignment between the predicted value and the true value. Hebbian models include vanilla linear attention (Katharopoulos et al., 2020; Arora et al., 2024b), GLA (Yang et al., 2024a), and Mamba-2. Beyond these two rules, varying the optimization procedure produces new layers. DeltaProduct (Siems et al., 2025) introduces several key-value pairs per time step, which raises the rank of the update, as does Mamba-3 (Lahoti et al., 2026). Longhorn (Liu et al., 2025), HGRN2 (Qin et al., 2024), and MesaNet (von Oswald et al., 2026) are further gated variants, with MesaNet solving the least-squares problem in closed form at each step.

Nonlinear parametric regressors have drawn recent interest under the name of test-time training. The original TTT layer (Sun et al., 2025) uses an MLP as the regressor and fits the key-to-value map by backpropagation. These regressors give a stronger form of memory; however, their update rules are not natively parallelizable, and their objectives require many gradient steps to fit the key-value pairs. Titans (Behrouz et al., 2025) adds a momentum state and computes gradients ahead of time from stale parameters. ATLAS (Behrouz et al., 2026a) further uses online Muon (Jordan et al., 2024), which orthogonalizes the momentum state with Newton–Schulz iterations (Bernstein & Newhouse, 2024). LaCT (Zhang et al., 2026) concurrently introduces online Muon and addresses parallelization with a hybrid design, modeling recent information with a 2048-token sliding window and updating an online MLP once per chunk. Lastly, Miras (Behrouz et al., 2026b) introduces context retention terms to the objective, similar to the retention term of Urchin.

Nonparametric regressors form the third family, in which attention itself is the Nadaraya–Watson kernel regressor (Nadaraya, 1964; Watson, 1964). This regressor estimates each neighborhood defined by the kernel with a constant and is called a *locally constant* estimator. Parallax (Zuo et al., 2026) pushes further with a *locally linear* estimator in attention. LUCID (Duvvuri et al., 2026) performs kernel ridge regression over the keys and values, which reduces the effect of high correlation among keys.

Other Mixture Models in Attention. The connection between attention and Gaussian mixture models has been explored in a variety of contexts. Alonso et al. (2026) use this connection to build an alternative attention that sends key-value pairs to their nearest centroid. Nguyen et al. (2022) increase the modeling capacity of attention by constructing many keys per token, each with its own mixture weight. This contrasts with our objective, which aims to reduce the number of past component. Furthermore, Li et al. (2019) applies these ideas to image segmentation. In this setting,

Table 16: Training throughput among dual-state SSMs at 400M tier over 2K sequences.

	Layer (ktok/s)			**Model** (tok/s/GPU)			**Training run** (8×H100)	
Model	B=2	B=4	B=8	B=2	B=4	B=8	tok/s/GPU	peak GiB
GSA	2088	2302	2408	55.2k	61.5k	64.7k	68.6k	33
Raven	1501	1618	1683	43.6k	47.4k	49.4k	53.1k	43
Urchin	1600	1957	2120	49.8k	58.4k	63.3k	62.3k	51

they can perform many iterations of EM over a fixed number of components. This contrasts with our setting, as we must fit each component online from streaming data.

Attention and Associative Memories. Attention draws on a long history of kernel regression (Nadaraya, 1964; Watson, 1964), adaptive filtering (Widrow & Hoff, 1960), and associative memories. When designing attention methods for recall, we draw inspiration from how associative memory networks store and retrieve patterns. Hopfield networks (Hopfield, 1982) store a bounded number of patterns in the weights of a network and recover a stored pattern from a corrupted query by descending an energy function. Modern Hopfield networks (Krotov & Hopfield, 2016; Ramsauer et al., 2021) use an exponential energy, in the same spirit as softmax attention. In this perspective, the query is a partial memory (either sparse, low-rank, or corrupted), requiring retrieval of the nearest stored pattern. While attention and fixed-state alternatives use a closed-form readout, Hopfield networks and Boltzmann machines (Ackley et al., 1985) motivate future use of iterative methods that optimize the query to the nearest minima in the energy landscape. Though, language models must maintain parallelizability across time.

F Training Details

All models were trained using the FLAME repository (Zhang & Yang, 2025), leveraging FLA (Yang & Zhang, 2024) baseline implementations. We trained three model sizes (120M, 400M, 850M), all using 24 layers and 32k-token Llama tokenizer (Touvron et al., 2023). We scaled up the width of these models to maintain similar training dynamics, we kept the head dimension constant at 256, ranging the number of heads between 2, 4, and 6. We carefully matched similar parameter and FLOP counts by maintaining $d_k = d_v = 256$. This scales the hidden sizes to 512, 1024, and 1536 respectively. The 120M and 400M models were trained on 15B tokens of 2k long sequences. The 850M models were trained on 30B tokens of 4k long sequences. For training, we used AdamW with peak learning rate of 4e-4 and cosine decay with a linear warmup of 2048 steps. We did not use weight decay except for the 850M Raven training run (where we use a value of 0.1) due to training instabilities; we did not find this choice to have a significant impact on other baselines. Mamba2 uses the same structure as other baselines by interleaving SwiGLU MLPs after each token mixer. For Urchin parameters, we use a chunk size of 256 tokens, meaning $\boldsymbol{z}_t$ is pre-computed over the next 256 tokens. At each time step, only 16 components are updated in Urchin out of the $D = 256$ components. GSA and Raven also use 256 components or slots. Finally, we report real-world training speeds for Urchin in Appendix G.

G Training speed

In this section, we measure the Urchin's training throughput for 400M models (24 layers, 1024 model dimension, 256 memory components / slots, 4 heads, and 256 head dimension). While we leverage a modifed GSA kernel from the FLA library, Urchin performs chunkwise parallelism to compute $\boldsymbol{z}_t$, increasing the number of kernel launches. This can be alleviated by capturing the CUDA graph, demonstrating comparable training speeds to other dual-state SSMs like Raven and GSA.

H Attention as the conditional mean of a Gaussian mixture

Proof of Eq. 6. Condition on the generating memory and use the tower property,

$$\mathbb{E}\big[\mathsf{V} \mid \mathsf{K} = \boldsymbol{q}_t\big] = \sum_{j=1}^{t} P(c_t = j \mid \mathsf{K} = \boldsymbol{q}_t)\, \mathbb{E}\big[\mathsf{V} \mid \mathsf{K} = \boldsymbol{q}_t, c_t = j\big]. \tag{18}$$

Given $c_t = j$, K and V are conditionally independent. Therefore,

$$\mathbb{E}[\mathsf{V} \mid \mathsf{K} = \boldsymbol{q}_t, c_t = j] = \mathbb{E}[\mathsf{V} \mid c_t = j] = \boldsymbol{v}_j. \tag{19}$$

Then for memory j,

$$P(c_t = j \mid \mathsf{K} = \boldsymbol{q}_t) = \frac{P(c_t = j)\,\mathcal{N}(\boldsymbol{q}_t; \boldsymbol{k}_j, \sigma^2 \mathbf{I}_{d_k})}{\sum_{i=1}^{t} P(c_t = i)\,\mathcal{N}(\boldsymbol{q}_t; \boldsymbol{k}_i, \sigma^2 \mathbf{I}_{d_k})} \tag{20}$$

$$= \frac{\pi_j \exp\big(-\|\boldsymbol{q}_t - \boldsymbol{k}_j\|_2^2 / 2\sigma^2\big)}{\sum_{i=1}^{t} \pi_i \exp\big(-\|\boldsymbol{q}_t - \boldsymbol{k}_i\|_2^2 / 2\sigma^2\big)}. \tag{21}$$

Expanding $\|\boldsymbol{q}_t - \boldsymbol{k}_j\|_2^2 = \|\boldsymbol{q}_t\|_2^2 + \|\boldsymbol{k}_j\|_2^2 - 2\boldsymbol{q}_t^\top \boldsymbol{k}_j$, the factor $\exp(-\|\boldsymbol{q}_t\|_2^2/2\sigma^2)$ can be factored out of every term and canceled. This gives Eq. 6. With $\pi_j = \exp(\|\boldsymbol{k}_j\|_2^2/2\sigma^2)$, the remaining norm terms cancel as well. □

I Understanding the Objective Terms

Throughout this appendix, θ_t is the parameter being solved for in Eq. 9, with $\theta_{t,j}$ being the new memory j and $\theta_{t-1,j}$ the previous one. Furthermore, the main text folds σ^2 into the learned quantities of the layer.

Joint and Conditional Likelihoods. Given $c_t = j$, the joint likelihood can be written as,

$$\log P(\boldsymbol{v}_t, \boldsymbol{k}_t \mid c_t = j, \theta_t) = -\frac{\|\boldsymbol{k}_t - \theta_{t,j}^{(1)}\|_2^2}{2\sigma^2} - \frac{\|\boldsymbol{v}_t - \theta_{t,j}^{(2)}\|_2^2}{2\sigma^2} + \mathsf{const}. \tag{22}$$

Taking the expectation over $c_t \sim \boldsymbol{z}_t$ weights the term for memory j by $\boldsymbol{z}_{t,j}$. This term pulls memory key j toward $\boldsymbol{k}_t$ and memory value j toward $\boldsymbol{v}_t$.

The conditional likelihood uses the same value terms as the joint,

$$\log P(\boldsymbol{v}_t, \boldsymbol{k}_t \mid c_t = j, \theta_t) = \log P(\boldsymbol{v}_t, \mid c_t = j, \theta_t^{(2)}) + \log P(\boldsymbol{k}_t \mid c_t = j, \theta_t^{(1)}) \tag{23}$$

$$\log P(\boldsymbol{v}_t, c_t = j \mid \boldsymbol{k}_t, \theta_t) = \log P(\boldsymbol{v}_t, \mid c_t = j, \theta_t^{(2)}) + \log P(c_t = j \mid \boldsymbol{k}_t, \theta_t^{(1)}). \tag{24}$$

The key term, however, becomes the posterior over memories given the key. By Bayes' rule with the key marginal of memory j and the mixture weights of Eq. 7,

$$P(c_t = j \mid \boldsymbol{k}_t, \theta_t^{(1)}) = \frac{P(c_t = j)\,\mathcal{N}(\boldsymbol{k}_t;\, \theta_{t,j}^{(1)},\, \sigma^2 \mathbf{I}_{d_k})}{\sum_{i=1}^{D} P(c_t = i)\,\mathcal{N}(\boldsymbol{k}_t;\, \theta_{t,i}^{(1)},\, \sigma^2 \mathbf{I}_{d_k})} = \mathrm{Softmax}_j\big(\theta_t^{(1)} \boldsymbol{k}_t / \sigma^2\big), \tag{25}$$

where the norm terms cancel as in the proof of Eq. 6. Its expectation under $\boldsymbol{z}_t$ is

$$\sum_{j=1}^{D} \boldsymbol{z}_{t,j} \log P(c_t = j \mid \boldsymbol{k}_t, \theta_t^{(1)}) = \sum_{j=1}^{D} \boldsymbol{z}_{t,j}\, \frac{\boldsymbol{k}_t^\top \theta_{t,j}^{(1)}}{\sigma^2} - \log \sum_{i=1}^{D} \exp\Big(\frac{\boldsymbol{k}_t^\top \theta_{t,i}^{(1)}}{\sigma^2}\Big), \tag{26}$$

the negative cross-entropy between $\boldsymbol{z}_t$ and the posterior, which equals $-\mathrm{KL}\big(\boldsymbol{z}_t \,\|\, P(c_t \mid \boldsymbol{k}_t, \theta_t^{(1)})\big)$ up to the entropy of $\boldsymbol{z}_t$. The two likelihoods differ only in this key term. The joint term moves $\theta_{t,j}^{(1)}$ toward the point $\boldsymbol{k}_t$, whereas the conditional term moves the posterior toward $\boldsymbol{z}_t$ and depends on $\theta_{t,j}^{(1)}$ only through $\boldsymbol{k}_t^\top \theta_{t,j}^{(1)}$.

Prior over Memories. We use a zero-mean Gaussian prior on each memory $\theta_{t,j} \in \mathbb{R}^{d_k+d_v}$, independent across memories, with block-diagonal covariance

$$\theta_{t,j} \sim \mathcal{N}\big(\mathbf{0}, \Sigma_t\big), \qquad \Sigma_t = \begin{bmatrix} \frac{1}{2\lambda_t^{(1)}}\mathbf{I}_{d_k} & \mathbf{0} \\ \mathbf{0} & \frac{1}{2\lambda_t^{(2)}}\mathbf{I}_{d_v} \end{bmatrix}, \tag{27}$$

which places separate isotropic variances on the key-side and value-side parameters. Both $\lambda_t^{(1)}, \lambda_t^{(2)} > 0$ are learned functions of the token $\boldsymbol{x}_t$. Summing the log density over memories,

$$-\log P(\theta_t) = \lambda_t^{(1)} \|\theta_t^{(1)}\|_F^2 + \lambda_t^{(2)} \|\theta_t^{(2)}\|_F^2 + \mathsf{const.}, \tag{28}$$

$$\nabla_{\theta_{t,j}^{(1)}}\big[-\log P(\theta_t)\big] = 2\lambda_t^{(1)}\,\theta_{t,j}^{(1)}, \qquad \nabla_{\theta_{t,j}^{(2)}}\big[-\log P(\theta_t)\big] = 2\lambda_t^{(2)}\,\theta_{t,j}^{(2)}. \tag{29}$$

The gradient points back toward the origin with a strength set by $\lambda_t^{(i)}$. In the minimizer of Eq. 9, this term acts as a decay on the previous parameter iteration, with one decay per state (Appendix J). A larger $\lambda_t^{(i)}$ decays more aggressively, because the prior expects θ to sit closer to the origin.

Divergence Between Previous and Updated Memories. The context retention term in Eq. 9 measures the divergence between the updated memory j and the previous memory j. For the joint likelihood, the data is (K, V), and the two densities are Gaussians with means $\theta_{t,j}$ and $\theta_{t-1,j}$ and the same covariance $\sigma^2\mathbf{I}$. By definition of the KL divergence,

$$\begin{aligned}
&\mathrm{KL}\big(P(\mathsf{K},\mathsf{V} \mid c_t = j, \theta_t) \,\|\, P(\mathsf{K},\mathsf{V} \mid c_t = j, \theta_{t-1})\big) \\
&= \mathbb{E}_{(\mathsf{K},\mathsf{V})\sim P(\cdot \mid c_t = j, \theta_t)}\Big[\log \frac{P(\mathsf{K},\mathsf{V} \mid c_t = j, \theta_t)}{P(\mathsf{K},\mathsf{V} \mid c_t = j, \theta_{t-1})}\Big] && (30)\\
&= \mathbb{E}\big[\|[\mathsf{K},\mathsf{V}] - \theta_{t-1,j}\|_2^2 - \|[\mathsf{K},\mathsf{V}] - \theta_{t,j}\|_2^2\big]/2\sigma^2 && (31)\\
&= \mathbb{E}\big[\|\theta_{t-1,j}\|_2^2 - \|\theta_{t,j}\|_2^2 - 2[\mathsf{K},\mathsf{V}]^\top(\theta_{t-1,j} - \theta_{t,j})\big]/2\sigma^2 && (32)\\
&= \big(\|\theta_{t-1,j}\|_2^2 - \|\theta_{t,j}\|_2^2 - 2\,\mathbb{E}\big[[\mathsf{K},\mathsf{V}]\big]^\top(\theta_{t-1,j} - \theta_{t,j})\big)/2\sigma^2 && (33)\\
&= \big(\|\theta_{t-1,j}\|_2^2 - \|\theta_{t,j}\|_2^2 - 2\,\theta_{t,j}^\top\theta_{t-1,j} + 2\|\theta_{t,j}\|_2^2\big)/2\sigma^2 && (34)\\
&= \|\theta_{t,j} - \theta_{t-1,j}\|_2^2/2\sigma^2. && (35)
\end{aligned}$$

Holding the model near its previous version is therefore the same as holding each memory near its previous position.

For the conditional likelihood, the data is (V, c_t) conditioned on the observed key $\boldsymbol{k}_t$. This divergence splits into separate value and key parts,

$$\mathrm{KL}\big(P(\mathsf{V}, c_t \mid \boldsymbol{k}_t, \theta_t) \,\|\, P(\mathsf{V}, c_t \mid \boldsymbol{k}_t, \theta_{t-1})\big) = \mathrm{KL}\big(P(c_t \mid \boldsymbol{k}_t, \theta_t^{(1)}) \,\|\, P(c_t \mid \boldsymbol{k}_t, \theta_{t-1}^{(1)})\big) + \ldots \tag{36}$$

$$\ldots \sum_{j=1}^{D} P(c_t = j \mid \boldsymbol{k}_t, \theta_t^{(1)})\,\mathrm{KL}\big(P(\mathsf{V} \mid c_t = j, \theta_t^{(2)}) \,\|\, P(\mathsf{V} \mid c_t = j, \theta_{t-1}^{(2)})\big) \tag{37}$$

Following the logic above, the value-side divergence of memory j is the Euclidean distance between $\theta_{t,j}^{(2)}$ and $\theta_{t-1,j}^{(2)}$. Eq. 9 sums these over memories with equal weight in place of the routing weights, so that every memory value is held near its previous position regardless of where the current key is routed. The key-side divergence ensures that the component distribution given the key stays the same. This can help prevent changing a key's routing, which would ultimately leave its associated value stranded in the wrong component.

J Derivation of Urchin

Here, we derive Urchin from several design choices. We chose to maximize the conditional likelihood, using a zero-mean Gaussian prior, with the Reverse KL divergence. Plugging these into Eq. 9

we minimize,

$$\min_{\theta_t} \ \beta_t \sum_{j=1}^{D} \boldsymbol{z}_{t,j} \, \frac{\|\boldsymbol{v}_t - \theta_{t,j}^{(2)}\|_2^2}{2\sigma^2} + (1-\beta_t) \sum_{j=1}^{D} \frac{\|\theta_{t,j}^{(2)} - \theta_{t-1,j}^{(2)}\|_2^2}{2\sigma^2} \tag{38}$$

$$+\beta_t \, \mathrm{KL}\big(P(c_t \mid \boldsymbol{k}_t, \theta_t^{(1)}) \,\|\, \boldsymbol{z}_t\big) + \big(1-\beta_t\big) \, \mathrm{KL}\big(P(c_t \mid \boldsymbol{k}_t, \theta_t^{(1)}) \,\|\, P(c_t \mid \boldsymbol{k}_t, \theta_{t-1}^{(1)})\big) \tag{39}$$

$$+\lambda_t^{(1)} \|\theta_t^{(1)}\|_F^2 + \lambda_t^{(2)} \|\theta_t^{(2)}\|_F^2. \tag{40}$$

Here, $\boldsymbol{z}_t$ is the state-aware component distribution of Eq. 13. We can solve for the key- and value-side states separately.

The Value-Side Update. Setting the gradient with respect to $\theta_{t,j}^{(2)}$ to zero,

$$0 = \beta_t \boldsymbol{z}_{t,j} \, \frac{\theta_{t,j}^{(2)} - \boldsymbol{v}_t}{\sigma^2} + (1-\beta_t) \, \frac{\theta_{t,j}^{(2)} - \theta_{t-1,j}^{(2)}}{\sigma^2} + 2\lambda_t^{(2)} \, \theta_{t,j}^{(2)} \tag{41}$$

$$\theta_{t,j}^{(2)} = \frac{(1-\beta_t)}{(1-\beta_t) + \beta_t \boldsymbol{z}_{t,j} + 2\sigma^2 \lambda_t^{(2)}} \, \theta_{t-1,j}^{(2)} + \frac{\beta_t \boldsymbol{z}_{t,j}}{(1-\beta_t) + \beta_t \boldsymbol{z}_{t,j} + 2\sigma^2 \lambda_t^{(2)}} \, \boldsymbol{v}_t. \tag{42}$$

The fit term pulls memory j towards $\boldsymbol{v}_t$ with weight $\beta_t \boldsymbol{z}_{t,j}$, the retention term pulls it back towards $\theta_{t-1,j}^{(2)}$ with weight $1-\beta_t$, and the prior pulls it towards the origin with weight $2\sigma^2\lambda_t^{(2)}$. These three anchors decide the minimizer.

Deviating from the theory, we parametrize the gate function in the same way as previous Mamba2 and Gated DeltaNet. Rather than directly computing $\lambda_t^{(2)}$, we mimic its effects with $\alpha_t^{(2)}$ as in

$$\theta_{t,j}^{(2)} = \frac{(1-\beta_t)}{(1-\beta_t) + \beta_t \boldsymbol{z}_{t,j}} \, \alpha_t^{(2)} \theta_{t-1,j}^{(2)} + \frac{\beta_t \boldsymbol{z}_{t,j}}{(1-\beta_t) + \beta_t \boldsymbol{z}_{t,j}} \, \boldsymbol{v}_t. \tag{43}$$

The Key-Side Update. The routing, or component distribution given the key, only depends on $\theta_{t,j}^{(1)}$ through the logit $\boldsymbol{k}_t^\top \theta_{t,j}^{(1)}$, as in

$$P(c_t = j \mid \boldsymbol{k}_t, \theta_t^{(1)}) = \mathrm{Softmax}_j(\theta_t^{(1)} \boldsymbol{k}_t / \sigma^2). \tag{44}$$

Putting the prior aside for simplicity, the KL terms in the objective simplify to

$$\sum_{j=1}^{D} P(c_t = j \mid \boldsymbol{k}_t, \theta_t^{(1)}) \Big[\log P(c_t = j \mid \boldsymbol{k}_t, \theta_t^{(1)}) - \beta_t \log \boldsymbol{z}_{t,j} - (1-\beta_t) \log P(c_t = j \mid \boldsymbol{k}_t, \theta_{t-1}^{(1)}) \Big]. \tag{45}$$

The optimal routing is the weighted average between the two targets, $\boldsymbol{z}_t$ and $P(c_t \mid \boldsymbol{k}_t, \theta_{t-1}^{(1)})$, in log-space,

$$\log P(c_t = j \mid \boldsymbol{k}_t, \theta_t^{(1)}) = \beta_t \log \boldsymbol{z}_{t,j} + (1-\beta_t) \log P(c_t = j \mid \boldsymbol{k}_t, \theta_{t-1}^{(1)}) + \mathsf{const}. \tag{46}$$

Each distribution here is a softmax, so its log is its logit divided by σ^2 up to a constant,

$$\boldsymbol{k}_t^\top \theta_{t,j}^{(1)} = \boldsymbol{k}_t^\top \theta_{t-1,j}^{(1)} + \beta_t \big(\boldsymbol{l}_{t,j} - \boldsymbol{k}_t^\top \theta_{t-1,j}^{(1)} \big) + \mathsf{const}. \tag{47}$$

In log-space, the learning rate β_t controls how far the new component distribution moves from its previous iteration to the new target logit $\boldsymbol{l}_{t,j}$. Eq. 47 only constrains each memory key along $\boldsymbol{k}_t$, so we choose the smallest change from the previous key under the retention weight and the prior together, $(1-\beta_t)\|\theta_{t,j}^{(1)} - \theta_{t-1,j}^{(1)}\|_2^2/2\sigma^2 + \lambda_t^{(1)}\|\theta_{t,j}^{(1)}\|_2^2$, subject to Eq. 47. The minimizer decays the previous key by $\alpha_t^{(1)} = (1-\beta_t)/\big((1-\beta_t) + 2\sigma^2\lambda_t^{(1)}\big)$ and restores the decayed logit along $\boldsymbol{k}_t$,

$$\theta_{t,j}^{(1)} = \alpha_t^{(1)} \theta_{t-1,j}^{(1)} + \Big[\beta_t \big(\boldsymbol{l}_{t,j} - \boldsymbol{k}_t^\top \theta_{t-1,j}^{(1)} \big) + \big(1 - \alpha_t^{(1)}\big) \, \boldsymbol{k}_t^\top \theta_{t-1,j}^{(1)} \Big] \, \boldsymbol{k}_t. \tag{48}$$

Deviating from the theory, we parametrize $\alpha_t^{(1)}$ the same way as Mamba2 and GDN and simplify the key-side update

$$\theta_{t,j}^{(1)} = \alpha_t^{(1)} \theta_{t-1,j}^{(1)} + \beta_t \big(\boldsymbol{l}_{t,j} - \boldsymbol{k}_t^\top \theta_{t-1,j}^{(1)} \big) \, \boldsymbol{k}_t. \tag{49}$$

K Previous Layers as Minimizers of the Objective

With this objective, we show that previous layers such as Raven, GSA, and ABC implicitly maximize the joint likelihood with a uniform prior. Afzal et al. generalize this class of models as "Routing Slot Memories". We first show how this class is a subclass of our objective. Then, we look at each method from the perspective of parametric Gaussian mixture modeling.

Maximizing the Joint Likelihood. A Routing Slot Memory keeps D memory slots and routes the current pair into them with a router $\boldsymbol{r}_t \in [0,1]^D$ computed from the token $\boldsymbol{x}_t$. Each slot moves towards the pair by its routing weight,

$$\theta^{(1)}_{t,j} = (1 - r_{t,j})\,\theta^{(1)}_{t-1,j} + r_{t,j}\,\boldsymbol{k}_t, \qquad \theta^{(2)}_{t,j} = (1 - r_{t,j})\,\theta^{(2)}_{t-1,j} + r_{t,j}\,\boldsymbol{v}_t, \tag{50}$$

and the readout is Eq. 8. A slot with $r_{t,j} = 0$ is left untouched, and a slot with $r_{t,j} = 1$ is overwritten.

Under the joint likelihood and a uniform prior, Appendix I turns Eq. 9 into a sum of squared distances, one per slot,

$$\beta_t \boldsymbol{z}_{t,j}\,\|[\boldsymbol{k}_t, \boldsymbol{v}_t] - \theta_{t,j}\|_2^2 + (1 - \beta_t)\,\|\theta_{t,j} - \theta_{t-1,j}\|_2^2. \tag{51}$$

The fit term pulls slot j towards the pair with weight $\beta_t \boldsymbol{z}_{t,j}$, and the retention term pulls it back towards its previous position with weight $1 - \beta_t$. Setting the gradient to zero, the minimizer is Eq. 50 with

$$r_{t,j} = \frac{\beta_t \boldsymbol{z}_{t,j}}{(1 - \beta_t) + \beta_t \boldsymbol{z}_{t,j}}. \tag{52}$$

Therefore, a router is a choice of component distribution and learning rate. Under the perspective of Gaussian mixture modeling, the router is the mixture weight of each component. Rather than computing the probability of each component given the key-value pair, as we do with Urchin, previous methods compute the routing from the token $\boldsymbol{x}_t$ alone. We refer to this as state-agnostic routing.

ABC. ABC routes every pair to every slot with a softmax over time,

$$r_{t,j} = \frac{\exp(\boldsymbol{w}_j^\top \boldsymbol{x}_t)}{\sum_{i \le t} \exp(\boldsymbol{w}_j^\top \boldsymbol{x}_i)}. \tag{53}$$

Therefore each memory is the weighted average of every pair seen so far,

$$\theta_{t,j} = \frac{\sum_{s \le t} \exp(\boldsymbol{w}_j^\top \boldsymbol{x}_i)\,[\boldsymbol{k}_i, \boldsymbol{v}_i]}{\sum_{s \le t} \exp(\boldsymbol{w}_j^\top \boldsymbol{x}_i)}. \tag{54}$$

With $\exp(\boldsymbol{w}_j^\top \boldsymbol{x}_s)$ playing the role of the responsibility of pair s under component j, this is the M-step of EM for a Gaussian mixture, computed online. ABC is the exact maximum likelihood estimate over all past tokens for each memory given its responsibilities. In Eq. 51, this corresponds to a retention weight that grows with the responsibilities of every past pair a slot has absorbed, so that every past pair counts as much as the current one. This is what makes the estimate exact, and it is also what prevents recall over long contexts. Each memory grows rigid, and a new observation can no longer move it. Our objective keeps both terms of Eq. 51 and learns the learning rate β_t instead, which is what gives Urchin its plasticity.

GSA. GSA determines the routing with a sigmoid per memory,

$$r_{t,j} = 1 - \sigma(\boldsymbol{w}_j^\top \boldsymbol{x}_t)^{1/\tau}, \tag{55}$$

with a temperature τ shared across slots. Compared to ABC, the retention weight no longer grows with the number of pairs a slot has absorbed. Each memory is an exponentially weighted average of its past pairs, so old pairs are forgotten at a rate set by the token. This restores plasticity. However, every $r_{t,j}$ is positive, so every slot absorbs a share of every pair if $\boldsymbol{w}_j$ is not trained properly. The memory components drift towards a moving average of the context, and a rare pair can be overwritten future pairs.

Raven. Raven routes each token to a top-k subset S_t of slots with normalized sigmoid scores

$$r_{t,j} = 1 - \exp(-g_{t,j}\, a_t), \qquad g_{t,j} = \frac{\sigma(\boldsymbol{w}_j^\top \boldsymbol{x}_t)\, \mathbb{1}_{[j \in S_t]}}{\sum_{i \in S_t} \sigma(\boldsymbol{w}_i^\top \boldsymbol{x}_t)}, \qquad a_t = A \,\mathrm{softplus}(\boldsymbol{w}_a^\top \boldsymbol{x}_t + b). \quad (56)$$

A slot outside S_t has $r_{t,j} = 0$, so by Eq. 52 its component weight is $\boldsymbol{z}_{t,j} = 0$. Compared to GSA, the responsibilities are restricted to k components. This is in-spirit of the sparse E-step of Neal & Hinton (1998), and we adopt the same choice for Urchin.