GateRank: A linear attention extension with learnable rank structure inspired by fast and slow synapses

Constantin Kronbichler [1] **Jan Finkbeiner** [1 2] **Emre Nefcti** [1 2]

Abstract

Long-context sequence modeling demands mechanisms that retain and selectively access information across diverse timescales. Biological neural systems achieve this through fast and slow synaptic dynamics, enabling rapid context adaptation alongside stable long-term memory. In contrast, state-space and linear attention models typically compress history into a single globally decaying state per head, enforcing a 1-semiseparable (rank-1) temporal structure that limits granular retrieval and associative recall. We introduce GateRank, a multi-timescale extension to linear attention that maintains fast and slow memory states and uses an input-dependent gate to consolidate information and reset the fast state, relaxing the 1-semiseparable decay constraint while preserving streaming inference and chunk-parallel training. Integrated with Mamba-2 and Gated DeltaNet and supported by efficient Triton kernels, GateRank achieves higher throughput than FlashAttention at 32k tokens with modest overhead relative to the base models. With this we train models with 760M parameters that match or slightly improves standard language modeling results and yields significant gains on retrieval-heavy tasks such as Needle-In-A-Haystack, as well as improved learning on synthetic structured-memory tasks. These findings show that biologically inspired multi-timescale memory can extend the expressive capacity of linear-time sequence models without sacrificing efficiency.

1. Introduction

Natural language exhibits both short-range structure and long-distance dependencies, a challenge already emphasized in classical linguistic theory (Chomsky, 1956). Biological neural systems address similar demands during natural sequence processing by maintaining information across multiple temporal scales. In particular, neuroscience highlights that synapses operate with distinct update dynamics: fast synaptic mechanisms enable rapid context adaptation and flexible working memory, while slow synaptic processes consolidate information across longer timescales for stable retrieval and structured computation (Zenke et al., 2017; Tetzlaff et al., 2012). These multi-timescale mechanisms provide a compelling blueprint for machine learning architectures tasked with processing long sequences efficiently.

Modern high-performing sequence models reflect aspects of this biological perspective as both transformers (Finkbeiner & Neftci, 2025; Ellwood, 2024) and particularly linear attention and state-space models can naturally be viewed from the perspective of synaptic plasticity (Leroux et al., 2025; Vermani et al., 2025) and fast-weight mechanisms (Ba et al., 2016; Schlag et al., 2021). Softmax-based self-attention offers granular and content-selective retrieval, by maintaining a full list of past keys and values (Dao et al., 2022; Dao, 2023), but exhibits quadratic computational cost in sequence length. Linear attention and state-space models (SSMs) mitigate this cost by compressing history (key-value pairs) into a fixed memory state. By accumulating key–value interactions through a recurrent update rather than storing all past interactions, these models enable streaming inference and chunk-parallel training (Katharopoulos et al., 2020). Recent architectures such as RetNet (Sun et al., 2023), GLA (Yang et al., 2024), Mamba-2 (Dao & Gu, 2024), and Gated DeltaNet (Yang et al., 2025a) enhance this mechanism by introducing input-conditional gating that controls memory decay, leading to strong performance at long sequence lengths and competitive results with Transformers (Touvron et al., 2023; Brown et al., 2020).

However, despite their success, a core limitation remains: current linear attention architectures apply a global decay per head that enforces a rank-1 dependency structure over time. Under the formulation introduced in Mamba-2 (Dao & Gu, 2024), these models exhibit 1-semiseparable state dynamics, meaning each token's influence decays with a fixed functional form shared across the entire temporal horizon.

[*]Equal contribution [1]Forschungszentrum Jülich [2]RWTH Aachen University. Correspondence to: Constantin Kronbichler <ckronbichler@proton.me>, Jan Finkbeiner <j.finkbeiner@fz-juelich.de>.

Consequently, the memory cannot express heterogeneous decay patterns, such as protecting certain distant information while rapidly forgetting others, and lacks the granular retrieval and associative recall characteristic of self-attention (Arora et al., 2023). This constraint makes these models efficient, but potentially too rigid, limiting their ability to represent circuit-like compositions (Merrill & Sabharwal, 2023; Merrill et al., 2025; Grazzi et al., 2025) and the selective retention seen in biological systems.

Motivated by the role of fast and slow synapses in biological memory, we introduce GateRank, a multi-timescale extension to linear attention. GateRank maintains two coupled memory states: a fast synaptic state that rapidly integrates incoming information and a slow synaptic state that selectively captures persistent patterns. An input-dependent gate controls the transfer from fast to slow memory and resets the fast state when consolidation occurs. This mechanism breaks the strict 1-semiseparable decay constraint by allowing token-dependent transitions between timescales, enabling different tokens to follow distinct decay trajectories. In doing so, GateRank preserves the streaming efficiency of linear attention while expanding its expressive capacity and memory selectivity. We implement GateRank as an additive module on top of existing architectures, demonstrating integration with Mamba-2 (Dao & Gu, 2024) and Gated DeltaNet (Yang et al., 2025a). To support practical training, we develop efficient Triton kernels (Tillet et al., 2019) and a chunked-scan algorithm that maintains high throughput with modest overhead and achieves faster training than FlashAttention (Dao, 2023) at long context lengths.

Empirically, GateRank matches or slightly exceeds the performance of the underlying architectures on standard language modeling and common sense reasoning tasks (Gao et al., 2024; Biderman et al., 2024), while providing consistent improvements on retrieval-heavy benchmarks such as Needle-In-A-Haystack (Hsieh et al., 2024). On synthetic structured-memory tasks, GateRank improves learning stability and generalization, supporting the hypothesis that multi-timescale synaptic dynamics increase representational complexity beyond 1-semiseparable models. Together, these results suggest that borrowing architectural priors from biological memory systems enables more expressive, selective, and robust long-context processing without sacrificing the efficiency benefits of linear attention.

Summarizing, our contributions are the following:

- Biologically inspired multi-timescale memory mechanism introducing fast and slow synapses extending linear attention and state-space models
- Relaxation of the 1-semiseparable decay constraint, enabling token-dependent memory consolidation and reset
- Efficient Triton kernels and chunked-scan implementation supporting scalable training and streaming inference
- Empirical gains in associative recall, retrieval, and structured reasoning, with competitive language modeling performance for models with 760M parameters
- Demonstrated compatibility with Mamba-2 and Gated DeltaNet, suggesting general applicability to linear-time architectures

2. Related work

We give a brief summary of the different approaches to examining and enhancing the memory management of linear attention models.

Replicating softmax' low-entropy query-key interactions One advantage of the softmax over linear attention models is the low-entropy attention matrix, i.e. softmax $\left(QK^T/\sqrt{d_k}\right)$, meaning that only very few query-key relations become very large while most of them stay very low (Zhang et al., 2024). This results in an unstructured attention matrix unlike in linear attention where the stronger relations between keys thanks to the common decay render memory accesses less granular. Since the introduction of linear attention attempts have been made to approximate the low-entropy behavior of the softmax via kernel methods, e.g. in the Performer models (Choromanski et al., 2022). The most successful approach to this problem was the use of a second order Taylor expansion as described in in the Based model (Arora et al., 2025). Recently, the DeltaFormer architecture has shown that such kernel methods could also be a useful addition for Gated DeltaNet (Zhong et al., 2025). While our work doesn't use any feature maps to approximate exponentials, it could potentially be a way to obtain less entropy ("spikier") attention weights by breaking up the monotonicity of the decay dynamics.

Gating in linear attention / RNNs / state-space models (SSMs) Initial linear RNNs used input-independent gating, such as models S4 (Gu et al., 2022), S5 (Smith et al., 2023), LRU (Orvieto et al., 2023), RWKV 4/5 (Peng et al., 2023) or RetNet (Sun et al., 2023). In subsequent works, input-dependent gating was added to state-space models as in Mamba (Gu & Dao, 2024) which required some restrictions to the decay matrix to be parallelizable over the sequence length via a scan. Subsequent works established more parallelizable RNN dynamics by simplifying the decay to be able to better utilize the matrix-matrix multiplication units on GPUs, such as Mamba2 (Dao & Gu, 2024), GLA (Yang et al., 2024), and RWKV 6 (Peng et al., 2024). The parallelization enabled efficient training with larger state-sizes, thereby increasing the memory capacity, and the associative

recall capabilities. However, the decayed outer product accumulation could lead to overwriting the state unlike fixed size sliding-window attention methods that can maintain all information from the past keys and values, it stores. Only by using the delta rule to orthogonalize the state updates, the full capacity of the hidden state can be used. The use of the delta-rule in linear attention was proposed in the fast-weight programmer approach (Schlag et al., 2021) and improved to be parallelizable with a chunk scan in DeltaNet (Yang et al., 2025b). Combining this with the Mamba2-style gating mechanism in Gated DeltaNet (Yang et al., 2025a) led to better language modeling performance while maintaining the more effective use of the state size. Similar update rules include RWKV 7 (Peng et al., 2025) and LongHorn which derives its rule from a an online learning perspective of linear attention (Liu et al., 2024).

Complexity class analysis of attention mechanisms Another way to compare attention mechanisms is to analyse the complexity class of boolean circuits they can represent. It has been shown that softmax-based attention is restricted to computations with simpler circuits that lie in the TC^0 class (Merrill & Sabharwal, 2023). Most linear attention models also lie in TC^0 (Merrill et al., 2025). Yet, the delta-rule has been shown to be able to represent more complex circuits in NC^1 and, hence, should be able to perform more complex reasoning in a single forward pass thanks to their diagonal plus rank-1 state-transition structure (Grazzi et al., 2025). Note that problems with NC^1 circuit complexity can only be solved via gradient descent learning when allowing negative eigenvalues of the state-transition matrix (Grazzi et al., 2025). Finally, even more complex structure of the state-transition matrix can lead to higher circuit complexity, e.g. via the application of multiple householder products has led to even higher circuit complexity of these models (Siems et al., 2025) or the use of a product of a column one-hot matrix and a complex-valued diagonal matrix (Terzić et al., 2025). The downside of more complex transition matrices is the throughput penalty of current implementations, as in (Cirone & Salvi, 2025), because they require indexing operations that do not align well with the Triton way to write GPU kernels (Tillet et al., 2019). Others suffer from lacking a chunk scan implementation and fall back to a less efficient parallel scan (Terzić et al., 2025).

The idea of higher rank transition matrices is very related to the idea of breaking up the rank-1 structure in the decay mask in GateRank. However, GateRank is different from the state updates based on householder products: the latter leads to a different structure of the hidden dimension while maintaining the rank-1 structure across the temporal dimension. GateRank, on the other hand, introduces a more complex structure of the temporal dimension. The difference between the hidden state structure and the temporal structure can be described with the tensor-centric view of state transitions introduced in (Guo et al., 2025).

Unstructured masking for linear attention models With log-linear attention there has been a concurrent attempt to enhance linear attention models by breaking up the monotone structure of the decay mask (Guo et al., 2025). While GateRank's state-size is still fixed with respect to the sequence length, albeit doubled with respect to the baseline linear attention model, log-linear attention uses a pre-defined split into decay segments that leads to logarithmic memory growth during auto-regressive inference. The main difference between GateRank and log-linear attention is the learnable location of decay segment boundaries in GateRank.

3. Methods

3.1. Preliminary: Mamba2 style gating

Mamba2 adds a gating mechanism to linear attention for input-conditioned forgetting (Dao & Gu, 2024).

Recurrent form The state update and output computation are described by the following state-space representation which are commonly referred to as the recurrent form for linear attention.

$$\boldsymbol{S}_t = \alpha_t \boldsymbol{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^T \tag{1}$$

$$\boldsymbol{o}_t = \boldsymbol{q}_t^T \boldsymbol{S}_t \tag{2}$$

The input-conditioned decay α_t differentiates Mamba2 from the linear transformer (Katharopoulos et al., 2020). It is a scalar that is applied across the entire hidden state and is computed in log-space via a projection with softplus activation: $\log(\alpha_t) = -A\,\text{softplus}(W_\alpha x_t + b_\alpha)$ (A is scalar constant learned via gradient descent), ensuring $\alpha_t \in (0, 1]$ for stable exponential decay.

Parallel form The state-space duality (SSD) of Mamba2 describes another way to phrase the linear attention dynamics: the parallel form that makes use of a mask matrix Γ (Dao & Gu, 2024).

$$\mathbf{O} = (\mathbf{Q}\mathbf{K}^T \odot \Gamma \odot \mathbf{M})\mathrm{V} \tag{3}$$

where $\mathbf{M}$ is the causal mask. Cumulative products of gates $\gamma_i = \prod_{j=1}^{i} \alpha_j$ define the mask Γ as

$$\Gamma_{i,j} = \frac{\gamma_i}{\gamma_j} \tag{4}$$

Figure 1a shows an example where each position accumulates products of all previous α gates, creating the characteristic rank-1 triangular structure. Crucially, SSD shows

that a state-space representation can be converted to a parallel form if the state transition matrix of the state-space representation is a scalar, which results in a 1-semiseparable (1-SS) decay mask Γ, meaning that every sub-matrix below the diagonal has rank at most 1. The reason for the 1-SS structure is that every element is a division of two scalars and hence each sub-matrix on or below the diagonal can be expressed as a single outer product (division in equation 4 is multiplication by reciprocal), the definition of a rank-1 matrix.

Chunk scan Furthermore, SSD has a third representation that makes use of the 1-SS structure of the mask to scale linearly with sequence length, unlike the quadratic scaling of the parallel form. As illustrated in Figure 1b, the chunk scan implementation computes a linear recurrence across chunks of sub-matrices below the diagonal. Because of the 1-SS nature of the chunks below the diagonal, a single chunk can be computed sub-quadratically with a matrix multiplication as demonstrated in the structured mask attention implementation of Mamba2 (Dao & Gu, 2024) and flash linear attention (Yang et al., 2024). For the chunks on the diagonal, the parallel form is used (Yang et al., 2024).

We denote $\mathbf{Q}_{[t]}$ as the query block for chunk t and $\boldsymbol{q}^r_{[t]}$ for the r-th query within chunk t, with full indexing details provided in the GateRank section. The chunk-level recurrence is:

$$\mathbf{S}_{[t+1]} = \overrightarrow{\mathbf{S}_{[t]}} + \mathbf{V}^\intercal_{[t]}\overrightarrow{\mathbf{K}_{[t]}} \tag{5}$$

$$\mathbf{O}_{[t]} = \overleftarrow{\mathbf{Q}_{[t]}}\mathbf{S}^\intercal_{[t]} + \left(\mathbf{Q}_{[t]}\mathbf{K}^\intercal_{[t]} \odot \Gamma_{[t]} \odot \mathbf{M}\right)\mathbf{V}_{[t]} \tag{6}$$

using the notation from Gated DeltaNet (Yang et al., 2025a) with $(\Gamma_{[t]})_{ij} = \frac{\gamma^i_{[t]}}{\gamma^j_{[t]}}, \gamma^j_{[t]} = \prod_{j=tC+1}^{tC+j} \alpha_j$. The left arrow ($\overleftarrow{\cdot}$) and right arrow ($\overrightarrow{\cdot}$) denote variables with decay factors applied to enable chunk-parallel computation while maintaining equivalence to the recurrent form:

$$\begin{aligned}
\overleftarrow{\boldsymbol{q}^r_{[t]}} &= \gamma^r_{[t]}\boldsymbol{q}^r_{[t]} && \text{(decay to first position of chunk)}\\
\overrightarrow{\boldsymbol{k}^r_{[t]}} &= \frac{\gamma^C_{[t]}}{\gamma^r_{[t]}}\boldsymbol{k}^r_{[t]} && \text{(decay to last position of chunk)}\\
\overrightarrow{\mathbf{S}_{[t]}} &= \gamma^C_{[t]}\mathbf{S}_{[t]} && \text{(decay over entire chunk)}
\end{aligned} \tag{7}$$

This gating is not exclusively used in Mamba2 but can also be added to the delta rule as shown in Gated DeltaNet (Yang et al., 2025a). Hence, it suffices to show how GateRank modifies the gating of Mamba2 to understand how GateRank can be implemented on top of both Mamba2 and Gated DeltaNet.

3.2. GateRank

GateRank splits the Mamba2-like gated state update into a hierarchy of two states: a state with fast updates that accumulates key-value outer products at every timestep, just like the hidden state in Mamba2, and, a slow state that sparsely accumulates the fast state. Whenever the fast state is added to the slow state, the fast state is reset to zero.

Gate computation Both gates are computed in log-space. The slow state gate is computed via a projection followed by a short convolution of length 4 and a ReLU activation:

$$\log(\alpha_t) = -A_{\text{fast}}\,\text{ReLU}\left(\text{ShortConv}\left(W_\alpha x_t + b_\alpha\right)\right) \tag{8}$$

where A_{fast} is a scalar decay coefficient, learned via gradient descent. Since ReLU outputs are non-negative, $\log(\alpha_t) = 0$ corresponds to $\alpha_t = 1$ (no decay), while $\log(\alpha_t) > 0$ corresponds to $\alpha_t > 1$ which, after normalization in cumulative products, creates decay boundaries that we call *reset segments*. When $\log(\alpha_t) \neq 0$, the slow state is updated by accumulating the fast state, and the fast state is reset to zero.

The fast decay β_t is computed just like the decay in Mamba2 via `softplus`:

$$\log(\beta_t) = -A_{\text{slow}}\text{softplus}(W_\beta x_t + b_\beta) \tag{9}$$

ensuring continuous decay within reset segments.

Recurrent form The recurrent formulation is based on two states with different gates α_t and β_t:

$$\mathbf{S}^{\text{slow}}_t = \alpha_t\mathbf{S}^{\text{slow}}_{t-1} + \begin{cases}\mathbf{S}^{\text{fast}}_t & \text{if } \log(\alpha_t) \neq 0\\ 0 & \text{otherwise}\end{cases} \tag{10}$$

$$\mathbf{S}^{\text{fast}}_t = \left(\begin{cases}\beta_t\mathbf{S}^{\text{fast}}_{t-1} & \text{if } \log(\alpha_t) = 0\\ 0 & \text{otherwise}\end{cases}\right) + \boldsymbol{k}_t\boldsymbol{v}^T_t \tag{11}$$

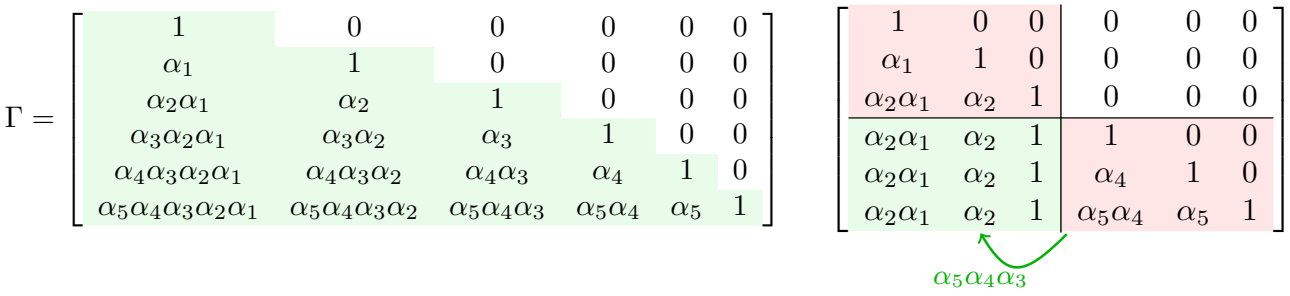
$$\boldsymbol{o}_t = \boldsymbol{q}^T_t(\mathbf{S}^{\text{slow}}_t + \mathbf{S}^{\text{fast}}_t) \tag{12}$$

See Figure 3 for an illustration of the recurrent form of GateRank.

Parallel form The parallel form multiplies by two mask matrices as shown in Figure 2a:

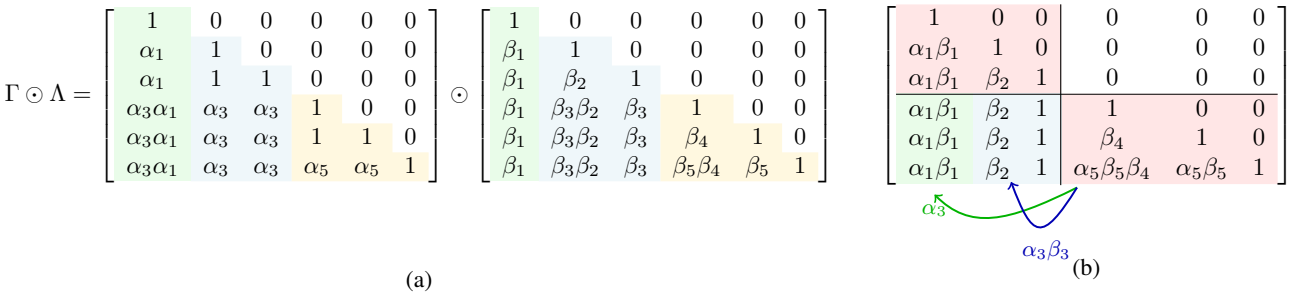
$$\mathbf{O} = (\mathbf{Q}\mathbf{K}^T \odot \Gamma \odot \Lambda \odot \mathbf{M})\mathrm{V} \tag{13}$$

where $\Gamma_{i,j} = \frac{\gamma_i}{\gamma_j}$ using $\gamma_i = \prod_{j=1}^{i}\alpha_j$ as in Mamba2. The mask Λ encodes the higher-rank structure from reset segments. Figure 2a shows an example with resets at positions $\{1, 3, 5\}$ creating three reset segments colored distinctly: positions 1 (green), positions 2-3 (blue), and positions 4-6 (yellow). Within each reset segment, β gates create local cumulative products.



(a) (b)

Figure 1. (a) Mamba2 decay mask structure for parallel form: all positions have decay coefficients α_t applied, resulting in a standard rank-1 semi-separable structure where $\Gamma_{i,j} = \gamma_i/\gamma_j$ with $\gamma_i = \prod_{j=1}^{i} \alpha_j$. *(b)* Illustration of the intra-chunk decay application in Mamba2's chunk scan. The intra-chunk decay application only requires a single state materialized at the chunk boundary since each column has to be multiplied with the same gate as every other column on its row, unlike in GateRank's intra-chunk scan, visualized in Figure 2.



(a) (b)

Figure 2. (a) GateRank decay mask structure for parallel form: the slow state is decayed only at $t \in \{1, 3, 5\}$ (where $\log(\alpha_t) \neq 0$) indicating the reset segment boundaries. At other positions, $\log(\alpha_t) = 0$ corresponding to $\alpha_t = 1$ (no decay). *(b)* Illustration of the intra-chunk decay application in GateRank's chunk scan. Because the reset segments incorporated into the slow state (green) and the final reset segment (blue) require different intra-chunk decay updates, GateRank materializes the slow and the fast state at the chunk boundary.

To define Λ, we introduce $r_{\text{next}}(i, l)$ which returns the start of the next reset segment after position i up to position l:

$$r_{\text{next}}(i,l) = \begin{cases} \min\{t \in [i,l] \mid \log(\alpha_t) \neq 0\} & \text{if such } t \text{ exists} \\ l & \text{otherwise} \end{cases} \tag{14}$$

The local cumulative product within each reset segment is:

$$\lambda_i = \prod_{j=s_i}^{i} \beta_j \tag{15}$$

$$s_i = \begin{cases} 1 + \max\{t \leq i \mid \log(\alpha_t) \neq 0\} & \text{if such } t \text{ exists} \\ 1 & \text{otherwise} \end{cases}$$

Using these definitions, the decay mask Λ is:

$$\Lambda_{i,j} = \frac{\lambda_{r_{\text{next}}(j,i)}}{\lambda_j} \tag{16}$$

As visible in Figure 2a, this creates higher-rank structure: each reset segment forms a rank-1 block, but multiple segments yield rank greater than 1 overall.

Chunk recurrent form For efficient computation, GateRank uses a chunked implementation as illustrated in Figure 2b. Using standard chunk notation, we denote $\mathbf{Q}_{[t]} := \boldsymbol{q}_{tC+1:(t+1)C+1}$ as the query block for chunk t with chunk size C, and $\boldsymbol{q}^r_{[t]} := \boldsymbol{q}_{tC+r}$ as the r-th query within chunk t. We use $\lambda^r_{[t]}$ to denote λ_{tC+r}, the cumulative product at position r within chunk t.

The chunk scan requires tracking which key-value pairs belong to completed reset segments (added to slow state) versus the ongoing final segment (added to fast state). Figure 2b illustrates this: the green column shows the single decay to the slow state, while the blue columns show the different decays for the fast state's ongoing reset segment.

We define $r_{[t]}$ as a flag: $r_{[t]} = 1$ if chunk t contains any position with $\log(\alpha_{tC+r}) \neq 0$, and $r_{[t]} = 0$ otherwise. This flag determines whether the fast state gets consolidated into the slow state at the chunk boundary.

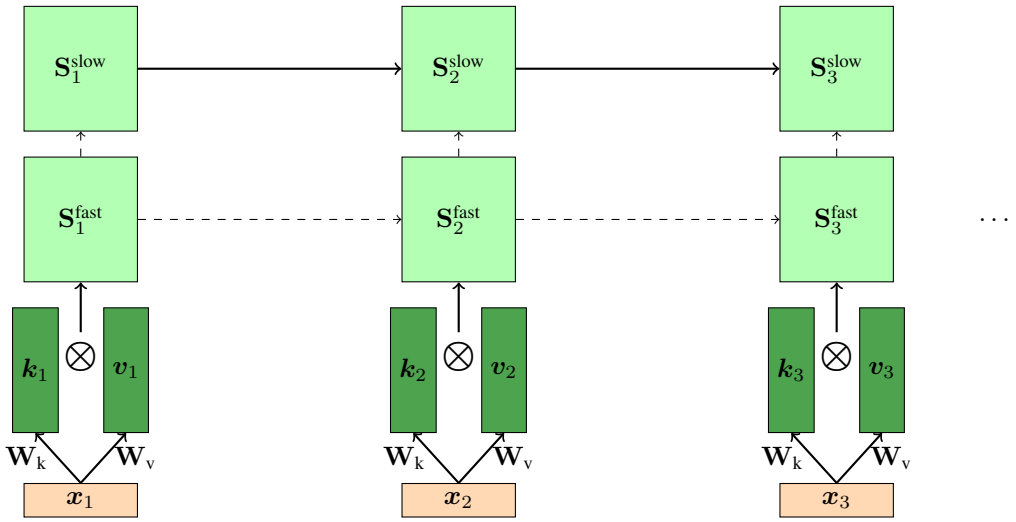


Figure 3. Recurrent mode for GateRank. Dashed arrows indicate a gating mechanism that does not necessarily propagate the value along this arrow at every timestep, depending on what $\log(\alpha)$ is obtained from the gated convolution of the input.

For decay computations, we need:

$$\boldsymbol{d}^r_{\text{last},[t]} = \frac{\lambda^{r_{\text{next}}(r,C)}_{[t]}}{\lambda^r_{[t]}} \quad \text{(decay to last position of chunk)}$$

$$\boldsymbol{d}^r_{\text{first},[t]} = \lambda^{r_{\text{next}}(0,r)}_{[t]} \quad \text{(decay from first position of chunk)}$$

The chunk-level recurrence becomes:

$$\begin{aligned}
\mathbf{S}^{\text{fast}}_{[t+1]} &= (1 - r_{[t]})\, \overrightarrow{\mathbf{S}}^{\text{fast}}_{[t]} + \mathbf{V}^\top_{[t]} \overrightarrow{\mathbf{K}}^{\text{fast}}_{[t]} \\
\mathbf{S}^{\text{slow}}_{[t+1]} &= \overrightarrow{\mathbf{S}}^{\text{slow}}_{[t]} + r_{[t]}\, \overrightarrow{\mathbf{S}}^{\text{fast}}_{[t]} + \mathbf{V}^\top_{[t]} \overrightarrow{\mathbf{K}}^{\text{slow}}_{[t]} \qquad (17) \\
\mathbf{O}_{[t]} &= \left(\overleftarrow{\mathbf{Q}}^{\text{fast}}_{[t]} \mathbf{S}^{\text{fast}\top}_{[t]} + \overleftarrow{\mathbf{Q}}^{\text{slow}}_{[t]} \mathbf{S}^{\text{slow}\top}_{[t]} \right) \\
&\quad + \left(\mathbf{Q}_{[t]} \mathbf{K}^\top_{[t]} \odot \Gamma_{[t]} \odot \Lambda_{[t]} \odot \mathbf{M} \right) \mathbf{V}_{[t]} \qquad (18)
\end{aligned}$$

The arrows denote decay-weighted quantities. The blue terms track the fast state (ongoing reset segment) while green terms track the slow state (completed reset segments). The asymmetry in the chunk boundary update arises because key-value pairs from completed reset segments accumulate in the slow state, while pairs from the final ongoing segment accumulate in the fast state.

We define the *last reset segment* in chunk t as all positions r such that no reset occurs between r and the chunk end C. Formally: $\{r \mid r \in [1, C], r_{\text{next}}(r, C) = C\}$. This determines which keys contribute to each state:

$$\begin{aligned}
\overleftarrow{\boldsymbol{q}}^{\text{fast},r}_{[t]} &= \gamma^r_{[t]} \boldsymbol{d}^r_{\text{first},[t]} \boldsymbol{q}^r_{[t]} \\
\overleftarrow{\boldsymbol{q}}^{\text{slow},r}_{[t]} &= \gamma^r_{[t]} \boldsymbol{q}^r_{[t]} \\
\overrightarrow{\boldsymbol{k}}^{\text{fast},r}_{[t]} &= \begin{cases} \frac{\gamma^C_{[t]}}{\gamma^r_{[t]}} \boldsymbol{d}^r_{\text{last},[t]} \boldsymbol{k}^r_{[t]}\,, & \text{if } r \text{ in last reset segment} \\ 0\,, & \text{otherwise} \end{cases} \\
\overrightarrow{\boldsymbol{k}}^{\text{slow},r}_{[t]} &= \begin{cases} \frac{\gamma^C_{[t]}}{\gamma^r_{[t]}} \boldsymbol{d}^r_{\text{last},[t]} \boldsymbol{k}^r_{[t]}\,, & \text{if } r \text{ not in last reset segment} \\ 0\,, & \text{otherwise} \end{cases} \\
\overrightarrow{\mathbf{S}}^{\text{fast}}_{[t]} &= \gamma^C_{[t]} \boldsymbol{d}^C_{\text{first},[t]} \mathbf{S}_{[t]} \\
\overrightarrow{\mathbf{S}}^{\text{slow}}_{[t]} &= \gamma^C_{[t]} \mathbf{S}_{[t]} \qquad (19)
\end{aligned}$$

See appendix A for GPU kernel implementation details that exploit the reset segment structure for efficient computation.

4. Experimental Setup

4.1. Model Architecture and Training Configuration

All models are trained at approximately 760M parameters for direct comparability. Each model uses the Mamba2 attention layer combined with MLP layers. The architecture has a model dimension of 1536, with Mamba2-based models using 20 layers and Gated DeltaNet-based models using 18 layers.

Training Details We train all models on 30B tokens from the FineWeb-Edu dataset (HuggingFace, 2024). The training uses an effective batch size of 120 with sequences of 4096 tokens. We employ the AdamW optimizer with a learning rate of 1.25e-3, cosine decay schedule, and weight

Table 1. Performance of models on language modeling and common sense reasoning tasks.

Model	Wiki. ppl ↓	LMB. ppl ↓	Avg. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	OBQA acc_n ↑	SCIQ acc_n ↑	BoolQ acc ↑	Avg. ↑
Mamba2	21.13	19.65	20.39	39.59	**71.00**	**51.17**	**56.83**	66.50	33.36	**39.20**	77.50	56.85	54.67
GateRank + Mamba2	20.80	19.98	20.39	39.12	70.29	50.57	54.70	67.30	33.96	36.80	81.20	**59.88**	54.87
Gated DeltaNet	**20.79**	17.53	19.16	40.71	70.62	50.91	55.41	67.76	**36.01**	36.60	81.60	56.12	55.08
GateRank + Gated DeltaNet	21.12	**16.29**	**18.71**	**43.06**	70.95	50.25	53.59	**68.10**	33.28	37.80	**82.80**	57.58	**55.27**

Table 2. Zero-shot performance comparison on S-NIAH benchmark suite

	S-NIAH-1			S-NIAH-2			S-NIAH-3			S-NIAH-MULTI-KEY			S-NIAH-MULTI-QUERY			S-NIAH-MULTI-VALUE			Average		
Model	1K	2K	4K	1K	2K	4K	1K	2K	4K	1K	2K	4K	1K	2K	4K	1K	2K	4K	1K	2K	4K
Mamba2	99.6	99.4	99.2	93	99	80.4	27.6	41.6	11.4	23.6	20.6	17.6	51.95	33.5	24	53.3	28.15	20.95	58.18	54.41	42.82
GateRank + Mamba2	**100**	99.8	90.8	**100**	99.2	52	75.6	47.8	21	27	22.6	22	44.4	32.6	18.8	51.65	36.35	22.21	66.44	56.39	37.8
Gated DeltaNet	**100**	**100**	**99.8**	**100**	99.8	**98.4**	50.2	28.4	17.8	38.6	30.6	27	50.4	39.25	24.8	58.5	48.85	29.6	66.28	57.82	49.57
GateRank + Gated DeltaNet	99.8	99.4	89.4	99.8	**100**	94	**91.2**	**78**	**44.2**	**44.2**	**31.4**	**28.6**	**71.55**	**49.05**	**29.25**	**69.6**	**49.15**	**29.7**	**79.36**	**67.83**	**52.53**

decay of 0.1. Gradient clipping is applied with a norm of 1.0, and we use a 1% warmup period. For parallel training across 8 GPUs, we utilize the Flame framework (Zhang & Yang, 2025).

4.2. Language Modeling and Common Sense Reasoning

We evaluate language modeling performance using perplexity on validation sets and assess common sense reasoning capabilities using tasks from the EleutherAI evaluation harness (Gao et al., 2024; Biderman et al., 2024). The evaluation includes WikiText (Merity et al., 2016), LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), and WinoGrande (Sakaguchi et al., 2020). Science question answering is evaluated through ARC-easy and ARC-challenge (Clark et al., 2018), OBQA (Mihaylov et al., 2018), and SCIQ (Welbl et al., 2017). Additionally, we include BoolQ (Clark et al., 2019) for reading comprehension with yes/no questions requiring complex reasoning.

4.3. Associative Recall Evaluation

To evaluate associative recall capabilities, we use the Structured Needle-In-A-Haystack (S-NIAH) benchmark suite from RULER (Hsieh et al., 2024). This suite includes six task variants: S-NIAH-1, S-NIAH-2, and S-NIAH-3 for single-needle retrieval with increasing difficulty; S-NIAH-MULTI-KEY for retrieving multiple keys; S-NIAH-MULTI-QUERY for multiple queries about the same information; and S-NIAH-MULTI-VALUE for multiple values associated with keys. We evaluate models at three context lengths: 1K, 2K, and 4K tokens. These tasks test the model's ability to retrieve specific information associated with keys embedded within longer contexts.

4.4. Circuit Complexity Evaluation: S5 Permutation Composition

To assess the computational complexity that can be represented in a single attention layer, we use the S5 permutation composition task. This task requires computing the product of sequences of permutations on the symmetric group S5, which can only be solved by circuits in the NC^1 complexity class (circuits with logarithmic depth) (Merrill et al., 2024; Barrington, 1989). The task is NC^1-complete based on Barrington's Theorem, which establishes that permutation composition for non-solvable groups requires logarithmic-depth circuits.

Training Configuration We train a single attention layer on S5 sequences of length 16. The training uses the AdamW optimizer with a learning rate of 0.01 with linear decay to 0 and weight decay of 0.01. We train for 4000 steps with a batch size of 128, using a 128-step warmup period and bfloat16 mixed precision. For validation, we evaluate on sequences longer than the training length to assess extrapolation capabilities.

4.5. Throughput Evaluation

We benchmark the efficiency of our Triton-based chunk scan implementation by measuring the time per training step across different sequence lengths. The benchmark compares GateRank against the Mamba2 baseline (linear attention), the Gated DeltaNet baseline (linear attention with gating), and FlashAttention-v2 (Dao, 2023) (softmax attention, Triton implementation). Measurements are taken for sequence lengths ranging from short contexts to 32K+ tokens to characterize the crossover point where linear attention becomes more efficient than softmax attention.

5. Results and Discussion

5.1. Language Modeling and Common Sense Reasoning

Table 1 presents the performance of GateRank-enhanced models compared to their baseline counterparts on language modeling and common sense reasoning tasks. GateRank maintains or improves performance on these benchmarks, with the magnitude and pattern of changes differing between the two base architectures.

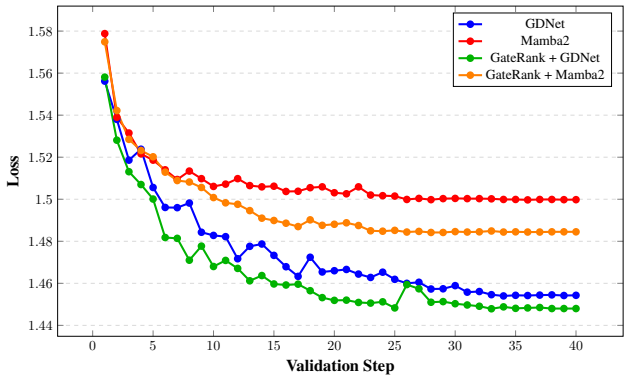


Figure 4. Validation losses for S5 state exchange on length 64 sequences. Models are trained on the same, randomly generated sequences of length 16.

Language Modeling Performance GateRank modestly improves language modeling performance across both base architectures. For Mamba2, WikiText perplexity improves from 21.13 to 20.80, while LAMBADA performance remains comparable. For Gated DeltaNet, the model achieves the best overall perplexity average of 18.71, with improvements on both WikiText and LAMBADA metrics. While these changes are not necessarily significant, the modest improvements suggest that GateRank can at least match the baseline models in language modeling if not slightly improve them.

Common Sense Reasoning GateRank improves average common sense reasoning performance for both architectures, though individual benchmark results vary. GateRank + Mamba2 achieves an average score of 54.87 versus 54.67 for baseline, with notable gains on BoolQ (59.88 vs. 56.85), SCIQ (81.20 vs. 77.50), and ARC-challenge (33.96 vs. 33.36). GateRank + Gated DeltaNet achieves an average of 55.27 versus 55.08 for baseline, with improvements on LAMBADA accuracy (43.06 vs. 40.71) and SCIQ (82.80 vs. 81.20). While some individual benchmarks show small decreases, the overall pattern demonstrates that the dual-state architecture maintains competitive performance across diverse reasoning tasks with modest improvements in aggregate scores.

5.2. Associative Recall with Needle-In-A-Haystack

The S-NIAH benchmark results in Table 2 provide compelling evidence for GateRank's enhanced associative recall capabilities, revealing a clear pattern: benefits scale with task complexity.

Simple Retrieval Tasks On the simplest retrieval tasks (S-NIAH-1 and S-NIAH-2), all models achieve near-perfect performance at short contexts, with GateRank maintaining competitive accuracy. At 1K tokens, GateRank + Mamba2 achieves perfect accuracy on both tasks, demonstrating that the dual-state architecture does not compromise basic retrieval capabilities. However, at longer contexts (4K tokens), the advantage becomes less consistent, with baseline models occasionally matching or exceeding GateRank performance on these simpler tasks. This suggests that for straightforward key-value retrieval, standard linear attention's rank-1 structure is often sufficient.

Complex Retrieval Tasks The most significant improvements emerge on harder retrieval variants requiring simultaneous tracking of multiple pieces of information. On S-NIAH-3, GateRank + Gated DeltaNet demonstrates substantial gains across all context lengths, nearly doubling baseline performance at 1K tokens and achieving even larger relative improvements at 2K and 4K tokens. Similar patterns appear across MULTI-KEY, MULTI-QUERY, and MULTI-VALUE tasks, where GateRank consistently outperforms baselines by significant margins.

The overall average scores reveal the cumulative benefit: GateRank + Gated DeltaNet achieves 79.36% at 1K tokens compared to 66.28% for the baseline, with advantages maintained across longer contexts (67.83% vs. 57.82% at 2K, 52.53% vs. 49.57% at 4K). These improvements are particularly notable given that all models face increasing difficulty as context length grows, yet GateRank maintains its advantage throughout.

Interpretation These results support the hypothesis that breaking the rank-1 decay structure of linear attention enables more sophisticated memory management. The dual-state architecture allows GateRank to maintain both recently accessed information (in the fast state) and selectively preserved long-term associations (in the slow state). This is particularly beneficial when multiple pieces of information must be simultaneously tracked and retrieved based on different keys, as required by the harder NIAH variants.

The contrasting performance on simple versus complex tasks suggests that GateRank's benefits scale with task complexity. Simple key-value retrieval can be handled adequately by standard linear attention's rank-1 structure, but complex retrieval may require the additional representational capacity that GateRank provides. This finding has important implications for long-context language modeling, where models must frequently perform complex information retrieval from extended documents or conversations.

5.3. Circuit Complexity: S5 Permutation Composition

Figure 4 presents validation losses for single-layer models trained on the S5 permutation composition task. This task requires computing sequential products of permutations on

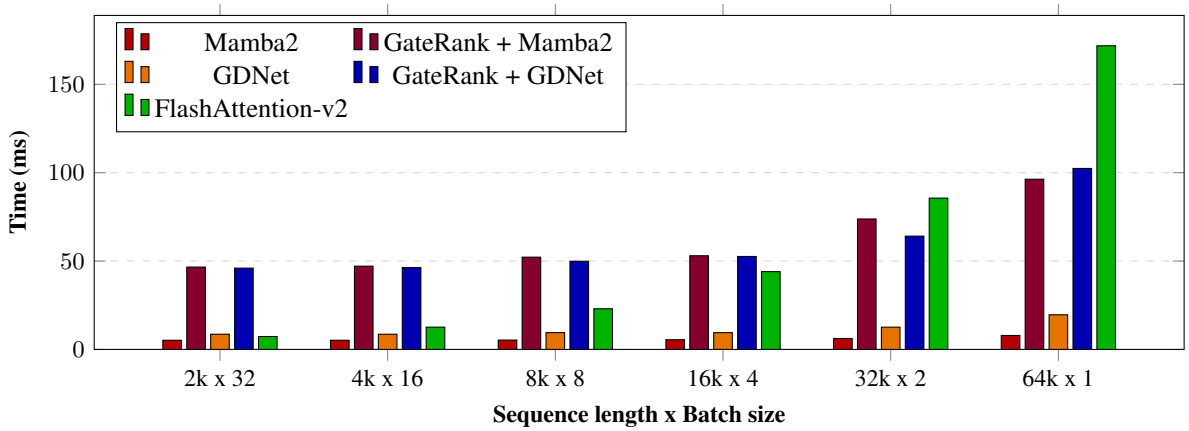


Figure 5. Benchmark results showing time for a training step (we use the Triton implementation of FlashAttention-v2).

the symmetric group S5, which is NC^1-complete based on Barrington's Theorem (Barrington, 1989). The NC^1 complexity class contains problems solvable by circuits with logarithmic depth, representing a fundamental barrier for constant-depth architectures.

Performance on Training Length None of the models successfully solved the task at the training length of 16 positions, as evidenced by validation losses remaining well above zero. This aligns with theoretical predictions from recent work showing that linear attention mechanisms and even standard transformers are fundamentally limited by TC^0 complexity bounds (Merrill et al., 2024). The S5 permutation composition problem requires tracking state transformations through sequential operations, which constant-depth circuits cannot efficiently represent.

Extrapolation to Longer Sequences The key finding appears in the extrapolation behavior when evaluating on sequences longer than the training length of 16. GateRank models maintain lower validation losses on longer sequences compared to baseline models, demonstrating superior length generalization. This improved extrapolation is significant because it suggests that GateRank may learn more compositional representations of the permutation operations rather than merely memorizing patterns specific to length-16 sequences.

Implications for Computational Power While we cannot definitively conclude that GateRank operates in a higher complexity class than baseline linear attention (as no model solved the task perfectly), the improved performance suggests enhanced computational capacity. The dual-state architecture with separate fast and slow memory streams may provide additional computational pathways that better approximate logarithmic-depth computations. This is theoretically plausible because GateRank's conditional state updates create data-dependent computation graphs that can vary based on reset patterns, potentially enabling more flexible information flow than fixed rank-1 decay structures.

The practical implication is that GateRank may be better suited for tasks requiring iterative refinement or multi-step reasoning, even if it cannot fundamentally escape the TC^0 complexity bounds of attention-based architectures. This enhanced approximation capacity could prove valuable for long-context reasoning tasks in language modeling, where models must perform quasi-symbolic operations over extended contexts.

5.4. Throughput and Efficiency

Figure 5 presents throughput measurements comparing GateRank's chunk scan implementation against baseline linear attention models and FlashAttention-v2. The results reveal both the promise and current limitations of the GateRank implementation.

Comparison to Softmax Attention GateRank achieves a critical milestone by surpassing FlashAttention-v2 throughput at sequence lengths beyond 32K tokens. This crossover point is significant because it demonstrates that GateRank maintains the fundamental efficiency advantage of linear attention for long sequences. As sequence length increases, GateRank's O(n) complexity provides increasingly substantial benefits over FlashAttention-v2's $O(n^2)$ complexity, making it viable for applications requiring very long context processing.

Overhead Compared to Linear Attention Baselines However, GateRank introduces considerable overhead compared to vanilla Mamba2 and Gated DeltaNet implementations. This performance gap stems from two primary factors. First, GateRank must materialize both fast and slow states at chunk boundaries, effectively doubling the state size that

must be managed during the chunk scan. This increases memory bandwidth requirements and creates a larger memory bottleneck. Second, the conditional state updates based on reset patterns require additional `gather` operations that lack the regular memory access patterns of structured linear attention, leading to less efficient memory coalescing on modern GPUs.

Implementation Optimization Potential The current Triton implementation represents an initial proof-of-concept rather than a fully optimized kernel. Several optimization opportunities remain unexplored. First, a more IO-aware implementation could reduce redundant memory transfers by carefully scheduling when fast and slow states are materialized and consumed. Second, the `gather` operations could potentially be replaced with more structured operations by exploiting patterns in typical reset sequences. Third, kernel fusion opportunities may exist where multiple operations on the state can be combined to reduce memory round-trips.

Despite current limitations, the fact that GateRank achieves competitive throughput with FlashAttention at long sequence lengths while providing superior associative recall validates the approach's practical viability. Future work on kernel optimization should focus on closing the gap with baseline linear attention while preserving GateRank's enhanced memory management capabilities.

5.5. Overall Assessment

The experimental results demonstrate that GateRank successfully addresses key limitations of linear attention models through its learnable rank structure with fast and slow memory states. The dual-state architecture provides measurable benefits across multiple dimensions: enhanced complex associative recall (S-NIAH-3 and multi-* variants), better approximation of sequential computation (S5), and maintained efficiency for long sequences (throughput beyond 32K tokens). The trade-off is equally clear: modest overhead compared to baseline linear attention.

Overall, the results suggest that GateRank is particularly well-suited for applications requiring long-context reasoning and complex information retrieval.

6. Conclusion

This paper introduces GateRank, a novel extension to linear attention that overcomes fundamental limitations of rank-1 semi-separable decay structures through learnable memory segmentation inspired by multi-timescale synaptic plasticity in biological neural systems. By maintaining separate fast and slow states with conditional reset mechanisms, GateRank provides linear attention models with fine-grained memory management while preserving computational efficiency. Our theoretical contribution breaks the rank-1 constraint of standard linear attention by introducing a dual-state architecture with learnable reset patterns that determine when information consolidates from fast transient memory into slow persistent memory. This maintains linear-time complexity while enabling higher-rank decay structures for more sophisticated associative recall.

Furthermore, we show the general applicability of GateRank to linear attention by implementing it on top of both Mamba2 and Gated DeltaNet. Empirically, our 760M parameter GateRank model shows modest improvements on language modeling and more meaningful gains on complex associative recall tasks requiring simultaneous tracking of multiple information streams. Circuit complexity experiments reveal enhanced capacity for approximating sequential computations with superior extrapolation to longer sequences. Critically, these improvements maintain linear scaling with sequence length, surpassing FlashAttention-v2 throughput beyond 32K tokens.

The learnable reset patterns provide a principled mechanism for dynamically adjusting memory granularity based on input characteristics, discovering optimal segmentation patterns during training rather than imposing a fixed strategy. GateRank demonstrates that the rank-1 constraint is not fundamental to linear complexity, but a design choice that can be relaxed through architectural innovation guided by neuroscience principles, opening new possibilities for efficient long-context processing with sophisticated memory management.

Acknowledgment

This work was sponsored by the Federal Ministry of Education, Germany BMBF under grants no. 16ME0398K, 16ME0399, 01IS22094E; and Neurosys as part of the initiative "Cluster4Future" funded by the Federal Ministry of Education and Research BMBF (03ZU1106CB)

References

Arora, S., Eyuboglu, S., Timalsina, A., Johnson, I., Poli, M., Zou, J., Rudra, A., and Ré, C. Zoology: Measuring and improving recall in efficient language models. *arXiv:2312.04927*, 2023.

Arora, S., Eyuboglu, S., Zhang, M., Timalsina, A., Alberti, S., Zinsley, D., Zou, J., Rudra, A., and Ré, C. Simple linear attention language models balance the recall-throughput tradeoff, 2025. URL https://arxiv.org/abs/2402.18668.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. In

Advances in Neural Information Processing Systems, volume 29, pp. 4628–4636. Curran Associates, Inc., 2016.

Barrington, D. A. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

Biderman, S., Schoelkopf, H., Sutawika, L., Gao, L., Tow, J., Abbasi, B., et al. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*, 2024.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Chomsky, N. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956. doi: 10.1109/TIT.1956.1056813.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L., and Weller, A. Rethinking attention with performers, 2022. URL https://arxiv.org/abs/2009.14794.

Cirone, N. M. and Salvi, C. Parallelflow: Parallelizing linear transformers via flow discretization, 2025. URL https://arxiv.org/abs/2504.00492.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL https://arxiv.org/abs/2307.08691.

Dao, T. and Gu, A. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024. URL https://arxiv.org/abs/2405.21060.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Ellwood, I. T. Short-term hebbian learning can implement transformer-like attention. *PLOS Computational Biology*, 20(1):e1011843, 2024. doi: 10.1371/journal.pcbi.1011843.

Finkbeiner, J. and Neftci, E. On-chip learning via transformer in-context learning. In *2025 IEEE 7th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 1–5, 2025. doi: 10.1109/AICAS64808.2025.11173092.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Grazzi, R., Siems, J., Zela, A., Franke, J. K. H., Hutter, F., and Pontil, M. Unlocking state-tracking in linear rnns through negative eigenvalues, 2025. URL https://arxiv.org/abs/2411.12537.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces, 2022. URL https://arxiv.org/abs/2111.00396.

Guo, H., Yang, S., Goel, T., Xing, E. P., Dao, T., and Kim, Y. Log-linear attention, 2025. URL https://arxiv.org/abs/2506.04761.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. Ruler: What's the real

context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

HuggingFace. Fineweb-edu: A high-quality educational dataset. https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu, 2024. Accessed: 2024.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL https://arxiv.org/abs/2006.16236.

Leroux, N., Finkbeiner, J., and Neftci, E. O. Neuromorphic principles in self-attention hardware for efficient transformers. *Nature Computational Science*, 5:708–710, 2025. doi: 10.1038/s43588-025-00868-9.

Liu, B., Wang, R., Wu, L., Feng, Y., Stone, P., and Liu, Q. Longhorn: State space models are amortized online learners, 2024. URL https://arxiv.org/abs/2407.14207.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Merrill, W. and Sabharwal, A. The parallelism tradeoff: Limitations of log-precision transformers, 2023. URL https://arxiv.org/abs/2207.00729.

Merrill, W., Petty, J., and Sabharwal, A. The illusion of state in state-space models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 35492–35506. PMLR, 2024.

Merrill, W., Petty, J., and Sabharwal, A. The illusion of state in state-space models, 2025. URL https://arxiv.org/abs/2404.08819.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, oct - nov 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. Resurrecting recurrent neural networks for long sequences. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 26670–26698. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/orvieto23a.html.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N. Q., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Erk, K. and Smith, N. A. (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/.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K., He, X., Hou, H., Lin, J., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Mantri, K. S. I., Mom, F., Saito, A., Song, G., Tang, X., Wang, B., Wind, J. S., Wozniak, S., Zhang, R., Zhang, Z., Zhao, Q., Zhou, P., Zhou, Q., Zhu, J., and Zhu, R.-J. Rwkv: Reinventing rnns for the transformer era, 2023. URL https://arxiv.org/abs/2305.13048.

Peng, B., Goldstein, D., Anthony, Q., Albalak, A., Alcaide, E., Biderman, S., Cheah, E., Du, X., Ferdinan, T., Hou, H., Kazienko, P., GV, K. K., Kocoń, J., Koptyra, B., Krishna, S., Jr., R. M., Lin, J., Muennighoff, N., Obeid, F., Saito, A., Song, G., Tu, H., Wirawan, C., Woźniak, S., Zhang, R., Zhao, B., Zhao, Q., Zhou, P., Zhu, J., and Zhu, R.-J. Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence, 2024. URL https://arxiv.org/abs/2404.05892.

Peng, B., Zhang, R., Goldstein, D., Alcaide, E., Du, X., Hou, H., Lin, J., Liu, J., Lu, J., Merrill, W., Song, G., Tan, K., Utpala, S., Wilce, N., Wind, J. S., Wu, T., Wuttke, D., and Zhou-Zheng, C. Rwkv-7 "goose" with expressive dynamic state evolution, 2025. URL https://arxiv.org/abs/2503.14456.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399. URL https://ojs.aaai.org/index.php/AAAI/article/view/6399.

Schlag, I., Irie, K., and Schmidhuber, J. Linear transformers are secretly fast weight programmers, 2021. URL https://arxiv.org/abs/2102.11174.

Siems, J., Carstensen, T., Zela, A., Hutter, F., Pontil, M., and Grazzi, R. Deltaproduct: Improving state-tracking in linear rnns via householder products, 2025. URL https://arxiv.org/abs/2502.10297.

Smith, J. T. H., Warrington, A., and Linderman, S. W. Simplified state space layers for sequence modeling, 2023. URL https://arxiv.org/abs/2208.04933.

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. Retentive network: A successor to transformer for large language models, 2023. URL https://arxiv.org/abs/2307.08621.

Terzić, A., Menet, N., Hersche, M., Hofmann, T., and Rahimi, A. Structured sparse transition matrices to enable state tracking in state-space models, 2025. URL https://arxiv.org/abs/2509.22284.

Tetzlaff, C., Kolodziejski, C., Markelic, I., and Wörgötter, F. Time scales of memory, learning, and plasticity. *Biological Cybernetics*, 106(11):715–726, 2012. doi: 10.1007/s00422-012-0529-z.

Tillet, P., Kung, H. T., and Cox, D. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, pp. 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Vermani, A., Nassar, J., Jeon, H., Dowling, M., and Park, I. M. Meta-dynamical state space models for integrative neural data analysis. In *Proceedings of the International Conference on Learning Representations (ICLR)*. OpenReview, 2025.

Welbl, J., Liu, N. F., and Gardner, M. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, Copenhagen, Denmark, sep 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Yang, S., Wang, B., Shen, Y., Panda, R., and Kim, Y. Gated linear attention transformers with hardware-efficient training, 2024. URL https://arxiv.org/abs/2312.06635.

Yang, S., Kautz, J., and Hatamizadeh, A. Gated delta networks: Improving mamba2 with delta rule, 2025a. URL https://arxiv.org/abs/2412.06464.

Yang, S., Wang, B., Zhang, Y., Shen, Y., and Kim, Y. Parallelizing linear transformers with the delta rule over sequence length, 2025b. URL https://arxiv.org/abs/2406.06484.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. HellaSwag: Can a machine really finish your sentence? In *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/.

Zenke, F., Agnes, E. J., and Gerstner, W. Hebbian plasticity requires compensatory processes on multiple timescales. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 372(1715):20160259, 2017. doi: 10.1098/rstb.2016.0259.

Zhang, M., Bhatia, K., Kumbong, H., and Ré, C. The hedgehog the porcupine: Expressive linear attentions with softmax mimicry, 2024. URL https://arxiv.org/abs/2402.04347.

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

Zhong, S., Xu, M., Ao, T., and Shi, G. Understanding transformer from the perspective of associative memory, 2025. URL https://arxiv.org/abs/2505.19488.

A. Implementation details

Λ is computed as the exponential of the output of the function `reverse_reset_cumsum` in listing 2.

Decaying to the last position for the keys requires reversing the cumulative sum in the log-space by taking the exponential of last row of the matrix computed in `reverse_reset_cumsum` from listing 2 which we denote as $\boldsymbol{d}_{\text{last}}$.

The log of extra fast state multiplier for decaying to the first position in a chunk is computed in the function `same_segment_bwd_mult` in listing 4, the log space fast gate being $\log(\lambda)$, which we denote as $\boldsymbol{d}_{\text{first}}$.

Lastly, we need the function `chunk_contains_reset` from listing 3 to model the conditional updates of both states at the chunk boundaries.

```
def reset_idcs_cm(log_alpha):
    """
    for masking operations in index space
    to avoid floating-point comparisons
    """
    reset_idcs = where(
        log_alpha != 0,
        arange(log_alpha.size(0)),
        0
    )
    reset_idcs_cm = cummax(reset_idcs)
```

Listing 1. PyTorch style reset index cummax computation which is used for masking to avoid imprecise floating-point comparisons

```
def reset_cumsum(log_beta, log_alpha):
    """
    computes the cumsum of log_beta
    with resets after log_alpha != 0
    """
    log_beta_cs = cumsum(log_beta)

    # use cummin because gates are
    # always negative in log-space
    lmbda =
        log_beta_cs - \
        cummin(
            where(
                left_pad(log_alpha)[:-1] != 0,
                left_pad(log_beta_cs)[:-1],
                0
            ),
            dim=-1
        )
    return lmbda

def reverse_reset_cumsum(
    log_alpha, lmbda,
    reset_idcs_cm,
):
    """
    computes the mask matrix which is used
    as Lambda in the parallel form
    log_alpha: (chunk_size,)
    lmbda: (chunk_size,)
    returns: (chunk_size,chunk_size)
    """
    last_reset_seg_2d_mask =
        lower_diag_mask & \
        (F.pad(reset_idcs_cm, (1,0))[:-1] == reset_idcs_cm)

    # perform inverse in log-space with
    # "outer subtraction"
```

```
    # for each reset segment
    # don't subtract at reset locations
    # since local cumsums from previous
    # segment go up to non-zero log_alpha
    # indicating start of next segment
    seg_inv_2d =
        lmbda[:,None] - \
        where(
            log_alpha[None,:] == 0,
            lmbda[None,:],
            0
        )
    seg_inv_2d = seq_inv_2d.masked_fill(
        ~last_reset_seg_2d_mask, 0)

    # each row is a sum of previous rows
    # with reset signals in log_alpha
    # (seg_inv_2d contains full reversed
    # cumsum of each past segment at rows
    # where log_alpha != 0)
    multiplier = (log_alpha[None,:] != 0) \
                    & lower_diag_mask

    return (identity + multiplier) \
            @ seg_inv_2d
```

Listing 2. PyTorch style reset cumsums and reverse computation with matrix multiplication (see listing **??** for computation of `reset_idcs_cm`)

```
def chunk_contains_reset(log_alpha):
    return log_alpha.sum() != 0
```

Listing 3. Check if there are resets in a chunk given the slow gate vector for the chunk note that a non-zero slow gate indicates a reset in the fast state.

```
def same_segment_bwd_mult(
    log_alpha, lmbda, reset_idcs_cm
):
    masked_offs = where(
        left_pad(reset_idcs_cm)[:-1] & \
            log_alpha[0] == 0,
        arange(log_alpha.size(0)),
        0
    )
    masked_offs = cummax(masked_offs)
    return gather(lmbda, masked_offs)
```

Listing 4. PyTorch style code for the inter-chunk decay. It uses the `reset_idcs_cm` as computed in listing 1