Unlocking State-Tracking in Linear RNNs through Negative Eigenvalues

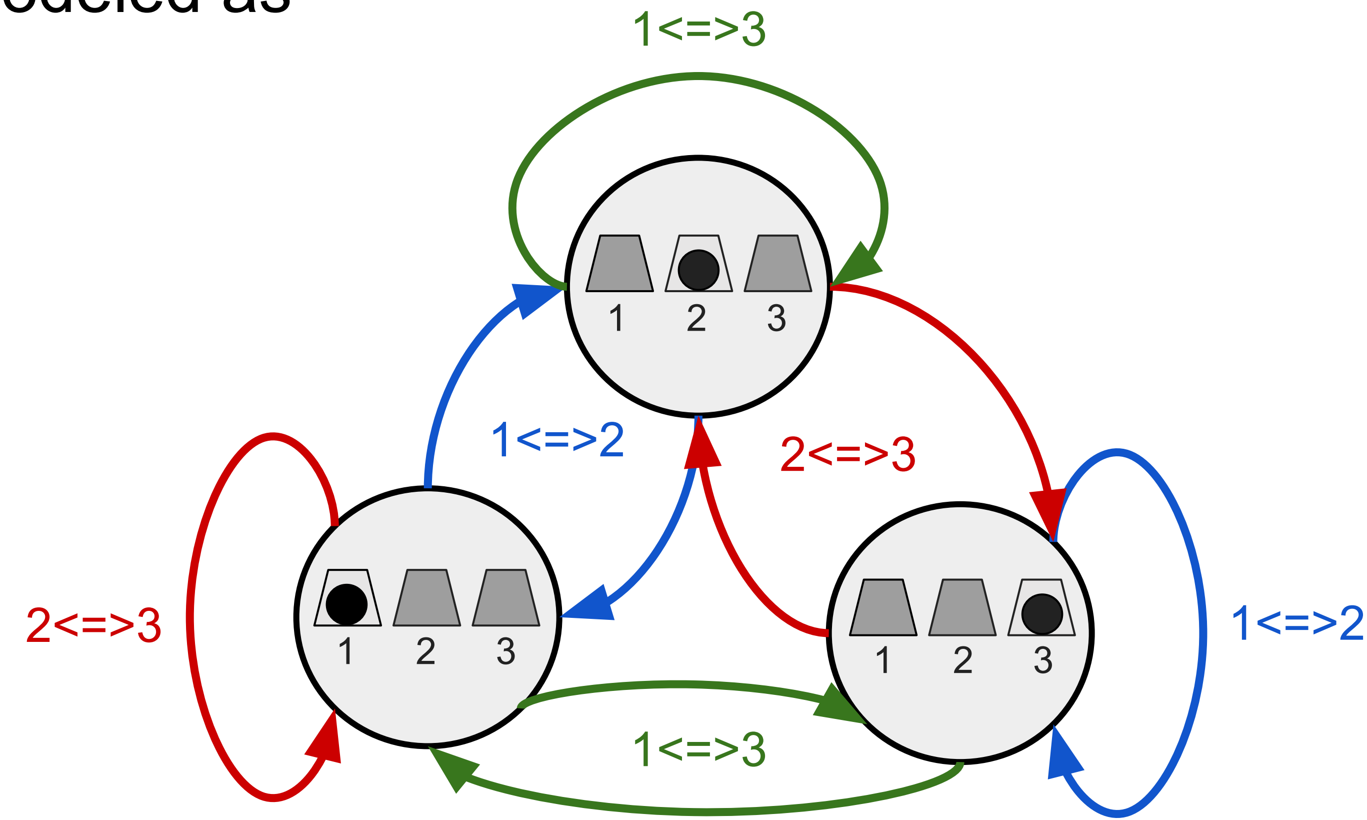
Riccardo Grazzi*, Julien Siems*, Arbër Zela,
Jörg KH Franke, Frank Hutter, Massimiliano Pontil
(*equal contribution)

State Tracking

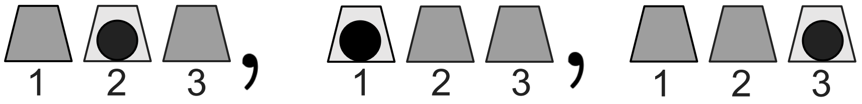
Initial State | State Transitions | Where is the ball?

- **State is not observable**: the ball position is shown only at the start
- The cat needs to **watch the entire sequence of transitions**

Modeled as



Finite State Automata (FSA)

States (Finite set) → $Q = \{$  $\}$

Alphabet (Finite set) → $\Sigma = \{ 1{<=>}2,\ 1{<=>}3,\ 2{<=>}3 \}$

Initial state

$$q_0 \in Q$$

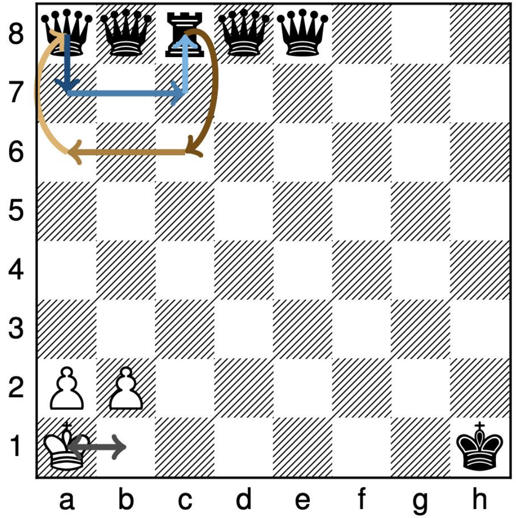
$$\delta : Q \times \Sigma \to Q$$

Transition function

State Tracking = mimic an FSA:

map the **sequences of transitions** (input) to **sequences of states** (output).

State Tracking Tasks in Text Data



Tracking a chessboard with non-standard (source, target) notation for moves

(A8, A7), (A1, B1), (C8, C6), (B1, A1), (A7, C7), (A1, B1), (C6, A6), (B1, A1), (C7, C8), (A1, B1), (A6, A8)

Input to the model

Code evaluation

```
x = [0, 0, 1, 0, 0]
x[1], x[3] = x[3], x[1] # Swap 1, 3
```

Entity Tracking

Alice, Bob and Carl each have a coin. Carl is the only one having a penny. Alice and Carl trade coins.

Images modified from Merrill, William, Jackson Petty, and Ashish Sabharwal. "The illusion of state in state-space models." *ICML* (2024).

Linear RNNs (One Layer)

State matrix — input token — Output — Channel mix (MLP)

$$\boldsymbol{H}_i = \underbrace{\boldsymbol{A}(\boldsymbol{x}_i)}_{\text{State-transition matrix}} \boldsymbol{H}_{i-1} + \boldsymbol{B}(\boldsymbol{x}_i), \quad \hat{\boldsymbol{y}}_i = \mathrm{dec}(\boldsymbol{H}_i, \boldsymbol{x}_i)$$

State-transition matrix

	$\boldsymbol{A}(\boldsymbol{x}_t)$	$\boldsymbol{B}(\boldsymbol{x}_t)$
Mamba	$\mathrm{Diag}\left(\exp\left(-\boldsymbol{\Delta}_t \odot \exp(\boldsymbol{w}_{1,i})\right)\right)$	$k_{t,i} \boldsymbol{\Delta}_t \odot \boldsymbol{x}_t$
GLA	$\mathrm{Diag}(\boldsymbol{\alpha}_t)$	$\boldsymbol{k}_t \boldsymbol{v}_t^\top$
DeltaNet	$\boldsymbol{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top$ GH, non-diagonal	$\beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top$

Gu, Albert, and Tri Dao. "Mamba: Linear-time sequence modeling with selective state spaces." *arXiv* (2023).

Yang, Songlin, et al. "Gated Linear Attention Transformers with Hardware-Efficient Training." *ICML 2024*

Yang, Songlin, et al. "Parallelizing Linear Transformers with the Delta Rule over Sequence Length.", NeurIPS 2024

Linearity + heavily structured matrices make the recurrence efficiently parallelizable

Linear RNNs (One Layer)

State matrix — input token — Output — Channel mix (MLP)

$$\boldsymbol{H}_i = \boldsymbol{A}(\boldsymbol{x}_i)\boldsymbol{H}_{i-1} + \boldsymbol{B}(\boldsymbol{x}_i), \quad \hat{\boldsymbol{y}}_i = \mathrm{dec}(\boldsymbol{H}_i, \boldsymbol{x}_i)$$

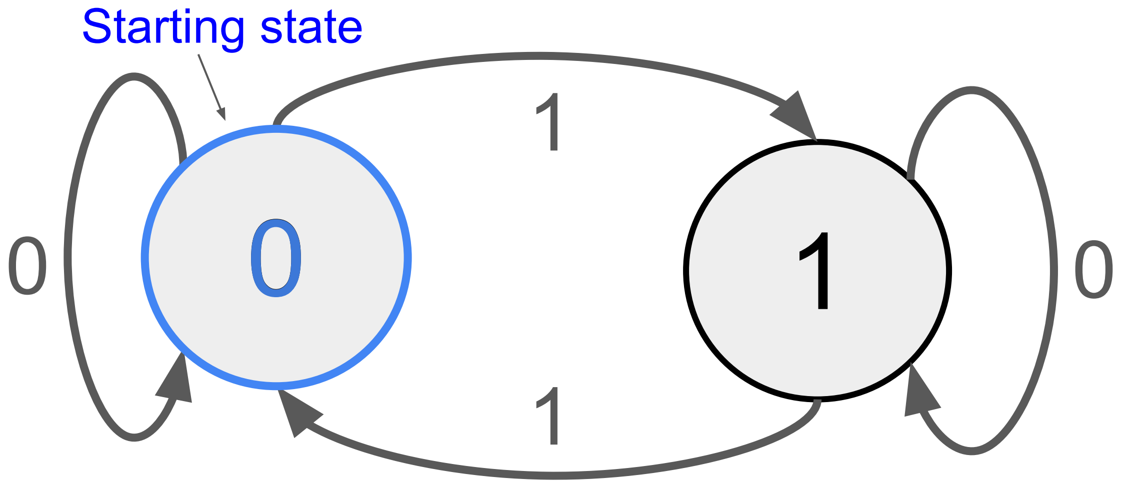
State-transition matrix

	$\boldsymbol{A}(\boldsymbol{x}_t)$	$\boldsymbol{B}(\boldsymbol{x}_t)$
Mamba	$\mathrm{Diag}(\exp(-\boldsymbol{\Delta}_t \odot \exp(\boldsymbol{w}_{1,i})))$	$k_{t,i}\boldsymbol{\Delta}_t \odot \boldsymbol{x}_t$
GLA	$\mathrm{Diag}(\boldsymbol{\alpha}_t)$	$\boldsymbol{k}_t\boldsymbol{v}_t^\top$
DeltaNet	$\boldsymbol{I} - \beta_t\boldsymbol{k}_t\boldsymbol{k}_t^\top$	$\beta_t\boldsymbol{k}_t\boldsymbol{v}_t^\top$

Transformers are Linear RNNs with infinite dimensional state and $\boldsymbol{A}(\boldsymbol{x}_t) = \boldsymbol{I}$

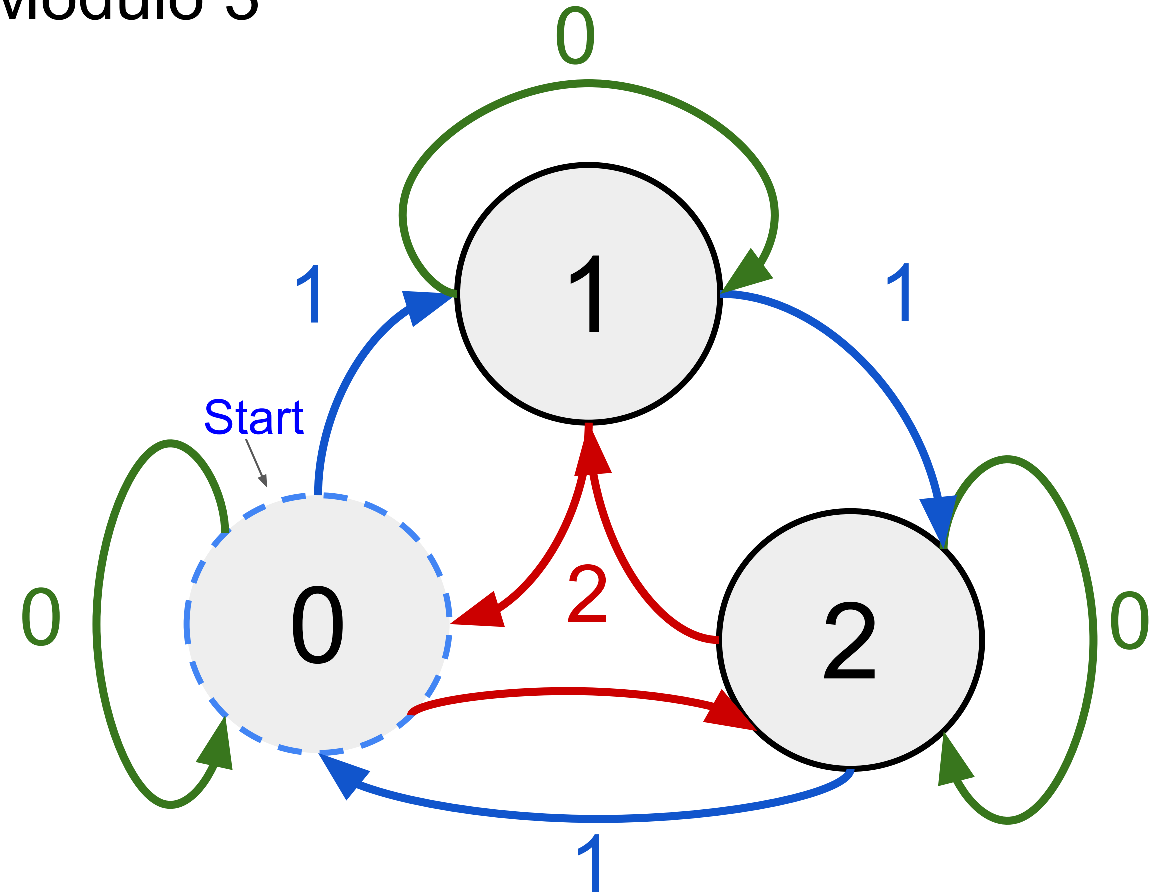
Katharopoulos, Angelos, et al. "Transformers are rnns: Fast autoregressive transformers with linear attention." ICML 2020.

Parity (2-cups game, addition modulo 2)



Input bits (transitions)	1	1	0	0	1	0	. . .
Parity (states)	1	0	0	0	1	1	. . .

Addition Modulo 3



Solving Parity with a Scalar Linear RNN

$$h_i = a(x_i)h_{i-1} + x_i$$

Solution 1: State = sum of previous values

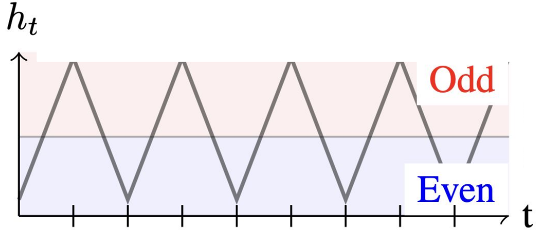
$$a(x_i) = 1$$

$$h_t = \sum_{i=1}^{t} x_i$$

$$y_t = h_t \bmod 2 \quad \textbf{(state blows up!)}$$

Solution 2: State = parity

$$a(1) = -1, \quad a(0) = 1 \quad y_t = h_t \qquad \textbf{(negative values)}$$



Issue with Linear RNNs

State-transition matrix $\longrightarrow \boldsymbol{A}(\boldsymbol{x}_t)$		$\boldsymbol{B}(\boldsymbol{x}_t)$
Mamba	$\mathrm{Diag}\left(\exp\left(-\boldsymbol{\Delta}_t \odot \exp(\boldsymbol{w}_{1,i})\right)\right)$	$k_{t,i}\boldsymbol{\Delta}_t \odot \boldsymbol{x}_t$
GLA	$\mathrm{Diag}\left(\boldsymbol{\alpha}_t\right)$	$\boldsymbol{k}_t\boldsymbol{v}_t^\top$
DeltaNet	$\boldsymbol{I} - \beta_t\boldsymbol{k}_t\boldsymbol{k}_t^\top$	$\beta_t\boldsymbol{k}_t\boldsymbol{v}_t^\top$

$$\Delta_{t,i} \geq 0, \quad \alpha_{t,i} \geq 0, \quad \beta_t \in (0,1), \boldsymbol{k}_t \in \mathbb{R}^n, \|\boldsymbol{k}_t\| = 1$$

All state-transition matrices have **positive eigenvalues** in [0,1].

diagonal Linear RNN with positive values *cannot* solve parity in finite precision (Sarrof et al. 2024)

Yash Sarrof, Yana Veitsman, and Michael Hahn. The Expressive Capacity of State Space Models: A Formal Language Perspective. NeurIPS 2024.

LLMs Struggle to Track States

Transformers and diagonal linear RNNs cannot track states in limited precision and for arbitrary input lengths (Hahn 2020, Merrill et al. 2023, 2024, Sarrof et al. 2024).

In contrast, RNNs and linear RNNs with **full state transition matrices** can track states with only one layer, but cannot be parallelized efficiently.

What about **scalable non-diagonal Linear RNNs** like DeltaNet?

Hahn, Michael. "Theoretical limitations of self-attention in neural sequence models." *Transactions of the Association for Computational Linguistics* 8 (2020): 156-171.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. Transactions of the Association for Computational Linguistics, 11:531–545, 2023.

William Merrill, Jackson Petty, and Ashish Sabharwal. The Illusion of State in State-Space Models. ICML 2024.

Yash Sarrof, Yana Veitsman, and Michael Hahn. The Expressive Capacity of State Space Models: A Formal Language Perspective. NeurIPS 2024.

Contribution: Limits of Linear RNNs in Finite Precision

Thm. 1 (Parity): Finite precision linear RNNs cannot solve parity at arbitrary input lengths if for all layers

$$\lambda \in \mathbb{R}, \lambda \geq 0 \quad \forall \lambda \in \operatorname{eigs}(\boldsymbol{A}(\boldsymbol{x})) \quad \forall \boldsymbol{x}$$

Thm. 2 (Modular Counting): Finite precision linear RNNs with L layers cannot count modulo m, with m not a power of two, if for every $i \in \{1, \ldots, L\}$ the i-th layer satisfies

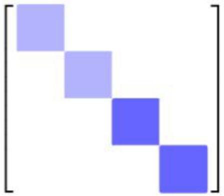
$$\lambda \in \mathbb{R} \quad \forall \lambda \in \operatorname{eigs}(\boldsymbol{A}(\boldsymbol{x}_1) \cdots \boldsymbol{A}(\boldsymbol{x}_{2^{i-1}})) \quad \forall \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{2^{i-1}}$$

⇒ Most linear RNNs cannot solve parity (only positive eigenvalues)

⇒ Diagonal real-valued linear RNNs cannot do modular counting

Trading off Expressivity and Computational Complexity

$$\boldsymbol{A}\left(\boldsymbol{x}_t\right) =$$

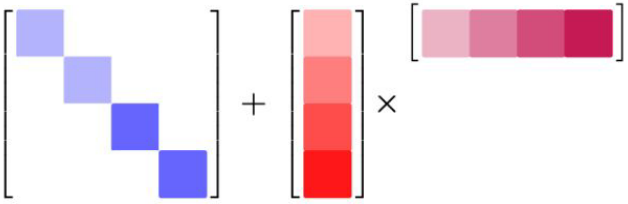


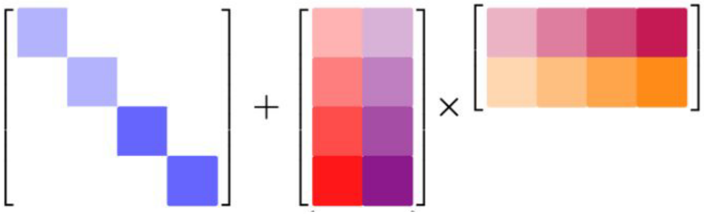
Diagonal: → Mamba, mLSTM, GLA

Very fast computation, but can’t go beyond parity

Trading off Expressivity and Computational Complexity

$\boldsymbol{A}(\boldsymbol{x}_t) =$





Rank 1 Update → DeltaNet

Rank 2 Update → DeltaProduct

$$I - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top$$

$$\left(I - \beta_{1,t} \boldsymbol{k}_{1,t} \boldsymbol{k}_{1,t}^\top\right) \left(I - \beta_{2,t} \boldsymbol{k}_{2,t} \boldsymbol{k}_{2,t}^\top\right)$$

$$(1 - \beta_{i,t}) \in [-1, 1] \implies ||\boldsymbol{A}(\boldsymbol{x}_t)|| \leq 1$$

$$||\boldsymbol{k}_{1,t}|| = 1$$

Stable recurrence!

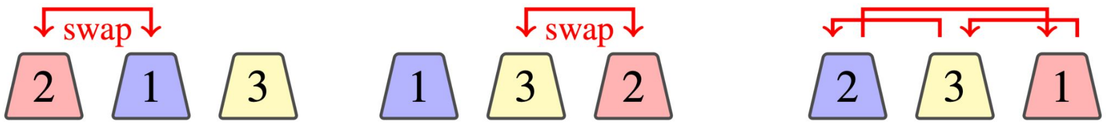
Products of Generalized Householder (GH) Matrices

$$\mathcal{M}_k^n(\Omega) := \{\boldsymbol{C}_1\boldsymbol{C}_2\cdots\boldsymbol{C}_k : \boldsymbol{C}_i = \boldsymbol{I} - \beta_i \boldsymbol{v}_i\boldsymbol{v}_i^\top, \quad (1-\beta_i) \in \Omega, \quad \boldsymbol{v}_i \in \mathbb{R}^n, \|\boldsymbol{v}_i\| = 1\}$$

Eigenvalue range

For DeltaNet, $k = 1, \Omega = [0, 1]$

Orthogonal matrices $(\neq \boldsymbol{I})$ are included only if $-1 \in \Omega \;\; (\beta_i = 2)$



$$\underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{I} - 2\boldsymbol{v}_1\boldsymbol{v}_1^\top} \times \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}}_{\boldsymbol{I} - 2\boldsymbol{v}_2\boldsymbol{v}_2^\top} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Contribution: Expressivity of Products of GH Matrices

Thm. 3 (Permutations): Finite precision linear RNNs with one layer where state-transition matrices are in $\mathcal{M}^n_{k-1}([-1,1])$ can model any FSA whose transitions $\delta(\cdot, w) : Q \to Q$ correspond to permutations of at most k elements, when n is large enough.

Thm. 4 (General FSA): Finite precision linear RNNs with multiple layers where state-transition matrices are in $\mathcal{M}^n_n([-1,1])$ for a large enough n, can model any finite state automaton.

⇒ We can easily modify DeltaNet to have state transition matrices in $\mathcal{M}^n_1([-1,1])$ and thus model **swap permutations**

Eigenvalue Extension for Mamba and DeltaNet

$\boldsymbol{A}(\boldsymbol{x}_t) \Rightarrow$

	$[0,1]$	$[-1,1]$
Mamba	$\mathrm{Diag}(\boldsymbol{s}(\boldsymbol{x}_t))$	
DeltaNet	$\boldsymbol{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top$	

Change for DeltaNet is a *one-liner*!

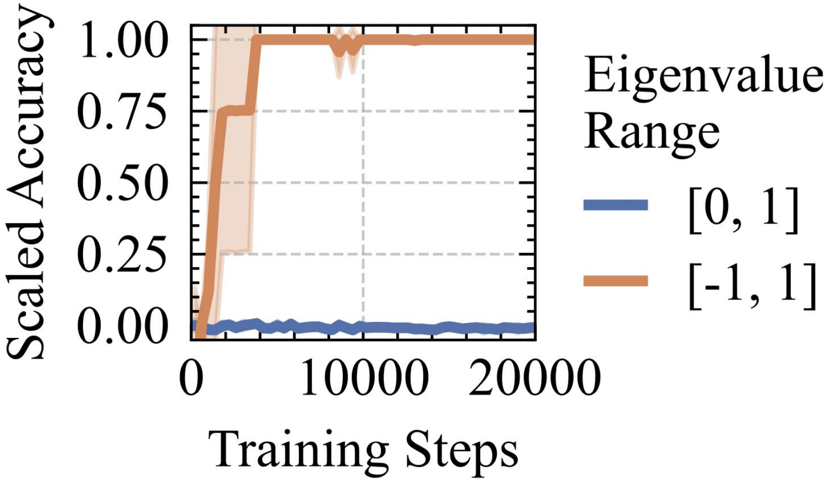
```
      if self.use_beta:
- beta = rearrange(self.b_proj(hidden_states), 'b l h -> b h l').sigmoid()
+ beta = 2 * rearrange(self.b_proj(hidden_states), 'b l h -> b h l').sigmoid()
      else:
         beta = q.new_ones(q.shape[0], q.shape[1], q.shape[2])
```

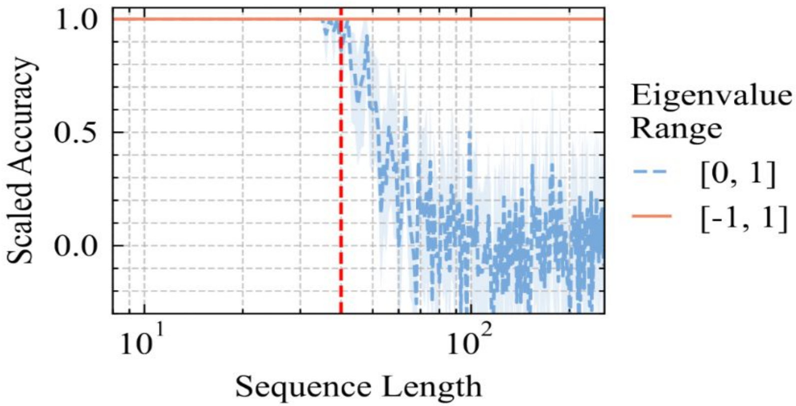
Code from [Flash Linear Attention](#) (Yang et al. 2024)

Experiments - Parity

	Parity
Transformer	0.022
mLSTM	0.087 (0.04)
sLSTM	**1.000** (1.00)
Mamba $[0, 1]$	0.000
Mamba $[-1, 1]$	**1.000**
DeltaNet $[0, 1]$	0.017
DeltaNet $[-1, 1]$	**1.000**

Accuracy



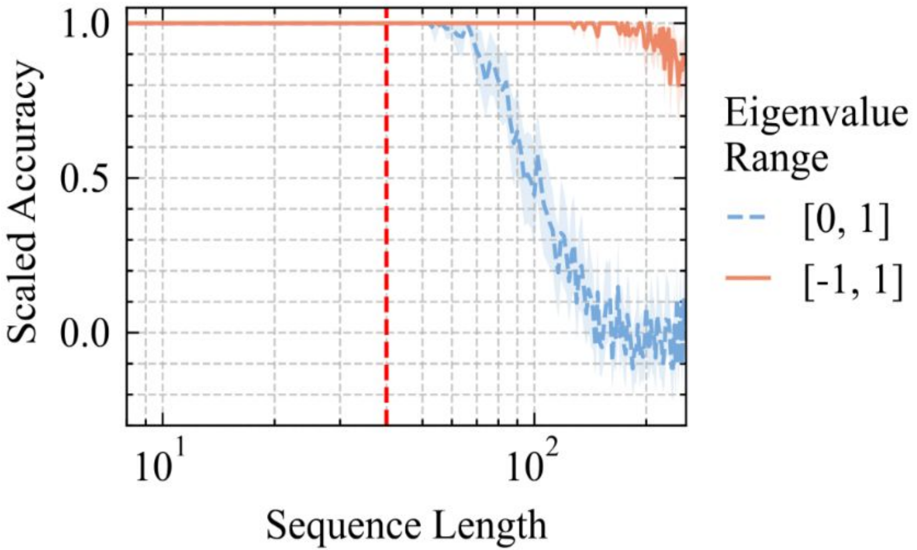


Experiments - Modular Arithmetic

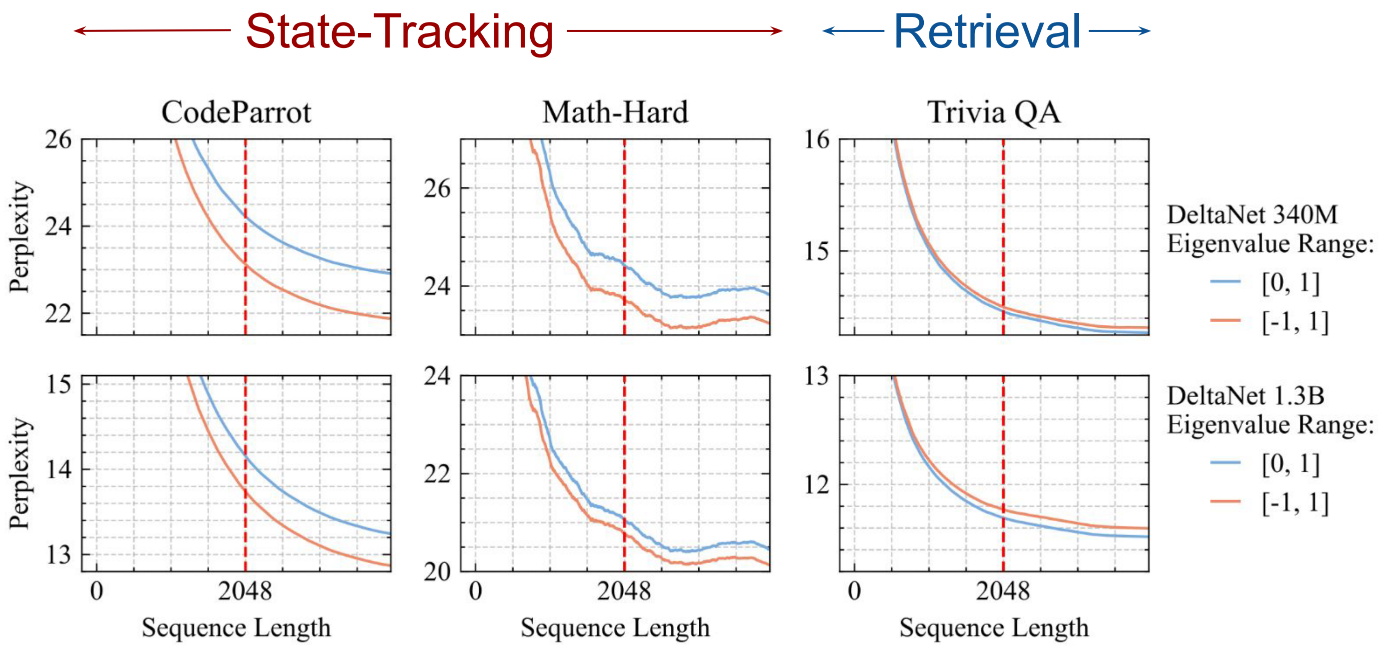
***Mod. Arithm.* (w/o brackets):**
2 - 3 - 3 * 2 mod 5 = 3

	Mod. Arithm. (w/o brackets)
Transformer	0.031
mLSTM	0.040 (0.04)
sLSTM	**0.787** (1.00)
Mamba $[0, 1]$	0.095
Mamba $[-1, 1]$	**0.241**
DeltaNet $[0, 1]$	0.314
DeltaNet $[-1, 1]$	**0.971**

DeltaNet



Experiments - Language Modelling



→ *Note:* Extended eigenvalue range doesn't cause training instability

Conclusion

- Inclusion of negative eigenvalues expands the expressivity of linear RNNs allowing them to solve state-tracking problems
- Efficient non-diagonal linear RNNs such as DeltaNet and RWKVv7 are promising due to their superior expressivity compared to Mamba.

Future Directions:

- What is the limit of the expressivity of DeltaNet [-1,1]?
- Is standard pretraining exploiting the increased expressivity? Are there better ways?
- What is the trade-off between in-context associative recall and state-tracking?