Unlocking State-Tracking in Linear RNNs through Negative Eigenvalues

AutoML Seminar
9th of January 2025

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

Outline

Part 1 (Riccardo)

1. State-tracking
2. Linear RNNs and Known Limits of Current Architectures
3. Contribution 1: Limits of Linear RNNs.
4. Contribution 2: How to unlock State-tracking.

Part 2 (Julien)

1. Extending the Eigenvalue Range of Mamba and DeltaNet
2. Synthetic Experiments
3. Language Modeling Experiments
4. Conclusion and Future Works

State Tracking

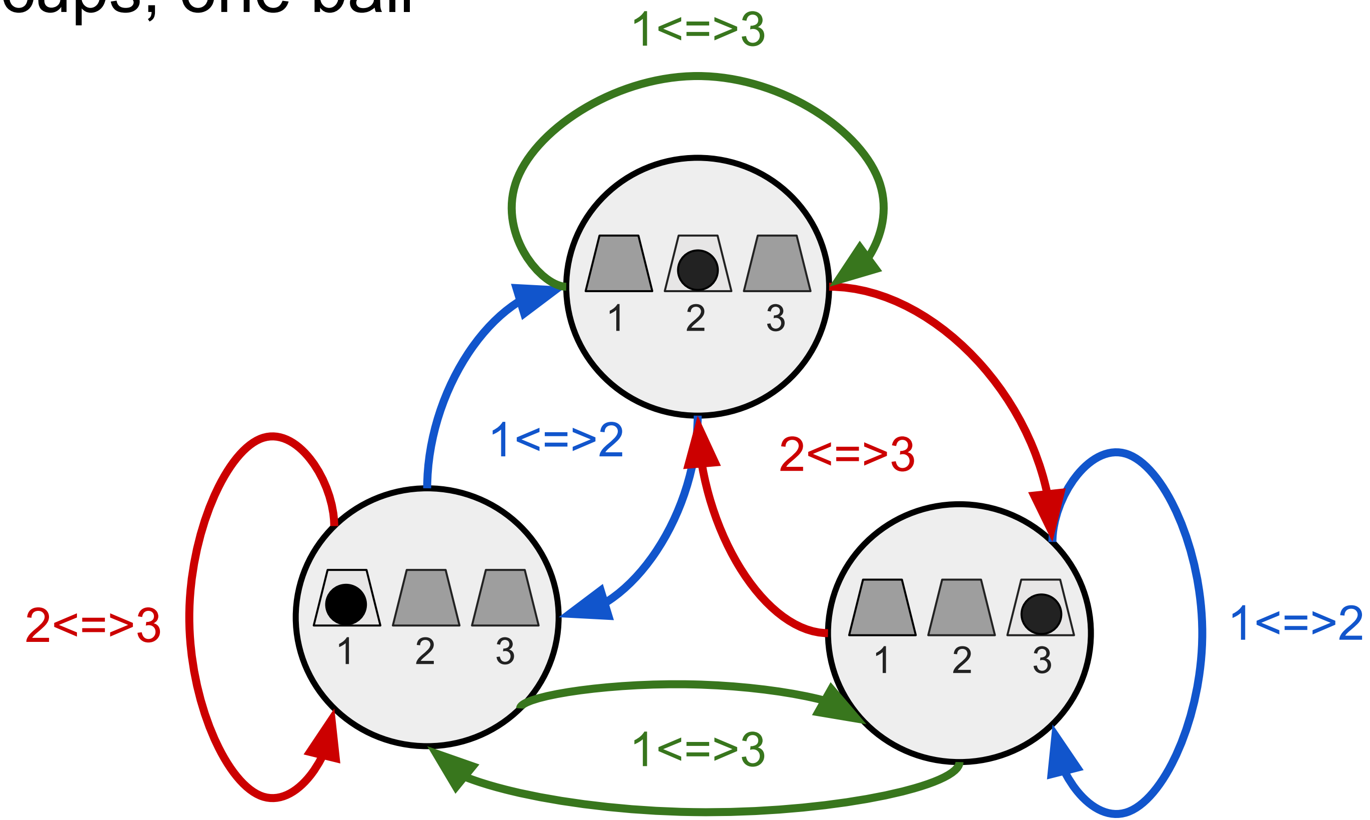
Show Initial State

State Transitions

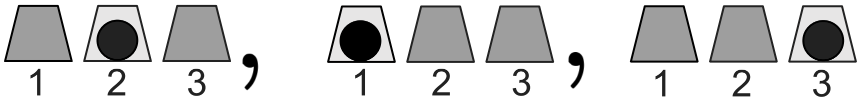
Where is the ball?

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

3 cups, one ball



Finite State Automata (FSA)

States (Finite set) → $Q = \{$  $\}$

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

Initial state

$$q_0 \in Q$$

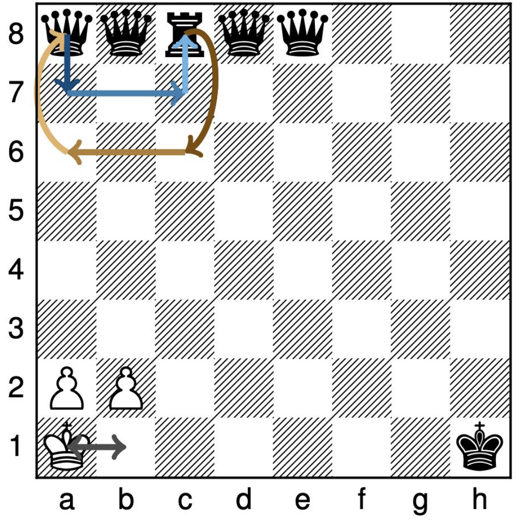
$$\delta : Q \times \Sigma \rightarrow Q$$

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.

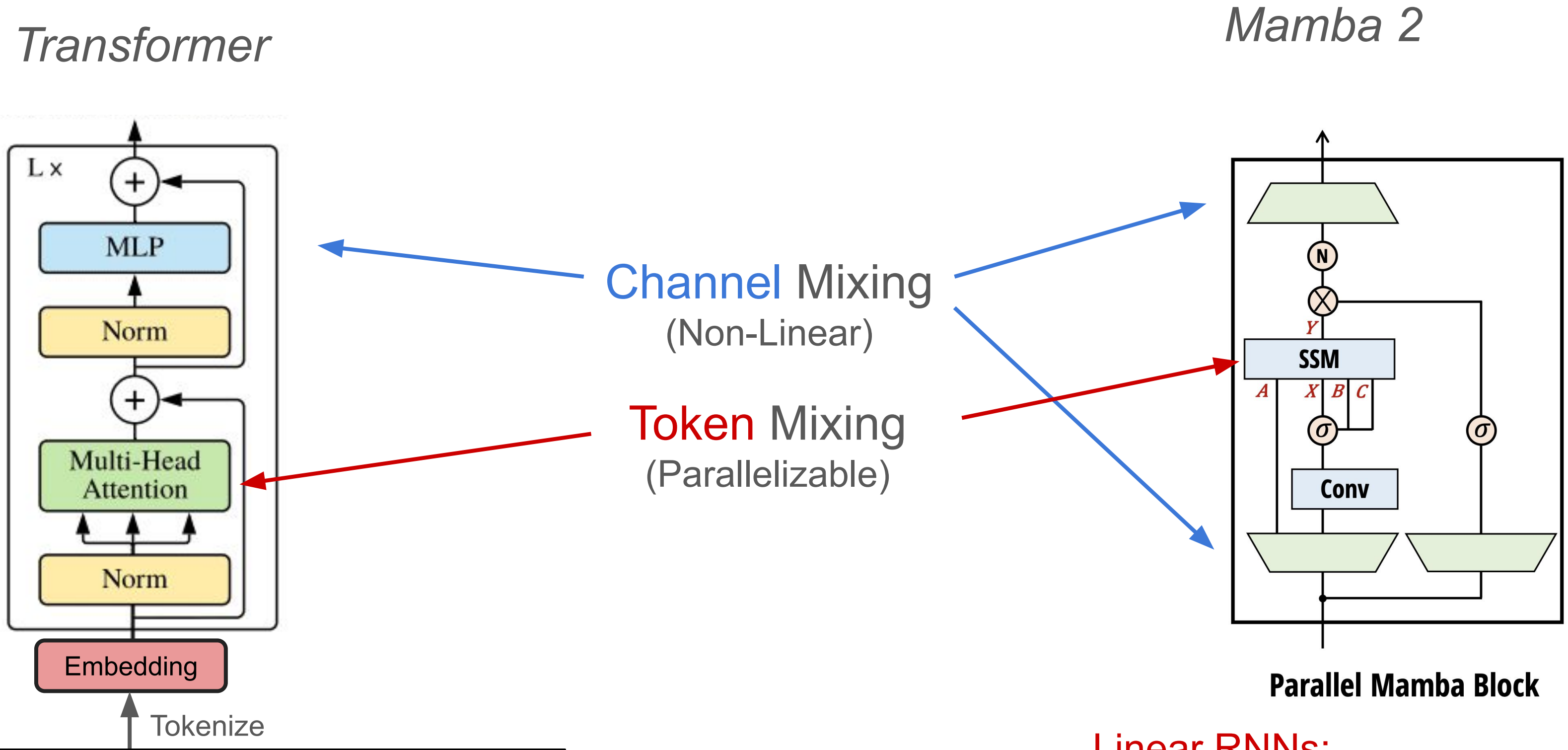
Images modified from Merrill, William, Jackson Petty, and Ashish Sabharwal. "The illusion of state in state-space models." *ICML* (2024).

State-tracking?

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

Alice, Bob and Carl each have a coin. Carl is the only one having a penny. Carl trades ***his penny*** *with Alice.*

Modern Language Modeling Architectures



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

Linear RNNs:
More efficient for long sequences

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

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 recursion efficiently parallelizable

Linear RNNs (One Layer)

State matrix

input token

Output

Channel mix (MLP)

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

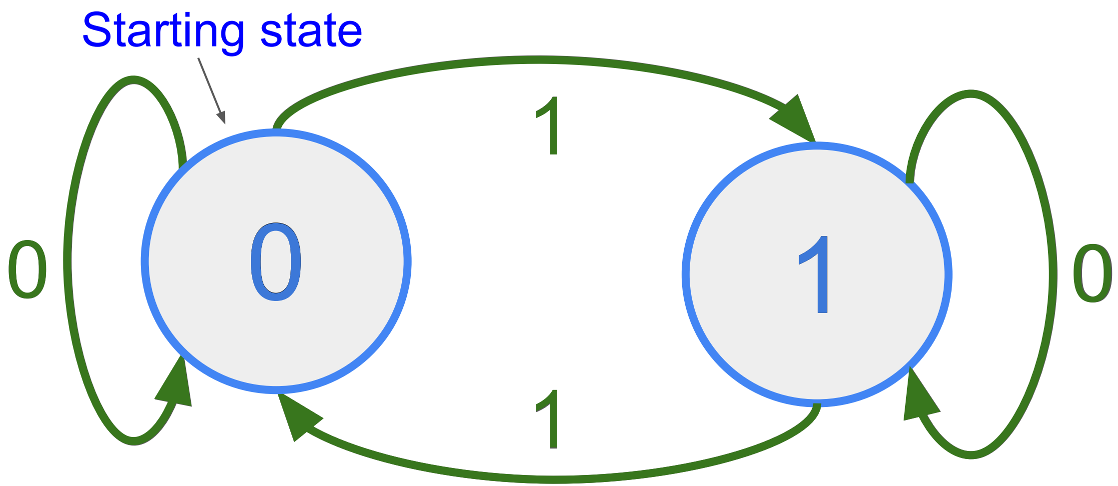
State-transition matrix

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

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.

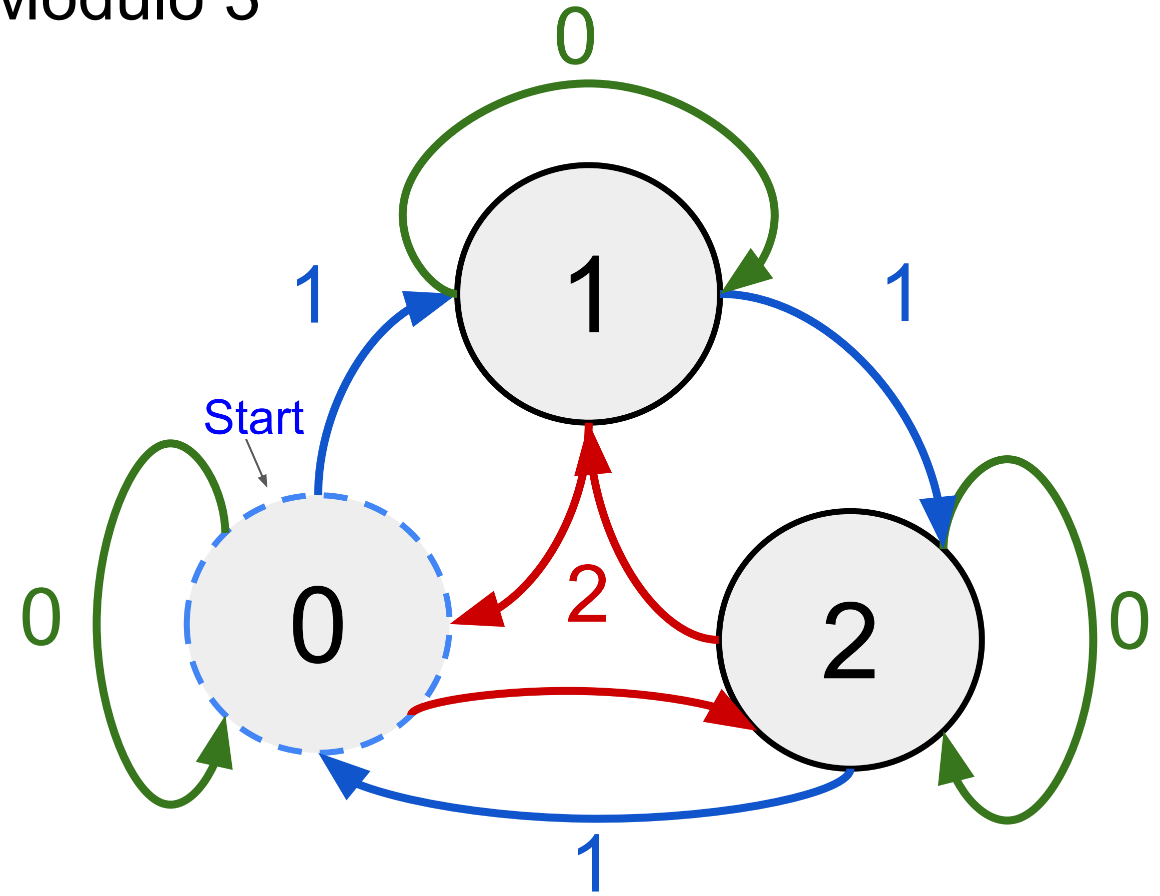
Parity (2-cups game, addition modulo 2)



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

We'd like to handle arbitrary long sequences

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

$$a(x_i) = 1$$

$$h_t = \sum_{i=1}^{t} x_i \qquad y_t = h_t \bmod 2 \quad \textbf{(state blows up!)}$$

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 → $\boldsymbol{A}(\boldsymbol{x}_t)$	$\boldsymbol{B}(\boldsymbol{x}_t)$
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	$\boldsymbol{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top$ ← GH, non-diagonal	$\beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top$

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

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 provably 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 **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 trans-
formers. 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

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

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

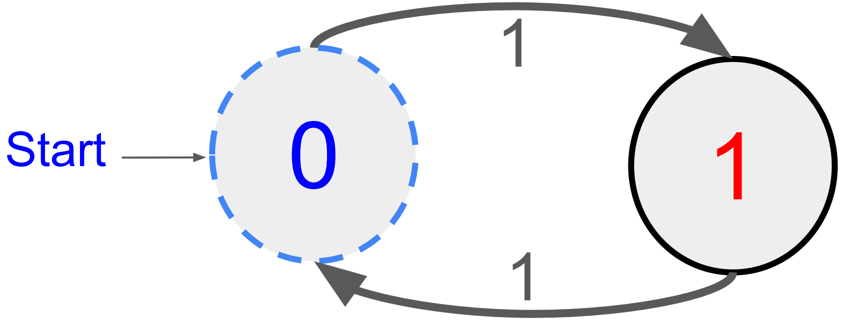
$$\lambda \in \mathbb{R} \quad \forall \lambda \in \mathrm{eigs}(\boldsymbol{A}(\boldsymbol{x}_1) \cdots \boldsymbol{A}(\boldsymbol{x}_{2^{i-1}})) \quad \forall \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{2^{i-1}}$$

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

⇒ Diagonal real-valued linear RNNs cannot do modular counting

Theorem 1 - Proof Idea (Same as Sarrof et al. 2024)

When the input is constant, i.e. $\boldsymbol{x} = 1, 1, 1, 1, \ldots$, then the Linear RNN output in finite precision becomes constant while the state of the parity automaton alternates between 0 and 1.



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

Proof Sketch (One Layer): Unrolling the Recurrence

$$\boldsymbol{x} = (1, 1, 1, 1, \ldots, 1)$$

$$\Downarrow \quad \text{Linear Time Invariant Recurrence}$$

State $\longrightarrow$ $$\boldsymbol{H}_t = \boldsymbol{A}(1)\boldsymbol{H}_{t-1} + \boldsymbol{B}(1)$$

Unroll the recurrence with $\boldsymbol{H}_0 = 0$ (for simplicity) gives

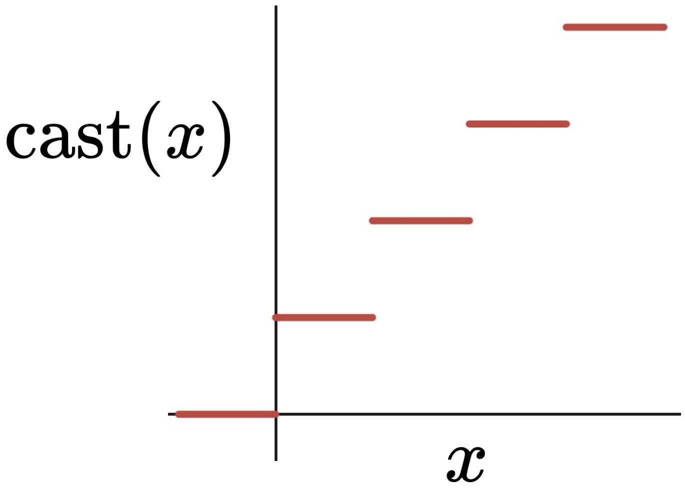
$$\boldsymbol{H}_t = \sum_{k=0}^{t-1} \boldsymbol{A}(1)^k \boldsymbol{B}(1)$$

Finite Precision

$$\mathrm{cast} : \mathbb{R} \to \mathbb{D}$$

Finite Set of reals

$$\mathrm{cast}(x) \in \arg\min_{z \in \mathbb{D}} |z - x|$$



Extended to matrices by applying it separately to real and imaginary part of each element of the matrix

State in Finite Precision

Infinite precision:

$$\boldsymbol{H}_t = \sum_{k=0}^{t-1} \boldsymbol{A}(1)^k \boldsymbol{B}(1)$$

Finite precision:

$$\widehat{\boldsymbol{H}}_t = \mathrm{cast}\left(\sum_{k=0}^{t-1} \mathrm{cast}\left(\boldsymbol{A}(1)^k \boldsymbol{B}(1) \right) \right)$$

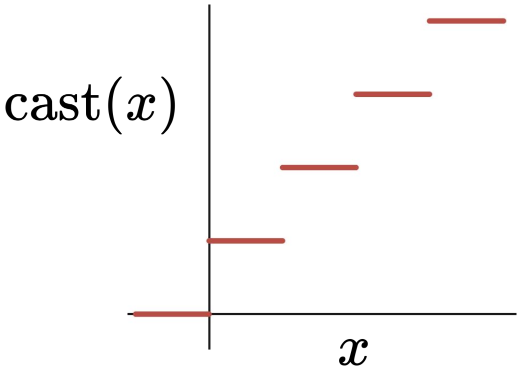
Simplified model: repeated matrix multiplies and sums are in infinite precision, similar to Merrill et al. (2024)

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

State in Finite Precision

$$\mathrm{cast}(x) \in \arg\min_{z \in \mathbb{D}} |z - x|$$

Finite Set of reals



Cast is applied to matrices by applying it separately to real and imaginary part of each element of matrices and converts real numbers into finite precision:

$$\widehat{\boldsymbol{H}}_t = \mathrm{cast}\left(\sum_{k=0}^{t-1} \mathrm{cast}\left(\boldsymbol{A}(1)^k \boldsymbol{B}(1)\right)\right)$$

Simplified model: repeated matrix multiplies and sums are in infinite precision

Matrix Powers Using the Jordan Canonical Form

$$\boldsymbol{A}(1)^k = \boldsymbol{P}\boldsymbol{J}^k\boldsymbol{P}^{-1}$$

$\boldsymbol{J}^k$ Block Diagonal with block i

$$\boldsymbol{J}_i^k = \begin{bmatrix} \lambda_i^k & \binom{k}{1}\lambda_i^{k-1} & \binom{k}{2}\lambda_i^{k-2} & \cdots & \cdots & \binom{k}{k_i-1}\lambda_i^{k-k_i+1} \\ & \lambda_i^k & \binom{k}{1}\lambda_i^{k-1} & \cdots & \cdots & \binom{k}{k_i-2}\lambda_i^{k-k_i+2} \\ & & \ddots & \ddots & \vdots & \vdots \\ & & & \ddots & \ddots & \vdots \\ & & & & \lambda_i^k & \binom{k}{1}\lambda_i^{k-1} \\ & & & & & \lambda_i^k \end{bmatrix}$$

$$\boldsymbol{k_i} \in \mathbb{N}, \boldsymbol{\lambda_i} \in \mathbb{C}$$

Eigenvalue

Real and Positive Eigenvalues

The imaginary and real part of each element of $\boldsymbol{A}(1)^k\boldsymbol{B}(1)$ take the form

$$a_k = \sum_{i=1}^{n^2} c_i \binom{k}{m_i} \lambda_i^k \qquad \lambda_i \in \mathbb{C}, c_i \in \mathbb{R}, m_i \in \mathbb{N}$$

Eigenvalue

Exponential (Complex base)

$$\binom{k}{m_i} \lambda_i^k = O(k^{m_i} \lambda_i^k)$$

Polynomial

Real and Positive Eigenvalues

The imaginary and real part of each element of $\boldsymbol{A}(1)^k \boldsymbol{B}(1)$ take the form

$$a_k = \sum_{i=1}^{n^2} c_i \binom{k}{m_i} \underset{\substack{\uparrow \\ \text{Eigenvalue}}}{\lambda_i^k} \qquad \lambda_i \in \mathbb{C}, c_i \in \mathbb{R}, m_i \in \mathbb{N}$$

Lemma 1 (values become constant in finite precision)

$$\text{if } \lambda_i \in \mathbb{R}, \lambda_i > 0 \implies \exists \bar{k} \geq 0, \bar{a} \in \mathbb{R} : \text{cast}\left(a_k\right) = \bar{a} \quad \forall k \geq \bar{k}$$

$\implies$ state and layer output become constant in finite precision

$$\exists \bar{t} \geq 0 : \begin{array}{c} \widehat{\boldsymbol{H}}_t = \overline{\boldsymbol{H}} \\ \boldsymbol{y}_t = \text{dec}(\widehat{\boldsymbol{H}}_t, \boldsymbol{x}_t) = \bar{y} \end{array} \quad \forall t \geq \bar{t}$$

Parity Can't Be Modeled - End of Proof Sketch

If $\mathbf{A}(1)$ has only **real positive** eigenvalues, then

$$
\begin{array}{llcccccccccccccc}
\text{input} & \boldsymbol{x} = & \dots & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\
\text{LRNN output} & \bar{\boldsymbol{y}} = & \dots & \bar{y} & \bar{y} & \bar{y} & \bar{y} & \bar{y} & \bar{y} & \bar{y} & \bar{y} & \bar{y} & \bar{y} & \bar{y} & \bar{y} & \dots \\
 & & & & \neq & & \neq & & \neq & & \neq & & \neq & & \neq & \\
\text{parity} & y = & \dots & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & \dots
\end{array}
$$

The proof can then proceed by induction over the number of layers

Theorem 2 Proof Sketch - Real Eigenvalues

$$a_k = \sum_{i=1}^{n^2} c_i \binom{k}{m_i} \lambda_i^k \qquad \lambda_i \in \mathbb{C}, c_i \in \mathbb{R}, m_i \in \mathbb{N}$$

Lemma 1 (Part 2): $\text{if} \quad \lambda_i \in \mathbb{R} \implies \lambda_i^k = \text{sign}(\lambda_i)^k |\lambda_i|^k$

$$\Downarrow$$

$$\exists \bar{k}, \geq 0, \bar{a}_1, \bar{a}_2 \in \mathbb{R} : \text{cast}\left(a_{2k}\right) = \bar{a}_1, \text{cast}\left(a_{2k+1}\right) = \bar{a}_2 \quad \forall k \geq \bar{k}$$

$$\implies \quad \begin{gathered} \exists \bar{t} \geq 0 : \widehat{\boldsymbol{H}}_{2t} = \overline{\boldsymbol{H}}_1, \widehat{\boldsymbol{H}}_{2t+1} = \overline{\boldsymbol{H}}_2 \quad \forall t \geq \bar{t} \\ \overline{\boldsymbol{y}}_{2t} = \overline{y}_1, \overline{\boldsymbol{y}}_{2t+1} = \overline{y}_2 \end{gathered}$$

layer output

Addition Modulo 3 Can't Be Modeled

If $\mathbf{A}(1)$ has only **real** eigenvalues, then

input $\quad \boldsymbol{x} = \ldots 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \ldots$

layer output $\quad \bar{\boldsymbol{y}} = \ldots \bar{y}_1\bar{y}_2\bar{y}_1\bar{y}_2\bar{y}_1\bar{y}_2\bar{y}_1\bar{y}_2\bar{y}_1\bar{y}_2\bar{y}_1\bar{y}_2 \ldots$

add mod 3 $\quad \boldsymbol{y} = \ldots 0\ 1\ 2\ 0\ 1\ 2\ 0\ 1\ 2\ 0\ 1\ 2 \ldots$

Multiple layers is not as easy as for parity since the output is not constant as the input

Products of Generalized Householder (GH) Matrices

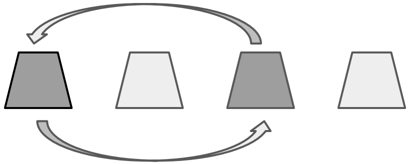
$$\mathcal{M}_k^n(\Omega) := \left\{ \boldsymbol{C}_1 \boldsymbol{C}_2 \cdots \boldsymbol{C}_k : \boldsymbol{C}_i = \boldsymbol{I} - \beta_i \boldsymbol{v}_i \boldsymbol{v}_i^\top, \quad (1 - \beta_i) \in \Omega, \quad \boldsymbol{v}_i \in \mathbb{R}^n, \|\boldsymbol{v}_i\| = 1 \right\}$$

Eigenvalue range

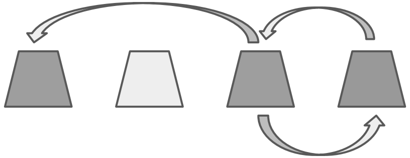
DeltaNet has state-transition matrices in $\mathcal{M}_1^n([0,1])$

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

$$\boldsymbol{I} - 2\boldsymbol{v}_1\boldsymbol{v}_1^\top =$$



$$(\boldsymbol{I} - 2\boldsymbol{v}_1\boldsymbol{v}_1^\top)(\boldsymbol{I} - 2\boldsymbol{v}_2\boldsymbol{v}_2^\top) =$$



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 $\delta(\cdot, w) : Q \to Q$ are permutations of at most k elements.

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**

Recap of Theoretical Contributions

1. Any Linear RNN with state transition matrices having only **positive real eigenvalues cannot solve parity**.
2. **Diagonal and Triangular Linear RNNs cannot solve modular counting**, even with negative real eigenvalues.

1. Linear RNNs with **products of GH state transition matrices, each with negative eigenvalues**, can mimic any FSA and
2. Can do it with products of k-1 GH matrices and **one layer** if the transitions are only permutations of at most k elements.

Open question:

- What can be done with **a single GH matrix + multiple layers**?
  Addition modulo m can be done with 2 layers! (See Appendix).

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

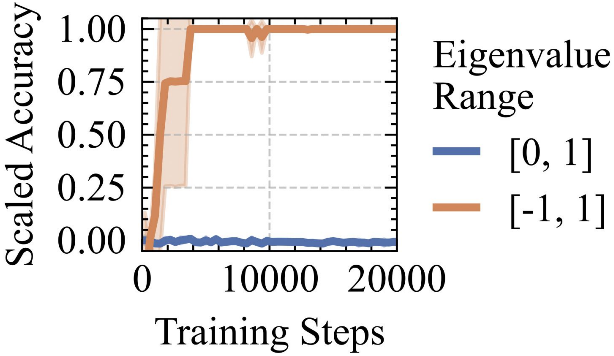
Change for DeltaNet is a *one-liner*!

```
        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])
```

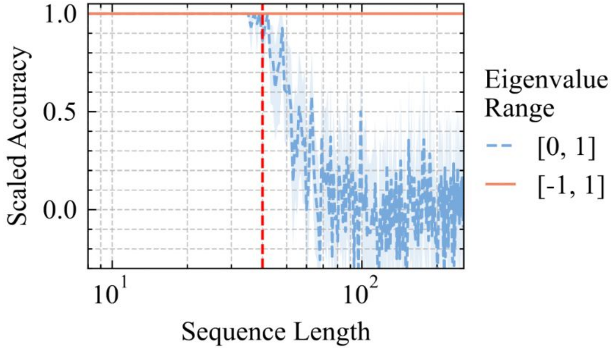
Code from [Flash Linear Attention](#) (Yang et al. 2024)

Experiments - Chomsky Hierarchy

→ Can we actually solve parity using linear RNNs?



	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**



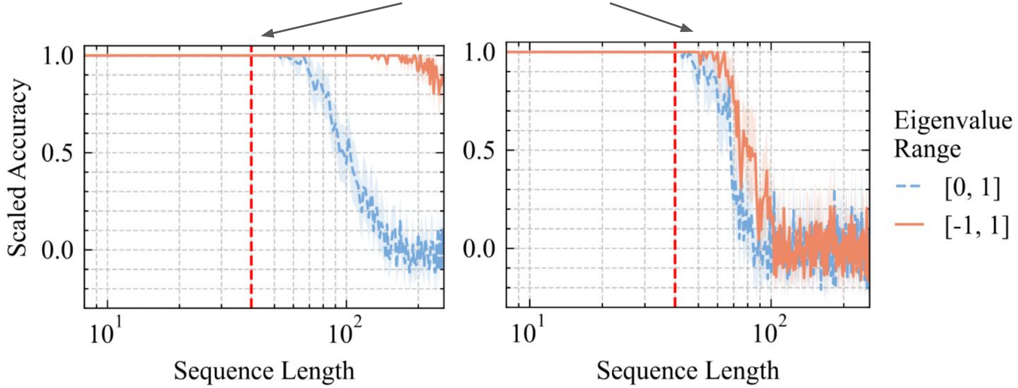
Experiments - Chomsky Hierarchy

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

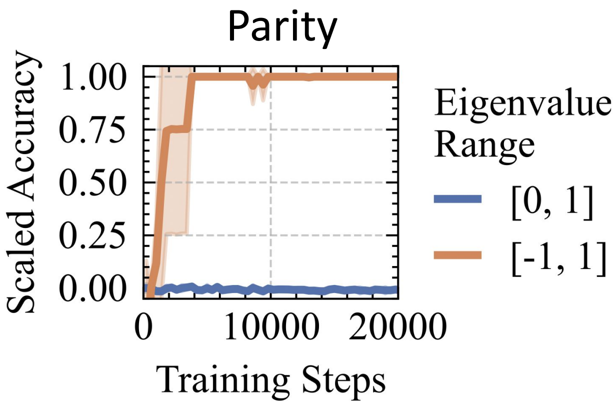
(w/ brackets): ((((3+3)+-1)+-2)-((3-(-3))+((1)+4))) mod 5 = 2

	Mod. Arithm. (w/o brackets)	Mod. Arithm. w/ brackets)
Transformer	0.031	0.025
mLSTM	0.040 (0.04)	0.034 (0.03)
sLSTM	**0.787** (1.00)	**0.173** (0.57)
Mamba $[0, 1]$	0.095	0.092
Mamba $[-1, 1]$	**0.241**	**0.136**
DeltaNet $[0, 1]$	0.314	0.137
DeltaNet $[-1, 1]$	**0.971**	**0.200**

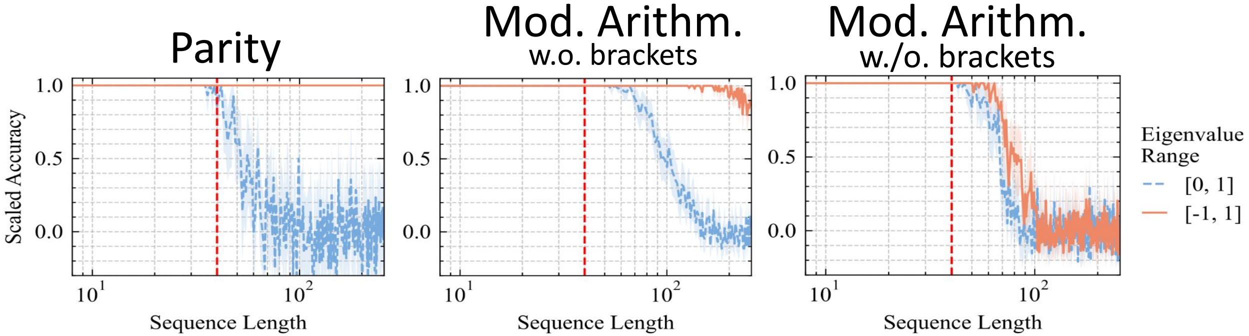
DeltaNet



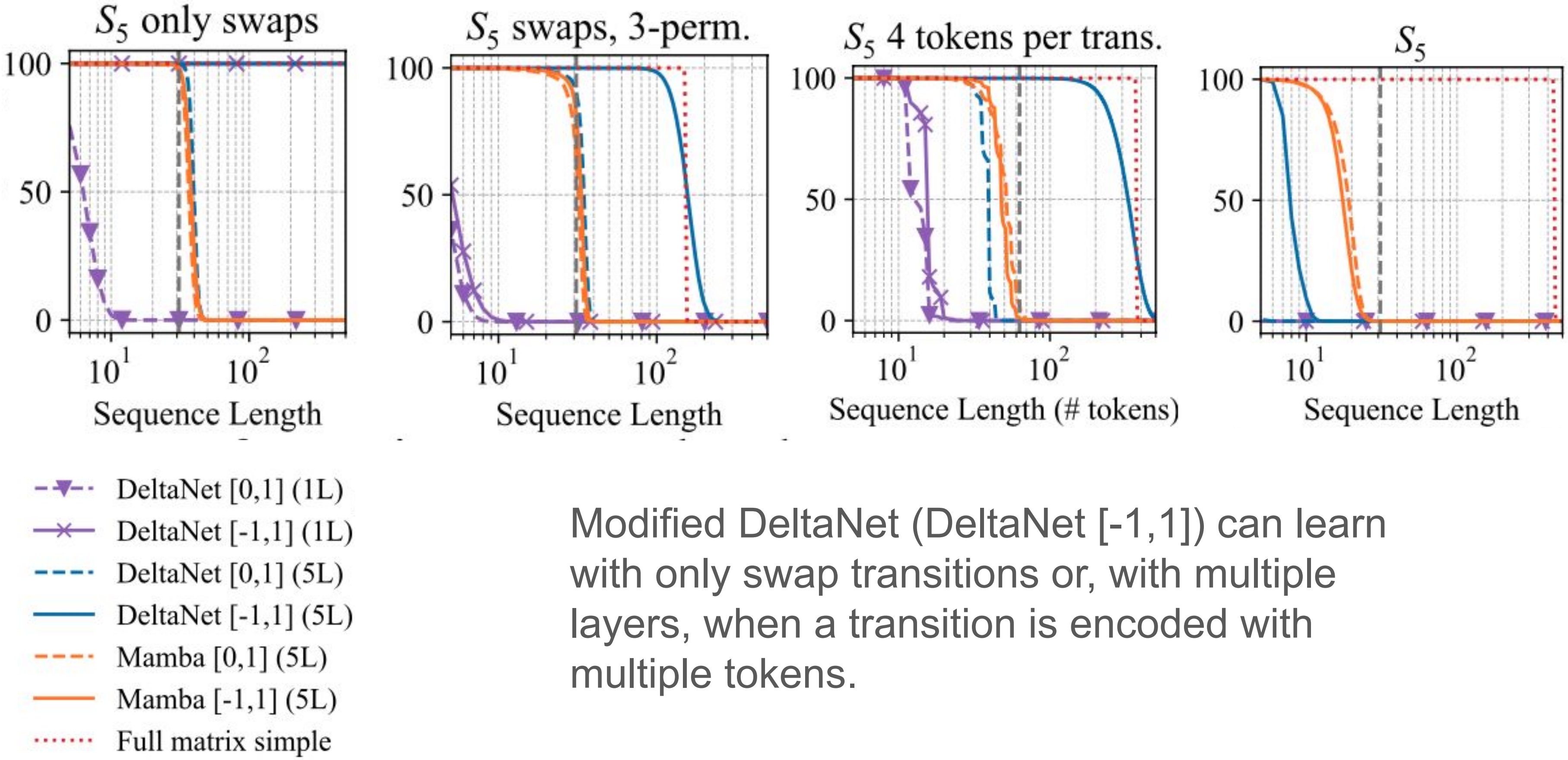
Synthetic Experiments - Part 1



	Parity	Mod. Arithm. (w/o brackets)	Mod. Arithm. w/ brackets)
Transformer	0.022	0.031	0.025
mLSTM	0.087 (0.04)	0.040 (0.04)	0.034 (0.03)
sLSTM	**1.000** (1.00)	**0.787** (1.00)	**0.173** (0.57)
Mamba $[0, 1]$	0.000	0.095	0.092
Mamba $[-1, 1]$	**1.000**	**0.241**	**0.136**
DeltaNet $[0, 1]$	0.017	0.314	0.137
DeltaNet $[-1, 1]$	**1.000**	**0.971**	**0.200**



Synthetic Experiments - Part 2

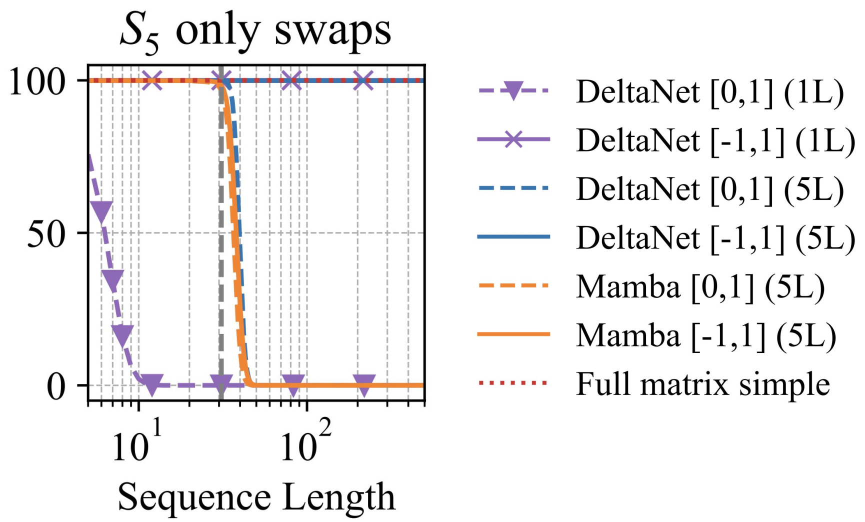


Modified DeltaNet (DeltaNet [-1,1]) can learn with only swap transitions or, with multiple layers, when a transition is encoded with multiple tokens.

Experiments - Permutation Groups

S_5 (Permutation group of 5 elements) (Only swaps)

Example: S_5 only swaps:
(1, 2, 3, 4, 5) ∘ (1 → 2, 2 → 1)
= (2, 1, 3, 4, 5)



Results:
→ DeltaNet [-1, 1] can solve S5 only swaps (even with 1 layer).
→ Mamba [-1, 1] can't.

Experiments - Permutation Groups

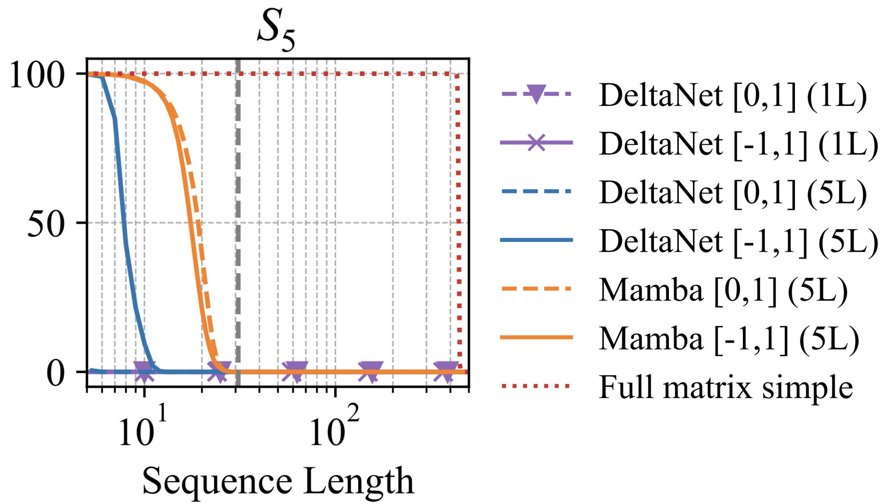
S_5 (Permutation group of 5 elements)

Example: S_5

$(1, 2, 3, 4, 5) \circ$

$(1 \rightarrow 2, 2 \rightarrow 4, 3 \rightarrow 2, 4 \rightarrow 3, 5 \rightarrow 1)$

$= (2, 4, 2, 3, 1)$



Results:

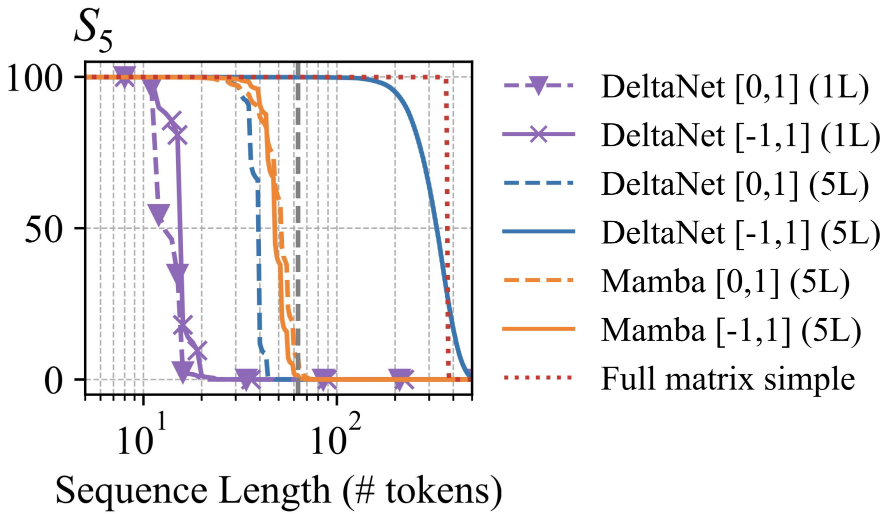
→ DeltaNet and Mamba can't solve S5 (expected)

→ Linear RNN with full state-transition matrix can learn to solve S5

Experiments - Permutation Groups

S_5 (Permutation group of 5 elements)

How can we solve S5 using DeltaNet?
→ **Products of Householders**



Experiments - Permutation Groups

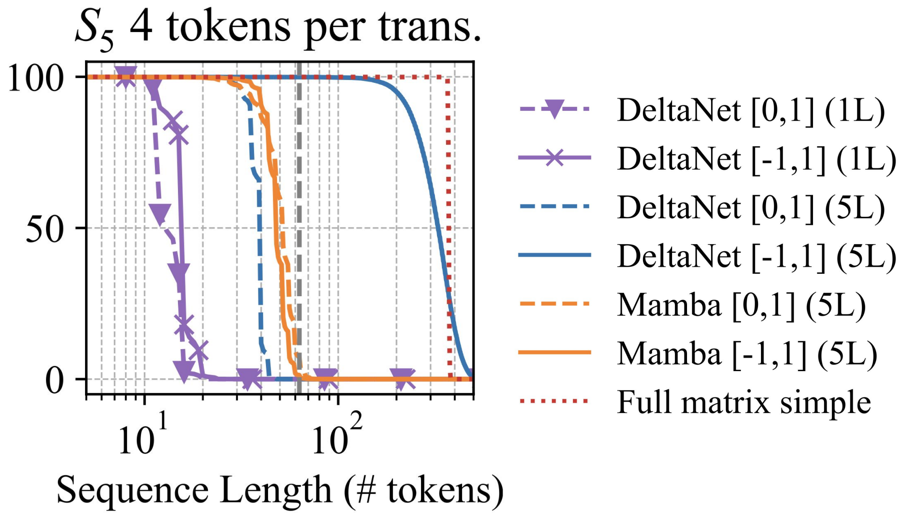
S_5 (Permutation group of 5 elements)

Example: S_5

$(1, 2, 3, 4, 5) \circ$

$(1 \rightarrow 2, 2 \rightarrow 4, 3 \rightarrow 2, 4 \rightarrow 3, 5 \rightarrow 1)$

$= (2, 4, 2, 3, 1)$

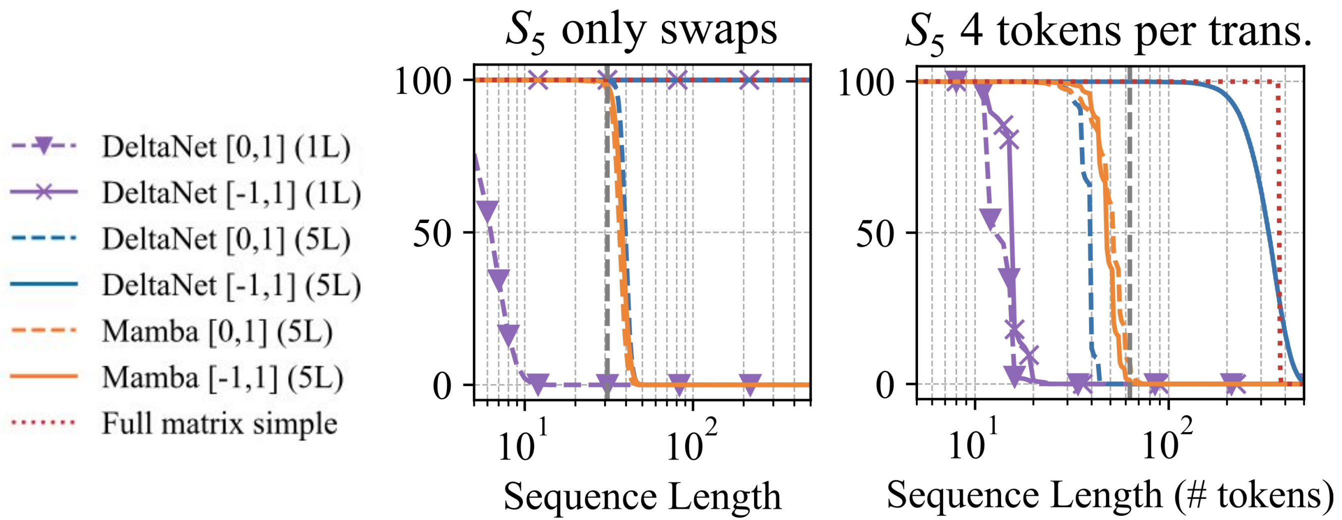


Results:

→ DeltaNet and Mamba can't solve S5 (expected)

→ Linear RNN with full state-transition matrix can learn to solve S5

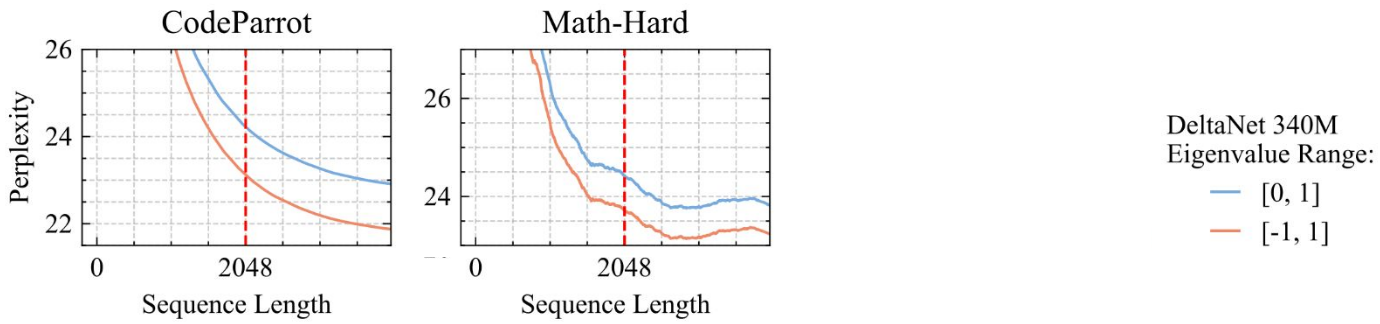
Experiments - Permutation Groups



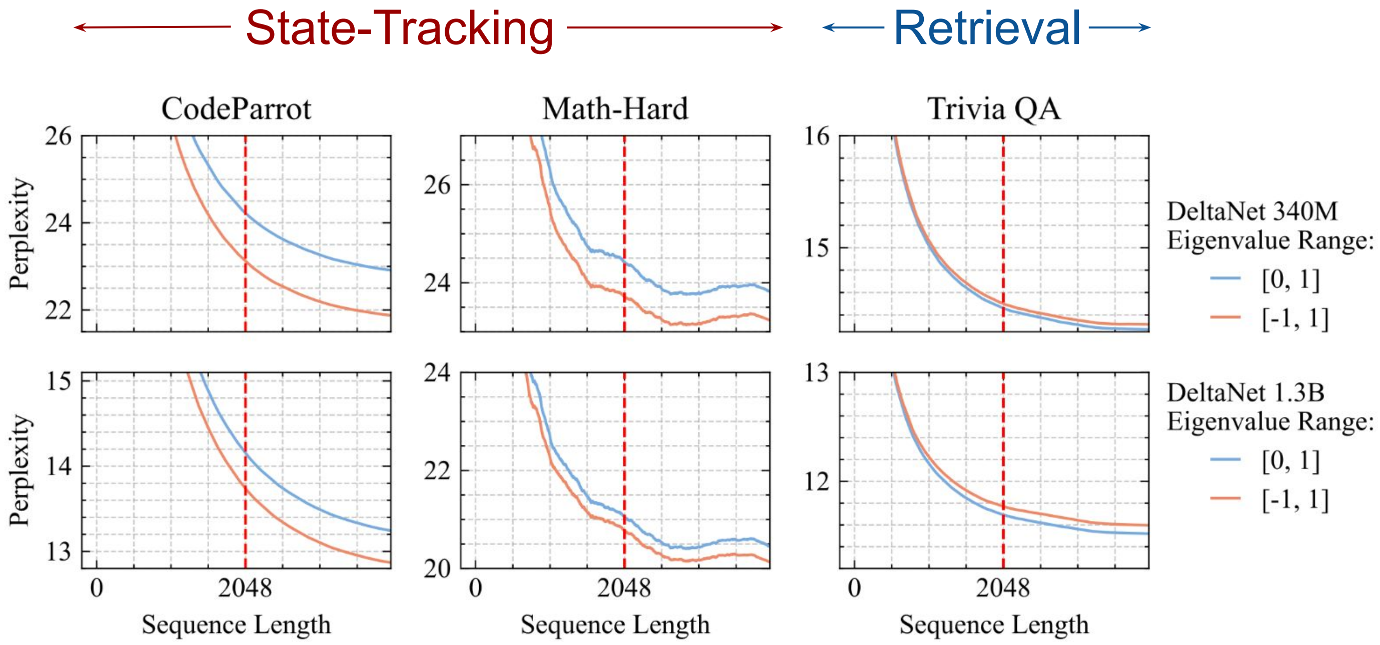
Experiments - Language Modelling

State-Tracking

Retrieval



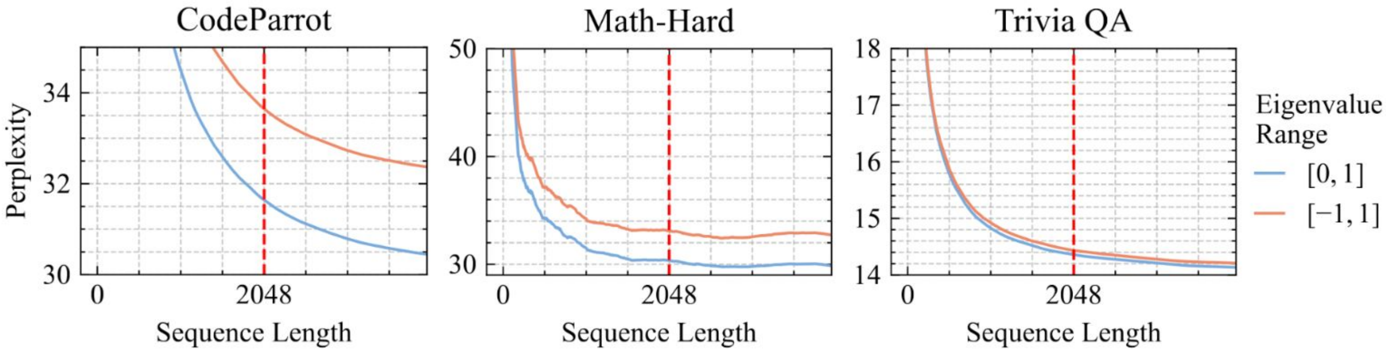
Experiments - Language Modelling



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

Experiments - Language Modelling

Results for **Mamba 370M:**



- *Mamba* doesn't benefit from extended eigenvalue range in language modelling.

Conclusion

- Inclusion of negative eigenvalues expands the expressivity of linear RNNs.
- DeltaNet is promising due to its superior expressivity compared to Mamba.

- Future Directions:
    - Assess real-world improvements in language modeling.
    - Increase expressivity of linear RNNs through more complex state-transition matrices.
    - Understanding the trade-off between associative recall and state-tracking.