

State-Tracking in Scalable Linear RNNs

ASAP Seminar

Riccardo Grazzi, Julien Siems

Microsoft
Research

universität freiburg

Published as a conference paper at ICLR 2025

UNLOCKING STATE-TRACKING IN LINEAR RNNs THROUGH NEGATIVE EIGENVALUES

Riccardo Grazi*[♡], **Julien Siems***[◇], **Jörg K.H. Franke**[◇],
Arber Zela[◇], **Frank Hutter**^{◇♣}, **Massimiliano Pontil**^{♡♣}
Equal contribution*, CSML, Istituto Italiano di Tecnologia[♡], University of Freiburg[◇],
ELLIS Institute Tübingen[♣], AI Centre, University College London[♣]

Preprint

DELTAPRODUCT: INCREASING THE EXPRESSIVITY OF DELTANET THROUGH PRODUCTS OF HOUSEHOLDERS

Julien Siems*[◇], **Timur Carstensen***^{◇♣}, **Arber Zela**[◇],
Frank Hutter^{◇♣}, **Massimiliano Pontil**^{♡♣}, **Riccardo Grazi***^{†★}
Equal contribution*, CSML, Istituto Italiano di Tecnologia[♡], University of Freiburg[◇],
ELLIS Institute Tübingen[♣], AI Centre, University College London[♣], Microsoft Research[★]

Outline

- State-tracking
- Limits of Linear RNNs
- How to Enable State-Tracking
- DeltaProduct: Higher rank updates through multiple steps of GD
- Conclusion and Future Directions

State Tracking

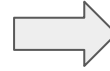


State Tracking

Show Initial State



State Transitions

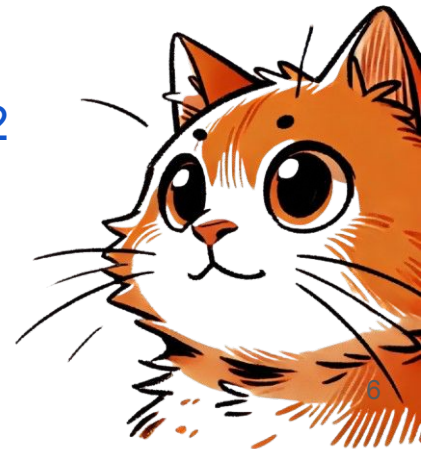
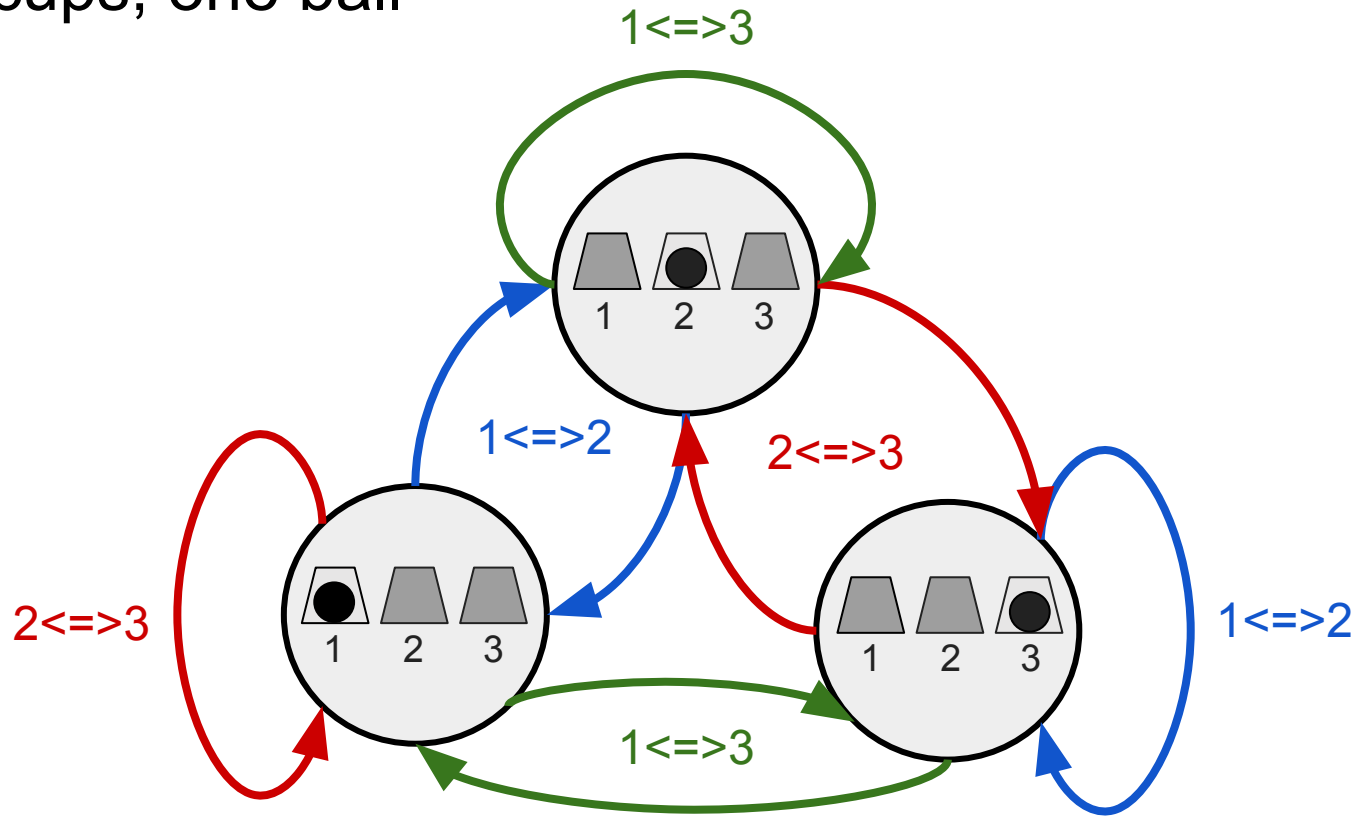


Where is the ball?



- **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) $\longrightarrow Q = \left\{ \begin{array}{c} \text{trapezoid}_1 \text{ trapezoid}_2 \text{ trapezoid}_3 \\ \text{trapezoid}_1 \text{ trapezoid}_2 \text{ trapezoid}_3 \end{array} , \begin{array}{c} \text{trapezoid}_1 \text{ trapezoid}_2 \text{ trapezoid}_3 \\ \text{trapezoid}_1 \text{ trapezoid}_2 \text{ trapezoid}_3 \end{array} , \begin{array}{c} \text{trapezoid}_1 \text{ trapezoid}_2 \text{ trapezoid}_3 \\ \text{trapezoid}_1 \text{ trapezoid}_2 \text{ trapezoid}_3 \end{array} \right\}$

Alphabet (Finite set) $\longrightarrow \Sigma = \{ 1 \leq 2, 1 \leq 3, 2 \leq 3 \}$

Initial state

$$q_0 \in Q$$

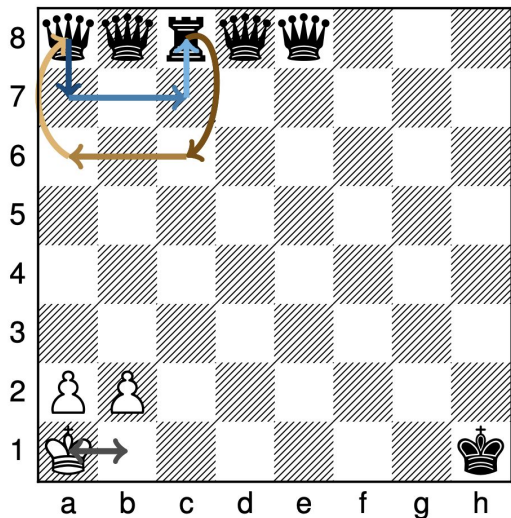
$$\delta : Q \times \Sigma \longrightarrow Q$$

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.

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.*

State Tracking vs Associative Recall



Associative Recall

State Tracking

Memory

The More the Better

Not so High

Data

Abundant

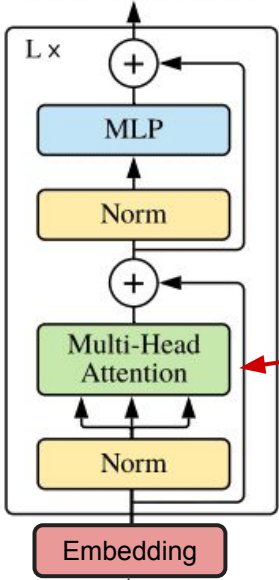
Scarce

LLMs



Modern Language Modeling Architectures

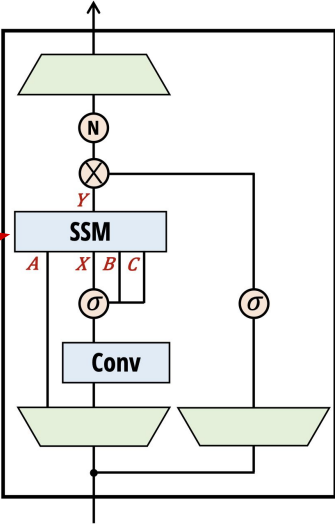
Transformer



Channel Mixing
(Non-Linear)

Token Mixing
(Parallelizable)

Mamba 2



Parallel Mamba Block

Linear RNNs:
More efficient for long sequences

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

Linear RNNs (One Layer)

State matrix input token Output Channel mix (MLP)

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

State-transition matrix

	$\mathbf{A}(\mathbf{x}_t)$	$\mathbf{B}(\mathbf{x}_t)$
Mamba	$\text{Diag}(\exp(-\Delta_t \odot \exp(\mathbf{w}_{1,i})))$	$k_{t,i} \Delta_t \odot \mathbf{x}_t$
GLA	$\text{Diag}(\alpha_t)$	$\mathbf{k}_t \mathbf{v}_t^\top$
DeltaNet	$\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top$ ←	$\beta_t \mathbf{k}_t \mathbf{v}_t^\top$

GH, non-diagonal:
token+channel mix

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)

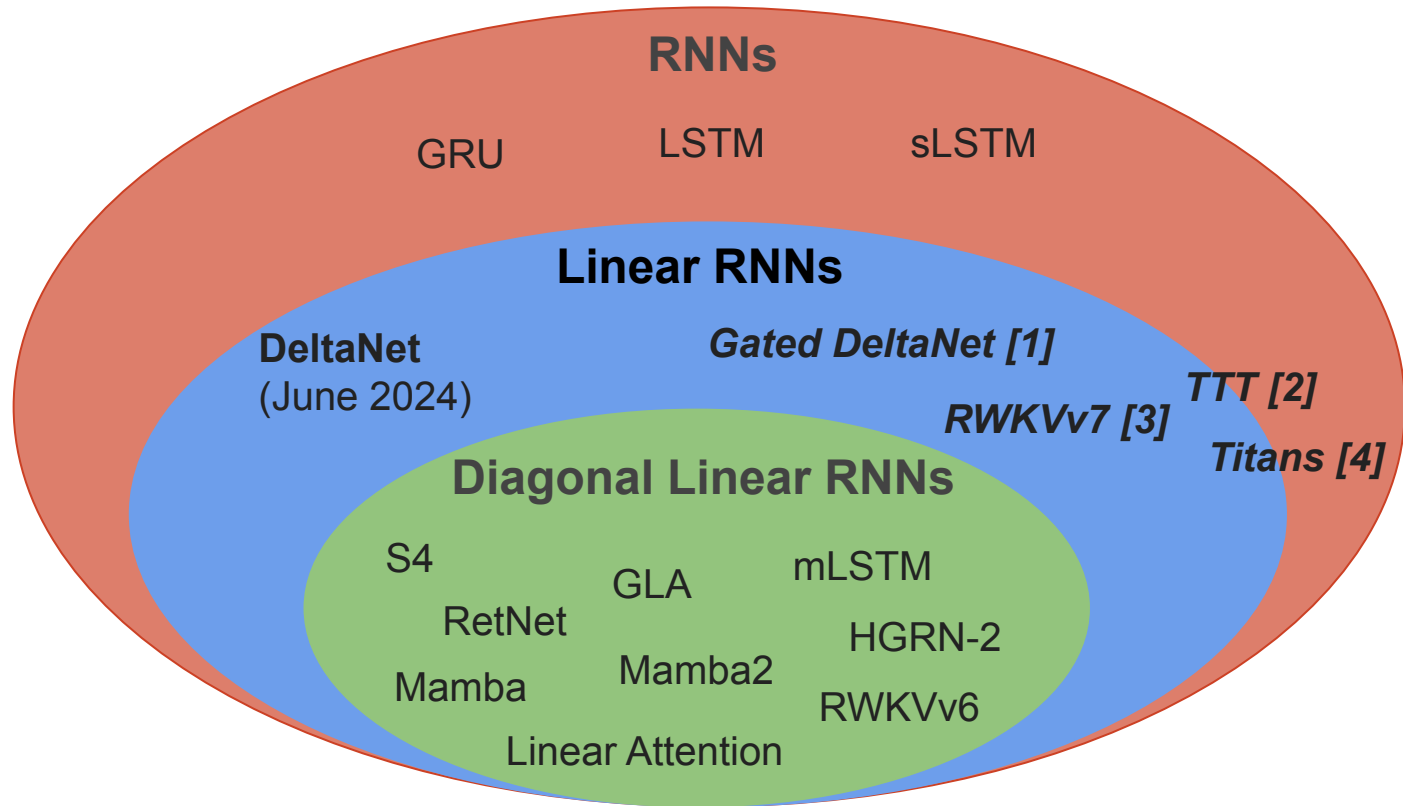
$$\mathbf{H}_i = \mathbf{A}(\mathbf{x}_i)\mathbf{H}_{i-1} + \mathbf{B}(\mathbf{x}_i), \quad \hat{\mathbf{y}}_i = \text{dec}(\mathbf{H}_i, \mathbf{x}_i)$$

State-transition matrix

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

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

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



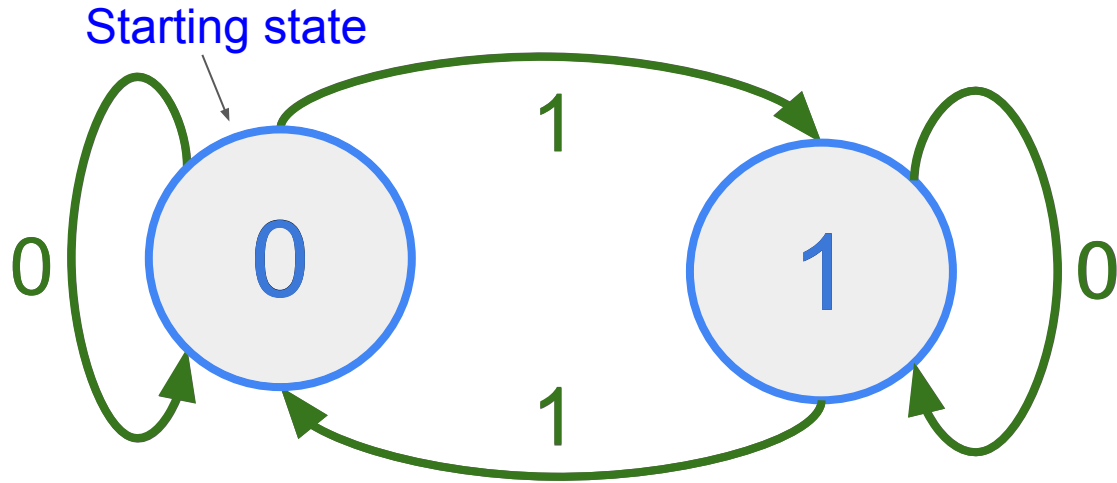
[1] Yang, Songlin, Jan Kautz, and Ali Hatamizadeh. "Gated Delta Networks: Improving Mamba2 with Delta Rule." *arXiv:2412.06464* (2024).

[2] Sun, Yu, et al. "Learning to (learn at test time): Rnns with expressive hidden states." *arXiv:2407.04620* (2024).

[3] <https://github.com/BlinkDL/RWKV-LM/tree/main/RWKV-v7>

[4] Behrouz, Ali, Peilin Zhong, and Vahab Mirrokni. "Titans: Learning to Memorize at Test Time." *arXiv:2501.00663* (2024).

Parity (2-cups game, addition modulo 2)



Input bits (transitions)

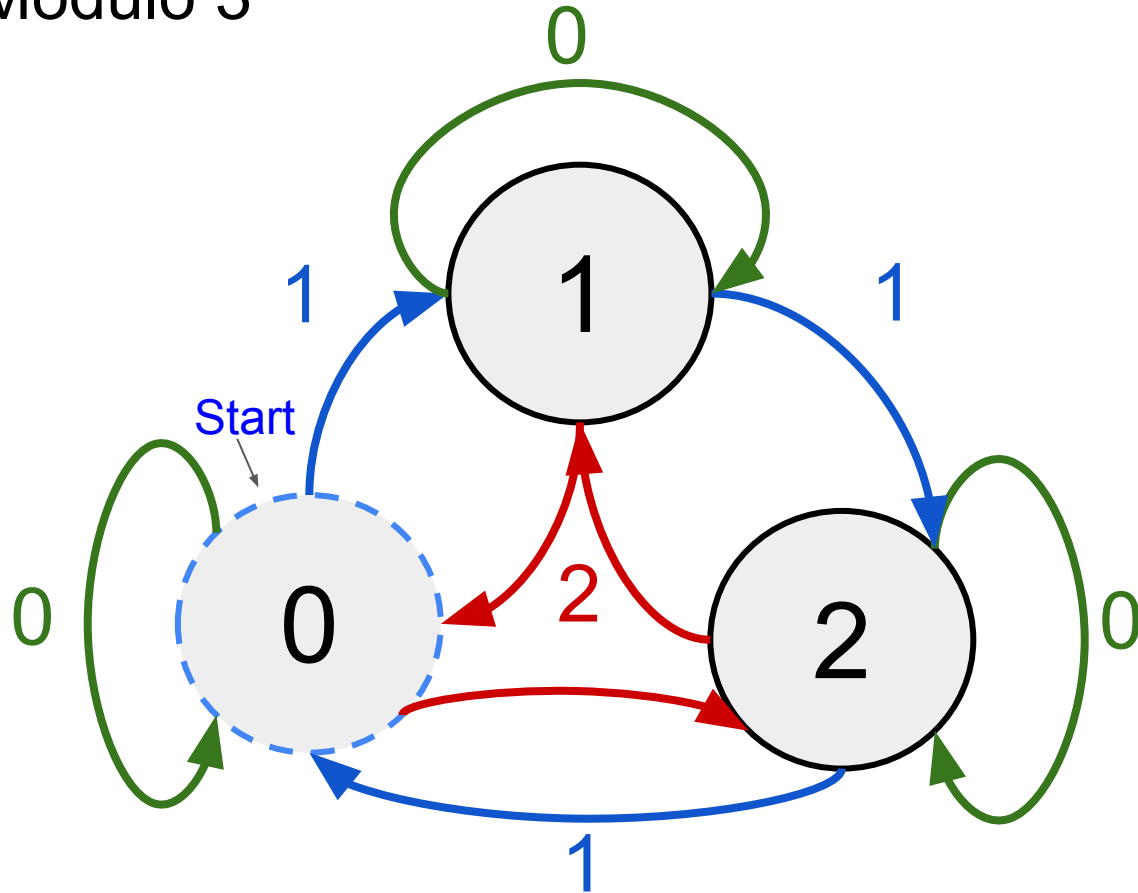
1	1	0	0	1	0	...
---	---	---	---	---	---	-----

Parity (states)

						...
--	--	--	--	--	--	-----

We'd like to generalize to arbitrarily long sequences in one forward pass

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

$$a(x_i) = 1$$

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

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

Solution 2: State = parity

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

Issue with Linear RNNs

	State-transition matrix $\rightarrow \mathbf{A}(\mathbf{x}_t)$	$\mathbf{B}(\mathbf{x}_t)$
Mamba	$\text{Diag}(\exp(-\Delta_t \odot \exp(\mathbf{w}_{1,i})))$	$k_{t,i} \Delta_t \odot \mathbf{x}_t$
GLA	$\text{Diag}(\alpha_t)$	$\mathbf{k}_t \mathbf{v}_t^\top$
DeltaNet	$\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top$	$\beta_t \mathbf{k}_t \mathbf{v}_t^\top$

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

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)

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

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

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 $i \in \{1, \dots, L\}$ the i -th layer satisfies

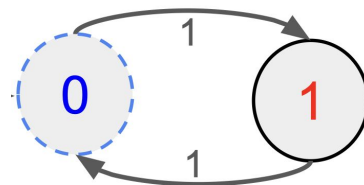
$$\lambda \in \mathbb{R} \quad \forall \lambda \in \text{eigs}(\mathbf{A}(\mathbf{x}_1) \cdots \mathbf{A}(\mathbf{x}_{2^{i-1}})) \quad \forall \mathbf{x}_1, \dots, \mathbf{x}_{2^{i-1}}$$

⇒ 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)

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



input $\mathbf{x} = \dots 1 1 1 1 1 1 1 1 1 1 1 1 \dots$

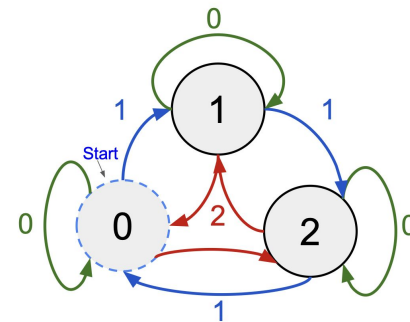
1st Layer Output (Finite Precision) $\bar{\mathbf{y}} = \dots \bar{y} \bar{y} \bar{y} \bar{y} \bar{y} \bar{y} \bar{y} \bar{y} \bar{y} \bar{y} \bar{y} \bar{y} \dots$

parity $\mathbf{y} = \dots 0 1 0 1 0 1 0 1 0 1 0 1 \dots$

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

Theorem 2 - Proof Idea

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



input $\mathbf{x} = \dots 1 1 1 1 1 1 1 1 1 1 1 1 \dots$

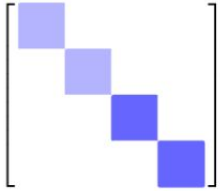
1st Layer Output (Finite Precision) $\bar{\mathbf{y}} = \dots \bar{y}_1 \bar{y}_2 \bar{y}_1 \bar{y}_2 \bar{y}_1 \bar{y}_2 \bar{y}_1 \bar{y}_2 \bar{y}_1 \bar{y}_2 \bar{y}_1 \bar{y}_2 \dots$

\parallel \parallel \parallel \parallel

add mod 3 $\mathbf{y} = \dots 0 1 2 0 1 2 0 1 2 0 1 2 \dots$

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

Trading off Expressivity and Computational Complexity



Diagonal

Very fast computation, but
can't go beyond parity

Trading off Expressivity and Computational Complexity

$$\mathbf{A}(\mathbf{x}_t) = \begin{bmatrix} \text{blue} & & & \\ & \text{blue} & & \\ & & \text{blue} & \\ & & & \text{blue} \end{bmatrix} + \begin{bmatrix} \text{red} \\ \text{red} \\ \text{red} \\ \text{red} \end{bmatrix} \times \begin{bmatrix} \text{pink} & \text{pink} & \text{pink} & \text{pink} \end{bmatrix}$$

Rank 1 Update => DeltaNet

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

$$\begin{bmatrix} \text{blue} & & & \\ & \text{blue} & & \\ & & \text{blue} & \\ & & & \text{blue} \end{bmatrix} + \begin{bmatrix} \text{red} & \text{purple} \\ \text{red} & \text{purple} \\ \text{red} & \text{purple} \\ \text{red} & \text{purple} \end{bmatrix} \times \begin{bmatrix} \text{pink} & \text{pink} & \text{pink} & \text{pink} \\ \text{orange} & \text{orange} & \text{orange} & \text{orange} \end{bmatrix}$$

Rank 2 Update => DeltaProduct

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

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

Stable recurrence!

Products of Generalized Householder (GH) Matrices

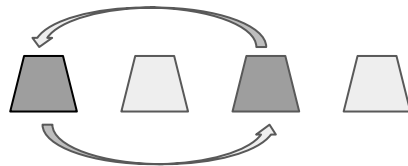
$$\mathcal{M}_k^n(\Omega) := \{C_1 C_2 \cdots C_k : C_i = \mathbf{I} - \beta_i \mathbf{v}_i \mathbf{v}_i^\top, \quad (1 - \beta_i) \in \Omega, \quad \mathbf{v}_i \in \mathbb{R}^n, \|\mathbf{v}_i\| = 1\}$$

↑
Eigenvalue range

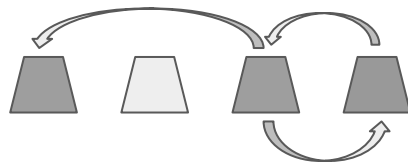
For DeltaNet, $k = 1, \Omega = [0, 1]$

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

$$\mathbf{I} - 2\mathbf{v}_1 \mathbf{v}_1^\top = \text{Reflection, e.g.}$$



$$(\mathbf{I} - 2\mathbf{v}_1 \mathbf{v}_1^\top)(\mathbf{I} - 2\mathbf{v}_2 \mathbf{v}_2^\top) = \text{2D Rotation, e.g.}$$



Contribution: Expressivity of Products of GH Matrices

Thm. 3 (Permutations): Finite precision linear RNNs with one layer where state-transition matrices are in $\mathcal{M}_{k-1}^n([-1, 1])$ can model any FSA whose transitions $\delta(\cdot, w) : Q \rightarrow Q$ correspond to 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.

\Rightarrow We can easily modify DeltaNet to have state transition matrices in $\mathcal{M}_1^n([-1, 1])$ 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 permutations of at most k elements.

Open question:

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

Eigenvalue Extension for Mamba and DeltaNet

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

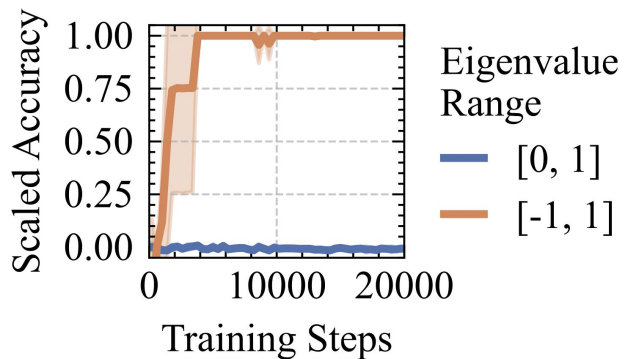
Change for DeltaNet is a *one-liner!*

```
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])
```

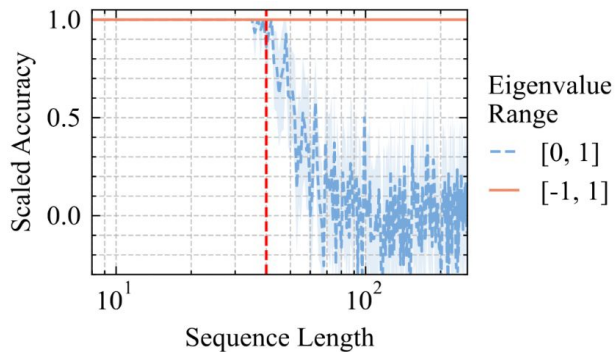
Code from [Flash Linear Attention](#) (Yang et al. 2024)

Experiments - Parity

→ 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

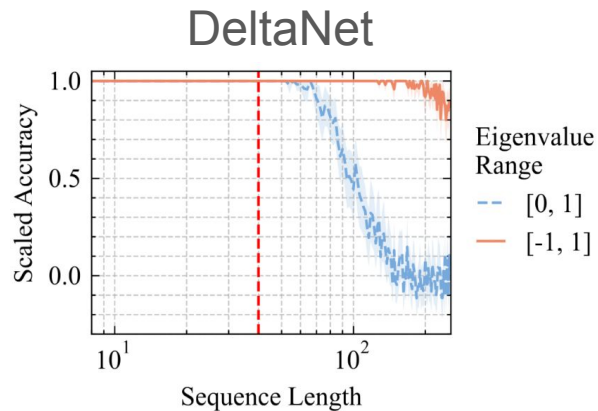


Experiments - Modular Arithmetic

Mod. Arithm. (w/o brackets):

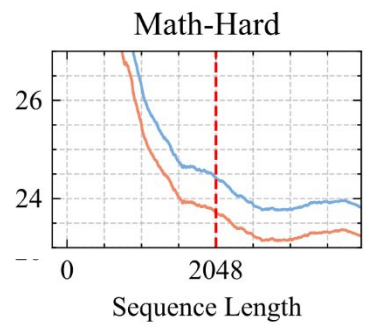
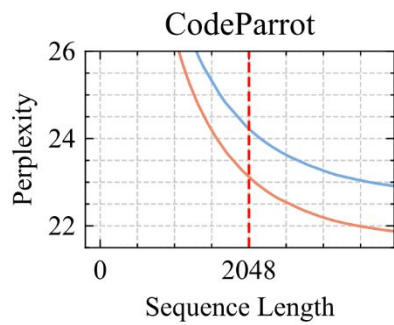
$$2 - 3 - 3 * 2 \bmod 5 = 3$$

	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



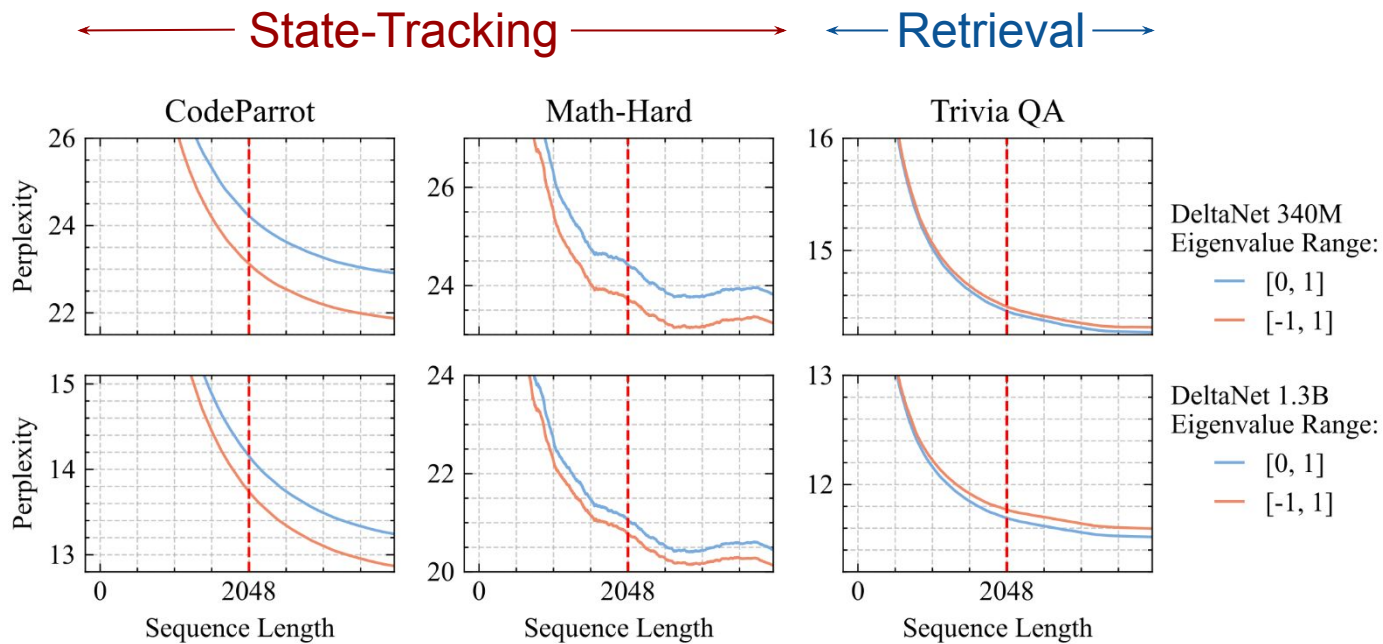
Experiments - Language Modelling

← State-Tracking → ← Retrieval →



DeltaNet 340M
Eigenvalue Range:
— [0, 1]
— [-1, 1]

Experiments - Language Modelling



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

Recalling from Alex's talk on Test-Time Regression

$$\mathcal{L}_t(\mathbf{H}) = \frac{1}{2} \|\mathbf{H}^\top \mathbf{k}_t - \mathbf{v}_t\|_2^2,$$

↓ Unrolling as
online-learning
and taking **1 step** of GD

$$\begin{aligned} \mathbf{H}_t &= \mathbf{H}_{t-1} - \beta_t \nabla \mathcal{L}_t(\mathbf{H}_{t-1}) \\ &= \mathbf{H}_{t-1} - \beta_t \mathbf{k}_t (\mathbf{k}_t^\top \mathbf{H}_{t-1} - \mathbf{v}_t^\top) \end{aligned}$$

DeltaNet: $\mathbf{H}_t = (\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top) \mathbf{H}_{t-1} + \beta_t \mathbf{k}_t \mathbf{v}_t^\top$

↳ What if we take *more* steps of GD?

Multiple Steps of GD on Associative Recall

Key idea: Take multiple steps of GD per token.

$$\text{DeltaNet: } \mathbf{x}_i \longrightarrow \begin{aligned} \mathbf{k}_i &= \mathbf{W} \mathbf{x}_i / \|\mathbf{W} \mathbf{x}_i\|_2 \\ \mathbf{v}_i &= \mathbf{V} \mathbf{x}_i \end{aligned} \longrightarrow \begin{aligned} \mathbf{H}_i &= \mathbf{H}_{i-1} - \beta_i \nabla \mathcal{L}_i(\mathbf{H}_{i-1}) \\ &= (\mathbf{I} - \beta_i \mathbf{k}_i \mathbf{k}_i^\top) \mathbf{H}_{i-1} + \beta_i \mathbf{k}_i \mathbf{v}_i^\top \end{aligned}$$

$$\text{DeltaProduct}_2: \mathbf{x}_i \longrightarrow \begin{aligned} \mathbf{k}_{i,1} &= \mathbf{W}_1 \mathbf{x}_i / \|\mathbf{W}_1 \mathbf{x}_i\|_2 \\ \mathbf{v}_{i,1} &= \mathbf{V}_1 \mathbf{x}_i \end{aligned} \quad \begin{aligned} \mathbf{H}_{i,1} &= \mathbf{H}_{i,0} - \beta_{i,1} \nabla \mathcal{L}_{i,1}(\mathbf{H}_{i,0}) \\ &= (\mathbf{I} - \beta_{i,1} \mathbf{k}_{i,1} \mathbf{k}_{i,1}^\top) \mathbf{H}_{i,0} + \beta_{i,1} \mathbf{k}_{i,1} \mathbf{v}_{i,1}^\top \end{aligned}$$

$$\begin{aligned} \mathbf{k}_{i,2} &= \mathbf{W}_2 \mathbf{x}_i / \|\mathbf{W}_2 \mathbf{x}_i\|_2 \\ \mathbf{v}_{i,2} &= \mathbf{V}_2 \mathbf{x}_i \end{aligned} \quad \begin{aligned} \mathbf{H}_{i,2} &= \mathbf{H}_{i,1} - \beta_{i,2} \nabla \mathcal{L}_{i,2}(\mathbf{H}_{i,1}) \\ &= (\mathbf{I} - \beta_{i,2} \mathbf{k}_{i,2} \mathbf{k}_{i,2}^\top) \mathbf{H}_{i,1} + \beta_{i,2} \mathbf{k}_{i,2} \mathbf{v}_{i,2}^\top \end{aligned}$$

From Recurrence to Products of Householders

$$\mathbf{H}_{i,1} = (\mathbf{I} - \beta_{i,1} \mathbf{k}_{i,1} \mathbf{k}_{i,1}^\top) \mathbf{H}_{i,0} + \beta_{i,1} \mathbf{k}_{i,1} \mathbf{v}_{i,1}^\top$$

$$\mathbf{H}_{i,2} = (\mathbf{I} - \beta_{i,2} \mathbf{k}_{i,2} \mathbf{k}_{i,2}^\top) \mathbf{H}_{i,1} + \beta_{i,2} \mathbf{k}_{i,2} \mathbf{v}_{i,1}^\top$$

Products of
Householders

$$\mathbf{H}_{i,2} = (\mathbf{I} - \beta_{i,2} \mathbf{k}_{i,2} \mathbf{k}_{i,2}^\top) (\mathbf{I} - \beta_{i,1} \mathbf{k}_{i,1} \mathbf{k}_{i,1}^\top) \mathbf{H}_{i,0} + (\mathbf{I} - \beta_{i,2} \mathbf{k}_{i,2} \mathbf{k}_{i,2}^\top) \beta_{i,1} \mathbf{k}_{i,1} \mathbf{v}_{i,1}^\top + \beta_{i,2} \mathbf{k}_{i,2} \mathbf{v}_{i,2}^\top$$

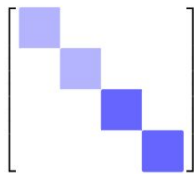
Generalizing for n_h GD steps

DeltaProduct $_{n_h}$

$$\mathbf{H}_i = \mathbf{A}(\mathbf{x}_i) \mathbf{H}_{i-1} + \mathbf{B}(\mathbf{x}_i)$$

$$\mathbf{A}(\mathbf{x}_i) = \prod_{j=1}^{n_h} (\mathbf{I} - \beta_{i,j} \mathbf{k}_{i,j} \mathbf{k}_{i,j}^\top) \quad \mathbf{B}(\mathbf{x}_i) = \sum_{j=1}^{n_h} \left(\prod_{k=j+1}^{n_h} (\mathbf{I} - \beta_{i,k} \mathbf{k}_{i,k} \mathbf{k}_{i,k}^\top) \right) \beta_{i,j} \mathbf{k}_{i,j} \mathbf{v}_{i,j}^\top$$

Comparison to other models



Diagonal:

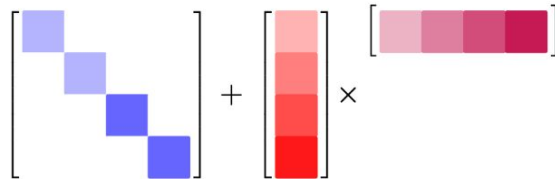
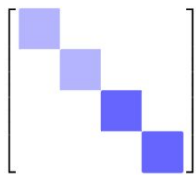
Token Mix: ✓

Channel Mix: ×

Expressivity: Parity

Examples: Mamba, GLA

Comparison to other models



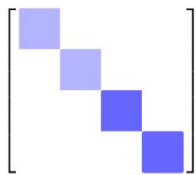
Diagonal:

Token Mix: ✓
Channel Mix: ×
Expressivity: Parity
Examples: Mamba, GLA

Rank 1 Update:

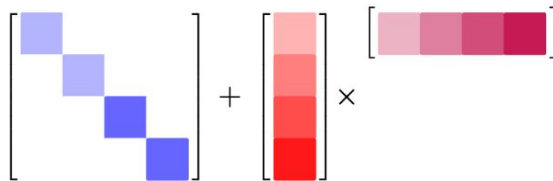
✓
✓
Reflections
DeltaNet, TTT, RWKV-7

Comparison to other models



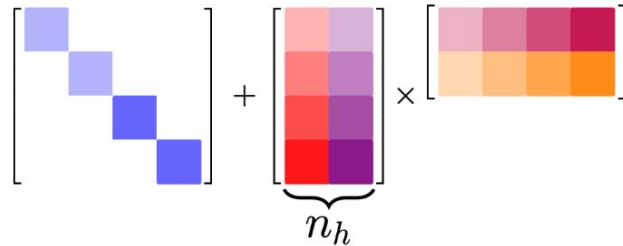
Diagonal:

Token Mix: ✓
Channel Mix: ×
Expressivity: Parity
Examples: Mamba, GLA



Rank 1 Update:

✓
 ✓
 Reflections
 DeltaNet, TTT, RWKV-7



Rank n_h Update:

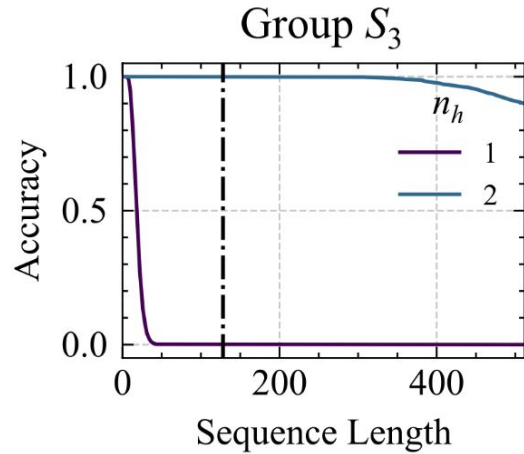
✓
 ✓
 Rotations
DeltaProduct

Experiments - State-Tracking

S_5 (Permutation group of 5 elements)

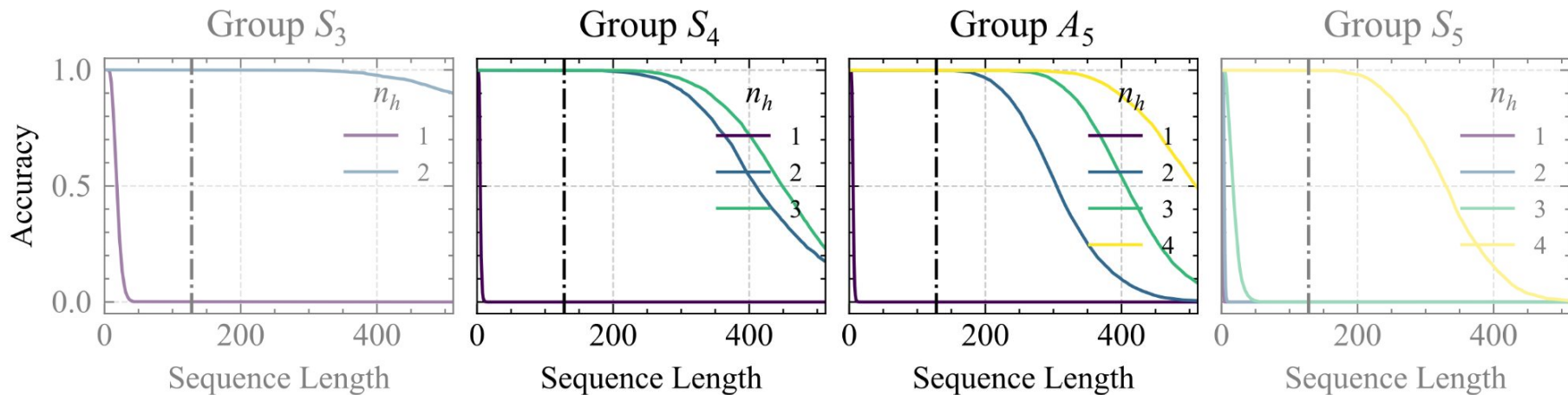
$$(1, 2, 3, 4, 5) \circ (1 \rightarrow 2, 2 \rightarrow 4, 3 \rightarrow 5, 4 \rightarrow 3, 5 \rightarrow 1) = (5, 1, 4, 2, 3)$$

Experiments - State-Tracking (Single layer)



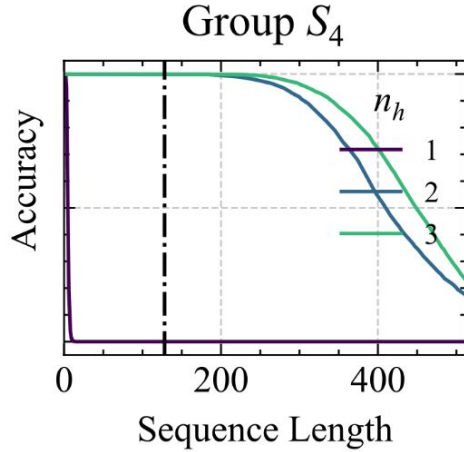
→ Predicted by Theorem 3, for groups S_3 and S_5 , 2 and 4 householders are required

Experiments - State-Tracking (Single layer)

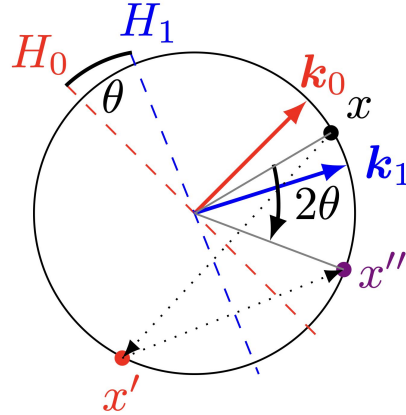


→ Surprisingly, only 2 householders are necessary for S_4 and A_5 , why?

Experiments - State-Tracking (Single layer)



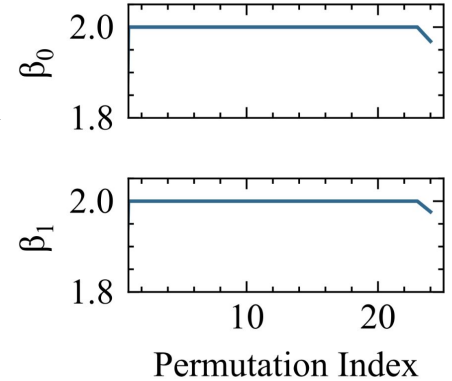
Isomorphic to
subgroups of
 $SO(3, \mathbb{R})$



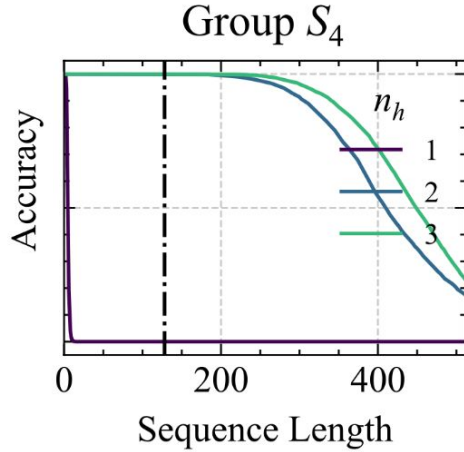
Can we do rotations
using 2 householders?

$$(\mathbf{I} - \beta_0 \mathbf{k}_0 \mathbf{k}_0^\top) (\mathbf{I} - \beta_1 \mathbf{k}_1 \mathbf{k}_1^\top)$$

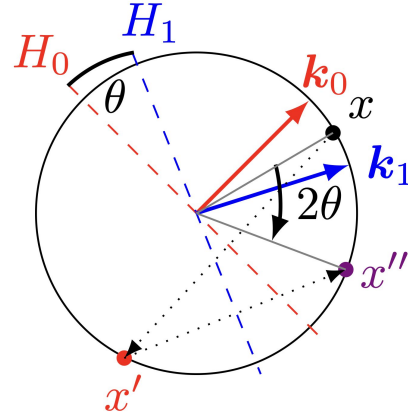
Are the householders
reflections (i.e. $\beta = 2$)?



Experiments - State-Tracking (Single layer)



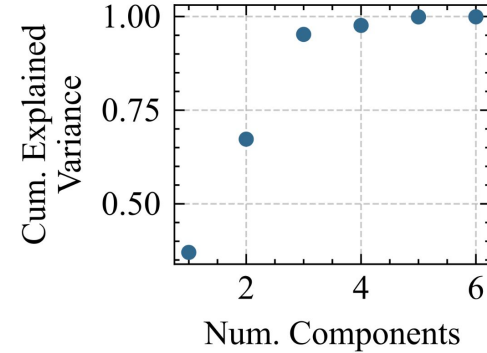
Isomorphic to
subgroups of
 $SO(3, \mathbb{R})$



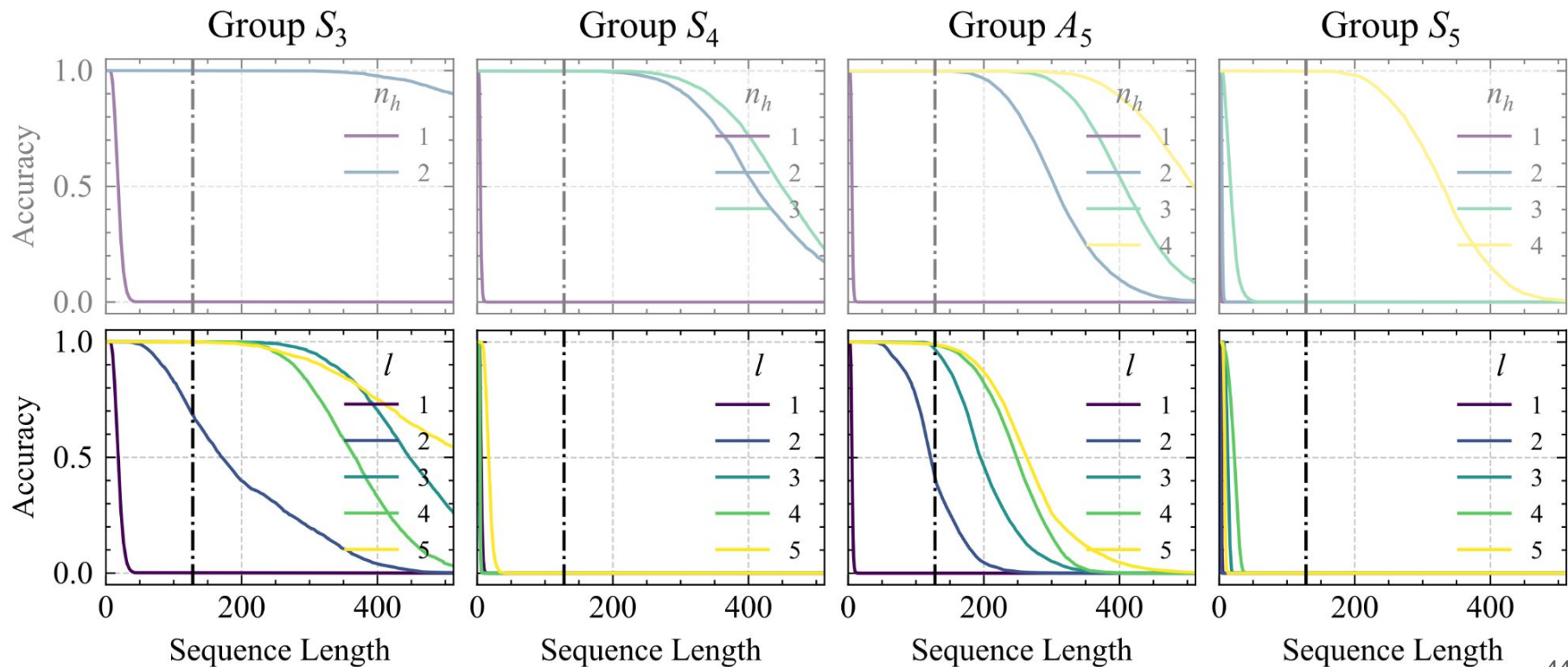
Can we do rotations
using 2 householders?

$$(\mathbf{I} - \beta_0 \mathbf{k}_0 \mathbf{k}_0^\top) (\mathbf{I} - \beta_1 \mathbf{k}_1 \mathbf{k}_1^\top)$$

Are the keys in a 3D
subspace?



Experiments - State-Tracking (Multiple layers)

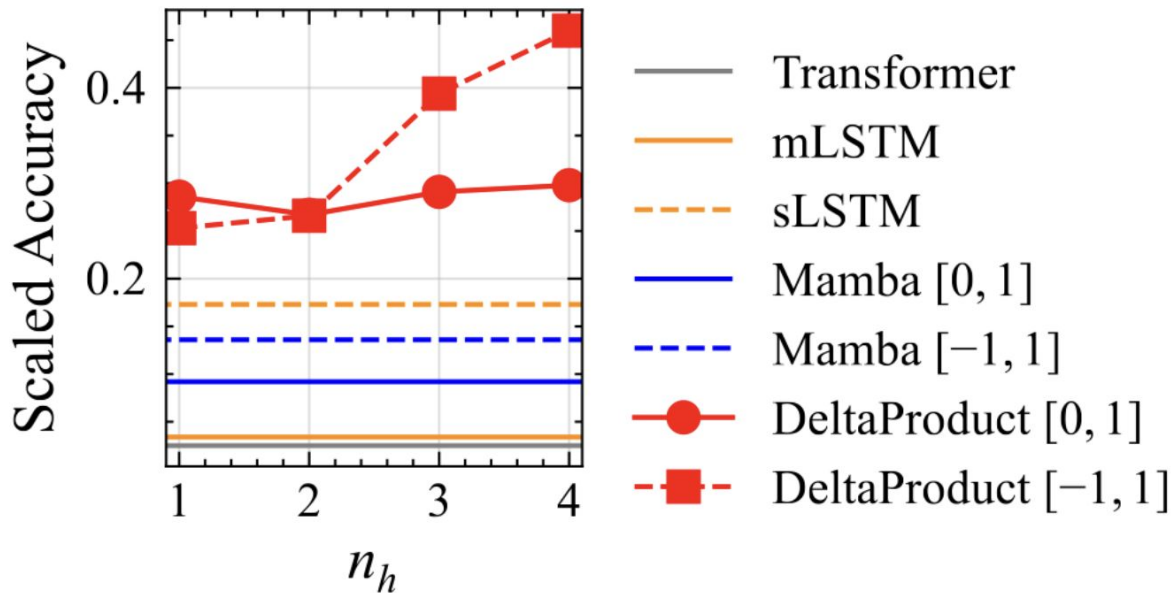


Experiments - Chomsky Hierarchy

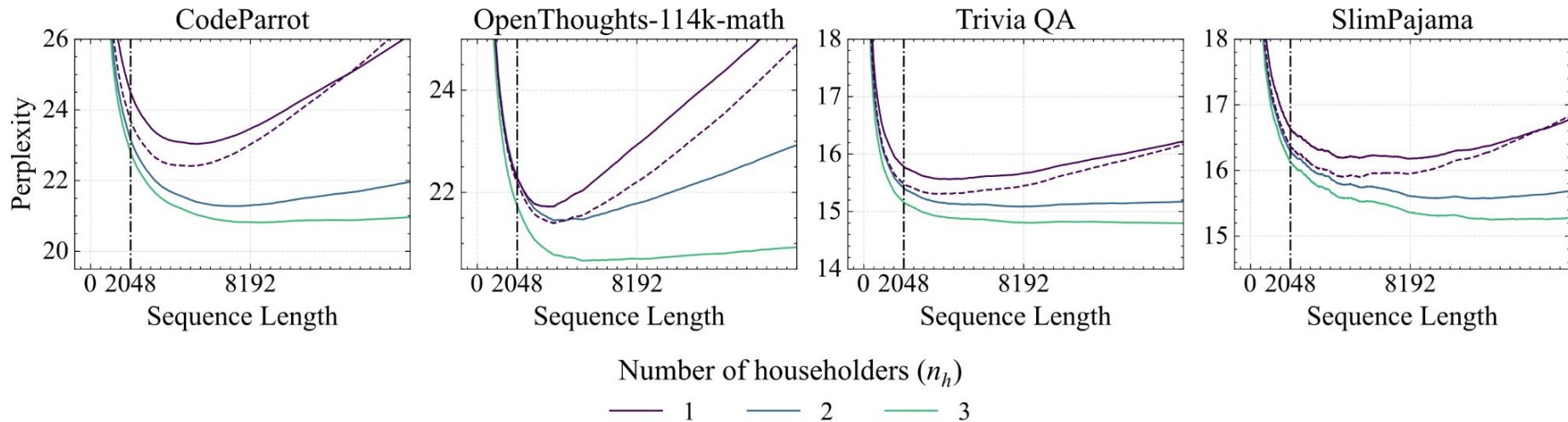
Mod. Arithm.
(w/ brackets):

Example:

$$(((3-2)*2) - (2*3) \bmod 5 = 1$$



Experiments - Language Modeling

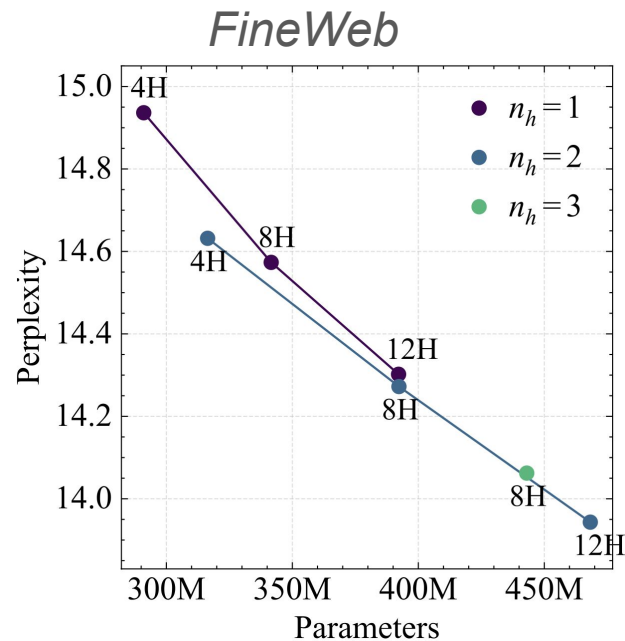


→ Length extrapolation improves significantly with more householders
Hypothesis: Faster forgetting mechanism

Experiments - Language Modeling

Increasing n_h , increases parameter count.

Is DeltaProduct still competitive when accounting for the increase in parameters?



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 RWKV-7 are promising due to their superior expressivity compared to Mamba.
- *DeltaProduct* leverages multiple steps of gradient descent, leading to higher order updates to the hidden state, represented as products of householders.

Future Directions:

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

Thank you!