
Unlocking State-Tracking in Linear RNNs Through Negative Eigenvalues

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 Linear Recurrent Neural Networks (LRNNs) such as Mamba, RWKV, GLA, mL-
2 STM, and DeltaNet have emerged as efficient alternatives to transformers in large
3 language modeling, offering linear scaling with sequence length and improved
4 training efficiency. However, LRNNs struggle with state-tracking which is impor-
5 tant for e.g. code comprehension and tracking chess pieces across a board. Even
6 parity, the simplest state-tracking task, which non-linear RNNs like LSTMs han-
7 dle effectively, cannot be solved by current LRNNs. Recently, Sarrof et al. (2024)
8 demonstrated that the failure of LRNNs to solve parity stems from restricting the
9 eigenvalue range of their diagonal state-transition matrices to $[0, 1]$, and that incor-
10 porating negative eigenvalues can resolve this issue. We generalize this result to
11 full matrix LRNNs. We prove that no finite-precision LRNN with state-transition
12 matrices having only positive real eigenvalues can solve parity. Notably, we prove
13 that LRNNs can learn any regular language when their state-transition matrices
14 are products of identity plus vector outer product matrices with eigenvalues in the
15 range $[-1, 1]$. Our empirical results confirm that extending the eigenvalue range
16 of models like Mamba and DeltaNet to include negative values not only enables
17 them to solve parity but consistently improves their performance on state-tracking
18 tasks. Furthermore, pre-training LRNNs with an extended eigenvalue range for
19 language modeling achieves comparable performance and stability while showing
20 promise for coding tasks.

21 1 Introduction

22 Transformer architectures (Vaswani et al., 2017) have revolu-
23 tionized NLP but scale quadratically in sequence length, posing
24 computational challenges for long sequences. To address this,
25 Linear Recurrent Neural Networks (LRNNs) have emerged as
26 promising alternatives that offer linear scaling while maintaining
27 competitive performance (Gu & Dao, 2023; Dao & Gu, 2024;
28 Yang et al., 2024a; Peng et al., 2023; Deletang et al., 2023; Sun
29 et al., 2024; Beck et al., 2024). LRNNs update their state via
30 matrix-vector products with structured, learnable state-transition
31 matrices, enabling parallelization techniques like the associa-
32 tive scan (Blelloch, 1990) for efficient training. The structure
33 of state-transition matrices largely determines LRNN capabili-
34 ties. While successful models like Mamba (Gu & Dao, 2023),
35 GLA (Yang et al., 2024a), and mLSTM (Beck et al., 2024) use
36 diagonal matrices (diagonal LRNN), recent work explores more
37 complex forms. Notably, non-diagonal matrices using general-

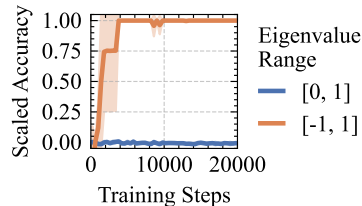


Figure 1: Extending the eigenvalue range of the state transition matrices of diagonal LRNNs improves performance from random guessing (range $[0, 1]$), to perfect score (range $[-1, 1]$) on learning parity. Trained on sequences up to length 40; Tested on lengths 40–256 (3 seeds).

38 ized Householder (GH) transformations, defined as $\mathbf{I} + \mathbf{u}\mathbf{u}^\top$ where \mathbf{u} is a learnable vector and \mathbf{I} is
 39 the identity, show promise in models like DeltaNet (Yang et al., 2024b) and TTT-Linear (Sun et al.,
 40 2024), potentially enhancing expressiveness while preserving efficiency.

41 Despite these successes, both transformers and current LRNNs face a fundamental limitation: they
 42 struggle to learn and track the state of even simple finite-state machines from sequences of state-
 43 transitions (Deletang et al., 2023). This limitation may impair performance on tasks such as entity
 44 tracking in narratives, handling nested structures in code, and other reasoning tasks that can benefit
 45 from maintaining and updating an internal state over time (Merrill et al., 2024). Even the simplest
 46 state-tracking task, computing the parity of a sequence of bits, cannot be solved by current LRNNs,
 47 while non-linear RNNs like LSTMs (Hochreiter & Schmidhuber, 1997) and sLSTM (Beck et al.,
 48 2024) can solve parity easily (Merrill et al., 2024). However, in-contrast to modern linear RNNs,
 49 non-linear RNNs lack an efficient method for parallelizing the training across sequence length.

50 Recently, Sarrof et al. (2024) demonstrated that the inability of diagonal LRNNs to solve parity
 51 problems stems from the constraining the eigenvalues of their state-transition matrices to be posi-
 52 tive. Specifically, they proved that diagonal LRNNs, when implemented with finite precision and
 53 exclusively positive real eigenvalues, cannot solve the *parity* problem for sequences of arbitrary
 54 length. However, their work did not provide empirical evidence showing that diagonal LRNNs with
 55 negative eigenvalues can be successfully trained to overcome this limitation. We prove that the same
 56 limitation also affects LRNNs with non-diagonal state-transition matrices, and further prove that
 57 complex eigenvalues are necessary to solve the more challenging task of modular counting when
 58 the modulus is a power of two. Our findings also apply to the GH matrices employed by DeltaNet,
 59 as they share the same eigenvalue limitations. To overcome this, we propose a simple yet powerful
 60 solution: extend the range of possible eigenvalues from $[0, 1]$ to $[-1, 1]$. This change enables state-
 61 tracking and significantly improves LRNNs’ expressivity without compromising their efficiency and
 62 training stability. We illustrate in Figure 1 that this change allows diagonal LRNNs to learn parity.
 63 We open-source the code for our experiments in this anonymous repository.

64 **Contributions:**

- 65 (i) We prove that any finite-precision LRNN with only positive real eigenvalues in the state-
 66 transition matrices (most LRNNs used in practice) cannot solve parity at arbitrary sequence lengths
 67 (Theorem 1), while complex eigenvalues are required to learn counting modulo 3 (Theorem 2).
- 68 (ii) By extending the eigenvalue range to $[-1, 1]$, we significantly improve LRNN state-tracking
 69 capabilities. We prove that LRNNs with state-transition matrices formed by products of generalized
 70 Householder (GH) matrices in the range $[-1, 1]$ can learn any regular language (Theorem 4), in
 71 some cases with just a single layer (Theorem 3). Importantly, such an extension allows even practical
 72 LRNNs, using just one GH matrix (like DeltaNet), to learn substantially harder tasks, such as
 73 the composition of permutations of two elements (swaps), compared to diagonal LRNNs.
- 74 (iii) We show that the eigenvalue range of Mamba and DeltaNet can be extended to $[-1, 1]$ with-
 75 out compromising efficiency or training stability. We test the modified methods on parity, modular
 76 arithmetic, and permutation composition, demonstrating improved state-tracking performance.
- 77 (iv) We pre-train modified versions of DeltaNet (340M parameters) and Mamba (370M parameters)
 78 and show that they reach performance comparable to the original models on generative language
 79 modeling tasks, while DeltaNet shows improved perplexity on coding and math datasets. Notably,
 80 we gain 2 points of perplexity on CodeParrot (Tunstall et al., 2022).

81 **2 Background: Linear Recurrent Neural Networks (LRNNs)**

82 We describe LRNNs using notation inspired by Sarrof et al. (2024), focusing on the core linear
 83 recurrences while abstracting away non-linear computations for each token. LRNNs are, in fact,
 84 stacks of layers with common structure but distinct learnable parameters. For further related work
 85 on LRNNs, we refer to Appendix A. Each layer takes input vectors $\mathbf{x}_1, \dots, \mathbf{x}_t \in \mathbb{R}^l$ and outputs
 86 $\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_t \in \mathbb{R}^p$ as:

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

$$\mathbf{H}_0 \in \mathbb{C}^{n \times d}, \quad \mathbf{A} : \mathbb{R}^l \rightarrow \mathbb{C}^{n \times n}, \quad \mathbf{B} : \mathbb{R}^l \rightarrow \mathbb{C}^{n \times d}, \quad \text{dec} : \mathbb{C}^{n \times d} \times \mathbb{R}^l \rightarrow \mathbb{R}^p \quad (1)$$

87 Here, \mathbf{A} , \mathbf{B} and dec are learnable, generally non-linear functions, with dec usually expressed by a
 88 feed-forward neural network. This definition encompasses most LRNN variants, which differ in the

	$A(x_t)$	$B(x_t)$	\hat{y}_t
Mamba	$\text{Diag}(\exp(-\Delta_{t,i} \exp(w_{1,i})))$	$\Delta_{t,i} x_{t,i} \mathbf{k}_t$	$\psi(\mathbf{q}_t^\top \mathbf{H}_t^\top + \mathbf{w}_2 \odot \mathbf{x}_t)$
GLA	$\text{Diag}(\alpha_t)$	$\mathbf{k}_t \mathbf{v}_t^\top$	$\psi(\mathbf{q}_t^\top \mathbf{H}_t^\top)$
DeltaNet	$\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top$	$\mathbf{k}_t \mathbf{v}_t^\top$	$\psi(\mathbf{q}_t^\top \mathbf{H}_t^\top)$

Table 1: Different LRNNs layers as instances of (1), where $\alpha_t = \text{sigmoid}(\mathbf{W}_\alpha \mathbf{x}_t)$, $\Delta_t = \text{softplus}(\mathbf{W}_\Delta \mathbf{x}_t)$, $\beta_t = \text{sigmoid}(\mathbf{w}_\beta \mathbf{x}_t)$, while $\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t$ are the output of learnable and possibly non-linear functions of \mathbf{x}_t . Also $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is another learnable function usually containing an MLP and a normalization, while $\mathbf{W}_1 \in \mathbb{R}^d$, $\mathbf{W}_\Delta \in \mathbb{R}^{d \times l}$, $\mathbf{W}_\alpha \in \mathbb{R}^{n \times l}$, $\mathbf{w}_\beta \in \mathbb{R}^l$ and $\mathbf{w}_2 \in \mathbb{R}^d$ are learnable parameters. For simplicity, for Mamba, we wrote the matrices for the recursion of each row of the state \mathbf{H}_t and set $\mathbf{x}_t = (x_{t,1}, \dots, x_{t,d})$, $\mathbf{W}_1 = (\mathbf{w}_{1,1}, \dots, \mathbf{w}_{1,d})$ and $\Delta_t = (\Delta_{t,1}, \dots, \Delta_{t,d})$.

89 form of \mathbf{A} and \mathbf{B} , dec parameterization, and use of positional encoding. Table 1 illustrates how
90 three popular LRNNs fit this framework. For other architectures, see (Yang et al., 2024b, Table 4).

91 The *state-transition matrices* $\mathbf{A}(x_t)$ are typically diagonal or generalized Householder (GH), i.e.,
92 identity plus vector outer product, as shown in Table 1, to enable efficient matrix-vector products on
93 modern hardware. These matrices consistently have eigenvalues (and norm) in the range $[0, 1]$.

94 3 Theoretical Analysis

95 We begin by highlighting the limitations of current LRNNs, demonstrating that they fail to meet
96 a necessary condition for solving parity and modular counting problems: the eigenvalues of their
97 state-transition matrices are restricted to the range $[0, 1]$. Subsequently, we illustrate how extending
98 this eigenvalue range to $[-1, 1]$ significantly enhances the expressive power of LRNNs.

99 3.1 Limitations of Current LRNNs

100 The parity $y_t \in \{0, 1\}$ of a sequence of ones and zeros $x_1 \dots x_t \in \{0, 1\}^t$ is 1 if the total num-
101 ber of ones in the sequence is odd, and 0 if it’s even. Equivalent to addition modulo 2, it can be
102 computed by summing the values in the input sequence and then applying the modulo 2 function:
103 $y_t = (\sum_{i=1}^t x_i) \bmod 2$. We can also express this as the linear recursion

$$h_t = h_{t-1} + x_t, \quad h_0 = 0, \quad y_t = h_t \bmod 2 \quad (2)$$

104 where h_t contains the total number of ones. This solution can be implemented by an LRNN with one
105 layer and scalar states by setting $\mathbf{A}(x_t) = 1$, $\mathbf{B}(x_t) = x_t$, $\mathbf{H}_0 = 0$, and $\text{dec}(\mathbf{H}_t, x_t) = \mathbf{H}_t \bmod 2$
106 in Equation (1). However, implementing such a solution with finite precision presents an issue: the
107 state h_t can grow indefinitely with t , eventually reaching the limit of our precision range. Indeed,
108 $h_t \in \{0, \dots, t\}$, requiring $\log_2(t + 1)$ bits for storage. Such solutions, referred to as *shortcut*
109 *solutions*, are the only ones learnable by transformers when allowing $O(\log(t))$ bits of precision and
110 either depth $O(\log(t))$ or width $O(t)$ (Liu et al., 2023). Moreover, the MLP in dec must approximate
111 the modulus 2 function, which is challenging to learn due to its discontinuous and periodic nature.
112 A more efficient solution, which implements the two-state FSA solving this problem, can still be
113 realized by a finite-precision LRNN with one layer and scalar states (and consequently with vector
114 states and diagonal state-transition matrices) using the recursion:

$$h_t = a(x_t)h_{t-1} + b(x_t), \quad h_0 = 0, \quad b(1) = a(0) = 1, a(1) = -1, \quad y_t = h_t. \quad (3)$$

115 Note that the state-transition scalar $a(1)$ is negative. However, current diagonal LRNNs do not allow
116 negative values, and so are unable to learn parity. This raises the question: can non-diagonal LRNNs,
117 such as DeltaNet, solve parity? The following result gives an answer to this question by providing
118 necessary condition for a LRNN to solve parity. It generalizes (Sarrof et al., 2024, Theorem 2) to
119 non-diagonal matrices, showing that there must be at least one eigenvalue which is not real and
120 positive. This eigenvalue could simply have a nonzero imaginary part without necessarily being
121 negative and real.

122 **Theorem 1** (Parity). *A finite-precision LRNN with finitely many layers of the form (1) can solve*
123 *parity for arbitrary input lengths, in particular it can recognize the language $(11)^*$, only if at every*
124 *layer, $\mathbf{A}(x_t)$ admits at least one eigenvalue λ with $|\lambda| \geq 1$ and that is not real and positive.*

125 Notice that Mamba, mLSTM, GLA and even non-diagonal LRNNs such as DeltaNet do not satisfy
 126 such requirement. The proof in Appendix C.1 uses the same core idea in the one of (Sarraf et al.,
 127 2024, Theorem 2). For one layer, we show that when $\mathbf{x} = 1^k$ and the conditions for the eigenvalues
 128 of $\mathbf{A}(1)$ are not met, each element of state \mathbf{H}_k in finite precision will be constant for large enough
 129 k . Thus, $\hat{\mathbf{y}}_k$ cannot be equal to \mathbf{y}_k (for k large enough) no matter the choice of dec. To show this, we
 130 use the expression for the powers of the Jordan Canonical form of $\mathbf{A}(1)$, to prove that each element
 131 of $\mathbf{A}(1)^k$ either converges or diverges to a point in the complex infinity when $k \rightarrow \infty$.

132 We now study the problem of counting modulo m , which can be seen as an easier version of addition
 133 modulo m . For this problem the input of length k never changes and is equal to $\mathbf{x} = 1^k$, while
 134 the correct output is $y_k = \sum_{i=1}^k x_i \bmod m$. The following theorem establishes that to solve this
 135 problem, the state-transition matrices of the LRNN must have at least one eigenvalue with a nonzero
 136 imaginary part and modulus greater than or equal to one.

137 **Theorem 2** (Modular Counting). *A finite-precision LRNN with finitely many layers can solve mod-*
 138 *ular counting with modulus greater than 2 for arbitrary input lengths, in particular it can recognize*
 139 *the language $(1^m)^*$ with m not a power of two, only if at every layer $\mathbf{A}(\mathbf{x}_t)$ admits at least one*
 140 *eigenvalue λ with nonzero imaginary part and such that $|\lambda| \geq 1$.*

141 Note that all LRNNs allowing only symmetric or triangular state-transition matrices with real entries
 142 do not satisfy the assumptions of Theorem 2. For one layer, the proof in Appendix C.2 is similar
 143 that of Theorem 1 but we consider the two sequences \mathbf{H}_{2k} and \mathbf{H}_{2k+1} , showing that they have
 144 a defined limit when $k \rightarrow \infty$, even when $\mathbf{A}(1)$ admits negative eigenvalues less or equal than -1 .
 145 Theorems 1 and 2 identify a fundamental limitation of current design choices on the structure of the
 146 state-transition matrices of LRNNs. Specifically, the most popular approaches outlined in Table 1
 147 are incapable of solving parity problems, as the eigenvalues of their state-transition matrices are
 148 confined to the interval $[0, 1]$. Further, even if we allow negative eigenvalues that are still real,
 149 we cannot solve counting modulus m . However, as we will see in the next section, LRNNs with
 150 state-transition matrices that are products of generalized Householder (GH) matrices, each with
 151 eigenvalues in $[-1, 1]$, are more powerful than LRNNs with diagonal state-transition matrices.

152 3.2 Allowing Negative Eigenvalues

153 In this section, we explore the implications of extending the eigenvalue range to include negative
 154 values. We focus on two classes of LRNNs: those with diagonal state-transition matrices, which are
 155 currently the most prevalent (including GLA, Mamba, and Mamba2), and those with generalized
 156 Householder (GH) state-transition matrices, as recently proposed in the DeltaNet architecture. In
 157 particular, if we let $\mathbf{s} : \mathbb{R}^l \rightarrow [0, 1]^n$ and $\phi : \mathbb{R}^l \rightarrow [0, 1]$, $\mathbf{v} : \mathbb{R}^l \rightarrow \mathbb{R}^n$, being learnable functions
 158 such that $\|\mathbf{v}(\mathbf{x})\| = 1$ for every $\mathbf{x} \in \mathbb{R}^l$, then the state transition matrices of each layer of many
 159 LRNNs, such as those in Table 1, can be written as either

$$\mathbf{A}_{\text{diag}}(\mathbf{x}) := \text{Diag}(\mathbf{s}(\mathbf{x})), \quad \text{or} \quad \mathbf{A}_{\text{GH}}(\mathbf{x}) := \mathbf{I} - \phi(\mathbf{x})\mathbf{v}(\mathbf{x})\mathbf{v}(\mathbf{x})^\top,$$

160 where $\mathbf{A}_{\text{diag}}(\mathbf{x})$ is diagonal and has every eigenvalue $s(\mathbf{x})_i \in [0, 1]$ and $\mathbf{A}_{\text{GH}}(\mathbf{x})$ is GH and has
 161 all eigenvalues equal to one except for the one associated to the eigenvector $\mathbf{v}(\mathbf{x})$, which is equal
 162 to $1 - \phi(\mathbf{x}) \in [0, 1]$. To address the limitations discussed in the previous section, we propose the
 163 following modification that can be easily applied to any LRNN belonging to either class.

$$\mathbf{A}_{\text{diag}}^-(\mathbf{x}) := \text{Diag}(2\mathbf{s}(\mathbf{x}) - \mathbf{1}), \quad \mathbf{A}_{\text{GH}}^-(\mathbf{x}) := \mathbf{I} - 2\phi(\mathbf{x})\mathbf{v}(\mathbf{x})\mathbf{v}(\mathbf{x})^\top. \quad (4)$$

164 Note that $\mathbf{A}_{\text{diag}}^-(\mathbf{x})$ has eigenvalues $2s(\mathbf{x})_i - 1 \in [-1, 1]$ and $\mathbf{A}_{\text{GH}}^-(\mathbf{x})$ has all eigenvalues equal
 165 to one, except for one that is equal to $1 - 2\phi(\mathbf{x}) \in [-1, 1]$. Thus, we have extended the range of
 166 eigenvalues from $[0, 1]$ to $[-1, 1]$. We know from the previous section, that LRNNs with the modified
 167 state transition matrices can implement the solution to the parity problem by setting $s(1) = 0$ and
 168 $\phi(1) = 1$ so that if we consider a scalar recursion, then $\mathbf{A}_{\text{diag}}^-(1) = \mathbf{A}_{\text{GH}}^-(1) = -1$. However,
 169 we have also shown that since the eigenvalues are real, we cannot solve counting modulo m with
 170 $m \geq 3$. Despite this, we note that counting modulo m is linked to rotation by an angle of $2\pi/m$
 171 radians in \mathbb{R}^2 , and we can express a rotation as a product of two reflections, each of which can
 172 be written as a GH matrix. In other words, for any integer $m \geq 3$ there exist unit norm vectors
 173 $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^2$ such that

$$\mathbf{R}(\theta) := \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} = (\mathbf{I} - 2\mathbf{v}_1\mathbf{v}_1^\top) (\mathbf{I} - 2\mathbf{v}_2\mathbf{v}_2^\top), \quad \theta = \frac{2\pi}{m}.$$

174 Interestingly, this construction can be done only with GH matrices having one eigenvalue equal
 175 to -1 . If we set the state-transition matrix in eq. (1) to $\mathbf{A}(1) = \mathbf{R}(\theta)$, an LRNN with one layer
 176 can count modulo m , since if we also set $\mathbf{H}_0 = (1, 0)^\top$, $\mathbf{M} = (\mathbf{H}_0, \mathbf{R}(\theta)\mathbf{H}_0, \dots, \mathbf{R}((m -$
 177 $1)\theta)\mathbf{H}_0)^\top \in \mathbb{R}^{m \times 2}$, $\mathbf{B}(1) = 0$ and $\text{dec}(\mathbf{H}, x) = \arg \max_i \mathbf{M}_i^\top \mathbf{H} - 1$, then for the input $\mathbf{x} = 1^t$
 178 we get

$$\hat{y}_t = \text{dec}(\mathbf{H}_t, 1) = \text{dec}(\mathbf{A}(1)^t \mathbf{H}_0, 1) = \text{dec}(\mathbf{R}(t\theta)\mathbf{H}_0, 1) = t \bmod m.$$

179 Therefore, we examine the impact of our change to the eigenvalue range on state-transition matrices
 180 constructed as repeated products of GH matrices.

181 3.3 Products of Generalized Householder Matrices

182 We define the set of all matrices in $\mathbb{R}^{n \times n}$ that can be expressed as a product of k GH matrices with
 183 a given range $\Omega \subseteq \mathbb{R}$ of allowed eigenvalues:

$$\mathcal{M}_k(\Omega) := \{ \mathbf{C}_1 \mathbf{C}_2 \cdots \mathbf{C}_k : \mathbf{C}_i = \mathbf{I} - \beta_i \mathbf{v}_i \mathbf{v}_i^\top, \quad (1 - \beta_i) \in \Omega, \quad \mathbf{v}_i \in \mathbb{R}^n, \|\mathbf{v}_i\| = 1 \}. \quad (5)$$

184 We first observe that if $\mathbf{M}_H \in \mathcal{M}_1(\{-1\})$, then \mathbf{M}_H is a reflection, or Householder matrix, and
 185 that for any $\mathbf{x} \in \mathbb{R}^l$, $\mathbf{A}_{\text{GH}}(\mathbf{x}) \in \mathcal{M}_1([0, 1])$ and $\mathbf{A}_{\text{GH}}^-(\mathbf{x}) \in \mathcal{M}_1([-1, 1])$ so that with our change
 186 we also include reflections. Note also that $\mathcal{M}_k(\Omega) \subseteq \mathcal{M}_{k'}(\Omega')$ if either $k' \geq k$ and $1 \in \Omega'$, or if
 187 $\Omega \subseteq \Omega'$. Repeated products of diagonal matrices with values in the range $[-1, 1]$ remain diagonal,
 188 with eigenvalues in the same range. More interestingly, our next result shows that products of
 189 GH matrices can represent any matrix with Euclidean norm less than or equal to 1. Without our
 190 modification, however, or even just by restricting to $\mathcal{M}_k((-1, 1])$, they are limited to matrices where
 191 any eigenvalue is either equal to 1 or has modulus strictly smaller than 1.

192 **Proposition 1** (Expressivity of products of GH matrices). *The following hold for \mathcal{M}_k in eq. (5):*

- 193 1. For any $k \in \mathbb{N}$ and $\mathbf{N} \in \mathcal{M}_k([-1, 1])$, $\|\mathbf{N}\| \leq 1$.
- 194 2. For any $\mathbf{M} \in \mathbb{R}^{n \times n}$ with $\|\mathbf{M}\| \leq 1$, then $\mathbf{M} \in \mathcal{M}_{3n}([-1, 1])$ and if \mathbf{M} is orthogonal
 195 then $\mathbf{M} \in \mathcal{M}_n(\{-1, 1\})$, while $\mathbf{M} \in \mathcal{M}_{n-1}(\{-1, 1\})$ when \mathbf{M} is a permutation matrix.
- 196 3. Any eigenvalue λ of $\mathbf{N} \in \mathcal{M}_k((-1, 1])$ is either 1 or satisfies $|\lambda| < 1$ and if in addition
 197 $\mathbf{N} \in \mathcal{M}_k([0, 1])$, then $\lambda \in \mathbb{R}$.

198 The proof in Appendix D.1 uses linear algebra arguments, the SVD decomposition, and the fact that
 199 that every $n \times n$ orthogonal matrix can be written as a product of n reflections. A consequence of
 200 Proposition 1 is that if for every layer of an LRNN there exists $k \in \mathbb{N}$ such that the map \mathbf{A} from
 201 inputs to state transition matrix is such that $\mathbf{A} : \mathbb{R}^l \rightarrow \mathcal{M}_k((-1, 1]) \subset \mathbb{R}^{n \times n}$, then the LRNN
 202 cannot learn to count modulo m , with $m \geq 3$, due to Theorem 2. On the contrary, if we allow
 203 $\mathbf{A} : \mathbb{R}^l \rightarrow \mathcal{M}_k([-1, 1])$ and k is large enough, we can show that an LRNN with one layer can
 204 implement any FSA whose transition monoid is a group and that with multiple layers the LRNN can
 205 recognize any regular language. This is the content of the following two theorems.

206 **Theorem 3.** *Every FSA $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ whose transition monoid $\mathcal{T}(\mathcal{A})$ is also a group, can be*
 207 *implemented by a finite-precision LRNN with one layer and $\mathbf{A} : \Sigma \rightarrow \mathcal{M}_{k-1}(\{-1, 1\}) \subset \mathbb{R}^{n \times n}$,*
 208 *where n is the smallest natural number such that $\mathcal{T}(\mathcal{A})$ is isomorphic to a subgroup of S_n , and*
 209 *$k = \max_{w \in \Sigma} \sum_{q \in Q} \mathbf{1}\{\delta(q, w) \neq q\}$ is the maximum number of changed states after applying a*
 210 *single transition. Moreover, if $\mathcal{T}(\mathcal{A})$ is isomorphic to the cyclic group \mathbb{Z}_m , then we can set $\mathbf{A} : \Sigma \rightarrow$
 211 $\mathcal{M}_2([-1, 1]) \subset \mathbb{R}^{2 \times 2}$. If $m = 2$ (parity) we can set $\mathbf{A} : \Sigma \rightarrow \{-1, 1\}$.*

212 The proof in Appendix D.2 uses matrix representations of groups to map each state-transition func-
 213 tion to the corresponding matrix representation. This can always be done using permutation matri-
 214 ces, but for cyclic groups we can also use 2×2 rotation matrices. In the case of permutations, if
 215 every state-transitions permutes at most k states then the corresponding permutation matrix will be
 216 in $\mathcal{M}_{k-1}(\{-1, 1\})$, since if it is not the identity, it can be written as a product of at most $k - 1$
 217 permutations of two elements (swaps), each in $\mathcal{M}_1(\{-1\})$. A consequence of Theorem 3 is that if
 218 every transition function of the FSA has a permutation representation corresponding to a swap or to
 219 the identity, then a LRNN layer with $\mathbf{A} = \mathbf{A}_{\text{GH}}^-$, can implement it. This is useful in practice because
 220 the cost of the recursion increases k -fold if we use a product of k GH matrices compared to just one.
 221 Also, for many problems, state-transition for the FSA might either be simple or be encoded using
 222 multiple letters. For example, for addition modulo 5, a word may look like “3+2+6=1” (two letters
 223 per addition operations). This makes it possible even for an LRNN with state-transition matrices in

224 $\mathcal{M}_1([-1, 1])$ to model complex transitions. Indeed if each transition uses k letters, then if we set
 225 $\mathbf{B} \equiv 0$ and $\mathbf{A} : \mathbb{R}^l \rightarrow \mathcal{M}_1([-1, 1])$ in eq. (1), then

$$\mathbf{H}_t = \mathbf{C}(x_t, \dots, x_{t-k})\mathbf{H}_{t-k}, \quad \mathbf{C}(x_t, \dots, x_{t-k}) := \mathbf{A}(x_1)\mathbf{A}(x_2) \cdots \mathbf{A}(x_{t-k}) \in \mathcal{M}_k([-1, 1]),$$

226 which allows to model permutations that change up to $k + 1$ elements.

227 **Theorem 4.** *Every FSA $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ can be implemented by a finite-precision LRNN with*
 228 *$s \leq 2^{|\mathcal{Q}|}$ layers, each of the form 1, where $n \leq |\mathcal{Q}|$, $p \leq s$, $d = 1$, $\mathbf{A} : \mathbb{R}^l \rightarrow \mathcal{M}_n([-1, 1]) \subset \mathbb{R}^{n \times n}$*
 229 *and $\mathbf{B} : \mathbb{R}^l \rightarrow \mathbb{N}^n$. Therefore, LRNNs with state transition matrices that are repeated products of*
 230 *GH matrices each with eigenvalues in the range $[-1, 1]$ can recognize any regular languages.*

231 The proof in Appendix D.4 exploits the landmark Theorem by Krohn & Rhodes (1965), which states
 232 that every FSA can be decomposed as a *cascade* of simpler FSAs whose state-transition functions
 233 are either one-to-one or constant. Each layer of the LRNN will implement each FSA of the cascade
 234 using $n \times n$ permutation matrices, with n being the number of states of the FSA, which are in
 235 $\mathcal{M}_{n-1}(\{-1, 1\})$, for the one-to-one transitions, while for constant (state-independent) transitions it
 236 will set the state-transition matrix to $0 \in \mathcal{M}_n(\{0\})$ and set \mathbf{B} in (1) accordingly. Note that we can
 237 obtain the matrix $0 \in \mathbb{R}^{n \times n}$ only inefficiently as a product of n GH matrices, while it can also be
 238 obtained with a single diagonal matrix. This points towards LRNNs hybrids using a mix of GH and
 239 diagonal matrices, whose exploration we leave for future work.

240 **Discussion** The results in Theorems 3 and 4 for LRNNs are in sharp contrast with the ones for
 241 transformers (Liu et al., 2023; Merrill & Sabharwal, 2023) and diagonal LRNNs (Merrill et al.,
 242 2024), which always require either the number of layers or the precision growing with the sequence
 243 length, and in most cases can only solve group word problems where the group is *solvable*, i.e.
 244 excluding S_n with $n \geq 5$. Moreover, we note that compared to LRNNs without any restriction
 245 to the norm of the state-transition matrices, which need only one layer to recognize any regular
 246 language, our result requires both the number of layers and the width of the LRNN to be (in the
 247 worst case) exponential in the number of states of the FSA, although we conjecture that the number
 248 of layers can probably be reduced to at most linear using a more refined decomposition.

249 4 Experiments

250 We investigate the effects of expanding the eigenvalue
 251 range of state-transition matrices from $[0, 1]$ to $[-1, 1]$, as
 252 explained in Section 3.2, on both synthetic tasks and lan-
 253 guage modeling. Our experiments involve Mamba, and
 254 DeltaNet, with variants trained using both the original
 255 and extended eigenvalue ranges, as shown in Table 2. We
 256 label these variants accordingly. Note, that the changes
 257 increase the expressivity of Mamba and DeltaNet while
 258 coming at no additional computational cost. Detailed im-
 259 plementation information can be found in Appendix E.1.

Table 2: Summary of modifications to the state-transition matrices $\mathbf{A}(\mathbf{x}_t)$ to extend the eigenvalue range from $[0, 1]$ (Table 1) to $[-1, 1]$. We set $\mathbf{s}(\mathbf{x}_t) = \exp(-\Delta_{t,i} \exp(\mathbf{w}_{1,i}))$.

	$[0, 1]$	$[-1, 1]$
Mamba	$\text{Diag}(\mathbf{s}(\mathbf{x}_t))$	$\text{Diag}(2\mathbf{s}(\mathbf{x}_t) - 1)$
DeltaNet	$\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top$	$\mathbf{I} - 2\beta_t \mathbf{k}_t \mathbf{k}_t^\top$

260 4.1 Chomsky Hierarchy

261 We conducted experiments with some of the formal language tasks proposed by Deletang et al.
 262 (2023) and similarly used to benchmark xLSTM (Beck et al., 2024). Our focus was on tasks where
 263 mLSTM (a linear RNN) previously underperformed while sLSTM (a non-linear RNN) succeeded,
 264 specifically parity, modular arithmetic (regular languages) and modular arithmetic with brackets
 265 (context-free language). As in Beck et al. (2024), we trained each model with sequence lengths
 266 ranging from 3 to 40 and evaluated on lengths from 40 to 256, to assess length generalization. We
 267 compared mLSTM and sLSTM with two models: Mamba and DeltaRule. Our findings, presented
 268 in Table 3, demonstrate that expanding the range of eigenvalues from $[0, 1]$ to $[-1, 1]$ enables all
 269 examined models to fully solve the parity task. For modular arithmetic, this expansion led to sub-
 270 stantial performance improvements for Mamba and DeltaNet. It is worth noting that we were unable
 271 to replicate the sLSTM results reported by Beck et al. (2024) for the modular arithmetic tasks. In
 272 Figure 2, we also report performance vs sequence length for DeltaNet. We provide details on the
 273 tasks and experimental setup in Appendix E.2.

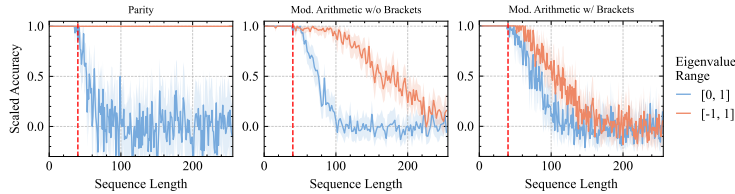


Figure 2: *DeltaNet* variants’ performance (scaled accuracy) on formal language tasks across sequence lengths. Models trained on sequences up to length 40 (red dashed line). Test on 8192 randomly sampled sequences, lengths 40-256. Curves show mean and 95% confidence interval.

274 4.2 State-Tracking

275 We perform experiments on group word problems, relying on the code provided by Merrill
 276 et al., 2024. In particular, we focus on the
 277 S_5 group, which is the first *unsolvable* symmetric
 278 group where current LRNN and transformers are known to perform poorly. We also
 279 report results for the addition modulo 60, i.e. the cyclic group \mathbb{Z}_{60} , in Appendix E.3.2. We
 280 note that parity is S_2 . In these experiments, the input to the model is a sequence of group
 281 elements, while the supervision is given by another sequence of group elements, each being
 282 the product of the previous ones in the input. Since solving S_5 would require LRNNs with
 283 state-transition matrices that are repeated products of 4 GH matrices, each with eigenvalues
 284 $[-1, 1]$, we also consider three simplified setups: (i) allowing as inputs only permutations
 285 up to 2 elements (identity and swaps), (ii) allowing only permutations up to 3 elements, (iii)
 286 using 4 tokens for each permutation by having each token followed by a special token repeated 3
 287 times. Additional details are in Appendix E.3. We stress that, even when restricting the inputs,
 288 possible outputs remains the same, since swaps are generators of the group.

289 **Results** Figure 3 shows that, as predicted by Theorem 3, restricting the inputs to only swap
 290 permutations allows *DeltaNet* $[-1, 1]$ with one layer to fully learn the task (since its state
 291 transition matrix can model a swap), while *DeltaNet* $[0, 1]$ only manages to fit the training
 292 length, even with 5 layers. On the contrary, just by including also permutations of 3 elements
 293 we notice a substantial decrease in the performance of all models, although extending the range
 294 is still advantageous and *DeltaNet* $[-1, 1]$ with 5 layers reaches a good length generalization.
 295 Moreover, using 4 tokens per group element seems also beneficial compared to standard S_5 ,
 296 since *DeltaNet* $[-1, 1]$ with 5 layers manages to extrapolate very well until around length 120,
 297 which corresponds to 30 group elements, while all model trained on standard S_5 have 0
 298 sequence accuracy prior to sequence length 20. We also report that Mamba, being a diagonal
 299 model, performs poorly on all setups, with and without increased eigenvalue range.

300 4.3 Language Modeling

301 **Experimental Setup** We train *DeltaNet* models with 340M parameters and Mamba models
 302 with 370M parameters, each using both original and extended eigenvalue ranges. The training
 303 is done on 32B tokens from the FineWeb-100B dataset (Penedo et al., 2024). For training
 304 details we refer to Appendix E.4.1. Given our previous theoretical and experimental findings,
 305 we hypothesize that models (especially *DeltaNet*) with extended eigenvalue range will perform
 306 better on language modeling tasks that require state-tracking such as coding or mathematics,
 307 compared to unmodified models. To test this hypothesis, we evaluate the perplexity of these
 308 models in a length extrapolation setup using various datasets: CodeParrot (Tunstall et al.,
 2022) for coding, Math-Hard (Hendrycks et al., 2022) for mathematics, TriviaQA (Joshi et al.,
 2017), and SlimPajama (Soboleva et al., 2023).

309 **Results** Both models trained stably even with our modification and without changing the
 310 learning rate. The validation perplexity was comparable, albeit slightly lower, throughout
 311 training (See Fig-

Table 3: Performance comparison of various recurrent models on regular and context-free language tasks. We report the best of 3 runs with different seeds (Table 5 in the Appendix reports the median). Scores represent scaled accuracy, with 1.0 indicating perfect performance and 0.0 random guessing. The positive impact of allowing negative eigenvalues ($[-1, 1]$ range) versus restricting to positive eigenvalues ($[0, 1]$ range) is evident across different architectures. Results in parenthesis are as reported in Beck et al. (2024).

	Parity	Mod. Arithm. (w/o brackets)	Mod. Arithm. (w/ brackets)
mLSTM	0.087 (0.04)	0.122 (0.04)	0.120 (0.03)
sLSTM	1.000 (1.00)	0.135 (1.00)	0.133 (0.57)
Mamba $[0, 1]$	0.000	0.003	0.018
Mamba $[-1, 1]$	1.000	0.111	0.036
<i>DeltaNet</i> $[0, 1]$	0.017	0.187	0.182
<i>DeltaNet</i> $[-1, 1]$	1.000	0.612	0.339

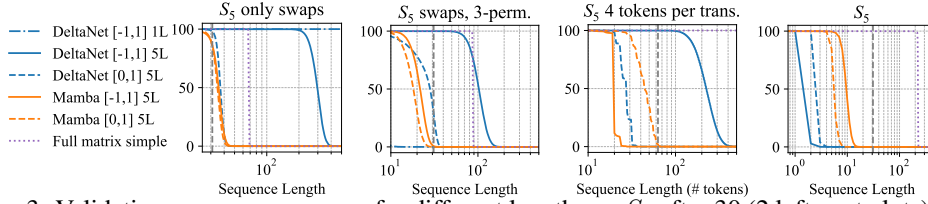


Figure 3: Validation sequence accuracy for different lengths on S_5 after 30 (2 leftmost plots) and 90 epochs of training (1 seed). The dashed vertical line indicates the sequence length used during training (32 except for the third plot from the left where it is 64 since we use 4 tokens per permutation). Each method is labeled with name, eigenvalues range and number of layers. The dashed vertical line indicates the sequence length used during training. Full matrix simple, is a simple one-layer baseline where the state update matrices are unstructured and has no control on the eigenvalues.

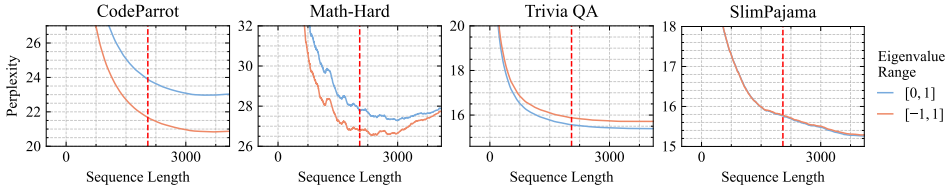


Figure 4: Performance vs sequence length of DeltaNet variants on different datasets. DeltaNet with eigenvalue range $[-1, 1]$ shows improved perplexity on coding and math tasks compared to the $[0, 1]$ baseline. The dashed vertical line indicates the training context length of 2048 tokens.

321 ure 7). The experiments in Figure 4 demonstrate that on coding and math datasets, DeltaNet with
 322 an eigenvalue range of $[-1, 1]$ achieves lower perplexity than the baseline with range $[0, 1]$. For
 323 TriviaQA, the perplexity of DeltaNet $[-1, 1]$ is slightly higher. Note that this is a task relying on
 324 memorization, not linked to state-tracking and hence we do not expect an improvement. On SlimPa-
 325 jama, we observe no significant difference between the two DeltaNet variants. For Mamba instead,
 326 we see a general degradation of the performance on these tasks compared to the unmodified version
 327 (Figure 8 in the Appendix). To ensure our models are comparable with those obtained by Yang
 328 et al. (2024b), we evaluate them on the same benchmark tasks from lm-harness (Gao et al., 2024)
 329 in Table 4. It’s worth noting that we trained on 32B tokens of FineWeb, while Yang et al. (2024b)
 330 reported results from training on 15B tokens of SlimPajama. We find that our models perform worse
 331 in terms of perplexity on WikiText and LAMBADA, while achieving better average accuracy on
 332 classic benchmarks. Furthermore, we report that DeltaNet $[0, 1]$ performs better on recall-intensive
 333 tasks SWDE and SQuAD, where our eigenvalue extension degrades performance.

334 **5 Conclusion**

335 We showed the substantial impact of extending the eigenvalue range of state-transition matrices in
 336 LRNNs from $[0, 1]$ to $[-1, 1]$. This modification significantly enhances the expressivity of LRNNs
 337 in state-tracking tasks with no additional overhead in training or inference. While Mamba success-
 338 fully solves the parity problem, its diagonal matrix structure inherently limits further performance
 339 gains. In contrast, DeltaNet, by leveraging its non-diagonal matrix structure, excels across a broader
 340 spectrum of tasks. Our results underscore the critical role of non-diagonal state-transition matri-
 341 ces in augmenting state-tracking capabilities, highlighting a promising direction for future LRNN
 342 advancements, and potential impact in tasks such as natural language processing, time-series fore-
 343 casting and reinforcement learning. In our language modeling experiments, we did not observe any
 344 performance gains with the Mamba model. Furthermore, diagonal models such as Mamba2 and
 345 GLA utilize the positivity of state transition matrices to compute repeated products in log space, a
 346 technique our modification does not directly support. This limitation may introduce potential in-
 347 stabilities in certain cases (refer to Appendix E.1 for more details). Further research is needed to
 348 assess the impact of training large-scale language models with state-tracking capabilities. To this
 349 end, we aim to understand the potential downsides of increased expressivity, which could guide hy-
 350 brid model design. For example, we hypothesize a fundamental trade-off between state-tracking and
 351 memorization, which holds theoretical interest.

352 References

- 353 Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer,
354 and Christopher Ré. Language models enable simple systems for generating structured views
355 of heterogeneous data lakes. *Proceedings of the VLDB Endowment*, 17(2):92–105, 2023.
- 356 Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova,
357 Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Ex-
358 tended Long Short-Term Memory. In *Advances in Neural Information Processing Systems*. Cur-
359 ran Associates, Inc., 2024.
- 360 Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about
361 physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial*
362 *Intelligence*, 34(05):7432–7439, Apr. 2020.
- 363 Guy E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of
364 Computer Science, Carnegie Mellon University, 1990.
- 365 Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and
366 Oyvind Tafjord. Think you have solved question answering? Try arc, the ai2 reasoning challenge.
367 *arXiv preprint arXiv:1803.05457*, 2018.
- 368 Tri Dao and Albert Gu. Transformers are SSMS: Generalized models and efficient algorithms
369 through structured state space duality. In *International Conference on Machine Learning*. PMLR,
370 2024.
- 371 Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt,
372 Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, et al. Neural Networks and the Chomsky
373 Hierarchy. In *The Eleventh International Conference on Learning Representations*, 2023.
- 374 Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Re.
375 Hungry hungry hippos: Towards language modeling with state space models. In *The Eleventh*
376 *International Conference on Learning Representations*, 2021.
- 377 Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Fos-
378 ter, Laurence Golding, Jeffrey Hsu, Alain Le Noac’h, Haonan Li, Kyle McDonell, Niklas Muen-
379 nighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang
380 Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for
381 few-shot language model evaluation, 07 2024.
- 382 Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv*
383 *preprint arXiv:2312.00752*, 2023.
- 384 Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured
385 state spaces. In *International Conference on Learning Representations*, 2022.
- 386 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn
387 Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset.
388 In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks*
389 *Track (Round 2)*, 2021.
- 390 Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8):
391 1735–1780, 1997.
- 392 Roger A Horn and Charles R Johnson. *Matrix Analysis*. Cambridge University Press, 2012.
- 393 Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly
394 supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meet-*
395 *ing of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611,
396 2017.
- 397 Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are
398 rns: Fast autoregressive transformers with linear attention. In *International Conference on Ma-*
399 *chine Learning*, pp. 5156–5165. PMLR, 2020.

- 400 Kenneth Krohn and John Rhodes. Algebraic theory of machines. i. prime decomposition theorem
401 for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:
402 450–464, 1965.
- 403 Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers
404 learn shortcuts to automata. In *The Eleventh International Conference on Learning Representa-*
405 *tions*, 2023.
- 406 Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. When open information extraction meets
407 the semi-structured web. *NAACL-HLT. Association for Computational Linguistics*, 2019.
- 408 Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In
409 *International Conference on Learning Representations*, 2017.
- 410 Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Con-*
411 *ference on Learning Representations*, 2019.
- 412 Oded Maler and Amir Pnueli. On the cascaded decomposition of automata, its complexity and its
413 application to logic. *ACTS Mobile Communication*, 48, 1994.
- 414 William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision trans-
415 formers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023.
- 416 William Merrill, Jackson Petty, and Ashish Sabharwal. The Illusion of State in State-Space Models.
417 In *Forty-first International Conference on Machine Learning*, 2024.
- 418 Antonio Orvieto, Soham De, Caglar Gulcehre, Razvan Pascanu, and Samuel L Smith. Universality
419 of linear recurrences followed by non-linear projections: Finite-width guarantees and benefits of
420 complex eigenvalues. In *Forty-first International Conference on Machine Learning*, 2024.
- 421 Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi,
422 Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset:
423 Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting*
424 *of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, 2016.
- 425 Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin
426 Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the
427 finest text data at scale, 2024.
- 428 Bo Peng, Eric Alcaide, Quentin Gregory Anthony, Alon Albalak, Samuel Arcadinho, Stella Bider-
429 man, Huanqi Cao, Xin Cheng, Michael Nguyen Chung, Leon Derczynski, et al. Rwkv: Rein-
430 venting rns for the transformer era. In *The 2023 Conference on Empirical Methods in Natural*
431 *Language Processing*, 2023.
- 432 Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *Journal of Ma-*
433 *chine Learning Research*, 22(75):1–35, 2021.
- 434 Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don’t know: Unanswerable questions
435 for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational*
436 *Linguistics (Volume 2: Short Papers)*, pp. 784–789, 2018.
- 437 Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adver-
438 sarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.
- 439 Yash Sarrof, Yana Veitsman, and Michael Hahn. The Expressive Capacity of State Space Models:
440 A Formal Language Perspective. *Advances in Neural Information Processing Systems*, 2024.
- 441 Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey.
442 SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, June 2023.
- 443 Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei
444 Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): RNNs with expressive
445 hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

- 446 Alexandre Torres. mamba.py: A simple, hackable and efficient Mamba implementation in pure
447 PyTorch and MLX., 2024. URL <https://github.com/alxndrTL/mamba.py>.
- 448 Lewis Tunstall, Leandro Von Werra, and Thomas Wolf. *Natural language processing with trans-*
449 *formers*. ” O’Reilly Media, Inc.”, 2022.
- 450 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez,
451 Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Infor-*
452 *mation Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- 453 Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated Linear Atten-
454 tion Transformers with Hardware-Efficient Training. In *Forty-first International Conference on*
455 *Machine Learning*, 2024a.
- 456 Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing Linear Trans-
457 formers with the Delta Rule over Sequence Length. *Advances in Neural Information Processing*
458 *Systems*, 36, 2024b.
- 459 Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a ma-
460 chine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association*
461 *for Computational Linguistics*, pp. 4791–4800, 2019.

462 A Related Work

463 **Linear RNNs.** Linear RNNs have emerged as scalable alternatives to transformers, encompass-
464 ing state-space models and linear attention mechanisms. State-space models, originally used for
465 continuous dynamical systems, inspired LRNN variants like S4 (Gu et al., 2022) and H4 (Fu et al.,
466 2021). Recent advancements, such as Mamba (Gu & Dao, 2023; Dao & Gu, 2024), introduced input-
467 dependent gating of the hidden state, significantly improving language modeling performance. Con-
468 currently, linear attention emerged as an alternative to classical softmax attention, with Katharopoulos
469 et al. (2020) demonstrating that causal, linear attention transformers can be reformulated as
470 RNNs with linear scaling in sequence length. Building on this, Yang et al. (2024a) proposed Gated
471 Linear Attention (GLA), adding a gating mechanism similar to Mamba, while DeltaNet (Yang et al.,
472 2024b) and TTT-Linear (Sun et al., 2024) explored more expressive gating with non-diagonal state-
473 transition matrices. Recent work has combined non-linear and linear RNNs, as seen in xLSTM
474 (Beck et al., 2024), a successor to the traditional LSTM (Hochreiter & Schmidhuber, 1997).

475 **Expressivity Results.** Several studies have explored the expressive power of transformers. Liu
476 et al. (2023) demonstrated that transformers can learn shortcut solutions for *solvable* finite state
477 automata, though these solutions lack generalizability to arbitrary sequence lengths and perform
478 poorly out-of-distribution. Unlike RNNs, transformers’ high parallelizability prevents them from
479 learning *unsolvable* finite state automata Merrill & Sabharwal (2023). These findings typically use
480 algebraic formal language theory and circuit complexity (we refer to Liu et al. (2023), for a tutorial
481 on these topics), using the *log-precision assumption* and a finite number of layers scaling linearly
482 or logarithmically with sequence length. While earlier research established transformers’ Turing
483 completeness, it relied on either arbitrary precision (Deletang et al., 2023) or infinite depth (Pérez
484 et al., 2021).

485 Diagonal LRNNs can simulate any RNN with infinite depth Gu & Dao (2023) and approximate reg-
486 ular enough functions when the state dimension grows linearly with sequence length (Orvieto et al.,
487 2024). However, things change when depth state size are fixed. Merrill et al. (2024) demonstrated
488 that finite-depth diagonal LRNNs, like transformers, cannot learn unsolvable finite state automata
489 when restricted to log-precision arithmetic. In a finite-precision setting, Sarrof et al. (2024) showed
490 that diagonal LRNNs with positive values in the state-transition matrix, while capable of learning
491 all star-free languages, cannot solve even the simple *parity* problem, a non-star-free language recog-
492 nizable by a solvable automaton with two states. However, their analysis was limited to the diagonal
493 case and they did not test the benefit of negative eigenvalues in practice. Differently from these
494 works, we also study non-diagonal LRNNs that can still be trained efficiently at large scale.

495 B Additional Background

496 B.1 Notation

497 We use lower case letters for scalar quantities (e.g. $x \in \mathbb{R}$), bold lower case letters for (column)
498 vectors (e.g. $\mathbf{v} \in \mathbb{R}^n$), and bold upper case letters for matrices (e.g. $\mathbf{M} \in \mathbb{R}^{n \times d}$). Some functions
499 with matrix (vector) outputs, such as \mathbf{A} and \mathbf{B} in Equation (1), are also bold upper (lower) case
500 letters to put emphasis on the fact that they output matrices (vectors). We denote with $\|\mathbf{v}\|$ the
501 euclidean norm of the vector $\mathbf{v} \in \mathbb{R}^n$. When $\mathbf{M} \in \mathbb{R}^{n \times d}$, $\|\mathbf{M}\|$ also refers to the euclidean norm,
502 corresponding to the largest singular value. We also define for a boolean s

$$\mathbf{1}\{s\} := \begin{cases} 1 & \text{if } s \text{ is true} \\ 0 & \text{if } s \text{ is false} \end{cases}.$$

503 We define $\text{sigmoid}(x) := 1/(1 + e^{-x})$ and $\text{softplus}(x) := \ln(1 + e^x)$.

504 We sometimes use regular expressions to represent their corresponding formal language. So that
505 e.g. $(11)^* = \{\epsilon\}^*$ is the set of words containing the empty word ϵ the words with an even number
506 of ones. When 1^m is the word containing 1 repeated m times.

507 B.2 Formal Language Theory

508 **Finite State Automata and Regular Languages.** A (deterministic) finite state automaton (FSA)
509 is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ where Σ is a finite set of letters called alphabet, Q is a finite set of

510 states, $q_0 \in Q$ is the starting state and $\delta: Q \times \Sigma \rightarrow Q$ is the state-transition function. We define
511 the set Σ^* , whose elements are sequences called words, as the smallest superset of Σ that con-
512 tains the empty word ε and is closed under word concatenation. We extend the state-transition
513 function to $\delta: Q \times \Sigma^* \rightarrow Q$ by defining $\delta(q, \varepsilon) = q$ and $\delta(q, \mathbf{w}) = \delta(\delta(q, w_1 \dots w_{i-1}), w_i)$ for any
514 $\mathbf{w} = w_1 \dots w_i \in \Sigma^*$ with $i \geq 2$. We say that $\delta(q_0, \mathbf{w})$ is the state that \mathcal{A} reaches after read-
515 ing the word $\mathbf{w} \in \Sigma^*$. A language $L \subseteq \Sigma^*$ is said to be recognized by \mathcal{A} if there exists a
516 recognizing set $R \subseteq Q$ such that $L = \{\mathbf{w} \in \Sigma^* : \delta(q_0, \mathbf{w}) \in R\}$. A language is called regular (or
517 rational) if there exists a finite state automaton that can recognize it. Given a FSA \mathcal{A} , the set
518 $\mathcal{T}(\mathcal{A}) = \{\delta(\cdot, \mathbf{w}) : \mathbf{w} \in \Sigma^*\}$ of functions $\rho: Q \rightarrow Q$, together with the function composition op-
519 eration forms a Monoid called *transition monoid*, i.e. it is associative, closed and contains the
520 identity $\delta(\cdot, \varepsilon)$. Such monoid has a finite number of elements, since $|Q| < \infty$. Moreover, if $\delta(\cdot, w)$
521 is bijective for every $w \in \Sigma$, then $\mathcal{T}(\mathcal{A})$ forms a *group*, i.e. it contains the inverse for each element.

522 **State-Tracking and Monoid Word Problems.** State-tracking is the problem of determining the
523 state of a system only by observing a sequence of updates applied to it. Merrill et al. (2024) showed
524 that it can be formally captured by *monoid word problems*, where given a monoid (M, \cdot) (M is
525 the set and \cdot is the associative operation), we want to send words $m_1 \dots m_t \in M^*$, describing the
526 sequence of updates, to their product $m_1 \cdot m_2 \cdot \dots \cdot m_t \in M$, representing the state of the system after
527 the updates. If M is finite there is a corresponding FSA (M, M, e, δ) that solves the word problem,
528 where the alphabet is equal to the set of states, the starting state is e (the identity element), and the
529 transition function is $\delta(m_1, m_2) = m_2 \cdot m_1$ for $m_1, m_2 \in M$. In this work, we focus on group word
530 problems, i.e. problems where the monoid is also a group. In particular, on the cyclic group \mathbb{Z}_m ,
531 i.e. addition modulo m and symmetric groups S_m , i.e. the group of permutations on m elements.
532 Parity is equivalent to the S_2 word problem, while many state-tracking problems such as tracking
533 chess moves or code evaluation, can be shown to be harder than the S_5 word problem, which cannot
534 be solved by transformers and diagonal LRNNs even in log-precision (Merrill et al., 2024; Merrill
535 & Sabharwal, 2023).

536 **One LRNN Layer is an FSA.** Given an alphabet $\Sigma \subset \mathbb{N}$, we can view one layer of an LRNN in
537 (1) as the FSA $\mathcal{A}_{\text{lin}} = (\Sigma, \mathcal{H}, \mathbf{H}_0, \delta_{\text{lin}})$, where $\delta_{\text{lin}}(\mathbf{H}, w) = \mathbf{A}(w)\mathbf{H} + \mathbf{B}(w)$, which is extended
538 as we saw previously¹, and $\mathcal{H} = \{\delta_{\text{lin}}(\mathbf{H}_0, \mathbf{w}) : \mathbf{w} \in \Sigma^*\}$. We say that a LRNN layer in (1)
539 implements the FSA $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ if \mathcal{A}_{lin} can mimic the state transitions of \mathcal{A} ². Formally, if
540 there exist a surjective function $g: \mathcal{H} \rightarrow Q$, such that for any $\mathbf{H} \in \mathcal{H}, w \in \Sigma$

$$\delta(g(\mathbf{H}), w) = g(\delta_{\text{lin}}(\mathbf{H}, w)) = g(\mathbf{A}(w)\mathbf{H} + \mathbf{B}(w))$$

541 Every language L recognized by \mathcal{A} can also be recognized by this LRNN layer with a sufficiently
542 powerful dec. In particular if $R \subseteq Q$ is the recognizing set for L and \mathbf{H}_0 is such that $q_0 = g(\mathbf{H}_0)$,
543 then the decoder $\text{dec}(\mathbf{H}_t, w_t) = \mathbf{1}\{g(\mathbf{H}_t) \in R\}$, will correctly determine if $\mathbf{w} \in L$. Therefore,
544 implementing \mathcal{A} is harder than recognizing L . A principal goal of this work is to show that that cur-
545 rent LRNNs cannot recognize simple languages such as parity (negative results) while appropriate
546 modifications to the state-transition matrices, enable LRNNs to recognize broader classes of FSA
547 (positive results), with certain classes of FSA requiring a single layer. Notice, that while LRNNs
548 with one layer can recognize any regular language, the state transition matrix might not fit into the
549 structure imposed by current LRNNs, such as those in Table 1 (see Appendix B.3).

550 B.3 Regular Languages and Recurrent Neural Networks

551 **RNNs Can Recognize Any Regular Language** A layer of a general RNN can be formulated
552 similarly to eq. (1) just by replacing the linear state update with a generic state-transition function g
553 as:

$$\mathbf{h}_t = g(\mathbf{h}_{t-1}, \mathbf{x}_t), \quad \mathbf{h}_0 \in \mathbb{R}^n.$$

554 Therefore, any FSA can be implemented by an RNN layer if g is sufficiently expressive to model its
555 state transition function.

556 **LRNNs Can Recognize Any Regular Language** As explained in (Liu et al., 2023, Appendix A.2)
557 and in (Merrill et al., 2024, Theorem 5), we can always implement any FSA, and thus recognize any
558 regular language, using matrix-vector multiplication and hence also a one layer LRNN by choosing

¹We let $\delta_{\text{lin}}: \mathbb{R}^{n \times d} \times \Sigma \rightarrow \mathbb{R}^{n \times d}$ and extend it to $\delta_{\text{lin}}: \mathbb{R}^{n \times d} \times \Sigma^* \rightarrow \mathbb{R}^{n \times d}$ since we didn't define \mathcal{H} yet
²This definition is equivalent to that of FSA homomorphism, see (Maler & Pnueli, 1994, Definition 3)

559 $n = |Q|$, $\mathbf{H}_0 = (1, 0 \dots, 0)^\top$ and by letting, for any $w \in \Sigma$, $\mathbf{B}(w) = 0$ and $\mathbf{A}(w) \in \mathbb{R}^{n \times n}$ the matrix with entries $\mathbf{A}(w)_{q',q} = \mathbf{1}\{\delta(w, q) = q'\}$. However, such construction cannot be implemented
 560 by modern LRNNs since in general $\mathbf{A}(w)$ can have norm greater than one and might not be sym-
 561 metric. Both conditions are not allowed by the state-transition matrices in modern LRNNs (see e.g.
 562 the ones in Table 1).
 563

564 B.4 Finite Precision

565 For our positive results on LRNNs expressivity (Theorems 3 and 4), by finite precision we mean that
 566 since all quantities involved in the computations are a finite number, there exist a finite set $\mathbb{D} \subset \mathbb{R}$
 567 that contains them and thus we do not require computations to be done in the reals but we can
 568 use \mathbb{D} as datatype. In particular, \mathbb{D} does not depend on the length of the input sequence. In practice,
 569 such datatype is chosen beforehand, e.g. floating point numbers requiring a given number of bits,
 570 which may not capture all quantities in our constructions.

571 In our negative results of Theorems 1 and 2 instead, we can pick the finite set \mathbb{D} is arbitrarily, e.g.
 572 floating point numbers, and we also make the use of the function $\text{cast} : \mathbb{R} \rightarrow \mathbb{D}$, that we extend
 573 to \mathbb{C} by applying it separately to real and imaginary part and to vector and matrices by applying it
 574 element wise. The cast function is used because some computations of the state of the LRNN will
 575 be allowed to be in infinite precision and then transformed to finite-precision using cast as specified
 576 in the proofs.

577 We believe that the finite precision setup is not only realistic but also allows a better focus on the
 578 drawbacks of modern LRNN. We note that for transformers, results usually rely instead on the notion
 579 of log-precision, meaning that the size of \mathbb{D} grows logarithmically with the sequence length.
 580 This is mainly due to their limited expressivity compared to LRNNs. We also note that concern-
 581 ing the state-transition matrices of modern LRNNs (see Table 1), the values at the extremes of the
 582 eigenvalue range are technically not included (because of the use of the sigmoid and softplus func-
 583 tions). However, since we are working with finite precision, we can still include them by choosing
 584 the appropriate datatype \mathbb{D} , which also in practice includes key values such as 0, 1 and -1 .

585 B.4.1 The Decoder function

586 In (1), to compute the output \hat{y}_t from the state \mathbf{H}_t and the vector \mathbf{x}_t , of an LRNN layer, we use
 587 the function `dec`, to abstract away the computations that is done on \mathbf{H}_t and \mathbf{x}_t , since they are not
 588 part of the recurrence. In this work, we do not consider the internal structure of `dec`, but it usually
 589 contains a normalization and a feed-forward neural network and thus it can usually approximate any
 590 function.

591 In our negative results on LRNNs expressivity Theorems 1 and 2, our choice of arbitrary decoder
 592 guarantees the stronger results. For our positive results instead we either do not consider the decoder
 593 (Theorem 3) or we make use of a linear decoder (Theorem 4). We point out that to recognize regular
 594 languages efficiently and with a smaller LRNN state it is beneficial to have a more powerful (non-
 595 linear) decoder, as in the case of word problems for cyclic or permutation groups. However, such
 596 decoder might probably be harder to approximate.

597 C Parity and Modular Counting – Proofs

598 We report the full proofs for the theorems in Section 3.1.

599 C.1 Proof of Theorem 1

600 The language $(11)^*$ contains all sequences with an even number of ones. An FSA recognizing the
 601 language, for the sequence 1^k will output $y_k = 1$ if k is even and $y_k = 0$ if k is odd. Consider an
 602 LRNN with one layer as in (1). We will prove that if the assumptions on the eigenvalues of $\mathbf{A}(1)$
 603 are not satisfied, then there exists a $\bar{k} > 0$ such that for every $k \geq \bar{k}$, the finite-precision version
 604 of the state \mathbf{H}_k corresponding to the sequence 1^k does not depend on k and is equal to $\bar{\mathbf{H}}$. Hence,
 605 no matter the choice of `dec`, also the finite-precision version of \hat{y}_k will not vary with k and thus for
 606 some $k' \geq \bar{k}$, $\hat{y}_{k'} \neq k' \bmod 2 = y_{k'}$. An inductive argument can then be used for the case of

607 LRNNs with multiple (finitely many) layers, using the fact that the input of the next layer will be
 608 constant for k large enough, as the input of the first layers.

609 By unrolling the recursion in 1 we obtain a closed form expression for the state

$$\mathbf{H}_k = \sum_{i=1}^{k-1} \left(\prod_{j=i+1}^{k-1} \mathbf{A}(\mathbf{x}_j) \right) \mathbf{B}(\mathbf{x}_i) + \left(\prod_{i=1}^k \mathbf{A}(\mathbf{x}_i) \right) \mathbf{H}_0,$$

610 where we set $\prod_{j=k}^{k-1} \mathbf{A}(\mathbf{x}_j) = \mathbf{I}$ to avoid clutter. We follow Merrill et al. (2024) and make the sim-
 611 plifying assumption that \mathbf{H}_k is computed using the above expression by first evaluating all products
 612 involving the matrices $\mathbf{A}(\mathbf{x}_j)$ separately and in infinite precision, then casting them into finite preci-
 613 sion, and finally executing the sum also in infinite precision and casting the result in finite precision.
 614 Hence, if we set $\mathbf{x}_1 \dots \mathbf{x}_t = \mathbf{1}^k$, we get the following the exact and finite precision expressions for
 615 the state at time k .

$$\mathbf{H}_k = \sum_{i=0}^{k-1} \mathbf{A}(1)^i \mathbf{B}(1) + \mathbf{A}(1)^k \mathbf{H}_0, \quad \widehat{\mathbf{H}}_k = \text{cast} \left(\sum_{i=0}^{k-1} \text{cast}(\mathbf{A}(1)^i \mathbf{B}(1)) + \text{cast}(\mathbf{A}(1)^k \mathbf{H}_0) \right),$$

616 where cast is an operation which rounds matrices with complex values elementwise into finite-
 617 precision. In particular, we consider the case where both the real and imaginary parts are casted
 618 separately.

619 Using the Jordan Canonical Form (e.g. Horn & Johnson, 2012) we can write $\mathbf{A}(1) = \mathbf{P}\mathbf{J}\mathbf{P}^{-1}$,
 620 where \mathbf{J} is block diagonal made of the Jordan blocks $\mathbf{J}_1, \dots, \mathbf{J}_s$ with $s \leq n$, $\mathbf{J}_i \in \mathbb{R}^{k_i \times k_i}$ and
 621 with corresponding complex eigenvalues $\lambda_1 \dots \lambda_s$. Such decomposition is useful because it allows
 622 to write matrix powers as

$$\mathbf{A}(1)^k = \mathbf{P}\mathbf{J}^k\mathbf{P}^{-1}, \quad \mathbf{J}_i^k = \begin{bmatrix} \lambda_i^k & \binom{k}{1}\lambda_i^{k-1} & \binom{k}{2}\lambda_i^{k-2} & \dots & \dots & \binom{k}{k_i-1}\lambda_i^{k-k_i+1} \\ & \lambda_i^k & \binom{k}{1}\lambda_i^{k-1} & \dots & \dots & \binom{k}{k_i-2}\lambda_i^{k-k_i+2} \\ & & \ddots & \ddots & \vdots & \vdots \\ & & & \ddots & \ddots & \vdots \\ & & & & \lambda_i^k & \binom{k}{1}\lambda_i^{k-1} \\ & & & & & \lambda_i^k \end{bmatrix}.$$

623 Therefore, to study $\lim_{k \rightarrow \infty} \mathbf{A}(1)^k$, we can study the behaviour of the elements of the Jordan blocks
 624 when $k \rightarrow \infty$. If $|\lambda_i| < 1$ then all elements of \mathbf{J}_i^k converge to zero, since the exponential is faster
 625 than the binomial $\binom{k}{j}$ with fixed j . Thus $\lim_{k \rightarrow \infty} \mathbf{J}_i^k = \mathbf{0}$. If instead $\lambda_i \in \mathbb{R}$ and $\lambda_i > 1$, then
 626 all nonzero elements of the Jordan block diverge to $+\infty$. Finally, when $\lambda_i \in \mathbb{R}$ and $\lambda_i = 1$,
 627 the diagonal elements are $\lambda_i^k = 1$, while the other nonzero elements diverge to ∞ . Therefore we
 628 have that if $|\lambda_i| < 1$ or λ_i is real and positive then there exists $\overline{\mathbf{J}}_i \in \{0, 1, \infty\}^{k_i \times k_i}$ such that
 629 $\lim_{k \rightarrow \infty} \mathbf{J}_i^k = \overline{\mathbf{J}}_i$. Now, assume that for every i either $|\lambda_i| < 1$ or $\lambda_i \in \mathbb{R}$ with $\lambda_i \geq 1$. Then, from
 630 the structure of the Jordan decomposition, each element of the matrices $\mathbf{A}(1)^k \mathbf{B}(1)$ and $\mathbf{A}(1)^k \mathbf{H}_0$
 631 will be a linear combination (with complex coefficients) of sequences of real numbers with well
 632 defined limits (either 0, 1 or $+\infty$), and thus, when $k \rightarrow \infty$ either converges to a point in \mathbb{C} or
 633 diverges to a specific point in the complex infinity.

634 Now let $\widehat{\mathbf{C}}_k = \text{cast}(\mathbf{A}(1)^k \mathbf{B}(1))$ and $\widehat{\mathbf{D}}_k = \text{cast}(\mathbf{A}(1)^k \mathbf{H}_0)$, then since cast operates elementwise
 635 and has bounded and finite range we have that there exists $\tau \in \mathbb{N}$, $\widehat{\mathbf{C}} \in \mathbb{C}^{n \times d}$ and $\widehat{\mathbf{D}} \in \mathbb{C}^{n \times d}$ such
 636 that for every $k \geq \tau$, $\widehat{\mathbf{C}}_k = \widehat{\mathbf{C}}$ and $\widehat{\mathbf{D}}_k = \widehat{\mathbf{D}}$ and

$$\widehat{\mathbf{H}}_k = \text{cast} \left(\sum_{i=0}^{\bar{k}-1} \widehat{\mathbf{C}}_i + \widehat{\mathbf{D}} + (k - \bar{k} + 1) \widehat{\mathbf{C}} \right).$$

637 Note that only the last term inside cast varies with k and in particular each element of the matrix
 638 inside cast converges to a point in $\overline{\mathbb{C}}$, that is the union of \mathbb{C} and the complex infinity. Therefore,
 639 since we are applying again the cast operation we obtain that there exists $\overline{\mathbf{H}} \in \mathbb{C}^{n \times d}$ and $\bar{k} \geq \tau$
 640 such that for every $k \geq \bar{k}$ we have $\widehat{\mathbf{H}}_k = \overline{\mathbf{H}}$, which concludes the proof. \square

641 **C.2 Proof of Theorem 2**

642 Let $\widehat{\mathbf{H}}_k$ and $\hat{y}_k := \text{cast}(\text{dec}(\widehat{\mathbf{H}}_k, x_k))$ be the finite-precision versions of the state \mathbf{H}_k and (scalar)
 643 output of a one-layer LRNN on the input $\mathbf{x} = x_1 \dots x_k = 1^k$. Let also $y_k = \mathbf{1}\{k \bmod m = 0\}$
 644 be the correct output recognizing the word. We will show that if the assumptions on the eigenvalues
 645 are not satisfied, there exist $\overline{\mathbf{H}}_1, \overline{\mathbf{H}}_2 \in \mathbb{C}^{n \times n}$, $\bar{y}_1, \bar{y}_2 \in \mathbb{R}^p$ and $\tau \in \mathbb{N}$ such that for all $k \geq \tau$

$$\widehat{\mathbf{H}}_k := \begin{cases} \overline{\mathbf{H}}_1 & \text{if } k \bmod 2 = 0 \\ \overline{\mathbf{H}}_2 & \text{otherwise} \end{cases}, \quad \hat{y}_k = \begin{cases} \bar{y}_1 & \text{if } k \bmod 2 = 0 \\ \bar{y}_2 & \text{otherwise.} \end{cases} \quad (6)$$

646 where without loss of generality we take $\bar{y}_1, \bar{y}_2 \in \{0, 1\}$. If $\bar{y}_1 = \bar{y}_2$, then, similarly to parity,
 647 $\hat{y}_k = \hat{y}_{k+1}$ for all $k > \tau$, while if $k \bmod m = m - 1$, then $1 = y_{k+1} \neq y_k = 0$. Otherwise if
 648 $\bar{y}_1 \neq \bar{y}_2$ then if we assume that $k \bmod d = 1$ and $\hat{y}_k = y_k = 0$, then $1 = \hat{y}_{k+1} \neq y_{k+1} = 0$ since
 649 $m > 2$. Therefore proving the result for a one-layer LRNN. Then we will use an inductive argument
 650 for the case of finitely many layers.

651 The proof can proceed similar to Theorem 1. Indeed, using the k -th power formula for the Jordan
 652 Decomposition of the matrix $\mathbf{A}(1)$ with eigenvalues $\lambda_1, \dots, \lambda_s$ we can prove that if $1 \leq i \leq s$,
 653 $|\lambda_i| < 1$ or $\lambda_i \in \mathbb{R}$ and $\lambda_i \geq 1$, then when $k \rightarrow \infty$ each element of the corresponding Jordan block
 654 of $\mathbf{A}(1)^k$ either converges to a single value or diverges to $+\infty$. If instead $\lambda_i \in \mathbb{R}$ and $\lambda_i \leq -1$,
 655 the diagonal element of the corresponding jordan block takes the form $c_k = (-1)^k |\lambda_i|^k$, while the
 656 ones in above the diagonal diagonal take the form $z_k = \binom{k}{j} (-1)^{k-t} |\lambda_i|^{k-t}$ with $t, j \leq n$. It can be
 657 shown that if we let $\bar{c} \in \{1, \infty\}$, then

$$\lim_{k \rightarrow \infty} c_{2k} = \bar{c}, \quad \lim_{k \rightarrow \infty} c_{2k+1} = -\bar{c}, \quad \lim_{k \rightarrow \infty} z_{2k} = \infty, \quad \lim_{k \rightarrow \infty} z_{2k+1} = -\infty.$$

658 Therefore we can apply the same reasoning of Theorem 1 using the finite-precision assumption to
 659 show that there exist $\bar{\tau} \in \mathbb{N}$, $\overline{\mathbf{C}}_1, \overline{\mathbf{C}}_2, \overline{\mathbf{D}}_1, \overline{\mathbf{D}}_2 \in \mathbb{C}^{n \times d}$ such that for every $k \geq \tau$ we have

$$\widehat{\mathbf{C}}_k := \text{cast}(\mathbf{A}(1)^k \mathbf{B}) = \begin{cases} \overline{\mathbf{C}}_1 & \text{if } k \bmod 2 = 1 \\ \overline{\mathbf{C}}_2 & \text{if } k \bmod 2 = 0 \end{cases} \quad \widehat{\mathbf{D}}_k := \text{cast}(\mathbf{A}(1)^k \mathbf{H}_0) = \begin{cases} \overline{\mathbf{D}}_1 & \text{if } k \bmod 2 = 1 \\ \overline{\mathbf{D}}_2 & \text{if } k \bmod 2 = 0 \end{cases}$$

660 Finally if for simplicity we consider $\tau \bmod 2 = 0$, we have that for $2k \geq \tau$

$$\widehat{\mathbf{H}}_{2k} = \text{cast} \left(\sum_{i=1}^{\tau-1} \widehat{\mathbf{C}}_i + \left(k - \frac{\tau}{2} + 1\right) \overline{\mathbf{C}}_2 + \left(k - \frac{\tau}{2}\right) \overline{\mathbf{C}}_1 + k \overline{\mathbf{D}}_2 \right)$$

$$\widehat{\mathbf{H}}_{2k+1} = \text{cast} \left(\sum_{i=1}^{\tau-1} \widehat{\mathbf{C}}_i + \left(k - \frac{\tau}{2} + 1\right) (\overline{\mathbf{C}}_2 + \overline{\mathbf{C}}_1) + k \overline{\mathbf{D}}_1 \right)$$

661 where we note that the limit for $k \rightarrow \infty$ of the term inside cast is well defined. Thus there exist
 662 $\overline{\mathbf{H}}_1, \overline{\mathbf{H}}_2 \in \mathbb{C}^{n \times d}$ and $\bar{k} \geq \tau$ such that eq. (6) is satisfied, concluding the proof for the case of a
 663 single layer.

664 **Multiple Layers** Note that since for one layer we have two sequences (even and odd) of outputs
 665 converging in finite time, there exist $\mathbf{a}, \mathbf{b} \in \mathbb{R}^p$ such that for all $k \geq \bar{k}$ we have

$$\hat{\mathbf{y}}_{2k} = \mathbf{a}, \quad \hat{\mathbf{y}}_{2k+1} = \mathbf{b}.$$

666 Therefore, consider an additional layer that takes as input $\mathbf{x}_1^{(2)}, \dots, \mathbf{x}_k^{(2)}$, with $\mathbf{x}_i^{(2)} = \hat{\mathbf{y}}_i$ and outputs
 667 $\hat{\mathbf{y}}_1^{(2)}, \dots, \hat{\mathbf{y}}_k^{(2)}$ as

$$\mathbf{H}_k^{(2)} = \mathbf{A}^{(2)}(\mathbf{x}_k^{(2)}) \mathbf{H}_{k-1}^{(2)} + \mathbf{B}^{(2)}(\mathbf{x}_k^{(2)}), \quad \hat{\mathbf{y}}_k^{(2)} = \text{dec}^{(2)}(\mathbf{H}_k^{(2)}, \mathbf{x}_k^{(2)})$$

668 without loss of generality assume for simplicity that $\bar{k} = 1$ and that $\hat{\mathbf{y}}_{2k}^{(2)} = \mathbf{a}$ and $\hat{\mathbf{y}}_{2k+1}^{(2)} = \mathbf{b}$. We
 669 also set $\mathbf{A}_1 = \mathbf{A}^{(2)}(\mathbf{a})$, $\mathbf{A}_2 = \mathbf{A}^{(2)}(\mathbf{b})$ and $\mathbf{B}_1 = \mathbf{B}^{(2)}(\mathbf{a})$, $\mathbf{B}_2 = \mathbf{B}^{(2)}(\mathbf{b})$ and $\mathbf{C}_1 = \mathbf{A}_1 \mathbf{A}_2$,
 670 $\mathbf{C}_2 = \mathbf{A}_1 \mathbf{B}_2 + \mathbf{B}_1$. Then we can write

$$\mathbf{H}_{2k}^{(2)} = \mathbf{A}_1 \mathbf{H}_{2k-1}^{(2)} + \mathbf{B}_1 = \mathbf{A}_1 \mathbf{A}_2 \mathbf{H}_{2k-2}^{(2)} + \mathbf{A}_1 \mathbf{B}_2 + \mathbf{B}_1$$

$$\mathbf{H}_{2k}^{(2)} = \mathbf{C}_1 \mathbf{H}_{2(k-1)}^{(2)} + \mathbf{C}_2 = \sum_{i=0}^{k-1} \mathbf{C}_1^i \mathbf{C}_2 + \mathbf{C}_1^k \mathbf{H}_0$$

671 Furthermore for the odd sequences of states we have

$$\mathbf{H}_{2k+1}^{(2)} = \mathbf{A}_2 \mathbf{H}_{2k}^{(2)} + \mathbf{B}_2 = \sum_{i=0}^{k-1} \mathbf{A}_2 \mathbf{C}_1^i \mathbf{C}_2 + \mathbf{C}_1^k \mathbf{H}_0 + \mathbf{B}_2.$$

672 We notice that the sequences $\mathbf{H}_{2k}^{(2)}$ and $\mathbf{H}_{2k+1}^{(2)}$ are in a form similar to \mathbf{H}_k of the first layer and when
 673 allowing for real but possibly negative eigenvalues we can use the same reasoning using the powers
 674 of the Jordan canonical form to show that if we let $\widehat{\mathbf{H}}_{2k}^{(2)}$ and $\widehat{\mathbf{H}}_{2k+1}^{(2)}$ being their finite-precision
 675 counterparts, then there exist $\overline{\mathbf{H}}_1^{(2)}, \overline{\mathbf{H}}_2^{(2)}, \overline{\mathbf{H}}_3^{(2)}, \overline{\mathbf{H}}_4^{(2)} \in \mathbb{R}^{n \times d}$, $\bar{k}_2 \geq 0$ such that for every $k \geq \bar{k}$

$$\widehat{\mathbf{H}}_{2k}^{(2)} = \begin{cases} \overline{\mathbf{H}}_1^{(2)} & \text{if } 2k \bmod 2 = 0 \\ \overline{\mathbf{H}}_2^{(2)} & \text{if } 2k \bmod 2 = 1 \end{cases}, \quad \widehat{\mathbf{H}}_{2k+1}^{(2)} = \begin{cases} \overline{\mathbf{H}}_3^{(2)} & \text{if } (2k+1) \bmod 2 = 0 \\ \overline{\mathbf{H}}_4^{(2)} & \text{if } (2k+1) \bmod 2 = 1 \end{cases}.$$

676 Therefore, for $k \geq \bar{k}_2$, the the function $k \mapsto \overline{\mathbf{H}}_k^{(2)}$ will be periodic with period four and hence no
 677 matter the choice of $\text{dec}^{(2)}$, also the function $k \mapsto \hat{\mathbf{y}}_k^{(2)}$ will be periodic with period 4. Consequently,
 678 with two layers one can recognize the language $(1^m)^*$ only when $m = 1, m = 2, m = 4$, since that
 679 is the only case where $k \mapsto y_k$ has a period which is a divisor of 4. We can extend this argument
 680 inductively to the case of an LRNN with L layers, to say that there exists $\bar{k}_L \geq 0$ such that for every
 681 $k \geq \bar{k}_L$, if we let $\mathbf{y}_k^{(L)}$ be the output of the last layer, the function $k \mapsto \hat{\mathbf{y}}_k^{(L)}$ is periodic with period
 682 2^L and thus it can recognize the language $(1^m)^*$ only when $2^L \bmod m = 0$, which happens only
 683 when there exists $p \leq L$ such that $m = 2^p$ and hence m is a power of two, ending the proof. \square

684 D Products of Generalized Householder Matrices – Proofs

685 We provide proofs for the results stated in Section 3.3.

686 D.1 Proof of Proposition 1

687 **First item** It can be shown by noting that if $\mathbf{C} \in \mathcal{M}_1([-1, 1])$, then $\|\mathbf{C}\| \leq 1$ and using the
 688 sub-multiplicative property of the euclidean norm, i.e the fact that $\|\mathbf{A}\mathbf{B}\| \leq \|\mathbf{A}\|\|\mathbf{B}\|$.

689 **Second item** Note that any real matrix has a singular value decomposition. Hence we can write

$$\mathbf{M} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$$

690 with $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n \times n}$ orthogonal and $\mathbf{S} = \text{Diag}(\sigma_1, \dots, \sigma_n)$ with $\sigma_i \in [0, 1]$, since $\|\mathbf{M}\| \leq 1$.
 691 It follows from the n -reflections theorem³ that we can write \mathbf{U} and \mathbf{V} as either the identity $\mathbf{I} \in$
 692 $\mathcal{M}_1(\{1\})$ or the product of at most n reflections, each of which is in $\mathcal{M}_1(\{-1\})$. Hence $\mathbf{U}, \mathbf{V} \in$
 693 $\mathcal{M}_n(\{-1, 1\})$. We can also write the matrix \mathbf{S} as the product of n GH matrices as

$$\mathbf{S} = \mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_n, \quad \mathbf{S}_i = \mathbf{I} - (1 - \sigma_i) \mathbf{e}_i \mathbf{e}_i^\top$$

694 where \mathbf{e}_i is the i -th element of the canonical basis of \mathbb{R}^n . Hence, $\mathbf{S} \in \mathcal{M}_n([0, 1])$. The proof of the
 695 first part is concluded since we wrote each of $\mathbf{U}, \mathbf{S}, \mathbf{V}$ as a product of at most n GH matrices. If \mathbf{M} is
 696 orthogonal we apply the n -reflections theorem directly. While we note that if $\mathbf{M} = \mathbf{P} \in \{0, 1\}^{n \times n}$
 697 with \mathbf{P} being a permutation matrix different from the identity, it can be written as products of at
 698 most $n - 1$ swaps, i.e. permutation matrices permuting only two elements. Therefore we have that
 699 there exists an integer $k \leq n - 1$ and indices i_1, \dots, i_k and j_1, \dots, j_k such that $i_l \neq j_l$ and

$$\mathbf{P} = \prod_{l=1}^{k-1} \mathbf{P}_{i_l j_l}, \quad \mathbf{P}_{ij} = (\mathbf{I} - 2\mathbf{v}_{ij} \mathbf{v}_{ij}^\top) \quad \mathbf{v}_{ijl} = \begin{cases} 1/\sqrt{2} & \text{if } l = i \\ -1/\sqrt{2} & \text{if } l = j \\ 0 & \text{otherwise} \end{cases},$$

700 where we set $\mathbf{v}_{ij} = (v_{ij1}, \dots, v_{ijn})$. Note that since $\|\mathbf{v}_{ij}\| = 1$, $\mathbf{P}_{ij} \in \mathcal{M}_k(\{-1\})$ with $k \leq n$. For
 701 the the case where $\mathbf{M} = \mathbf{I}$ we can use the fact that $\mathbf{I} \in \mathcal{M}_1(\{1\})$

³This is a specialization of the Cartan–Dieudonné Theorem to \mathbb{R}^n , see Theorem 3 in <https://faculty.um1.edu/dklain/orthogonal.pdf> for a proof.

702 **Third item** Let $N = C_1 C_2 \cdots C_k \in \mathcal{M}_k((-1, 1])$, with $C_i = I - \beta_i \mathbf{k}_i \mathbf{k}_i^\top$ with $\|\mathbf{k}_i\| = 1$ and
703 $\beta_i \in [0, 2)$. If $N = I$ the statement is satisfied, otherwise, let $\mathcal{V} = \text{span}\{\mathbf{k}_i : i \in \{1, \dots, k\}, \beta_i >$
704 $0\}$. Any unit vector $\mathbf{v} \in \mathbb{R}^n$ can then be written as $\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2$ with $\mathbf{v}_1 \in \mathcal{V}$, $\mathbf{v}_2 \in \mathcal{V}^\top$ and
705 $\|\mathbf{v}_1\|, \|\mathbf{v}_2\| \leq 1$. Now, if $\mathbf{v}_1 = 0$, then $N\mathbf{v} = \mathbf{v}$, and hence \mathbf{v} is an eigenvector with eigenvalue
706 1. Instead, if $\mathbf{v}_1 \neq 0$, then there exists i' such that $\beta_{i'} \in (0, 2)$ and $\mathbf{v}^\top \mathbf{k}_{i'} = \mathbf{v}_1^\top \mathbf{k}_{i'} \in (0, 1]$ and if
707 $i' < k$ either $\beta_j = 0$ or $\mathbf{v}^\top \mathbf{k}_j = 0$ for all $j \in \{i' + 1, \dots, k\}$. Moreover, we have that

$$\|C_{i'} \mathbf{v}\|^2 = \|\mathbf{v} - \beta_{i'} \mathbf{k}_{i'} \mathbf{k}_{i'}^\top \mathbf{v}\|^2 = 1 - \beta_{i'}(2 - \beta_{i'}) (\mathbf{v}^\top \mathbf{k}_{i'})^2 < 1,$$

708 where the last line comes from the fact that $\min_{x \in [0, 2]} x(2 - x) = 1$ and is only reached at 0
709 and 2 while $\beta_{i'} \in (0, 2)$. Therefore, since for every i , $\|C_i\| \leq 1$ and the euclidean norm is sub-
710 multiplicative we have

$$\|N\mathbf{v}\| = \|C_1 C_2 \cdots C_k \mathbf{v}\| = \|C_1 C_2 \cdots C_{i'} \mathbf{v}\| \leq \|C_1\| \cdots \|C_{i'}\mathbf{v}\| < 1.$$

711 Therefore, if \mathbf{v} is also an eigenvector with eigenvalue $\lambda \in \mathbb{C}$, then $\|N\mathbf{v}\| = |\lambda| < 1$. Hence, we
712 proved that for every eigenvector with eigenvalue λ either $\lambda = 1$ or $|\lambda| < 1$. It remains to show
713 that all eigenvalues of $N \in \mathcal{M}_k([0, 1])$ are real. For $k = 1$ it follows due to N being symmetric,
714 for $k \geq 2$ let $D = C_1 C_2 \cdots C_{k-1}$ so that $N = DC_k$ and let \mathbf{v} be any eigenvector of N with
715 eigenvalue λ and $\|\mathbf{v}\| = 1$. Then it holds that

$$\mathbf{v}^\top C_k N \mathbf{v} = \lambda \mathbf{v}^\top C_k \mathbf{v}.$$

716 Therefore if $\mathbf{v}^\top C_k \mathbf{v} \neq 0$, then $\lambda = \mathbf{v}^\top C_k N \mathbf{v} / \mathbf{v}^\top C_k \mathbf{v} \in \mathbb{R}$. Otherwise when $\mathbf{v}^\top C_k \mathbf{v} = 0$ it
717 follows that

$$\mathbf{v}^\top C_k \mathbf{v} = \|\mathbf{v}\|^2 - \beta_k (\mathbf{k}_k^\top \mathbf{v})^2 = 1 - \beta_k (\mathbf{k}_k^\top \mathbf{v})^2 = 0,$$

718 which is true only if $\beta_k = 1$ and either $\mathbf{v} = \mathbf{k}_k$ or $\mathbf{v} = -\mathbf{k}_k$ and thus $C_k \mathbf{v} = \pm C_k \mathbf{k}_k = 0$ and
719 hence $\lambda = 0$, which concludes the proof. \square

720 D.2 Proof of Theorem 3

721 We first recall the notion of group isomorphism. Two groups $(G, *)$ and (H, \cdot) where G, H are
722 the sets and $*$ and \cdot are the associative operations, are isomorphic, if there exist a bijective map
723 $f : G \rightarrow H$ such that for every $g \in G, h \in H$

$$f(g * h) = f(g) \cdot f(h).$$

724 We view the LRNN layer in eq. (1) as the FSA $\mathcal{A}_{\text{lin}} = (\Sigma, \mathcal{H}, \mathbf{H}_0, \delta_{\text{lin}})$, where $\delta_{\text{lin}}(\mathbf{H}, w) =$
725 $\mathbf{A}(w)\mathbf{H} + \mathbf{B}(w)$, which is extended in the usual way, and $\mathcal{H} = \{\delta_{\text{lin}}(\mathbf{H}_0, \mathbf{w}) : \mathbf{w} \in \Sigma^*\}$. Since
726 $\mathcal{T}(\mathcal{A})$ is a group, from Cayley's theorem we have that it is isomorphic to a subgroup of S_n , which
727 is the set of permutations on a set of a set of n elements. Furthermore, each element in S_n can
728 be represented as an $n \times n$ permutation matrix. Since in general $n \neq |Q|$, we cannot let \mathcal{H} to
729 be a set of one hot vectors each corresponding to states in Q . Instead, we let $\mathbf{H}_0 = (1, \dots, n)^\top$,
730 $\mathcal{P} \subset \{0, 1\}^{n \times n}$ be the set of permutation matrices and define $\mathbf{B} \equiv 0$ and $\mathbf{A} : \Sigma \rightarrow \mathcal{P}$ to be the
731 function mapping each letter $w \in \Sigma$ to the permutation matrix corresponding to $\delta(\cdot, w)$. With this
732 choice we can see that the function $f : \mathcal{T}(\mathcal{A}_{\text{lin}}) \rightarrow \mathcal{T}(\mathcal{A})$ such that $f(\delta_{\text{lin}}(\cdot, \mathbf{w})) = \delta(\cdot, \mathbf{w})$ for
733 every $\mathbf{w} \in \Sigma^*$ is one-to-one (bijejective), and from our choice of \mathbf{H}_0 , also the map $h : \mathcal{T}(\mathcal{A}_{\text{lin}}) \rightarrow \mathcal{H}$
734 such that for every $\mathbf{w} \in \Sigma^*$, $h(\delta_{\text{lin}}(\cdot, \mathbf{w})) = \delta_{\text{lin}}(\mathbf{H}_0, \mathbf{w})$ is also bijective. Moreover, the map
735 $\phi : \mathcal{T}(\mathcal{A}) \rightarrow Q$ such that $\phi(\delta(\cdot, \mathbf{w})) = \delta(q_0, \mathbf{w})$ is surjective because we consider states that are
736 only reachable from q_0 , i.e. $Q = \{\delta(q_0, \mathbf{w}) : \mathbf{w} \in \Sigma^*\}$. Hence if we set $g = \phi \circ f \circ h^{-1}$, then
737 $g : \mathcal{H} \rightarrow Q$ is surjective and for every $w \in \Sigma$ and $\mathbf{H} \in \mathcal{H}$ we have that

$$g(\delta_{\text{lin}}(\mathbf{H}, w)) = \delta(g(\mathbf{H}), w)$$

738 Thus, we have shown that such LRNN implements \mathcal{A} and it does so with finite precision because
739 the entries of all vectors and matrices are bounded integers. The proof is concluded by noting that
740 permutation matrices have euclidean norm equal to one and real entries.

741 Moreover, Let $k = \max_{w \in \Sigma} \sum_{q \in Q} \mathbf{1}\{\delta(q, w) \neq q\} = \max_{w \in \Sigma} \sum_{i=1}^n \mathbf{1}\{(\mathbf{A}(w)\mathbf{H}_0)_i \neq \mathbf{H}_0, i\}$
742 be the maximum number of displaced element of the permutation associated with the alphabet Σ .
743 Then, we know that every $\mathbf{A}(w) \in \mathcal{M}_{k-1}(\{-1, 1\})$.

744 If in addition there exists $m \in \mathbb{N}$ such that $\mathcal{T}(\mathcal{A})$ is isomorphic to a subgroup of the cyclic group \mathbb{Z}_m
745 with elements $\{0, \dots, m-1\}$, we can modify the construction above to use a smaller dimension.
746 If $m = 2$, then \mathbb{Z}_2 has elements $\{0, 1\}$, and \mathcal{A} implements the parity automaton. Thus, we can
747 set $\mathbf{H}_0 = -1$, $\mathbf{A}(0) = 1$, $\mathbf{A}(1) = -1$ and $g(1) = 1$ while $g(0) = 1$, which means that we
748 can use a scalar recursion. Otherwise, if $m \geq 3$, we can modify the construction above by setting
749 $\mathbf{H}_0 = (1, 0)^\top$ and, if for simplicity we assume $\Sigma \in \{0, \dots, m-1\}$, for every $w \in \Sigma$ we let $\mathbf{A}(w)$
750 be the 2×2 rotation matrix corresponding to $\delta(\cdot, w)$:

$$\mathbf{A}(w) = \mathbf{R}(\theta) = \begin{bmatrix} \cos \theta_w & -\sin \theta_w \\ \sin \theta_w & \cos \theta_w \end{bmatrix}, \quad \theta_w = \frac{2\pi w}{m},$$

751 such that $\mathbf{R}(\theta) \in \mathcal{M}_2(\{-1\})$ (from Proposition 1). □

752 D.3 Krohn-Rhodes Theorem

753 Before presenting the proof for Theorem 4, we provide the statement for the landmark result of
754 Krohn-Rhodes (Krohn & Rhodes, 1965), after giving the definition of cascade product of two FSA.

755 **Definition 1** (Cascade product). *Given two FSA $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ and $\mathcal{B} = (Q \times \Sigma, Q', q'_0, \delta')$, we
756 define the cascade product FSA as $\mathcal{C} = \mathcal{B} \circ \mathcal{A} = (\Sigma, Q \times Q', (q_0, q'_0), \delta'')$ where for any $w \in \Sigma$*

$$\delta''((q, q'), w) := (\delta(q, w), \delta(q', (q, w)))$$

757 **Theorem 5** (Krohn-Rhodes, Theorem 4 in Maler & Pnueli (1994)). *For every FSA $\mathcal{A} =$
758 (Σ, Q, q_0, δ) there exists $s \leq 2^{|Q|}$ and a cascade product FSA $\mathcal{C} = \mathcal{A}^{(s)} \circ \dots \circ \mathcal{A}^{(1)} =$
759 $(\Sigma, Q^\times, q_0^\times, \delta^\times)$, with $\mathcal{A}^{(i)} = (\Sigma^{(i)}, Q^{(i)}, q_0^{(i)}, \delta^{(i)})$, with $|Q^{(i)}| \leq |Q|$, and a function $\mathcal{W} : Q^\times \rightarrow$
760 Q such that for any $\mathbf{w} \in \Sigma^*$, $\delta(q_0, \mathbf{w}) = \mathcal{W}(\delta^\times(q_0^\times, \mathbf{w}))$ and each $\mathcal{A}^{(i)}$ is permutation-reset au-
761 tomaton, which means that for every $w^{(i)} \in \Sigma^{(i)}$, $\delta^{(i)}(\cdot, w^{(i)})$ is either a bijection (i.e. a permutation
762 over Q) or constant, i.e. $\delta(\cdot, w^{(i)}) = q(w) \in Q^{(i)}$.*

763 D.4 Proof of Theorem 4

764 We apply the Krohn-Rhodes theorem (Theorem 5) to write \mathcal{A} as the cascade product FSA $\mathcal{C} =$
765 $\mathcal{A}^{(s)} \circ \dots \circ \mathcal{A}^{(1)}$ with each FSA $\mathcal{A}^{(i)} = (\Sigma^{(i)}, Q^{(i)}, q_0^{(i)}, \delta^{(i)})$ being permutation-reset and we show
766 how the LRNN can implement \mathcal{C} .

767 We now show how the i -th layer of the LRNN with the structure in 1 can implement $\mathcal{A}^{(i)}$.

768 Let $n = |Q^{(i)}|$ and without loss of generality assume that $\Sigma = \{1, 2, \dots, |\Sigma|\}$ and $Q^{(i)} =$
769 $\{1, 2, \dots, n\}$ with $q_0^{(i)} = 1$. For every $w \in \Sigma^{(i)}$ we set $\mathbf{A}^{(i)}(w) \in \{0, 1\}^{n \times n}$, $\mathbf{B}^{(i)}(w) \in \{0, 1\}^n$
770 such that $q, q' \in Q^{(i)}$

$$\begin{aligned} \mathbf{A}^{(i)}(w)_{q', q} &= \mathbf{1}\{\delta(q, w) = q'\}, & \mathbf{B}^{(i)}(w)_{q'} &= 0, & \text{if } \delta^{(i)}(\cdot, w) \text{ is bijective, or} \\ \mathbf{A}^{(i)}(w)_{q', q} &= 0, & \mathbf{B}^{(i)}(w)_{q'} &= \mathbf{1}\{q' = q(w)\}, & \text{if } \delta^{(i)}(\cdot, w) \equiv q(w). \end{aligned}$$

771 Then, for every word $\mathbf{w}^{(i)} = w_1^{(i)} \dots w_t^{(i)} \in \Sigma^{(i)*}$, we set $g : \mathbb{R}^n \rightarrow \mathbb{R}$, such that $g(x) =$
772 $(1, \dots, n)^\top x$ and

$$\begin{aligned} \mathbf{H}_t^{(i)} &= \mathbf{A}^{(i)}(w_t^{(i)})\mathbf{H}_{t-1}^{(i)} + \mathbf{B}^{(i)}(w_t^{(i)}), & \mathbf{H}_0^{(i)} &= (1, 0, \dots, 0)^\top \in \mathbb{R}^n \\ y_t^{(i)} &= \text{dec}^{(i)}(\mathbf{H}_t^{(i)}, w_t^{(i)}) = (g(\mathbf{H}_t^{(i)}), w_t^{(i)}) = (\delta^{(i)}(q_0^{(i)}, \mathbf{w}^{(i)}), w_t^{(i)}) \end{aligned}$$

773 So that such construction implements $\mathcal{A}^{(i)}$. In addition, by letting $\mathbf{w} = w_1 \dots w_t \in \Sigma^*$ be the input
774 to the LRNN, i.e. $w_j^{(1)} = w_j$, and setting the output of each layer as the input to the next, i.e.
775 $w_j^{(i)} = y_j^{(i-1)}$ for $i \geq 2$, for the output of the last layer we get

$$\begin{aligned} y_t^{(s)} &= \text{dec}^{(s)}(\mathbf{H}_t, w_t^{(s)}) \\ &= (\delta^{(s)}(q_0^{(s)}, \mathbf{w}^{(s)}), y_t^{(s-1)}) \\ &= (\delta^{(s)}(q_0^{(s)}, \mathbf{w}^{(s)}), \delta^{(s-1)}(q_0^{(s-1)}, \mathbf{w}^{(s-1)}), y_t^{(s-2)}) \\ &= (\delta^{(s)}(q_0^{(s)}, \mathbf{w}^{(s)}), \dots, \delta^{(1)}(q_0^{(1)}, \mathbf{w}), w_t) \in \mathbb{N}^{s+1}, \end{aligned}$$

776 where we removed the nested parenthesis for simplicity. Hence, the first s elements of $y_t^{(s)}$ are
 777 exactly the output of the cascade FSA \mathcal{C} . Note that our construction can be implemented in finite
 778 precision, since we only used matrices/vectors with entries either in $\{0, 1\}$, requiring only one bit,
 779 or in $Q^{(i)} \subset \mathbb{N}$, that can also be implemented using finite precision with $|Q^{(i)}|$ integers, requiring
 780 $\log_2(|Q^{(i)}|)$ bits. Note that we can exclude the last element of $y_t^{(s)}$ to get a dimension \mathbb{N}^s .

781 It is also the case that $\|\mathbf{A}^{(i)}(w)\| \leq 1$ for every $w \in \Sigma^{(i)}$ since $\mathbf{A}^{(i)}(w)$ is either a permutation
 782 matrix ($\|\mathbf{A}^{(i)}(w)\| = 1$) or the zero matrix ($\|\mathbf{A}^{(i)}(w)\| = 0$). Also, for every permutation matrix
 783 $\mathbf{P} \in \{0, 1\}^{n \times n}$ which permutes only $k \leq n$ elements we have that $\mathbf{P} \in \mathcal{M}_{k-1}(\{-1, 1\})$.

784 Furthermore, for the zero matrix we have

$$0 = \prod_{i=1}^n (I - e_i e_i^\top) \in \mathcal{M}_n(\{0\})$$

785 It follows that $\mathcal{A}^{(i)}(w) \in \mathcal{M}_n([-1, 1])$ for $i \in \{1, \dots, s\}$. □

786 E Experiments

787 E.1 Implementation

788 We build on the original code for Mamba⁴ and DeltaNet⁵. For DeltaNet, implementing the extended
 789 eigenvalue range is straightforward, since there is no need to modify the Triton kernel. However,
 790 Mamba requires modifications to the CUDA code of the associative scan for both forward and back-
 791 ward passes which however had no impact computational cost. We ensured the accuracy of the mod-
 792 ifications by comparing the results with a naive implementation using a for-loop. For initial testing
 793 of the extended eigenvalue range, we used the pure pytorch implementation of Mamba by Torres
 794 (2024).

795 **Products in Log-space** We note that some diagonal models such as Mamba2 (Dao & Gu, 2024),
 796 GLA Yang et al. (2024a), mLSTM Beck et al. (2024) take advantage of the fact that all values of the
 797 state-transition matrices are positive to compute their repeated products in log-space. Our change
 798 would not allow to do this directly, and early tests on the chunkwise parallel form of GLA showed
 799 degraded performance. Therefore, for this work, we decided to focus on Mamba and DeltaNet since
 800 they do not compute the products in log-space. We mention however, that at the cost of increased
 801 computation time, it would be possible to do products in log-space by converting each value in
 802 the diagonal state-transition matrix to the product of its absolute value and sign. This way, absolute
 803 values can be multiplied in log space, while products of signs is coincidentally equivalent to addition
 804 modulo 2, i.e. parity, and hence can be done stably. We leave the investigation of this approach to
 805 future work. Furthermore, we also believe that our change may be less suited to method which use
 806 a normalized RNN state, such as mLSTM.

807 E.2 Chomsky Hierarchy

808 Here, we provide details on the state-tracking tasks and experimental protocol of Section 4.1.

809 E.2.1 Details on the experimental setup

810 Like Beck et al. (2024), we trained each model with sequence lengths ranging from 3 to 40 and
 811 evaluated on lengths from 40 to 256, to understand the length generalization capabilities. We com-
 812 pared mLSTM and sLSTM with two models: Mamba (Gu & Dao, 2023) and DeltaRule (Yang et al.,
 813 2024b). All models contain 2 blocks, with 4 heads for the xLSTM and DeltaRule models. We set
 814 the embedding and heads' dimension to 128 across all experiments. For Mamba and DeltaRule,
 815 we also enable the 1-D depthwise-separable convolution layer with kernel size equal to 4 after the
 816 query/key/value projection. We train each model using AdamW (Loshchilov & Hutter, 2019) with-
 817 out gradient clipping, using 3 different learning rates (1e-2, 1e-3, 1e-4), with 3 different seeds each.

⁴<https://github.com/state-spaces/mamba>

⁵<https://github.com/sustcsonglin/flash-linear-attention>

Table 4: Performance comparison using lm-harness benchmark (Gao et al., 2024) (top row reproduced from Yang et al. (2024b), bottom ours). Results shown for original and extended eigenvalue range. Our models show comparable performance across tasks.

	Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc.n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-e acc.n ↑	Avg. ↑	SWDE cont. ↑	SQUAD cont. ↑	FDA cont. ↑
15B tokens SPJ	340M params												
	Transformer++	28.39	42.69	31.0	63.3	34.0	50.4	44.5	24.2	41.2	42.2	22.1	21.4
	Mamba [0, 1]	28.39	39.66	30.6	65.0	35.4	50.1	46.3	23.6	41.8	12.4	23.0	2.1
	GLA [0, 1]	29.47	45.53	31.3	65.1	33.8	51.6	44.4	24.6	41.8	24.0	24.7	7.3
	DeltaNet [0, 1]	28.24	37.37	32.1	64.8	34.3	52.2	45.8	23.5	42.1	26.4	28.9	12.8
5B tokens FineWeb	340M params												
	DeltaNet [0, 1]	28.71	42.63	28.5	67.5	40.4	51.3	46.8	24.5	43.2	34.3	30.0	10.5
	DeltaNet [-1, 1]	29.01	46.38	28.3	68.0	39.1	51.0	48.4	23.4	43.0	31.3	26.3	9.6
5B tokens Maastricht	370M params												
	Mamba [0, 1]	32.04	42.82	29.1	67.4	39.6	52.7	47.0	24.4	43.4	14.2	20.1	1.3

Table 5: Performance comparison of various recurrent models on regular and context-free language tasks. We report the median \pm median absolute deviation of 3 independent runs with different random seeds. Scores represent scaled accuracy, with 1.0 indicating perfect performance and 0.0 random guessing. The positive impact of allowing negative eigenvalues ($[-1, 1]$ range) versus restricting to positive eigenvalues ($[0, 1]$ range) is evident across different model architectures.

	Parity	Mod. Arithmetic (w/o brackets)	Mod. Arithmetic (w/ brackets)
mLSTM	0.018 \pm 0.035	0.093 \pm 0.028	0.097 \pm 0.022
sLSTM	1.000 \pm 0.000	0.130 \pm 0.004	0.082 \pm 0.003
Mamba [0, 1]	0.000 \pm 0.000	0.000 \pm 0.005	0.016 \pm 0.002
Mamba [-1, 1]	1.000 \pm 0.000	0.079 \pm 0.032	0.029 \pm 0.007
DeltaNet [0, 1]	0.010 \pm 0.005	0.126 \pm 0.002	0.174 \pm 0.008
DeltaNet [-1, 1]	0.999 \pm 0.006	0.422 \pm 0.189	0.212 \pm 0.008

818 We pick the best based on the median of the 3 seeds for every learning rate value. We use a batch
819 size of 1024 (except for sLSTM, where we use 512) and a cosine annealing learning rate sched-
820 ular (Loshchilov & Hutter, 2017) (minimum learning rate: 1e-6) after 10% warm-up steps. The
821 weight decay is set to 0.1 during training. We train on every task for 100k steps in total. At each
822 training step, we make sure to generate a valid random sample from the task at hand (see below).

823 E.2.2 Details on the evaluated tasks

824 In Section 4.1 we conducted empirical evaluations on 3 tasks –namely parity, modular arithmetic
825 without brackets and with brackets – from various levels of the Chomsky Hierarchy, as proposed
826 by Deletang et al. (2023) and similarly used in xLSTM (Beck et al., 2024). Details for each task are
827 given below, where $|\Sigma|$ is the vocabulary size and Acc_{rand} is the accuracy of random guessing:

- 828 • **Parity** ($|\Sigma| = 2$, $Acc_{rand} = 0.5$). The parity $y_t \in \{0, 1\}$ of a sequence of ones and zeros
829 $\mathbf{x} = x_1 \dots x_t \in \{0, 1\}^t$ is equal to 1 (resp. 0) if the total number of ones in the sequence is odd
830 (resp. even). It is equivalent to addition modulo 2, it can be computed by summing all previous
831 values and then using the modulo 2 function as $y_t = (\sum_{i=1}^t x_i) \bmod 2$.
- 832 • **Modular Arithmetic w/o Brackets** ($|\Sigma| = 10$, $Acc_{rand} = 1/(|\Sigma| - 5)$). Given a set of special to-
833 kens $\Sigma_s = \{+, -, *, =, [\text{PAD}]\}$ and a modulus $m \geq 1$, we compute the remainder $y_t = \mathbf{x} \bmod m$,
834 where $\mathbf{x} = x_1 \dots x_t \in \Sigma^t$ and $y_t \in \{1, \dots, m - 1\}$. Here, $\Sigma = \Sigma_s \cup \{0, \dots, m - 1\}$. In our
835 experiments $m = 5$. An example sequence is as follows:

$$2 - 3 - 3 * 2 = 3 [\text{PAD}]$$

- 836 • **Modular Arithmetic w/o Brackets** ($|\Sigma| = 12$, $Acc_{rand} = 1/(|\Sigma| - 7)$). Same definition as the
837 modular arithmetic without brackets with a set of special tokens $\Sigma_s = \{+, -, *, =, (,), ([\text{PAD}])\}$. In
838 our experiments $m = 5$. An example sequence is as follows:

$$(((3 + 3) + -1) + -2) - ((3 - (-3)) + ((1) + 4)) = 2 [\text{PAD}]$$

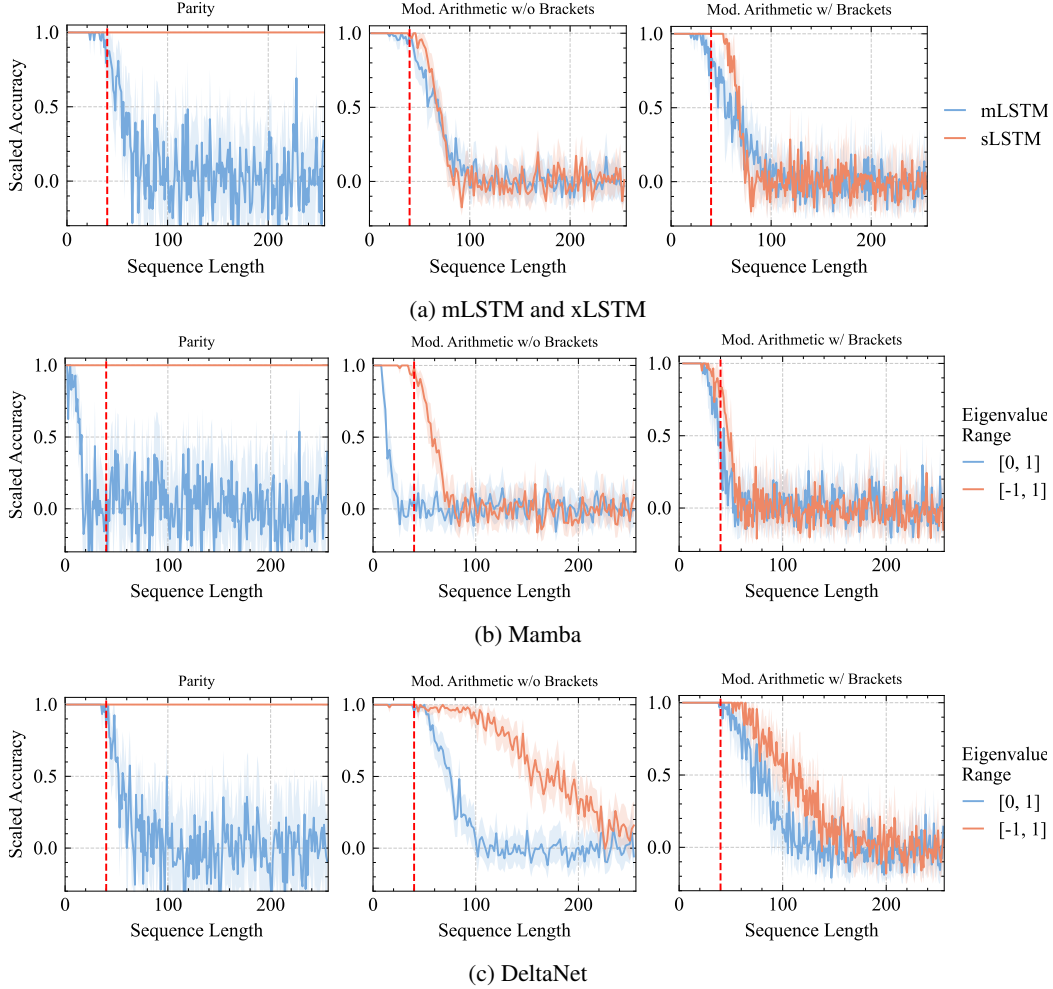


Figure 5: Performance (scaled accuracy) vs sequence length of *mLSTM*, *sLSTM*, *Mamba* and *DeltaNet* variants on different formal language tasks. Trained on sequences up to length 40 (dashed vertical red line). At test time, we sample uniformly at random 8192 sequences with lengths between 40 and 256. The curves show the mean and 95% CI.

839 E.3 State-Tracking

840 E.3.1 Details of the Experiments

841 For the experiments in Figure 3, we map each element of the group S_5 to an integer from 0 to 119,
 842 where 0 corresponds to the identity permutation, and then construct inputs and output sequences of
 843 integers x_1, \dots, x_t and y_1, \dots, y_t as follows

- 844 • **S_5** We sample x_i uniformly at random from $\{0, \dots, 119\}$. y_i is computed as the product
 845 of the permutations corresponding to x_1, \dots, x_i .
- 846 • **S_5 only swaps** As S_5 but x_i is sampled from the permutations that permute up to two
 847 elements (swaps and identity).
- 848 • **S_5 swaps, 3-permutations** As S_5 but x_i is sampled from the permutations that permute up
 849 to three elements.
- 850 • **S_5 4 tokens per transition** If $i \bmod 4 = 0$, then x_i is sampled uniformly at random from
 851 $\{0, \dots, 119\}$, otherwise $x_i = 120$ (special token). For $i > 3$, y_{i+3} is the product of the
 852 permutations corresponding to x_1, \dots, x_i , where 120 is treated as the identity permutation.
 853 $y_i = 0$ for $i \in \{1, 2, 3\}$.

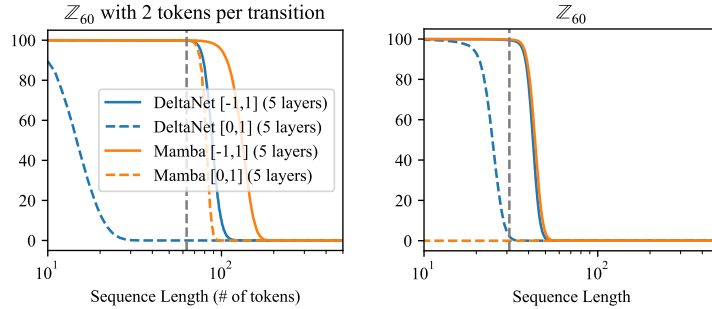


Figure 6: Validation sequence accuracy at different sequence lengths on the cyclic group \mathbb{Z}_{60} (1 seed). Dashed vertical lines indicate the sequence length used for training (left 32, right 64). Using 2 tokens per transition seems to help only marginally in this case. In this Mamba [-1,1] performed better than DeltaNet. The variants with eigenvalues in [0,1] performed worse.

854 For each input we also add a beginning of sequence token. For each setup we always sample 1.6M
 855 examples for training and 40K examples of length 500 for testing. We note that we are using a
 856 substantially larger training set compared to Merrill & Sabharwal (2023), to reduce the chances of
 857 overfitting.

858 We train all models using AdamW with weight decay 0.01, learning rate 0.0001, gradient clipping
 859 to 1.0 and a batch size of 512.

860 Both models use an embedding dimension of 128 and 4 heads for DeltaNet. In the case of DeltaNet
 861 we do not use the 1-D convolutions for these experiments. Other parameters are kept as defaults.

862 **Full Matrix Baseline.** For the full matrix baseline we use a single layer and map directly each
 863 token x_i to a learnable full state-transition matrix $A(x_i)$ via one-hot encoding. We then compute,
 864 for $i \in \{1, \dots, t\}$ the recursion

$$H_i = A(x_i)H_{i-1}, \quad H_0 = I \in \mathbb{R}^{n \times n}$$

865 where n is set to 32 for efficiency reason (memory and compute time grow quickly with n). After
 866 that we flatten each H_i into a vector and apply first a projection on the unit ball and then a linear
 867 decoder to get the final outputs. Since this model uses a full matrix, with $n \geq 5$ it should be fully able
 868 to learn S_5 without restricting the permutation in input. However in some situations the performance
 869 degrade quickly after some length, probably due to the fact that the norm of the learned $A(x_i)$ is not
 870 close enough to one.

871 E.3.2 Cyclic Groups

872 We report in Figure 6 some experiments on group word problems with the group \mathbb{Z}_{60} . For this
 873 experiment we also consider the simplified version where each transition is encoded using 2 tokens.
 874 This is done as in the experiments of S_5 with 4 tokens, but using two tokens instead of 4. Using one
 875 additional token should allow DeltaNet [-1,1] with more layers to learn the rotations needed to solve
 876 the task. However, using more tokens does not seem to help in this case. Extending the eigenvalue
 877 range seems to help in both settings, although surprisingly, Mamba [-1,1], even though it has a
 878 diagonal state-transition matrix, seems to perform best. We conjecture that in this case, the models
 879 might learn the shortcut solutions, also because they do not generalize well to longer sequences.

880 E.4 Language Modeling

881 E.4.1 Details on the experimental setup

882 Each model is trained for 200,000 steps with a per-GPU batch size of 10, distributed across 8 Nvidia
 883 A100 GPUs, using a context length of 2048. For optimization, we use AdamW (Loshchilov &
 884 Hutter, 2019) with learning rates of 3.1e-3 for DeltaNet and 5e-4 for Mamba (higher rates led to
 885 training instability). The learning rate was adjusted using cosine annealing (Loshchilov & Hutter,
 886 2017) following a linear warm-up period of 5000 steps. We applied a weight decay of 0.1 throughout
 887 the training process.

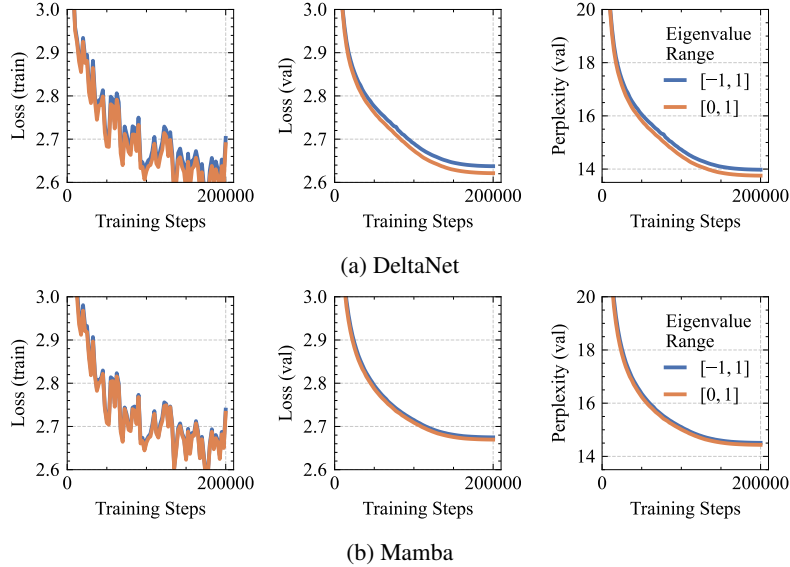


Figure 7: Learning curves of Mamba and DeltaNet when training on 32B tokens of Fine-Web 100B.

888 **E.4.2 Details on the evaluated tasks**

889 To produce the results in Table 4, we use the lm-harness benchmark (Gao et al., 2024), focusing on
 890 the same tasks as Yang et al. (2024b): LAMBADA (LMB) (Paperno et al., 2016), PIQA (Bisk et al.,
 891 2020), HellaSwag (Hella.) (Zellers et al., 2019), Winogrande (Wino.) (Sakaguchi et al., 2021), and
 892 ARC-easy (ARC-e) and ARC-challenge (ARC-c) (Clark et al., 2018). Additionally, we evaluate the
 893 performance on recall-intensive tasks (like Yang et al. (2024b)), including FDA (Arora et al., 2023),
 894 SWDE (Lockard et al., 2019), and SQUAD (Rajpurkar et al., 2018), to provide a comprehensive
 895 evaluation of our models’ capabilities.

896 **E.4.3 Additional results**

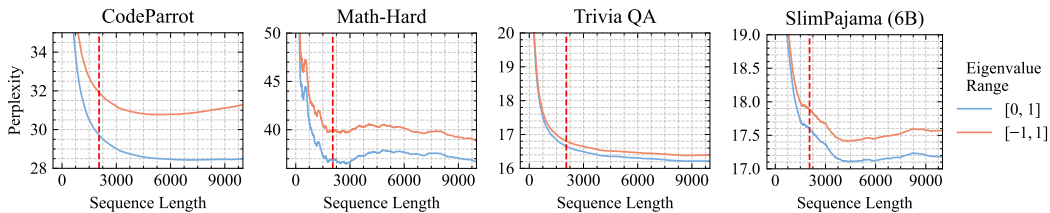


Figure 8: Length extrapolation performance of Mamba variants on different datasets. Mamba with eigenvalue range $[-1, 1]$ shows worse perplexity on coding and math tasks compared to the $[0, 1]$ baseline. The dashed, vertical line indicates the training context length of 2048 tokens.

897 **E.5 Implementation of Extended Eigenvalue Range**

```

220 if constexpr (!kIsComplex) {
221 - thread_data[i] = make_float2(exp2f(delta_vals[r][i] * A_val[r]),
222 + thread_data[i] = make_float2(2.0f * exp2f(delta_vals[r][i] * A_val[r]) - 1.0f,
223 !kIsVariableB ? delta_u_vals[r][i] : B_vals[i] * delta_u_vals[r][i]);
224     if constexpr (!Ktraits::kIsEvenLen) {
225         if (threadIdx.x * kNItems + i >= params.seqlen - chunk * kChunkSize) {
226             thread_data[i] = make_float2(1.f, 0.f);
227         }
228     }
229 }

```

Figure 9: Modifications to the forward pass of the Mamba associative scan . These changes extend the eigenvalue range from $[0, 1]$ to $[-1, 1]$, enhancing the model's expressive capacity. Adapted from selective_scan_fwd_kernel.cuh.

```

253 - const float delta_a_exp = exp2f(delta_vals[i] * A_scaled)
254 + const float delta_a_exp = 2.0f * exp2f(delta_vals[i] * A_scaled) - 1.0f

```

```

272 - typename Ktraits::BlockScanT(smem_scan).InclusiveScan(
273 + typename Ktraits::BlockScanT(smem_scan).ExclusiveScan(
274     thread_data, thread_data, SSMScanOp<weight_t>(), prefix_op
275 );

```

```

288 - const float a = thread_data[i].y - (!kIsVariableB ? delta_vals[i] * float(u_vals[i]) :
289 -     delta_vals[i] * float(u_vals[i]) * B_vals[i]);
290 + float delta_a_exp = 2.0f * exp2f(delta_vals[i] * A_scaled) - 1.0f;
291 + const float ddelta_a_exp = delta_a_exp + 1;
292 + const float a = ddelta_a_exp * thread_data[i].y;
293 + const float hi = delta_a_exp * thread_data[i].y + (!kIsVariableB ? delta_vals[i] *
294 +     float(u_vals[i]) : delta_vals[i] * float(u_vals[i]) * B_vals[i]);

```

```

288 - const float a = thread_data[i].y - (!kIsVariableB ? delta_vals[i] * float(u_vals[i]) :
289 -     delta_vals[i] * float(u_vals[i]) * B_vals[i]);
290 + float delta_a_exp = 2.0f * exp2f(delta_vals[i] * A_scaled) - 1.0f;
291 + const float ddelta_a_exp = delta_a_exp + 1;
292 + const float a = ddelta_a_exp * thread_data[i].y;
293 + const float hi = delta_a_exp * thread_data[i].y + (!kIsVariableB ? delta_vals[i] *
294 +     float(u_vals[i]) : delta_vals[i] * float(u_vals[i]) * B_vals[i]);

```

```

291 if constexpr (!kIsVariableB || !kIsVariableC) {
292     if constexpr (!kIsVariableB) { // dBC_val is dB_val
293 - dBC_val += dout_vals[i] * (!kIsVariableC ? thread_data[i].y : thread_data[i].y * C_vals[i]);
294 + dBC_val += dout_vals[i] * (!kIsVariableC ? hi : hi * C_vals[i]);
295     } else { // dBC_val is dC_val
296 - dBC_val += dout_vals[i] * thread_data[i].y;
297 + dBC_val += dout_vals[i] * thread_data[i].y;
298     }
299 }
300 if constexpr (kIsVariableB) { dB_vals[i] = dx * delta_vals[i] * float(u_vals[i]); }
301 if constexpr (kIsVariableC) {
302 - dC_vals[i] = dout_vals[i] * (!kIsVariableB ? thread_data[i].y * B_val : thread_data[i].y);
303 + dC_vals[i] = dout_vals[i] * (!kIsVariableB ? hi * B_val : hi);
304 }

```

Figure 10: Necessary changes to selective_scan_bwd_kernel.cuh

```

196     if self.use_beta:
197 - beta = rearrange(self.b.proj(hidden_states), 'b l h -> b h l').sigmoid()
198 + beta = 2 * rearrange(self.b.proj(hidden_states), 'b l h -> b h l').sigmoid()
199     else:
200         beta = q.new_ones(q.shape[0], q.shape[1], q.shape[2])

```

Figure 11: Simple modification to the beta calculation in DeltaNet (Source) allowing the extension of the eigenvalues to the range $[-1, 1]$. The original implementation (in red) is replaced with an adjusted version (in green).