DELTAPRODUCT: IMPROVING STATE-TRACKING IN LINEAR RNNS VIA HOUSEHOLDER PRODUCTS

Julien Siems[*◇], **Timur Carstensen**[*◇♣], **Arber Zela**[◇],
Frank Hutter[◇♣], **Massimiliano Pontil**[♡♠], **Riccardo Grazzi**[*†★]
Equal contribution[*], University of Freiburg[◇], ELLIS Institute Tübingen[♣], Microsoft Research[★]
CSML, Istituto Italiano di Tecnologia[♡], AI Centre, University College London[♠],
juliensiems@gmail.com timurcarstensen@gmail.com riccardograzzi4@gmail.com

ABSTRACT

Linear Recurrent Neural Networks (linear RNNs) have emerged as competitive alternatives to Transformers for sequence modeling, offering efficient training and linear-time inference. However, existing architectures face a fundamental trade-off between expressivity and efficiency, dictated by the structure of their state-transition matrices. While diagonal matrices used in architectures like Mamba, GLA, or mLSTM yield fast runtime, they suffer from severely limited expressivity. To address this, recent architectures such as (Gated) DeltaNet and RWKVv7 adopted a diagonal plus rank-1 structure, allowing simultaneous token-channel mixing, which overcomes some expressivity limitations with only a slight decrease in training efficiency. Building on the interpretation of DeltaNet's recurrence as performing one step of online gradient descent per token on an associative recall loss, we introduce DeltaProduct, which instead takes multiple (n_h) steps per token. This naturally leads to diagonal plus rank-n_h state-transition matrices, formed as products of n_h generalized Householder transformations, providing a tunable mechanism to balance expressivity and efficiency and a stable recurrence. Through extensive experiments, we demonstrate that DeltaProduct achieves superior state-tracking and language modeling capabilities while exhibiting significantly improved length extrapolation compared to DeltaNet. Additionally, we also strengthen the theoretical foundation of DeltaNet by proving that it can solve dihedral group word problems in just two layers.

1 INTRODUCTION

The Transformer architecture (Vaswani et al., 2017) has revolutionized natural language processing through its self-attention mechanism, enabling both parallel computation across sequence length and effective context retrieval. Despite outperforming traditional LSTM models (Hochreiter & Schmidhuber, 1997) across numerous tasks, Transformers' quadratic computational complexity with sequence length presents challenges when dealing with longer sequences. Linear RNNs have emerged as a promising solution that combines parallel training across sequence length with linear inference-time complexity. At the core of these models are the state-transition matrices governing the recurrence, which fundamentally determine their expressivity (Merrill et al., 2024). While early linear RNNs like S4 (Gu et al., 2022) or LRU (Orvieto et al., 2023) use token-independent state-transition matrices, current linear RNNs exclusively use token-dependent state-transition matrices due to their superior expressivity. The first generation of token-dependent linear RNNs, including Mamba (Gu & Dao, 2024; Dao & Gu, 2024), GLA (Yang et al., 2024a), and mLSTM (Beck et al., 2024), uses diagonal state-transition matrices for efficient sequence processing. Newer architectures have incorporated non-diagonal structures, often diagonal plus rank-1, enabling simultaneous mixing of information across both tokens and channels. This innovation has led to more expressive models such as (Gated) DeltaNet (Yang et al., 2024b; 2025), TTT-Linear (Sun et al., 2024), RWKV-7 (Peng et al., 2025), and Titans (Behrouz et al., 2024), which demonstrate superior language modeling and in-context retrieval performance, often with only a reasonable decrease in training efficiency.

[†]Work started while at Istituto Italiano di Tecnologia.

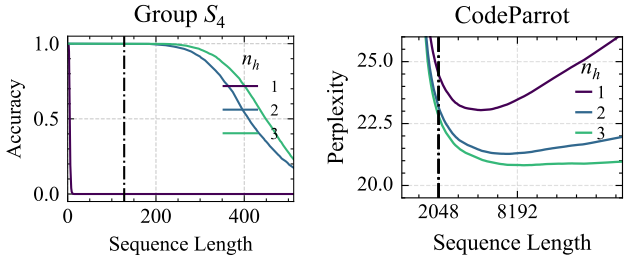

Figure 1: (*Left*) DeltaProduct$_{n_h}$ learns higher-order permutation groups like S_4 in one layer, while DeltaNet (n_h=1) is limited to S_2 (parity). (*Right*) Length extrapolation of DeltaProduct improves significantly with higher n_h.

Recent work has revealed a fundamental trade-off between training efficiency and expressivity of linear RNNs, dictated by the structure of their state-transition matrices (Merrill et al., 2024; Sarrof et al., 2024; Grazzi et al., 2025). Models with diagonal state-transition matrices, such as Mamba and GLA, are highly efficient to train but face severe expressivity limitations - for instance, they cannot perform addition modulo 3 on arbitrary length sequences in finite precision (Grazzi et al., 2025, Theorem 2). Also Transformers face similar limitations (Hahn, 2020; Merrill & Sabharwal, 2023), since they can be seen as special linear RNNs with state-transition matrix equal to the identity, albeit with an infinite dimensional state (Katharopoulos et al., 2020). DeltaNet partially overcomes these limitations through generalized Householder matrices, achieving greater expressivity with only a modest increase in training cost, though it still requires multiple layers for certain tasks. At the other extreme, linear RNNs with full state-transition matrices offer maximal expressivity (Cirone et al., 2024), capable of recognizing any regular language with a single layer (Merrill et al., 2024), but are prohibitively expensive to train.

To bridge this gap, we propose *DeltaProduct*, a method that balances expressivity and efficiency of the recurrence computation. While DeltaNet's recurrence performs a single gradient step per token on the squared loss of a linear key-to-value mapping (Wang et al., 2025; Yang et al., 2024b), DeltaProduct takes n_h gradient steps using additional keys and values, yielding state-transition matrices that are products of n_h generalized Householder matrices. This connection between the number of optimization steps and the matrix structure provides an elegant way to interpolate between diagonal and dense matrices: increasing the number of gradient steps automatically increases the number of Householder matrices in the product, providing a tunable mechanism to control the recurrence's expressivity. Additionally, this structure enables precise control over the norm of state transition matrices, ensuring they remain ≤ 1 to maintain stability during training on long sequences.

Concretely, we make the following contributions:

- We propose *(gated) DeltaProduct*, which generalizes (gated) DeltaNet by using products of generalized Householder transformations as state-transition matrices (Section 3).
- We prove that DeltaNet (DeltaProduct with $n_h = 1$) with 2 layers and an expanded eigenvalue range can solve word problems for dihedral groups (including S_3), extending prior analysis that was limited to cyclic groups (Grazzi et al., 2025, Theorem 6). This advances our understanding of DeltaNet's expressivity when multiple layers are used (Section 4).
- We empirically validate DeltaProduct's superior performance across multiple domains: solving complex state-tracking tasks beyond DeltaNet's capabilities (see Figure 1), achieving better results on Chomsky hierarchy benchmarks, and improving language modeling performance with significantly enhanced length extrapolation (Section 5).

2 Background & Related Work

2.1 Linear RNNs

Linear RNNs consist of stacked layers, each processing an input sequence of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$ (output of the previous layer) to produce an output sequence of vectors $\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_t$. We write the forward pass of each layer placing emphasis on the linear recurrence (as in Grazzi et al. (2025)) as

$$\boldsymbol{H}_i = \boldsymbol{A}(\boldsymbol{x}_i)\boldsymbol{H}_{i-1} + \boldsymbol{B}(\boldsymbol{x}_i), \qquad \hat{\boldsymbol{y}}_i = \mathrm{dec}(\boldsymbol{H}_i, \boldsymbol{x}_i), \qquad \text{where } i \in 1, \ldots, t \tag{1}$$

$\boldsymbol{H}_0 \in \mathbb{R}^{n \times d}$ is the initial hidden state, $\boldsymbol{A} : \mathbb{R}^l \to \mathbb{R}^{n \times n}$ maps the input to a state-transition matrix, $\boldsymbol{B} : \mathbb{R}^l \to \mathbb{R}^{n \times d}$, and $\mathrm{dec} : \mathbb{R}^{n \times d} \times \mathbb{R}^l \to \mathbb{R}^p$. The functions $\boldsymbol{A}$, $\boldsymbol{B}$, and dec are learnable, with dec

typically containing a feedforward neural network. Different linear RNN variants are distinguished by their specific implementations of these functions. For example, Mamba (Gu & Dao, 2024; Dao & Gu, 2024), GLA (Yang et al., 2024a), and mLSTM (Beck et al., 2024) use variations of a diagonal state-transition matrix $\boldsymbol{A}(\boldsymbol{x}_i)$. For a comparison of different linear RNN architectures see Yang et al. (2024b, Table 4). The linearity of the recurrence allows it to be parallelized along the sequence length, either via a chunkwise parallel form (Hua et al., 2022; Sun et al., 2023; Yang et al., 2025) or using a parallel scan (Blelloch, 1990; Martin & Cundy, 2018; Smith et al., 2023; Fan et al., 2024; Gu & Dao, 2024).

DeltaNet. We base our work on the DeltaNet architecture (Schlag et al., 2021a;b), which has recently attracted renewed interest through the work of Yang et al. (2024b; 2025) who demonstrated how to parallelize DeltaNet across the sequence length. The DeltaNet layer is parameterized as

$$\boldsymbol{A}(\boldsymbol{x}_t) = \boldsymbol{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top,\ \boldsymbol{B}(\boldsymbol{x}_t) = \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^\top,\ \mathrm{dec}(\boldsymbol{H}_t, \boldsymbol{x}_t) = \psi(\boldsymbol{H}_t^\top \boldsymbol{q}_t),$$

where $\beta_t = \mathrm{sigmoid}(\boldsymbol{w}_\beta^\top \boldsymbol{x}_t)$, $\boldsymbol{q}_t, \boldsymbol{k}_t \in \mathbb{R}^n$ (with $\|\boldsymbol{k}_t\| = 1$), $\boldsymbol{v}_t \in \mathbb{R}^d$ are output of learnable functions of $\boldsymbol{x}_t$ and $\boldsymbol{w}_\beta \in \mathbb{R}^l$ is a learnable parameter. DeltaNet's state-transition matrices are generalized Householder transformations. Unlike diagonal matrices which only mix tokens, these non-diagonal transformations enable token-channel mixings, significantly enhancing the model's expressivity compared to diagonal linear RNNs (Grazzi et al., 2025; Merrill et al., 2024; Peng et al., 2025). From a geometric perspective, the parameter β_t controls the type of transformation. For instance, $\beta_t = 0$ corresponds to the identity, $\beta_t = 1$ yields a projection operation, and $\beta_t = 2$ produces a reflection in the hyperplane with normal vector $\boldsymbol{k}_t$. DeltaNet also has a natural interpretation from an online learning perspective. As noted by Yang et al. (2024b); Wang et al. (2025); Liu et al. (2025), each step of the DeltaNet recurrence can also be viewed as one step of online gradient descent on a quadratic loss:

$$\mathcal{L}_t(\boldsymbol{H}) = \frac{1}{2}\|\boldsymbol{H}^\top \boldsymbol{k}_t - \boldsymbol{v}_t\|_2^2,\ \boldsymbol{H}_t = \boldsymbol{H}_{t-1} - \beta_t \nabla \mathcal{L}_t(\boldsymbol{H}_{t-1}) = \boldsymbol{H}_{t-1} - \beta_t\, \boldsymbol{k}_t \Big(\boldsymbol{k}_t^\top \boldsymbol{H}_{t-1} - \boldsymbol{v}_t^\top\Big) \quad (2)$$

State-Tracking. Recent work by Grazzi et al. (2025) demonstrates that expanding the eigenvalue range of linear RNNs' state transition matrices from $[0, 1]$ to $[-1, 1]$ significantly enhances their expressivity. For DeltaNet, this modification requires only a simple scaling of β_t by 2, enabling one layer to handle state-tracking tasks such as parity checking and, more generally, any *group word problem* where each element of the input sequence corresponds to a permutation of at most two elements, while for other state-tracking tasks, DeltaNet requires multiple layers (Grazzi et al., 2025, Theorem 2 and 6). A group word problem associated with a group G consists in mapping sequences of group elements $x_1, \ldots, x_t$ with $x_i \in G$ into sequences $y_1, \ldots, y_t$, where $y_i = x_1 \cdot x_2 \cdots x_i$ and $\cdot$ is the group operation. Group word problems are a way to model state-tacking tasks, and the one of the permutation group of 5 elements (S_5) is notoriously hard to solve for both Transformers and linear RNNs (Liu et al., 2023; Merrill & Sabharwal, 2023; Merrill et al., 2024).

2.2 Related Work

Linear RNNs have recently been studied from two main perspectives: state-space models and causal linear attention. State-space models, originating from continuous dynamical systems, inspired variants such as S4 (Gu et al., 2022), H4 (Fu et al., 2021), and LRU (Orvieto et al., 2023) (see Tiezzi et al. (2024) for a comprehensive survey). Models like Mamba (Gu & Dao, 2024; Dao & Gu, 2024) further enhance these by incorporating input-dependent gating mechanisms, significantly improving language modeling performance. In parallel, Katharopoulos et al. (2020) showed that causal linear attention Transformers can be reformulated as RNNs with linear sequence-length scaling. Following this, Gated Linear Attention (GLA) (Yang et al., 2024a) introduced gating mechanisms similar to Mamba. Recent studies explored more expressive recurrences via non-diagonal transition matrices, such as DeltaNet (Schlag et al., 2021a; Irie et al., 2023; Yang et al., 2024b), TTT-Linear (Sun et al., 2024), RWKV-7 (Peng et al., 2025), B'MOJO (Zancato et al., 2024), and Titans (Behrouz et al., 2024). Additionally, Beck et al. (2024) introduced xLSTM, combining linear and nonlinear RNN architectures inspired by LSTM (Hochreiter & Schmidhuber, 1997). Our work shares conceptual similarities with Adaptive Computation Time (ACT) (Graves, 2016), as both approaches allow RNNs to dynamically determine the computational steps required for each input, resulting in enhanced flexibility and task performance. This adaptive approach has been further developed in works like the Universal Transformer (Dehghani et al., 2019), with recent work by Geiping et al.

(2025) demonstrating its effectiveness in modern reasoning tasks. Concurrently to our work, Schöne et al. (2025) and Movahedi et al. (2025) have explored how fixed point iterations can increase the expressivity of linear RNNs. Unlike our approach, which enhances the expressivity by increasing the complexity of the linear recurrence, their approach works by applying the same recurrence multiple times, effectively increasing the depth of the model without increasing the parameter count. The two approaches are orthogonal and could be combined.

Products of structured matrices (Kissel & Diepold, 2023) have previously been used as state-transition matrices in non-linear RNNs—including (Givens) rotation matrices (Dorobantu et al., 2016; Jing et al., 2017), kronecker products (Jose et al., 2018), and householder reflections (Mhammedi et al., 2017)—chosen for their orthogonal, norm-preserving properties that encourage long-term dependency learning (Hochreiter, 1991; Bengio et al., 1994). Recently, Biegun et al. (2024) applied rotation matrices as state-transition matrices in non-selective state-space models. In contrast, DeltaProduct is more flexible, since we use products of generalized householder matrices, which can interpolate between identity, projection, or reflection transformations on a per token basis.

3 DeltaProduct

In this work, we propose *DeltaProduct*, a generalization of DeltaNet that enhances its expressivity by featuring state transition matrices formed as a product of generalized Householder matrices. While DeltaNet's recurrence can be seen as performing one step of online gradient descent per token, DeltaProduct further refines the hidden state update *multiple times per token*, naturally leading to a more expressive state-transition matrix, where each additional step expands the range of achievable linear transformations.

Formally, for each input token x_i to the layer we generate n_h keys as $\boldsymbol{k}_{i,j} = \boldsymbol{W}_j\boldsymbol{x}_i/\|\boldsymbol{W}_j\boldsymbol{x}_i\|_2$, n_h values as $\boldsymbol{v}_{i,j} = \boldsymbol{V}_j\boldsymbol{x}_i$, and n_h betas as $\beta_{i,j} = \phi(\boldsymbol{U}_j\boldsymbol{x}_i)$ where $\boldsymbol{W}_j, \boldsymbol{V}_j, \boldsymbol{U}_j$, are learnable weight matrices specific to the j-th gradient step, while ϕ is either the sigmoid or $2\times$ the sigmoid as suggested by Grazzi et al. (2025) to increase the expressivity and state-tracking capabilities. Then, we compute n_h gradient descent steps using the losses $\mathcal{L}_{i,j}(\boldsymbol{H}) = \|\boldsymbol{H}^\top\boldsymbol{k}_{i,j} - \boldsymbol{v}_{i,j}\|_2^2/2$, i.e. for $j = 1 \dots n_h$

$$\boldsymbol{H}_{i,j} = \boldsymbol{H}_{i,j-1} - \beta_{i,j}\nabla\mathcal{L}_{i,j}(\boldsymbol{H}_{i,j-1}) = \left(\boldsymbol{I} - \beta_{i,j}\boldsymbol{k}_{i,j}\boldsymbol{k}_{i,j}^\top\right)\boldsymbol{H}_{i,j-1} + \beta_{i,j}\boldsymbol{k}_{i,j}\boldsymbol{v}_{i,j}^\top,$$

where $\boldsymbol{H}_{i,0} = \boldsymbol{H}_{i-1}$ and $\boldsymbol{H}_{i,n_h} = \boldsymbol{H}_i$. Unrolling, we get $\boldsymbol{H}_i = \boldsymbol{A}(\boldsymbol{x}_i)\boldsymbol{H}_{i-1} + \boldsymbol{B}(\boldsymbol{x}_i)$ with

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

Hence, by taking multiple gradient descent steps per token, DeltaProduct's state-transition matrices are *products of generalized Householder transformations*, and by expanding such product, $\boldsymbol{A}(\boldsymbol{x}_i)$ takes the form of identity plus a matrix of rank at most n_h as shown in Figure 2. This formulation enables DeltaProduct to interpolate between generalized Householder ($n_h = 1$ as in DeltaNet) and dense matrices, since increasing n_h increases the rank of the update performed on the hidden state.

	Diagonal:	**Rank 1 Update:**	**Rank n_h Update:**
Token Mix:	✓	✓	✓
Channel Mix:	×	✓	✓
Expressivity:	Parity	Reflections	Reflections, Rotations
Examples:	Mamba, GLA	DeltaNet, TTT, RWKV-7	**DeltaProduct**

Figure 2: Overview of state-transition matrices in linear RNNs: Diagonal matrices (e.g., in Mamba/mLSTM) mix tokens only; rank-1 updates (DeltaNet/TTT) mix tokens and channels; and our proposed DeltaProduct performs rank-n_h updates, further increasing the expressivity.

An important consequence of using Householder products is that it allows us to efficiently bound the norm of $\boldsymbol{A}(\boldsymbol{x}_i)$, since the norm of the product is upper bounded by the product of the norms (each ≤ 1), ensuring the stability of the recurrence. This bound would not be possible with a the direct formulation $\boldsymbol{A}(\boldsymbol{x}_i) = I - \sum_{j=1}^{n_h} \beta_{i,j} \boldsymbol{k}_{i,j} \boldsymbol{k}_{i,j}^\top$, which would also restrict the matrix to be symmetric. Notably, for each layer, DeltaProduct uses n_h distinct key projection matrices to generate different keys $\boldsymbol{k}_{i,1}, \ldots, \boldsymbol{k}_{i,n_h}$ for the same input token. This ability to generate multiple, potentially orthogonal keys is essential for enhancing the recurrence's expressivity beyond DeltaNet. For instance, if we were to use identical keys across steps, the product of generalized Householders would collapse into a single generalized Householder transformation with a scaled β (see Lemma 1). Just as DeltaNet extends to Gated DeltaNet by incorporating a forget gate (Yang et al., 2025), DeltaProduct can similarly be extended to Gated DeltaProduct (see Appendix B for details).

Implementation. Since each step of DeltaProduct follows the same recurrence structure as DeltaNet, we can reuse its recurrence implementation written in Triton (Tillet et al., 2019), which is available through the FLASH-LINEAR-ATTENTION library (Yang & Zhang, 2024). However, DeltaProduct differs by using n_h keys and values per token, resulting in a recurrence n_h times longer than DeltaNet's. For a sequence of length l with n_h Householders, the keys (and similarly the values and betas) are arranged as: $[\mathbf{k}_{1,1}, \ldots, \mathbf{k}_{1,n_h}, \mathbf{k}_{2,1}, \ldots, \mathbf{k}_{2,n_h}, \ldots]$. For gating, we multiply a scalar gate to the state transition matrix, as in Gated DeltaNet (Yang et al., 2025), using a single gate $g_i \in \mathbb{R}$ per token $\mathbf{x}_i$, structured as: $[g_1, 1, \ldots, 1, g_2, 1, \ldots, 1, \ldots]$ where each g_i is followed by $(n_h - 1)$ ones to match the number of keys and values. Once the recurrence is evaluated, we keep only every n_h-th element of the output, so that the output sequence retains the same length as the input sequence. Note that the runtime of DeltaProduct scales linearly with n_h as demonstrated in Appendix C.

Theoretical Expressivity. As shown by Grazzi et al. (2025, Theorem 3 and 4), a linear RNN with state-transition matrices parameterized as the product of n_h generalized Householder transformations (whose eigenvalues lie in $[-1, 1]$) can solve any group word problem with state-transitions limited to permutations of at most $n_h + 1$ elements in one layer. Moreover, multiple layers allow it to recognize any regular language with sufficiently large n_h. Hence, by increasing n_h, DeltaProduct provides a tunable mechanism to control the expressivity of the recurrence, making it particularly effective for tasks requiring complex state tracking. Moreover, since each Householder transform is weighted by a coefficient $\beta_{i,j}$, the model can learn to set specific $\beta_{i,j}$ values to zero when processing certain tokens. This adaptively "skips" one or more gradient steps, effectively allowing the network to modulate compute on a token-by-token basis, thereby providing a route toward dynamic computation, reminiscent of ACT (Graves, 2016).

4 TWO LAYER DELTANET CAN SOLVE DIHEDRAL GROUP WORD PROBLEMS

In contrast to increasing the number of gradient steps per token, the expressivity of DeltaNet (equivalent to DeltaProduct with $n_h = 1$) can also be enhanced by increasing the number of layers and its theoretical limit is still unknown. In Grazzi et al. (2025, Theorem 6) it is shown that with 2 layers and the extended eigenvalue range, DeltaNet can compute addition modulo m, which corresponds to solving the group word problem for the cyclic group $\mathbb{Z}_m$, for any $m \in \mathbb{N}$.

We extend Grazzi et al. (2025, Theorem 6) to prove that, under identical assumptions, DeltaNet can solve the group word problem for the dihedral group D_m, for any $m \in \mathbb{N}$. The dihedral group D_m represents the symmetries (both rotations and reflections) of a regular m-sided polygon. As a notable example, D_3 is isomorphic to the symmetric group S_3. The linear RNN construction used in this result can be implemented using a 2-layer DeltaNet Model with two heads in the first layer.

Theorem 1 (Dihedral group word problems with reflections)**.** *A finite precision linear RNN with two layers in the form (1), where in the first layer $\boldsymbol{A}(\boldsymbol{x}_t) = \mathrm{diag}(\boldsymbol{a}(\boldsymbol{x}_t))$, with $\boldsymbol{a}(\boldsymbol{x}_t) \in \{1, -1\}^2$ and in the second layer $\boldsymbol{A}(\boldsymbol{x}_t) \in \mathcal{H} \subset \mathbb{R}^{2\times 2}$ where $\mathcal{H} = \{I - 2\boldsymbol{v}\boldsymbol{v}^\top : \boldsymbol{v} \in \mathbb{R}^2, \|\boldsymbol{v}\| = 1\}$ is the set of all 2D reflections, can solve the group world problem of the dihedral group D_m for any $m \in N$.*

The complete proof is provided in Appendix D and uses the following construction. In the first layer, the linear RNN will compute parity for rotations and reflections separately, i.e. it will record if the number of past rotations (reflections) is even or odd. The recurrent state of the second layer will have $2m$ possible values (same as the order of D_m) and each will be decoded differently based on the parity of reflections. The parity of rotations, combined with the group element, determines which reflection matrix to use as the state transition matrix of the second layer.

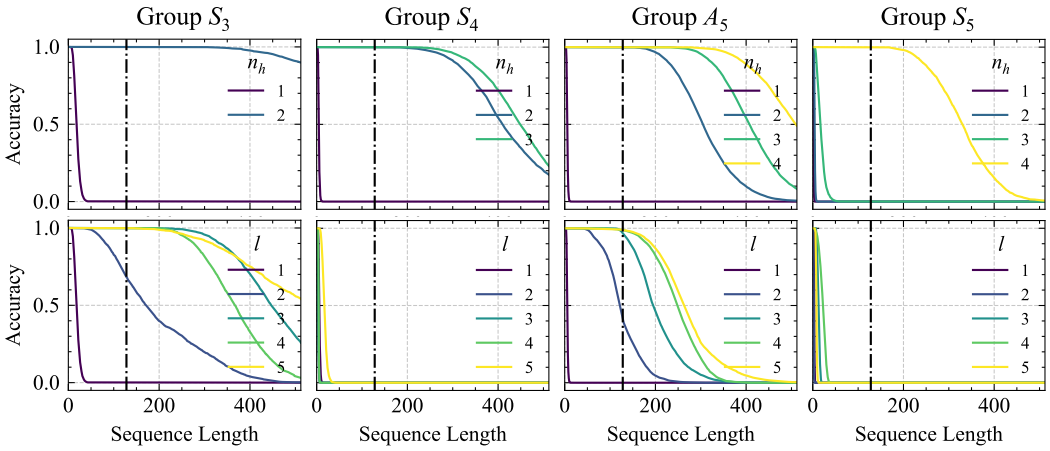

Figure 3: Accuracy on state-tracking tasks for permutation groups S_3, S_4, A_5, and S_5, plotted against sequence length (x-axis). *(Top row)* Varying the number of Householder products n_h for a single layer DeltaProduct$_{n_h}[-1, 1]$. *(Bottom row)* Varying the number of layers l of DeltaProduct$_1[-1, 1]$/DeltaNet$[-1, 1]$ (single Householder). Dashed vertical line at training context length 128. Higher n_h improves extrapolation to longer sequences of permutations, e.g., S_3 can be learned with $n_h = 2$ with a single layer while three layers are required when keeping $n_h = 1$.

5 Experiments

We evaluate DeltaProduct on a range of tasks—from state-tracking and chomsky hierarchy problems to standard language modeling—to assess its expressivity and efficiency. In each experiment, unless otherwise specified, we vary the number of Householder transformations per token (n_h) while keeping other parameters fixed, thereby trading-off increased computational cost and parameter count for enhanced expressivity. Throughout the experiments we use either the suffix $[-1, 1]$ or $[0, 1]$ after each method, to denote the eigenvalue ranges of its state transition matrices.

5.1 State-Tracking

Setup. We evaluate DeltaProduct's ability to capture complex state dynamics using group word problems of increasing difficulty, specifically on the permutation groups S_3, S_4, A_5, and S_5, as implemented by Merrill et al. (2024). We train on sequences of up to 128 products of permutations/tokens in length and subsequently measure extrapolation on sequences of up to 512 tokens. Throughout, we use the extended setting, allowing eigenvalues in $[-1, 1]$. DeltaProduct models failed to learn even the training context-length when restricted to the standard eigenvalue range $[0, 1]$, regardless of the number of Householder transformations n_h and so we omit the results. See Appendix E.1 for details on the experimental setup.

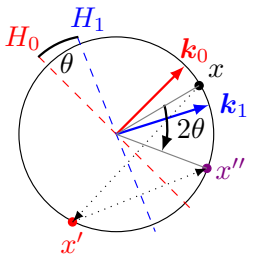

Figure 4: Two reflections produce a rotation: Reflecting x across planes H_0 and H_1 (with normals $\boldsymbol{k}_0$ and $\boldsymbol{k}_1$) yields a rotation by 2θ, where θ is the angle between the planes.

Results. Figure 3 (top row) demonstrates the benefits of increasing the number of gradient steps n_h per token for a single layer DeltaProduct. In agreement with Grazzi et al. (2025, Theorem 3), for S_3, achieving reliable performance beyond sequence lengths of 128 requires $n_h = 2$, while S_5 needs $n_h = 4$. Unexpectedly, S_4 and A_5 can extrapolate robustly using only $n_h = 2$ despite the theorem suggesting 3 and 4 respectively. This efficiency arises from their isomorphism to subgroups of $\mathrm{SO}(3, \mathbb{R})$ (Schwarzbach, 2010, Ch. 1, Sec. 2.4): each element can be mapped to a 3D rotation via just two Householder reflections (see Figure 4); a consequence of the Cartan-Dieudonné theorem (Gallier, 2011) stating that any orthogonal transformation (e.g. rotations) can be expressed as a composition of reflections. See Appendix E.1 for details on the isomorphisms of S_4, A_5, and S_5.

In Figure 3 (bottom row), we explore the limits of the expressivity of DeltaNet (i.e. $n_h = 1$) with multiple layers, which is still not fully understood theoretically. Increasing the number of layers

while keeping $n_h = 1$ improves performance but less effectively than increasing n_h. Interestingly, DeltaNet cannot learn S_4 and S_5 even with 5 layers, and with two layers, it performs poorly also on S_3, even though Theorem 1 shows that it can solve it. This suggests that simply increasing the number of layers may not be sufficient to attain the improvements of DeltaProduct enabled by incorporating more Householder transformations.

Analysis. To empirically validate whether DeltaProduct$_2[-1, 1]$ exploits the isomorphism of S_4 to subgroups of $\mathrm{SO}(3, \mathbb{R})$, we verified two hypotheses: whether both householders act as reflections ($\beta_0 = \beta_1 = 2$) composing to form rotations, and whether the keys exist in a three-dimensional subspace. By recording β_0 and β_1 values (representing the first and second householder in the product) across all 24 permutations of S_4, we find that a single head had indeed learned to use both Householder transformations as reflections where $\beta_0 = \beta_1 = 2$, effectively creating rotation matrices as shown in Figure 11. This pattern is evident in Figure 5 (left), where this head consistently displays beta values of approximately 2, confirming that the model successfully learns to approximate rotations by combining two reflections. To further verify whether the keys are in a three-dimensional subspace, we apply Principal Component Analysis (PCA) (Pearson, 1901) to the key vectors of the head where beta values approached 2. The results, displayed in Figure 5 (right), demonstrate that three principal components account for over 95% of the variance in the key space. This finding strongly supports our theoretical understanding, as it indicates that the model primarily operates in a three-dimensional subspace, which aligns with the structure of $\mathrm{SO}(3, \mathbb{R})$.

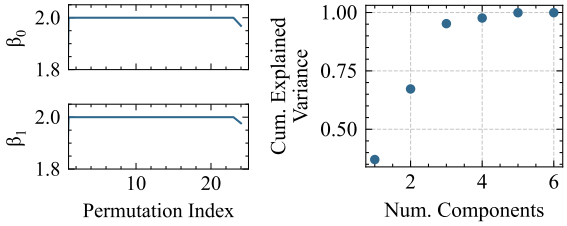

Figure 5: *(Left)* Estimated beta values for DeltaProduct$_2[-1, 1]$ on all permutations of S_4, clustering near 2 (reflection). *(Right)* PCA of key vectors shows that the first three components explain most of the variance.

5.2 Chomsky Hierarchy

Setup. We conducted experiments on selected formal language tasks originally introduced by Deletang et al. (2023). Our goal is to demonstrate the improvements in length extrapolation that can be achieved using multiple Householder matrices in the state-transition matrix compared to DeltaNet. Following Grazzi et al. (2025), we focus on three tasks: parity, modular arithmetic without brackets (both regular languages), and modular arithmetic with brackets (a context-free language). We trained DeltaProduct$_{n_h}$ with $n_h \in \{2, 3, 4\}$ on sequences of length 3 to 40 and tested on sequences ranging from 40 to 256 to evaluate generalization to longer inputs. We compare our results against Transformer, mLSTM and sLSTM from Beck et al. (2024), Mamba (Gu & Dao, 2024), and DeltaNet (Yang et al., 2024b). For both Mamba and DeltaNet, we experiment with eigenvalue ranges restricted to $[0, 1]$ and extended to $[-1, 1]$.

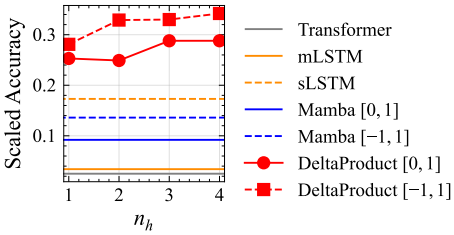

Figure 6: Results on Modular Arithmetic with brackets: DeltaProduct $[-1, 1]$ consistently outperforms all other methods.

Results. As shown in Table 2, DeltaProduct$_{n_h}$ with $n_h \geq 2$ demonstrates greater expressivity compared to DeltaNet and other baselines. This performance improvement is particularly pronounced when using the extended eigenvalue range $[-1, 1]$, which aligns with the findings of Grazzi et al. (2025). Notably, we observe the most significant improvement in the modular arithmetic with brackets task, where DeltaNet$[-1, 1]$ previously showed limitations (Grazzi et al., 2025) (see Figure 6). Across all tasks, DeltaProduct$_4[-1, 1]$ achieves an average accuracy of 0.778, improving 19.14% and 8.51% over sLSTM and DeltaNet$[-1, 1]$, respectively. Additional experimental details and hyperparameter values are provided in Appendix E.2.

5.3 Language Modeling

Setup. We trained two model variants: DeltaProduct$_{n_h}[-1, 1]$ and Gated DeltaProduct$_{n_h}[-1, 1]$ using the FineWeb dataset (Penedo et al., 2024) with 35B tokens. We adopted the training hyperpa-

Table 1: Performance comparison using lm-eval-harness benchmark (Gao et al., 2024) (SlimPajama (SPJ) reproduced from Yang et al. (2024b), Fine-Web (FW) ours). Results are shown for DeltaProduct and Gated DeltaProduct. We use 8 heads for each layer, unless otherwise specified.

	Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c 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
35B tokens FW	DeltaNet$[-1,1]$ 340M	26.92	43.07	29.8	69.0	41.0	50.9	46.6	24.5	43.6	26.4	30.2	3.7
	DeltaNet$[-1,1]$ 12 heads, 392M	26.57	36.76	31.8	69.2	42.3	50.9	47.2	24.4	44.3	15.8	11.0	0.18
	DeltaProduct$_2[-1,1]$ 392M	26.43	30.66	34.0	68.9	42.4	53.1	48.9	25.9	45.5	**32.0**	30.0	3.9
	DeltaProduct$_3[-1,1]$ 443M	25.94	29.91	34.2	**69.9**	43.2	51.9	48.2	24.1	45.2	30.6	30.4	5.3
	Gated DeltaNet$[-1,1]$ 340M	25.97	33.57	33.1	69.5	44.1	51.1	**50.9**	**26.7**	45.9	27.4	31.4	4.2
	Gated DeltaProduct$_2[-1,1]$ 393M	25.12	30.03	34.2	69.1	44.6	**55.3**	49.8	25.3	**46.4**	30.1	**31.6**	**6.6**

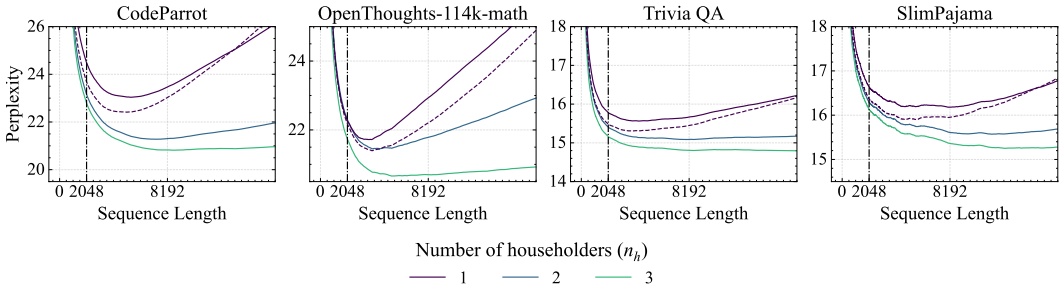

Figure 7: Length extrapolation results for DeltaProduct$_{n_h}[-1,1]$ on long context datasets, where $n_h \in \{1,2,3\}$, tested on sequences up to 16,384 tokens. Solid and dashed lines represent models with 8 and 12 heads respectively. Note that DeltaProduct$_2[-1,1]$ with 8 heads (392M parameters) matches the parameter count of DeltaNet ($n_h = 1$) with 12 heads, while achieving significantly better length extrapolation.

rameters and pipeline of Grazzi et al. (2025) (detailed in Appendix E.3.1). We evaluated the models using language understanding, reasoning, and retrieval benchmarks from lm-eval-harness (Gao et al., 2024), with task specifics in Appendix E.3.2. To assess extrapolation, we measured perplexity beyond the training context length of 2048 tokens on CodeParrot (Tunstall et al., 2022) for coding, OpenThoughts-114k-Math (Team, 2025) for math, TriviaQA (Joshi et al., 2017) for knowledge retrieval, and SlimPajama (Soboleva et al., 2023) for language modeling.

Results. Our experiments demonstrate that both DeltaProduct and Gated DeltaProduct on average outperform their baseline counterparts (DeltaNet$[-1,1]$ and Gated DeltaNet$[-1,1]$) across the considered language modeling benchmarks when we increase n_h, as shown in Table 1. Interestingly, DeltaProduct$_3[-1,1]$ achieves comparable performance to Gated DeltaNet$[-1,1]$, despite lacking a forget gate mechanism - a component considered crucial for language modeling tasks (Hochreiter & Schmidhuber, 1997; Gu & Dao, 2024). Furthermore, our training process remained stable even as we increased the value of n_h (see Figure 12). Remarkably, as shown in Figure 7, DeltaProduct's length extrapolation performance increases significantly with higher n_h values, and at $n_h = 3$, the performance degradation is minimal across the sequence length (see also Figure 13 for results up to 32k sequence length). We hypothesize that DeltaProduct achieves better length extrapolation by enhancing DeltaNet's forgetting mechanism. While DeltaNet requires n rank-1 updates to reset its state to zero, DeltaProduct can accelerate this forgetting process by a factor of n_h. This improvement could allow DeltaProduct$_3[-1,1]$ to learn effective forgetting patterns during training without an

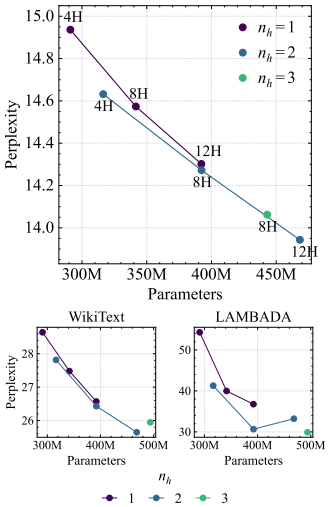

Figure 8: Scaling analysis of DeltaProduct$_{n_h}[-1,1]$ (H=heads) w.r.t. final training perplexity on FineWeb (top), WikiText, and Lambada via lm-eval harness.

additional scalar forget gate, unlike Gated DeltaNet. With a head embedding dimension of 128, DeltaProduct$_3[-1, 1]$ can reset its state in approximately 43 tokens, making it much more efficient at handling long-range dependencies. However, our experiments show that DeltaProduct$_2[-1, 1]$ still performs better with a forget gate, as demonstrated by its improved results when compared to the non-gated version (see Figure 13). To fairly compare DeltaProduct and DeltaNet, we conducted scaling experiments that accounted for DeltaProduct's additional parameters from its extra key, value, and β projection layers. We varied the number of heads in both models. As shown in Figure 8 (top), DeltaProduct consistently achieves better performance than DeltaNet, though the perplexity difference is modest. We expanded our analysis by evaluating DeltaProduct$_2[-1, 1]$ and DeltaNet across multiple benchmarks from lm-eval-harness (Gao et al., 2024). The results, shown in Figure 8 (bottom), reinforce our findings from the fineweb dataset. DeltaProduct maintains its performance advantage on both WikiText and Lambada tasks, showing improved perplexity across all model configurations. Our results align with the findings from Section 5.1 on state tracking: adding more householders proves more effective for improving length extrapolation compared to increasing the number of heads/layers.

6 Conclusion, Limitations and Future Work

We presented DeltaProduct, an extension of DeltaNet that uses products of Householder transformations as state-transition matrices. This approach bridges the gap between structured and dense matrices, with each recurrence step interpretable as multiple steps of gradient descent on an associative recall loss (compared to DeltaNet's single step). The number of Householder matrices (n_h) serves as a tunable parameter balancing expressivity and computational efficiency. Our experiments demonstrate DeltaProduct's superior performance over DeltaNet in state tracking, formal language recognition, and language modeling, with particularly strong length extrapolation results. DeltaProduct represents a promising step toward developing sequence models that are both more capable while still remaining scalable. **Limitations.** DeltaProduct has several key limitations. First, compared to DeltaNet, it requires more computational resources and memory, with requirements scaling linearly in n_h. Second, while both models can learn group word problems, we lack a comprehensive theoretical framework for understanding which problems can be learned when using multiple layers with relatively small values of n_h (relative to the group's complexity). **Future Work.** Future research could focus on integrating alternative matrix parametrizations, such as those used in RWKV-V7, and identifying problems that DeltaProduct cannot solve under finite precision constraints, following the work by Sarrof et al. (2024) and Grazzi et al. (2025).

Acknowledgements

We would like to thank Songlin Yang and Eddie Bergman for constructive discussions. We acknowledge the support and assistance of the Data Science and Computation Facility and its Support Team, in particular Mattia Pini, in utilizing the IIT High-Performance Computing Infrastructure, on which we run part of our experiments. This research was partially supported by the following sources: PNRR MUR Project PE000013 CUP J53C22003010006 "Future Artificial Intelligence Research (FAIR)", funded by the European Union – NextGenerationEU, and EU Project ELSA under grant agreement No. 101070617. TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215; the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828; the European Research Council (ERC) Consolidator Grant 'Deep Learning 2.0' (grant no. 10 This research was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 539134284, through EFRE (FEIH 2698644) and the state of Baden-Württemberg. Frank Hutter acknowledges financial support by the Hector Foundation. The authors acknowledge support from ELLIS and ELIZA. Funded by the European Union. The authors gratefully acknowledge the Gauss Center for Supercomputing eV (`www.gauss-centre.eu`) for funding this project by providing computing time on the GCS supercomputer JUWELS at Jülich Supercomputing Center (JSC). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the ERC. Neither the European Union nor the ERC can be held responsible for them.

Baden-Württemberg

Co-funded by the European Union

REFERENCES

Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proceedings of the VLDB Endowment*, 17(2):92–105, 2023.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended Long Short-Term Memory. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2024.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Kai Biegun, Rares Dolga, Jake Cunningham, and David Barber. RotRNN: Modelling Long Sequences with Rotations. *arXiv preprint arXiv:2407.07239*, 2024.

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, Apr. 2020.

Guy E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, 1990.

Nicola Muca Cirone, Antonio Orvieto, Benjamin Walker, Cristopher Salvi, and Terry Lyons. Theoretical Foundations of Deep Selective State-Space Models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*. PMLR, 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal Transformers. In *International Conference on Learning Representations*, 2019.

Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, et al. Neural Networks and the Chomsky Hierarchy. In *The Eleventh International Conference on Learning Representations*, 2023.

Victor D. Dorobantu, Per Andre Stromhaug, and Jess Renteria. Dizzyrnn: Reparameterizing recurrent neural networks for norm-preserving backpropagation. *ArXiv*, 2016.

Ting-Han Fan, Ta-Chung Chi, and Alexander Rudnicky. Advancing Regular Language Reasoning in Linear Recurrent Neural Networks. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 45–53, 2024.

Jaden Fried Fiotto-Kaufman, Alexander Russell Loftus, Eric Todd, Jannik Brinkmann, Koyena Pal, Dmitrii Troitskii, Michael Ripa, Adam Belfki, Can Rager, Caden Juang, et al. Nnsight and ndif: Democratizing access to foundation model internals. In *The Thirteenth International Conference on Learning Representations*, 2024.

Lorraine L Foster. On the symmetry group of the dodecahedron. *Mathematics Magazine*, 63(2): 106–107, 1990.

Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Re. Hungry Hungry Hippos: Towards Language Modeling with State Space Models. In *The Eleventh International Conference on Learning Representations*, 2021.

Joseph Gallian. *Contemporary abstract algebra*. Chapman and Hall/CRC, 2021.

Jean Gallier. The cartan–dieudonné theorem. In *Geometric Methods and Applications*, volume 38 of *Texts in Applied Mathematics*, pp. 123–145. Springer, New York, NY, 2011. doi: 10.1007/978-1-4419-9961-0_8.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Riccardo Grazzi, Julien Siems, Arber Zela, Jörg K.H. Franke, Frank Hutter, and Massimiliano Pontil. Unlocking State-Tracking in Linear RNNs Through Negative Eigenvalues. In *The Thirteenth International Conference on Learning Representations*, 2025.

Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *First Conference on Language Modeling*, 2024.

Albert Gu, Karan Goel, and Christopher Re. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*, 2022.

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

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1):31, 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In *International conference on machine learning*, pp. 9099–9117. PMLR, 2022.

Kazuki Irie, Róbert Csordás, and Jürgen Schmidhuber. Practical computational power of linear transformers and their recurrent and self-referential extensions. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1733–1741. JMLR.org, 2017.

Cijo Jose, Moustapha Cissé, and Francois Fleuret. Kronecker recurrent units. In *International Conference on Machine Learning*, pp. 2380–2389. PMLR, 2018.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *International Conference on Machine Learning*, pp. 5156–5165. PMLR, 2020.

Matthias Kissel and Klaus Diepold. Structured matrices and their application in neural networks: A survey. *New Generation Computing*, 41(3):697–722, 2023.

Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers Learn Shortcuts to Automata. In *The Eleventh International Conference on Learning Representations*, 2023.

Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and qiang liu. Longhorn: State space models are amortized online learners. In *The Thirteenth International Conference on Learning Representations*, 2025.

Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. When open information extraction meets the semi-structured web. *NAACL-HLT. Association for Computational Linguistics*, 2019.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *International Conference on Learning Representations*, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019.

Eric Martin and Chris Cundy. Parallelizing Linear Recurrent Neural Nets Over Sequence Length. In *International Conference on Learning Representations*, 2018.

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

William Merrill, Jackson Petty, and Ashish Sabharwal. The Illusion of State in State-Space Models. In *Forty-first International Conference on Machine Learning*, 2024.

Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *International Conference on Machine Learning*, pp. 2401–2409. PMLR, 2017.

Sajad Movahedi, Felix Sarnthein, Nicola Muca Cirone, and Antonio Orvieto. Fixed-point RNNs: From diagonal to dense in a few iterations. In *First Workshop on Scalable Optimization for Efficient and Adaptive Foundation Models*, 2025.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pp. 26670–26698. PMLR, 2023.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale, 2024.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Haowen Hou, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S. Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. RWKV-7 "Goose" with Expressive Dynamic State Evolution, 2025.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, 2018.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Yash Sarrof, Yana Veitsman, and Michael Hahn. The Expressive Capacity of State Space Models: A Formal Language Perspective. *Advances in Neural Information Processing Systems*, 2024.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pp. 9355–9366. PMLR, 2021a.

Imanol Schlag, Tsendsuren Munkhdalai, and Jürgen Schmidhuber. Learning associative inference using fast weight memory. In *International Conference on Learning Representations*, 2021b.

Mark Schöne, Babak Rahmani, Heiner Kremer, Fabian Falck, Hitesh Ballani, and Jannes Gladrow. Implicit Language Models are RNNs: Balancing Parallelization and Expressivity. *arXiv preprint arXiv:2502.07827*, 2025.

Yvette Kosmann Schwarzbach. Groups and symmetries from finite groups to lie groups, 2010.

Jimmy TH Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, June 2023.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): RNNs with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

OpenThoughts Team. Open Thoughts. https://open-thoughts.ai, January 2025.

Matteo Tiezzi, Michele Casoni, Alessandro Betti, Marco Gori, and Stefano Melacci. State-Space Modeling in Long Sequence Processing: A Survey on Recurrence in the Transformer Era, 2024.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Lewis Tunstall, Leandro Von Werra, and Thomas Wolf. *Natural Language Processing with Transformers*. O'Reilly Media, Inc., 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Ke Alexander Wang, Jiaxin Shi, and Emily B Fox. Test-time regression: A unifying framework for designing sequence models with associative memory. *arXiv preprint arXiv:2501.12352*, 2025.

Songlin Yang and Yu Zhang. FLA: A Triton-Based Library for Hardware-Efficient Implementations of Linear Attention Mechanism, January 2024. URL `https://github.com/sustcsonglin/flash-linear-attention`.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated Linear Attention Transformers with Hardware-Efficient Training. In *Forty-first International Conference on Machine Learning*, 2024a.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing Linear Transformers with the Delta Rule over Sequence Length. *Advances in Neural Information Processing Systems*, 36, 2024b.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated Delta Networks: Improving Mamba2 with Delta Rule. In *The Thirteenth International Conference on Learning Representations*, 2025.

Luca Zancato, Arjun Seshadri, Yonatan Dukler, Aditya Golatkar, Yantao Shen, Benjamin Bowman, Matthew Trager, Alessandro Achille, and Stefano Soatto. B'MOJO: Hybrid state space realizations of foundation models with eidetic and fading memory. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

A DeltaProduct with Identical Keys

Lemma 1. *Let* $\mathbf{k} \in \mathbb{R}^n$ *be nonzero, and let* $\alpha_1, \ldots, \alpha_m$ *be real scalars. Then*

$$\prod_{j=1}^{m} (\mathbf{I} - \alpha_j \mathbf{k}\mathbf{k}^\top) = \mathbf{I} - \alpha^* \mathbf{k}\mathbf{k}^\top$$

for some real scalar α^* *depending on* $\{\alpha_j\}_{j=1}^m$.

Proof. Without loss of generality, assume $\|\mathbf{k}\| = 1$. Otherwise, factor out $\|\mathbf{k}\|$ to rescale each α_j.

If $m = 1$, then the statement is trivially satisfied with $\alpha^* = \alpha$. Suppose the statement is true for $m \geq 1$, i.e. $\prod_{j=1}^m (\mathbf{I} - \alpha_j \mathbf{k}\mathbf{k}^\top) = \mathbf{I} - \alpha^{(m)} \mathbf{k}\mathbf{k}^\top$. Multiplying by $(\mathbf{I} - \alpha_{m+1} \mathbf{k}\mathbf{k}^\top)$ produces

$$\mathbf{I} - \left[\alpha^{(m)} + \alpha_{m+1} - \alpha^{(m)}\alpha_{m+1}\right] \mathbf{k}\mathbf{k}^\top.$$

Hence, by induction, the product of any number of such factors remains of the form $\mathbf{I} - \alpha^* \mathbf{k}\mathbf{k}^\top$. □

B Gated DeltaProduct

When adapting the Gated DeltaNet approach to Gated DeltaProduct, we define the state-transition matrices $\boldsymbol{A}$ and input matrices $\boldsymbol{B}$ as follows:

$$\boldsymbol{A}(\boldsymbol{x}_i) = g_i \prod_{j=1}^{n_h} \Big(\boldsymbol{I} - \beta_{i,j}\, \boldsymbol{k}_{i,j} \boldsymbol{k}_{i,j}^\top\Big), \quad \boldsymbol{B}(\boldsymbol{x}_i) = \sum_{j=1}^{n_h} \Big(\prod_{k=j+1}^{n_h} \big(\boldsymbol{I} - \beta_{i,k}\, \boldsymbol{k}_{i,k} \boldsymbol{k}_{i,k}^\top\big)\Big) \beta_{i,j}\, \boldsymbol{k}_{i,j} \boldsymbol{v}_{i,j}^\top.$$

where the gating term g_i is computed as:

$$g_i = \text{sigmoid}(W_g x_i) \in [0, 1]$$

C Runtime of DeltaProduct

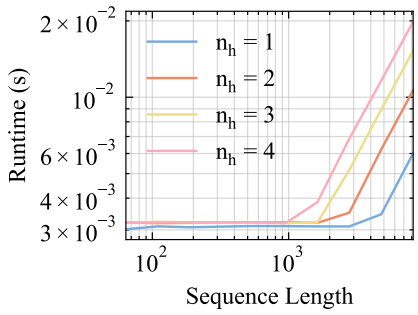

Figure 9: *DeltaProduct*$_{n_h}$ *scales linearly with* n_h. Runtime analysis of a single layer of DeltaProduct.

We evaluate the DeltaProduct layer on sequence lengths logarithmically spaced between 64 and 8192, while varying the number of Householder reflections from 1 to 4. Experiments were conducted on an Nvidia L40 GPU using `bfloat16` precision with a fixed batch size of 4. For each configuration, the model was run for 200 iterations and the runtimes were averaged following a burn-in phase of 3 iterations. The model was configured with a hidden size of 1024, a head dimension of 128 with 8 attention heads, and an expansion factor of 1.0. All models were compiled using `torch.compile` to ensure optimized performance. At a sequence length of 8192, the Delta Product layer achieved runtimes of approximately 0.006 seconds for 1 Householder, 0.010644 seconds for 2 Householder reflections, 0.01527 seconds for 3, and 0.01986 seconds for 4, respectively. These results, as shown in Figure 9, provide empirical evidence that the runtime increases linearly with the number of householders, since the computational cost scales as n_h times the sequence length.

D PROOF OF THEOREM 1

Proof. The elements of the dihedral group D_m can be divided into m rotations $\mathcal{R} = \{r_0, \dots, r_{m-1}\}$ and m reflections $\mathcal{S} = \{s_0, \dots, s_{m-1}\}$. The identity is r_0. To be able to solve the corresponding word problem, we would like to map sequences of group elements $x_1, \dots, x_t$ with $x_i \in \mathcal{R} \cup \mathcal{S}$ into sequences $y_1, \dots, y_t$ with $y_i = x_1 \cdot x_2 \cdots x_i$ and $\cdot$ is the group operation, that for dihedral groups is defined as

$$r_i \cdot r_j = r_{i+j \bmod m}, \quad r_i \cdot s_j = s_{i+j \bmod m}, \quad s_i \cdot r_j = s_{i-j \bmod m}, \quad s_i \cdot s_j = r_{i-j \bmod m}. \tag{3}$$

Note that a product of two rotations is commutative, while the product of two reflections or a reflection with a rotation is not. Indeed for $m \geq 3$ D_m, is not an abelian group.

The constructions of the two layers of the linear RNN builds upon the one for the cyclic group Z_m outlined in (Grazzi et al., 2025, Theorem 6). The first layer is constructed to output parity separately for rotations and reflections. In particular, using the following diagonal recurrence which indicates in the first (second) coordinate whether the number of rotations (reflections) is even (0) or odd (1).

$$\boldsymbol{h}_0^{(1)} = 0, \quad \boldsymbol{h}_t^{(1)} = \boldsymbol{a}(x_t) \odot \boldsymbol{h}_{t-1}^{(1)} + \boldsymbol{b}(x_t), \quad \boldsymbol{y}_t^{(1)} = \mathrm{dec}^{(1)}(\boldsymbol{h}_t, x_t) = (x_t, h_{t,1}, h_{t,2}).$$

$$\boldsymbol{a}(x_i)_1 = \begin{cases} -1 & \text{if } x_i \in \mathcal{R} \\ 1 & \text{if } x_i \in \mathcal{S} \end{cases} \quad \boldsymbol{a}(x_i)_2 = \begin{cases} -1 & \text{if } x_i \in \mathcal{S} \\ 1 & \text{if } x_i \in \mathcal{R} \end{cases}$$

$$\boldsymbol{b}(x_i)_1 = \begin{cases} 1 & \text{if } x_i \in \mathcal{R} \\ 0 & \text{if } x_i \in \mathcal{S} \end{cases} \quad \boldsymbol{b}(x_i)_2 = \begin{cases} 1 & \text{if } x_i \in \mathcal{S} \\ 0 & \text{if } x_i \in \mathcal{R} \end{cases}$$

This recurrence can be implemented also by DeltaNet, which uses generalized Householder matrices, but it requires at least 2 heads. For the second layer, we have instead the following constructions, which selects the appropriate reflection based on the parity of the rotations and uses the parity of the reflections for dec.

$$\boldsymbol{h}_0^{(2)} = (1,0)^\top, \quad \boldsymbol{h}_t^{(2)} = \boldsymbol{A}^{(2)}(\boldsymbol{y}_t^{(1)})\boldsymbol{h}_{t-1}^{(2)}, \quad \boldsymbol{y}_t^{(2)} = \mathrm{dec}^{(2)}(\boldsymbol{h}_t^{(2)}, \boldsymbol{y}_t^{(1)})$$

$$\boldsymbol{A}^{(2)}(\boldsymbol{y}) = \boldsymbol{H}(\theta(y_1, y_2)) = \begin{bmatrix} \cos\theta(y_1, y_2) & \sin\theta(y_1, y_2) \\ \sin\theta(y_1, y_2,) & -\cos\theta(y_1, y_2) \end{bmatrix}$$

$$\mathrm{dec}^{(2)}(\boldsymbol{h}, \boldsymbol{y}) = \begin{cases} r_{i^*} & \text{if } y_3 = 0 \\ s_{m-i^*} & \text{if } y_3 = 1 \end{cases}, \quad i^* = \underset{i \in \{0,\dots,m-1\}}{\arg\max} \max(\boldsymbol{c}_i^\top \boldsymbol{h}, \boldsymbol{d}_i^\top \boldsymbol{h})$$

where $\boldsymbol{y} = (y_1, y_2, y_3)^\top \in \mathcal{R} \cup \mathcal{S} \times \{0,1\} \times \{0,1\}$, $\boldsymbol{H}(\alpha)$ is the 2×2 reflection matrix that reflects all vectors by a line having an angle of $\alpha/2$ with the line passing from the origin and the vector $(1,0)^\top$ and $\theta : \mathcal{R} \cup \mathcal{S} \times \{0,1\} \to \mathbb{R}$ determines the angle of the reflection and is defined for all $i \in \{0, \dots, m-1\}$ as

$$\theta(r_i, 1) = \frac{(1-2i)\pi}{m}, \quad \theta(r_i, 0) = \frac{(1+2i)\pi}{m}, \qquad \theta(s_j, 1) = \frac{-2i\pi}{m}, \quad \theta(s_i, 0) = \frac{(2+2j)\pi}{m}.$$

Moreover, $\mathcal{C} = \{\boldsymbol{c}_0, \dots, \boldsymbol{c}_{m-1}\}$ and $\mathcal{D} = \{\boldsymbol{d}_0, \dots, \boldsymbol{d}_{m-1}\}$ are two sets of states and are defined as

$$\boldsymbol{d}_0 = \boldsymbol{h}_0^{(2)} = (1,0)^\top, \quad \boldsymbol{c}_0 = \boldsymbol{H}(\pi/m)\boldsymbol{d}_0,$$
$$\boldsymbol{d}_i = \boldsymbol{R}(2i\pi/m)\boldsymbol{d}_0, \quad \boldsymbol{c}_i = \boldsymbol{R}(-2i\pi/m)\boldsymbol{c}_0 \quad \text{for all } i \in \{0, \dots, m-1\},$$

where $\boldsymbol{R}(\beta)$ is a rotation matrix with angle $\beta \in \mathbb{R}$.

Let $\alpha, \gamma \in \mathbb{R}$, the following are standard identities of products of 2D rotations and reflections.

$$\boldsymbol{R}(\alpha)\boldsymbol{R}(\gamma) = \boldsymbol{R}(\alpha+\gamma), \qquad \boldsymbol{H}(\alpha)\boldsymbol{H}(\gamma) = \boldsymbol{R}(\alpha-\gamma),$$
$$\boldsymbol{R}(\alpha)\boldsymbol{H}(\gamma) = \boldsymbol{H}(\alpha+\gamma) \qquad \boldsymbol{H}(\gamma)\boldsymbol{R}(\alpha) = \boldsymbol{H}(\gamma-\alpha).$$

From our choice of $\boldsymbol{d}_0 = (1,0)^\top$ and $\boldsymbol{c}_0$, for any $\alpha \in \mathbb{R}$ we have

$$\begin{aligned} \boldsymbol{R}(\alpha)\boldsymbol{d}_0 &= \boldsymbol{H}(\alpha)d_0, \quad \text{and} \\ \boldsymbol{R}(\alpha)\boldsymbol{c}_0 &= \boldsymbol{R}(\alpha)\boldsymbol{H}(\pi/m)\boldsymbol{d}_0 = \boldsymbol{R}(\alpha)\boldsymbol{R}(\pi/m)\boldsymbol{d}_0 = \boldsymbol{R}(\alpha+\pi/m \pm \pi/m)d_0 \\ &= \boldsymbol{H}(\alpha+2\pi/m)\boldsymbol{H}(\pi/m)\boldsymbol{d}_0 = \boldsymbol{H}(\alpha+2\pi/m)\boldsymbol{c}_0. \end{aligned}$$

Moreover, from our choice of θ, $\boldsymbol{d}_i$ and $\boldsymbol{c}_i$, using the identities above and the the fact that $\boldsymbol{R}$ is a periodic function with period 2π we have that

$$\begin{aligned}
\boldsymbol{d}_i &= \boldsymbol{R}(2i\pi/m)\boldsymbol{d}_0 = \boldsymbol{R}(2i\pi/m)\boldsymbol{H}(\pi/m)\boldsymbol{c}_0 = \boldsymbol{H}(\theta(r_i,0))\boldsymbol{c}_0 \\
\boldsymbol{c}_i &= \boldsymbol{R}(-2i\pi/m)\boldsymbol{c}_0 = \boldsymbol{R}(-2i\pi/m)\boldsymbol{H}(\pi/m)\boldsymbol{d}_0 = \boldsymbol{H}(\theta(r_i,1))\boldsymbol{d}_0 \\
\boldsymbol{d}_{m-i} &= \boldsymbol{R}(-2i\pi/m)\boldsymbol{d}_0 = \boldsymbol{H}(-2i\pi/m)\boldsymbol{d}_0 = \boldsymbol{H}(\theta(s_i,1))\boldsymbol{d}_0 \\
\boldsymbol{c}_{m-i} &= \boldsymbol{R}(+2i\pi/m)\boldsymbol{c}_0 = \boldsymbol{H}((2+2i)\pi/m)\boldsymbol{c}_0 = \boldsymbol{H}(\theta(s_i,0))\boldsymbol{c}_0
\end{aligned}$$

for every $i \in \{0,\dots,m-1\}$. Therefore, we can write

$$\begin{aligned}
\boldsymbol{H}(\theta(r_j,1))\boldsymbol{d}_i &= \boldsymbol{R}(\theta(r_j,1)-\theta(r_i,0))\boldsymbol{c}_0 = \boldsymbol{R}(-2(i+j)\pi/m)\boldsymbol{c}_0 = \boldsymbol{c}_{i+j \bmod m}, \\
\boldsymbol{H}(\theta(r_j,0))\boldsymbol{c}_i &= \boldsymbol{R}(\theta(r_j,0)-\theta(r_i,1))\boldsymbol{d}_0 = \boldsymbol{R}(2(i+j)\pi/m)\boldsymbol{d}_0 = \boldsymbol{d}_{i+j \bmod m}, \\
\boldsymbol{H}(\theta(s_j,1))\boldsymbol{d}_i &= \boldsymbol{R}(\theta(s_j,1)-\theta(s_{m-i},1))\boldsymbol{d}_0 = \boldsymbol{R}(-2(i+j)\pi/m)\boldsymbol{d}_0 = \boldsymbol{d}_{-i-j \bmod m}, \\
\boldsymbol{H}(\theta(s_j,0))\boldsymbol{c}_i &= \boldsymbol{R}(\theta(s_j,0)-\theta(s_{m-i},0))\boldsymbol{c}_0 = \boldsymbol{R}((2+2(i+j))\pi/m)\boldsymbol{c}_0 = \boldsymbol{c}_{-i-j \bmod m},
\end{aligned} \tag{4}$$

for every $i,j \in \{0,\dots,m-1\}$. We proceed to verify that the output of the second layer is computed correctly: satisfying the product rule for the dihedral group in (3), i.e. we want to verify that

$$y_t^{(2)} = \begin{cases} r_{i+j \bmod m} & \text{if } y_{t-1}^{(2)} = r_i, x_t = r_j \\ s_{i+j \bmod m} & \text{if } y_{t-1}^{(2)} = r_i, x_t = s_j \\ s_{i-j \bmod m} & \text{if } y_{t-1}^{(2)} = s_i, x_t = r_j \\ r_{i-j \bmod m} & \text{if } y_{t-1}^{(2)} = s_i, x_t = s_j \end{cases} \tag{5}$$

Where we set $y_0^{(2)} = r_0$. First note that when $y_t^{(2)} \in \mathcal{S}$, then $y_{t,3}^{(1)} = 1$ and when $y_t^{(2)} \in \mathcal{R}$, then $y_{t,3}^{(1)} = 0$. We consider two cases.

Case 1. If $y_{t-1}^{(2)} = r_i$ and hence $y_{t-1,3}^{(1)} = 0$, then using (4) we obtain

$$\boldsymbol{h}_t^{(2)} = \boldsymbol{A}^{(2)}(\boldsymbol{y}^{(1)})\boldsymbol{h}_{t-1}^{(2)} = \begin{cases} \boldsymbol{H}(\theta(r_j,1))\boldsymbol{d}_i = \boldsymbol{c}_{i+j \bmod m} & \text{if } x_t = r_j, y_{t,2}^{(1)} = 1 \\ \boldsymbol{H}(\theta(r_j,0))\boldsymbol{c}_i = \boldsymbol{d}_{i+j \bmod m} & \text{if } x_t = r_j, y_{t,2}^{(1)} = 0 \\ \boldsymbol{H}(\theta(s_j,1))\boldsymbol{d}_i = \boldsymbol{d}_{-i-j \bmod m} & \text{if } x_t = s_j, y_{t,2}^{(1)} = 1 \\ \boldsymbol{H}(\theta(s_j,0))\boldsymbol{c}_i = \boldsymbol{c}_{-i-j \bmod m} & \text{if } x_t = s_j, y_{t,2}^{(1)} = 0 \end{cases}$$

This, together with the definition of $\mathrm{dec}^{(2)}$ implies that

$$y_t^{(2)} = \mathrm{dec}^{(2)}(\boldsymbol{h}_t^{(2)}, \boldsymbol{y}_t^{(1)}) = \begin{cases} r_{i+j \bmod m} & \text{if } x_t = r_j, y_{t,3}^{(1)} = 0 \\ s_{i+j \bmod m} & \text{if } x_t = s_j, y_{t,3}^{(1)} = 1 \end{cases} \tag{6}$$

Case 2. If instead $y_{t-1}^{(2)} = s_i$ and hence $y_{t-1,3}^{(1)} = 1$, then using (4) we obtain

$$\boldsymbol{h}_t^{(2)} = \boldsymbol{A}^{(2)}(\boldsymbol{y}^{(1)})\boldsymbol{h}_{t-1}^{(2)} = \begin{cases} \boldsymbol{H}(\theta(r_j,1))\boldsymbol{d}_{m-i} = \boldsymbol{c}_{j-i \bmod m} & \text{if } x_t = r_j, y_{t,2}^{(1)} = 1 \\ \boldsymbol{H}(\theta(r_j,0))\boldsymbol{c}_{m-i} = \boldsymbol{d}_{j-i \bmod m} & \text{if } x_t = r_j, y_{t,2}^{(1)} = 0 \\ \boldsymbol{H}(\theta(s_j,1))\boldsymbol{d}_{m-i} = \boldsymbol{d}_{i-j \bmod m} & \text{if } x_t = s_j, y_{t,2}^{(1)} = 1 \\ \boldsymbol{H}(\theta(s_j,0))\boldsymbol{c}_{m-i} = \boldsymbol{c}_{i-j \bmod m} & \text{if } x_t = s_j, y_{t,2}^{(1)} = 0 \end{cases}$$

This, together with the definition of $\mathrm{dec}^{(2)}$ implies that

$$y_t^{(2)} = \mathrm{dec}^{(2)}(\boldsymbol{h}_t^{(2)}, \boldsymbol{y}_t^{(1)}) = \begin{cases} s_{i-j \bmod m} & \text{if } x_t = r_j, y_{t,3}^{(1)} = 1 \\ r_{i-j \bmod m} & \text{if } x_t = s_j, y_{t,3}^{(1)} = 0 \end{cases}. \tag{7}$$

Note that (6) and (7) imply (5). Setting the output of the linear RNN equal to the output of the second layer concludes the proof. □

E EXPERIMENTS

E.1 STATE-TRACKING

Clarification on the isomorphisms of S_4, A_5, and S_5 The rotation group of a cube is isomorphic to the symmetric group S_4. This correspondence arises because the cube has exactly four space diagonals, and every *proper rotation*—that is, every orientation-preserving isometry of the cube about an axis through its center—permutes these diagonals in all possible ways (see Figure 10). In particular, these proper rotations include, for example, the 90°, 180°, and 270° rotations about axes passing through the centers of opposite faces, the 180° rotations about axes through the midpoints of opposite edges, and the 120°/240° rotations about axes through opposite vertices. Hence, the proper rotational symmetries of the cube correspond precisely to the permutations of its four space diagonals (Gallian, 2021).

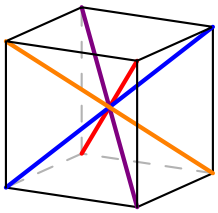

Figure 10: The permutations of the diagonals of the cube resulting from rotating the cube are exactly the S_4 group.

Similarly, a regular dodecahedron contains exactly five special cubes symmetrically arranged within it. Each *proper rotation* of the dodecahedron—that is, every orientation-preserving rigid motion mapping the dodecahedron onto itself—rearranges these inscribed cubes by an *even permutation*. This property makes the rotation group of the dodecahedron isomorphic to the alternating group A_5, the group of all even permutations of five elements (Foster, 1990).

When both proper rotations and reflections (orientation-reversing symmetries) are considered, the full symmetry group of the dodecahedron corresponds exactly to the symmetric group S_5, since reflections allow both even and odd permutations of the five hidden cubes (Foster, 1990).

Experimental Details. We used the experimental setup from Merrill et al. (2024) and sampled 2,000,000 training datapoints at sequence length 128 and 500,000 test datapoints at sequence length 512. We did not use a curriculum over sequence length during training. The models were trained using AdamW optimizer (Loshchilov & Hutter, 2019) with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ in PyTorch (Paszke et al., 2019). We used a learning rate of 10^{-3} with cosine annealing (Loshchilov & Hutter, 2017) and trained for 100 epochs with a batch size of 1024, except for the S_3 models which required a batch size of 2048 for more reliable results. All models used a single-layer DeltaProduct architecture featuring 12 heads (more heads made the results more reliable) and a head dimension of 32. We applied a weight decay coefficient of 10^{-6}. The β values were extracted from the forward pass of the trained models using NNsight (Fiotto-Kaufman et al., 2024). We use the PCA implementation in scikit-learn (Pedregosa et al., 2011).

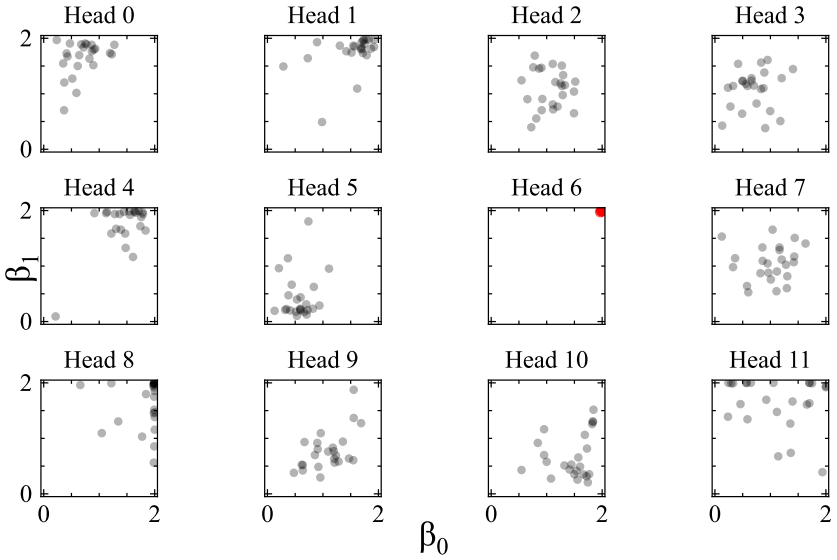

Figure 11: β_0 and β_1 values across all 24 permutations in S_4 in DeltaProduct$_2[-1, 1]$. We find that only head 6 (shown in Figure 5) learns to use both Householders as reflections ($\beta_0 \approx 2$, $\beta_1 \approx 2$) allowing it to learn the rotation to solve S_4.

E.2 CHOMSKY HIERARCHY

Here, we provide additional details on the formal language tasks and experimental protocol of Section 5.2.

E.2.1 EXPERIMENTAL SETUP

Similar to Beck et al. (2024), we trained each model with sequence lengths ranging from 3 to 40 and evaluated on lengths from 40 to 256, to understand the length generalization capabilities. We take the results shown in Table 2 for Transformer (Vaswani et al., 2017), mLSTM, sLSTM (Beck et al., 2024), Mamba (Gu & Dao, 2024) directly from Grazzi et al. (2025). All DeltaProduct and DeltaNet models contain 3 layers with 1 head each and heads' dimensions set to 128, except for modular arithmetic with brackets, where we use 12 heads and set the heads' dimensions to 32. Both models use a causal depthwise 1D convolution with a kernel size of 4 after the query/key/value projection. For modular arithmetic, we also use a gradient clipping norm of 1.0. We train each model using AdamW (Loshchilov & Hutter, 2019) using a learning rate of 5e-4, batch size of 1024, 0.1 weight decay, and a cosine annealing learning rate schedule (Loshchilov & Hutter, 2017) (minimum learning rate: 1e-6) after 10% warm-up steps. We train on the modular arithmetic and parity tasks for 100k and 20k steps in total, respectively. At each training step, we make sure to generate a valid random sample from the task at hand (see below). We repeat the runs 3 times with different seeds each, and later pick the best to report in Table 2.

E.2.2 EVALUATED TASKS

In Section 5.2, we empirically evaluated three tasks—parity, modular arithmetic without brackets, and modular arithmetic with brackets—spanning different levels of the Chomsky Hierarchy. These tasks were originally introduced by Deletang et al. (2023) and later used for benchmarking xLSTM (Beck et al., 2024, Figure 4). Below, we provide details for each task, where $|\Sigma|$ denotes the vocabulary size and Acc_{rand} represents the accuracy of random guessing:

- **Parity ($|\Sigma| = 2$, $Acc_{rand} = 0.5$).** Given a binary sequence $\boldsymbol{x} = x_1 \dots x_t \in \{0,1\}^t$, the parity label $y_t \in \{0,1\}$ is 1 if the total number of ones in the sequence is odd, and 0 otherwise. This task is equivalent to computing the sum of all previous values modulo 2, i.e., $y_t = (\sum_{i=1}^{t} x_i) \bmod 2$.
- **Modular Arithmetic without Brackets ($|\Sigma| = 10$, $Acc_{rand} = 1/5$).** Given a set of special tokens $\Sigma_s = \{+, -, *, =, [\texttt{PAD}]\}$ and a modulus $m \geq 1$, we define $\Sigma = \Sigma_s \cup \{0, \dots, m-1\}$. The label y_t corresponds to the result of evaluating the arithmetic operations in the sequence $\boldsymbol{x} = x_1, \dots, x_t$, computed modulo m. In our experiments, we set $m = 5$. An example is:

$$\texttt{2} + \texttt{1} - \texttt{2} * \texttt{2} - \texttt{3} = \texttt{1}\ [\texttt{PAD}]$$

- **Modular Arithmetic with Brackets ($|\Sigma| = 12$, $Acc_{rand} = 1/5$).** This task follows the same definition as modular arithmetic without brackets but includes an extended set of special tokens, $\Sigma_s = \{+, -, *, =,), (, [\texttt{PAD}]\}$, allowing for nested expressions. Again, we set $m = 5$. An example sequence is:

$$((\texttt{1} - (-\texttt{2})) + ((\texttt{4}) + \texttt{3})) = \texttt{0}\ [\texttt{PAD}]$$

E.3 LANGUAGE MODELING

E.3.1 EXPERIMENTAL SETUP

We follow the same basic training setup as in (Grazzi et al., 2025). We use the training pipeline `flame` from the flash-linear-attention (Yang & Zhang, 2024) repository. All of our models are trained on NVIDIA L40s or NVIDIA A100 40GB GPUs. We used 16 to 32 GPUs at a time to train one model, in either a 2 or 4 node setup, depending on resource availability. We used DeepSpeed with ZeRO-2 (Rajbhandari et al., 2020) for distributed training. All models were trained for $66\,758$ steps with a global batch size of $524\,288$, a learning rate of 3e-4, and a training context length of $2\,048$ tokens. We used two steps of gradient accumulation in the 16 GPU setup. We optimized the models with AdamW (Loshchilov & Hutter, 2019) (0.01 weight decay) and used cosine annealing (Loshchilov & Hutter, 2017) for the learning rate schedule with linear warm up for 512 steps.

Table 2: Performance of DeltaProduct$_{n_h}[-1,1]$, $n_h \in \{2,3,4\}$, on formal language tasks. We report the best of 3 runs. Scores are scaled accuracy, with 1.0 indicating perfect performance and 0.0 random guessing. The results for the other models were taken directly from Grazzi et al. (2025).

Model	Parity	Mod. Arithm. (w/o brackets)	Mod. Arithm. (w/ brackets)	Avg.
Transformer	0.022	0.031	0.067	0.040
mLSTM	0.087	0.040	0.114	0.080
sLSTM	**1.000**	**0.787**	**0.178**	**0.655**
Mamba $[0,1]$	0.000	0.095	**0.123**	0.073
Mamba $[-1,1]$	**1.000**	**0.241**	0.116	**0.452**
DeltaNet $[0,1]$	0.233	0.302	0.253	0.263
DeltaProduct$_2$ $[0,1]$	0.264	**0.402**	0.249	0.305
DeltaProduct$_3$ $[0,1]$	0.285	**0.402**	0.288	**0.325**
DeltaProduct$_4$ $[0,1]$	**0.295**	0.369	**0.288**	0.317
DeltaNet $[-1,1]$	**0.982**	**0.915**	0.281	0.726
DeltaProduct$_2$ $[-1,1]$	0.896	0.887	0.329	0.704
DeltaProduct$_3$ $[-1,1]$	0.932	0.736	0.330	0.666
DeltaProduct$_4$ $[-1,1]$	**0.982**	0.893	**0.342**	**0.739**

E.3.2 Evaluation Tasks

We use the lm-eval-harness benchmark (Gao et al., 2024) to assess model performance. Following Yang et al. (2024b), the evaluation encompasses multiple task categories: **Language Understanding Tasks.** The evaluation includes LAMBADA (LMB) (Paperno et al., 2016) for testing text comprehension, PIQA (Bisk et al., 2020) for physical reasoning assessment, HellaSwag (Hella.) (Zellers et al., 2019) for situational understanding, and Winogrande (Wino.) (Sakaguchi et al., 2021) for commonsense reasoning evaluation. **Reasoning.** The ARC dataset provides two distinct testing sets: ARC-easy (ARC-e) and ARC-challenge (ARC-c) (Clark et al., 2018), measuring varying levels of scientific knowledge comprehension. **Recall-Based Tasks.** The evaluation incorporates recall-intensive assessments through FDA (Arora et al., 2023), SWDE (Lockard et al., 2019), and SQUAD (Rajpurkar et al., 2018).

E.3.3 Training behavior

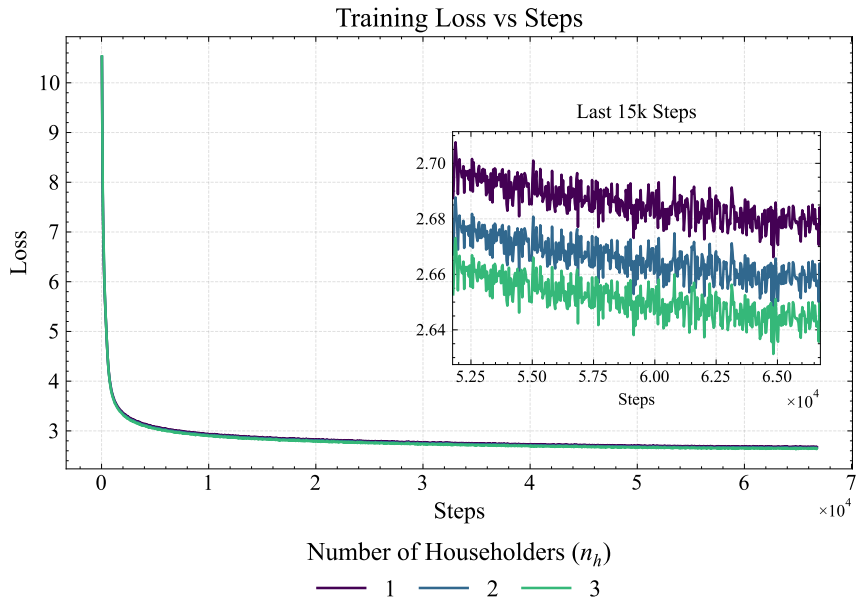

Figure 12: Training loss curves of DeltaProduct$_{n_h}[-1,1]$. The curves demonstrate stable training behavior as n_h increases, with higher values of n_h consistently yielding lower losses throughout training and convergence. While the absolute differences in loss between different n_h values are relatively small, they correspond to significant differences in length extrapolation performance.

E.3.4 ADDITIONAL RESULTS ON LENGTH EXTRAPOLATION

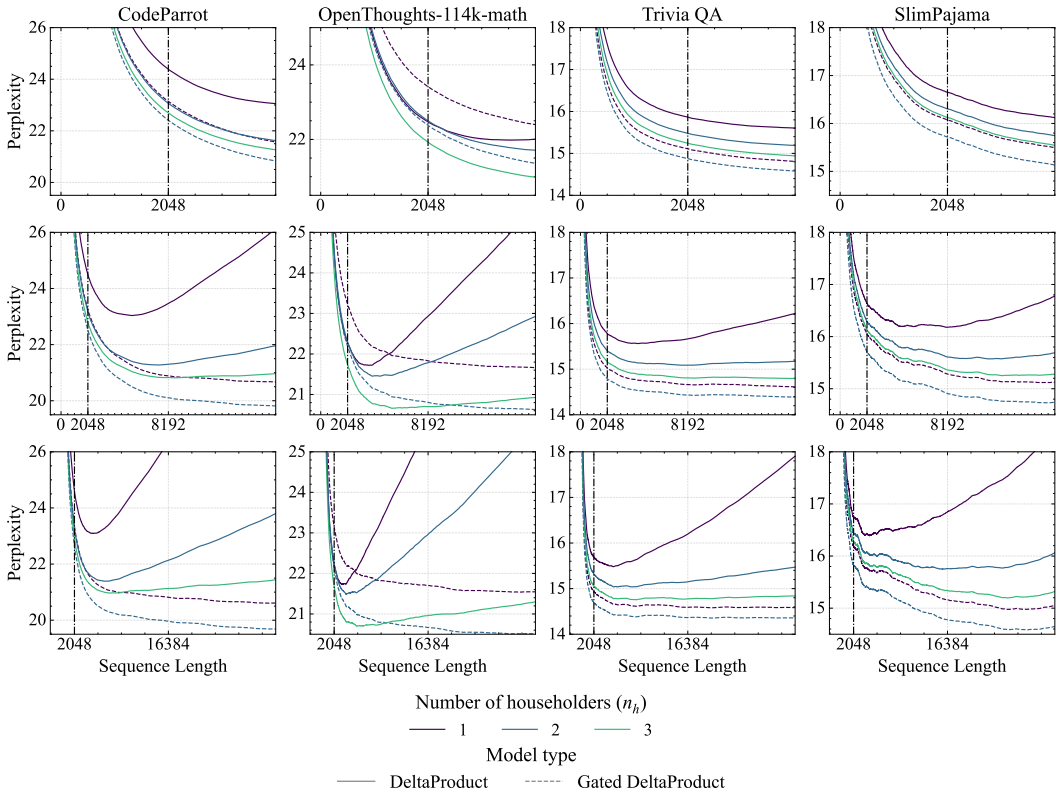

Figure 13: Gated DeltaProduct$_{n_h}$ $[-1, 1]$ and DeltaProduct$_{n_h}$ $[-1, 1]$ show improved length extrapolation when increasing n_h. *(Top)* 4 096 token context. *(Middle)* 16 384 token context. *(Bottom)* 32 768 token context.