ROTARY-STATE GATED DELTANET: ENHANCING STATE TRACKING THROUGH ROTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce **Ro**tary-**s**tat**e** **G**ated **D**elta**N**et (Rose-GDN), which augments Gated DeltaNet with input-dependent block-diagonal rotations. This adds rotational dynamics to the generalized-Householder transition without requiring multiple delta updates per token, while preserving fixed-size memory, linear-time recurrence, and exact chunkwise-parallel computation. We show that, under exact arithmetic, a one-layer Rose-GDN can track any finite abelian group. Increasing depth extends this result to all finite solvable groups, and a finite-depth Rose-GDN recognizes all regular languages. Empirically, Rose-GDN shows gains on state-tracking and long-context retrieval tasks while remaining competitive with strong recurrent baselines on language modeling and commonsense reasoning benchmarks. Together, these results suggest that combining input-dependent rotations with delta-rule memory provides a practical way to broaden the capabilities of efficient recurrent models.

1 INTRODUCTION

Many reasoning tasks require the ability of *state-tracking* - maintaining and measuring an evolving hidden state under a stream of affective, order-sensitive updates (Ebrahimi & Memisevic, 2026; Merrill et al., 2024). Examples include evaluating code, tracking entities through a narrative, and accumulating the effect of operations in formal-language problems (Delétang et al., 2022; Merrill et al., 2024; Grazzi et al., 2025). Formally, each input selects a state transition, and the state after a prefix is obtained by composing these transitions in sequence (Hopcroft et al., 2001; Merrill et al., 2024).

Many current frontier models show limited capabilities on related tasks. Theoretical analyses show that fixed-depth Transformers (Vaswani et al., 2017) cannot solve some periodic finite-state languages as sequence length grows (Hahn, 2020). Empirically, Transformers trained on formal languages and finite automata often learn solutions that fail to generalize beyond the training regime and struggle with permutation-composition state tracking (Delétang et al., 2022; Liu et al., 2022; Merrill & Sabharwal, 2023).

Recurrent models offer a different computational structure: they update a persistent state at every step. Linear recurrent neural networks retain this sequential state update while using linear-time computation and fixed-size recurrent memory, but achieving practical efficiency requires restricting the transitions applied to that state. Gated DeltaNet (GDN) maintains a matrix-valued associative memory and updates it through the rank-one generalized Householder factor $\mathbf{H}_t = \mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top$ (Schlag et al., 2021; Yang et al., 2024; 2025). For a unit key $\mathbf{k}_t$, $\mathbf{H}_t$ has eigenvalue $1-\beta_t$ along $\mathbf{k}_t$ and eigenvalue 1 on its orthogonal complement; hence $\beta_t \in [0, 1]$ permits only nonnegative eigenvalues, whereas $\beta_t \in [0, 2]$ allows the former eigenvalue to be negative, yielding the Householder reflection needed to track parity (Grazzi et al., 2025). However, a single generalized Householder factor still has only real eigenvalues. DeltaProduct extends the transition family by applying several generalized Householder factors per token, at the cost of scaling necessary computation with the number of factors and capability of state tracking (Siems et al., 2026). We therefore ask whether rotational dynamics can instead be introduced directly into GDN's state transition, avoiding multiple delta steps per token while retaining linear-time inference, a fixed-size recurrent state, and chunkwise-parallel training.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To answer this question, we introduce ***Ro**tary-stat**e** **G**ated **D**eltaNet (Rose-GDN)*, which adds an input-dependent block-diagonal rotation to the GDN transition:

$$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top)\mathbf{R}_t^\top + \beta_t \mathbf{v}_t \mathbf{k}_t^\top.$$

Each 2×2 block of $\mathbf{R}_t$ rotates one fixed coordinate pair by an angle determined by the current input. The recurrence remains linear in sequence length and admits an exact chunkwise-parallel form. Theoretical results establish one-layer tracking of all finite abelian groups and extend with depth to finite solvable groups and arbitrary regular languages, while experiments show improved finite-group tracking and long-context retrieval alongside competitive language modeling performance.

Our contributions are threefold:

- **A rotary delta rule.** We introduce Rose-GDN, which combines one generalized Householder update with input-dependent planar rotations while preserving linear recurrent complexity and admitting an exact chunkwise-parallel form.
- **Finite-state expressivity.** We give explicit exact-arithmetic constructions showing that one layer tracks every finite abelian group, depth equal to derived length suffices for finite solvable groups, and a finite-depth Rose-GDN recognizes every regular language.
- **Empirical evaluation.** We compare Rose-GDN with GDN, DeltaProduct, Mamba-3, and Transformers on formal-language and group-tracking tasks, and evaluate 340M-parameter models on language modeling, commonsense reasoning, and long-context retrieval.

Organization. We arrange our paper into the following parts. Section 2 includes the background on which our problem is based. In Section 3, our main methodology, Rose-GDN, and its chunkwise parallel form are discussed, while the state-tracking capability of our method is derived theoretically in Section 4. Experimental results can be found in section 5. In section 6, we provide related materials that work on colliding branches with our method. Other aspects such as further theoretical and experimental details are left in the Appendices of this paper.

Notation. Unless otherwise stated, bold lowercase and uppercase letters denote column vectors and matrices; calligraphic letters denote sets or matrix families. The index t denotes sequence position, and $\mathbf{S}_t$ is the state after input t. Key and value dimensions are d_k and d_v. We use $\mathbf{I}$ for identity, $(\cdot)^\top$ for transpose, $\|\cdot\|_2$ for Euclidean norm, $\odot$ for elementwise multiplication, diag for block-diagonal assembly, and $\mathrm{strictLower}$ for the strictly lower-triangular part. Chunk positions are indexed by i, j. For group tracking, e is the identity, $\mathbb{Z}_m$ is the cyclic group of order m, and $g_t = x_t g_{t-1} = x_t \cdots x_1$.

2 Background

2.1 Linear Recurrent Neural Networks

Linear recurrent neural networks (LRNNs) summarize a sequence in a fixed-size hidden state. Each step transforms the previous state and adds new information, with both operations controlled by the current input. A layer takes the form

$$\mathbf{h}_t = A(\mathbf{x}_t)\mathbf{h}_{t-1} + b(\mathbf{x}_t), \qquad \mathbf{y}_t = D(\mathbf{h}_t, \mathbf{x}_t), \tag{1}$$

where $\mathbf{x}_t \in \mathcal{X}$ is the current input, $\mathbf{h}_t \in \mathcal{H} \subseteq \mathbb{R}^d$ is the recurrent state, and $\mathbf{y}_t \in \mathcal{Y}$ is the output. The decoder $D : \mathcal{H} \times \mathcal{X} \to \mathcal{Y}$ maps the current state and input to the output $\mathbf{y}_t$. Although A and b may depend nonlinearly on $\mathbf{x}_t$, and D may be nonlinear in both arguments, the state update remains affine in $\mathbf{h}_{t-1}$ (Grazzi et al., 2025).

Gated DeltaNet and variants. Gated DeltaNet uses a matrix memory to store associations between keys and values. DeltaNet's delta rule corrects these associations as new inputs arrive (Schlag et al., 2021; Yang et al., 2024), while Gated DeltaNet introduces an additional decay gate that controls how much of the existing memory is retained (Yang et al., 2025). For keys and queries $\mathbf{k}_t, \mathbf{q}_t \in \mathbb{R}^{d_k}$, values $\mathbf{v}_t \in \mathbb{R}^{d_v}$, and gates $\alpha_t, \beta_t \in [0, 1]$, its recurrent core is

$$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1}\left(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top\right) + \beta_t \mathbf{v}_t \mathbf{k}_t^\top, \qquad \mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t, \tag{2}$$

where $\mathbf{S}_t \in \mathbb{R}^{d_v \times d_k}$ is the recurrent state, $\mathbf{S}_0 = \mathbf{0}$, and $\|\mathbf{k}_t\|_2 = 1$. The gate α_t controls global forgetting, while β_t controls correction of the association at $\mathbf{k}_t$. Setting $\alpha_t = 1$ recovers DeltaNet. The transition factor $A_t = \alpha_t(I - \beta_t \mathbf{k}_t \mathbf{k}_t^\top)$ has eigenvalue $\alpha_t(1 - \beta_t)$ along $\mathbf{k}_t$ and α_t on its orthogonal complement. Extending β_t to $[0, 2]$ permits negative eigenvalues (Grazzi et al., 2025); at $\beta_t = 2$, the factor $I - \beta_t \mathbf{k}_t \mathbf{k}_t^\top$ is a Householder reflection. The recurrent update supports chunkwise parallel training (Yang et al., 2024; 2025).

2.2 State tracking

State tracking is the ability of a model to maintain the current state of a system from an initial state s_0 and a sequence of updates $x_1, \ldots, x_t$. Unlike recalling past inputs, state tracking requires the model to compose updates in the correct order, preserve unaffected information, and replace obsolete information. This capability is essential in tasks such as tracking objects after moves or swaps, updating an agent's position during navigation, maintaining dialogue goals and corrected slot values, executing program instructions, and updating balances or inventories after transactions.

Formally, state tracking can be modeled as a monoid word problem. A monoid $(M, \cdot)$ consists of a set M, an associative binary operation $\cdot$, and an identity element e satisfying $e \cdot m = m \cdot e = m$ for every $m \in M$. Given a sequence of monoid elements $x_1, \ldots, x_t \in M$, state tracking computes the cumulative states $y_1, \ldots, y_t$ such that $y_i = x_i \cdot x_{i-1} \cdots x_1$. Since recognizing a regular language can be reduced to solving the word problem of an associated finite monoid, we focus on this setting throughout the paper. A particularly important case arises when M is a finite group, yielding a group word problem. These problems are known to be challenging for both Transformers and linear RNNs. Among them, word problems over symmetric (permutation) groups are especially significant because every finite group is isomorphic to a subgroup of a symmetric group. For example, parity corresponds to the S_2 word problem, which cannot be solved in finite precision by Transformers (Hahn, 2020) or by diagonal linear RNNs with positive parameters (Sarrof et al., 2024). More generally, the S_5 word problem remains unsolvable by these architectures even when the precision grows logarithmically with the sequence length (Liu et al., 2022; Merrill & Sabharwal, 2023; Merrill et al., 2024). In contrast, linear RNNs equipped with unconstrained full state-transition matrices can recognize any regular language using a single layer (see, e.g., (Merrill et al., 2024, Theorem 5.2)). However, training with fully unstructured transition matrices is often computationally inefficient and can be unstable due to the lack of explicit control over matrix norms.

3 Rotary-state Gated DeltaNet

3.1 Formulation

Original Gated DeltaNet performs efficient associative-memory updates through the rank-one generalized Householder (GH) transition $\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top$. Under the conventional constraint $\beta_t \in [0, 1]$, this matrix has eigenvalue $1 - \beta_t$ along $\boldsymbol{k}_t$ and eigenvalue 1 on its orthogonal complement. Its spectrum is therefore restricted to the nonnegative real axis, limiting the state dynamics that a single transition can represent. Grazzi et al. (2025) extend the gate range to $\beta_t \in [0, 2]$, allowing the eigenvalue $1 - \beta_t$ to become negative and recovering a Householder reflection when $\beta_t = 2$. This extension enables richer state transitions, including the alternating dynamics required for parity (Grazzi et al., 2025). However, a single generalized-Householder factor still has only real eigenvalues and therefore cannot directly represent nontrivial rotational dynamics.

To overcome this limitation, we propose *Rotary-state Gated DeltaNet (Rose-GDN)*, which augments the generalized-Householder transition with an input-dependent block-diagonal rotation.

Definition 1 (Rotary-state Gated DeltaNet)**.** *For key dimension $n = 2r$ and value dimension d_v, a Rose-GDN layer maintains a state $\mathbf{S}_t \in \mathbb{R}^{d_v \times n}$. Given a unit key $\mathbf{k}_t \in \mathbb{R}^n$, query $\mathbf{q}_t \in \mathbb{R}^n$, value $\mathbf{v}_t \in \mathbb{R}^{d_v}$, and gates $\alpha_t \in [0, 1]$ and $\beta_t \in [0, 1]$, it computes*

$$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top)\mathbf{R}_t^\top + \beta_t \mathbf{v}_t \mathbf{k}_t^\top, \qquad \mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t, \tag{3}$$

where β_t may instead range over $[0, 2]$ when reflections are allowed. The input-dependent rotation is

$$\mathbf{R}_t = \mathbf{R}(\boldsymbol{\theta}_t) = \mathrm{diag}(\mathbf{R}_{\theta_{t,1}}, \ldots, \mathbf{R}_{\theta_{t,r}}), \qquad \mathbf{R}_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}. \tag{4}$$

Unless stated otherwise, $\boldsymbol{\theta}_t$ is a function of the current layer input.

We use a Mamba-3 style (Lahoti et al., 2026) parameterization for $\boldsymbol{\theta}_t$, which are specified in Appendix C.1.

The rotary transition also changes how the memory is addressed. To make this effect explicit, define the cumulative frame $\boldsymbol{\Phi}_0 = I$, $\boldsymbol{\Phi}_i = \boldsymbol{R}_i\boldsymbol{\Phi}_{i-1}$, and the transformed memory $\widetilde{\boldsymbol{S}}_i = \boldsymbol{S}_i\boldsymbol{\Phi}_i$. Let

$$\boldsymbol{a}_i = \boldsymbol{\Phi}_{i-1}^\top \boldsymbol{k}_i, \qquad \boldsymbol{b}_i = \boldsymbol{\Phi}_i^\top \boldsymbol{k}_i, \qquad \boldsymbol{c}_i = \boldsymbol{\Phi}_i^\top \boldsymbol{q}_i$$

denote the erase key, write key, and query in this cumulative coordinate system. Multiplying equation 3 by $\boldsymbol{\Phi}_i$ gives

$$\widetilde{\boldsymbol{S}}_i = \alpha_i \widetilde{\boldsymbol{S}}_{i-1}(I - \beta_i \boldsymbol{a}_i \boldsymbol{a}_i^\top) + \beta_i \boldsymbol{v}_i \boldsymbol{b}_i^\top, \qquad \boldsymbol{o}_i = \widetilde{\boldsymbol{S}}_i \boldsymbol{c}_i. \tag{5}$$

In Gated DeltaNet, the same key selects both the association removed from memory and the location of the new write. Rose-GDN instead uses distinct addresses in the cumulative frame: the erase key $\boldsymbol{a}_i = \boldsymbol{\Phi}_{i-1}^\top \boldsymbol{k}_i$ includes rotations through the previous token, whereas the write key $\boldsymbol{b}_i = \boldsymbol{\Phi}_{i-1}^\top \boldsymbol{R}_i^\top \boldsymbol{k}_i$ additionally includes the current rotation. Equation 5 therefore describes a two-address delta recurrence that erases in the previous frame and writes in the updated frame.

3.2 Chunkwise parallel form

For a chunk of C tokens, stack $\boldsymbol{a}_i^\top, \boldsymbol{b}_i^\top, \boldsymbol{c}_i^\top$, and $\boldsymbol{v}_i^\top$ into $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}_R, \boldsymbol{V}$, respectively. Set $\gamma_i = \prod_{j=1}^{i} \alpha_j$, $D_\gamma = \mathrm{diag}(\gamma_i)$, and $D_\beta = \mathrm{diag}(\beta_i)$. The causal decay matrix has entries $\Gamma_{ij} = \prod_{\ell=j+1}^{i} \alpha_\ell$ for $i \geq j$ and zero otherwise, with empty products equal to one. For positive decay gates, $\Gamma_{ij} = \gamma_i/\gamma_j$ on the causal entries. Define

$$\boldsymbol{L}^\Gamma_{AA} = \mathrm{strictLower}(\Gamma \odot \boldsymbol{A}\boldsymbol{A}^\top),$$
$$\boldsymbol{L}^\Gamma_{AB} = \mathrm{strictLower}(\Gamma \odot \boldsymbol{A}\boldsymbol{B}^\top),$$

where strictLower retains entries strictly below the diagonal. Following the erase-transition and write-correction decomposition in Appendix B, compute

$$\boldsymbol{T}_A = [I + \mathrm{strictLower}(D_\beta \boldsymbol{A}\boldsymbol{A}^\top)]^{-1} D_\beta, \qquad \boldsymbol{W} = \boldsymbol{T}_A \boldsymbol{A},$$
$$\boldsymbol{U} = -(I + D_\beta \boldsymbol{L}^\Gamma_{AA})^{-1} D_\beta \boldsymbol{L}^\Gamma_{AB} D_\beta \boldsymbol{V}. \tag{6}$$

Here $\boldsymbol{W}$ represents the accumulated erase transition, while $\boldsymbol{U}$ collects the corrections to earlier writes. Use the decay notation

$$\overleftarrow{\boldsymbol{W}} = D_\gamma \boldsymbol{W}, \qquad \overleftarrow{\boldsymbol{Q}}_R = D_\gamma \boldsymbol{Q}_R,$$
$$\overrightarrow{\boldsymbol{A}}_{i,:} = \Gamma_{Ci} \boldsymbol{A}_{i,:}, \qquad \overrightarrow{\boldsymbol{B}}_{i,:} = \Gamma_{Ci} \boldsymbol{B}_{i,:},$$

and $\overrightarrow{\widetilde{\boldsymbol{S}}_0} = \gamma_C \widetilde{\boldsymbol{S}}_0$. The final memory and chunk outputs, with $\boldsymbol{O}_{i,:} = \boldsymbol{o}_i^\top$, are

$$\begin{aligned} \widetilde{\boldsymbol{S}}_C &= \overrightarrow{\widetilde{\boldsymbol{S}}_0} + \boldsymbol{V}^\top D_\beta \overrightarrow{\boldsymbol{B}} + (\boldsymbol{U} - \overleftarrow{\boldsymbol{W}} \widetilde{\boldsymbol{S}}_0^\top)^\top \overrightarrow{\boldsymbol{A}}, \\ \boldsymbol{O} &= \overleftarrow{\boldsymbol{Q}}_R \widetilde{\boldsymbol{S}}_0^\top + [(\boldsymbol{Q}_R \boldsymbol{B}^\top) \odot \Gamma] D_\beta \boldsymbol{V} \\ &\quad + [(\boldsymbol{Q}_R \boldsymbol{A}^\top) \odot \Gamma](\boldsymbol{U} - \overleftarrow{\boldsymbol{W}} \widetilde{\boldsymbol{S}}_0^\top). \end{aligned} \tag{7}$$

Finally, $\boldsymbol{S}_C = \widetilde{\boldsymbol{S}}_C \boldsymbol{\Phi}_C^\top$. Appendix B provides the full derivation.

4 State-tracking Capabilities

The state-tracking ability of Rose-GDN stems from the expressivity of rotation matrices, as well as GH matrices. We introduce a family of *GH–rotation matrices* $\mathcal{F}_n(\Omega)$ as

$$\mathcal{F}_n(\Omega) = \{(\mathbf{I} - \beta \boldsymbol{u}\boldsymbol{u}^\top)\mathbf{R} : \mathbf{R} \in \mathcal{T}_n,\ \|\boldsymbol{u}\|_2 = 1,\ (1-\beta) \in \Omega\}, \tag{8}$$

where $\mathcal{T}_n$ is the group of block-diagonal planar rotations. With this family of state-transition matrices, Rose-GDN with a state-transition matrix belonging to $\mathcal{F}_n([-1;1])$ can track not only the cyclic group $\mathbb{Z}_m$, but also all finite abelian groups and all finite solvable groups, as well as recognize all regular languages.

4.1 FINITE GROUP TRACKING

Rose-GDN extends the state-tracking capabilities of a single GDN$[-1, 1]$ layer (Grazzi et al. (2025)) from parity to modular addition with arbitrary moduli and, more generally, every finite abelian group, without requiring multiple GH updates per token (Grazzi et al., 2025). Moreover, Rose-GDN can track every finite solvable group of derived length d using at most d layers. These guarantees hold under our exact finite-state assumptions and require no additional processing tokens. A detailed comparison of state-tracking ability between Rose-GDN and other architectures is presented in Table 1.

Theorem 1 (Finite group tracking). *1. Let $G \cong \prod_{j=1}^{r} \mathbb{Z}_{m_j}$ be a finite abelian group. There exists a one-layer Rose-GDN with state $\boldsymbol{S}_t \in \mathbb{R}^{d_v \times n}$, where n is at most $2r + 2$, and state transition matrix in $\mathcal{F}_n((-1; 1])$ that can track G.*

2. Let G be a finite solvable group of derived length d. Then there exists a Rose-GDN with at most d layers and a state transition matrix in $\mathcal{F}_n((-1; 1])$ that can track G.

Each planar rotation block acts as a modular counter for one cyclic factor $\mathbb{Z}_{m_j}$. Let $a_j(g) \in \{0, \dots, m_j - 1\}$ denote the j-th coordinate of a group element g. This coordinate is encoded by a rotation through the angle $2\pi a_j(g)/m_j$. Because planar rotations compose by adding their angles, the relation $a_j(gh) = a_j(g) + a_j(h) \pmod{m_j}$ implies that the rotation associated with gh is the product of the rotations associated with g and h. For a finite abelian group

$$G \cong \prod_{j=1}^{r} \mathbb{Z}_{m_j},$$

these cyclic representations can be combined into the block-diagonal matrix

$$\rho(g) = \operatorname{diag}\left(\mathbf{R}\left(\frac{2\pi a_1(g)}{m_1} \right), \dots, \mathbf{R}\left(\frac{2\pi a_r(g)}{m_r} \right) \right),$$

where each two-dimensional block independently tracks one cyclic coordinate. The resulting representation uses r planar rotation blocks and satisfies $\rho(gh) = \rho(g)\rho(h)$. Consequently, applying the rotation associated with an input x_t updates the stored representation from $\rho(g_{t-1})$ to $\rho(g_t)$, where $g_t = x_t g_{t-1}$. A fixed query extracts a cosine–sine pair from each rotation plane, and the resulting collection uniquely determines all cyclic coordinates. The combined representation is therefore injective, allowing a single Rose-GDN layer to track the prefix products of any finite abelian group. A detailed proof of Theorem 1 is included in Appendix D.

4.2 REGULAR LANGUAGE RECOGNITION

The group-tracking results above establish guarantees for reversible updates drawn from finite abelian and finite solvable groups. We next show, using a cascade of permutation and reset components, that Rose-GDN can implement any deterministic finite-state automaton and hence recognize every regular language.

Theorem 2 (Regular language recognition). *Rotary-state Gated DeltaNet can recognize all regular languages. In particular, any finite state automata with N states can be implemented by a Rotary-state Gated DeltaNet with $L \leq 2^{N+1} + 1$ layers.*

Appendix E gives the full construction. It uses the cascade theorem stated by Grazzi et al. (2025, Appendix C.4, Theorem 5), based on Maler (2010), and adapts delayed block simulation from Siems et al. (2026). A buffer stores recent component updates, and a second layer executes elementary factors of the preceding block. Current pointwise readouts recover both the current and previous component states, including after resets, so later components can obey the cascade's previous-state convention.

5 EXPERIMENTS

This evaluate Rose-GDN on formal-language and abelian-group tracking tasks, followed by language modeling, commonsense reasoning, and long-context retrieval. Following Grazzi et al.

Table 1: Sufficient layer counts for exact prefix-product state tracking, with one group element per token. K counts GH factors per head and token; d is derived length. Bounds need not be minimal; width, heads, decoders, gate endpoints, and exact finite-state arithmetic may depend on the task.

Task / group	**Rose-GDN** $[-1,1]$	**GDN** $[-1,1]$	**DeltaProduct**$_K$ $[-1,1]$, $K \geq 2$
$\mathbb{Z}_2$	1[e]	1[a]	1[b]
$\mathbb{Z}_m$, $m \geq 3$	1[e]	2[c]	1[a]
Finite abelian group	1[e]	Not established	Not established
Finite solvable G	d[e]	Not established [e]	Not established [e]

Table 2: Formal-language evaluation. Scores are the best across four learning rates. Results are reported as scaled validation accuracy: 1 denotes perfect accuracy and 0 denotes chance performance. The highest score in each column is shown in bold. Training and evaluation lengths are specified in Appendix C.

Model	Parity	Modular arithmetic	
		Without brackets	With brackets
Transformer	0.114	0.148	0.138
Mamba-3 (SISO)	0.330	0.245	0.153
Gated DeltaProduct$_2$ $[-1,1]$	0.916	0.999	**0.258**
Gated DeltaNet $[-1,1]$	0.992	**1.000**	0.221
Rose-GDN $[-1,1]$	**1.000**	0.998	<u>0.249</u>

(2025), we extend the state learning rate to $\beta_t \in [0,2]$ for the delta-rule variants marked $[-1,1]$. This allows the eigenvalue $1-\beta_t$ of each generalized Householder factor to range over $[-1,1]$, permitting negative eigenvalues. All models are implemented using `flash-linear-attention` (Yang & Zhang, 2024) and trained using the `flame` (Zhang & Yang) framework. Full experimental details are provided in Appendix C.

5.1 State-tracking Capability

Experiments in this section focus on evaluating the state-tracking capability of Rose-GDN compared to other frontier models, including Transformer (Vaswani et al., 2017), Mamba-3 in its single-input, single-output (SISO) configuration (Lahoti et al., 2026), Gated DeltaProduct (Siems et al., 2026) with $K = 2$ Householder matrices, and Gated DeltaNet (Yang et al., 2025).

5.1.1 Chomsky Hierarchy

Generalization to sequences longer than those seen during training provides evidence that a model has learned the underlying transition rules of a formal-language task. Following the evaluation setup of Grazzi et al. (2025), we consider parity and modular arithmetic with and without brackets. We train models on sequences with lengths sampled uniformly from 3 to 40 and evaluate at lengths from 40 to 256 to assess generalization beyond the training range. Task descriptions and evaluation metrics are provided in Appendix C.

We report our results for those tasks in Table 2. It can be derived from our results that our method achieves the best parity score, Gated DeltaNet the best unbracketed-arithmetic score, and Gated DeltaProduct$_2$ the best bracketed-arithmetic score. Relative to Gated DeltaNet, Rose-GDN implied an enhancement on parity and bracketed modular arithmetic tasks, only losing in unbracketed modular arithmetic task by a negligible margin. Rose-GDN also suffer from a small drop from Gated

[a] Grazzi et al. (2025, Thm. 3); [b] Siems et al. (2026, Thm. 1); [c] Grazzi et al. (2025, Thm. 6); [d] Siems et al. (2026, Thm. 7); [e] This work, Props. 1–6; [f] This work, using the orthogonal-subgroup result of Siems et al. (2026, Thm. 4) and Dummit & Foote (2004); [g] Siems et al. (2026, Lem. 4); [h] Siems et al. (2026, Thm. 2 and Lem. 4).

Table 3: Finite abelian-group tracking. Each entry reports validation exact-sequence accuracy followed by per-position validation state accuracy, both as percentages. Training and evaluation use the same sequence length T within each column. Nonzero maxima for each task and metric are shown in bold. Experimental details are provided in Appendix C.

Model	$T = 20$	$T = 40$	$T = 60$	$T = 80$
		$G \cong \mathbb{Z}_3^2$		
Transformer	88.3 / 99.1	75.3 / 98.1	60.8 / 96.6	51.7 / 95.1
Mamba-3 (SISO)	99.0 / 99.9	0.00 / 37.1	0.00 / 35.9	0.00 / 34.6
GDN $[-1, 1]$	0.00 / 46.08	0.00 / 31.04	0.00 / 19.43	0.00 / 22.81
Gated DeltaProduct$_2$ $[-1, 1]$	**100.00 / 100.00**	99.90 / 99.99	**100.00 / 100.00**	0.00 / 38.85
Rose-GDN $[-1, 1]$	**100.00 / 100.00**	**100.00 / 100.00**	**100.00 / 100.00**	**98.93 / 99.97**
		$G \cong \mathbb{Z}_3^3$		
Transformer	90.8 / 98.9	0.00 / 51.1	0.00 / 33.1	0.00 / 38.1
Mamba-3 (SISO)	**100.0 / 100.0**	0.00 / 36.8	0.00 / 13.7	0.00 / 12.2
GDN $[-1, 1]$	0.00 / 25.42	0.00 / 14.78	0.00 / 11.16	0.00 / 9.23
Gated DeltaProduct$_2$ $[-1, 1]$	0.00 / 55.41	0.00 / 39.76	2.83 / 73.53	0.00 / 12.51
Rose-GDN $[-1, 1]$	**100.00 / 100.00**	**100.00 / 100.00**	**100.00 / 100.00**	**98.73 / 99.97**

DeltaProduct in modular arithmetics containing bracket. Overall, Rose-GDN is competitive with the strongest baselines under this length-generalization protocol, although bracketed arithmetic remains challenging for all evaluated models.

5.1.2 Abelian Group Tracking

Abelian-group tracking requires a model to predict the cumulative composition of group elements at each position in a sequence. We consider $\mathbb{Z}_3^n$ for $n = 2, 3$, with group orders 9 and 27, respectively. Composition in these groups is componentwise addition modulo 3, so increasing n increases the number of components that must be tracked simultaneously. These tasks examine the finite abelian-group tracking capability established by Theorem 1. Task definitions and evaluation protocol are provided in Appendix C.

Results. Table 3 evaluates $\mathbb{Z}_3^2$ and $\mathbb{Z}_3^3$ tracking. We report two metrics: *state accuracy*, the fraction of positions with a correct state prediction, and *exact-sequence accuracy*, the fraction of sequences for which every state is predicted correctly. For each experiment, training and evaluation use the same sequence length $T \in \{20, 40, 60, 80\}$. Table 3 reports exact-sequence accuracy followed by state accuracy.

On $\mathbb{Z}_3^2$, Rose-GDN maintains near-perfect accuracy through $T = 80$. Gated DeltaProduct achieves comparable performance through $T = 60$, but its exact-sequence accuracy drops to zero at $T = 80$. Both unrotated Gated DeltaNet variants have zero exact-sequence accuracy across all evaluated lengths. On $\mathbb{Z}_3^3$, the baselines generally struggle at $T \geq 40$, whereas Rose-GDN maintains high accuracy through $T = 80$. These results demonstrate reliable state tracking across both groups and all evaluated lengths, consistent with Rose-GDN's theoretical capacity for finite abelian-group tracking.

5.2 Natural Language Evaluation

We evaluate Rose-GDN on natural language tasks against Gated DeltaNet and Mamba-3 (SISO). All three models have approximately 340M parameters and are trained on the FineWeb-Edu 10B dataset (Penedo et al., 2024). Training and architectural details are provided in Appendix C.

5.2.1 Language Modeling and Common-sense Reasoning

We present in Table 4 model performances at the stated 340M-parameter scale on common language modelling and reasoning benchmarks. We report perplexity for language modeling, where

Table 4: Language-model comparison at the stated 340M-parameter scale. Lower perplexity and higher accuracy are better; accuracies are reported as percentages. acc_n denotes the normalized accuracy. Training budgets and evaluation settings are provided in Appendix C.

Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	SIQA acc ↑	BoolQ acc ↑	Avg. ↑
Gated DeltaNet	29.49	37.15	31.01	65.34	38.69	49.64	56.52	27.47	**39.51**	57.71	45.74
Mamba-3 (SISO)	29.21	**34.17**	32.60	**67.25**	**39.44**	49.41	**58.25**	27.22	37.51	**61.53**	**46.65**
Rose-GDN	**28.95**	34.90	**33.22**	65.02	39.12	**50.36**	56.90	**27.99**	36.95	60.06	46.20

Table 5: Needle-in-a-Haystack retrieval accuracy across context lengths. All models have approximately 340M parameters. Higher accuracy is better; the highest value in each column is shown in bold.

	S-NIAH-1 (pass-key retrieval)				**S-NIAH-2** (number in haystack)				**S-NIAH-3** (UUID in haystack)			
Model	**512**	**1K**	**2K**	**4K**	**512**	**1K**	**2K**	**4K**	**512**	**1K**	**2K**	**4K**
Gated DeltaNet	**1.000**	**1.000**	**0.998**	**0.994**	**1.000**	0.876	0.294	0.158	0.486	0.098	0.048	0.014
Mamba-3 (SISO)	0.992	0.996	0.762	0.312	0.992	0.982	0.576	0.180	0.110	0.010	0.000	0.002
Rose-GDN	**1.000**	0.998	**0.998**	0.982	**1.000**	**1.000**	**0.928**	**0.346**	**0.686**	**0.424**	**0.260**	**0.048**

lower values are better, and accuracy for the remaining evaluations, where higher values are better. Benchmark descriptions and evaluation settings are provided in Appendix C.

Rose-GDN improves on Gated DeltaNet in both perplexity evaluations and six of the eight accuracy benchmarks, increasing average accuracy from 45.74% to 46.20%. Against Mamba-3 (SISO), which posts the best average score, Rose-GDN still leads on LAMBADA accuracy, WinoGrande and ARC-challenge. Taken together, these results indicate competitive language modeling and reasoning performance alongside the state-tracking gains reported above.

5.2.2 Retrieval Capabilities

We evaluate retrieval using three single-needle variants of RULER's Needle-in-a-Haystack benchmark (Hsieh et al., 2024), at context lengths from 512 to $4{,}096$ tokens. Each example requires recovering a key-value pair embedded in a longer context; details are in Appendix C. In Table 5, Rose-GDN performs similarly to Gated DeltaNet on S-NIAH-1, with slightly lower accuracy at 1K and 4K. On the harder variants, it matches or exceeds both baselines at every length on S-NIAH-2 and achieves the highest accuracy throughout S-NIAH-3.

5.2.3 Efficiency Analysis

At the 340M model scale, we compare prefill, per-token decode, and total inference time on a single NVIDIA H100 GPU against Gated DeltaNet and Mamba-3, additionally including Transformer and Gated DeltaProduct$_2$. Figure 1 reports results for prefilling and generating L tokens; the protocol is in Appendix C.3.3. Rose-GDN incurs approximately $1.5\times$ Gated DeltaNet's prefill cost but matches its decode latency through 4K. Total latency remains within approximately 2% of Gated DeltaNet through 4K and 13% at 16K, and below Mamba-3 and Gated DeltaProduct$_2$ throughout.

6 Related Works

Positional encoding. RoPE uses fixed positional rotations, while Selective RoPE learns input-dependent increments and combines them with decay (Su et al., 2024; Movahedi et al., 2026). RetNet and ALiBi, as well as the scalar-gated recurrence in FoX, provide related retention-based constructions (Sun et al., 2023; Press et al., 2021; Lin et al., 2025). PaTH uses accumulated Householder transformations, with PaTH-FoX additionally incorporating scalar forgetting (Yang et al., 2026). For Rose-GDN, as shown in Section 3, the rotary transformation produces input-dependent distinct erase and write keys, which connects the recurrence to relative positional encoding.

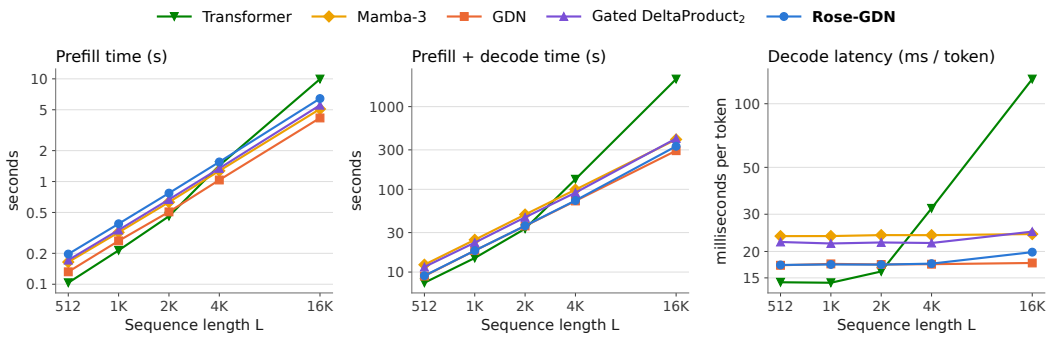

Figure 1: Inference latency on H100 80GB in bfloat16, prefilling and generating L tokens at target batch size 128. Batch adjustments and timing details are in Appendix C.3.3.

Memory editing and broader families. DeltaNet and Gated DeltaNet combine associative correction with structured recurrence, while DeltaProduct increases the number of inner updates (Schlag et al., 2021; Yang et al., 2024; 2025; Siems et al., 2026). PaTH's motivating linear recurrence already separates its Householder vector from the write key, and Gated DeltaNet-2 separates channelwise erase and write controls (Yang et al., 2026; Hatamizadeh et al., 2026). In Rose-GDN, the erase and write keys are linked by the same rotary transition: the erase key uses the cumulative rotation through the previous token, while the write key includes the rotation at the current token

State-tracking capabilities. State tracking requires a model to compose input-dependent transitions over long sequences, rather than only retain previous inputs. Fixed-depth Transformers and restricted linear recurrent models have known limitations on periodic languages and permutation-composition tasks (Hahn, 2020; Liu et al., 2022; Merrill & Sabharwal, 2023; Merrill et al., 2024; Sarrof et al., 2024). For Gated DeltaNet, allowing negative eigenvalues enables the alternating transition required for parity, while DeltaProduct represents richer transitions by composing multiple generalized Householder factors at each step (Grazzi et al., 2025; Siems et al., 2026). Bilinear RNNs can simulate arbitrary finite-state machines, whereas complex-diagonal SSMs extend to solvable groups and PD-SSM represents any N-state automaton in one N-dimensional layer using column-one-hot–diagonal transitions (Shakerinava et al., 2026; Terzic et al., 2026). As shown in Section 4, one Rose-GDN layer can track every finite abelian group, while additional layers extend this capability to finite solvable groups and arbitrary regular languages.

7 Concluding Remarks

We introduced Rose-GDN, which augments Gated DeltaNet with input-dependent block-diagonal rotations while retaining a single delta update per token, fixed-size memory, linear-time recurrence, and exact chunkwise-parallel computation. These rotations enrich the state-transition dynamics and induce distinct erase and write addresses in the cumulative coordinate frame. Under exact arithmetic, one layer can track any finite abelian group without negative eigenvalues in the generalized-Householder factor. Additional depth extends this capacity to finite solvable groups and all regular languages. Empirically, Rose-GDN improves abelian-group tracking and long-context retrieval while remaining competitive with Gated DeltaNet and Mamba-3 on language modeling and commonsense reasoning. Together, these findings demonstrate that structured rotations can broaden the state-tracking capabilities of delta-rule memory while preserving its efficient recurrent computation.

Limitations and future work. Our theory characterizes representational capacity under exact arithmetic; understanding how reliably these constructions can be learned and maintained in finite precision remains open. Beyond the present focus on block-diagonal planar rotations, future work could investigate richer structured orthogonal transitions, such as larger orthogonal blocks or learnable couplings between rotation subspaces.

AI USE STATEMENT

We used generative AI tools to assist with method implementation and code editing, provide feedback on experimental settings and suggest experimental parameters, modify scientific figures, improve manuscript readability, and identify potentially relevant literature. We did not use these tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, develop or write proofs, propose or refine research hypotheses, or interpret experimental results. All AI-assisted contributions were reviewed by the authors. In particular, we critically evaluated suggestions concerning experimental settings and implementation, checked the resulting code, reviewed all AI-assisted figures and manuscript revisions, and read and verified the suggested references before citing them. The authors take full responsibility for the final content of this work, including all text, theoretical claims, experimental results, and accompanying artifacts.

ETHICS STATEMENT

Given the nature of the work, we do not foresee any negative societal and ethical impacts of our work.

REPRODUCIBILITY STATEMENT

The Rose-GDN architecture and state-update formulation are described in Section 3, and the chunkwise-parallel formulation is provided in the appendix. Section 5 and Appendix C document the experimental tasks, model configurations, training hyperparameters, and evaluation protocols. Our experiments use synthetic formal-language and finite abelian-group tracking tasks, the publicly available FineWeb-Edu corpus, and established language modeling, commonsense reasoning, and RULER retrieval benchmarks. We provide task definitions and references to the datasets and benchmarks to support independent implementation and replication.

REFERENCES

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7432–7439, 2020.

Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 2924–2936, 2019.

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.

Grégoire Delétang, 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. *arXiv preprint arXiv:2207.02098*, 2022.

D.S. Dummit and R.M. Foote. *Abstract Algebra*. Wiley, 2004. ISBN 9788126532285.

Reza Ebrahimi and Roland Memisevic. Revisiting bi-linear state transitions in recurrent neural networks. *Advances in Neural Information Processing Systems*, 38:69615–69642, 2026.

Riccardo Grazzi, Julien Siems, Arber Zela, Jörg Franke, Frank Hutter, et al. Unlocking state-tracking in linear rnns through negative eigenvalues. In *International Conference on Learning Representations*, volume 2025, pp. 36565–36597, 2025.

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

Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated deltanet-2: Decoupling erase and write in linear attention. *arXiv preprint arXiv:2605.22791*, 2026.

John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Aakash Sunil Lahoti, Kevin Li, Berlin Chen, Caitlin Wang, Aviv Bick, Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. In *International Conference on Learning Representations*, volume 2026, pp. 86173–86202, 2026.

Zhixuan Lin, Evgenii Nikishin, Xu He, and Aaron Courville. Forgetting transformer: Softmax attention with a forget gate. In *International Conference on Learning Representations*, volume 2025, pp. 69704–69738, 2025.

Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

Oded Maler. On the krohn-rhodes cascaded decomposition theorem. In *Time for Verification: Essays in Memory of Amir Pnueli*, pp. 260–278. Springer, 2010.

Oded Maler and Amir Pnueli. On the cascaded decomposition of automata, its complexity and its application to logic (draft). 1994.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

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. *arXiv preprint arXiv:2404.08819*, 2024.

Sajad Movahedi, Timur Carstensen, Arshia Afzal, Frank Hutter, Antonio Orvieto, and Volkan Cevher. Selective rotary position embedding. In *International Conference on Learning Representations*, volume 2026, pp. 10219–10247, 2026.

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*, pp. 1525–1534, 2016.

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. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0970.

Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 8732–8740, 2020.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pp. 4463–4473, 2019.

Yash Sarrof, Yana Veitsman, and Michael Hahn. The expressive capacity of state space models: A formal language perspective. *Advances in Neural Information Processing Systems*, 37:41202–41241, 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, 2021.

Mehran Shakerinava, Behnoush Khavari, Siamak Ravanbakhsh, and Sarath Chandar. The expressive limits of diagonal ssms for state-tracking. In *International Conference on Learning Representations*, volume 2026, pp. 53488–53505, 2026.

Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. Deltaproduct: Improving state-tracking in linear rnns via householder products. *Advances in Neural Information Processing Systems*, 38:153738–153782, 2026.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 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.

Aleksandar Terzic, Nicolas Menet, Michael Hersche, Thomas Hofmann, and Abbas Rahimi. Structured sparse transition matrices to enable state tracking in state-space models. *Advances in Neural Information Processing Systems*, 38:83072–83111, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, january 2024. *URL https://github.com/fla-org/flash-linear-attention*, 22, 2024.

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*, 37:115491–115522, 2024.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025.

Songlin Yang, Yikang Shen, Kaiyue Wen, Shawn Tan, Mayank Mishra, Liliang Ren, Rameswar Panda, and Yoon Kim. Path attention: Position encoding via accumulating householder transformations. *Advances in Neural Information Processing Systems*, 38:62220–62247, 2026.

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.

Yu Zhang and Songlin Yang. Flame: Flash language modeling made easy, january 2025. *URL https://github.com/fla-org/flame*.

APPENDIX OF "ROTARY-STATE GATED DELTANET: ENHANCING STATE TRACKING THROUGH ROTATIONS"

A Additional Background **14**

A.1 Group theory 14

A.2 Formal language Theory 15

A.3 Chomsky Hierarchy 15

B Chunkwise Parallelization **16**

C Experimental Details **19**

C.1 Angle Parameterization 19

C.2 State Tracking 19

C.3 Natural Language Evaluation 20

D Proof of Theorem 1 **21**

E Proof of Theorem 2 **24**

A Additional Background

A.1 Group theory

Monoids and groups. A *monoid* is a pair $(M, \cdot)$ consisting of a set M and an associative binary operation

$$\cdot \colon M \times M \longrightarrow M$$

with an identity element $e \in M$, so $e \cdot a = a \cdot e = a$ for every $a \in M$. The identity is unique. A *group* is a monoid in which every element a has an inverse a^{-1} satisfying $a \cdot a^{-1} = a^{-1} \cdot a = e$; this inverse is also unique. A subgroup H of a group G, denoted by $H \leq G$, is a subset that is itself a group under the operation inherited from G. These standard definitions, together with the basic facts used below, can be found in Dummit & Foote (2004).

Finite abelian groups. A group G is *abelian* if $a \cdot b = b \cdot a$ for all $a, b \in G$. It is *finite* if its underlying set is finite, in which case its number of elements is its *order*, denoted by $|G|$. The fundamental theorem of finite abelian groups states that every finite abelian group is isomorphic to a direct product of cyclic groups of prime-power order:

$$G \cong \mathbb{Z}/p_1^{a_1}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_r^{a_r}\mathbb{Z},$$

where each p_i is prime and each a_i is positive. The multiset of prime-power factors is unique up to reordering. We use this theorem without proof; see Dummit & Foote (2004). Intuitively, the theorem says that a finite commutative group can be resolved into independent cyclic coordinates, each behaving like a counter modulo a prime power. A group element is then specified by the value of each coordinate, and the group operation updates all coordinates independently.

Solvable groups. Let G be a group. For $x, y \in G$, their *commutator* is

$$[x, y] = x^{-1}y^{-1}xy.$$

The *commutator subgroup*, or *derived subgroup*, is

$$G' = [G, G] = \langle [x, y] \mid x, y \in G \rangle .$$

It is a normal subgroup of G, and G/G' is abelian. Moreover, G' is the smallest normal subgroup N for which G/N is abelian. The *derived series* is defined recursively by

$$G^{(0)} = G, \qquad G^{(i+1)} = [G^{(i)}, G^{(i)}].$$

The group is *solvable* if $G^{(r)} = \{e\}$ for some $r \geq 0$; the smallest such r is its *derived length*. Equivalently, G is solvable if it has a finite subnormal series

$$\{e\} = G_r \triangleleft G_{r-1} \triangleleft \cdots \triangleleft G_1 \triangleleft G_0 = G$$

whose quotients G_i/G_{i+1} are abelian. For a finite solvable group, the series can be refined so that every nontrivial quotient is cyclic of prime order. Every abelian group is solvable. Subgroups, quotients, and extensions of solvable groups are solvable; in particular, if $N \triangleleft G$, then G is solvable if and only if both N and G/N are solvable. See Dummit & Foote (2004) for these equivalent characterizations and closure properties.

The symmetric group S_n. The *symmetric group* S_n consists of all permutations of $[n] = \{1, 2, \ldots, n\}$ under composition. It has order $|S_n| = n!$. Every permutation can be written as a product of disjoint cycles and also as a product of transpositions. Although the transposition factorization need not be unique, its parity is unique. The sign homomorphism

$$\operatorname{sgn} \colon S_n \longrightarrow \{1, -1\}$$

maps even permutations to 1 and odd permutations to -1. Its kernel is the *alternating group* A_n. For $n \geq 2$, A_n is a normal subgroup of index two, and therefore

$$|A_n| = \frac{n!}{2} \qquad \text{and} \qquad S_n/A_n \cong C_2.$$

The symmetric group S_n is solvable if and only if $n \leq 4$ (Dummit & Foote, 2004).

A.2 Formal language Theory

Formal language. Let the set Σ be a finite alphabet. A *word* is a finite sequence of symbols from Σ, and Σ^* denotes the set of all such words, including the empty word ε. A *formal language* over Σ is any subset $L \subseteq \Sigma^*$. Thus, a language may contain infinitely many words even though its alphabet is finite (Hopcroft et al., 2001).

A *generative grammar* is a finite description of such a language. Formally, a grammar is a tuple $\mathcal{G} = (N, \Sigma, P, S)$, where N is a finite set of nonterminal symbols disjoint from Σ, $S \in N$ is the start symbol, and P is a finite set of productions $\alpha \to \beta$. Here $\alpha, \beta \in (N \cup \Sigma)^*$ and α contains at least one nonterminal. A production may replace an occurrence of α by β inside a larger string. The nonterminals record unfinished structure during a derivation, whereas a word belongs to the language only after all nonterminals have been eliminated. For example, the two productions $S \to \varepsilon$ and $S \to aSb$ generate $\{a^n b^n : n \geq 0\}$. Restrictions on the allowed productions give the language classes in the Chomsky hierarchy (Chomsky, 1956; 1959; Hopcroft et al., 2001); their use as controlled tests of neural sequence models is discussed by Delétang et al. (2022).

Finite state automata. A deterministic finite-state automaton (FSA) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta)$, where Σ is a finite input alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, and $\delta : Q \times \Sigma \to Q$ is the transition function (Hopcroft et al., 2001). The set Σ^* consists of all finite words over Σ, including the empty word ε. We extend δ to words by defining

$$\delta(q, \varepsilon) = q, \qquad \delta(q, \boldsymbol{w}) = \delta(\delta(q, (w_1, ...w_{T-1})), w_T)$$

for $q \in Q$, $\boldsymbol{w} = (w_1, ..., w_T) \in \Sigma^*$, and $a \in \Sigma$. Thus, $\delta(q_0, \boldsymbol{w})$ is the state reached after reading $\boldsymbol{w}$. The automaton recognizes a *language* $L \subseteq \Sigma^*$ if there is an accepting set $R \subseteq Q$ such that

$$L = \{\boldsymbol{w} \in \Sigma^* : \delta(q_0, \boldsymbol{w}) \in R\}.$$

Languages recognized by finite-state automata are called *regular language*. The set

$$\mathcal{T}(\mathcal{A}) = \{\delta(\cdot, \boldsymbol{w}) : \boldsymbol{w} \in \Sigma^*\}$$

forms the automaton's *transition monoid* under function composition: it is closed under composition, composition is associative, and the empty word induces the identity transformation $\delta(\cdot, \varepsilon)$. This monoid is finite because Q is finite. If every input symbol induces a bijection on Q, the monoid also contains the inverse of each transformation and therefore forms a *group*.

A.3 Chomsky Hierarchy

Since a finite grammar can describe an infinite set of strings, the central question is therefore not whether a grammar is finite, but which restrictions on its production rules are sufficient to generate a given language. The *Chomsky hierarchy* answers this question by organizing formal languages according to the expressive power of their grammars (Chomsky, 1956; 1959; Hopcroft et al., 2001). Restricting the production rules limits which dependencies a grammar can express. Through the corresponding automata, these restrictions also specify the type of memory required to recognize the language. Let N and Σ denote the sets of nonterminal and terminal symbols, respectively. The four levels of the hierarchy are summarized in Table 6.

Table 6: The Chomsky hierarchy, ordered from the least to the most expressive language class.

Type	Language class	Representative rule restriction	Equivalent recognizer and memory
Type 3	Regular	$A \to aB$ or $A \to a$	Finite-state automaton; finite memory
Type 2	Context-free	$A \to \gamma$	Pushdown automaton; one unbounded stack
Type 1	Context-sensitive	$\alpha \to \beta$ with $\lvert\alpha\rvert \leq \lvert\beta\rvert$	Linear-bounded automaton; $O(T)$ memory
Type 0	Recursively enumerable	$\alpha \to \beta$	Turing machine; unbounded tape

Here $A, B \in N$, $a \in \Sigma$, and α, β, and γ are strings over $N \cup \Sigma$. The left-hand side of every production contains at least one nonterminal. For Type 3, the optional rule $S \to \varepsilon$ allows the empty word. For Type 1, the same rule is conventionally permitted when the start symbol S does not

appear on the right-hand side of any production. The corresponding language classes satisfy the strict inclusions

$$\mathrm{REG} \subsetneq \mathrm{CFL} \subsetneq \mathrm{CSL} \subsetneq \mathrm{RE}.$$

Moving from Type 3 to Type 0 relaxes the production rules and increases the available memory. Regular languages require only a finite state. Context-free languages can express stack-like dependencies, such as balanced parentheses. Context-sensitive languages can express dependencies requiring several synchronized quantities, such as $\{a^n b^n c^n : n \geq 1\}$. Type 0 contains all Turing-recognizable languages; the recognizer is allowed to run forever on a string outside the language.

For language modelling, the hierarchy provides a sequence of controlled problems with increasingly demanding memory requirements. Regular tasks such as parity and modular counting test whether a model can maintain and update a finite state. Context-free tasks such as Dyck languages test whether it can represent an unbounded stack, while context-sensitive tasks test more general forms of coordinated memory. These problems help distinguish fitting the training distribution from learning an update rule that generalizes to longer sequences (Delétang et al., 2022). A language model produces conditional probabilities rather than a binary accept-or-reject decision, so the hierarchy is not a literal classification of neural language models. It instead provides a principled benchmark for studying which computational structures their recurrent states can represent.

B Chunkwise Parallelization

We use the transpose convention of Equation 3, set $\mathbf{R}_0 = I$, and consider the Rotary DeltaNet recurrence

$$\mathbf{S}_t = \mathbf{S}_{t-1}\alpha_t(\mathbf{I} - \beta_t \mathbf{k}_t\mathbf{k}_t^\top)\mathbf{R}_t^\top + \beta_t \mathbf{v}_t\mathbf{k}_t^\top, \qquad \mathbf{S}_t \in \mathbb{R}^{d_v \times d_k}, \tag{9}$$

with output

$$\mathbf{o}_t = \mathbf{S}_t\mathbf{q}_t. \tag{10}$$

Define the rotated-frame state

$$\widetilde{\mathbf{S}}_t := \mathbf{S}_t\mathbf{R}_t\mathbf{R}_{t-1}\cdots\mathbf{R}_0. \tag{11}$$

Then $\mathbf{S}_t = \widetilde{\mathbf{S}}_t\mathbf{R}_0^\top\mathbf{R}_1^\top\cdots\mathbf{R}_t^\top$. Multiplying Equation 9 on the right by $\mathbf{R}_t\mathbf{R}_{t-1}\cdots\mathbf{R}_0$, we obtain

$$\begin{aligned}
\widetilde{\mathbf{S}}_t &= \alpha_t\mathbf{S}_{t-1}(\mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top)\mathbf{R}_t^\top\mathbf{R}_t\mathbf{R}_{t-1}\cdots\mathbf{R}_0 + \beta_t\mathbf{v}_t\mathbf{k}_t^\top\mathbf{R}_t\mathbf{R}_{t-1}\cdots\mathbf{R}_0 && (12)\\
&= \alpha_t\widetilde{\mathbf{S}}_{t-1}\left[\mathbf{R}_0^\top\mathbf{R}_1^\top\cdots\mathbf{R}_{t-1}^\top(\mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top)\mathbf{R}_{t-1}\cdots\mathbf{R}_0\right] + \beta_t\mathbf{v}_t\mathbf{k}_t^\top\mathbf{R}_t\mathbf{R}_{t-1}\cdots\mathbf{R}_0 && (13)\\
&= \alpha_t\widetilde{\mathbf{S}}_{t-1}\left(\mathbf{I} - \beta_t\mathbf{a}_t\mathbf{a}_t^\top\right) + \beta_t\mathbf{v}_t\mathbf{b}_t^\top, && (14)
\end{aligned}$$

where

$$\mathbf{a}_t := \mathbf{R}_0^\top\mathbf{R}_1^\top\cdots\mathbf{R}_{t-1}^\top\mathbf{k}_t, \qquad \mathbf{b}_t := \mathbf{R}_0^\top\mathbf{R}_1^\top\cdots\mathbf{R}_t^\top\mathbf{k}_t. \tag{15}$$

The output becomes

$$\mathbf{o}_t = \mathbf{S}_t\mathbf{q}_t = \widetilde{\mathbf{S}}_t\mathbf{R}_0^\top\mathbf{R}_1^\top\cdots\mathbf{R}_t^\top\mathbf{q}_t = \widetilde{\mathbf{S}}_t\mathbf{c}_t, \qquad \mathbf{c}_t := \mathbf{R}_0^\top\mathbf{R}_1^\top\cdots\mathbf{R}_t^\top\mathbf{q}_t. \tag{16}$$

Thus, in the rotating frame, Rotary DeltaNet becomes a gated delta-style recurrence with different erase and write keys:

$$\boxed{\widetilde{\mathbf{S}}_t = \alpha_t\widetilde{\mathbf{S}}_{t-1}(\mathbf{I} - \beta_t\mathbf{a}_t\mathbf{a}_t^\top) + \beta_t\mathbf{v}_t\mathbf{b}_t^\top} \tag{17}$$

where $\mathbf{a}_t$ is the erase key and $\mathbf{b}_t$ is the write key.

Now, split the sequence into chunks of length C. We use chunk-local indices and let $\widetilde{\mathbf{S}}_0 = \mathbf{S}_0$ be the incoming memory. Assume $\alpha_i > 0$ for the ratios below. Denote

$$\mathbf{A} = \begin{bmatrix}\mathbf{a}_1^\top\\ \vdots \\ \mathbf{a}_C^\top\end{bmatrix} \in \mathbb{R}^{C\times d_k}, \quad \mathbf{B} = \begin{bmatrix}\mathbf{b}_1^\top\\ \vdots \\ \mathbf{b}_C^\top\end{bmatrix} \in \mathbb{R}^{C\times d_k}, \mathbf{Q}_R = \begin{bmatrix}\mathbf{c}_1^\top\\ \vdots \\ \mathbf{c}_C^\top\end{bmatrix} \in \mathbb{R}^{C\times d_k}, \quad \mathbf{V} = \begin{bmatrix}\mathbf{v}_1^\top\\ \vdots \\ \mathbf{v}_C^\top\end{bmatrix} \in \mathbb{R}^{C\times d_v}.$$

Let

$$D_\beta := \mathrm{diag}(\beta_1, \ldots, \beta_C), \qquad \gamma_r := \prod_{i=1}^{r}\alpha_i, \qquad D_\gamma := \mathrm{diag}(\gamma_1, \ldots, \gamma_C).$$

Define the causal decay matrix

$$\Gamma_{ri} = \begin{cases} \gamma_r/\gamma_i, & r \geq i, \\ 0, & r < i, \end{cases} \quad \Gamma \in \mathbb{R}^{C\times C} \tag{18}$$

For strict lower-triangular terms, define

$$\mathbf{L}^{\Gamma}_{AA} := \text{strictLower}\left(\Gamma \odot \mathbf{A}\mathbf{A}^\top\right), \qquad \mathbf{L}^{\Gamma}_{AB} := \text{strictLower}\left(\Gamma \odot \mathbf{A}\mathbf{B}^\top\right). \tag{19}$$

Here $\mathbf{W} = \mathbf{T}_A\mathbf{A}$, with $\mathbf{T}_A = [I + \text{strictLower}(D_\beta\mathbf{A}\mathbf{A}^\top)]^{-1}D_\beta$. Also define $\overrightarrow{\widetilde{\mathbf{S}}_0} = \gamma_C\widetilde{\mathbf{S}}_0$ and $\overrightarrow{\mathbf{A}}_{r,:} = (\gamma_C/\gamma_r)\mathbf{A}_{r,:}$.

Then the chunkwise parallel form for state and output update of Rotary Gated DeltaNet is

$$\boxed{\begin{aligned} \widetilde{\mathbf{S}}_C &= \overrightarrow{\widetilde{\mathbf{S}}_0} + \mathbf{V}^\top D_\beta \overrightarrow{\mathbf{B}} + (\mathbf{U} - \overleftarrow{\mathbf{W}}\widetilde{\mathbf{S}}_0^\top)^\top \overrightarrow{\mathbf{A}}, \\ \mathbf{O} &= \overleftarrow{\mathbf{Q}}_R\widetilde{\mathbf{S}}_0^\top + \left(\mathbf{Q}_R\mathbf{B}^\top \odot \Gamma\right) D_\beta\mathbf{V} + \left(\mathbf{Q}_R\mathbf{A}^\top \odot \Gamma\right)\left(\mathbf{U} - \overleftarrow{\mathbf{W}}\widetilde{\mathbf{S}}_0^\top\right), \\ \mathbf{U} &= -(\mathbf{I} + D_\beta\mathbf{L}^{\Gamma}_{AA})^{-1} D_\beta\mathbf{L}^{\Gamma}_{AB} D_\beta\mathbf{V}, \\ \overleftarrow{\mathbf{W}} &= D_\gamma\mathbf{W}, \overrightarrow{\mathbf{U}}_{r,:} := \frac{\gamma_C}{\gamma_r}\mathbf{U}_{r,:}, \overrightarrow{\mathbf{B}}_{r,:} := \frac{\gamma_C}{\gamma_r}\mathbf{B}_{r,:}, \overleftarrow{\mathbf{Q}}_R := D_\gamma\mathbf{Q}_R. \end{aligned}}$$

The detailed derivation is presented below. For $1 \leq r \leq C$, let $\widetilde{\mathbf{S}}_r$ be the state after processing the first r tokens of the chunk, with initial state $\widetilde{\mathbf{S}}_0$. Expanding 17 gives

$$\widetilde{\mathbf{S}}_r = \widetilde{\mathbf{S}}_0 \underbrace{\prod_{i=1}^{r} \alpha_i(\mathbf{I} - \beta_i\mathbf{a}_i\mathbf{a}_i^\top)}_{\mathbf{P}_r} + \underbrace{\sum_{i=1}^{r} \beta_i\mathbf{v}_i\mathbf{b}_i^\top \prod_{j=i+1}^{r} \alpha_j(\mathbf{I} - \beta_j\mathbf{a}_j\mathbf{a}_j^\top)}_{\mathbf{H}_r}. \tag{20}$$

The erase-side product $\mathbf{P}_r$ uses the same generalized Householder structure as DeltaNet, but with rotated erase keys $\mathbf{a}_i$. We have

$$\mathbf{P}_r = \gamma_r(\mathbf{I} - \sum_{i=1}^{r} \mathbf{w}_i\mathbf{a}_i^\top). \tag{21}$$

In matrix form,

$$\mathbf{W} = \mathbf{T}_A\mathbf{A}, \qquad \mathbf{T}_A = \left[I + \text{strictLower}\left(D_\beta\mathbf{A}\mathbf{A}^\top\right)\right]^{-1} D_\beta. \tag{22}$$

Therefore

$$\mathbf{P}_C = \gamma_C\mathbf{I} - \overleftarrow{\mathbf{W}}^\top \overrightarrow{\mathbf{A}} \tag{23}$$

where $\overleftarrow{\mathbf{W}}_{r,:} = \gamma_r\mathbf{W}_{r,:}$ and $\overrightarrow{\mathbf{A}}_{r,:} := \frac{\gamma_C}{\gamma_r}\mathbf{A}_{r,:}$.

Now, consider the accumulated write contribution $\mathbf{H}_r$:

$$\mathbf{H}_r = \alpha_r\mathbf{H}_{r-1}(I - \beta_r\mathbf{a}_r\mathbf{a}_r^\top) + \beta_r\mathbf{v}_r\mathbf{b}_r^\top, \qquad \mathbf{H}_0 = 0. \tag{24}$$

Let $\gamma_r := \prod_{\ell=1}^{r} \alpha_\ell$. We use the representation

$$\mathbf{H}_r = \sum_{i=1}^{r} \frac{\gamma_r}{\gamma_i}\beta_i\mathbf{v}_i\mathbf{b}_i^\top + \sum_{j=1}^{r} \frac{\gamma_r}{\gamma_j}\mathbf{u}_j\mathbf{a}_j^\top, \tag{25}$$

where

$$\mathbf{u}_r = -\beta_r\left[\sum_{i<r} \frac{\gamma_r}{\gamma_i}\beta_i\mathbf{v}_i(\mathbf{b}_i^\top\mathbf{a}_r) + \sum_{j<r} \frac{\gamma_r}{\gamma_j}\mathbf{u}_j(\mathbf{a}_j^\top\mathbf{a}_r)\right]. \tag{26}$$

Equivalently,

$$\mathbf{u}_r = -\beta_r \sum_{i<r} \frac{\gamma_r}{\gamma_i} \beta_i \mathbf{v}_i (\mathbf{a}_r^\top \mathbf{b}_i) - \beta_r \sum_{j<r} \frac{\gamma_r}{\gamma_j} \mathbf{u}_j (\mathbf{a}_r^\top \mathbf{a}_j). \tag{27}$$

This is a lower-triangular recurrence because $\mathbf{u}_r$ only depends on $\{\mathbf{u}_j : j < r\}$.

In matrix form, let

$$\mathbf{L}_{AA}^{\Gamma} := \text{strictLower}\left(\Gamma \odot \mathbf{A}\mathbf{A}^\top\right), \qquad \mathbf{L}_{AB}^{\Gamma} := \text{strictLower}\left(\Gamma \odot \mathbf{A}\mathbf{B}^\top\right). \tag{28}$$

Then

$$(\mathbf{L}_{AA}^{\Gamma})_{rj} = \frac{\gamma_r}{\gamma_j} \mathbf{a}_r^\top \mathbf{a}_j, \qquad j < r, \tag{29}$$

and

$$(\mathbf{L}_{AB}^{\Gamma})_{ri} = \frac{\gamma_r}{\gamma_i} \mathbf{a}_r^\top \mathbf{b}_i, \qquad i < r. \tag{30}$$

Stack the correction vectors into

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_1^\top \\ \vdots \\ \mathbf{u}_C^\top \end{bmatrix} \in \mathbb{R}^{C \times d_v}. \tag{31}$$

Then equation 26 becomes

$$\mathbf{U} = -D_\beta \mathbf{L}_{AB}^{\Gamma} D_\beta \mathbf{V} - D_\beta \mathbf{L}_{AA}^{\Gamma} \mathbf{U}. \tag{32}$$

Rearranging gives the triangular linear system

$$(\mathbf{I} + D_\beta \mathbf{L}_{AA}^{\Gamma}) \mathbf{U} = -D_\beta \mathbf{L}_{AB}^{\Gamma} D_\beta \mathbf{V}. \tag{33}$$

Therefore,

$$\boxed{\mathbf{U} = -(\mathbf{I} + D_\beta \mathbf{L}_{AA}^{\Gamma})^{-1} D_\beta \mathbf{L}_{AB}^{\Gamma} D_\beta \mathbf{V}.} \tag{34}$$

Therefore

$$\mathbf{H}_C = \mathbf{V}^\top D_\beta \overrightarrow{\mathbf{B}} + \mathbf{U}^\top \overrightarrow{\mathbf{A}} \tag{35}$$

where the matrices $\overrightarrow{\mathbf{A}}$ and $\overrightarrow{\mathbf{B}}$ are defined as

$$\overrightarrow{\mathbf{A}}_{r,:} := \frac{\gamma_C}{\gamma_r} \mathbf{A}_{r,:}, \qquad \overrightarrow{\mathbf{B}}_{r,:} := \frac{\gamma_C}{\gamma_r} \mathbf{B}_{r,:}. \tag{36}$$

Chunkwise state update. Define $\overrightarrow{\widetilde{\mathbf{S}}_0} := \gamma_C \widetilde{\mathbf{S}}_0$. Then the final state of the chunk is

$$\widetilde{\mathbf{S}}_C = \widetilde{\mathbf{S}}_0 (\gamma_C \mathbf{I} - \overleftarrow{\mathbf{W}}^\top \overrightarrow{\mathbf{A}}) + \mathbf{V}^\top D_\beta \overrightarrow{\mathbf{B}} + \mathbf{U}^\top \overrightarrow{\mathbf{A}} \tag{37}$$

$$= \overrightarrow{\widetilde{\mathbf{S}}_0} + \mathbf{V}^\top D_\beta \overrightarrow{\mathbf{B}} + (\mathbf{U} - \overleftarrow{\mathbf{W}} \widetilde{\mathbf{S}}_0^\top)^\top \overrightarrow{\mathbf{A}} \tag{38}$$

Chunkwise output formula. For $1 \le r \le C$, we have

$$\mathbf{o}_r = \widetilde{\mathbf{S}}_r \mathbf{c}_r \tag{39}$$

$$= \widetilde{\mathbf{S}}_0 \mathbf{P}_r \mathbf{c}_r + \mathbf{H}_r \mathbf{c}_r \tag{40}$$

$$= \gamma_r \widetilde{\mathbf{S}}_0 \mathbf{c}_r + (-\gamma_r \widetilde{\mathbf{S}}_0 \sum_{i=1}^{r} \mathbf{w}_i \mathbf{a}_i^\top + \sum_{i=1}^{r} \frac{\gamma_r}{\gamma_i} \beta_i \mathbf{v}_i \mathbf{b}_i^\top + \sum_{j=1}^{r} \frac{\gamma_r}{\gamma_j} \mathbf{u}_j \mathbf{a}_j^\top) \mathbf{c}_r \tag{41}$$

$$= \gamma_r \widetilde{\mathbf{S}}_0 \mathbf{c}_r + \sum_{i=1}^{r} (\frac{\gamma_r}{\gamma_i} \mathbf{b}_i^\top \mathbf{c}_r) \beta_i \mathbf{v}_i + \sum_{i=1}^{r} (\frac{\gamma_r}{\gamma_i} \mathbf{a}_i^\top \mathbf{c}_r)(\mathbf{u}_i - \widetilde{\mathbf{S}}_0 \gamma_i \mathbf{w}_i) \tag{42}$$

Let $\mathbf{M} = (m_{ri}) \in \mathbb{R}^{C \times C}$ be the causal mask:

$$m_{ri} = \begin{cases} 1, & r \ge i, \\ 0, & r < i. \end{cases}$$

Define the left-decayed rotated query matrix $\overleftarrow{\mathbf{Q}}_R := D_\gamma \mathbf{Q}_R$. Then the chunk output is

$$\boxed{\mathbf{O} = \overleftarrow{\mathbf{Q}}_R \widetilde{\mathbf{S}}_0^\top + \left(\mathbf{Q}_R \mathbf{B}^\top \odot \Gamma\right) D_\beta \mathbf{V} + \left(\mathbf{Q}_R \mathbf{A}^\top \odot \Gamma\right)\left(\mathbf{U} - \overleftarrow{\mathbf{W}} \widetilde{\mathbf{S}}_0^\top\right)} \tag{43}$$

The state passed to the next chunk is $\mathbf{S}_C = \widetilde{\mathbf{S}}_C \mathbf{R}_1^\top \cdots \mathbf{R}_C^\top$. Both triangular systems have unit diagonal and are evaluated by triangular solves. If a decay gate is zero, interpret each ratio γ_r/γ_i as $\prod_{\ell=i+1}^{r} \alpha_\ell$; the resulting formulas extend continuously to zero gates.

C Experimental Details

This appendix describes the parameterization for the rotated angle, experimental tasks and the available training and evaluation settings for Section 5. Every run is deployed on a single NVIDIA H100 GPU.

C.1 Angle Parameterization

Let $\mathbf{x}_t \in \mathbb{R}^d$ denote the pre-normalized layer input at position t; we suppress the head index. Following the input-dependent discretization used in Mamba-3 (Lahoti et al., 2026), the decay gate and rotation angles share a positive step size

$$\Delta_t = \mathrm{softplus}(\mathbf{w}_\Delta^\top \mathbf{x}_t + b_\Delta) \in (0, \infty), \tag{44}$$

where $\mathbf{w}_\Delta \in \mathbb{R}^d$ and $b_\Delta \in \mathbb{R}$ are learned. A learned rate $A > 0$, stored as $\log A$, gives the decay gate

$$\alpha_t = \exp(-A\Delta_t) \in (0, 1). \tag{45}$$

The correction gate is parameterized separately as $\beta_t = \sigma(\mathbf{w}_\beta^\top \mathbf{x}_t) \in (0, 1)$, with $\mathbf{w}_\beta \in \mathbb{R}^d$, or as $2\sigma(\mathbf{w}_\beta^\top \mathbf{x}_t) \in (0, 2)$ when reflections are allowed. The per-block angular velocity is

$$\boldsymbol{\omega}_t = \pi \tanh(\mathbf{W}_\theta \mathbf{x}_t) \in (-\pi, \pi)^r, \qquad \mathbf{W}_\theta \in \mathbb{R}^{r \times d}, \tag{46}$$

where $\tanh$ acts elementwise. Integrating this velocity over the shared step gives the angles in equation 4:

$$\boldsymbol{\theta}_t = \Delta_t \boldsymbol{\omega}_t = \pi \Delta_t \tanh(\mathbf{W}_\theta \mathbf{x}_t) \in (-\pi\Delta_t, \pi\Delta_t)^r. \tag{47}$$

Thus the angular projection produces one scalar per 2×2 block and per head, using rd learned weights per head, with no bias or fixed frequency table. No explicit positional input enters this parameterization.

C.2 State Tracking

For state tracking experiments, we compare Rose-GDN with Transformer (Vaswani et al., 2017), Mamba-3 (Lahoti et al., 2026), Gated DeltaProduct (Siems et al., 2026), and Gated DeltaNet (Yang et al., 2025).

C.2.1 Chomsky Hierarchy

We evaluate parity and modular arithmetic with and without brackets, following the task families used by Grazzi et al. (2025) and introduced in the formal-language benchmark of Delétang et al. (2022).

Parity. Inputs are binary strings $x_1 \cdots x_T$, with target $y_T = (\sum_{t=1}^{T} x_t) \bmod 2$. The vocabulary has two symbols and uniform random-guess accuracy is $1/2$. Reading 0 preserves the state; reading 1 switches even and odd.

Modular arithmetic without brackets. Inputs are arithmetic expressions over residues $0, \ldots, 4$, with addition, subtraction, and multiplication. Together with the equals and padding tokens, these give a vocabulary of ten symbols. The target is the value of the expression modulo 5, evaluated according to the benchmark's operator precedence. For example, $2 - 3 - 3 \times 2 \equiv 3 \pmod 5$. There are five output classes, with chance accuracy $1/5$.

Modular arithmetic with brackets. This variant adds opening and closing parentheses, increasing the vocabulary to twelve symbols. Parentheses alter evaluation order: for example, $(2 + 3) \times 4 \equiv 0 \pmod 5$. The target remains one of five residues, with uniform random-guess accuracy $1/5$.

Following Grazzi et al. (2025), models are trained for 100,000 steps with batch size 1,024. Training lengths are sampled uniformly from 3 to 40. The stated configuration uses hidden size 128, four heads of dimension 32, and a short convolution of width 4. Neither an MLP block nor an output gate is used. Parity uses 2 layers, while both arithmetic tasks use 3 layers. We train each model with AdamW using a batch size of 1024, weight decay 0.1 and a gradient clipping norm of 1.0. We sweep learning rates $\{10^{-2}, 10^{-3}, 5 \times 10^{-4}, 10^{-4}\}$. The learning rate increases linearly during the first 10% steps of training (10,000 steps), then follows a cosine decay to 10^{-6}.

For all three tasks, we evaluate sequence lengths in $\{40, 64, 96, 128, 160, 192, 256\}$. We report scaled accuracy $a_{\text{scaled}} = \frac{a - a_{\text{chance}}}{1 - a_{\text{chance}}}$ where $a_{\text{chance}} = 1/2$ for parity and $1/5$ for both arithmetic tasks. A scaled score of one denotes perfect accuracy, while zero denotes random-guess performance.

C.2.2 Abelian group tracking

Given inputs $x_1, \ldots, x_T$ from an abelian group, the model predicts $g_t = x_t \cdot g_{t-1} = x_t \cdot x_{t-1} \cdots x_1$ ($g_0 = e$) at every position t. The experiments use $G = \mathbb{Z}_3^n$ for $n \in \{2, 3\}$. Each group element is an n-tuple with coordinates in $\{0, 1, 2\}$, and composition is componentwise addition modulo 3. Each input token identifies one group element, and each output label identifies one cumulative state. The input vocabulary and output-label set therefore have size 9 for $\mathbb{Z}_3^2$ and 27 for $\mathbb{Z}_3^3$.

We use matched training and evaluation lengths $T \in \{20, 40, 60, 80\}$, as reported in Table 3. Each model uses one layer with four heads of dimension 32 and a short convolution of width 4. Neither an output gate nor an MLP block is used. We retain the AdamW optimizer, weight decay, gradient clipping, and warm-up–cosine schedule used for the Chomsky tasks, but train for 10,000 steps, including 1,000 warm-up steps. The sweeps use the same four learning rates as in the Chomsky Hierarchy experiments (Appendix C.2.1).

We report per-position state accuracy, the fraction of correctly predicted prefixes, and exact-sequence accuracy, the fraction of sequences with every prefix predicted correctly. For M evaluation sequences of length T, these are

$$a_{\text{state}} = \frac{1}{MT} \sum_{j=1}^{M} \sum_{t=1}^{T} \mathbf{1}\{\hat{g}_{j,t} = g_{j,t}\},$$

and

$$a_{\text{sequence}} = \frac{1}{M} \sum_{j=1}^{M} \mathbf{1}\{\hat{g}_{j,t} = g_{j,t} \text{ for all } t \in \{1, \ldots, T\}\}.$$

Table 3 reports both metrics as percentages, with exact-sequence accuracy first.

C.3 Natural Language Evaluation

We compare Gated DeltaNet (Yang et al., 2025), Mamba-3 (SISO) (Lahoti et al., 2026), and Rose-GDN at the 340M-parameter scale.

Training configuration. All three models are trained on the FineWeb-Edu 10B dataset (Penedo et al., 2024) for 20,480 optimization steps at a sequence length of 4,096 tokens. Training uses 4 NVIDIA H100 GPUs, with a per-device batch size of 4 and eight gradient-accumulation steps, giving an effective batch size of 128 sequences (524,288 tokens). We use AdamW with peak learning rate 4×10^{-4}, weight decay 0.1, and a gradient-clipping norm of 1.0. The schedule uses 1,024 warm-up steps, followed by cosine decay to 4×10^{-5}.

Architectural configuration. All models use 24 layers, a hidden size of 1,024, and eight heads. For Gated DeltaNet and Rose-GDN, the key and value dimensions are both 128 per head, with no

value expansion. The vocabulary contains 32,000 tokens using Llama2 tokenizer. Gated DeltaNet and Mamba-3 (SISO) use an output gate, while Rose-GDN does not.

C.3.1 Language Modeling and Common-sense Reasoning

We use benchmarks from the DeltaNet line of work (Yang et al., 2024; 2025). For language modeling, we evaluate WikiText (Merity et al., 2017) and LAMBADA (Paperno et al., 2016). WikiText measures prediction of held-out text. LAMBADA evaluates prediction of the final word of a passage using its preceding context. We report perplexity on both benchmarks and accuracy on LAMBADA. The common-sense reasoning task suite includes PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), ARC-easy and ARC-challenge (Clark et al., 2018), SIQA (Sap et al., 2019), and BoolQ (Clark et al., 2019). These tasks cover physical and social commonsense, plausible event continuation, pronoun resolution, science questions, and reading comprehension. Evaluation is zero-shot and runs with a batch size of 128 under one single NVIDIA H100 GPU.

C.3.2 Long-context Retrieval

We evaluate long-context retrieval using the single-needle Needle-in-a-Haystack tasks from RULER (Hsieh et al., 2024). We use the single-needle variants of the Needle-in-a-Haystack (NIAH) benchmark from RULER (Hsieh et al., 2024). Each example hides a key-value pair (the needle) at some position inside a long filler text (the haystack). After reading the full context, the model is given the key and must reproduce the associated value. We evaluate context lengths of 512, 1024, 2048, and 4096 tokens in Table 5. All runs are deployed on one single NVIDIA H100 GPU.

C.3.3 Efficiency Analysis

Following the protocol of Lahoti et al. (2026), for each input length $L \in \{512, 1024, 2048, 4096, 16384\}$, models prefill a batch of 128 sequences and greedily decode L further tokens in bfloat16 on a single H100 80GB GPU. We report median wall-clock times over three repetitions. Because the Transformer at $L = 4096$ and $L = 16384$ and Mamba-3 at $L = 16384$ do not fit on the GPU at batch 128, we measure the Transformer at batch 64 for $L = 4096$ and at batch 16 for $L = 16384$, scaling its results by 2 and 8 respectively, and we measure Mamba-3 at batch 32 for $L = 16384$, scaling its resutls by 4.

D Proof of Theorem 1

We use the left-prefix convention $g_0 = e$ and $g_t = x_t g_{t-1}$. Choose $\alpha_t = 1$ and write the recurrence as

$$\mathbf{S}_t = \mathbf{S}_{t-1}\mathbf{T}_t + \mathbf{B}_t, \qquad \mathbf{T}_t = (\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top)\mathbf{R}_t^\top, \qquad \mathbf{B}_t = \beta_t \mathbf{v}_t \mathbf{k}_t^\top. \tag{48}$$

All initial memories below are fixed parameters encoding the identity, as allowed in the theorem's convention. Each native query output is a numerical code; a fixed pointwise decoder converts that code to a group element.

(i) Finite abelian groups. Let $G \cong \prod_{j=1}^{r} \mathbb{Z}_{m_j}$ be nontrivial, with factors of order one omitted. Such a decomposition exists by the structure theorem for finite abelian groups (Dummit & Foote, 2004). Write $a_j(g)$ for the jth coordinate of g, represented in $\{0, \dots, m_j - 1\}$, and put $p = 2r$, $d_v = p$, and $n = p + 2$. Define

$$\boldsymbol{\rho}(g) = \operatorname{diag}\big(\mathbf{R}_{2\pi a_1(g)/m_1}, \dots, \mathbf{R}_{2\pi a_r(g)/m_r}\big), \qquad \mathbf{c} = (1, 0, \dots, 1, 0)^\top \in \mathbb{R}^p. \tag{49}$$

This uses the planar modular-counting construction of Ebrahimi & Memisevic (2026, Proposition 2 and Appendix A.2). Indeed, $a_j(gh) = a_j(g) + a_j(h) \pmod{m_j}$ and $\mathbf{R}_\theta \mathbf{R}_\varphi = \mathbf{R}_{\theta+\varphi}$ imply $\boldsymbol{\rho}(gh) = \boldsymbol{\rho}(g)\boldsymbol{\rho}(h)$ and $\boldsymbol{\rho}(e) = \mathbf{I}_p$. Moreover,

$$\boldsymbol{\phi}(g) = \boldsymbol{\rho}(g)^\top \mathbf{c} = \big(\cos(2\pi a_j(g)/m_j), -\sin(2\pi a_j(g)/m_j)\big)_{j=1}^{r} \tag{50}$$

is injective: equality of each cosine-sine pair forces equality of the corresponding angles modulo 2π, hence of every cyclic coordinate.

We next realize this code with nonzero writes. Choose a matrix $\mathbf{U} \in \mathbb{R}^{p\times 2}$ of column rank two, and, for each $x \in G$, choose a unit vector $\boldsymbol{\kappa}(x) \in \mathbb{R}^2$ and a gate $\beta(x) \in (0, 2)$. These choices may depend on the input; all gates can in particular be chosen in $(0, 1)$. Set

$$\begin{aligned} \mathbf{k}_t &= \begin{pmatrix} \mathbf{0}_p \\ \boldsymbol{\kappa}(x_t) \end{pmatrix}, \qquad \mathbf{v}_t = \mathbf{U}\boldsymbol{\kappa}(x_t), \quad \mathbf{q}_t = \mathbf{q}_* = \begin{pmatrix} \mathbf{c} \\ \mathbf{0}_2 \end{pmatrix}, \\ \mathbf{R}_t &= \operatorname{diag}(\boldsymbol{\rho}(x_t), \mathbf{I}_2), \quad \beta_t = \beta(x_t), \qquad \mathbf{S}_0 = [\,\mathbf{I}_p \mid \mathbf{U}\,]. \end{aligned} \tag{51}$$

The last two coordinates form an additional rotary pair with angle zero. The key is unit, and both the value and the write are nonzero because $\mathbf{U}$ has trivial kernel on $\mathbb{R}^2$ and $\beta_t > 0$. These are valid linear key, value, and query projections: for one-hot inputs $\mathbf{x}_t = \mathbf{e}_{x_t} \in \mathbb{R}^M$, $M = |G|$, let $\mathbf{K} \in \mathbb{R}^{2\times M}$ have columns $\boldsymbol{\kappa}(x)$ and take

$$\mathbf{W}_k = \begin{pmatrix} 0_{p\times M} \\ \mathbf{K} \end{pmatrix}, \qquad \mathbf{W}_v = \mathbf{U}\mathbf{K}, \qquad \mathbf{W}_q = \mathbf{q}_* \mathbf{1}_M^\top. \tag{52}$$

Then $\mathbf{k}_t = \mathbf{W}_k \mathbf{x}_t$, $\mathbf{v}_t = \mathbf{W}_v \mathbf{x}_t$, and $\mathbf{q}_t = \mathbf{W}_q \mathbf{x}_t$. In particular,

$$\mathbf{W}_k^\top \mathbf{W}_q = 0, \qquad \mathbf{B}_t \mathbf{q}_t = \beta_t \mathbf{v}_t \mathbf{x}_t^\top \mathbf{W}_k^\top \mathbf{W}_q \mathbf{x}_t = 0. \tag{53}$$

Input-dependent angles and gates are fixed functions of the finite input alphabet, so their selection never requires the previous memory.

For brevity write $\boldsymbol{\kappa}_t = \boldsymbol{\kappa}(x_t)$. Multiplying the factors in the order of equation 48 gives

$$\mathbf{T}_t = \begin{pmatrix} \boldsymbol{\rho}(x_t)^\top & 0 \\ 0 & \mathbf{I}_2 - \beta_t \boldsymbol{\kappa}_t \boldsymbol{\kappa}_t^\top \end{pmatrix}, \qquad \mathbf{B}_t = [\,\mathbf{0}_{p\times p} \mid \beta_t \mathbf{U}\boldsymbol{\kappa}_t \boldsymbol{\kappa}_t^\top\,] \neq \mathbf{0}. \tag{54}$$

To connect this to the homogeneous construction, put $\overline{\mathbf{S}}_0 = [\,\mathbf{I}_p \mid \mathbf{0}\,]$ and $\overline{\mathbf{S}}_t = \overline{\mathbf{S}}_{t-1}\mathbf{T}_t$. Induction gives $\overline{\mathbf{S}}_t = [\,\boldsymbol{\rho}(g_t)^\top \mid \mathbf{0}\,]$, since

$$\boldsymbol{\rho}(g_{t-1})^\top \boldsymbol{\rho}(x_t)^\top = \big(\boldsymbol{\rho}(x_t)\boldsymbol{\rho}(g_{t-1})\big)^\top = \boldsymbol{\rho}(x_t g_{t-1})^\top.$$

Let $\mathbf{M}_* = [\,\mathbf{0} \mid \mathbf{U}\,]$. The second block of $\mathbf{M}_*\mathbf{T}_t + \mathbf{B}_t$ equals

$$\mathbf{U}(\mathbf{I}_2 - \beta_t \boldsymbol{\kappa}_t \boldsymbol{\kappa}_t^\top) + \beta_t \mathbf{U}\boldsymbol{\kappa}_t \boldsymbol{\kappa}_t^\top = \mathbf{U},$$

so $\mathbf{M}_*\mathbf{T}_t + \mathbf{B}_t = \mathbf{M}_*$. As $\mathbf{S}_0 = \overline{\mathbf{S}}_0 + \mathbf{M}_*$, induction in equation 48 now proves the exact invariant

$$\mathbf{S}_t = \overline{\mathbf{S}}_t + \mathbf{M}_* = [\,\boldsymbol{\rho}(g_t)^\top \mid \mathbf{U}\,]. \tag{55}$$

Thus the nonzero write replenishes the part removed from the last block without altering the group encoding.

The cancellation in equation 53 also persists through all later transitions. Specifically, with $\mathbf{P} = \operatorname{diag}(\mathbf{I}_p, 0_{2\times 2})$, the block formulas give $\mathbf{B}_s\mathbf{P} = 0$, $\mathbf{T}_t\mathbf{P} = \mathbf{P}\mathbf{T}_t$, and $\mathbf{P}\mathbf{q}_* = \mathbf{q}_*$. Therefore, for every $s \le t$,

$$\mathbf{B}_s \mathbf{T}_{s+1} \cdots \mathbf{T}_t \mathbf{q}_* = \mathbf{B}_s \mathbf{P} \mathbf{T}_{s+1} \cdots \mathbf{T}_t \mathbf{q}_* = \mathbf{0}. \tag{56}$$

This proves that every historical write is invisible to the query, including after propagation through subsequent transitions.

The native readout consequently satisfies

$$D_{\mathrm{qry}}(\mathbf{S}_t) = \mathbf{S}_t \mathbf{q}_* = \boldsymbol{\rho}(g_t)^\top \mathbf{c} = \boldsymbol{\phi}(g_t). \tag{57}$$

A concrete group-label decoder is $\mathrm{dec}_G(\mathbf{o}) = \operatorname{argmax}_{h\in G} \boldsymbol{\phi}(h)^\top \mathbf{o}$, with arbitrary tie-breaking away from reachable outputs. For $\mathbf{o} = \phi(g)$, its score is

$$\boldsymbol{\phi}(h)^\top \boldsymbol{\phi}(g) = \sum_{j=1}^{r} \cos\left(\frac{2\pi(a_j(h) - a_j(g))}{m_j}\right) \le r,$$

with equality precisely when $h = g$. Hence the decoder returns every prefix exactly. The reachable memory set is $\{[\,\boldsymbol{\rho}(g)^\top \mid \mathbf{U}\,] : g \in G\}$, which has exactly $|G|$ elements; each is reachable by the one-symbol input g. Finally, $\mathbf{R}_t^\top \in \mathcal{T}_n$, $\|\mathbf{k}_t\|_2 = 1$, and $1 - \beta_t \in (-1, 1)$ show that $\mathbf{T}_t \in \mathcal{F}_n((-1; 1])$. We have used one layer and $n = 2r + 2$.

For the trivial group with $r = 0$, take $n = 2$, $d_v = 1$, $\mathbf{k} = \mathbf{e}_1$, $\mathbf{q} = \mathbf{e}_2$, $\mathbf{v} = 1$, $\mathbf{R} = \mathbf{I}_2$, and $\mathbf{S}_0 = (1, 0)$, with $0 < \beta < 2$. Then $\mathbf{S}_t = (1, 0)$, the write $(\beta, 0)$ is nonzero, and the constant query output encodes the unique group element. This proves (i). Notice that the identity-encoding initialization is substantive: setting $\mathbf{S}_0 = 0$ with the nontrivial-group parameters above would leave all counting columns zero. The construction uses the fixed initialization stated in the theorem, without an extra token or recurrent layer.

(ii) Finite solvable groups. We adapt the quotient-and-subgroup construction in the sufficiency direction of Shakerinava et al. (2026, Theorem 2 and Appendix C.6, equations (15)–(17)) to left multiplication and to the layers in (i). We prove the stronger assertion that the stack has an injective canonical memory encoding $E_G(g)$, starts at $E_G(e)$, sends $E_G(g)$ to $E_G(xg)$ on input x, and recovers g from its current query codes. Part (i) establishes this assertion for finite abelian groups.

First consider $N \trianglelefteq G$ with abelian quotient $Q = G/N$ and quotient map $\pi : G \to Q$. Suppose a stack with L layers has the stated properties for N. Choose a section $s : Q \to G$ with $\pi(s(q)) = q$ and $s(e_Q) = e_G$. Every $g \in G$ is uniquely

$$g = s(q)n, \qquad q = \pi(g), \qquad n = s(q)^{-1}g \in N. \tag{58}$$

Membership in N follows by applying π, and uniqueness follows because $\pi(g)$ determines q, which then determines n. The section need not be a homomorphism; in particular, we do not assume a split extension.

Use the abelian construction (i) as the first layer, with input $\pi(x_t)$. Its query code determines $q_t = \pi(g_t)$, where

$$q_t = \pi(x_t)q_{t-1}, \qquad q_{t-1} = \pi(x_t)^{-1}q_t. \tag{59}$$

Here q_t denotes a quotient-group element, distinct from a bold query vector $\mathbf{q}_t$. Define the effective input to the upper N-stack by

$$\eta_t = \eta(x_t, q_t), \qquad \eta(x, q) = s(q)^{-1}x\, s\big(\pi(x)^{-1}q\big)\,. \tag{60}$$

This lies in N because $\pi(\eta(x, q)) = q^{-1}\pi(x)(\pi(x)^{-1}q) = e_Q$. For the residual $n_t = s(q_t)^{-1}g_t$, equations 58–59 give

$$\begin{aligned} n_t &= s(q_t)^{-1}x_t g_{t-1} = s(q_t)^{-1}x_t s(q_{t-1})n_{t-1} \\ &= s(q_t)^{-1}x_t s\big(\pi(x_t)^{-1}q_t\big)\, n_{t-1} = \eta_t n_{t-1}. \end{aligned} \tag{61}$$

Since $n_0 = e_N$, the N-stack run on the effective inputs η_t therefore tracks n_t exactly. The final decoder reconstructs $g_t = s(q_t)n_t$.

This construction respects the specified layer inputs. The first layer is updated before the upper stack, and its current query output determines q_t. By equation 59, x_t and this current output also determine q_{t-1}, so equation 60 requires no delayed output or additional recurrent layer. Composing each parameter map in the N-stack with this correction gives a fixed function of the current original token and current lower-layer outputs. No layer consults its own previous memory to select its transition. All effective alphabets and code sets are finite, so these are permitted fixed pointwise maps; one-hot encoding of an effective token allows the same linear projections as in equation 52. Every recurrent layer retains its native readout $\mathbf{S}_t\mathbf{q}_t$, followed by the permitted pointwise decoding and interlayer computations.

Writing $E_Q(q)$ and $E_N(n)$ for the component memory encodings, the joint memory is

$$E_G(g) = \big(E_Q(q), E_N(n)\big), \qquad g = s(q)n. \tag{62}$$

The preceding algebra proves that input x sends it to $E_G(xg)$. The uniqueness in equation 58 makes E_G injective, with exactly $|Q||N| = |G|$ reachable memories when the alphabet is G. Each component retains its transition family and nonzero writes; only its effective input has changed. We have therefore constructed an $(L + 1)$-layer tracker of G from a one-layer abelian tracker of Q and an L-layer tracker of N.

We now induct on the derived length d. Recall $G^{(0)} = G$ and $G^{(j+1)} = [G^{(j)}, G^{(j)}]$, using the project's commutator convention $[a, b] = a^{-1}b^{-1}ab$. For $d = 0$, G is trivial and a constant output requires zero recurrent layers. For $d = 1$, G is abelian and (i) applies. If $d \geq 2$, put $N = G' = [G, G]$ and $Q = G/G'$. The subgroup G' is normal because conjugation sends commutators to

commutators, and Q is abelian because its commutators are trivial. Induction on j gives $N^{(j)} = G^{(j+1)}$, so $N^{(d-1)} = \{e\}$ and $N^{(d-2)} = G^{(d-1)} \neq \{e\}$. Thus N has derived length $d-1$. By the induction hypothesis it has a tracker with at most $d-1$ layers. The extension construction adds the abelian quotient layer and gives at most $1 + (d-1) = d$ layers for G, with the canonical finite memory set and decoder just proved.

Every recurrent layer in this induction is an abelian layer from (i), so its transition belongs to the required family and its write is nonzero. If a common key dimension is required, choose the largest layer dimension and append zero memory columns, zero key and query coordinates, and identity rotary pairs to the smaller layers. This preserves their updates, outputs, and membership in $\mathcal{F}_n((-1;1])$, without increasing the number of recurrent layers. This proves (ii).

E Proof of Theorem 2

Let $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ be a deterministic finite-state automaton with $|Q| = N$. We construct a network that returns $q_t = \delta(q_{t-1}, x_t)$ after every input x_t, starting from q_0. As in the project, layer parameters may be fixed functions of the current token and current lower-layer outputs. We use exact real arithmetic, fixed initial memories, and the allowed gate ranges $\alpha_t \in [0,1]$ and $\beta_t \in [0,2]$. Each layer has one matrix memory and its native query output $\mathbf{S}_t\mathbf{q}_t$, followed by a fixed pointwise decoder. In particular, no additional heads or processing tokens are required. If $N = 1$, the constant output q_0 is realized by the trivial-group layer in the proof of Theorem 1, so assume $N \geq 2$.

Use the quantitative Krohn–Rhodes decomposition in Grazzi et al. (2025, Appendix C.4, Theorem 5), based on the cascade decomposition of Maler & Pnueli (1994). It provides $s \leq 2^N$ component automata with state sets Q_i, $n_i = |Q_i| \leq N$, initial states $a_{i,0}$, and a fixed reconstruction map $W : Q_1 \times \cdots \times Q_s \to Q$. Their updates can be written

$$\begin{aligned} f_{i,t} &= f_i(x_t, a_{1,t-1}, \ldots, a_{i-1,t-1}), \\ a_{i,t} &= f_{i,t}(a_{i,t-1}), \qquad\qquad q_t = W(a_{1,t}, \ldots, a_{s,t}). \end{aligned} \tag{63}$$

For every possible effective input, $f_{i,t} : Q_i \to Q_i$ is either a permutation or a constant map, called a reset. The reconstruction holds also at $t = 0$. These are the precise properties of the decomposition used below. In particular, the input to component i involves the *previous* states of the earlier components.

We will simulate each component using two recurrent layers, after one shared clock layer. The two-layer simulator exposes both $a_{i,t}$ and $a_{i,t-1}$ through its current decoded output. This will make equation 63 implementable with the project's current-output layer convention. The construction adapts the delayed block simulation of Siems et al. (2026, Appendices B.2–B.3, Lemma 4); we give the details because the shared clock and the recovery of the previous state are needed for the claimed layer count.

Put $m = N - 1 \geq 1$. A single layer from Theorem 1(i), applied to $\mathbb{Z}_{2m}$ with increment one on every input, outputs an injective code of $t \bmod 2m$. Its fixed decoder supplies

$$j_t = 1 + ((t-1) \bmod 2m), \qquad p_t = 1 + ((t-1) \bmod m), \qquad t \geq 1. \tag{64}$$

These are functions of the current clock code. The same clock output is available to all later layers, so this costs one recurrent layer in total.

Fix a component and temporarily omit the index i. Identify its states with $\{1, \ldots, n\}$, where $n \leq N$, and let $\mathcal{P}$ be the finite set of all permutations and resets of this set. It contains the identity and is closed under composition: a product of permutations is a permutation, while a product containing a reset is a reset. Choose a bijection $c : \mathcal{P} \to \{1, \ldots, A\}$ with $c(\mathrm{id}) = 1$, and put $b = A + 1$.

The first of this component's two layers stores the most recent $2m$ maps f_t in a circular buffer. Its row memory is $\mathbf{S}_t^{\mathrm{buf}} \in \mathbb{R}^{1 \times 2m}$, initialized to $(1, \ldots, 1)$. Take

$$\alpha_t = \beta_t = 1, \quad \mathbf{R}_t = \mathbf{I}_{2m}, \quad \mathbf{k}_t = \mathbf{e}_{j_t}, \quad \mathbf{v}_t = c(f_t), \quad \mathbf{q}_t = (1, b, \ldots, b^{2m-1})^\top. \tag{65}$$

Here and below a scalar value is a vector in $\mathbb{R}^1$. Substitution into equation 3 gives

$$\mathbf{S}_t^{\mathrm{buf}} = \mathbf{S}_{t-1}^{\mathrm{buf}}(\mathbf{I} - \mathbf{e}_{j_t}\mathbf{e}_{j_t}^\top) + c(f_t)\mathbf{e}_{j_t}^\top. \tag{66}$$

Thus exactly slot j_t is overwritten, and every other slot is retained. Its query output is the integer with these $2m$ slot values as base-b digits. Since all digits lie in $\{1, \dots, A\} \subset \{0, \dots, b-1\}$, this readout is injective on all possible buffers. A fixed finite decoder therefore recovers every slot. The clock specifies their cyclic order, so it recovers

$$f_{t-2m+1}, \dots, f_t, \qquad \text{where } f_u = \mathrm{id} \text{ for } u \le 0. \tag{67}$$

The identity entries for nonpositive indices are supplied by the initial memory, not by extra inputs. The buffer has a finite reachable set, and its additive write is nonzero at every actual time step because $c(f_t) > 0$.

Every permutation of n states is a product of at most $n - 1 \le m$ transpositions. To see the bound, decompose the permutation into disjoint cycles: a cycle of length ℓ is a product of $\ell - 1$ transpositions, and the sum of these lengths minus one is n minus the number of cycles. A reset already consists of one elementary operation. Consequently, for each $F \in \mathcal{P}$ fix a factorization

$$F = \tau_m(F) \circ \cdots \circ \tau_1(F), \tag{68}$$

where every factor is a transposition, an identity, or a reset. Pad with identities as necessary; choose all identity factors when $F = \mathrm{id}$. These are fixed tables on a finite set.

Each such elementary operation can be implemented by one update of a single Rotary Gated DeltaNet layer, with a nonzero additive write. Choose an even $d_k \ge n+1$, put $\mathbf{u} = (1, 2, \dots, d_k)^\top$, and encode a state a by the row memory

$$H(a) = (\mathbf{u} + \mathbf{e}_a)^\top \in \mathbb{R}^{1 \times d_k}. \tag{69}$$

The last coordinate is auxiliary, since $a \le n < d_k$. Set $\mathbf{R}_t = \mathbf{I}_{d_k}$ throughout this layer and choose

$$\mathbf{q}_t = (1, 2, \dots, n, 0, \dots, 0)^\top. \tag{70}$$

Then $H(a)\mathbf{q}_t = \mathbf{u}^\top \mathbf{q}_t + a$, so a fixed affine decoder recovers a from the native query output.

For a transposition interchanging r and r', take $\alpha = 1$, $\beta = 2$, $\mathbf{k} = (\mathbf{e}_r - \mathbf{e}_{r'})/\sqrt{2}$, and scalar value $\mathbf{v} = \mathbf{u}^\top \mathbf{k}$. The matrix $P = \mathbf{I} - 2\mathbf{k}\mathbf{k}^\top$ interchanges these two coordinates and fixes the others. Hence

$$H(a)P + 2(\mathbf{u}^\top \mathbf{k})\mathbf{k}^\top = \mathbf{u}^\top + \mathbf{e}_a^\top P = H(\tau(a)). \tag{71}$$

The write is nonzero because $u_r \ne u_{r'}$. For the identity operation, take $\alpha = \beta = 1$, $\mathbf{k} = \mathbf{e}_{d_k}$, and $\mathbf{v} = u_{d_k}$. This preserves $H(a)$, since its last coordinate is u_{d_k} and the update erases and rewrites exactly that coordinate. For a reset to r, let $\mathbf{w}_r = \mathbf{u} + \mathbf{e}_r$ and take

$$\alpha = 0, \qquad \beta = 1, \qquad \mathbf{k} = \mathbf{w}_r / \|\mathbf{w}_r\|_2, \qquad \mathbf{v} = \|\mathbf{w}_r\|_2. \tag{72}$$

The new memory is $\mathbf{v}\mathbf{k}^\top = H(r)$, independently of the old memory. All these keys are unit and all writes are nonzero. The unscaled transition factors belong to $\mathcal{F}_{d_k}([-1; 1])$; the reset additionally uses the allowed global gate $\alpha = 0$.

Delayed computation with exact current and previous outputs. It remains to specify which elementary operation this second layer applies at each time. Write $t = \ell m + p$, with $\ell \ge 0$ and $1 \le p \le m$. This partitions time into blocks of length m. Define the map of block ℓ by

$$F_\ell = f_{(\ell+1)m} \circ \cdots \circ f_{\ell m+1}, \qquad F_{-1} = \mathrm{id}. \tag{73}$$

During block ℓ, the second layer applies, in order, the m fixed factors of the *preceding* block map $F_{\ell-1}$. More precisely, initialize its memory at $H(a_0)$ and let its encoded state r_t evolve by

$$r_0 = a_0, \qquad r_t = \tau_p(F_{\ell-1})(r_{t-1}) \quad \text{when } t = \ell m + p. \tag{74}$$

The elementary updates above implement this recurrence exactly. Their parameters are computable from the current buffer output and clock: the buffer contains the preceding complete block and the current partial block because their combined length is $m + p \le 2m$. In particular, $F_{\ell-1}$ and p are known. The unbounded block index ℓ appears only in the proof; the actual parameter map uses only the finite clock phase and the finite buffer.

Let $a_t = f_t(a_{t-1})$ denote the true component state, and set $a_u = a_0$ for $u \le 0$. We claim that, at $t = \ell m + p$,

$$r_t = \big(\tau_p(F_{\ell-1}) \circ \cdots \circ \tau_1(F_{\ell-1})\big)(a_{(\ell-1)m}). \tag{75}$$

For $\ell = 0$, every factor of F_{-1} is the identity, so the claim holds. Within any subsequent block, it follows by induction on p from equation 74. At the end of block ℓ, it gives $r_{(\ell+1)m} = F_{\ell-1}(a_{(\ell-1)m}) = a_{\ell m}$, which is exactly the starting state required for the next block. This proves the claim for all times by induction over the blocks.

Apply the remaining factors pointwise to the current core state:

$$b_t = \big(\tau_m(F_{\ell-1}) \circ \cdots \circ \tau_{p+1}(F_{\ell-1})\big)(r_t) = a_{\ell m}. \tag{76}$$

The product is the identity when $p = m$. The decoder can now recover *both* states needed by the cascade:

$$\begin{aligned} a_t &= (f_t \circ \cdots \circ f_{\ell m+1})(b_t), \\ a_{t-1} &= (f_{t-1} \circ \cdots \circ f_{\ell m+1})(b_t). \end{aligned} \tag{77}$$

The second product is empty when $p = 1$, in which case it returns $b_t = a_{t-1}$. All factors in these formulas are already in the current buffer, and r_t is recovered from the current core query. Thus these are fixed pointwise functions of current outputs. They require neither inverting a reset nor reading any layer's previous memory. At initialization all buffer entries are identities and $r_0 = a_0$, so the same decoder returns a_0 as both outputs. In particular the empty input is handled correctly.

This simulator has two recurrent layers: one buffer and one core. Its buffers have at most A^{2m} possible memories, and its core has at most n possible memories. Their decoder may be large, but its domain is finite and its size does not depend on the sequence length. It is a permitted fixed pointwise map under the project's definition. If desired, all parameter tables can be expressed as linear projections after a finite one-hot encoding of the available input codes. No recurrent depth is charged for these pointwise maps, in accordance with that definition.

Compose the simulators and count the layers. Place the shared clock first, followed by the buffer and core for component 1, then those for component 2, and so on. At time t, the current decoded output of each earlier simulator supplies its previous state $a_{j,t-1}$ by equation 77. Consequently the buffer for component i can compute and store exactly the map $f_{i,t}$ in equation 63. Its core then updates, and its decoder returns $a_{i,t}$ and $a_{i,t-1}$. Induction on the component index, together with the simulator invariant, proves that all decoded component states agree with the cascade at every time. The final pointwise decoder therefore returns $W(a_{1,t}, \ldots, a_{s,t}) = q_t$.

The clock has $2m$ reachable memories, and every buffer and core has a finite reachable set. Their finite Cartesian product contains the joint reachable memory set of the entire network. Thus the construction is an exact finite-state realization. It uses

$$L = 1 + 2s \le 1 + 2 \cdot 2^N = 2^{N+1} + 1 \tag{78}$$

recurrent layers. The only rotary counter is shared by all components; every other rotation is the identity, so all layers have the required Rotary Gated DeltaNet form. Finally, for any accepting set $R \subseteq Q$, compose the state decoder with $q \mapsto \mathbf{1}\{q \in R\}$. This accepts exactly the language recognized by $\mathcal{A}$, including the correct decision on the empty word. Since every regular language has such a finite automaton, the theorem follows.