Toward More Expressive yet Scalable RNNs: DeltaNet and Its Variants

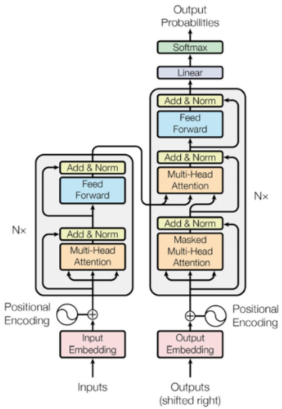
Songlin Yang

July 15, 2025

MIT CSAIL

Transformer

Attention Is All You Need



Softmax attention

Attention:

$$\text{Parallel training}: \quad \mathbf{O} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M})\mathbf{V} \quad \in \mathbb{R}^{L\times d}$$

$$\text{Iterative inference}: \quad \mathbf{o_t} = \sum_{j=1}^{t} \frac{\exp(\mathbf{q}_t^\top \mathbf{k}_j)}{\sum_{l=1}^{t} \exp(\mathbf{q}_t^\top \mathbf{k}_l)} \mathbf{v}_j \quad \in \mathbb{R}^{d}$$

where $\mathbf{M} \in \mathbb{R}^{L\times L}$ is the casual mask:

$$\mathbf{M}_{i,j} = \begin{cases} -\infty & \text{if } j > i \\ 1 & \text{if } j \leq i \end{cases}$$

- Training: **quadratic** time complexity
- Inference: **linear** space complexity with **KV cache**.

Linear attention = standard attention - softmax

Linear attention (Katharopoulos et al. 2020):

$$\text{Parallel training}: \quad \mathbf{O} = \cancel{\text{softmax}}(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M})\mathbf{V} \quad \in \mathbb{R}^{L \times d}$$

$$\text{Iterative inference}: \quad \mathbf{o_t} = \sum_{j=1}^{t} \frac{\cancel{\exp}(\mathbf{q}_t^\top \mathbf{k}_j)}{\cancel{\sum_{l=1}^{t} \exp(\mathbf{q}_t^\top \mathbf{k}_l)}} \mathbf{v}_j \quad \in \mathbb{R}^{d}$$

where **M** is the causal mask for linear attention:

$$\mathbf{M}_{i,j} = \begin{cases} 0 & \text{if } j > i \\ 1 & \text{if } j \leq i \end{cases}$$

Equivalent View: Matrix-Valued Hidden States

$$\mathbf{o_t} = \sum_{j=1}^{t} (\mathbf{q}_t^\top \mathbf{k}_j)\mathbf{v}_j$$

$$= \sum_{j=1}^{t} \mathbf{v}_j(\mathbf{k}_j^\top \mathbf{q}_t) \quad \mathbf{k}_j^\top \mathbf{q}_t = \mathbf{q}_t^\top \mathbf{k}_j \in \mathbb{R}$$

$$= \underbrace{(\sum_{j=1}^{t} \mathbf{v}_j \mathbf{k}_j^\top)}_{\mathbf{S}_t \in \mathbb{R}^{d \times d}} \mathbf{q}_t \qquad \text{By associativity}$$

Linear attention = Linear RNN + matrix-valued hidden states

Let $\mathbf{S}_t = \sum_{j=1}^{t} \mathbf{v}_j \mathbf{k}_j^\top \in \mathbb{R}^{d \times d}$ be the matrix-valued hidden state, then:

$$\begin{aligned} \mathbf{S}_t &= \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top && \in \mathbb{R}^{d \times d} \\ \mathbf{o}_t &= \mathbf{S}_t \mathbf{q}_t && \in \mathbb{R}^{d} \end{aligned}$$

- Linear attention implements **elementwise linear recurrence**, allowing for efficient inference.

Linear attention = Linear RNN + matrix-valued hidden states

Let $\mathbf{S}_t = \sum_{j=1}^{t} \mathbf{v}_j \mathbf{k}_j^\top \in \mathbb{R}^{d \times d}$ be the matrix-valued hidden state, then:

$$\begin{aligned} \mathbf{S}_t &= \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top && \in \mathbb{R}^{d \times d} \\ \mathbf{o}_t &= \mathbf{S}_t \mathbf{q}_t && \in \mathbb{R}^{d} \end{aligned}$$

- Linear attention has a **matrix-valued hidden state**, significantly increasing the state size (and thereby real-world task performance).

Hardware-efficient training of linear attention

- The **outer-product structure** allows **hardware-efficient** state expansion by leveraging **matrix multiplication**, which is highly optimized with **tensor cores** in modern GPUs.

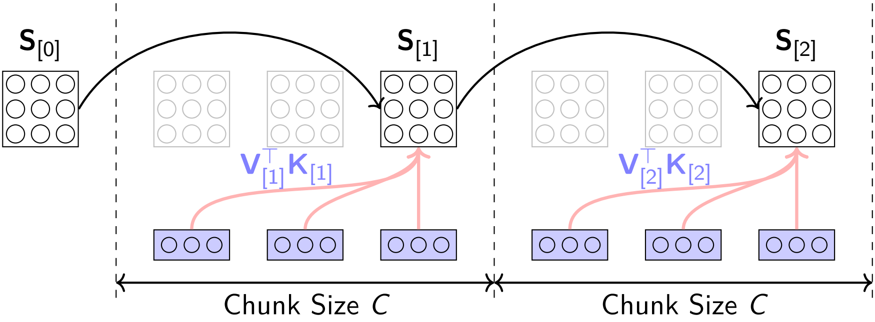
$$\sum_{i=1}^{L} \mathbf{v}_i \mathbf{k}_i^\top = \mathbf{V}^\top \mathbf{K}$$

where $\mathbf{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_L]^\top \in \mathbb{R}^{L \times d}, \mathbf{K} = [\mathbf{k}_1, \cdots, \mathbf{k}_L]^\top \in \mathbb{R}^{L \times d}$.

Autoregressive Modeling Tip

For autoregressive modeling, we can checkpoint some intermediate states, enabling efficient computation of outputs at any position. $\rightarrow$ **Chunkwise parallel form**

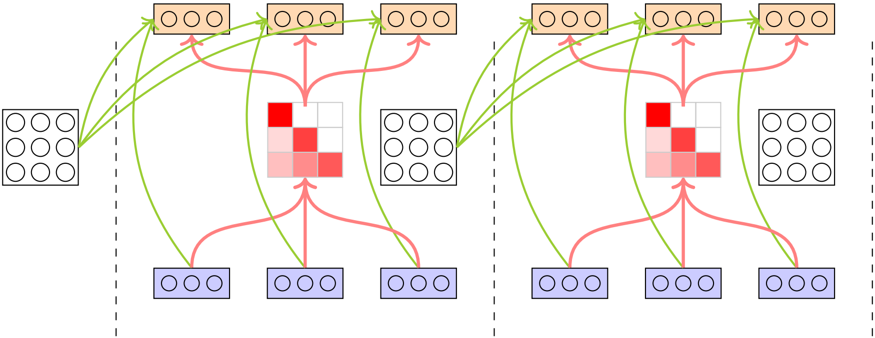
Sequential Chunk-Level State Passing:



$$\mathbf{S}_{[t+1]} = \underbrace{\mathbf{S}_{[t]}}_{\mathbb{R}^{d\times d}} + \underbrace{\mathbf{V}_{[t]}^\top}_{\mathbb{R}^{d\times C}} \underbrace{\mathbf{K}_{[t]}}_{\mathbb{R}^{C\times d}} \qquad \in \mathbb{R}^{d\times d}$$

Computational Complexity: $\mathcal{O}(Cd^2)$ per chunk and $\mathcal{O}(Ld^2)$ for the entire sequence.

Parallel Output Computation:



$$\mathbf{O}_{[t]} = \underbrace{\mathbf{Q}_{[t]}\mathbf{S}_{[t]}^{\top}}_{\text{inter-chunk:}\mathbf{O}_{[t]}^{\text{inter}}} + \underbrace{(\mathbf{Q}_{[t]}\mathbf{K}_{[t]}^{\top} \odot \mathbf{M})\mathbf{V}_{[t]}}_{\text{intra-chunk:}\mathbf{O}_{[t]}^{\text{intra}}} \in \mathbb{R}^{C \times d}$$

Computational Complexity: $\mathcal{O}(C^2 d + Cd^2)$ per chunk.

$\mathcal{O}(Ld^2 + LCd)$ for the entire sequence.

Key limitations of linear attention

However, linear attention has **fundamental limitations** in **in-context retrieval**

Computer Science > Machine Learning

[Submitted on 28 Feb 2024 (v1), last revised 6 Dec 2024 (this version, v4)]

RNNs are not Transformers (Yet): The Key Bottleneck on In-context Retrieval

Kaiyue Wen, Xingyu Dang, Kaifeng Lyu

or **in-context copy**:

Repeat After Me:
Transformers are Better than State Space Models at Copying
Transformers are Better than State Space Models at Copying

Samy Jelassi [1] David Brandfonbrener [2] Sham M. Kakade [2 3] Eran Malach [2]

Linear Attention: Associative Memory View

Key Idea: Linear attention builds a key-value memory via outer products:

$$\mathbf{S} = \sum_i \mathbf{v}_i \mathbf{k}_i^\top$$

To retrieve $\mathbf{v}_j$, we compute:

$$\mathbf{S}\mathbf{k}_j = \sum_i \mathbf{v}_i (\mathbf{k}_i^\top \mathbf{k}_j)$$

This includes the **desired $\mathbf{v}_j$** and **unwanted** cross-terms:

$$= \mathbf{v}_j + \underbrace{\sum_{i \neq j} (\mathbf{k}_i^\top \mathbf{k}_j) \mathbf{v}_i}_{\text{retrieval error}}$$

(assuming all $\mathbf{k}_i$ are l2-normalized)

Goal: Minimize retrieval error

Fundamental Limitation: Orthogonality

To eliminate retrieval error:

$$\mathbf{k}_i^\top \mathbf{k}_j = 0 \quad \text{for all } i \neq j$$

But: In $\mathbb{R}^d$, there are at most d orthogonal vectors!

Implication:

- Limited capacity for distinct key-value pairs
- Explains why increasing head dimensions helps (more room in space)

Retrieval Overload in Practice

In practice:

- Vanilla linear attention underperforms softmax
- Can't erase previous associations (no "forgetting")
- Accumulated interference $\rightarrow$ degraded performance on long sequences

> *"The enemy of memory is not time; it's other memories."*
> *— David Eagleman*

DeltaNet: Linear attention with delta rule

DeltaNet: Key-Value Memory Update

DeltaNet (Schlag, Irie, and Schmidhuber 2021) uses an intuitive memory update mechanism:

$\mathbf{q}_t = \mathbf{W}_Q \mathbf{x}_t$	**Query vector is computed**
$\mathbf{k}_t = \mathbf{W}_K \mathbf{x}_t$	**Key vector is computed**
$\mathbf{v}_t = \mathbf{W}_V \mathbf{x}_t$	**Value vector is computed**
$\beta_t = \text{sigmoid}(\mathbf{W}_\beta \mathbf{x}_t)$	**Beta scalar value is computed**
$\mathbf{v}_t^{\text{old}} = \mathbf{S}_{t-1} \mathbf{k}_t$	**Old value is retrieved using current key**
$\mathbf{v}_t^{\text{new}} = \beta_t \mathbf{v}_t + (1 - \beta_t) \mathbf{v}_t^{\text{old}}$	**New value combines current and old values**
$\mathbf{S}_t = \mathbf{S}_{t-1} - \underbrace{\mathbf{v}_t^{\text{old}} \mathbf{k}_t^\top}_{\text{remove old}} + \underbrace{\mathbf{v}_t^{\text{new}} \mathbf{k}_t^\top}_{\text{write new}}$	**State matrix is updated**
$\mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t$	**Output is retrieved from memory using query**

Compared to vanilla linear attention, DeltaNet can not only *write* new values to memory, but also *remove* old values from memory.

In-context associative recall on MQAR

Multi-Query Associative Recall (MQAR, Arora et al. 2023)

A synthetic benchmark for testing in-context associative recall. **Example:**

- Given key-value pairs: “A 4 B 3 C 6 F 1 E 2”
- Query: “A ? C ? F ? E ? B ?”
- Expected output: “4, 6, 1, 2, 3”

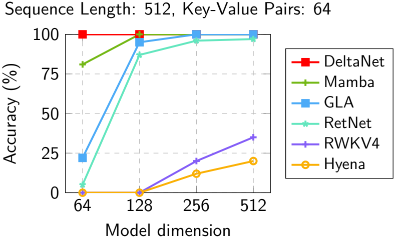


Figure 1: Accuracy (%) on MQAR. DeltaNet achieves the perfect recall.

DeltaNet: Chunkwise Parallel Training

$$
\begin{aligned}
\mathbf{S}_t &= \mathbf{S}_{t-1} + (\mathbf{v}_t^{\text{new}} - \mathbf{v}_t^{\text{old}})\mathbf{k}_t^\top \\
&= \mathbf{S}_{t-1} + \underbrace{\beta_t(\mathbf{v}_t - \mathbf{S}_{t-1}\mathbf{k}_t)}_{\text{defined as } \mathbf{u}_t}\mathbf{k}_t^\top \\
&= \mathbf{S}_{t-1} + \mathbf{u}_t\mathbf{k}_t^\top \\
&= \sum_{i=1}^{t} \mathbf{u}_i\mathbf{k}_i^\top
\end{aligned}
$$

Once "pseudo-values" $\mathbf{u}_t$ are computed, DeltaNet can be trained using the same kernel as linear attention.

DeltaNet: Chunkwise Parallel Training

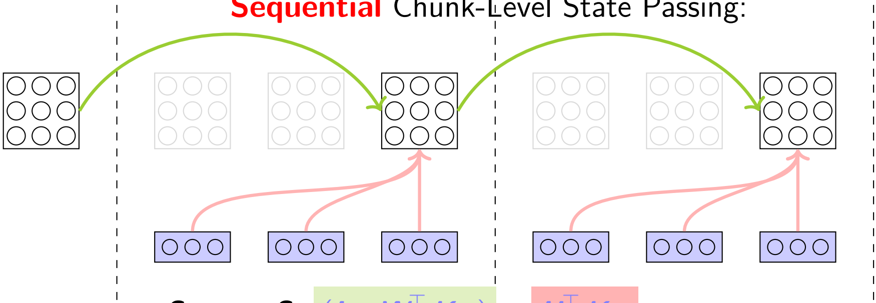
$$\begin{aligned}
\mathbf{S}_t &= \mathbf{S}_{t-1} + \beta_t(\mathbf{v}_t - \mathbf{S}_{t-1}\mathbf{k}_t)\mathbf{k}_t^\top \\
&= \mathbf{S}_{t-1}\left(\mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top\right) + \beta_t\mathbf{v}_t\mathbf{k}_t^\top \\
&= \sum_{i=1}^{t}\left(\beta_i\mathbf{v}_i\mathbf{k}_i^\top \underbrace{\prod_{j=i+1}^{t}\left(\mathbf{I} - \beta_j\mathbf{k}_j\mathbf{k}_j^\top\right)}_{\mathbf{P}_j^t}\right)
\end{aligned}$$

Using the WY representation (Bischof and Loan 1985):

$$\mathbf{P}_1^t = \mathbf{I} - \sum_{i=1}^{t}\mathbf{w}_i\mathbf{k}_i^\top.$$

Key Insights:: The cumulative product $\prod$ becomes a cumulative sum $\sum$, enabling efficient matrix-multiply-based training.

Sequential Chunk-Level State Passing:



$$\boldsymbol{S}_{[t+1]} = \boldsymbol{S}_{[t]} \left(\boldsymbol{I} - \boldsymbol{W}_{[t]}^{\top}\boldsymbol{K}_{[t]}\right) + \boldsymbol{U}_{[t]}^{\top}\boldsymbol{K}_{[t]}$$

Using hardware-friendly UT transform (Joffrain et al. 2006):

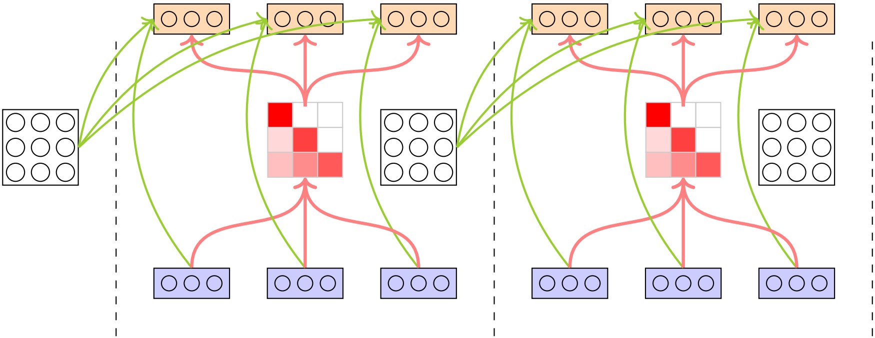
$$\mathbf{T}_{[t]} = \left(\mathbf{I} + \mathrm{tril}(\mathrm{diag}(\beta_{[t]})\mathbf{K}_{[t]}\mathbf{K}_{[t]}^{\mathsf{T}}, -1)\right)^{-1} \mathrm{diag}\left(\beta_{[t]}\right) \qquad \in \mathbb{R}^{C\times C}$$

⋆ Lower triangular matrix inversion can be computed efficiently

$$\mathbf{W}_{[t]} = \mathbf{T}_{[t]}\mathbf{K}_{[t]}, \quad \mathbf{U}_{[t]} = \mathbf{T}_{[t]}\mathbf{V}_{[t]} \qquad \in \mathbb{R}^{C\times d}$$

See https://sustcsonglin.github.io/blog/2024/deltanet-2/ for details.

Parallel Output Computation:



$$\mathbf{O}_{[t]} = \mathbf{Q}_{[t]}\mathbf{S}_{[t]}^{\top} + \left(\mathbf{Q}_{[t]}\mathbf{K}_{[t]}^{\top} \odot \mathbf{M}\right)\left(\mathbf{U}_{[t]} - \mathbf{W}_{[t]}\mathbf{S}_{[t]}^{\top}\right)$$

Compared to vanilla linear attention, the "pseudo-values" need to be adjusted by the historical context: $\mathbf{W}_{[t]}\mathbf{S}_{[t]}^{\top}$.

Speed Comparison

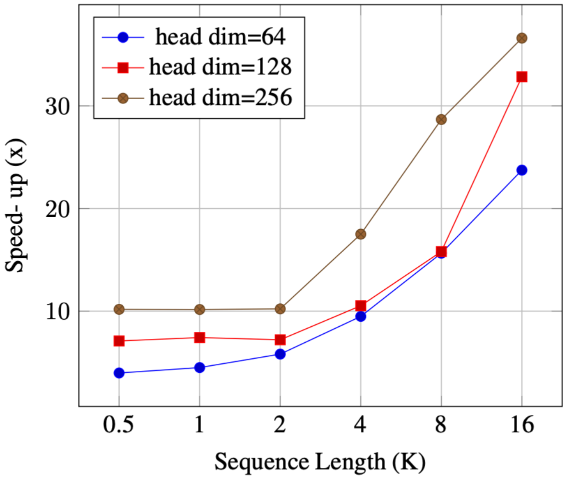
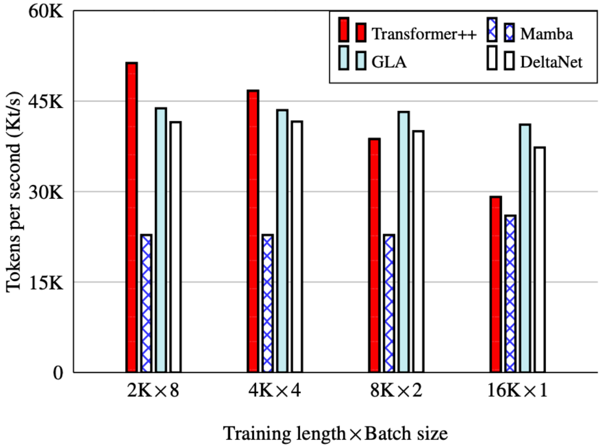


Figure 2: Chunkwise parallel form provides significant speedup over recurrent form.

Speed Comparison



DeltaNet with forget gates

DeltaNet updates only a single key-value association pair at each time step.

$$\Downarrow$$

This results in slow forgetting speed, requiring d steps to erase the entire memory.

DeltaNet with forget gates

Computer Science > Neural and Evolutionary Computing

[Submitted on 13 Apr 2018 (v1), last revised 13 Sep 2018 (this version, v3)]

The unreasonable effectiveness of the forget gate

Jos van der Westhuizen, Joan Lasenby

Computer Science > Machine Learning

[Submitted on 8 Jun 2017 (v1), last revised 25 Oct 2017 (this version, v3)]

Gated Orthogonal Recurrent Units: On Learning to Forget

Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljačić, Yoshua Bengio

Computer Science > Machine Learning

[Submitted on 11 Dec 2023 (v1), last revised 27 Aug 2024 (this version, v6)]

Gated Linear Attention Transformers with Hardware-Efficient Training

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, Yoon Kim

A key lesson we've learned from the linear attention and broader RNN literature is that forget gates (a.k.a. data-dependent decay) are unreasonably effective!

DeltaNet with forget gates

Gated linear attention (Yang et al. 2023) formulation:

$$\mathbf{S}_t = \mathbf{S}_{t-1} \odot \mathbf{G}_t + \mathbf{v}_t \mathbf{k}_t^\top \in \mathbb{R}^{d \times d}$$

where $\mathbf{G}_t \in \mathbb{R}^{d \times d}$ can be defined in various ways:

- GLA/RWKV6/HGRN2: $\mathbf{G}_t = \mathbf{1}_t \boldsymbol{\alpha}_t^\top$
- Decaying Fast weight: $\mathbf{G}_t = \boldsymbol{\beta}_t \boldsymbol{\alpha}_t^\top$
- Mamba1: $\mathbf{G}_t = \exp(-(\Delta_t \mathbf{1}^\top) \odot \exp(A))$
- Mamba2: $\mathbf{G}_t = \gamma_t \mathbf{1}\mathbf{1}^\top$

See Table 1 of GLA (Yang et al. 2023) for a comprehensive summary.

Gated DeltaNet

Gated DeltaNet (Yang, Kautz, and Hatamizadeh 2024) uses a Mamba2-style scalar-valued forget gate $\alpha_t \in [0, 1]$:

$$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top \qquad \text{Mamba2}$$

$$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top) + \beta_t \mathbf{v}_t \mathbf{k}_t^\top \qquad \text{Gated DeltaNet}$$

Model	ppl ↓	LM-eval ↑	Recall ↑	Long ↑
Mamba1	17.92	53.12	21.0	14.6
Mamba2	16.56	54.89	29.8	13.5
DeltaNet	17.72	52.14	26.2	13.6
Gated DeltaNet	**16.42**	**55.32**	**30.6**	**16.6**
Mamba+SWA	16.13	54.00	37.3	15.9
Gated DeltaNet+SWA	**16.07**	**56.41**	**40.1**	**17.8**

Table 1: Performance comparison of 1.3B models trained on 100B tokens. Source: Yang, Kautz, and Hatamizadeh 2024.

RWKV-7

RWKV-7 (Peng et al. 2025) uses a GLA-style vector-valued forget gate $\alpha_t \in [0, 1]^d$:

$$\mathbf{S}_t = \mathbf{S}_{t-1}\mathrm{diag}(\alpha_t) + \mathbf{v}_t\mathbf{k}_t^\top \qquad \text{GLA/RWKV-6}$$

$$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathrm{diag}(\alpha_t) + \mathbf{a}_t\mathbf{b}_t^\top) + \mathbf{v}_t\mathbf{k}_t^\top \qquad \text{RWKV-7}$$

D Expressivity of RWKV-7

We show that the RWKV-7 architecture can express NC^1-complete state tracking problems that cannot be expressed by transformers or other recurrent architectures such as S4 and Mamba, under standard complexity conjectures. We first show a particular NC^1-complete problem that can be expressed by RWKV-7 in Section D and then generalize the argument to show that any regular language can be recognized by RWKV-7 in Section D.2. As regular language recognition can be understood to formalize finite state tracking problems, this suggests an expressivity advantage of RWKV-7 on state-tracking problems.

RWKV-7 can solve problems that are NC^1-complete under AC^0 reductions (as can DeltaNet and Gated DeltaNet), demonstrating their enhanced computational power.

DeltaNet's expressivity

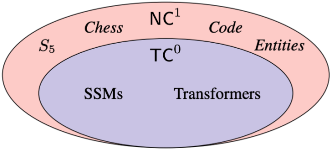


Figure 3: Source: Merrill, Petty, and Sabharwal 2024

- TC^0: Constant-depth parallel networks with threshold gates and massive fan-in
  - Transformers
  - Linear RNNs with diagonal transition matrices (e.g., Mamba, Gated Linear Attention)
- NC^1: Logarithmic-depth networks with limited fan-in, capable of more complex tasks
  - Nonlinear RNNs
  - Linear RNNs with data-dependent nondiagonal transition matrices

DeltaNet's expressivity

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

DeltaNet uses Generalized Householder (GH) transition matrices, which are both data-dependent and nondiagonal, making it possible to achieve expressiveness beyond TC^0.

DeltaNet's expressivity

$$\mathbf{S}_t = \mathbf{S}_{t-1} \underbrace{(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top)}_{\text{GH transition}} + \beta_t \mathbf{v}_t \mathbf{k}_t^\top = \sum_{i=1}^{t} \left(\beta_i \mathbf{v}_i \mathbf{k}_i^t \underbrace{\prod_{j=i+1}^{t} (\mathbf{I} - \beta_j \mathbf{k}_j \mathbf{k}_j^\top)}_{\text{cumulative GH products}} \right)$$

Key Properties:

- **Expressiveness**: When allowing negative eigenvalues in GH matrices (Grazzi et al. 2024), the cumulative products of GH matrices can represent *any* matrix with Euclidean norm < 1.
- **Complexity Class**: Cumulative products of general matrices cannot be computed in TC^0 (Mereghetti and Palano 2000).
- **Conclusion**: DeltaNet with negative eigenvalues has expressiveness beyond TC^0, strictly exceeding SSMs and Transformers.

DeltaNet's expressivity

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

Figure 4: Synthetic tasks performance comparison (source: Grazzi et al. 2024). $[0, 1]$ and $[-1, 1]$ denotes the ranges of eigenvalues for each model's transition matrix.

- Allowing negative eigenvalues could boost performance for both Mamba and DeltaNet.
- DeltaNet achieves superior performance due to its richer expressiveness.

DeltaNet as Test-Time Training

Sequence Modeling as Test-Time Regression (Wang, Shi, and Fox 2025)

Summary

A unified framework for sequence model design

Parametric regression (first order optimizer)	Parametric regression (second order optimizer)	Nonparametric regression
Batch gradient descent Linear attention, Mamba, GLA, HGRN, Gateloop, RWKV, RetNet, mLSTM, LRU **Batch gradient descent with nonlinear feature maps** Performer, cosFormer, RFA, Hedgehog, Based, Rebased, DiJiang **Stochastic gradient descent** DeltaNet, TTT, DeltaProduct Longhorn (adaptive step size) Gated DeltaNet (L2 regularization) Titans (momentum)	**Newton's method** *Mesa-layer*	**Kernel regression** *Intention* **Local polynomial estimation** *Self-attention* & *higher order generalizations*

All of these sequence layers construct and query an **associative memory via test-time regression** in their forward pass

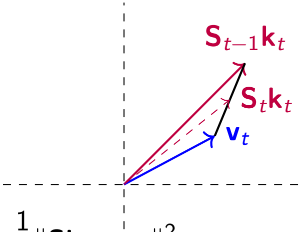
Parametric associative memory usually has an efficient **recurrent update**, at the cost of forgetting the past

We are here

Source: *Test-Time Regression (Slide Content)*

DeltaNet: test-time objective

Directly minimize Euclidean distance



Objective: $\mathcal{L}_t(\mathbf{S}) = \frac{1}{2}\|\mathbf{S}\mathbf{k}_t - \mathbf{v}_t\|^2$

SGD update: $\mathbf{S}_t = \mathbf{S}_{t-1} - \beta_t \nabla \mathcal{L}_t(\mathbf{S}_{t-1}) = \mathbf{S}_{t-1} - \beta_t(\mathbf{S}_{t-1}\mathbf{k}_t - \mathbf{v}_t)\mathbf{k}_t^\top$

Key Insight: hidden state as a proxy for KV cache

DeltaNet encodes key-value associations directly in the hidden state matrix $\mathbf{S}_t$ as a neural memory, enabling efficient in-context learning and retrieval without an explicit KV cache.

DeltaProduct

Generalizing the DeltaNet by performing multiple gradient descent steps (i.e., n_h) per token:

$$\begin{aligned}\mathbf{S}_{t,j} &= \mathbf{S}_{t,j-1} - \beta_{t,j}\nabla\mathcal{L}_{t,j}(\mathbf{S}_{t,j-1}) \\ &= \left(\mathbf{I} - \beta_{t,j}\mathbf{k}_{t,j}\mathbf{k}_{t,j}^\top\right)\mathbf{S}_{t,j-1} + \beta_{t,j}\mathbf{v}_{t,j}\mathbf{k}_{t,j}^\top\end{aligned}$$

where $\mathbf{S}_{t,0} = \mathbf{S}_{t-1}$ and $\mathbf{S}_{t,n_h} = \mathbf{S}_t$. This results in a **high-rank** recurrent updates

$$\begin{aligned}\mathbf{S}_t &= \mathbf{S}_{t-1}\mathbf{A}_t + \mathbf{B}_t \\ \mathbf{A}_t &= \prod_{j=1}^{n_h}\left(\mathbf{I} - \beta_{t,j}\mathbf{k}_{t,j}\mathbf{k}_{t,j}^\top\right) \\ \mathbf{B}_t &= \sum_{j=1}^{n_h}\beta_{t,j}\mathbf{v}_{t,j}\mathbf{k}_{t,j}^\top\prod_{l=1}^{j-1}\left(\mathbf{I} - \beta_{t,l}\mathbf{k}_{t,l}\mathbf{k}_{t,l}^\top\right)\end{aligned}$$

where both the transition matrix $\mathbf{A}_t$ and the input $\mathbf{B}_t$ are rank-n_h.

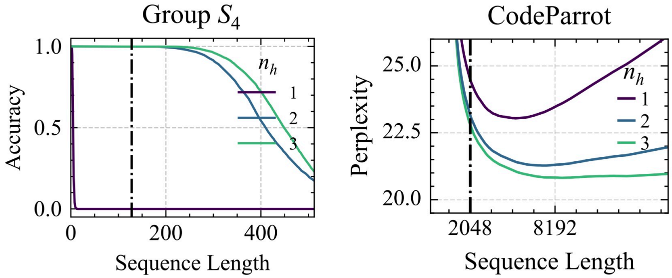


Figure 5: (*Left*) DeltaProduct$_{n_h}$ learns higher-order permutation groups like S_4 in one layer, while DeltaNet ($n_h = 1$) is limited to S_2. (*Right*) Length extrapolation of DeltaProduct improves significantly with higher n_h.

TTT layer

TTT (Sun et al. 2024) used a nonlinear regression objective loss:

$$\mathcal{L}_t(\mathbf{S}) = \frac{1}{2}\|f_{\mathbf{S}}(\mathbf{k}_t) - \mathbf{v}_t\|^2$$

where $f_{\mathbf{S}}$ is a nonlinear transformation parameterized by $\mathbf{S}$.

Examples:

- TTT-linear:

$$f_{\mathbf{S}}(x) = \mathrm{LN}(\mathbf{S}x) + x,$$

- TTT-MLP:

$$f_{\mathbf{S}}(x) = \mathrm{LN}(\mathrm{MLP}_{\mathbf{S}}(x)) + x$$

where LN denotes layer normalization.

TTT layer

The nonlinear loss induces a nonlinear recurrence, posing challenges for parallelization.

Solution: Mini-batch Gradient Descent

- Minibatch size aligns with chunk size.
- Each token within a chunk is treated as an independent training example for parallel processing.
- Sequential dependencies are preserved via a lightweight linear recurrence within chunks.

This approach essentially combines intra-chunk linear recurrence with inter-chunk nonlinear recurrence.

Titans

TTT (Sun et al. 2024) extends this to a nonlinear regression loss:

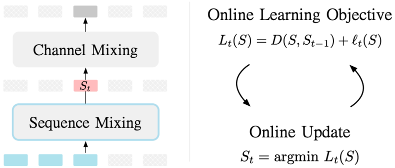
$$\mathcal{L}_t(\mathbf{S}) = \frac{1}{2}\|f_{\mathbf{S}}(\mathbf{k}_t) - \mathbf{v}_t\|^2$$

where $f_{\mathbf{S}}$ is a nonlinear transformation parameterized by $\mathbf{S}$.

- Titans (Behrouz, Zhong, and Mirrokni 2024) further enhances TTT by integrating momentum and weight decay into the mini-batch SGD update.

Instead of performing (multiple) gradient descent to optimize the objective, can we get a closed-form solution?

LongHorn



Longhorn (Liu et al. 2024) optimizes the following objective:

$$\mathcal{L}_t(\mathbf{S}) = \underbrace{\|\mathbf{S} - \mathbf{S}_{t-1}\|_F^2}_{D(\mathbf{S},\mathbf{S}_{t-1})} \underbrace{-\beta_t \|\mathbf{S}\mathbf{k}_t - \mathbf{v}_t\|^2}_{l_t(S)}$$

with a closed-form solution:

$$\mathbf{S}_t = \mathbf{S}_{t-1}\left(\mathbf{I} - \epsilon_t \mathbf{k}_t \mathbf{k}_t^\top\right) + \epsilon_t \mathbf{v}_t \mathbf{k}_t^\top, \quad \epsilon_t = \frac{\beta_t}{1 + \beta_t \mathbf{k}_t^\top \mathbf{k}_t}$$

Key difference: DeltaNet's β_t does not depend on $\mathbf{k}_t$, while Longhorn's ϵ_t depends on $\mathbf{k}_t$.

Mesa layer

DeltaNet/Longhorn only considers the prediction error of the current token, while Mesa layer (Oswald et al. 2024) considers the prediction error of all historical tokens:

$$\mathcal{L}_t(\mathbf{S}) = \underbrace{\|\mathbf{S} - \mathbf{S}_{t-1}\|_F^2}_{D(\mathbf{S},\mathbf{S}_{t-1})} + \underbrace{\sum_{i=1}^{t} -\beta_i \|\mathbf{S}\mathbf{k}_i - \mathbf{v}_i\|^2}_{l_t(S)}$$

with a closed-form solution:

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \beta_t \mathbf{P}_t \mathbf{k}_t \left(\mathbf{S}_{t-1}\mathbf{k}_t - \mathbf{v}_t\right)^\top$$

$$\mathbf{P}_t = \mathbf{P}_{t-1} - \frac{\beta_t \mathbf{P}_{t-1}\mathbf{k}_t\mathbf{k}_t^\top \mathbf{P}_{t-1}}{1 + \beta_t \mathbf{k}_t^\top \mathbf{P}_{t-1}\mathbf{k}_t}$$

where $\mathbf{P}_t$ is updated recursively using the Matrix Inversion Lemma:

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^\top)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\top\mathbf{A}^{-1}}{1 + \mathbf{v}^\top\mathbf{A}^{-1}\mathbf{u}}$$

DeltaNet/Longhorn vs. Mesa Layer

DeltaNet/Longhorn vs. Mesa Layer

- **DeltaNet/Longhorn**: Like Least Mean Square (LMS)
  - Only considers current prediction error
  - Simple and computationally efficient
  - May require more iterations to converge
- **Mesa Layer**: Like Recursive Least Squares (RLS)
  - Considers all historical prediction errors
  - Optimal in terms of minimizing cumulative error
  - Faster convergence but higher computational cost

Thanks!

References

Arora, Simran et al. (2023). **"Zoology: Measuring and Improving Recall in Efficient Language Models".** In: *CoRR* abs/2312.04927.

Behrouz, Ali, Peilin Zhong, and Vahab Mirrokni (2024). ***Titans: Learning to Memorize at Test Time.*** arXiv: 2501.00663 [cs.LG]. URL: https://arxiv.org/abs/2501.00663.

Bischof, Christian H. and Charles Van Loan (1985). **"The WY representation for products of householder matrices".** In: *SIAM Conference on Parallel Processing for Scientific Computing*. URL: https://api.semanticscholar.org/CorpusID:36094006.

- Grazzi, Riccardo et al. (2024). **"Unlocking State-Tracking in Linear RNNs Through Negative Eigenvalues".** In: URL: https://api.semanticscholar.org/CorpusID:274141450.
- Joffrain, Thierry et al. (2006). **"Accumulating Householder transformations, revisited".** In: *ACM Trans. Math. Softw.* 32, pp. 169–179. URL: https://api.semanticscholar.org/CorpusID:15723171.
- Katharopoulos, Angelos et al. (2020). **"Transformers are rnns: Fast autoregressive transformers with linear attention".** In: *International conference on machine learning*. PMLR, pp. 5156–5165.
- Liu, Bo et al. (2024). **"Longhorn: State Space Models are Amortized Online Learners".** In: *ArXiv* abs/2407.14207. URL: https://api.semanticscholar.org/CorpusID:271310065.

- Mereghetti, Carlo and Beatrice Palano (2000). **“Threshold circuits for iterated matrix product and powering”.** In: *RAIRO Theor. Informatics Appl.* 34, pp. 39–46. URL: https://api.semanticscholar.org/CorpusID:13237763.
- Merrill, William, Jackson Petty, and Ashish Sabharwal (2024). **“The Illusion of State in State-Space Models”.** In: *ArXiv* abs/2404.08819. URL: https://api.semanticscholar.org/CorpusID:269149086.
- Oswald, Johannes von et al. (2024). ***Uncovering mesa-optimization algorithms in Transformers*.** arXiv: 2309.05858 [cs.LG]. URL: https://arxiv.org/abs/2309.05858.

- Peng, Bo et al. (2025). ***RWKV-7 "Goose" with Expressive Dynamic State Evolution.*** arXiv: 2503.14456 [cs.CL]. URL: https://arxiv.org/abs/2503.14456.
- Schlag, Imanol, Kazuki Irie, and Jürgen Schmidhuber (2021). **"Linear Transformers Are Secretly Fast Weight Programmers".** In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 9355–9366.
- Sun, Yu et al. (2024). **"Learning to (Learn at Test Time): RNNs with Expressive Hidden States".** In: *ArXiv* abs/2407.04620. URL: https://api.semanticscholar.org/CorpusID:271039606.

Wang, Ke Alexander, Jiaxin Shi, and Emily B. Fox (2025). ***Test-time regression: a unifying framework for designing sequence models with associative memory.*** arXiv: 2501.12352 [cs.LG]. URL: https://arxiv.org/abs/2501.12352.

Yang, Songlin, Jan Kautz, and Ali Hatamizadeh (2024). ***Gated Delta Networks: Improving Mamba2 with Delta Rule.*** arXiv: 2412.06464 [cs.CL]. URL: https://arxiv.org/abs/2412.06464.

Yang, Songlin et al. (2023). **"Gated Linear Attention Transformers with Hardware-Efficient Training".** In: *CoRR* abs/2312.06635. DOI: 10.48550/ARXIV.2312.06635. arXiv: 2312.06635. URL: https://doi.org/10.48550/arXiv.2312.06635.