

Linear Transformers

Rainer Engelken

Advanced Topics in Theoretical Neuroscience, CTN 2024

February 26, 2025

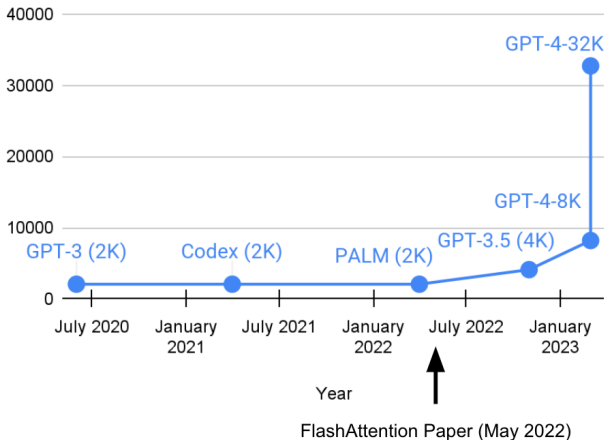
Introduction

Introduction

- ▶ **Transformers:** Powerful sequence models that have revolutionized many areas of AI. They rely on a mechanism called “attention” to weigh the importance of different parts of the input sequence.
- ▶ **The Problem:** Quadratic time and memory complexity in sequence length limits their scalability and biological plausibility.
- ▶ **Linear Attention:** A way to make Transformers more efficient and connect them to concepts familiar in theoretical neuroscience, like Recurrent Neural Networks (RNNs) and fast weights.
- ▶ **Outline:**
 - ▶ What is linear attention?
 - ▶ Connection to RNNs & Fast Weights
 - ▶ Efficient Training with Chunkwise Parallelism
 - ▶ Implications for Expressiveness & Biological Plausibility

Foundation Model's Context Length is growing rapidly

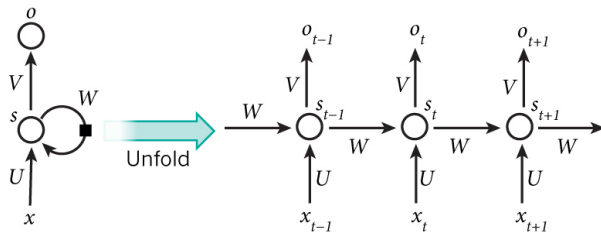
Foundation Model Context Length



Issues with Transformers

- ▶ Training: quadratic time complexity
 - ▶ Expensive for long sequence modeling (e.g., video or DNA modeling)
- ▶ Inference: linear memory complexity
 - ▶ Requires storing KV cache for each token
 - ▶ High memory burden.

Revisiting RNNs



- ▶ Training: linear complexity, however, traditional RNNs are not parallelizable.
- ▶ Inference: constant memory

Modern linear recurrent models

Use linear recurrence for parallel training

- ▶ Gated linear RNNs (HGRN, Griffin, ...)
- ▶ State-space models (S4, Mamba, ...)
- ▶ Linear attention (RetNet, GLA, xLSTM, DeltaNet, ...)

Modern linear recurrent models

Use linear recurrence for parallel training

- ▶ Gated linear RNNs (HGRN, Griffin, ...)
- ▶ State-space models (S4, Mamba, ...)
- ▶ **Linear attention (RetNet, GLA, xLSTM, DeltaNet, ...)**

Linear attention is the focus of this talk.

Linear attention

Softmax attention

Attention:

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

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

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

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

Linear attention = standard attention - softmax

Linear attention (Katharopoulos et al. 2020):

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

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

where \mathbf{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

$$\begin{aligned}\mathbf{o}_t &= \sum_{j=1}^t (\mathbf{q}_t^\top \mathbf{k}_j) \mathbf{v}_j \\ &= \sum_{j=1}^t \mathbf{v}_j (\mathbf{k}_j^\top \mathbf{q}_t) \quad \mathbf{k}_j^\top \mathbf{q}_t = \mathbf{q}_t^\top \mathbf{k}_j \in \mathbb{R} \\ &= \underbrace{\left(\sum_{j=1}^t \mathbf{v}_j \mathbf{k}_j^\top \right)}_{\mathbf{S}_t \in \mathbb{R}^{d \times d}} \mathbf{q}_t \quad \text{By associativity}\end{aligned}$$

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:

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

$$\mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t \quad \in \mathbb{R}^d$$

- ▶ Linear attention implements **elementwise linear recurrence**.
- ▶ Linear attention has a **matrix-valued hidden state**, significantly increasing the state size.

Challenges in training: the parallel form

$$\mathbf{O} = (\mathbf{Q}\mathbf{K}^{\top} \odot \mathbf{M})\mathbf{V} \in \mathbb{R}^{L \times d}$$

The time complexity is still quadratic in sequence length, which is problematic for long sequences.

Challenges in training: the recurrent form

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

$$\mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t \quad \in \mathbb{R}^d$$

Poor GPU utilization due to:

- ▶ Sequential computation limits parallelization opportunities across the sequence dimension.
- ▶ Rank-1 outer product updates and matrix-vector multiplications are not optimized for GPU tensor cores, which are designed for dense matrix-multiply operations (typically at least 16x16 matrix sizes).

Chunkwise parallel form

Chunkwise Form:

- ▶ Interpolates between recurrent and parallel forms.
- ▶ Splits a sequence of length L into L/C chunks of size C :
 - ▶ When $C = 1$, it reduces to the recurrent form.
 - ▶ When $C = L$, it reduces to the parallel form.
- ▶ **Key Property:** Chunkwise form is **NOT an approximation**—it computes the exact same output as the original formulation.

Chunkwise parallel form

Chunkwise form computes only the **last hidden state** per chunk.
Output is derived from:

- ▶ **Recurrent Form:** Historical context across chunks.
- ▶ **Parallel Form:** Local context within a chunk.

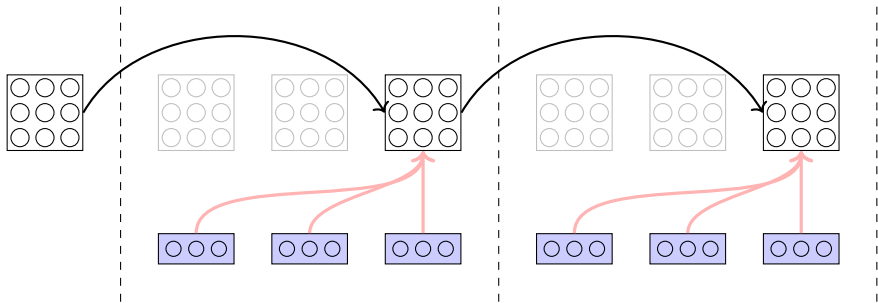
Notations

$\mathbf{S}_{[i]} := \mathbf{S}_{iC} \in \mathbb{R}^{d \times d}$ the last hidden state of chunk i ,

$\mathbf{Q}_{[i]} = \mathbf{Q}_{iC+1:(i+1)C} \in \mathbb{R}^{C \times d}$ the query block of chunk i .

We define $\mathbf{K}_{[i]}$, $\mathbf{V}_{[i]}$, $\mathbf{O}_{[i]}$ in a similar way.

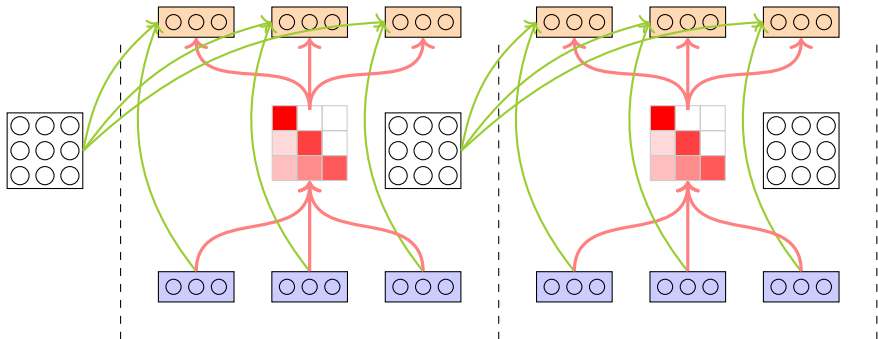
Sequential Chunk-Level State Passing:



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

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

Parallel Output Computation:



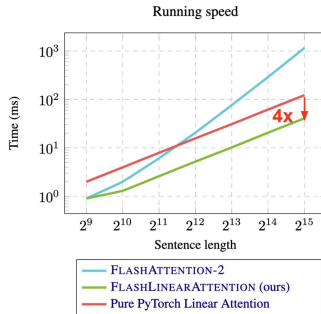
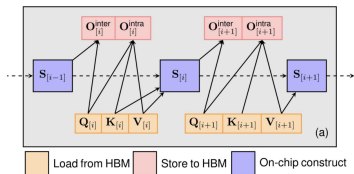
$$\mathbf{O}_{[t]} = \underbrace{\mathbf{Q}_{[t]} \mathbf{S}_{[t]}^{\top}}_{\text{inter-chunk: } \mathbf{O}_{[t]}^{\text{inter}}} + \underbrace{(\mathbf{Q}_{[t]} \mathbf{K}_{[t]}^{\top} \odot \mathbf{M}) \mathbf{V}_{[t]}}_{\text{intra-chunk: } \mathbf{O}_{[t]}^{\text{intra}}} \in \mathbb{R}^{C \times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

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

Chunkwise parallel form

- ▶ Total complexity: $\mathcal{O}(Ld^2 + LdC)$, achieving **subquadratic complexity** in sequence length when C is small.
- ▶ Practical settings: C is typically set to $\{64, 128, 256\}$.
- ▶ Extensibility: Can be generalized to linear attention with **decay or delta rule** (to be discussed later).
- ▶ Adoption: The **de facto standard** for **training modern linear attention models**, including:
 - ▶ Mamba2, Based, GLA, DeltaNet, Lightning Attention, mLSTM, and others.

Flash linear attention



I/O optimization significantly improves the wall-clock time.

Flash linear attention

flash-linear-attention

Public



Efficient implementations of state-of-the-art linear attention models in Pytorch and Triton



Python



1.6k



80

The Flash Linear Attention library provides hardware-efficient implementation of various linear attention models.

- RetNet, GLA, Based, HGRN2, RWKV6, GSA, Mamba2, DeltaNet, Gated DeltaNet, RWKV7 ...

Summary

- ▶ Linear attention = Softmax attention - softmax.
- ▶ Linear attention = Linear RNN + **matrix-valued hidden state**.
- ▶ The chunkwise parallel form is more **hardware-friendly** than the recurrent and parallel forms.
- ▶ Flash Linear Attention is an **I/O-aware** implementation of the chunkwise parallel form.

Linear attention with decay

Linear attention is not enough

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

Vanilla linear attention **performs poorly in language modeling**.

- ▶ Only stores key-value pairs in memory.
- ▶ Has no mechanism to forget old memories.
- ▶ Leads to memory saturation and degradation in performance since the memory size is fixed.

Linear attention with data-independent decay

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

- ▶ γ is a constant exponential decay factor $0 < \gamma < 1$.
- ▶ This mechanism **weighs recent tokens more than distant tokens**, and language modeling has a strong **recency bias**.
- ▶ Works well in practice: RetNet (Sun et al. 2023), Lightning Attention (Qin et al. 2024b).

Linear attention with data-dependent decay

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

- ▶ $\gamma_t \in (0, 1)$ is a data-dependent decay term that is a function of \mathbf{x}_t .
- ▶ Enables dynamic control of memory retention/forgetting based on input data.
- ▶ Examples: Mamba2 (Dao and Gu 2024), mLSTM (Beck et al. 2024), Gated Retention (Sun et al. 2024b).

The parallel form for linear attention with decay

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

Linear attention with decay has the following parallel form:

$$\begin{aligned}\mathbf{O} &= (\mathbf{QK}^\top \odot \mathbf{D})\mathbf{V} \in \mathbb{R}^{L \times d} \\ \mathbf{D}_{i,j} &= \begin{cases} \prod_{m=j+1}^i \gamma_m & \text{if } i > j \\ 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

The duality between recurrent and parallel forms is referred to as **state space duality (SSD)** in Mamba2 (Dao and Gu 2024).

Linear attention with more fine-grained decay

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

$$\mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t \quad \in \mathbb{R}^d$$

Condition for the matmul-based (chunkwise) parallel form (Yang et al. 2023):

$$\mathbf{G}_t = \boldsymbol{\beta}_t \boldsymbol{\alpha}_t^\top \in \mathbb{R}^{d \times d}, \quad \boldsymbol{\alpha}_t, \boldsymbol{\beta}_t \in \mathbb{R}^d$$

- ▶ Mamba-1 (Gu and Dao 2023) does not conform to this structure, preventing the use of tensor cores.
- ▶ It is common to set $\boldsymbol{\beta}_t = \mathbf{1}$ in practice, examples: GLA (Yang et al. 2023), RWKV6 (Peng et al. 2024), GSA (Zhang et al. 2024), HGRN2 (Qin et al. 2024a).

Summary

- ▶ Language modeling has a strong **recency bias**.
- ▶ Decay helps **bridge the perplexity gap** between linear and softmax attention.
- ▶ **Fine-grained decay** enhances performance but poses scaling challenges.
- ▶ **Outer-product structure** enables efficient chunkwise training with fine-grained decay.

Toward more expressive update rule:
the delta rule

Fast weight programming

Before diving into the delta rule, let's first review fast weight programming, which will help us understand the delta rule and other test-time-trainers (TTT, Titans, Mesa layer, etc.).

“Fast weights provide a neurally plausible way of implementing the type of temporary storage that is required by working memory, while slow weights capture more permanent associations learned over many experiences.” – Geoffrey Hinton

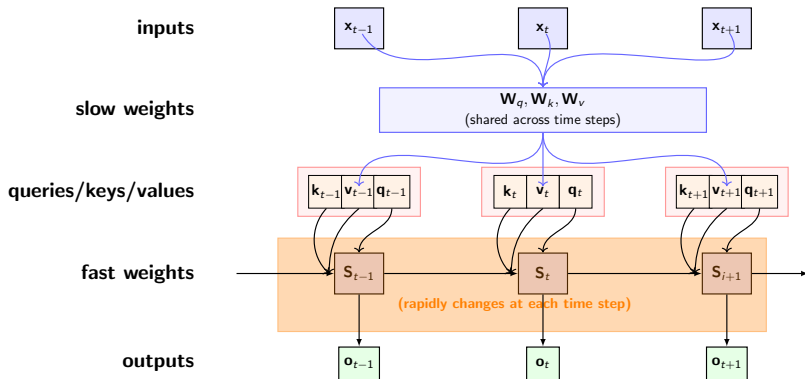
Recall the memory readout in linear attention:

$$\mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t$$

We can think of the recurrent hidden state \mathbf{S}_t as a fast weight matrix that maps input \mathbf{q}_t to output \mathbf{o}_t and is updated as it goes:

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

Linear attention is secretly a fast weight programmer



- **Fast Weight:** S_t maps q_t to o_t , updated dynamically during inference for rapid adaptation.
- **Slow Weight:** W_q, W_k , and W_v are fixed during inference and only updated during training (e.g., via gradient descent).

The choice of update rule



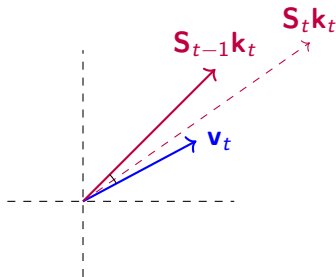
Figure: The principle of Hebbian learning.

- ▶ Hebbian update rule: $\mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top$
- ▶ Delta rule: $\mathbf{S}_t = \mathbf{S}_{t-1} - \beta_t (\mathbf{S}_{t-1} \mathbf{k}_t - \mathbf{v}_t) \mathbf{k}_t^\top$
- ▶ ...

Both Hebbian and delta update rules can be regarded as optimizing online learning objective via **test-time SGD**.

Linear Attention: Test-Time Objective

Maximize alignment
= Minimize angle difference + enlarge $|\mathbf{S}\mathbf{k}|$

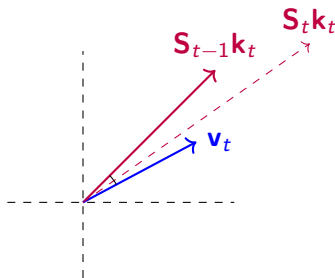


Objective: $\mathcal{L}_t(\mathbf{S}) = -\langle \mathbf{S}\mathbf{k}_t, \mathbf{v}_t \rangle$

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

Linear Attention: Test-Time Objective

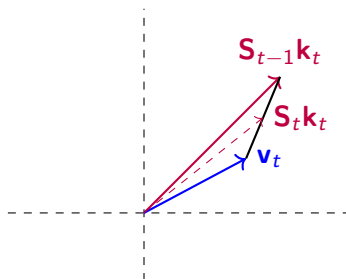
Maximize alignment
= Minimize angle difference + enlarge $|\mathbf{S}\mathbf{k}|$



- ▶ Linear attention may favor increasing $|\mathbf{S}\mathbf{k}|$ over angle alignment, leading to numerical instabilities.
- ▶ Mamba2 uses **decay** ($\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top$) to control $|\mathbf{S}\mathbf{k}|$, stabilizing the training process.

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

- ▶ **Better numerical stability:** the norm of \mathbf{S}_t is controlled.
- ▶ **Better in-context associative recall:** directly optimizes key-value association prediction (Liu et al. 2024)

In-context associative recall

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”

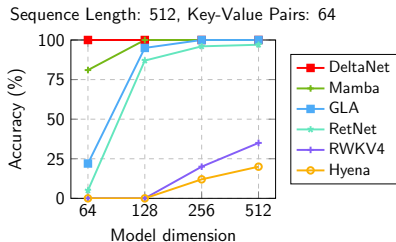
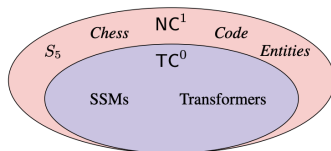


Figure: Accuracy (%) on MQAR. DeltaNet achieves the perfect recall.

Transformers and SSMs in TC^0



- ▶ Transformers are in TC^0 .
- ▶ Linear RNNs with **diagonal transition matrices** (e.g., $\mathbf{S}_t = \mathbf{S}_{t-1} \text{diag}(\alpha_t) + \mathbf{v}_t \mathbf{k}_t^\top$ in GLA) fall under TC^0 .
- ▶ **Nonlinear RNN** or linear RNN with **data-dependent nondiagonal transition matrices** could achieve expressiveness beyond TC^0 (Merrill, Petty, and Sabharwal 2024).

DeltaNet's expressiveness

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

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 expressiveness

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

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.

State tracking performance

	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: State tracking 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 state tracking performance for both Mamba and DeltaNet.
- ▶ DeltaNet achieves superior performance due to its **richer expressiveness**.

DeltaNet: Chunkwise Parallel Training

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

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

DeltaNet: Chunkwise Parallel Training

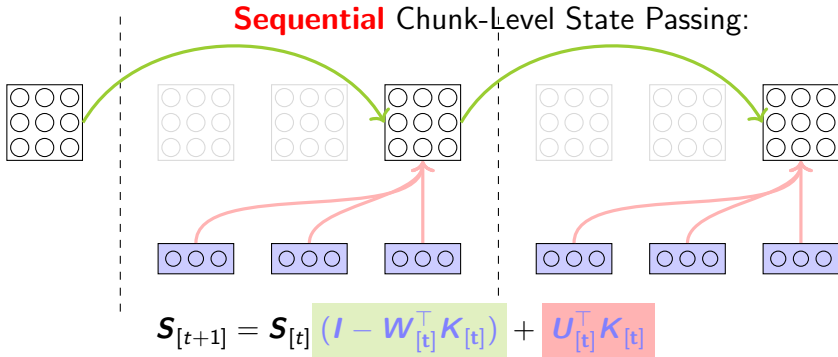
$$\begin{aligned}\mathbf{S}_t &= \mathbf{S}_{t-1} \left(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top \right) + \beta_t \mathbf{v}_t \mathbf{k}_t^\top \\ &= \sum_{i=1}^t \left(\beta_i \mathbf{v}_i \mathbf{k}_i^\top \underbrace{\prod_{j=i+1}^t (\mathbf{I} - \beta_j \mathbf{k}_j \mathbf{k}_j^\top)}_{\mathbf{P}_j^t}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

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

Sequential Chunk-Level State Passing:



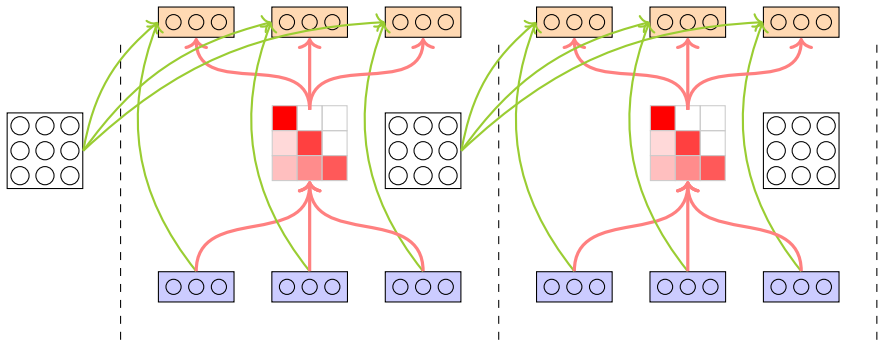
Using hardware-friendly UT transform (Joffrain et al. 2006):

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

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

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: $\mathbf{W}_{[t]} \mathbf{S}_{[t]}^{\top}$.

Gated DeltaNet

Enhancing DeltaNet with a Mamba2-like gating mechanism could boost performance on real-world tasks.

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

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

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

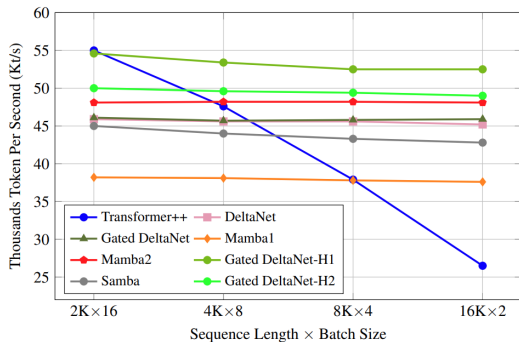


Figure: End-to-end training throughput comparison for different 1.3B models on a single H100. Source: Yang, Kautz, and Hatamizadeh 2024.

Combining DeltaNet and Mamba2's chunkwise algorithms for hardware-efficient training:

- Pure Gated DeltaNet is slightly slower than Mamba2.
- Hybridizing sliding window attention with Gated DeltaNet (i.e., Gated DeltaNet-H1) improves throughput.

DeltaNet's chunkwise algorithm can be extended to **diagonal**-plus-**low-rank** transition matrices:

$$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{D}_t + \boldsymbol{\alpha}_t \boldsymbol{\beta}_t^\top) + \mathbf{v}_t \mathbf{k}_t^\top, \quad \mathbf{D}_t \in \mathbb{R}^{d \times d}, \quad \boldsymbol{\alpha}_t, \boldsymbol{\beta}_t \in \mathbb{R}^d$$

- ▶ **RWKV-7** employs this linear recurrence, showing effectiveness.
- ▶ A fast implementation is available in the flash-linear-attention library
(<https://github.com/fla-org/flash-linear-attention/blob/main/fla/ops/rwkv7/chunk.py>).



Songlin Yang
@SonglinYang4



RWKV7 is now supported in the flash-linear-attention library, offering 30x faster pre-filling speed!



OpenMOSE @_m0se_ · Jan 19

RWKV-Infer Updated.

1. Test support Flash-Linear-Attention on RWKV v7

Pre-fill速度が、従来から30倍高速化しました🔥...

[Show more](#)

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 matrices.

DeltaProduct

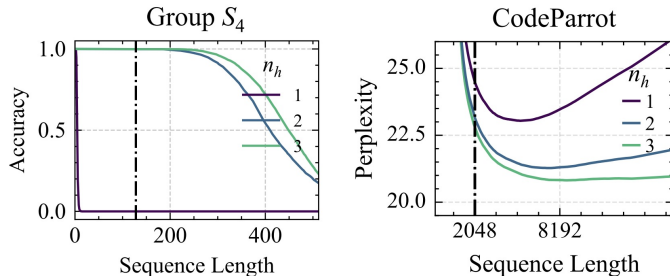


Figure: This figure is copied from Siems et al. 2025. (Left) $\text{DeltaProduct}_{n_h}$ learns higher-order permutation groups like S_4 in one layer, while DeltaNet ($n_h = 1$) is limited to S_2 . (Right) Length extrapolation of DeltaProduct improves significantly with higher n_h .

Summary

- ▶ Linear attention and DeltaNet are fast weight programmers with different test-time SGD weight updates.
- ▶ DeltaNet has strictly more expressive power than Mamba/GLA while maintaining efficient parallelization
- ▶ DeltaNet's chunkwise algorithm could be generalized to diagonal-plus-low-rank transition matrices
- ▶ Gating and high-rank updates further enhance DeltaNet's performance

Beyond linear regression objective

Beyond linear regression objective

DeltaNet optimizes the online linear regression loss:

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

- ▶ This optimization objective assumes linear relationships in historical data dependencies
- ▶ However, generative AI tasks involve complex, nonlinear dependencies
- ▶ A linear regression loss may be insufficient to capture these rich patterns.

Going beyond online linear regression objective

TTT (Sun et al. 2024a) 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} .

Examples:

► TTT-linear:

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

► TTT-MLP:

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

where LN denotes layer normalization.

Beyond Linear Regression Objective

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**.

Beyond linear regression objective

TTT (Sun et al. 2024a) 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.

Summary





- ▶ **Modern RNNs through the lens of online learning:**
 - ▶ (Decaying/Gated) Linear attention: Negative inner-product loss.
 - ▶ (Gated) DeltaNet: Linear regression loss.
 - ▶ TTT & Titans: Nonlinear regression losses.
- ▶ **Gradient-based optimization techniques prove valuable:**
 - ▶ Weight decay enables effective forgetting (e.g., Mamba2, Gated DeltaNet).
 - ▶ Momentum improves performance (e.g., Titans).
- ▶ **Efficient hardware utilization via:**
 - ▶ Chunkwise training for linear attention.
 - ▶ Hybrid linear/nonlinear approaches across chunks (e.g., TTT & Titans).
- ▶ **Promising future:** Bridging optimization and RNN architectures.

Thanks!





Slides adapted from Songlin Yang

(github.com/sustcsonglin/linear-attention-and-beyond-slides)




References I

-  Arora, Simran et al. (2023). “Zoology: Measuring and Improving Recall in Efficient Language Models”. In: *CoRR* abs/2312.04927.
-  Beck, Maximilian et al. (2024). “xLSTM: Extended Long Short-Term Memory”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. URL: <https://openreview.net/forum?id=ARAxPPIAhq>.
-  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>.






References II

-  Dao, Tri and Albert Gu (2024). “Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality”. In: *Forty-first International Conference on Machine Learning*. URL:
<https://openreview.net/forum?id=ztn8FCR1td>.
-  Grazi, Riccardo et al. (2024). “Unlocking State-Tracking in Linear RNNs Through Negative Eigenvalues”. In: URL:
<https://api.semanticscholar.org/CorpusID:274141450>.
-  Gu, Albert and Tri Dao (2023). “Mamba: Linear-Time Sequence Modeling with Selective State Spaces”. In.
-  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>.





References III

-  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>.

References IV

-  MiniMax et al. (2025). *MiniMax-01: Scaling Foundation Models with Lightning Attention*. arXiv: 2501.08313 [cs.CL]. URL: <https://arxiv.org/abs/2501.08313>.
-  Peng, Bo et al. (2024). “Eagle and Finch: RWKV with Matrix-Valued States and Dynamic Recurrence”. In.
-  Qin, Zhen et al. (2024a). “HGRN2: Gated Linear RNNs with State Expansion”. In: URL: <https://api.semanticscholar.org/CorpusID:269043328>.
-  Qin, Zhen et al. (2024b). “Various Lengths, Constant Speed: Efficient Language Modeling with Lightning Attention”. In: *ArXiv abs/2405.17381*. URL: <https://api.semanticscholar.org/CorpusID:270063820>.
-  Siems, Julien et al. (2025). *DeltaProduct: Increasing the Expressivity of DeltaNet Through Products of Householders*. arXiv: 2502.10297 [cs.LG]. URL: <https://arxiv.org/abs/2502.10297>.

References V

-  Sun, Yu et al. (2024a). “Learning to (Learn at Test Time): RNNs with Expressive Hidden States”. In: *ArXiv* abs/2407.04620. URL: <https://api.semanticscholar.org/CorpusID:271039606>.
-  Sun, Yutao et al. (2023). “Retentive network: A successor to transformer for large language models”. In: *arXiv preprint arXiv:2307.08621*.
-  Sun, Yutao et al. (2024b). “You Only Cache Once: Decoder-Decoder Architectures for Language Models”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. URL: <https://openreview.net/forum?id=25Ioxw576r>.
-  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>.

References VI



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>.



Zhang, Yu et al. (2024). “Gated Slot Attention for Efficient Linear-Time Sequence Modeling”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. URL: <https://openreview.net/forum?id=jY4PhQibmg>.