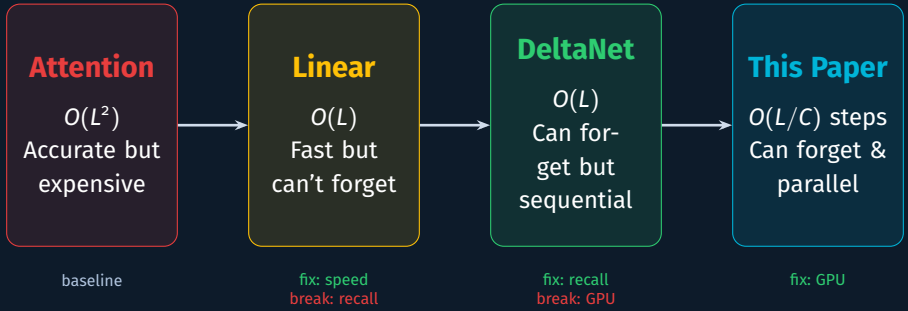
Parallelizing Linear Transformers with the Delta Rule over Sequence Length

Yang, Wang, Zhang, Shen, Kim – **NeurIPS 2024**
MIT · Soochow University · MIT-IBM Watson AI Lab

Presented by **Youngseo Cho, Yuanting Fan**
CS 8803-LLM · Georgia Tech

Why This Paper?



Why This Paper?

The Quadratic Wall

Transformer attention: $O(L^2 d)$

$L = 1\text{K} \rightarrow 1\text{M}$ ops

$L = 10\text{K} \rightarrow 100\text{M}$ ops

$L = 100\text{K} \rightarrow 10\text{B}$ ops

Plus: KV cache $\propto L$ memory

The Dream

✓ $O(L)$ computation

✓ Constant memory (no KV cache)

× But retrieval accuracy suffers…

$\Rightarrow$ This paper:

Linear cost + strong retrieval

+ GPU-friendly training

Linear Attention: The Associativity Trick

Key Insight

Just reorder the matrix multiplication (associativity). The $L \times L$ matrix is **never formed**. When $d{=}128,\ L{=}10{,}000 \Rightarrow$ **massive savings**.

Linear Attention = RNN with a Chalkboard

Recurrent Form

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top \quad \text{(write)}$$

$$\mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t \quad \text{(read)}$$

$\mathbf{S} \in \mathbb{R}^{d \times d}$ = fixed-size memory

✓ $O(1)$ per token at inference

✓ No KV cache needed

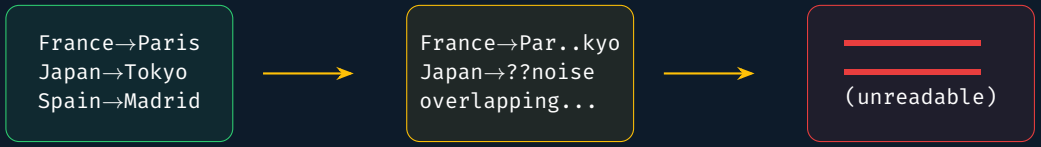
Chalkboard ($d \times d$)

```
France → Paris
Japan  → Tokyo
Korea  → Seoul ←new
```

Query “Korea” → “Seoul” ✓

The Fatal Flaw: No Eraser

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top \quad \longleftarrow \text{ only } +, \text{ never } -$$

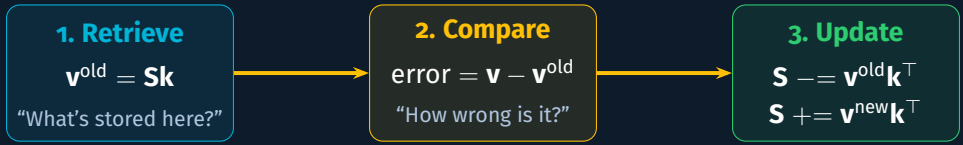


Root Cause

$d \times d$ board with $L \gg d$ entries. **Can't erase** → info piles up → retrieval fails. This is why Linear Attention loses on **in-context retrieval**.

Delta Rule: “Check, Erase, Rewrite”

Schlag et al. 2021 — Widrow–Hoff (1960) learning rule applied to Linear Attention



Combined Equation

$$\mathbf{S}_t = \mathbf{S}_{t-1}\underbrace{(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top)}_{\text{erase}} + \underbrace{\beta_t \mathbf{v}_t \mathbf{k}_t^\top}_{\text{write}} \qquad \beta_t = \sigma(\mathbf{W}_\beta \mathbf{x}_t) \in (0, 1)$$

Why Delta Rule Works: The Gradient View

Linear Attention

Loss: $\mathcal{L} = -\langle \mathbf{Sk}, \mathbf{v} \rangle$

$\nabla =$ constant

Same update whether memory is wrong or right

Delta Rule

Loss: $\mathcal{L} = \frac{1}{2}\|\mathbf{Sk} - \mathbf{v}\|^2$

$\nabla \propto$ error

Big mistake $\rightarrow$ big correction

Small mistake $\rightarrow$ small correction

Delta Rule = 1-step SGD on retrieval error

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

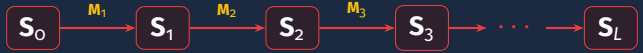
The more wrong the memory, the harder it fixes it. DeltaNet: 100% In-Context Recall (vs 80–94% for others)

The Problem: Delta Rule Can't Parallelize

Linear Attention: addition → parallelize!

$\mathbf{S}_L = \mathbf{v}_1\mathbf{k}_1^\top + \mathbf{v}_2\mathbf{k}_2^\top + \cdots + \mathbf{v}_L\mathbf{k}_L^\top \quad \Rightarrow \quad$ compute all at once

DeltaNet: must check the board before writing → sequential!



Each $\mathbf{M}_t = \mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top$ is **different** at every step (non-commutative)

10,000 tokens = 10,000 sequential steps. GPU sits idle.

$\Rightarrow$ This is the problem this paper solves.

Key Observation: $\mathbf{M}_t$ is a "Surgical Eraser"

The Transition Matrix

$$\mathbf{M}_t = \mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top \qquad \mathbf{M}_t\mathbf{x} = \mathbf{x} - \beta_t(\mathbf{k}_t^\top \mathbf{x})\,\mathbf{k}_t$$

What does this do to a vector?

$\mathbf{k}_t$ direction:	shrink by $(1-\beta_t)$
	$\beta=1$: completely erase
	$\beta=0$: untouched
Other $d-1$ dirs:	perfectly preserved
	eigenvalue $= 1$

"Surgical" forgetting — only the target direction is affected.

Why This Matters

It's a $d \times d$ matrix, but defined by:

- 1 vector ($\mathbf{k}$, d-dim)
- 1 scalar (β)

= **Generalized Householder Transformation** (since 1958)

$\Rightarrow$ Exploit its low-rank structure!

The WY Trick: No $d \times d$ Matrix Needed

Naive Approach

Materialize $\mathbf{S}_t$ ($d \times d$) every step

Memory: $O(d^2)$ per step

$d=128 \Rightarrow 16{,}384$ numbers/step

This Paper's Insight

$\mathbf{S}_t = \sum_{i=1}^{t} \mathbf{u}_i \mathbf{k}_i^\top$

Sum of rank-1 matrices!

Memory: $O(d)$ per step

How to compute $\mathbf{u}_t$ without ever forming $\mathbf{S}_{t-1}$

$$\mathbf{u}_t = \beta_t \left(\mathbf{v}_t - \sum_{i=1}^{t-1} \mathbf{u}_i \underbrace{(\mathbf{k}_i^\top \mathbf{k}_t)}_{\text{scalar!}} \right)$$

Only needs dot products between $\mathbf{k}$ vectors — no $d \times d$ matrix ever formed!

"WY Representation" (Bischof & Van Loan, 1987) for Householder products.

Chunkwise Parallelism: Best of Both Worlds

WITHIN chunk (parallel ✓)

C tokens via **matrix multiply**

$\mathbf{T} = (\mathbf{I} + \text{tril}(\beta\mathbf{K}\mathbf{K}^\top))^{-1}\text{diag}(\beta)$

$\mathbf{W} = \mathbf{T}\mathbf{K}, \quad \mathbf{U} = \mathbf{T}\mathbf{V}$

GPU tensor cores fully utilized

BETWEEN chunks (sequential, but few)

Pass state $\mathbf{S}$ forward:

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

Only L/C steps:

$10{,}000/64 = 156$ steps

(vs 10,000 before)

Complexity & Speed Results

Form	FLOPs	Seq. Steps	Memory	GPU?
Recurrent	$O(Ld^2)$	$O(L)$	$O(d^2)$	No
Fully Parallel	$O(L^2d)$	$O(1)$	$O(L^2)$	Yes
Chunkwise	$\boldsymbol{O(LCd)}$	$\boldsymbol{O(L/C)}$	$\boldsymbol{O(Ld)}$	**Yes**

Speed-up vs Recurrent (H100)

d=64, L=2K $\sim 5\times$

d=64, L=8K $\sim 12\times$

d=128, L=4K $\sim 13\times$

d=256, L=4K $\sim 30\times$+

Three Algorithmic Contributions

1. Householder structure of $\mathbf{M}_t$
   $\Rightarrow$ rank-1, compact storage
2. WY representation
   $\Rightarrow O(d^2) \rightarrow O(d)$ memory
3. Chunkwise parallelism
   $\Rightarrow O(L) \rightarrow O(L/C)$ steps + matmul

DeltaNet Transformer Architecture

Overall Structure

Follow **LLaMA / Transformer++** layout

- ✓ Replace self-attention with DeltaNet layer
- ✓ Keep RMSNorm, SwiGLU FFN
- ✓ Same parameter budget as Transformer++

DeltaNet layer	$4d^2$ params
SwiGLU FFN layer	$8d^2$ params

Short Convolution (borrowed from Mamba)

Lightweight depthwise convolution after Q/K/V projections

- ✓ Adds local positional awareness
- ✓ Negligible parameter cost

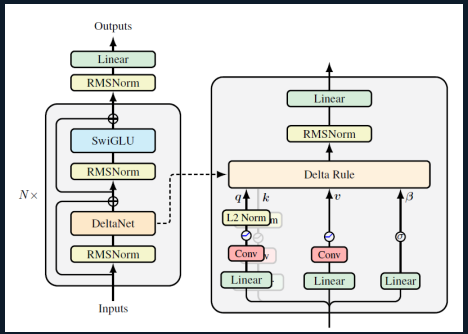


Figure 1: An illustration of DeltaNet neural architecture.

Design Choices: L2-Normalization & Feature Map

Key / Query Normalization

$$\mathbf{k}_t = \frac{\mathrm{SiLU}(\mathbf{W}_K\mathbf{x}_t)}{\|\mathrm{SiLU}(\mathbf{W}_K\mathbf{x}_t)\|_2}, \qquad \mathbf{q}_t = \frac{\mathrm{SiLU}(\mathbf{W}_Q\mathbf{x}_t)}{\|\mathrm{SiLU}(\mathbf{W}_Q\mathbf{x}_t)\|_2}$$

Why L2-norm beats L1-norm?

Eigenvalues of $\mathbf{I} - \beta\mathbf{k}\mathbf{k}^\top$:

- 1 with multiplicity $d-1$
- $1 - \beta\|\mathbf{k}\|_2^2$ with multiplicity 1

L2 norm ensures $\|\mathbf{k}\|_2 = 1$

$\Rightarrow$ eigenvalue = $1 - \beta \in [0, 1]$

No exploding hidden states!

Intuition: $\beta = 1$ case

$$\mathbf{I} - \mathbf{k}\mathbf{k}^\top = \text{projection matrix}$$

Erases exactly the $\mathbf{k}$ subspace

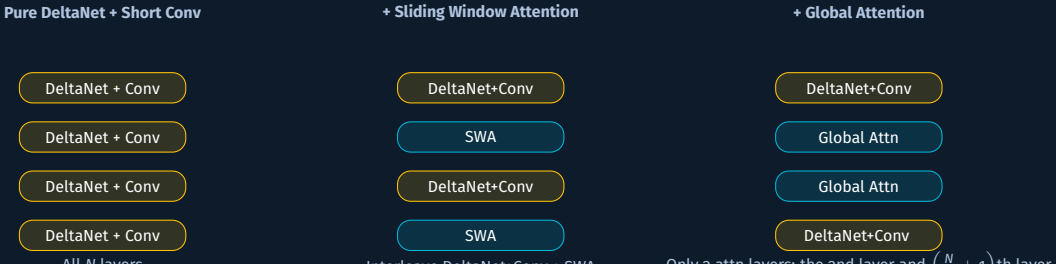
Preserves all $d-1$ other subspaces

"Targeted" forgetting: not wholesale

Ablation: SiLU performs the best!

Normalization & Feature map	Avg.acc ↑
L1-norm + 1+ELU	40.1
L2-norm + 1+ELU	42.1
L2-norm + ReLU	40.9
L2-norm + SiLU	**42.1**

Hybrid Models: DeltaNet + Short Conv + SWA + Global Attention



Motivation

Linear attention misses:

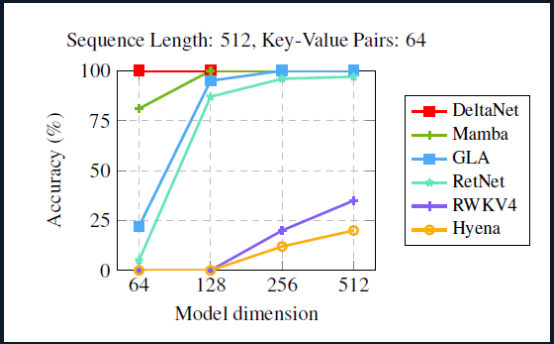
- Positional info & local shifts
- Retrieval-intensive comparisons

$\Rightarrow$ Sliding/global attention helps

Key Insight

- ✓ Short conv layers improve efficiency
- ✓ Interleaving SWA/2 global layers boosts recall
- ✓ Maintains near $O(L)$ cost

Synthetic Benchmarks: Multi-Query Associative Recall



MQAR measures multi-query in-context recall

DeltaNet achieves near-perfect accuracy at 128+ dimensions, outperforming Mamba, GLA, RetNet, RWKV4, and Hyena.

Synthetic Benchmarks: Mechanistic Architecture Design

Model	Compress	Fuzzy	In-Context	Memorize	Noisy	Selective	Average
Transformer	51.6	29.8	94.1	85.2	86.8	99.6	74.5
Hyena	45.2	7.9	81.7	89.5	78.8	93.1	66.0
Multihead Hyena	44.8	14.4	99.0	89.4	98.6	93.0	73.2
Mamba	52.7	6.7	90.4	89.5	90.1	86.3	69.3
GLA	38.8	6.9	80.8	63.3	81.6	88.6	60.0
DeltaNet	42.2	35.7	100	52.8	100	100	71.8

MAD measures capability arising from model structure, not training

- DeltaNet is better at recalling tasks.
- Especially on **Fuzzy Recall**, as expected.
- Struggles on the **"Memorize"** task.

Real-world Language Model Benchmarks: 340M & 1.3B Results

Model	Wiki ppl↓	LMB ppl↓	LMB acc↑	PIQA acc↑	Hella acc_n↑	Wino acc↑	ARC-e acc↑	ARC-c acc_n↑	Avg. acc↑	SWDE acc↑	SQuAD acc↑	FDA acc↑	State exp.
340M params / 15B tokens													
Transformer++	28.39	42.69	31.0	63.3	34.0	50.4	44.5	24.2	41.2	42.2	22.1	21.4	N/A
RetNet (w/o conv)	32.33	49.19	28.6	63.5	33.5	52.5	44.5	23.4	41.0	13.3	27.6	2.9	512x
Mamba (w. conv)	28.39	39.66	30.6	65.0	35.4	50.1	46.3	23.6	41.8	12.4	23.0	2.1	64x
GLA (w/o conv)	28.65	43.35	30.3	64.8	34.5	51.4	45.1	22.7	41.5	18.6	27.2	8.1	128x
GLA (w. conv)	29.47	45.53	31.3	65.1	33.8	51.6	44.4	24.6	41.8	24.0	24.7	7.3	128x
DeltaNet (w/o conv)	29.08	50.87	30.0	63.6	33.6	51.7	46.0	23.0	41.3	24.6	26.9	4.5	128x
DeltaNet (w. conv)	28.24	37.37	32.1	64.8	34.3	52.2	45.8	23.5	42.1	26.4	28.9	12.8	128x
+ Sliding Attn	27.06	38.17	33.4	64.0	35.3	50.9	45.9	23.2	42.1	39.3	32.5	18.8	N/A
+ Global Attn	27.51	35.04	33.5	64.0	34.5	51.7	46.0	23.3	42.1	42.9	32.1	23.1	N/A
1.3B params / 100B tokens													
Transformer++	16.85	13.44	48.9	70.8	49.6	53.6	56.0	26.5	50.9	66.6	31.5	27.4	N/A
RetNet (w/o conv)	18.64	17.27	43.3	70.0	47.3	52.5	54.8	25.6	48.9	42.8	34.7	14.3	512x
Mamba (w. conv)	17.06	13.89	46.2	72.2	40.1	54.1	59.0	28.2	50.0	41.4	35.2	6.2	64x
GLA (w/o conv)	17.22	14.47	46.9	71.8	49.8	53.9	57.2	26.6	51.0	50.6	42.6	19.9	256x
GLA (w. conv)	17.25	14.92	46.2	70.6	49.9	53.0	55.3	27.0	50.4	52.4	37.4	22.3	256x
DeltaNet (w. conv)	16.87	12.21	48.9	71.2	50.2	53.6	57.2	28.3	51.6	49.5	37.4	17.2	128x
+ Sliding Attn	16.56	11.74	49.2	71.8	51.1	52.8	58.9	28.8	52.1	53.3	43.3	22.3	N/A
+ Global Attn	16.55	12.40	48.8	70.8	50.7	54.2	58.4	28.1	51.8	71.0	43.0	29.8	N/A

Scaling to 3B & Training Throughput

3B Parameter Comparison (zero-shot)

Model	Avg	Type
Llama-3.2-3B	62.8	Transformer
PowerLM-3B	62.3	Transformer
DeltaNet-3B	**59.8**	RNN
RecurrentGemma-2B	57.9	RNN
RWKV-6-3B	54.9	RNN
Mamba-2.7B	53.3	RNN

Best-performing RNN at this scale
(Transformer models still lead overall)

Training Throughput (1.3B, H100)

Model	Tokens/sec, 16k x 1
Transformer++	∼30K
Mamba	∼28K (slowest)
GLA	∼43K
DeltaNet	∼41K

✓ DeltaNet ≈ GLA speed
✓ All linear models beat Transformers on **long sequences**
× Slight overhead vs GLA due to head-dim coupling

Limitations

1. Head Dimension Constraint

Intra-state dependencies require processing the **entire head at once**

× Can't tile like GLA can

× Head dim capped at ∼128

Consequence:

Smaller recurrent state size

⇒ recall-intensive tasks suffer at 1.3B scale vs GLA

⇒ training speed still lags behind GLA

2. No Length Extrapolation

DeltaNet trained at length L fails on sequences $> L$

GLA, RetNet, Mamba all generalize to longer sequences

Root cause:

No explicit decay factor

⇒ hidden state "fills up"

Fix: Gated DeltaNet (Yang et al. 2024) adds α_t decay term

Potential Fix for Head Dimension

Block-diagonal generalized Householder matrices:

block size fits GPU SRAM (e.g. 128) while overall head dim stays large

⇒ larger state, better recall — left for future work

Related Work & Real-World Impact

Where DeltaNet Fits (Table 2 in paper)

Model	Recurrence type
Linear Attn	$\mathbf{S} + \mathbf{v}\mathbf{k}^\top$ (additive)
RetNet / GLA	$\mathbf{S} \odot \alpha + \mathbf{v}\mathbf{k}^\top$ (Hadamard)
Mamba	Hadamard (selective)
mLSTM	Hadamard + gate
DeltaNet	$\mathbf{S}(\mathbf{I} - \beta\mathbf{k}\mathbf{k}^\top) + \mathbf{v}\mathbf{k}^\top$

First model to use rank-1 matrix multiplication in the recurrence at scale

Follow-up Work & Adoption

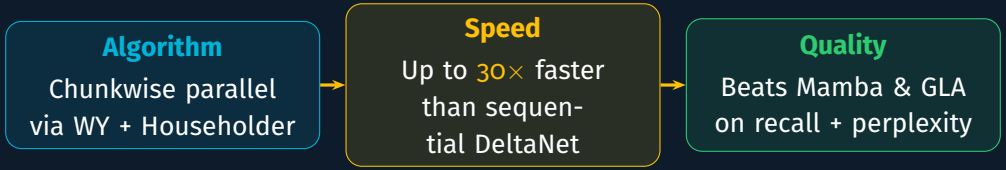
Gated DeltaNet (Yang et al. 2024)
Adds α_t gating $\rightarrow$ fixes length extrapolation

Mamba-2 / SSD framework (Dao & Gu)
Generalizes SSMs to semiseparable matrices
DeltaNet fits as a special case

Titans (Behrouz et al. 2024)
Extends TTT with momentum + weight decay
Delta rule as inner-loop optimizer

Available in `flash-linear-attention` library

Conclusion



One-sentence summary

By exploiting the Householder structure of the delta update and the WY representation, this paper makes DeltaNet trainable at scale — delivering **linear-time, constant-memory inference** with **recall quality close to full attention.**

Key takeaways

- ✓ Delta rule > additive rule for recall
- ✓ Hybrid models beat Transformers
- ✓ Hardware-efficiency is achievable

Open questions

- ? Scale state size without head-dim cost
- ? Length generalization
- ? Beyond rank-1 transition matrices

Appendix: Linear Attention in Transformers

$$A_t^{\text{softmax}} = \text{softmax}(Q_t K^\top)V = \sum_{i=1}^{L} \frac{\exp(Q_t K_i^\top)}{\sum_j \exp(Q_t K_j^\top)} V_i$$

Linear Attention: $A_t = \sum_{i=1}^{t} Q_t K_i^\top V_i = Q_t \underbrace{\sum_{i=1}^{t} K_i V_i^\top}_{S_t}, \quad S_t = S_{t-1} + K_t V_t^\top$

Unrolled: $A_t = Q_t(K_1 V_1^\top + \cdots + K_t V_t^\top)$

Softmax contrast: $\text{softmax}(Q_t K^\top)_i \approx \begin{cases} 1 & i = \arg\max_j Q_t K_j^\top \\ 0 & \text{otherwise} \end{cases}$

Additive Update Issues

$$S_t = \sum_{i=1}^{t} K_i V_i^\top \quad \text{(no forgetting)}$$

$$L > d \implies \exists i \neq j : K_i \cdot K_j \text{ collision}$$

$$A_t = Q_t \sum_i K_i V_i^\top \quad \text{(multiple terms contribute)}$$

Softmax: $\sum_i \text{softmax}(Q_t K^\top)_i V_i \approx V_{i*}$