Linear Transformers for Efficient Sequence Modeling

Yoon Kim
MIT
How do large language models work?

MIT is located in

LLM

?
How do large language models work?

MIT is located in Cambridge
How do large language models work?

MIT is located in Cambridge

LLM
How do large language models work?

MIT is located in Cambridge

LLM

Massachusetts
How do large language models work?
Generative AI through future prediction
Generative AI through future prediction
Generative AI through future prediction

MIT is located in Cambridge

"Transformers"
MIT is located in Cambridge?
Transformers

MIT is located in

Word vectors

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>3.4</td>
<td>-1.9</td>
<td>2.7</td>
</tr>
<tr>
<td>1.5</td>
<td>-2.1</td>
<td>-8.7</td>
<td>0.8</td>
</tr>
<tr>
<td>-2.1</td>
<td>3.5</td>
<td>0.3</td>
<td>2.3</td>
</tr>
</tbody>
</table>
Transformers

“Attend” over all previous words

MIT is located in
Transformers

MIT is located in Cambridge

“Attend” over all previous words

Predict the next token with the attended vector

MIT is located in
Transformers

MIT is located in Cambridge
Transformers

MIT is located in Cambridge, Massachusetts.
MIT is located in Cambridge, Massachusetts.

Attention Layer

MIT is located in Cambridge.
Transformers have difficulty scaling to long sequences
Transformers have difficulty scaling to long sequences

- Harry Potter series: 1M words
- Lord of the Rings films: 300M frame snippets
- Human DNA: 3.2B nucleotides
How can we maintain the **accuracy** of attention while enabling **efficient** training and inference?
Today: Efficient alternatives to attention in Transformers

Gated Linear Attention Transformers with Hardware-Efficient Training

Songlin Yang*, Bailin Wang*, Yikang Shen, Rameswar Panda, Yoon Kim
ICML ’24

Parallelizing Linear Transformers with the Delta Rule over Sequence Length

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, Yoon Kim
arXiv ’24
Today: Efficient alternatives to attention in Transformers

Gated Linear Attention Transformers with Hardware-Efficient Training

Songlin Yang*, Bailin Wang*, Yikang Shen, Rameswar Panda, Yoon Kim
ICML ’24

Parallelizing Linear Transformers with the Delta Rule over Sequence Length

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, Yoon Kim
arXiv ’24
Attention in Transformers [Vaswani et al. '17]

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez† University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin*† illia.polosukhin@gmail.com
Attention: Training

$L : \text{sequence length}$

d : hidden state dimension

$X \in \mathbb{R}^{L \times d}$
Attention: Training

$L$: sequence length
$d$: hidden state dimension

$O \in \mathbb{R}^{L \times d}$

$O = \text{SelfAttention}(X)$

$X \in \mathbb{R}^{L \times d}$
Attention: Training

$L$ : sequence length
$d$ : hidden state dimension

\[ O(Ld^2) \quad Q, K, V = XW_Q, XW_K, XW_V \]

\[ X \in \mathbb{R}^{L \times d} \]

Key \quad K
Value \quad V
Query \quad Q
Attention: Training

$L$ : sequence length
$d$ : hidden state dimension

\( O(L^2d) \quad A = \text{softmax}(QK^T \odot M) \in \mathbb{R}^{L \times L} \)

\( O(Ld^2) \quad Q,K,V = XW_Q, XW_K, XW_V \)

\( X \in \mathbb{R}^{L \times d} \)

Key \quad K

Value \quad V

Query \quad Q
Attention: Training

$L$ : sequence length
$d$ : hidden state dimension

\[ O(L^2d) \quad O = AV \in \mathbb{R}^{L \times d} \]

\[ O(L^2d) \quad A = \text{softmax}(QK^T \odot M) \in \mathbb{R}^{L \times L} \]

\[ O(Ld^2) \quad Q,K,V = XW_Q, XW_K, XW_V \]

\[ X \in \mathbb{R}^{L \times d} \]

Key  K
Value  V
Query  Q
Attention: Training

$L$ : sequence length
$d$ : hidden state dimension

$O(L^2d) \quad O = AV \in \mathbb{R}^{L \times d}$

$O(L^2d) \quad A = \text{softmax}(QK^T \circ M) \in \mathbb{R}^{L \times L}$

$O(Ld^2) \quad Q,K,V = XW_Q,XW_K,XW_V$

$X \in \mathbb{R}^{L \times d}$

Attention: Number sequential steps is independent of sequence length!
Attention: Training

Attention requires \( O(L^2d + Ld^2) \) work but can be done in \( O(1) \) steps → Parallel training that is rich in matmuls.

\[
O(L^2d) \quad O = AV \in \mathbb{R}^{L \times d}
\]

\[
O(L^2d) \quad A = \text{softmax}(QK^T \odot M) \in \mathbb{R}^{L \times L}
\]

\[
O(Ld^2) \quad Q,K,V = XW_Q, XW_K, XW_V
\]

\[
X \in \mathbb{R}^{L \times d}
\]

Attention: Number sequential steps is independent of sequence length!
Attention: Training

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training (&quot;Parallel Form&quot;)</strong></td>
<td></td>
</tr>
<tr>
<td>[ O = \text{softmax} \left( (QK^T) \odot M \right) V ]</td>
<td></td>
</tr>
<tr>
<td>Compute</td>
<td>[ O(L^2) ]</td>
</tr>
<tr>
<td>Memory</td>
<td>[ O(L) ]</td>
</tr>
<tr>
<td>Steps</td>
<td>[ O(1) ]</td>
</tr>
</tbody>
</table>
Attention: Generative Inference

\[ q_t, k_t, v_t = x_t W_Q, x_t W_K, x_t W_V \]

- Key: K
- Value: V
- Query: Q
Attention: Generative Inference

\[ q_t, k_t, v_t = x_t W_Q, x_t W_K, x_t W_V \]

<table>
<thead>
<tr>
<th>Key</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>V</td>
</tr>
<tr>
<td>Query</td>
<td>Q</td>
</tr>
</tbody>
</table>
Attention: Generative Inference

\[
\frac{\exp(q_t^T k_j)}{\sum_{i=1}^t \exp(q_t^T k_i)}
\]

\(q_t, k_t, v_t = x_t W_Q, x_t W_K, x_t W_V\)

Key  \(K\)
Value  \(V\)
Query  \(Q\)
Attention: Generative Inference

\[ o_t = \sum_{j=1}^{t} \frac{\exp(q_t^T k_{j})}{\sum_{l=1}^{t} \exp(q_t^T k_{l})} v_j \]

\[ q_t, k_t, v_t = x_t W_Q, x_t W_K, x_t W_V \]

Key  K
Value  V
Query  Q
Attention: Generative Inference

\[ o_t = \sum_{j=1}^{t} \frac{\exp(q_t^T k_j)}{\sum_{l=1}^{t} \exp(q_t^T k_l)} v_j \]

\[ q_t, k_t, v_t = x_t W_Q, x_t W_K, x_t W_V \]

Key \ K
Value \ V
Query \ Q
Attention: Generative Inference

\[
o_t = \sum_{j=1}^{t} \frac{\exp(q_t^T k_j)}{\sum_{i=1}^{t} \exp(q_t^T k_i)} v_j
\]

\[q_t, k_t, v_t = x_t W_Q, x_t W_K, x_t W_V\]

Key \( K \)
Value \( V \)
Query \( Q \)
Attention: Generative Inference

\[ o_t = \sum_{j=1}^{t} \frac{\exp(q_t^T k_j)}{\sum_{l=1}^{t} \exp(q_t^T k_l)} v_j \]

\( q_t, k_t, v_t = x_t W_Q, x_t W_K, x_t W_V \)

- Key: K
- Value: V
- Query: Q
Attention: Generative Inference

\[ o_t = \sum_{j=1}^{t} \frac{\exp(q_t^T k_j)}{\sum_{l=1}^{t} \exp(q_t^T k_l)} v_j \]

\[ q_t, k_t, v_t = x_t W_Q, x_t W_K, x_t W_V \]

Key \quad K
Value \quad V
Query \quad Q
Attention: Generative Inference

\[ o_t = \sum_{j=1}^{t} \frac{\exp(q_t^\top k_j)}{\sum_{l=1}^{t} \exp(q_t^\top k_l)} v_j \]

Need to keep around “KV-cache” that takes \( O(L) \) memory.

\[ q_t, k_t, v_t = x_t W_Q, x_t W_K, x_t W_V \]

Key \( K \)
Value \( V \)
Query \( Q \)
### Attention

<table>
<thead>
<tr>
<th></th>
<th>Training (“Parallel Form”)</th>
<th>Inference (“Recurrent Form”)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compute</strong></td>
<td>$O(L^2)$</td>
<td>$O(L^2)$</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>$O(L)$</td>
<td>$O(L)$</td>
</tr>
<tr>
<td><strong>Steps</strong></td>
<td>$O(1)$</td>
<td>$O(L)$</td>
</tr>
</tbody>
</table>

Training:

$$O = \text{softmax} \left( (QK^T) \odot M \right) V$$

Inference:

$$o_t = \frac{\sum_{i=1}^{t} \exp(q_t k_i^T) v_i}{\sum_{i=1}^{t} \exp(q_t k_i^T)}$$
### Attention

<table>
<thead>
<tr>
<th>Training (&quot;Parallel Form&quot;)</th>
<th>Inference (&quot;Recurrent Form&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O = \text{softmax}((QK^T) \odot M)V$</td>
<td>$o_t = \frac{\sum_{i=1}^{t} \exp(q_t^k_i) v_i}{\sum_{i=1}^{t} \exp(q_t^k_i)}$</td>
</tr>
<tr>
<td>Compute</td>
<td>$O(L^2)$</td>
</tr>
<tr>
<td>Memory</td>
<td>$O(L)$</td>
</tr>
<tr>
<td>Steps</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Attention enables scalable training of accurate sequence models, but requires:

- Quadratic compute for training.
- Linear memory for inference.
From Softmax to Linear Attention [Katharopoulos et al. ’20]

Softmax Attention

\[ O = \text{softmax} \left( (QK^T) \odot M \right) V \]

(Simple) Linear Attention

\[ O = (QK^T) \odot M V \]
From Softmax to Linear Attention [Katharopoulos et al. ’20]

Softmax Attention

(Simple) Linear Attention

\[ O = \text{softmax}((QK^T) \circ M)V \]

\[ \{\infty, 0\}^{L \times L} \]

\[ O = ((QK^T) \circ M)V \]

\[ \{0, 1\}^{L \times L} \]
# From Softmax to Linear Attention

[Katharopoulos et al. '20]

<table>
<thead>
<tr>
<th></th>
<th>Training (“Parallel Form”)</th>
<th>Inference (“Recurrent Form”)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Softmax Attention</strong></td>
<td>( O = \text{softmax} \left( (QK^T) \odot M \right) V )</td>
<td>( o_t = \sum_{j=1}^{t} \frac{\exp(q_t^T k_j)}{\sum_{l=1}^{t} \exp(q_t^T k_l)} v_j )</td>
</tr>
<tr>
<td><strong>(Simple) Linear Attention</strong></td>
<td>( O = ((QK^T) \odot M) V )</td>
<td>( o_t = \sum_{j=1}^{t} (q_t^T k_j) v_j )</td>
</tr>
</tbody>
</table>
Linear Attention: Inference

\[ o_t = \sum_{j=1}^{t} (q_t^\top k_j)v_j \]
Linear Attention: Inference

\[ o_t = \sum_{j=1}^{t} (q_t^\top k_j) v_j = q_t^\top \left( \sum_{j=1}^{t} k_j v_j^\top \right) \]
Linear Attention: Inference

\[ o_t = \sum_{j=1}^{t} (q_t^T k_j) v_j = q_t^T \left( \sum_{j=1}^{t} k_j v_j^T \right) \]

\[ S_t \in \mathbb{R}^{d \times d} \]
Linear Attention: Inference

\[ o_t = \sum_{j=1}^{t} (q_t^T k_j) v_j = q_t^T \left( \sum_{j=1}^{t} k_j v_j^T \right) \]

\[ S_t = S_{t-1} + k_t v_t^T \]

\[ o_t = q_t^T S_t \]
Linear Attention: Inference

\[ o_t = \sum_{j=1}^{t} (q_t^T k_j) v_j = q_t^T \left( \sum_{j=1}^{t} k_j v_j^T \right) \]

\[ S_t = S_{t-1} + k_t v_t^T \]

\[ o_t = q_t^T S_t \]
Linear Attention: Inference

\[ o_t = \sum_{j=1}^{t} (q_t^T k_j) v_j = q_t^T \left( \sum_{j=1}^{t} k_j v_j^T \right) \]

\[ S_t = S_{t-1} + k_t v_t^T \]

\[ o_t = q_t^T S_t \]
Linear Attention: Inference

\[ o_t = \sum_{j=1}^{t} (q_t^\top k_j) v_j = q_t^\top \left( \sum_{j=1}^{t} k_j v_j^\top \right) \]

\[ S_t = S_{t-1} + k_t v_t^\top \]

\[ o_t = q_t^\top S_t \]
Linear Attention: Inference

\[ o_t = \sum_{j=1}^{t} (q_t^T k_j) v_j = q_t^T \left( \sum_{j=1}^{t} k_j v_j^T \right) \]

\[ S_t = S_{t-1} + k_t v_t^T \]

\[ o_t = q_t^T S_t \]
Linear Attention: Inference

\[ o_t = \sum_{j=1}^{t} (q_t^T k_j) v_j = q_t^T \left( \sum_{j=1}^{t} k_j v_j^T \right) \]

\[ S_t = S_{t-1} + k_t v_t^T \]

\[ o_t = q_t^T S_t \]
Linear Attention: Inference

\[ o_t = \sum_{j=1}^{t} (q_t^T k_j) v_j = q_t^T \left( \sum_{j=1}^{t} k_j v_j^T \right) \]
\[ S_t = S_{t-1} + k_t v_t^T \]
\[ o_t = q_t^T S_t \]

Linear Attention = Linear RNNs with matrix-valued hidden states → Constant-memory inference!
## Linear Attention

<table>
<thead>
<tr>
<th>Training (&quot;Parallel Form&quot;)</th>
<th>Inference (&quot;Recurrent Form&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbf{O} = ((\mathbf{QK}^T) \odot \mathbf{M}) \mathbf{V} )</td>
<td>( o_t = \sum_{j=1}^{t} (q_t^T k_j) v_j )</td>
</tr>
<tr>
<td><strong>Compute</strong></td>
<td><strong>Compute</strong></td>
</tr>
<tr>
<td>( O(L^2) )</td>
<td>( O(L) )</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td><strong>Memory</strong></td>
</tr>
<tr>
<td>( O(L) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td><strong>Steps</strong></td>
<td><strong>Steps</strong></td>
</tr>
<tr>
<td>( O(1) )</td>
<td>( O(L) )</td>
</tr>
</tbody>
</table>
Linear Attention: Naive Parallel Form

<table>
<thead>
<tr>
<th></th>
<th>Training (“Parallel Form”)</th>
<th>Inference (“Recurrent Form”)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compute</strong></td>
<td>$O(L^2)$ ☹</td>
<td>$O(L)$ ☺</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>$O(L)$ ☺</td>
<td>$O(1)$ ☺</td>
</tr>
<tr>
<td><strong>Steps</strong></td>
<td>$O(1)$ ☺</td>
<td>$O(L)$</td>
</tr>
</tbody>
</table>

**Formula**

Training:

$$O = ((QK^T) \odot M)V$$

Inference:

$$o_t = \sum_{j=1}^{t} (q_t^T k_j)v_j$$

Linear attention has constant-memory inference, but still requires:

- Quadratic compute for training.
- (Can theoretically use recurrent form + parallel scan for $O(L)$ compute and $O(\log L)$ work, but this is not at all practical.)
Linear Attention: “Chunkwise Parallel Form” [Hua et al. ’22, Sun et al. ’23]

Pure RNN ·
Linear Attention: “Chunkwise Parallel Form” [Hua et al. ’22, Sun et al. ’23]

Pure RNN → “Chunk-level” RNN
Linear Attention: “Chunkwise Parallel Form” [Hua et al. ’22, Sun et al. ’23]

Pure RNN → “Chunk-level” RNN

\[ S_{i+1} = S_i + \sum_{j=iC+1}^{(i+1)C} k_j^\top v_j \]

\[ K_{[i]}^\top V_{[i]} \]

Chunk 1

\[ S_{[1]} \]

Chunk 2

\[ S_{[2]} \]

Chunk 3

\[ S_{[3]} \]
Linear Attention: “Chunkwise Parallel Form” [Hua et al. ’22, Sun et al. ’23]

Pure RNN → “Chunk-level” RNN
Linear Attention: “Chunkwise Parallel Form” [Hua et al. ’22, Sun et al. ’23]

Pure RNN $\rightarrow$ “Chunk-level” RNN

\[ O_{i+1} = Q_{i+1} S_i \]

Contribution from previous chunk.
Linear Attention: “Chunkwise Parallel Form” [Hua et al. ’22, Sun et al. ’23]

Pure RNN $\rightarrow$ “Chunk-level” RNN

$$
O_{[i+1]} = Q_{[i+1]} S_{[i]} + ((Q_{[i+1]} K_{[i+1]}^T) \odot M) V_{[i+1]}
$$

Contribution from previous chunk.

Chunk-level (linear) attention for contribution from current chunk.
Linear Attention: “Chunkwise Parallel Form” [Hua et al. ’22, Sun et al. ’23]

<table>
<thead>
<tr>
<th></th>
<th>Fully Parallel Form</th>
<th>Chunkwise Parallel Form</th>
<th>Fully Recurrent Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute</td>
<td>$O(L^2)$</td>
<td>$O(LC')$</td>
<td>$O(L)$</td>
</tr>
<tr>
<td>Memory</td>
<td>$O(L)$</td>
<td>$O(C)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Steps</td>
<td>$O(1)$</td>
<td>$O\left(\frac{L}{C}\right)$</td>
<td>$O(L)$</td>
</tr>
</tbody>
</table>

Chunkwise parallel form interpolates between fully parallel and recurrent forms.
- $C = L \rightarrow$ Fully parallel form
- $C = 1 \rightarrow$ Fully recurrent form
**Linear Attention: “Chunkwise Parallel Form”** [Hua et al. ’22, Sun et al. ’23]

<table>
<thead>
<tr>
<th></th>
<th>Fully Parallel Form</th>
<th>Chunkwise Parallel Form</th>
<th>Fully Recurrent Form</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compute</strong></td>
<td>$O(L^2)$</td>
<td>$O(LC')$</td>
<td>$O(L)$</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>$O(L)$</td>
<td>$O(C')$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Steps</strong></td>
<td>$O(1)$</td>
<td>$O\left(\frac{L}{C}\right)$</td>
<td>$O(L)$</td>
</tr>
</tbody>
</table>

Chunkwise parallel form interpolates between fully parallel and recurrent forms.

- $C = L \rightarrow$ Fully parallel form
- $C = 1 \rightarrow$ Fully recurrent form
Linear Attention: Issues

Issue 1:
Slower than optimized implementations of softmax attention in practice.
Linear Attention: Issues

Issue 2:
Underperforms softmax attention by a significant margin.

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL ↓</th>
<th>LM Eval ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softmax attention</td>
<td>16.9</td>
<td>50.9</td>
</tr>
<tr>
<td>Linear attention with decay (RetNet)</td>
<td>18.6</td>
<td>48.9</td>
</tr>
<tr>
<td>$S_t = \gamma S_{t-1} + k_t v_t^T$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Our Contributions

Issue 1:
Slower than optimized implementations of softmax attention in practice. → Hardware-aware implementation of linear attention.

Issue 2:
Underperforms softmax attention by a significant margin. → Linear attention with data-dependent “forget” gate
Background: FlashAttention [Dao et al. ‘22, Dao ‘23]

Memory Hierarchy with Bandwidth & Memory Size

- **SRAM:** 19 TB/s (20 MB)
- **HBM:** 1.5 TB/s (40 GB)
- **DRAM:** 12.8 GB/s (>1 TB)

Diagram of FlashAttention:
- **Q:** $N \times d$
- **K:** $d \times N$
- **V:** $N \times d$

Attention on GPT-2:
- **Matmul**
- **Dropout**
- **Softmax**
- **Mask**
- **Matmul**
- **Fused Kernel**

[Image credit: Dao et al. ‘22]
Background: FlashAttention [Dao et al. ’22, Dao ’23]

Want to minimize data movement between HBM and SRAM.
Background: FlashAttention [Dao et al. ’22, Dao ’23]

\[ Q, K, V = X W_Q, X W_K, X W_V \]

\[ A = \text{softmax}(QK^T \odot M) \]

\[ O = AV \]

Fused attention:
Never instantiate this in slower HBM.
Calculate \( O = AV \) in SRAM.

[Image credit: Dao et al. ’22]
Background: FlashAttention [Dao et al. ’22, Dao ’23]
FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention

**Algorithm 1** \textsc{FlashLinearAttention}: Forward Pass

**Input:** \( Q, K, V \in \mathbb{R}^{L \times d}, V \in \mathbb{R}^{L \times d} \), chunk size \( C \in [L] \), materialize \( \in \{\text{True, False}\} \)

Divide \( Q, K, V \) into \( N = \frac{L}{C} \) blocks \( \{Q_{[1]} \ldots Q_{[N]}\}, \{K_{[1]} \ldots K_{[N]}\} \) of size \( C \times d \) each.

Initialize \( S = 0 \in \mathbb{R}^{d \times d} \) on SRAM

On chip, construct causal mask \( M \in \mathbb{R}^{C \times C} \)

if materialize then \( \triangleright \) the materialization version

for \( n \leftarrow 1, N \) do

Store \( S \) to HBM as \( S_{[n]} \).

Load \( K_{[n]}, V_{[n]} \in \mathbb{R}^{C \times d} \) from HBM to SRAM.

On chip, compute \( O = Q_{[n]} S_{[n]} + (Q_{[n]} K_{[n]}^T \odot M) V_{[n]} \).

end for

parfor \( n \leftarrow 1, N \) do

Load \( Q_{[n]}, K_{[n]}, V_{[n]}, S_{[n]} \) from HBM to SRAM.

On chip, compute \( O' = Q_{[n]} S_{[n]} + (Q_{[n]} K_{[n]}^T \odot M) V_{[n]} \).

Store \( O' \) to HBM as \( O_{[n]} \).

end parfor

return \( O = \{O_{[1]} \ldots O_{[N]}\}, S = \{S_{[1]} \ldots S_{[N]}\} \).

else \( \triangleright \) the non-materialization version

for \( n \leftarrow 1, N \) do

Load \( Q_{[n]}, K_{[n]}, V_{[n]} \in \mathbb{R}^{C \times d} \) from HBM to SRAM.

On chip, compute \( O' = Q_{[n]} S + (Q_{[n]} K_{[n]}^T \odot M) V_{[n]} \).

On chip, compute \( S = S + K_{[n]}^T V_{[n]} \).

Store \( O' \) to HBM as \( O_{[n]} \).

end for

return \( O = \{O_{[1]} \ldots O_{[N]}\} \).

end if
FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention

![Diagram of FlashLinearAttention](image)

**Running speed**

- **FLASHATTENTION-2**
- **FLASHLINEARATTENTION (ours)**
- **Pure PyTorch Linear Attention**

![Graph showing running speed vs sentence length](image)
Flash Linear Attention

This repo aims at providing a collection of efficient Triton-based implementations for state-of-the-art linear attention models.

<table>
<thead>
<tr>
<th>Year</th>
<th>Author/Name</th>
<th>Title</th>
<th>Arxiv</th>
<th>Official</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-12</td>
<td>GLA (@MIT@IBM)</td>
<td>Gated Linear Attention Transformers with Hardware-Efficient Training</td>
<td>[arxiv]</td>
<td>[official]</td>
<td>code</td>
</tr>
<tr>
<td>2023-12</td>
<td>Based (@Stanford@Hazyresearch)</td>
<td>An Educational and Effective Sequence Mixer</td>
<td>[blog]</td>
<td>[official]</td>
<td>code</td>
</tr>
<tr>
<td>2024-01</td>
<td>Rebased</td>
<td>Linear Transformers with Learnable Kernel Functions are Better In-Context Models</td>
<td>[arxiv]</td>
<td>[official]</td>
<td>code</td>
</tr>
<tr>
<td>2021-02</td>
<td>Delta Net</td>
<td>Linear Transformers Are Secretly Fast Weight Programmers</td>
<td>[arxiv]</td>
<td>[official]</td>
<td>code</td>
</tr>
</tbody>
</table>
Gated Linear Attention: Data-dependent Multiplicative Gate

Simple Linear Attention

\[ S_t = S_{t-1} + k_t v_t^T \]
Gated Linear Attention: Data-dependent Multiplicative Gate

Simple Linear Attention

\[ S_t = S_{t-1} + k_t v_t^T \]

Gated Linear Attention

\[ S_t = G_t \odot S_{t-1} + k_t v_t^T \]

\[ G_t = \alpha_t 1^T, \quad \alpha_t = \sigma(x_t W_{\alpha_1} W_{\alpha_2})^{1/\tau} \]
Gated Linear Attention: Parallel Forms

Simple Linear Attention

\[ O = \left( (QK^T) \odot M \right) V \]

Gated Linear Attention

\[
O = \left( \left( \underbrace{\left( (Q \odot B) \left( \frac{K}{B} \right)^T \right)}_{P} \right) \odot M \right) V
\]

GLA also admits a chunkwise parallel form for subquadratic, parallel training!

\[
B_t := \prod_{j=1}^{t} \alpha_j
\]

\[
P_{ij} = \sum_{k=1}^{d} Q_{ik} K_{jk} \exp(\log B_{ik} - \log B_{jk})
\]
Gated Linear Attention: Throughput

Training throughput

GPU memory usage

Tokens per second (K/s)

Gigabyte (GB)

<table>
<thead>
<tr>
<th>Training length/Batch size</th>
<th>Transformer++</th>
<th>Mamba</th>
<th>GLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048/8</td>
<td>50</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>4096/4</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>8192/2</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>16284/1</td>
<td>20</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Training length/Batch size</th>
<th>Transformer++</th>
<th>Mamba</th>
<th>GLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048/8</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>4096/4</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>8192/2</td>
<td>20</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>16284/1</td>
<td>10</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
## Gated Linear Attention: Performance

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL</th>
<th>LM Eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer++</td>
<td>16.9</td>
<td>50.9</td>
</tr>
<tr>
<td>RetNet</td>
<td>18.6</td>
<td>48.9</td>
</tr>
<tr>
<td>Mamba</td>
<td>17.1</td>
<td>50.0</td>
</tr>
<tr>
<td>Gated Linear Attention</td>
<td>17.2</td>
<td>51.1</td>
</tr>
</tbody>
</table>

1.3B models trained on 100B tokens
Gated Linear Attention: Recall-oriented Tasks

SUBSTANTIAL EQUIVALENCE DETERMINATION DECISION SUMMARY

A. 510(k) Number: K143329

B. Purpose for Submission: To obtain clearance for a new device, Amplivue® Trichomonas Assay

C. Measurand: A conserved multi-copy sequence of Trichomonas vaginalis genomic DNA

D. Type of Test: Nucleic acid amplification assay (Helicase-dependent Amplification, HDA)

E. Applicant: Quidel Corporation

F. Proprietary and Established Names: Amplivue® Trichomonas Assay

G. Regulatory Information: 1. Regulation section: 21 CFR 866.3860

2. Classification: Class II

3. Product code: OUY - Trichomonas vaginalis nucleic acid amplification test system

4. Panel: 83 - Microbiology

H. Intended Use: 1. Intended use(s): The AmpliVue® Trichomonas Assay is an in vitro diagnostic test, uses isothermal amplification technology (helicase-dependent amplification, HDA) for the qualitative detection of Trichomonas vaginalis nucleic acids isolated from clinician-collected vaginal swab specimens obtained from symptomatic or asymptomatic females to aid in the diagnosis of trichomoniasis.

2. Indication(s) for use: Same as Intended Use

3. Special conditions for use statement(s): For prescription use only

4. Special instrument requirements: None

I. Device Description: The AmpliVue® Trichomonas Assay is a self-contained disposable amplicon detection device that uses an isothermal amplification technology named Helicase-Dependent Amplification (HDA) for the detection of Trichomonas vaginalis in clinician-collected vaginal swabs from symptomatic and asymptomatic women. The assay targets a conserved multi-copy sequence of the T. vaginalis genomic DNA. The vaginal swab is eluted in a lysis tube, and the cells are lysed by heat treatment. After heat treatment, an aliquot of the lysed specimen is transferred into a dilution tube. An aliquot of this diluted sample is then added to a reaction tube containing a lyophilized mix of HDA reagents including primers specific for the amplification of a...
SUBSTANTIAL EQUIVALENCE DETERMINATION DECISION SUMMARY

A. 510(k) Number: K143329

B. Purpose for Submission: To obtain clearance for a new device, Amplivue® Trichomonas Assay

C. Measuring Method: A conserved multi-copy sequence of Trichomonas vaginalis genomic DNA

D. **Type of Test:** Nucleic acid amplification assay (Helicase-dependent Amplification, HDA)

E. Applicant: Quidel Corporation

F. Proprietary and Established Names: Amplivue® Trichomonas Assay

G. Regulatory Information:
   1. Regulation section: 21 CFR 866.3860
   2. Classification: Class II
   3. Product code: OUY - Trichomonas vaginalis nucleic acid amplification test system
   4. Panel: 83 - Microbiology

H. Intended Use:
   1. Intended use(s): The Amplivue® Trichomonas Assay is an in vitro diagnostic test, uses isothermal amplification technology (helicase-dependent amplification, HDA) for the qualitative detection of Trichomonas vaginalis nucleic acids isolated from clinician-collected vaginal swab specimens obtained from symptomatic or asymptomatic females to aid in the diagnosis of trichomoniasis.
   2. Indication(s) for use: Same as Intended Use
   3. Special conditions for use statement(s): For prescription use only
   4. Special instrument requirements: None

I. Device Description:
   The Amplivue® Trichomonas Assay is a self-contained disposable amplicon detection device that uses an isothermal amplification technology named Helicase-Dependent Amplification (HDA) for the detection of Trichomonas vaginalis in clinician-collected vaginal swabs from symptomatic and asymptomatic women. The assay targets a conserved multi-copy sequence of the T. vaginalis genomic DNA. The vaginal swab is eluted in a lysis tube, and the cells are lysed by heat treatment. After heat treatment, an aliquot of the lysed specimen is transferred into a dilution tube. An aliquot of this diluted sample is then added to a reaction tube containing a lyophilized mix of HDA reagents including primers specific for the amplification of a...

**Type of Test → Nucleic acid amplification assay**

*Helicase-dependent Amplification, HDA*
## Gated Linear Attention: Recall-oriented Tasks

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL ↓</th>
<th>LM Eval ↑</th>
<th>Retrieval ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer++</td>
<td>16.9</td>
<td>50.9</td>
<td>41.8</td>
</tr>
<tr>
<td>RetNet</td>
<td>18.6</td>
<td>48.9</td>
<td>30.6</td>
</tr>
<tr>
<td>Mamba</td>
<td>17.1</td>
<td>50.0</td>
<td>27.6</td>
</tr>
<tr>
<td>Gated Linear Attention</td>
<td>17.2</td>
<td>51.1</td>
<td>37.7</td>
</tr>
</tbody>
</table>

1.3B models trained on 100B tokens
Gated Linear Attention: Length Generalization

![Graph showing position bucket (K) vs. some metric for different models: Transformer++, RetNet, RetNet*, Mamba, GLA, GLA* with various configurations (2K, 8K, 12x2K).]
Gated Linear Attention Transformors or State-Space Models?

Gated Linear Attention

\[ S_t = G_t \odot S_{t-1} + k_t v_t^T \]

Mamba

\[ h'(t) = Ah(t) + Bx(t) \quad (1a) \]
\[ h_t = \bar{A}h_{t-1} + Bx_t \quad (2a) \]
\[ y(t) = Ch(t) \quad (1b) \]
\[ y_t = Ch_t \quad (2b) \]
\[ \bar{K} = (\bar{C}B, C\bar{A}B, \ldots, C\bar{A}^kB, \ldots) \]
\[ y = x \ast \bar{K} \]

\[ \bar{A} = \exp(\Delta A) \quad \bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B \]

Algorithm 1 SSM (S4)

Input: \( x : (B, L, D) \)
Output: \( y : (B, L, D) \)

1. \( A : (D, N) \leftarrow \text{Parameter} \) \( \quad \) \( \rightarrow \) Represents structured \( N \times N \) matrix
2. \( B : (D, N) \leftarrow \text{Parameter} \)
3. \( C : (D, N) \leftarrow \text{Parameter} \)
4. \( \Delta : (D) \leftarrow \tau_{\Delta} (\text{Parameter}) \)
5. \( \bar{A}, \bar{B} : (D, N) \leftarrow \text{discretize}(\Delta, A, B) \)
6. \( y \leftarrow \text{SSM}(\bar{A}, \bar{B}, C)(x) \) \( \quad \rightarrow \) Time-invariant: recurrence or convolution

7. \( \text{return } y \)

Algorithm 2 SSM + Selection (S6)

Input: \( x : (B, L, D) \)
Output: \( y : (B, L, D) \)

1. \( A : (D, N) \leftarrow \text{Parameter} \) \( \quad \) \( \rightarrow \) Represents structured \( N \times N \) matrix
2. \( B : (B, L, N) \leftarrow s_B(x) \)
3. \( C : (B, L, N) \leftarrow s_C(x) \)
4. \( \Delta : (B, L, D) \leftarrow \tau_{\Delta} (\text{Parameter} + s_A(x)) \)
5. \( \bar{A}, \bar{B} : (B, L, D, N) \leftarrow \text{discretize}(\Delta, A, B) \)
6. \( y \leftarrow \text{SSM}(\bar{A}, \bar{B}, C)(x) \) \( \quad \rightarrow \) Time-varying: recurrence (scan) only

7. \( \text{return } y \)
Gated Linear Attention Transformers are State-Space Models!

\[ S_t = G_t \odot S_{t-1} + k_t v_t \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameterization</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mamba</td>
<td>[ G_t = \exp(-(1^T \alpha_t) \odot \exp(A)), \alpha_t = \text{softplus}(x_t W_{\alpha_1} W_{\alpha_2}) ]</td>
<td>( A, W_{\alpha_1}, W_{\alpha_2} )</td>
</tr>
<tr>
<td>Mamba-2</td>
<td>[ G_t = \gamma_t 1^T 1, \quad \gamma_t = \exp(-\text{softplus}(x_t W_{\gamma}) \exp(a)) ]</td>
<td>( W_{\gamma}, a )</td>
</tr>
<tr>
<td>mLSTM</td>
<td>[ G_t = \gamma_t 1^T 1, \quad \gamma_t = \sigma(x_t W_{\gamma}) ]</td>
<td>( W_{\gamma} )</td>
</tr>
<tr>
<td>Gated RetNet</td>
<td>[ G_t = \gamma_t 1^T 1, \quad \gamma_t = \sigma(x_t W_{\gamma})^{\frac{1}{r}} ]</td>
<td>( W_{\gamma} )</td>
</tr>
<tr>
<td>HGRN-2</td>
<td>[ G_t = \alpha_t^T 1, \quad \alpha_t = \gamma + (1 - \gamma)\sigma(x_t W_{\alpha}) ]</td>
<td>( W_{\alpha}, \gamma )</td>
</tr>
<tr>
<td>RWKV-6</td>
<td>[ G_t = \alpha_t^T 1, \quad \alpha_t = \exp(-\exp(x_t W_{\alpha})) ]</td>
<td>( W_{\alpha} )</td>
</tr>
<tr>
<td>GLA (ours)</td>
<td>[ G_t = \alpha_t^T 1, \quad \alpha_t = \sigma(x_t W_{\alpha_1} W_{\alpha_2})^{\frac{1}{r}} ]</td>
<td>( W_{\alpha_1}, W_{\alpha_2} )</td>
</tr>
</tbody>
</table>

Gated linear attention \( \subset \) State-space models
Linear attention enables subquadratic, parallel training, and linear constant-memory inference. But suffers from poor performance and lack of hardware-efficient implementations.

This work:
- Hardware-efficient implementation of linear attention.
- Gated parameterization that closes the gap between linear attention and Transformers/Mamba.
- Connections between gated linear attention and state-space models.
Today: Efficient alternatives to attention in Transformers

Gated Linear Attention Transformers with Hardware-Efficient Training

Songlin Yang*, Bailin Wang*, Yikang Shen, Rameswar Panda, Yoon Kim
ICML ’24

Parallelizing Linear Transformers with the Delta Rule over Sequence Length

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, Yoon Kim
arXiv ’24
Deficiencies of Linear Attention / State-Space Models

Multi-Query Associative Recall Task

Input


[Example from: Arora et al. ’24]
Deficiencies of Linear Attention / State-Space Models

Multi-Query Associative Recall Task

Input


[Example from: Arora et al. ’24]
Deficiencies of Linear Attention / State-Space Models

Multi-Query Associative Recall Task

Input


Output

4, 6, 1, 2, 3

[Example from: Arora et al. '24]
Deficiencies of Linear Attention / State-Space Models

Multi-Query Associative Recall Task

Input

Output
4, 6, 1, 2, 3

Sequence Length: 512
Key-Value Pairs: 64

Accuracy

Model dimension

[Example from: Arora et al. ’24]
How can we improve associative recall?

DeltaNet [Schlag et al. ’21]: Use vector representations to retrieve and update memory ("Fast Weight Programmers").
How can we improve associative recall?

DeltaNet [Schlag et al. ’21]: Use vector representations to retrieve and update memory (“Fast Weight Programmers”).

Key, query, value vectors

\[ q_t, k_t, v_t = W_Q x_t, W_K x_t, W_V x_t \]

Retrieve old memory

\[ v^\text{old}_t = S_{t-1} k_t \]
How can we improve associative recall?

DeltaNet [Schlag et al. ’21]: Use vector representations to retrieve and update memory (“Fast Weight Programmers”).

Key, query, value vectors

\[ q_t, k_t, v_t = W_Q x_t, W_K x_t, W_V x_t \]

Retrieve old memory

\[ v_t^{\text{old}} = S_{t-1} k_t \]

Combine old memory with current value vector

\[ v_t^{\text{new}} = \beta_t v_t + (1 - \beta_t) v_t^{\text{old}} \]
How can we improve associative recall?

DeltaNet [Schlag et al. ’21]: Use vector representations to retrieve and update memory (“Fast Weight Programmers”).

Key, query, value vectors

\[ q_t, k_t, v_t = W_Q x_t, W_K x_t, W_V x_t \]

Retrieve old memory

\[ v_t^{\text{old}} = S_{t-1} k_t \]

Combine old memory with current value vector

\[ v_t^{\text{new}} = \beta_t v_t + (1 - \beta_t) v_t^{\text{old}} \]

\[ \beta_t = \sigma(W_\beta x_t) \in (0, 1) \]
How can we improve associative recall?

DeltaNet [Schlag et al. ’21]: Use vector representations to retrieve and update memory (“Fast Weight Programmers”).

Key, query, value vectors

\[ q_t, k_t, v_t = W_Q x_t, W_K x_t, W_V x_t \]

Retrieve old memory

\[ v_t^{\text{old}} = S_{t-1} k_t \]

Combine old memory with current value vector

\[ v_t^{\text{new}} = \beta_t v_t + (1 - \beta_t) v_t^{\text{old}} \]

Remove old memory, write new memory

\[ S_t = S_{t-1} \underbrace{- v_t^{\text{old}} k_t^T}_{\text{remove}} + \underbrace{v_t^{\text{new}} k_t^T}_{\text{write}} \]

Get output

\[ o_t = S_t q_t \]
DeltaNet [Schlag et al. ’21]
\[ q_t, k_t, v_t = W_Q x_t, W_K x_t, W_V x_t \]
\[ \beta_t = \sigma(W_\beta x_t) \in (0, 1) \]
$v_t^{\text{old}} = S_{t-1} k_t$

$q_t, k_t, v_t = W_Q x_t, W_K x_t, W_V x_t$

$\beta_t = \sigma(W_\beta x_t) \in (0, 1)$

DeltaNet [Schlag et al. '21]
\[ \mathbf{v}_t^{\text{new}} = \beta_t \mathbf{v}_t + (1 - \beta_t) \mathbf{v}_t^{\text{old}} \]

\[ \mathbf{v}_t^{\text{old}} = \mathbf{S}_{t-1} \mathbf{k}_t \]

\[ q_t, k_t, \mathbf{v}_t = W_Q x_t, W_K x_t, W_V x_t \]

\[ \beta_t = \sigma(\mathbf{W}_\beta x_t) \in (0, 1) \]
DeltaNet [Schlag et al. '21]

\[ S_t = S_{t-1} - v_t^{\text{old}} k_t^T + v_t^{\text{new}} k_t^T \]

- **remove** \( v_t^{\text{new}} \)
- **write** \( v_t^{\text{old}} \)

\[ v_t^{\text{new}} = \beta_t v_t + (1 - \beta_t) v_t^{\text{old}} \]

\[ v_t^{\text{old}} = S_{t-1} k_t \]

\[ q_t, k_t, v_t = W_Q x_t, W_K x_t, W_V x_t \]

\[ \beta_t = \sigma(W_{\beta} x_t) \in (0, 1) \]
\[ o_t = S_t q_t \]

\[ S_t = S_{t-1} - v_t^{\text{old}} k_t^T + v_t^{\text{new}} k_t^T \]

\[ v_t^{\text{new}} = \beta_t v_t + (1 - \beta_t) v_t^{\text{old}} \]

\[ v_t^{\text{old}} = S_{t-1} k_t \]

\[ q_t, k_t, v_t = W_Q x_t, W_K x_t, W_V x_t \]

\[ \beta_t = \sigma(W_\beta x_t) \in (0, 1) \]
DeltaNet Associative Recall Performance

Multi-Query Associative Recall Task

Sequence Length: 512, Key-Value Pairs: 64

Accuracy (%)

Model dimension

- DeltaNet
- Mamba
- GLA
- RetNet
- RWKV4
- Hyena
\[ v_t^{\text{old}} = S_{t-1} k_t \]
\[ v_t^{\text{new}} = \beta_t v_t + (1 - \beta_t) v_t^{\text{old}} \]

\[ u_t = v_t^{\text{new}} - v_t^{\text{old}} \]

\[ S_t = S_{t-1} - v_t^{\text{old}} k_t^T + v_t^{\text{new}} k_t^T \]

\[ S_t = S_{t-1} + u_t k_t^T \]
DeltaNet Issue

\[ \begin{align*}
    \mathbf{v}^{\text{old}}_t &= \mathbf{S}_{t-1} \mathbf{k}_t \\
    \mathbf{v}^{\text{new}}_t &= \beta_t \mathbf{v}_t + (1 - \beta_t) \mathbf{v}^{\text{old}}_t
\end{align*} \]

\[ \mathbf{u}_t = \mathbf{v}^{\text{new}}_t - \mathbf{v}^{\text{old}}_t \]

\[ \begin{align*}
    \mathbf{S}_t &= \mathbf{S}_{t-1} - \mathbf{v}^{\text{old}}_t \mathbf{k}_t^T + \mathbf{v}^{\text{new}}_t \mathbf{k}_t^T \\
    \text{remove} & \quad \text{write}
\end{align*} \]

\[ \mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{u}_t \mathbf{k}_t^T \]

\[ \mathbf{O} = (\mathbf{QK}^T \odot \mathbf{M}) \mathbf{U} \]

DeltaNet: Ordinary linear attention with “pseudo”-value vectors \( \mathbf{U} = [\mathbf{u}_1; \ldots; \mathbf{u}_L] \)
DeltaNet Issue

\[ \mathbf{v}_{t}^{\text{old}} = S_{t-1} k_t \]
\[ \mathbf{v}_{t}^{\text{new}} = \beta_t \mathbf{v}_t + (1 - \beta_t) \mathbf{v}_{t}^{\text{old}} \]
\[ \mathbf{u}_t = \mathbf{v}_{t}^{\text{new}} - \mathbf{v}_{t}^{\text{old}} \]
\[ \mathbf{S}_t = \mathbf{S}_{t-1} - \mathbf{v}_{t}^{\text{old}} k_t^T + \mathbf{v}_{t}^{\text{new}} k_t^T \]
\[ \mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{u}_t k_t^T \]

\[ \mathbf{O} = (\mathbf{Q} k^T \odot \mathbf{M}) \mathbf{U} \]

DeltaNet: Ordinary linear attention with "pseudo"-value vectors \( \mathbf{U} = [\mathbf{u}_1; \ldots; \mathbf{u}_L] \)

Unlike in linear attention, the pseudo value vector \( \mathbf{u}_t \) depends on the previous hidden state \( \mathbf{S}_{t-1} \). → Not scalable!
Parallelizing DeltaNet

\[ v_t^{\text{old}} = S_{t-1} k_t \]
\[ v_t^{\text{new}} = \beta_t v_t + (1 - \beta_t) v_t^{\text{old}} \]
\[ u_t = v_t^{\text{new}} - v_t^{\text{old}} \]

\[ S_t = S_{t-1} - v_t^{\text{old}} k_t^T + v_t^{\text{new}} k_t^T \]
\[ S_t = S_{t-1} + u_t k_t^T \]

\[ O = (Q K^T \odot M) U \]

DeltaNet: Ordinary linear attention with “pseudo”-value vectors \( U = [u_1; \ldots; u_L] \)

If there is an efficient way to compute \( U \), we would be good to go!
Parallelizing DeltaNet: A Simple Reparameterization

\[ S_t = S_{t-1} - \nu_t^{\text{old}} k_t^T + \nu_t^{\text{new}} k_t^T \]

\[ = S_{t-1} (I - \beta_t k_t k_t^T) + \beta_t \nu_t k_t^T \]
Parallelizing DeltaNet: A Simple Reparameterization

\[ S_t = S_{t-1} - \mathbf{v}^{\text{old}}_t \mathbf{k}_t^T + \mathbf{v}^{\text{new}}_t \mathbf{k}_t^T \]

\[ = S_{t-1} (\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^T) + \beta_t \mathbf{v}_t \mathbf{k}_t^T \]

\[ = \sum_{i=1}^{t} \beta_i (\mathbf{v}_i \mathbf{k}_i^T) \left( \prod_{j=i+1}^{t} (\mathbf{I} - \beta_j \mathbf{k}_j \mathbf{k}_j^T) \right) \]
Parallelizing DeltaNet: A Simple Reparameterization

\[ S_t = S_{t-1} - \nu_t^{\text{old}} k_t^T + \nu_t^{\text{new}} k_t^T \]

\[ = S_{t-1}(I - \beta_t k_t k_t^T) + \beta_t \nu_t k_t^T \]

\[ = \sum_{i=1}^{t} \beta_i (\nu_i k_i^T) \left( \prod_{j=i+1}^{t} (I - \beta_j k_j k_j^T) \right) \]

Product of generalized Householder matrices.
THE WY REPRESENTATION FOR PRODUCTS OF HOUSEHOLDER MATRICES*

CHRISTIAN BISCHOF† AND CHARLES VAN LOAN†

\[ P_i = I - \beta k_i k_i^T \quad \rightarrow \quad Z = \prod_{t=1}^{C} P_t = I + W Y^T, \quad W, Y \in \mathbb{R}^{d \times C} \]
Parallelizing DeltaNet: Memory-efficient Representation

**THE WY REPRESENTATION FOR PRODUCTS OF HOUSEHOLDER MATRICES**

CHRISTIAN BISCOFF† AND CHARLES VAN LOAN†

\[ P_i = I - \beta k_i k_i^T \quad \rightarrow \quad Z = \prod_{t=1}^{C} P_t = I + WY^T, \quad W, Y \in \mathbb{R}^{d \times C} \]

Chunkwise parallel algorithm still applies!
Parallelized DeltaNet: Speed

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Length</th>
<th>Speed-up (vs. recurrent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>2048</td>
<td>5.5x</td>
</tr>
<tr>
<td></td>
<td>4096</td>
<td>7.6x</td>
</tr>
<tr>
<td></td>
<td>8192</td>
<td>11.5x</td>
</tr>
<tr>
<td>128</td>
<td>2048</td>
<td>8.9x</td>
</tr>
<tr>
<td></td>
<td>4096</td>
<td>13.2x</td>
</tr>
<tr>
<td>256</td>
<td>2048</td>
<td>13.7x</td>
</tr>
</tbody>
</table>

On a single H100
## Parallelized DeltaNet: Performance

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL ↓</th>
<th>LM Eval ↑</th>
<th>Retrieval ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer++</td>
<td>16.9</td>
<td>50.9</td>
<td>41.8</td>
</tr>
<tr>
<td>RetNet</td>
<td>18.6</td>
<td>48.9</td>
<td>30.6</td>
</tr>
<tr>
<td>Mamba</td>
<td>17.1</td>
<td>50.0</td>
<td>27.6</td>
</tr>
<tr>
<td>Gated Linear Attention</td>
<td>17.2</td>
<td>51.1</td>
<td>37.7</td>
</tr>
<tr>
<td>DeltaNet</td>
<td>16.9</td>
<td>51.6</td>
<td>34.7</td>
</tr>
</tbody>
</table>

1.3B models trained on 100B tokens
Hybridizing DeltaNet

Hybrid 1: Sliding window attention every other layer
Hybridizing DeltaNet

Hybrid 1: Sliding window attention every other layer
Hybridizing DeltaNet

Hybrid 2: Global attention on the 2nd and middle layer
## Hybrid DeltaNet: Performance

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL ↓</th>
<th>LM Eval ↑</th>
<th>Retrieval ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer++</td>
<td>16.9</td>
<td>50.9</td>
<td>41.8</td>
</tr>
<tr>
<td>RetNet</td>
<td>18.6</td>
<td>48.9</td>
<td>30.6</td>
</tr>
<tr>
<td>Mamba</td>
<td>17.1</td>
<td>50.0</td>
<td>27.6</td>
</tr>
<tr>
<td>Gated Linear Attention</td>
<td>17.2</td>
<td>51.1</td>
<td>37.7</td>
</tr>
<tr>
<td>DeltaNet</td>
<td>16.9</td>
<td>51.6</td>
<td>34.7</td>
</tr>
<tr>
<td>Hybrid 1: DeltaNet + Sliding window attention</td>
<td>16.6</td>
<td>52.1</td>
<td>40.0</td>
</tr>
<tr>
<td>Hybrid 2: DeltaNet + Global attention on 2 layers</td>
<td>16.6</td>
<td>51.8</td>
<td>47.9</td>
</tr>
</tbody>
</table>

1.3B models trained on 100B tokens
Generalizing Gated Linear Attention / State-Space Models

Gated Linear Attention / State-Space Models

\[ S_t = S_{t-1} \odot G_t + v_t k_t^T \]  
Recurrence with elementwise product

\[ o_t = S_t q_t \]  
Memory read-out

\[ S_{t-1} \]  
\[ G_t \]
Generalizing Gated Linear Attention / State-Space Models

Gated Linear Attention / State-Space Models

\[ S_t = S_{t-1} \odot G_t + v_t k_t \]

\[ o_t = S_t q_t \]

Recurrence with elementwise product

Memory read-out

Multiplicative updates take \( O(d^2) \) and are therefore efficient, but does not allow for interactions across channels.
Generalizing Gated Linear Attention / State-Space Models

Generalized Linear Transformers

\[
S_t = S_{t-1}G_t + v_t k_t^T
\]

\[
o_t = S_t q_t
\]

Recurrence with matmul

Memory read-out
Generalizing Gated Linear Attention / State-Space Models

Generalized Linear Transformers

\[ S_t = S_{t-1} G_t + v_t k_t^T \]
\[ o_t = S_t q_t \]

Matmul-based updates can model interactions across channels, but take \( O(d^3) \) and are thus too expensive.

Recurrence with matmul
Memory read-out
Generalizing Gated Linear Attention / State-Space Models

Generalized Linear Transformers with **Structured Matmuls**

\[ S_t = S_{t-1}(I - a_t b_t^T) + v_t k_t^T \]

Recurrence with identity + low-rank

\[ o_t = S_t q_t \]

Memory read-out
Generalizing Gated Linear Attention / State-Space Models

Generalized Linear Transformers with **Structured Matmuls**

\[
S_t = S_{t-1}(I - a_t b_t^T) + v_t k_t^T
\]

\[
\mathbf{o}_t = S_t q_t
\]

Recurrence with identity + low-rank Memory read-out

Can model interactions across channels in \(O(kd^2)\)! DeltaNet uses

\[
S = S_{t-1}(I - \beta_t k_t k_t^T) + \beta_t v_t k_t^T
\]

and is thus a special case.
Open/Future Work

What about more general associative operators?

\[ S_t = S_{t-1} \cdot M_t + \nu_t k_t^T \]
Summary

Linear attention and SSMs have trouble with recall-oriented tasks.

DeltaNet operationalizes a key-value retrieval/update mechanism, but unclear how to parallelize for efficient training.

This work:
- Recasts DeltaNet as linear attention with “pseudo”-value vectors ⇒ the chunkwise algorithm from GLA still applies!
- DeltaNet outperforms GLA/Mamba.
- Hybrid DeltaNet outperforms Transformers.
Thanks!