Linear Transformers for Efficient Sequence Modeling

Songlin Yang
MIT CSAIL

August, 2024

Today: Efficient alternatives to attention in Transformers

Gated Linear Attention Transformers with Hardware-Efficient Training

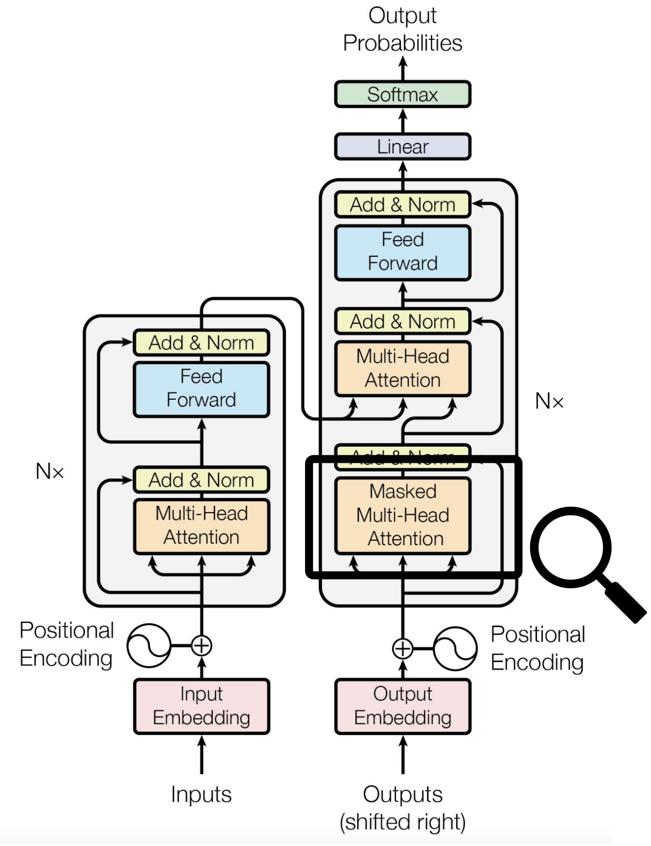
Songlin Yang*, Bailin Wang*, Yikang Shen,Rameswar Panda, Yoon Kim
ICML ’24

Parallelizing Linear Transformers with the Delta Rule over Sequence Length

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, Yoon Kim
arXiv ’24

Background

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

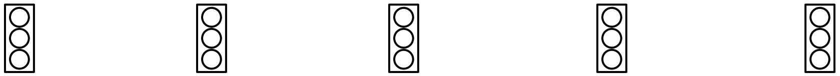
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Attention: Training

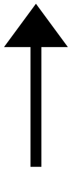
L : sequence length

d : hidden state dimension

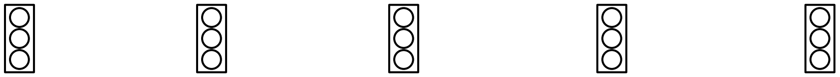
$\mathbf{O} \in \mathbb{R}^{L \times d}$



$$\mathbf{O} = \mathrm{SelfAttention}(\mathbf{X})$$



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



Attention: Training

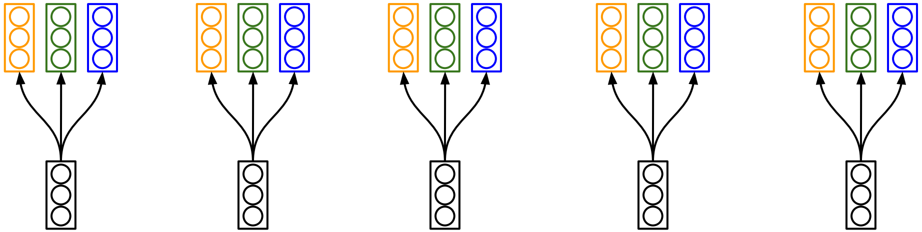
L : sequence length

d : hidden state dimension

$O(Ld^2)$ $\quad$ $\mathbf{Q},\mathbf{K},\mathbf{V} = \mathbf{X}\boldsymbol{W}_Q,\mathbf{X}\boldsymbol{W}_K,\mathbf{X}\boldsymbol{W}_V$

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

Key K
Value V
Query Q



Attention: Training

L : sequence length

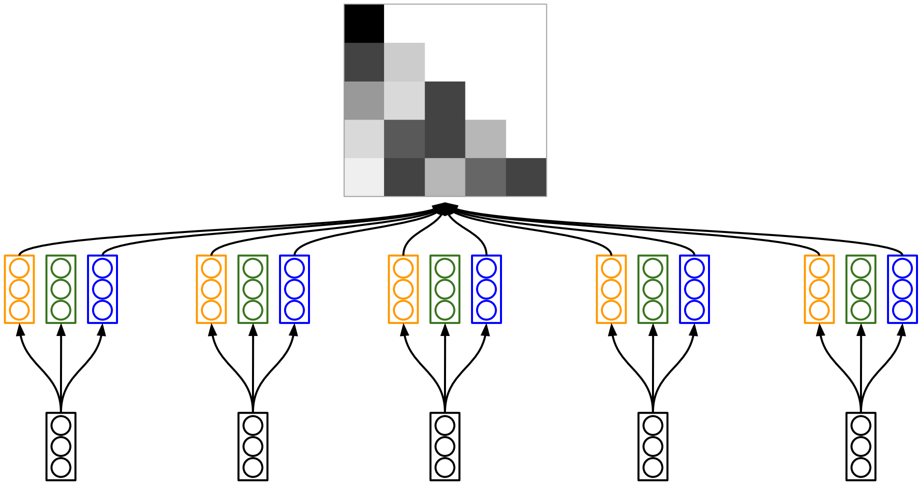
d : hidden state dimension

$O(L^2 d)$ $\quad \mathbf{A} = \text{softmax}(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M}) \in \mathbb{R}^{L \times L}$

$O(Ld^2)$ $\quad \mathbf{Q}, \mathbf{K}, \mathbf{V} = \mathbf{X}\boldsymbol{W}_Q, \mathbf{X}\boldsymbol{W}_K, \mathbf{X}\boldsymbol{W}_V$

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

Key K
Value V
Query Q



Attention: Training

L : sequence length

d : hidden state dimension

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

$O(L^2 d)$ $\quad$ $\mathbf{A} = \mathrm{softmax}(\mathbf{QK}^\top \odot \mathbf{M}) \in \mathbb{R}^{L \times L}$

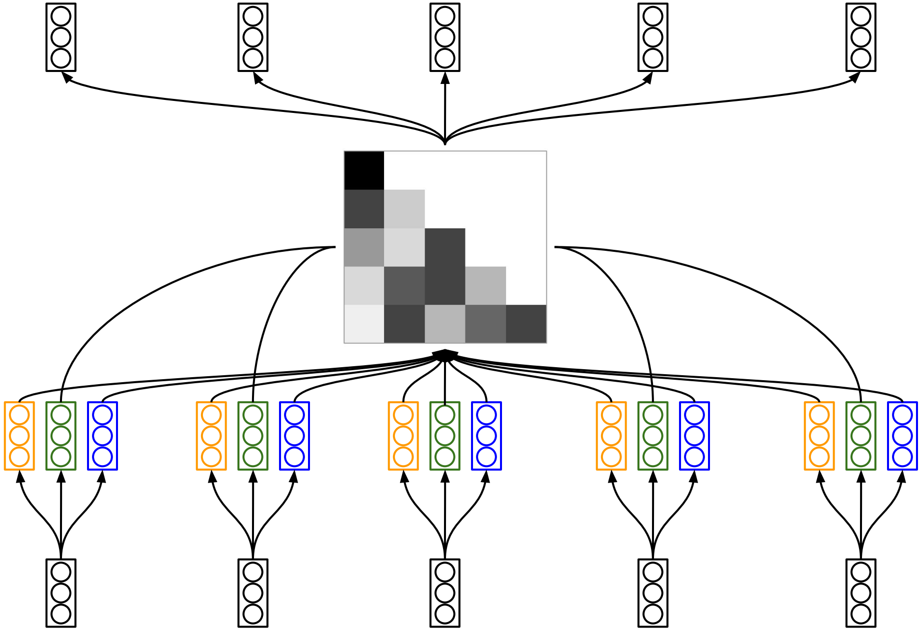
$O(Ld^2)$ $\quad$ $\mathbf{Q},\mathbf{K},\mathbf{V} = \mathbf{X}\boldsymbol{W}_Q, \mathbf{X}\boldsymbol{W}_K, \mathbf{X}\boldsymbol{W}_V$

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

Key K

Value V

Query Q



Attention: Training

L : sequence length

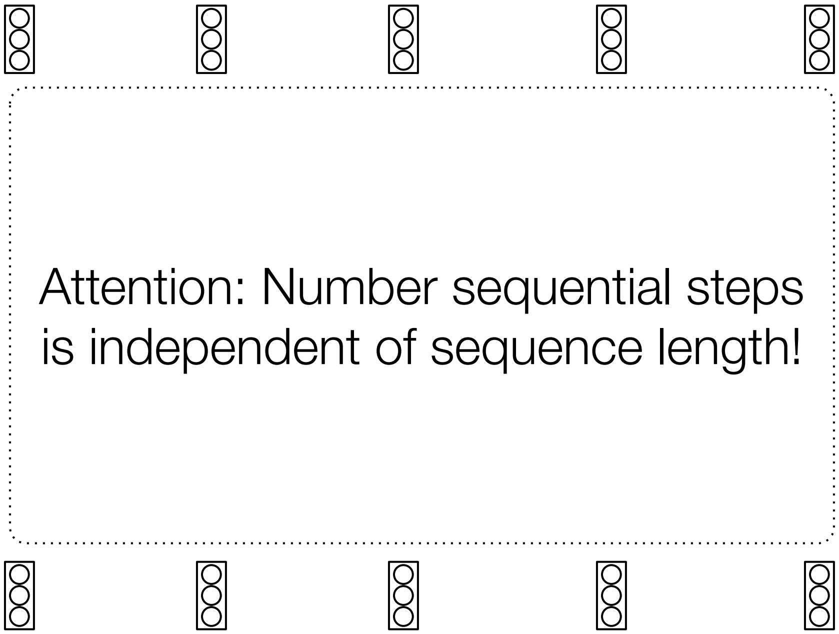
d : hidden state dimension

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

$O(L^2 d)$ $\quad \mathbf{A} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M}) \in \mathbb{R}^{L \times L}$

$O(L d^2)$ $\quad \mathbf{Q}, \mathbf{K}, \mathbf{V} = \mathbf{X}\boldsymbol{W}_Q, \mathbf{X}\boldsymbol{W}_K, \mathbf{X}\boldsymbol{W}_V$

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



Attention: Training

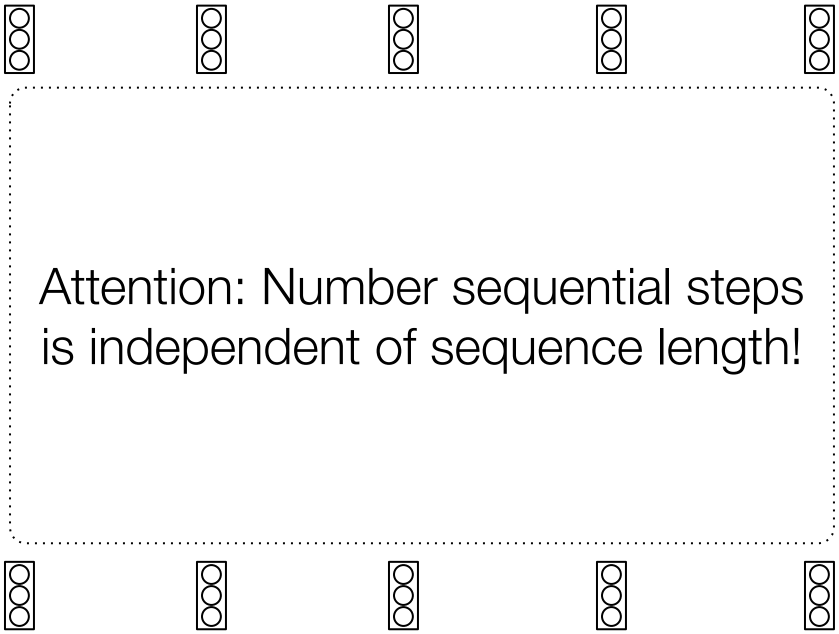
Attention requires $O(L^2 d + L d^2)$ work but can be done in $O(1)$ steps
→ Parallel training that is rich in matmuls.

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

$O(L^2 d)$ $\quad \mathbf{A} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M}) \in \mathbb{R}^{L \times L}$

$O(L d^2)$ $\quad \mathbf{Q}, \mathbf{K}, \mathbf{V} = \mathbf{X}\boldsymbol{W}_Q, \mathbf{X}\boldsymbol{W}_K, \mathbf{X}\boldsymbol{W}_V$

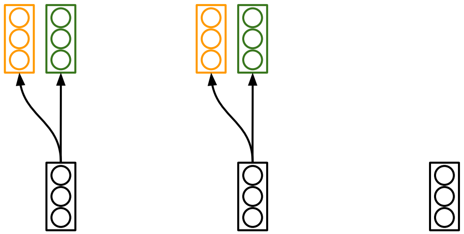
$\mathbf{X} \in \mathbb{R}^{L \times d}$



Attention: Generative Inference

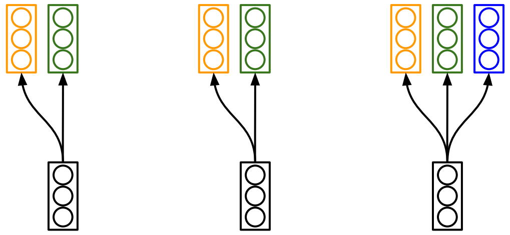
$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{x}_t\boldsymbol{W}_Q,\ \boldsymbol{x}_t\boldsymbol{W}_K,\ \boldsymbol{x}_t\boldsymbol{W}_V$$

Key K
Value V
Query Q



Attention: Generative Inference

$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{x}_t\boldsymbol{W}_Q,\ \boldsymbol{x}_t\boldsymbol{W}_K,\ \boldsymbol{x}_t\boldsymbol{W}_V$$



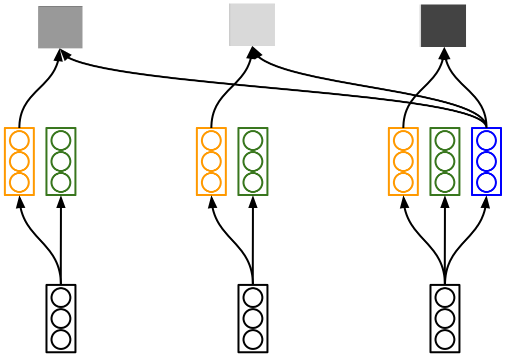
Key K
Value V
Query Q

Attention: Generative Inference

$$\frac{\exp(\boldsymbol{q}_t^\top \boldsymbol{k}_j)}{\sum_{l=1}^{t} \exp(\boldsymbol{q}_t^\top \boldsymbol{k}_l)}$$

$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{x}_t \boldsymbol{W}_Q,\ \boldsymbol{x}_t \boldsymbol{W}_K,\ \boldsymbol{x}_t \boldsymbol{W}_V$$

Key K
Value V
Query Q

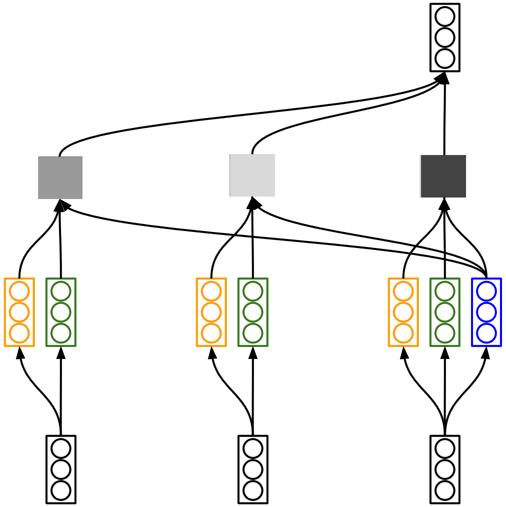


Attention: Generative Inference

$$\boldsymbol{o}_t = \sum_{j=1}^{t} \frac{\exp(\boldsymbol{q}_t^\top \boldsymbol{k}_j)}{\sum_{l=1}^{t} \exp(\boldsymbol{q}_t^\top \boldsymbol{k}_l)} \boldsymbol{v}_j$$

$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{x}_t \boldsymbol{W}_Q,\ \boldsymbol{x}_t \boldsymbol{W}_K,\ \boldsymbol{x}_t \boldsymbol{W}_V$$

Key K
Value V
Query Q

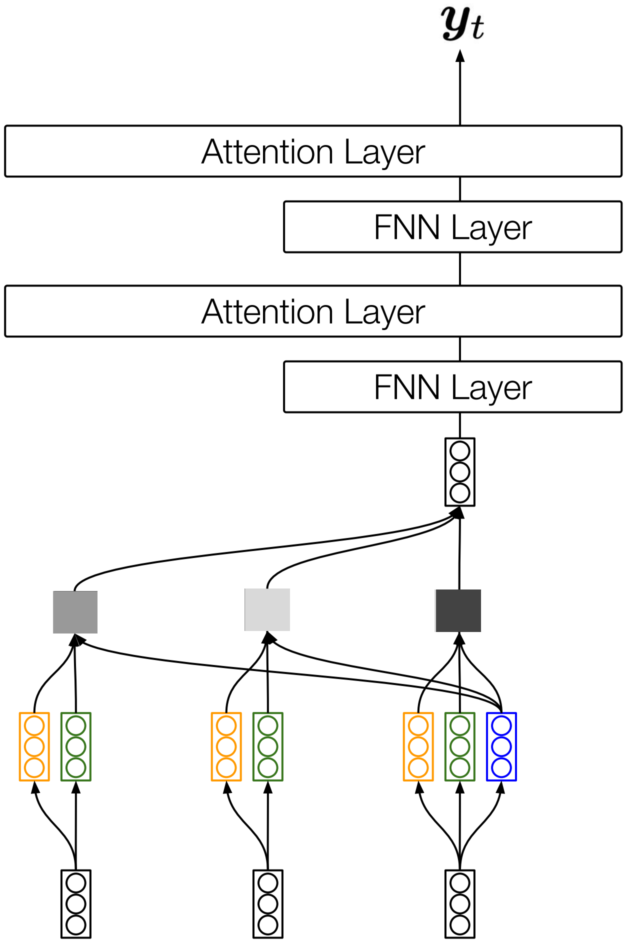


Attention: Generative Inference

$$\boldsymbol{o}_t = \sum_{j=1}^{t} \frac{\exp(\boldsymbol{q}_t^\top \boldsymbol{k}_j)}{\sum_{l=1}^{t} \exp(\boldsymbol{q}_t^\top \boldsymbol{k}_l)} \boldsymbol{v}_j$$

$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{x}_t \boldsymbol{W}_Q,\ \boldsymbol{x}_t \boldsymbol{W}_K,\ \boldsymbol{x}_t \boldsymbol{W}_V$$

Key K
Value V
Query Q

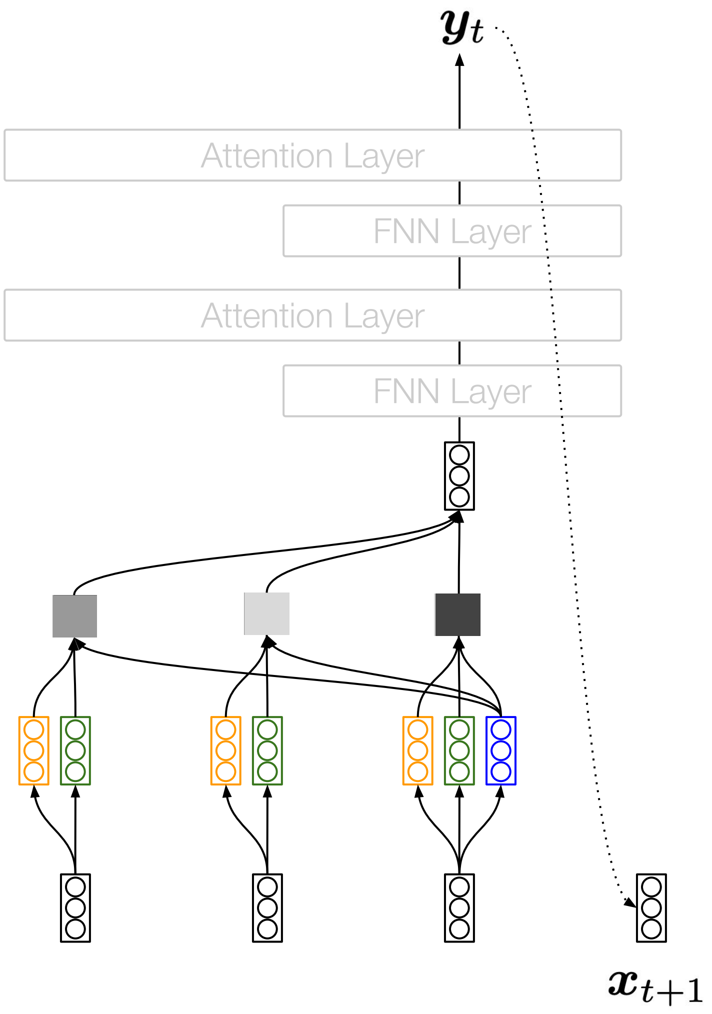


Attention: Generative Inference

$$\boldsymbol{o}_t = \sum_{j=1}^{t} \frac{\exp(\boldsymbol{q}_t^\top \boldsymbol{k}_j)}{\sum_{l=1}^{t} \exp(\boldsymbol{q}_t^\top \boldsymbol{k}_l)} \boldsymbol{v}_j$$

$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{x}_t \boldsymbol{W}_Q,\ \boldsymbol{x}_t \boldsymbol{W}_K,\ \boldsymbol{x}_t \boldsymbol{W}_V$$

Key K
Value V
Query Q

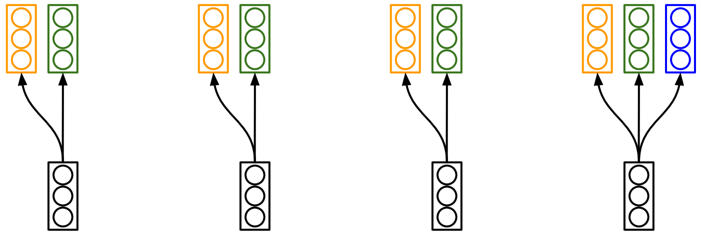


Attention: Generative Inference

$$\boldsymbol{o}_t = \sum_{j=1}^{t} \frac{\exp(\boldsymbol{q}_t^\top \boldsymbol{k}_j)}{\sum_{l=1}^{t} \exp(\boldsymbol{q}_t^\top \boldsymbol{k}_l)} \boldsymbol{v}_j$$

$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{x}_t \boldsymbol{W}_Q,\ \boldsymbol{x}_t \boldsymbol{W}_K,\ \boldsymbol{x}_t \boldsymbol{W}_V$$

Key	K
Value	V
Query	Q

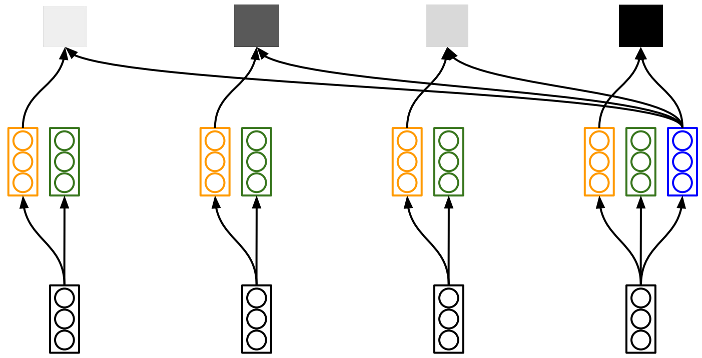


Attention: Generative Inference

$$\boldsymbol{o}_t = \sum_{j=1}^{t} \frac{\exp(\boldsymbol{q}_t^\top \boldsymbol{k}_j)}{\sum_{l=1}^{t} \exp(\boldsymbol{q}_t^\top \boldsymbol{k}_l)} \boldsymbol{v}_j$$

$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{x}_t \boldsymbol{W}_Q,\ \boldsymbol{x}_t \boldsymbol{W}_K,\ \boldsymbol{x}_t \boldsymbol{W}_V$$

Key K
Value V
Query Q



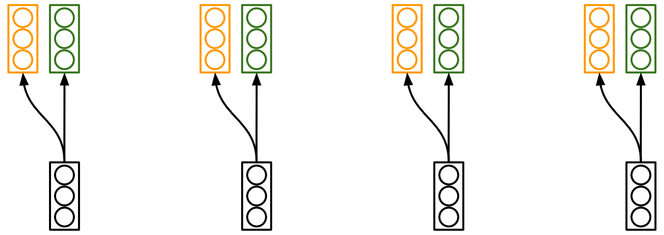
Attention: Generative Inference

$$\boldsymbol{o}_t = \sum_{j=1}^{t} \frac{\exp(\boldsymbol{q}_t^\top \boldsymbol{k}_j)}{\sum_{l=1}^{t} \exp(\boldsymbol{q}_t^\top \boldsymbol{k}_l)} \boldsymbol{v}_j$$

Need to keep around "KV-cache" that takes $O(L)$ memory.

$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{x}_t \boldsymbol{W}_Q,\ \boldsymbol{x}_t \boldsymbol{W}_K,\ \boldsymbol{x}_t \boldsymbol{W}_V$$

Key K
Value V
Query Q



Attention

	Training ("Parallel Form")	Inference ("Recurrent Form")
	$\mathbf{O} = \mathrm{softmax}\big((\mathbf{Q}\mathbf{K}^{\top}) \odot \mathbf{M}\big)\mathbf{V}$	$\boldsymbol{o}_t = \frac{\sum_{i=1}^{t} \exp(\boldsymbol{q}_t \boldsymbol{k}_i^{\top})\boldsymbol{v}_i}{\sum_{i=1}^{t} \exp(\boldsymbol{q}_t \boldsymbol{k}_i^{\top})}$
Compute	$O(L^2)$	$O(L^2)$
Memory	$O(L)$	$O(L)$
Steps	$O(1)$	$O(L)$

Attention

	Training ("Parallel Form")	Inference ("Recurrent Form")
	$\mathbf{O} = \mathrm{softmax}\left((\mathbf{Q}\mathbf{K}^\top) \odot \mathbf{M}\right)\mathbf{V}$	$\boldsymbol{o}_t = \frac{\sum_{i=1}^{t} \exp(\boldsymbol{q}_t \boldsymbol{k}_i^\top)\boldsymbol{v}_i}{\sum_{i=1}^{t} \exp(\boldsymbol{q}_t \boldsymbol{k}_i^\top)}$
Compute	$O(L^2)$ 😐	$O(L^2)$ 😐
Memory	$O(L)$ ☺	$O(L)$ ☹
Steps	$O(1)$ ☺	$O(L)$

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

$$\mathbf{O} = \cancel{\mathrm{softmax}}\big((\mathbf{Q}\mathbf{K}^\intercal) \odot \mathbf{M}\big)\mathbf{V}$$

(Simple) Linear Attention

$$\mathbf{O} = \big((\mathbf{Q}\mathbf{K}^\intercal) \odot \mathbf{M}\big)\mathbf{V}$$

From Softmax to Linear Attention [Katharopoulos et al. '20]

Softmax Attention

$$\mathbf{O} = \cancel{\mathrm{softmax}}\big((\mathbf{Q}\mathbf{K}^\top) \odot \mathbf{M}\big)\mathbf{V}$$

$\mathbf{M} \rightarrow \{-\infty, 0\}^{L \times L}$

(Simple) Linear Attention

$$\mathbf{O} = \big((\mathbf{Q}\mathbf{K}^\top) \odot \mathbf{M}\big)\mathbf{V}$$

$\mathbf{M} \rightarrow \{0, 1\}^{L \times L}$

From Softmax to Linear Attention [Katharopoulos et al. '20]

	Training ("Parallel Form")	Inference ("Recurrent Form")
Softmax Attention	$\mathbf{O} = \mathrm{softmax}\left((\mathbf{Q}\mathbf{K}^{\intercal}) \odot \mathbf{M}\right)\mathbf{V}$	$\boldsymbol{o}_t = \sum_{j=1}^{t} \frac{\exp(\boldsymbol{q}_t^{\intercal}\boldsymbol{k}_j)}{\sum_{l=1}^{t}\exp(\boldsymbol{q}_t^{\intercal}\boldsymbol{k}_l)}\boldsymbol{v}_j$
(Simple) Linear Attention	$\mathbf{O} = \left((\mathbf{Q}\mathbf{K}^{\intercal}) \odot \mathbf{M}\right)\mathbf{V}$	$\boldsymbol{o}_t = \sum_{j=1}^{t}(\boldsymbol{q}_t^{\intercal}\boldsymbol{k}_j)\boldsymbol{v}_j$

Linear Attention: Inference

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

Linear Attention: Inference

$$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^\top \boldsymbol{k}_j) \boldsymbol{v}_j = \boldsymbol{q}_t^\top \left(\sum_{j=1}^{t} \boldsymbol{k}_j \boldsymbol{v}_j^\top \right)$$

Linear Attention: Inference

$$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^{\top} \boldsymbol{k}_j) \boldsymbol{v}_j = \boldsymbol{q}_t^{\top} \underbrace{\left(\sum_{j=1}^{t} \boldsymbol{k}_j \boldsymbol{v}_j^{\top} \right)}_{\mathbf{S}_t \in \mathbb{R}^{d \times d}}$$

Linear Attention: Inference

$$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^\top \boldsymbol{k}_j) \boldsymbol{v}_j = \boldsymbol{q}_t^\top \underbrace{\left(\sum_{j=1}^{t} \boldsymbol{k}_j \boldsymbol{v}_j^\top \right)}_{\mathbf{S}_t \in \mathbb{R}^{d \times d}}$$

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

$$\boldsymbol{o}_t = \boldsymbol{q}_t^\top \mathbf{S}_t$$

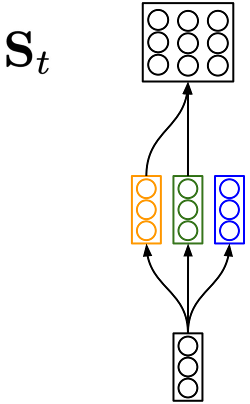
Linear Attention: Inference

Key K
Value V
Query Q

$$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^\top \boldsymbol{k}_j)\boldsymbol{v}_j = \boldsymbol{q}_t^\top \underbrace{\left(\sum_{j=1}^{t} \boldsymbol{k}_j \boldsymbol{v}_j^\top \right)}_{\mathbf{S}_t \in \mathbb{R}^{d \times d}}$$

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

$$\boldsymbol{o}_t = \boldsymbol{q}_t^\top \mathbf{S}_t$$



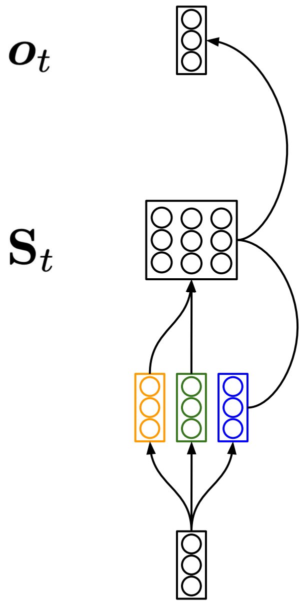
Linear Attention: Inference

Key K
Value V
Query Q

$$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^\top \boldsymbol{k}_j)\boldsymbol{v}_j = \boldsymbol{q}_t^\top \underbrace{\left(\sum_{j=1}^{t} \boldsymbol{k}_j \boldsymbol{v}_j^\top\right)}_{\mathbf{S}_t \in \mathbb{R}^{d \times d}}$$

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

$$\boldsymbol{o}_t = \boldsymbol{q}_t^\top \mathbf{S}_t$$



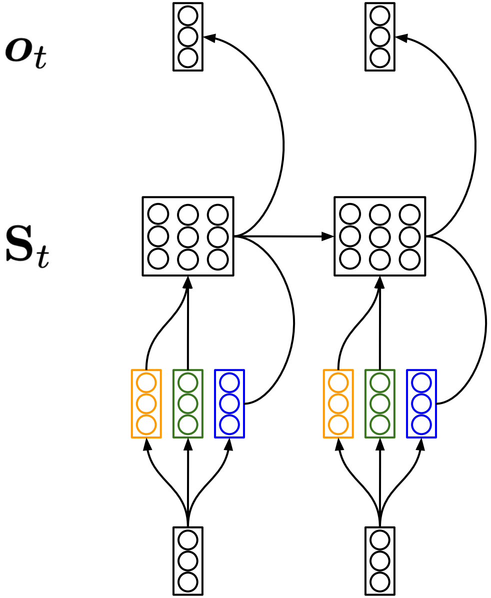
Linear Attention: Inference

Key K
Value V
Query Q

$$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^\top \boldsymbol{k}_j) \boldsymbol{v}_j = \boldsymbol{q}_t^\top \underbrace{\left(\sum_{j=1}^{t} \boldsymbol{k}_j \boldsymbol{v}_j^\top \right)}_{\mathbf{S}_t \in \mathbb{R}^{d \times d}}$$

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

$$\boldsymbol{o}_t = \boldsymbol{q}_t^\top \mathbf{S}_t$$



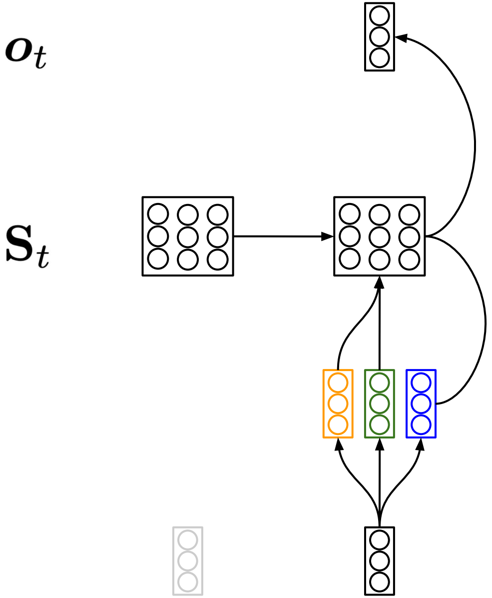
Linear Attention: Inference

Key K
Value V
Query Q

$$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^\top \boldsymbol{k}_j)\boldsymbol{v}_j = \boldsymbol{q}_t^\top \underbrace{\left(\sum_{j=1}^{t} \boldsymbol{k}_j \boldsymbol{v}_j^\top \right)}_{\mathbf{S}_t \in \mathbb{R}^{d \times d}}$$

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

$$\boldsymbol{o}_t = \boldsymbol{q}_t^\top \mathbf{S}_t$$



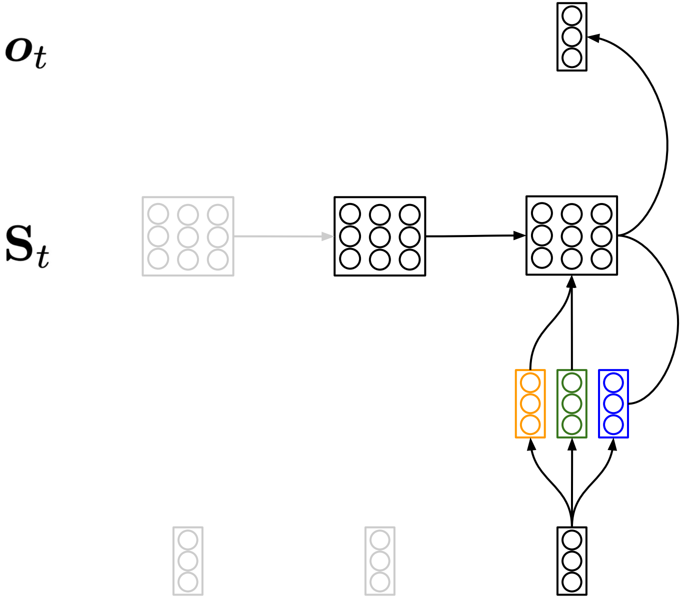
Linear Attention: Inference

Key K
Value V
Query Q

$$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^{\mathsf{T}} \boldsymbol{k}_j) \boldsymbol{v}_j = \boldsymbol{q}_t^{\mathsf{T}} \underbrace{\left(\sum_{j=1}^{t} \boldsymbol{k}_j \boldsymbol{v}_j^{\mathsf{T}} \right)}_{\mathbf{S}_t \in \mathbb{R}^{d \times d}}$$

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

$$\boldsymbol{o}_t = \boldsymbol{q}_t^{\mathsf{T}} \mathbf{S}_t$$



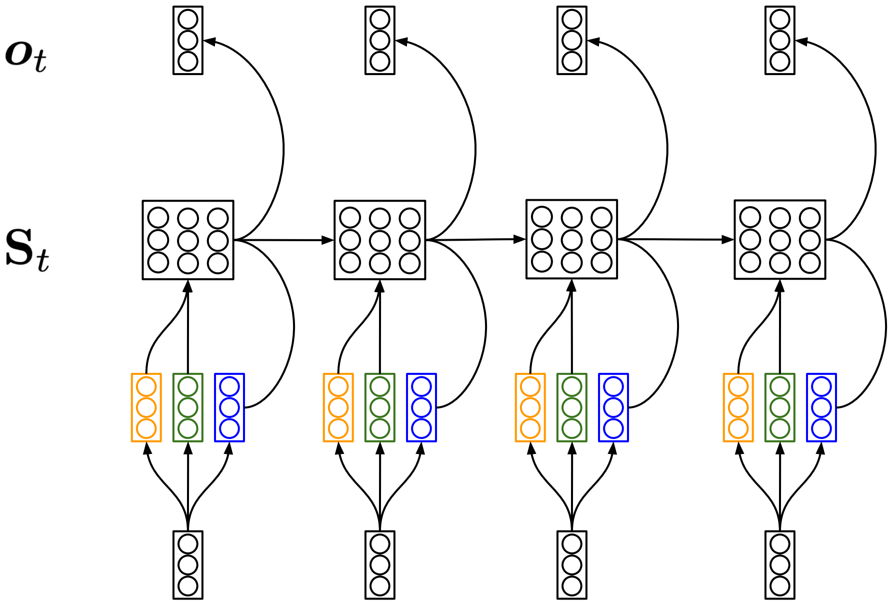
Linear Attention: Inference

Key K
Value V
Query Q

$$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^{\top} \boldsymbol{k}_j) \boldsymbol{v}_j = \boldsymbol{q}_t^{\top} \underbrace{\left(\sum_{j=1}^{t} \boldsymbol{k}_j \boldsymbol{v}_j^{\top} \right)}_{\mathbf{S}_t \in \mathbb{R}^{d \times d}}$$

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

$$\boldsymbol{o}_t = \boldsymbol{q}_t^{\top} \mathbf{S}_t$$



Linear Attention = Linear RNNs with matrix-valued hidden states

→ Constant-memory inference!

Linear Attention

	Training ("Parallel Form")	Inference ("Recurrent Form")
	$\mathbf{O} = \big((\mathbf{Q}\mathbf{K}^{\top}) \odot \mathbf{M}\big)\mathbf{V}$	$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^{\top} \boldsymbol{k}_j) \boldsymbol{v}_j$
Compute	$O(L^2)$	$O(L)$
Memory	$O(L)$	$O(1)$
Steps	$O(1)$	$O(L)$

Linear Attention: Naive Parallel Form

	Training ("Parallel Form")	Inference ("Recurrent Form")
	$\mathbf{O} = \left((\mathbf{Q}\mathbf{K}^\intercal) \odot \mathbf{M}\right)\mathbf{V}$	$\boldsymbol{o}_t = \sum_{j=1}^{t} (\boldsymbol{q}_t^\intercal \boldsymbol{k}_j)\boldsymbol{v}_j$
Compute	$O(L^2)$ 😐	$O(L)$ ☺
Memory	$O(L)$ ☺	$O(1)$ ☺
Steps	$O(1)$ ☺	$O(L)$

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: Why don't use recurrent form for training?

Recurrent form is slow in training

☹ Strict sequential computation, lacking sequence parallelism.

- All operations are either elementwise addition/multiplication or reduction, lacking matmul ops -> cannot leverage tensor cores.
- Requires materialization of each time step's hidden states
  - High I/O cost due to large hidden state size

Linear Attention: Why don't use recurrent form for training?

Recurrent form is slow in training

- Strict sequential computation, lacking sequence parallelism.
- ☹ All operations are either elementwise addition/multiplication or reduction, lacking matmul ops -> cannot leverage tensor cores.
- Requires materialization of each time step's hidden states
  - High I/O cost due to large hidden state size

Linear Attention: Why don't use recurrent form for training?

Recurrent form is slow in training

- Strict sequential computation, lacking sequence parallelism.
- All operations are either elementwise addition/multiplication or reduction, lacking matmul operations -> cannot leverage tensor cores.

☹ Requires materialization of each time step's hidden states
  - High I/O cost due to large hidden state size

Linear Attention: Why don't use recurrent form for training?

Why don't use parallel scan?

☺ ~~Strict sequential computation, lacking sequence parallelism.~~

☹ All operations are either elementwise addition/multiplication or reduction, lacking matmul operations -> cannot leverage tensor cores.

☹ Requires materialization of each time step's hidden states

Linear Attention: Why don't use recurrent form for training?

Why don't use parallel scan?

- ~~Strict sequential computation, lacking sequence parallelism.~~
- All operations are either elementwise addition/multiplication or reduction, lacking matmul operations -> cannot leverage tensor cores.
- Requires materialization of each time step's hidden states
  ☺ Mamba1 reduces I/O costs by keeping all hidden states in SRAM

Linear Attention: Why don't use recurrent form for training?

Why don't use parallel scan?

- ~~Strict sequential computation, lacking sequence parallelism.~~
- All operations are either elementwise addition/multiplication or reduction, lacking matmul operations -> cannot leverage tensor cores.
- Requires materialization of each time step's hidden states
  - 😐 Mamba1 reduces I/O costs by keeping all hidden states in SRAM
    - Due to the limited SRAM size, it is hard to scale up state size

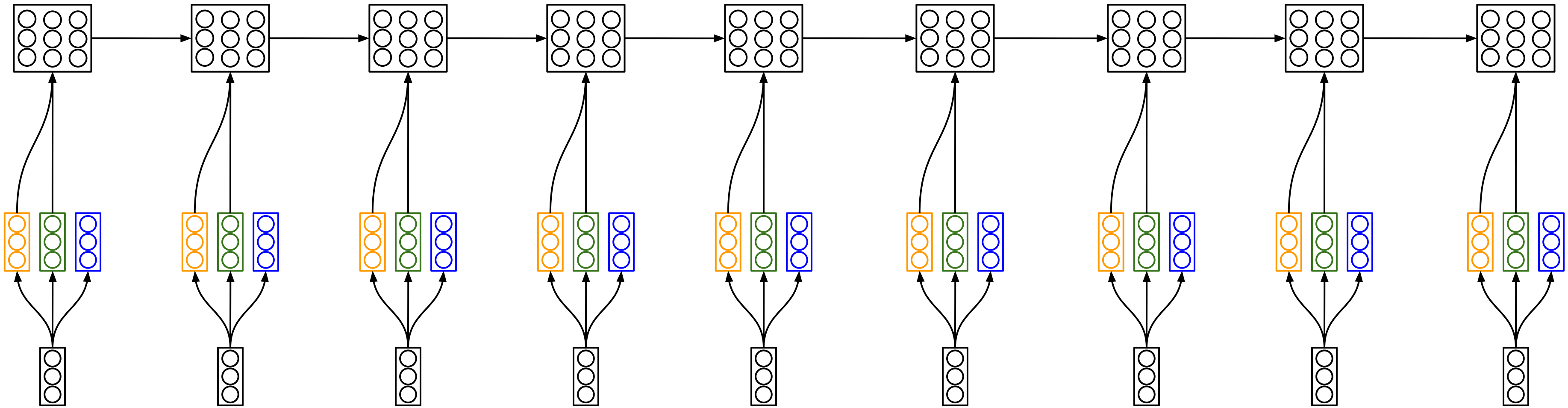
Linear Attention: Why don't use recurrent form for training?

Why don't use parallel scan?

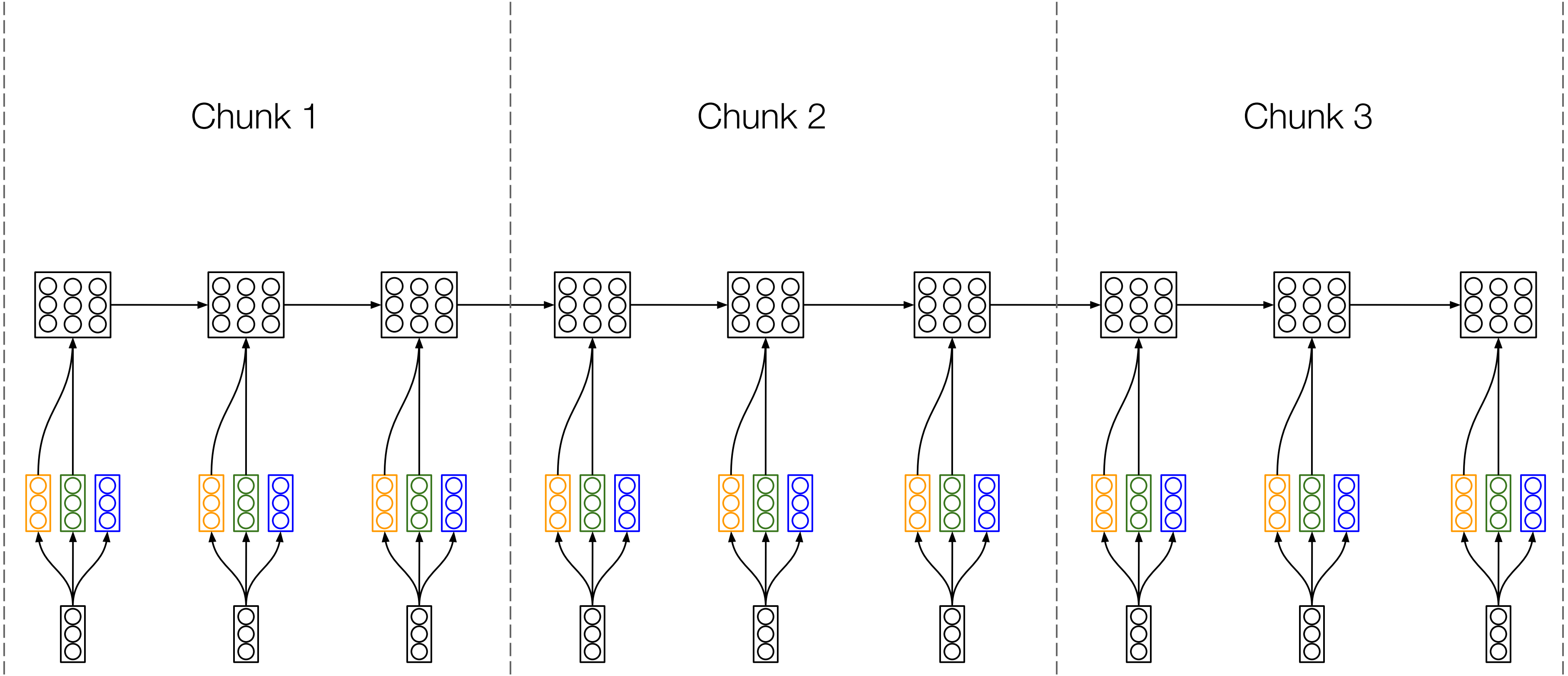
- ~~Strict sequential computation, lacking sequence parallelism.~~
- All operations are either elementwise addition/multiplication or reduction, lacking matmul operations -> cannot leverage tensor cores.
- Requires materialization of each time step's hidden states
  - 😐 Mamba1 reduces I/O costs by keeping all hidden states in SRAM
    - Due to the limited SRAM size, it is hard to scale up state size
      - **State expansion is important for RNNs**
        - *Mamba2, HGRN2, RWKV5/6, etc*

Linear Attention: "Chunkwise Parallel Form" [Hua et al. '22, Sun et al. '23]

Pure RNN ·

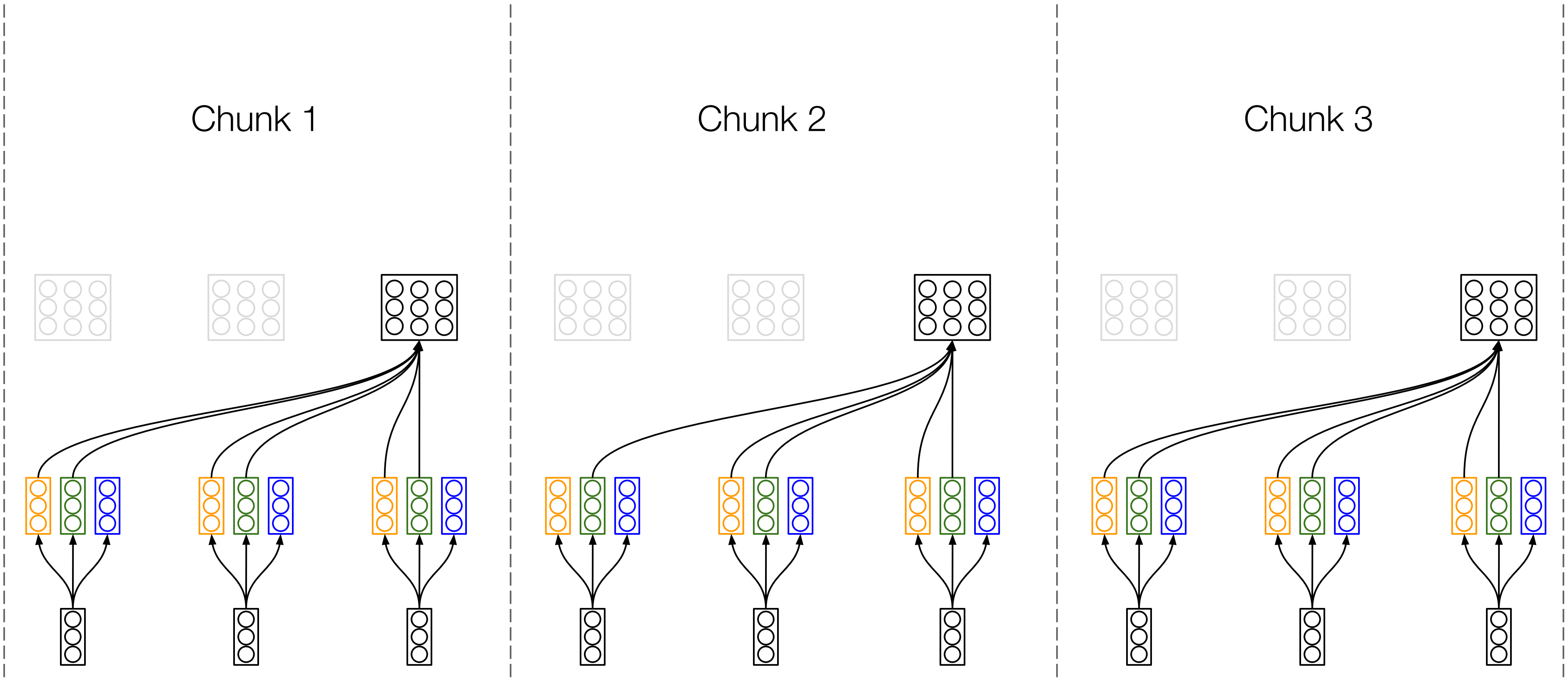


Linear Attention: "Chunkwise Parallel Form" [Hua et al. '22, Sun et al. '23]



Linear Attention: "Chunkwise Parallel Form" [Hua et al. '22, Sun et al. '23]

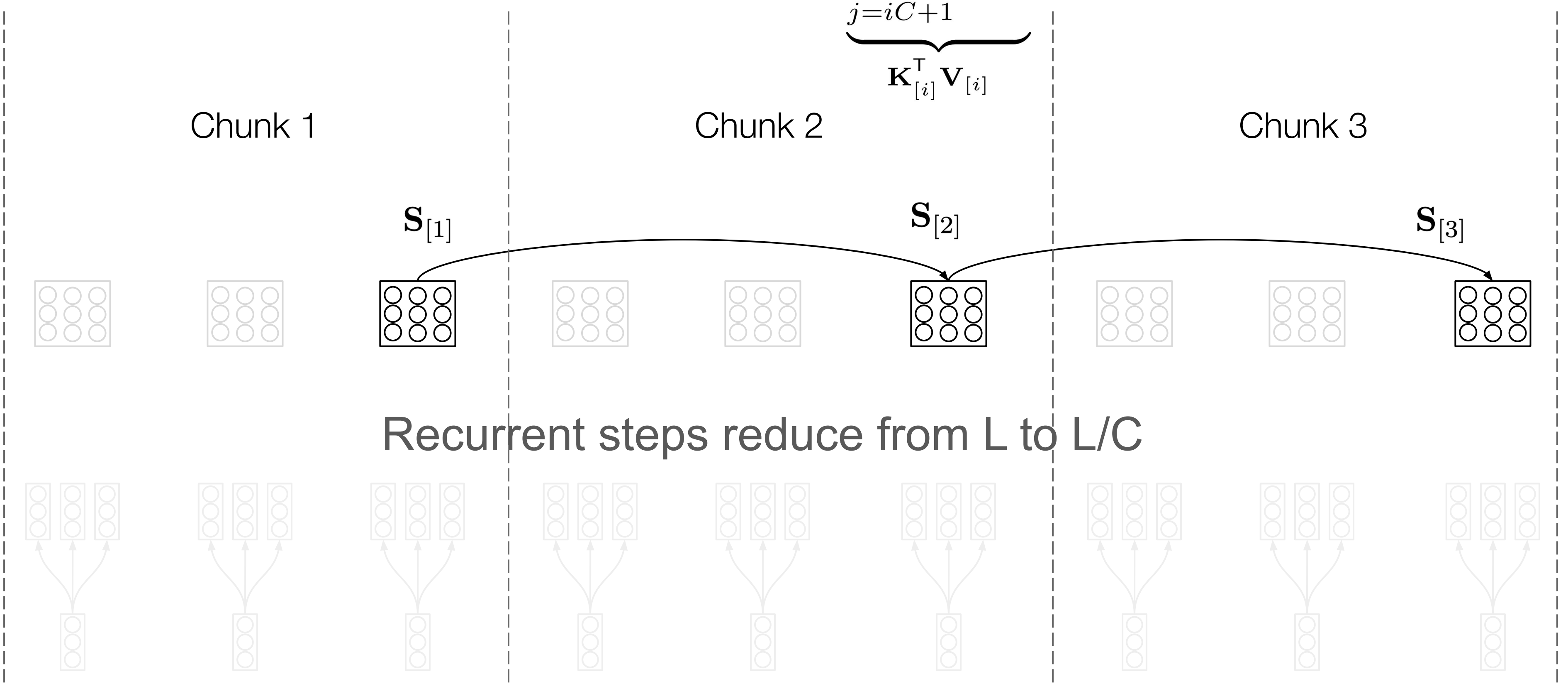
First step: local state computation $\mathbf{K}_{[i]}^{\top}\mathbf{V}_{[i]}$



Linear Attention: "Chunkwise Parallel Form" [Hua et al. '22, Sun et al. '23]

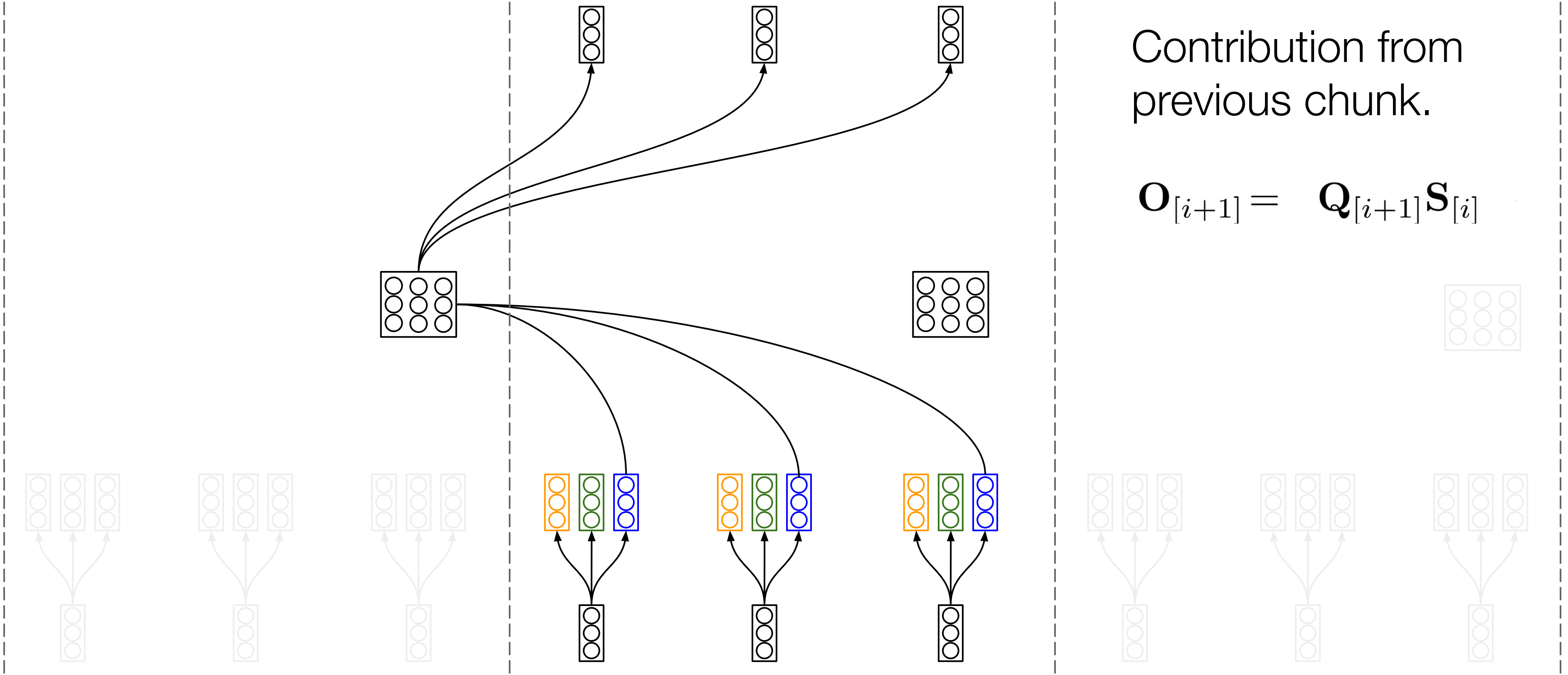
Second step: state passing

$$\mathbf{S}_{[i+1]} = \mathbf{S}_{[i]} + \underbrace{\sum_{j=iC+1}^{(i+1)C} \boldsymbol{k}_j^\top \boldsymbol{v}_j}_{\mathbf{K}_{[i]}^\top \mathbf{V}_{[i]}}$$



Linear Attention: "Chunkwise Parallel Form" [Hua et al. '22, Sun et al. '23]

Third step: output computation



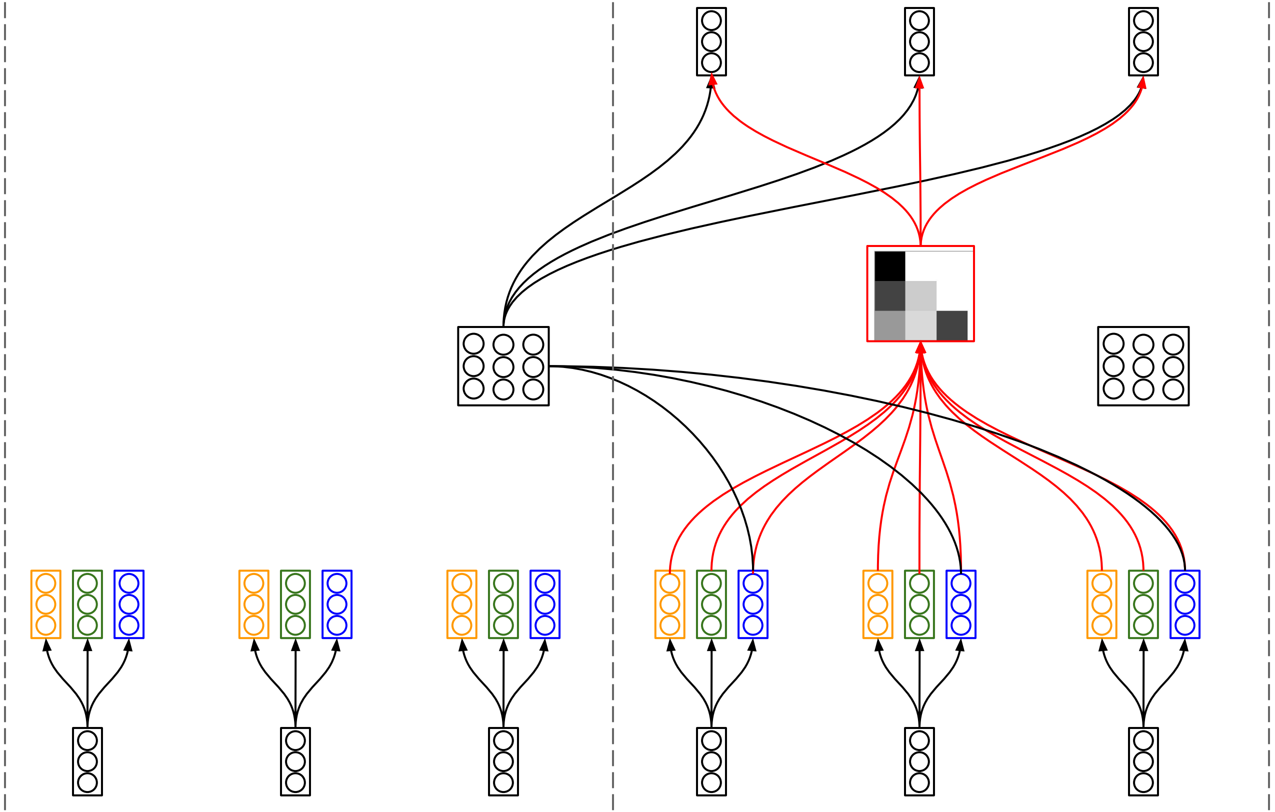
Contribution from previous chunk.

$$\mathbf{O}_{[i+1]} = \quad \mathbf{Q}_{[i+1]}\mathbf{S}_{[i]}$$

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

Third step: output computation

$$\mathbf{O}_{[i+1]} = \mathbf{Q}_{[i+1]}\mathbf{S}_{[i]} + \left(\left(\mathbf{Q}_{[i+1]}\mathbf{K}^{\mathsf{T}}_{[i+1]}\right)\odot\mathbf{M}\right)\mathbf{V}_{[i+1]}$$



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]

Chunkwise parallel form interpolates between fully parallel and recurrent forms.

- C = L → Fully parallel form
- C = 1 → Fully recurrent form
- C is set to multiple of 16 to leverage tensor cores
  - Larger/smaller C
    - Fewer/more recurrent step
    - Fewer/more hidden state materialization
    - Higher/smaller FLOPs
  - In practice we use C={64, 128, 256} to make a balance
    - Enables linear scaling of training length in a hardware-efficient manner

Linear Attention: "Chunkwise Parallel Form" [Hua et al. '22, Sun et al. '23]

Chunkwise parallel form interpolates between fully parallel and recurrent forms.

- C = L → Fully parallel form
- C = 1 → Fully recurrent form
- C is set to multiple of 16 to leverage tensor cores
  - Larger C
    - ☺ Fewer recurrent step
    - ☺ Fewer hidden state materialization
    - ☹ Higher FLOPs
  - In practice we use C={64, 128, 256} to make a balance
    - Enables linear scaling of training length in a hardware-efficient manner

Linear Attention: "Chunkwise Parallel Form" [Hua et al. '22, Sun et al. '23]

Chunkwise parallel form interpolates between fully parallel and recurrent forms.

- C = L → Fully parallel form
- C = 1 → Fully recurrent form
- C is set to multiple of 16 to leverage tensor cores
  - Larger/smaller C
    - Fewer/more recurrent step
    - Fewer/more hidden state materialization
    - Higher/smaller FLOPs
  - In practice we use C={64, 128, 256} to make a balance

    ☺ Hardware efficient linear scaling in training length

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

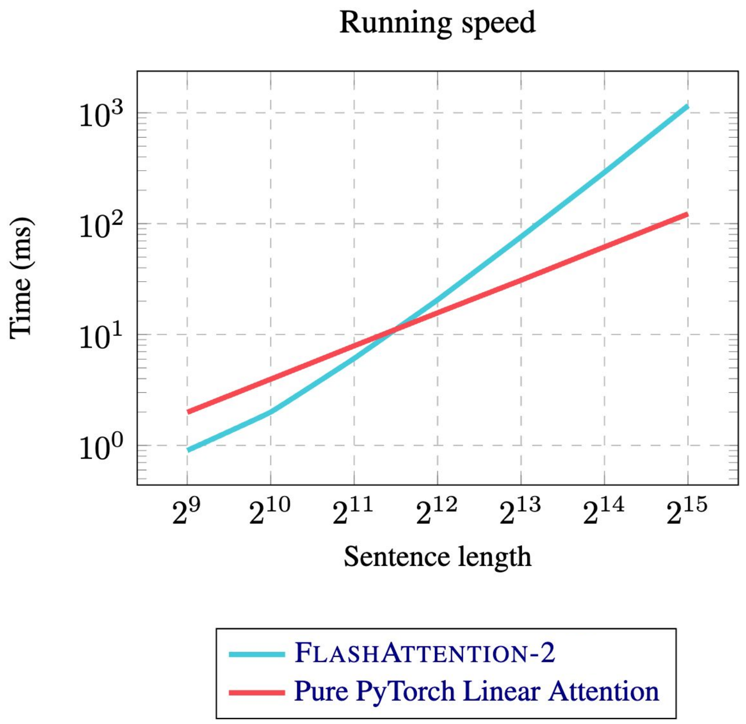
Parallelizing Linear Transformers with the Delta Rule over Sequence Length

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, Yoon Kim
arXiv '24

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.

Model	**PPL** ↓	**LM Eval** ↑
Softmax attention	16.9	50.9
Linear attention with decay (RetNet) $\mathbf{S}_t = \gamma \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top$	18.6	48.9

Our Contributions

Issue 1:

Slower than optimized implementations of softmax attention in practice.

→ **Flash Linear Attention**: Hardware-efficient I/O-aware implementation of linear attention

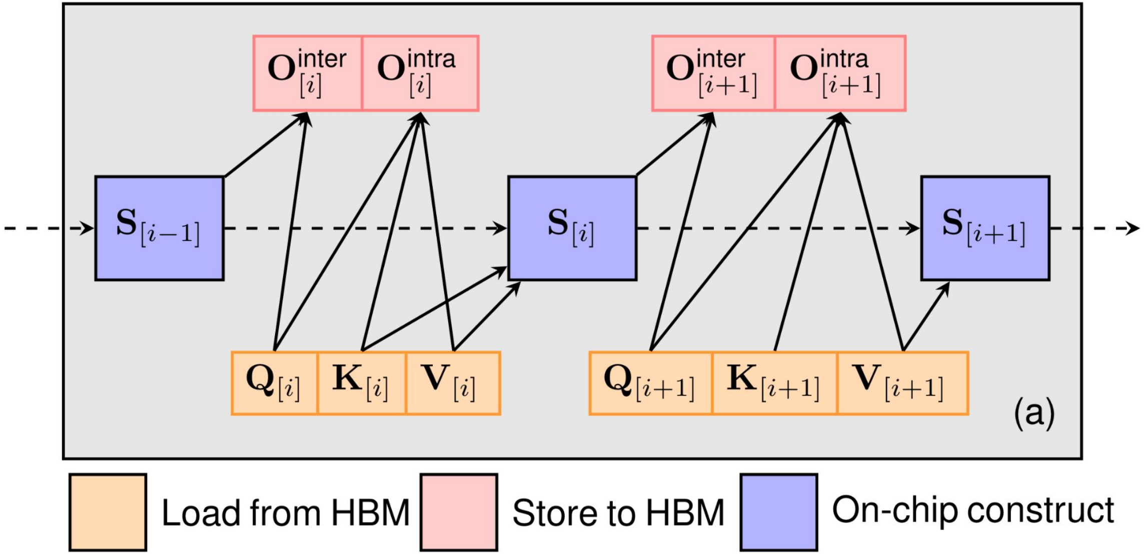
Issue 2:

Underperforms softmax attention by a significant margin.

→ **Gated Linear Attention:** Linear attention with data-dependent “forget” gate

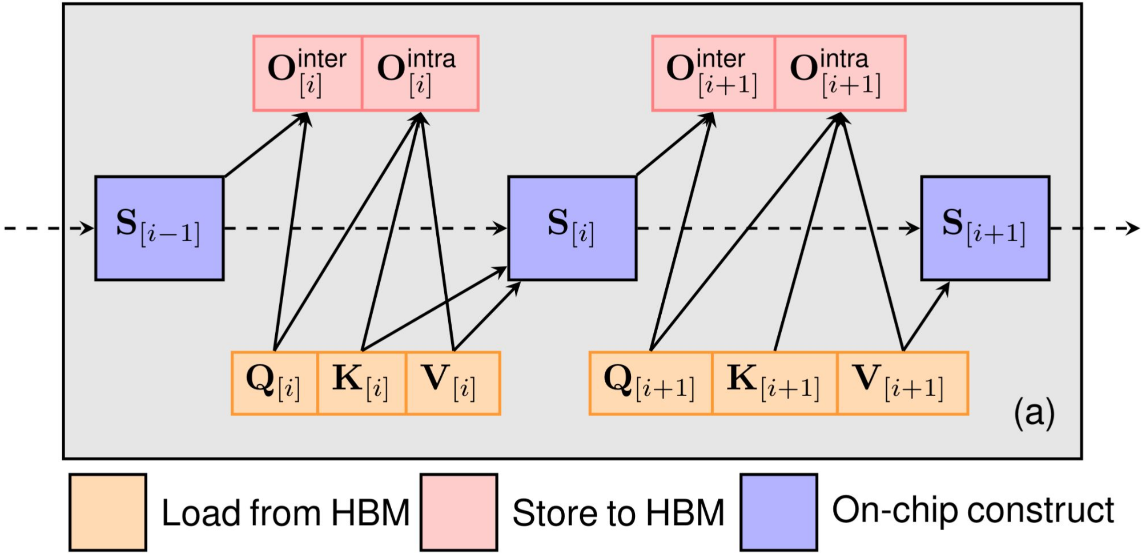
Flash Linear Attention

FlashLinearAttention: Hardware-Efficient I/O-aware Algorithm for Linear Attention (Nonmaterization version)



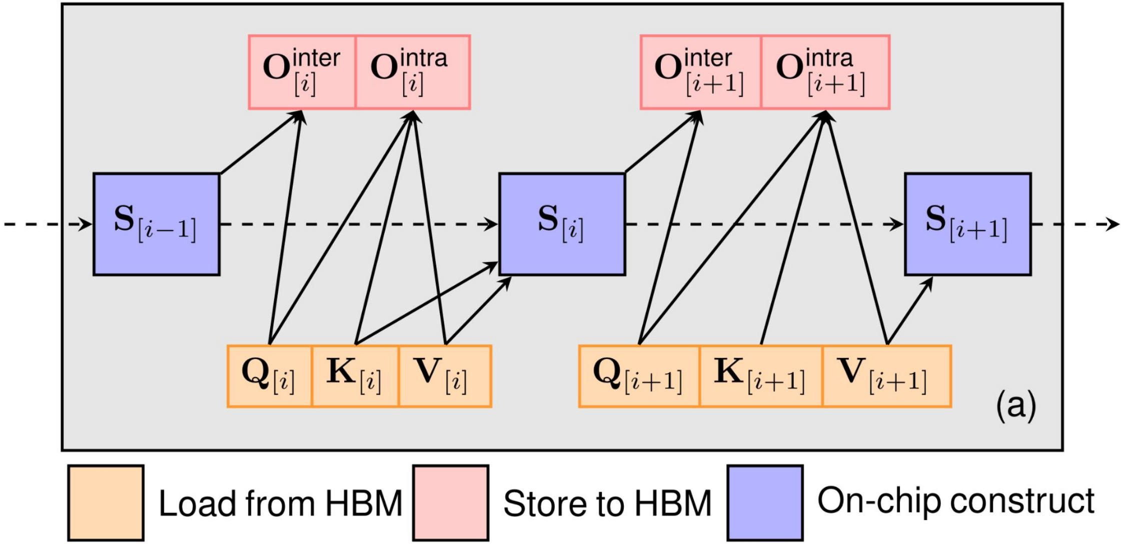
- Pros: minimal I/O cost
  - Hidden states are kept on SRAM throughout the recurrence
    - No I/O cost between HBM and SRAM
  - Only requires loading Q/K/V from HBM once
  - Ideal for short training length where I/O cost dominate

FlashLinearAttention: Hardware-Efficient I/O-aware Algorithm for Linear Attention (Nonmaterization version)



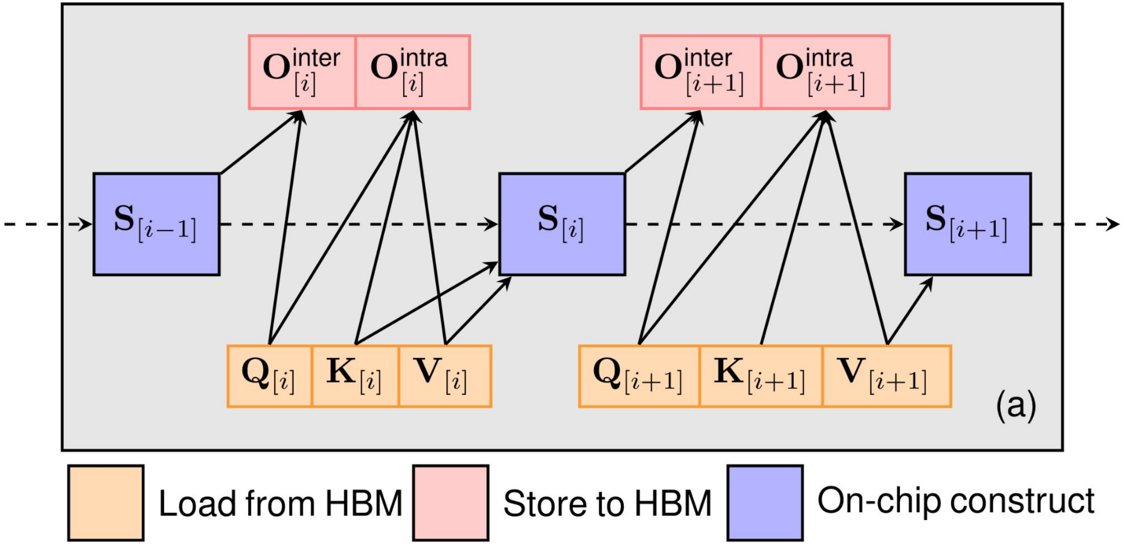
- Pros: minimal I/O cost
  - Hidden states are kept on SRAM throughout the recurrence
    - No I/O cost between HBM and SRAM
  - Only requires loading Q/K/V from HBM once
  - Ideal for short training length where I/O cost dominate

FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention (Nonmaterization version)



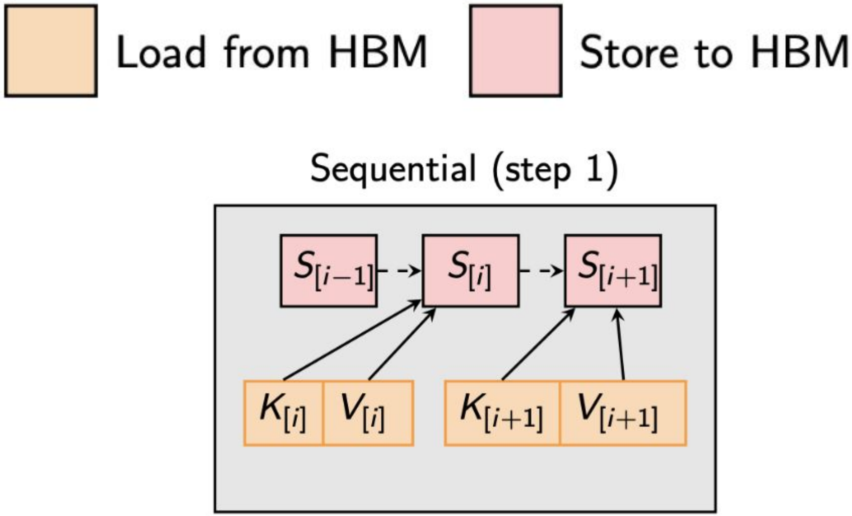
- Cons: lacking sequence-level parallelism across chunks
  - Requires a large batch size to keep SMs busy

FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention (Nonmaterization version)



- Sequence parallelism is important
  - Batch size would be small in large scale and long sequence training
  - SMs have low occupancy $\rightarrow$ Slow down training

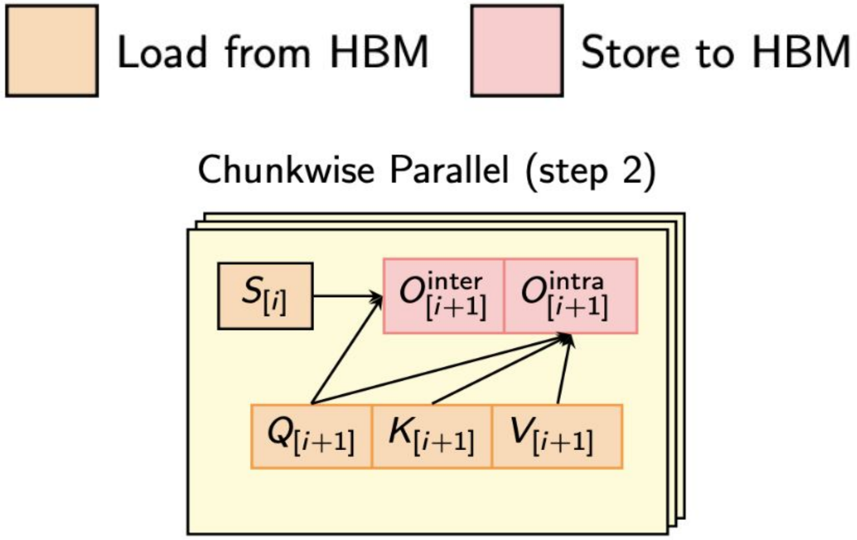
FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention (Materialization version)



Step1: **Sequential** state computation

- Fuse local state computation and state passing (i.e., step1-2 in chunkwise linear attention) in a single kernel to minimize I/O cost
  - One pass of loading K/V and storing S

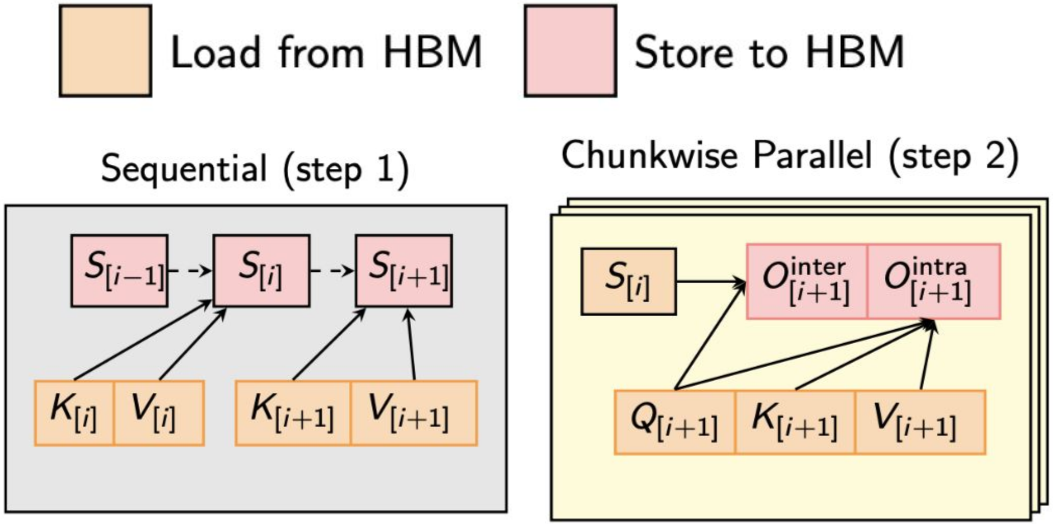
FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention (Materialization version)



Step2: **Parallel** output computation

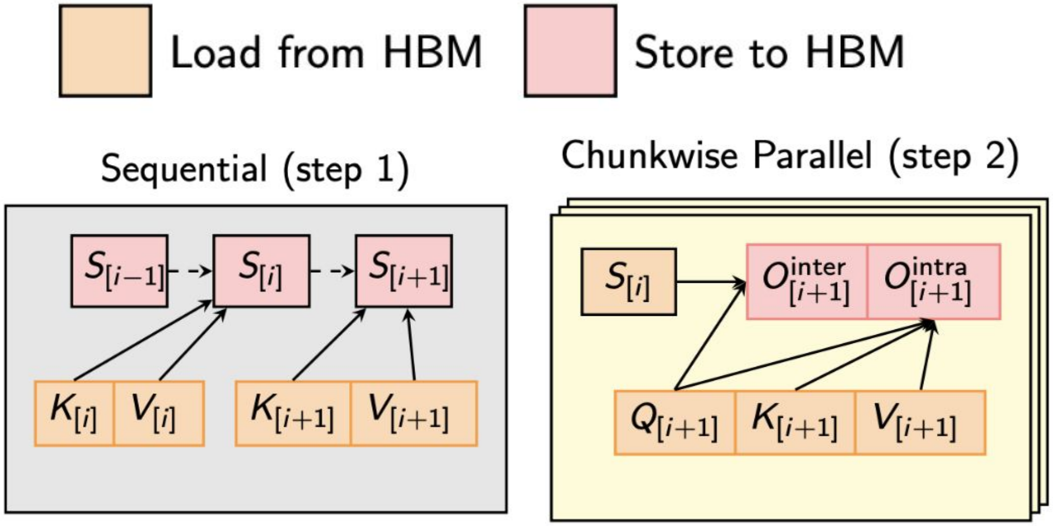
- Compute output of the each chunk **in parallel** based on previous chunk's state and current chunk's query/key/value blocks

FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention (Materialization version)



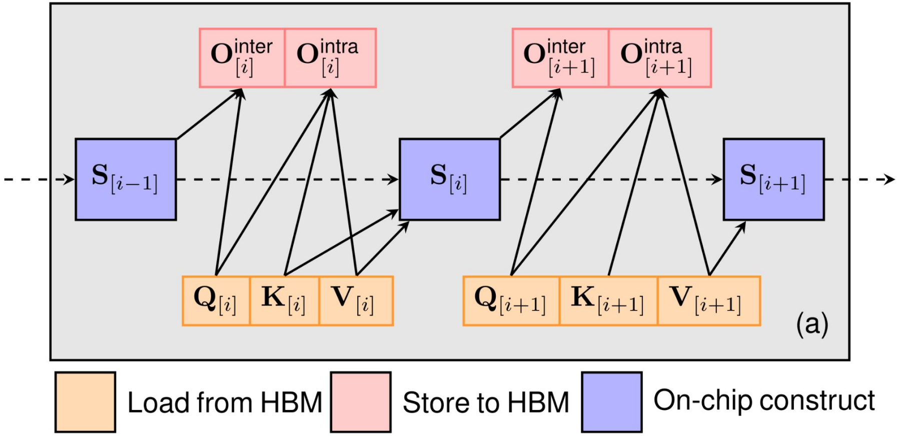
- Pros: enable chunkwise parallelism
  - High SM occupancy
  - Speedup large scale training

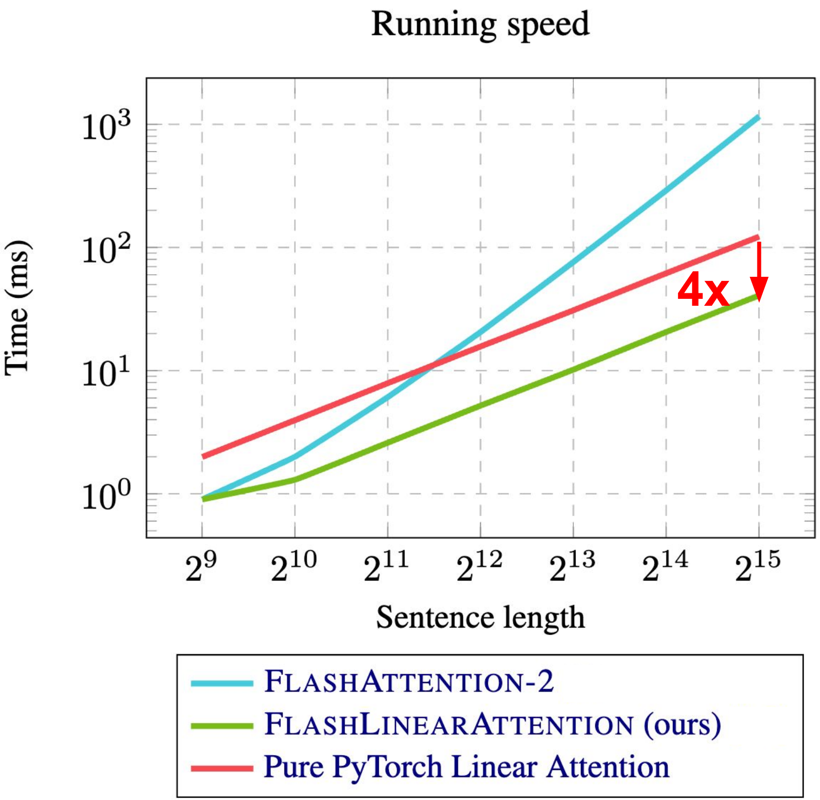
FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention (Materialization version)



- Cons: Higher I/O cost and memory use
  - K/V are loaded twice now; S is saved and loaded once
  - Reduce memory use via recomputation in backward pass

FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention





FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention

https://github.com/sustcsonglin/flash-linear-attention

Flash Linear Attention

Hub | Discord

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

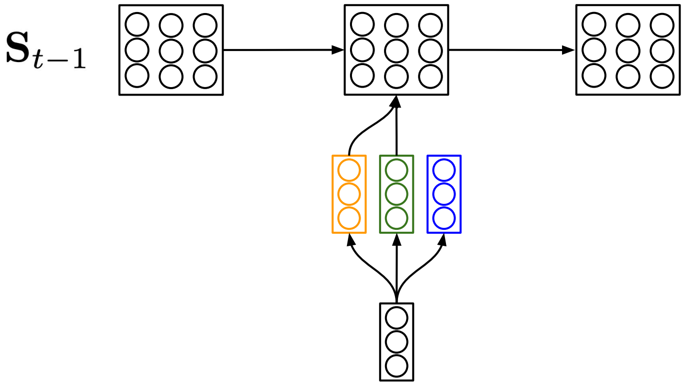
2023-12	GLA (@MIT@IBM)	Gated Linear Attention Transformers with Hardware-Efficient Training	[arxiv]	[official]	code
2023-12	Based (@Stanford@Hazyresearch)	An Educational and Effective Sequence Mixer	[blog]	[official]	code
2024-01	Rebased	Linear Transformers with Learnable Kernel Functions are Better In-Context Models	[arxiv]	[official]	code
2021-02	Delta Net	Linear Transformers Are Secretly Fast Weight Programmers	[arxiv]	[official]	code

Gated Linear Attention

Gated Linear Attention: Data-dependent Multiplicative Gate

Simple Linear Attention

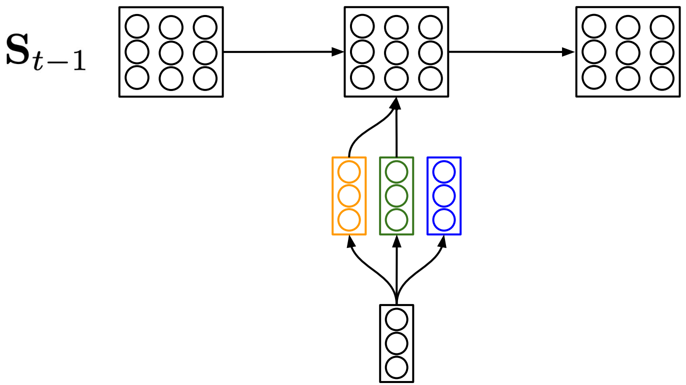
$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top$$



Gated Linear Attention: Data-dependent Multiplicative Gate

Simple Linear Attention

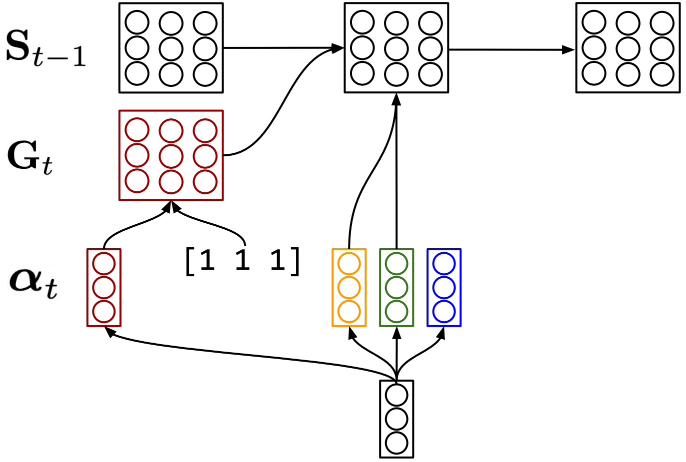
$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^{\top}$$



Gated Linear Attention

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

$$\mathbf{G}_t = \boldsymbol{\alpha}_t \mathbf{1}^{\top}, \quad \boldsymbol{\alpha}_t = \sigma(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2})^{\frac{1}{\tau}}$$



Gated Linear Attention: Parallel Forms

Simple Linear Attention

$$\mathbf{O} = \left((\mathbf{Q}\mathbf{K}^\top) \odot \mathbf{M}\right)\mathbf{V}$$

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

Gated Linear Attention

$$\mathbf{O} = \left(\left(\underbrace{(\mathbf{Q} \odot \mathbf{B})\left(\frac{\mathbf{K}}{\mathbf{B}}\right)^\top}_{\mathbf{P}}\right) \odot \mathbf{M}\right)\mathbf{V}$$

cumulative decay $\boldsymbol{b}_t := \prod_{j=1}^{t} \boldsymbol{\alpha}_j$

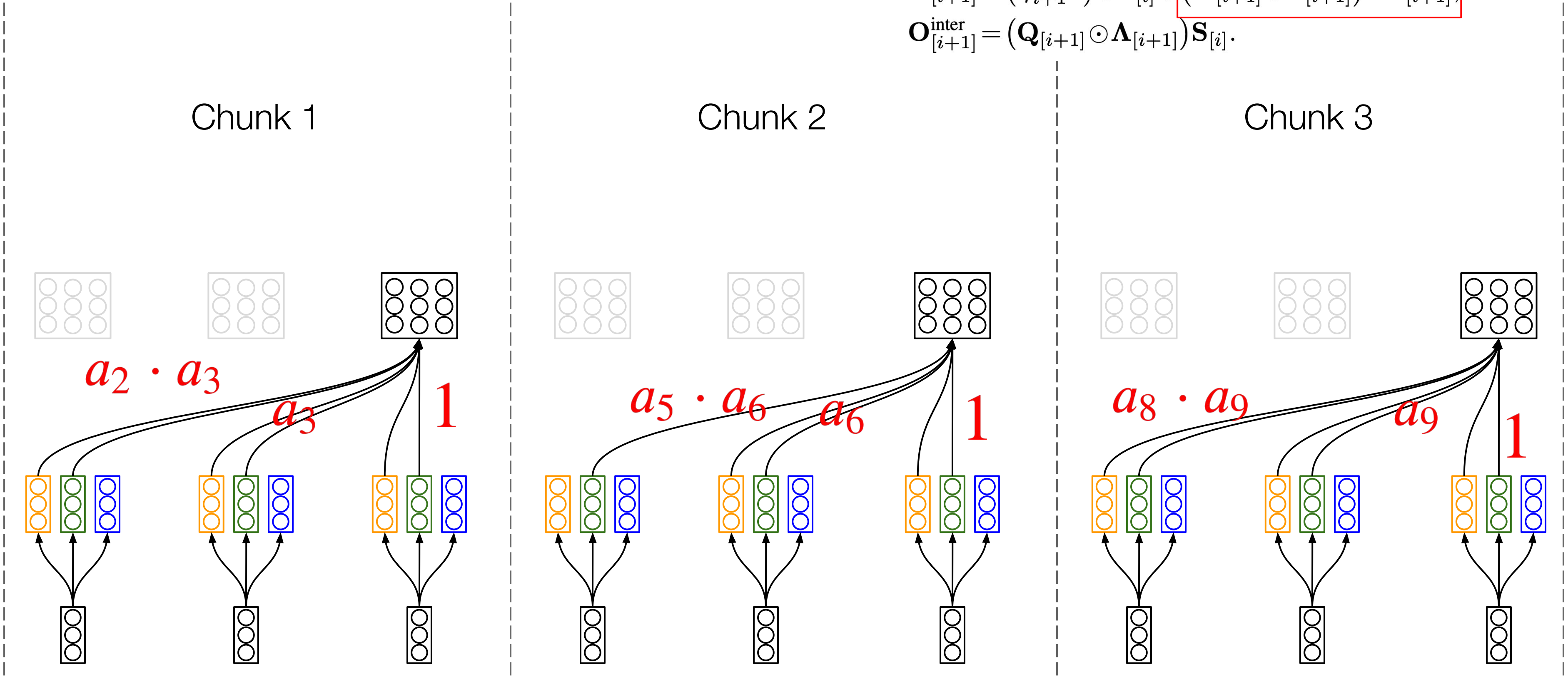
Gated Linear Attention: Decay-aware "Chunkwise Parallel Form"

First step: local state computation

$$\boldsymbol{\Lambda}_{iC+j} = \frac{\boldsymbol{b}_{iC+j}}{\boldsymbol{b}_{iC}}, \boldsymbol{\Gamma}_{iC+j} = \frac{\boldsymbol{b}_{(i+1)C}}{\boldsymbol{b}_{iC+j}}, \boldsymbol{\gamma}_{i+1} = \frac{\boldsymbol{b}_{(i+1)C}}{\boldsymbol{b}_{iC}},$$

$$\mathbf{S}_{[i+1]} = (\boldsymbol{\gamma}_{i+1}^\top \mathbf{1}) \odot \mathbf{S}_{[i]} + \left(\mathbf{K}_{[i+1]} \odot \boldsymbol{\Gamma}_{[i+1]}\right)^\top \mathbf{V}_{[i+1]},$$

$$\mathbf{O}_{[i+1]}^{\text{inter}} = (\mathbf{Q}_{[i+1]} \odot \boldsymbol{\Lambda}_{[i+1]}) \mathbf{S}_{[i]}.$$



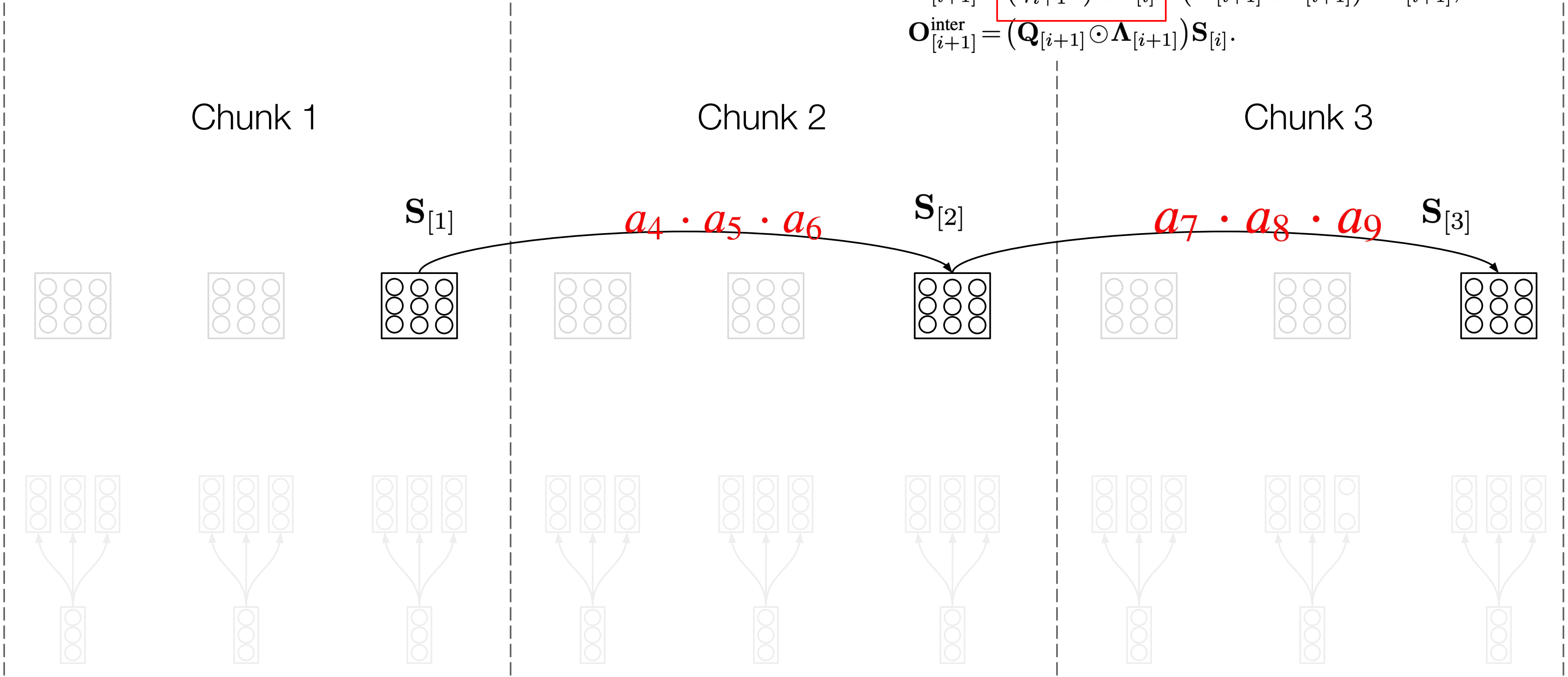
Gated Linear Attention: Decay-aware "Chunkwise Parallel Form"

Second step: state passing

$$\boldsymbol{\Lambda}_{iC+j} = \frac{\boldsymbol{b}_{iC+j}}{\boldsymbol{b}_{iC}}, \boldsymbol{\Gamma}_{iC+j} = \frac{\boldsymbol{b}_{(i+1)C}}{\boldsymbol{b}_{iC+j}}, \boldsymbol{\gamma}_{i+1} = \frac{\boldsymbol{b}_{(i+1)C}}{\boldsymbol{b}_{iC}},$$

$$\mathbf{S}_{[i+1]} = (\boldsymbol{\gamma}_{i+1}^{\top}\mathbf{1}) \odot \mathbf{S}_{[i]} + (\mathbf{K}_{[i+1]} \odot \boldsymbol{\Gamma}_{[i+1]})^{\top} \mathbf{V}_{[i+1]},$$

$$\mathbf{O}_{[i+1]}^{\text{inter}} = (\mathbf{Q}_{[i+1]} \odot \boldsymbol{\Lambda}_{[i+1]}) \mathbf{S}_{[i]}.$$



Gated Linear Attention: Decay-aware "Chunkwise Parallel Form"

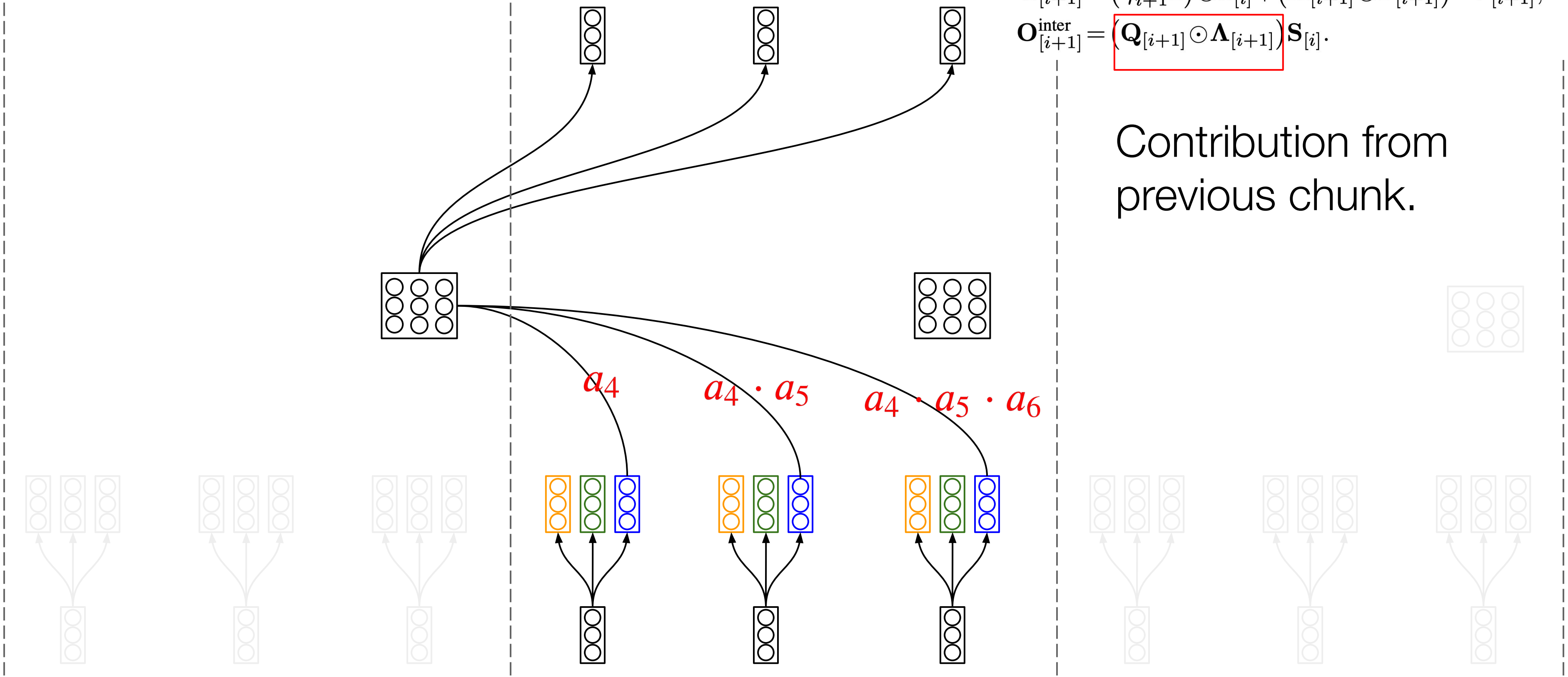
Third step: output computation

$$\mathbf{\Lambda}_{iC+j} = \frac{\boldsymbol{b}_{iC+j}}{\boldsymbol{b}_{iC}}, \mathbf{\Gamma}_{iC+j} = \frac{\boldsymbol{b}_{(i+1)C}}{\boldsymbol{b}_{iC+j}}, \boldsymbol{\gamma}_{i+1} = \frac{\boldsymbol{b}_{(i+1)C}}{\boldsymbol{b}_{iC}},$$

$$\mathbf{S}_{[i+1]} = \left(\boldsymbol{\gamma}_{i+1}^{\top}\mathbf{1}\right) \odot \mathbf{S}_{[i]} + \left(\mathbf{K}_{[i+1]} \odot \mathbf{\Gamma}_{[i+1]}\right)^{\top} \mathbf{V}_{[i+1]},$$

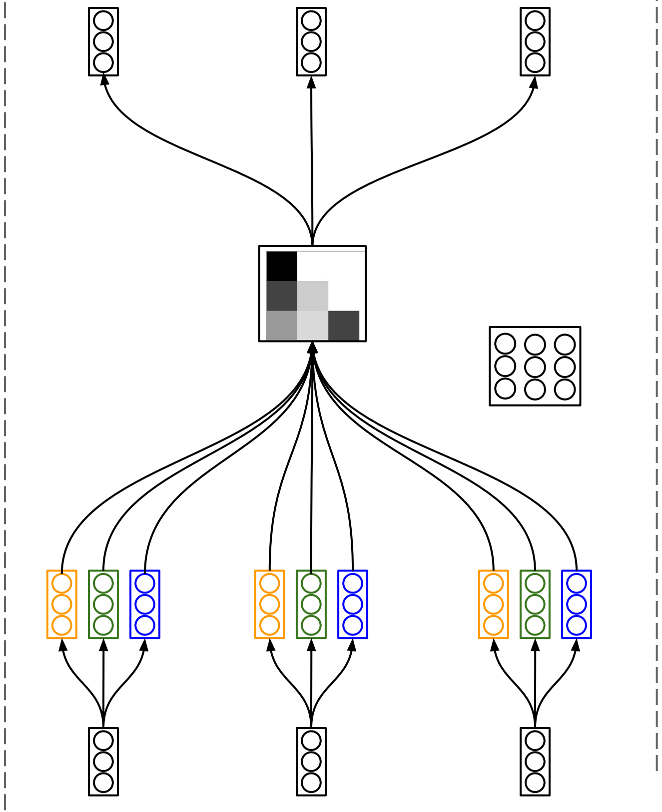
$$\mathbf{O}_{[i+1]}^{\text{inter}} = \left(\mathbf{Q}_{[i+1]} \odot \mathbf{\Lambda}_{[i+1]}\right) \mathbf{S}_{[i]}.$$

Contribution from previous chunk.



Gated Linear Attention: Decay-aware "Chunkwise Parallel Form"

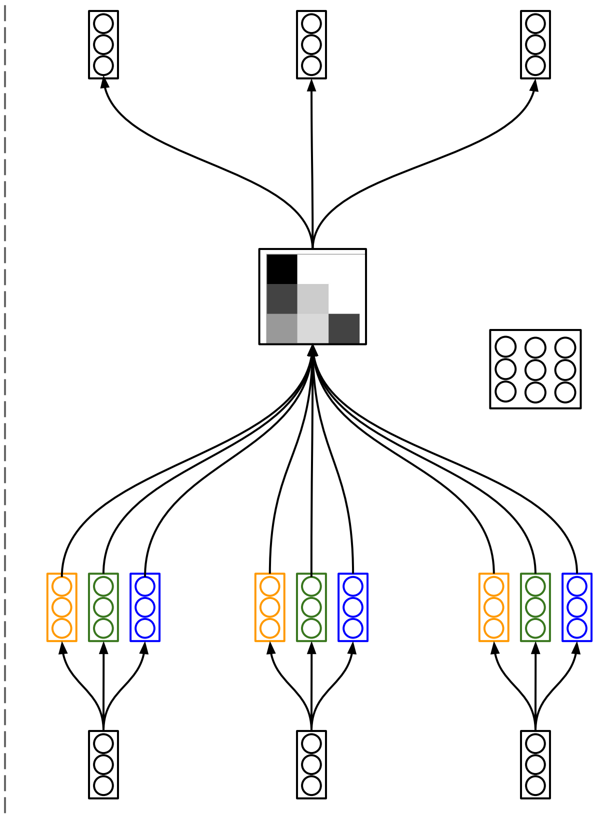
Chunk-level (linear) attention for contribution from current chunk



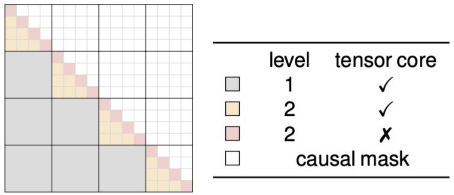
	Stable	Tensor core
$\mathbf{O} = \left(\left(\underbrace{(\mathbf{Q} \odot \mathbf{B})\left(\frac{\mathbf{K}}{\mathbf{B}}\right)^{\top}}_{\mathbf{P}}\right) \odot \mathbf{M}\right)\mathbf{V}$	☹	☺
$\mathbf{P}_{ij} = \sum_{k=1}^{d} \mathbf{Q}_{ik}\mathbf{K}_{jk}\exp(\log \mathbf{B}_{ik} - \log \mathbf{B}_{jk})$	☺	☹

Gated Linear Attention: Decay-aware "Chunkwise Parallel Form"

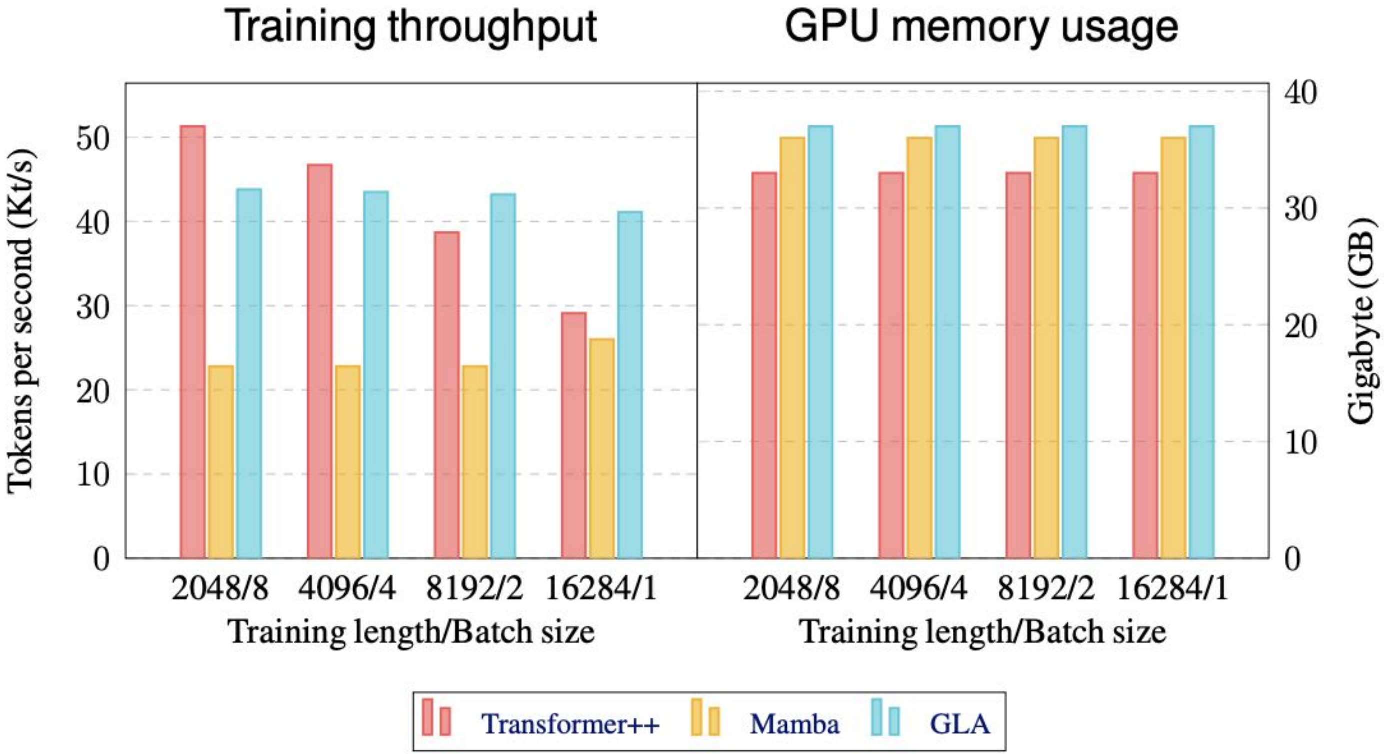
Chunk-level (linear) attention for contribution from current chunk



	Stable	Tensor core
$\mathbf{O} = \left(\left(\underbrace{(\mathbf{Q} \odot \mathbf{B})\left(\frac{\mathbf{K}}{\mathbf{B}}\right)^{\top}}_{\mathbf{P}}\right) \odot \mathbf{M}\right)\mathbf{V}$	☹	☺
$\mathbf{P}_{ij} = \sum_{k=1}^{d} \mathbf{Q}_{ik}\mathbf{K}_{jk}\exp(\log \mathbf{B}_{ik} - \log \mathbf{B}_{jk})$	☺	☹
Secondary chunking	☺	☺



Gated Linear Attention: Throughput



Gated Linear Attention: Performance

Model	PPL ↓	LM Eval ↑
Transformer++	16.9	50.9
RetNet	18.6	48.9
Mamba	17.1	50.0
Gated Linear Attention	17.2	51.1

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 2 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…

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 2 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…

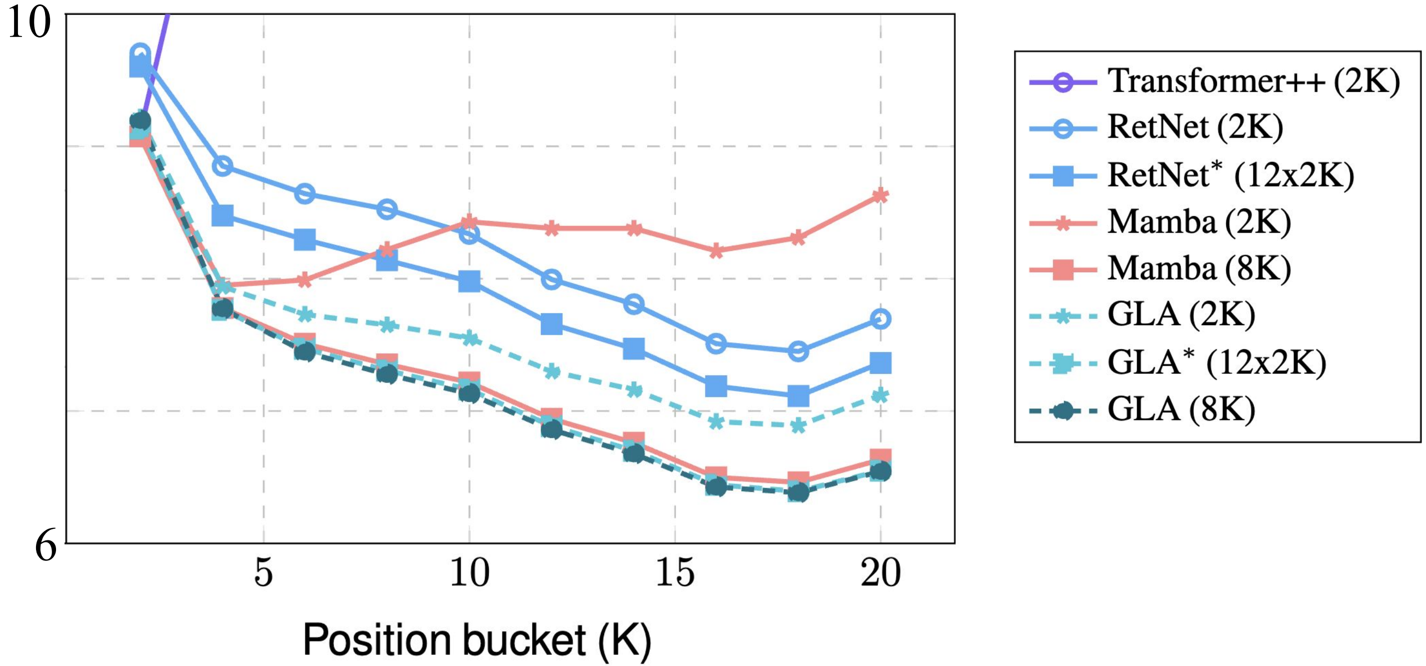
Type of Test → Nucleic acid amplification assay (Helicase-dependent Amplification, HDA)

Gated Linear Attention: Recall-oriented Tasks

Model	PPL ↓	LM Eval ↑	Retrieval ↑
Transformer++	16.9	50.9	41.8
RetNet	18.6	48.9	30.6
Mamba	17.1	50.0	27.6
Gated Linear Attention	17.2	51.1	37.7

1.3B models trained on 100B tokens

Gated Linear Attention: Length Generalization



Gated Linear Attention Transformers or State-Space Models?

Gated Linear Attention

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

Mamba

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \quad (1a)$$
$$y(t) = \boldsymbol{C}h(t) \quad (1b)$$

$$h_t = \overline{\boldsymbol{A}}h_{t-1} + \overline{\boldsymbol{B}}x_t \quad (2a)$$
$$y_t = \boldsymbol{C}h_t \quad (2b)$$

$$\overline{\boldsymbol{K}} = (\boldsymbol{C}\overline{\boldsymbol{B}}, \boldsymbol{C}\overline{\boldsymbol{AB}}, \dots, \boldsymbol{C}\overline{\boldsymbol{A}}^k\overline{\boldsymbol{B}}, \dots)$$
$$y = x * \overline{\boldsymbol{K}}$$

$$\overline{\boldsymbol{A}} = \exp(\Delta \boldsymbol{A}) \qquad \overline{\boldsymbol{B}} = (\Delta \boldsymbol{A})^{-1}(\exp(\Delta \boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta \boldsymbol{B}$$

Algorithm 1 SSM (S4)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▹ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
$\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
$\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter})$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▹ Time-invariant: recurrence or convolution
return y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▹ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
$\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{B}, \mathsf{L}, \mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▹ Time-varying: recurrence (*scan*) only
return y

Gated Linear Attention Transformers **are** State-Space Models!

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

Model	Parameterization	Parameters
Mamba	$\mathbf{G}_t = \exp(-(\mathbf{1}^{\top}\boldsymbol{\alpha}_t) \odot \exp(\boldsymbol{A})), \quad \boldsymbol{\alpha}_t = \text{softplus}(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2})$	$\boldsymbol{A}, \boldsymbol{W}_{\alpha_1}, \boldsymbol{W}_{\alpha_2}$
Mamba-2	$\mathbf{G}_t = \gamma_t \mathbf{1}^{\top}\mathbf{1}, \quad \gamma_t = \exp\left(-\text{softplus}\left(\boldsymbol{x}_t \boldsymbol{W}_{\gamma}\right) \exp\left(a\right)\right)$	$\boldsymbol{W}_{\gamma}, a$
mLSTM	$\mathbf{G}_t = \gamma_t \mathbf{1}^{\top}\mathbf{1}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\gamma}\right)$	$\boldsymbol{W}_{\gamma}$
Gated RetNet	$\mathbf{G}_t = \gamma_t \mathbf{1}^{\top}\mathbf{1}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\gamma}\right)^{\frac{1}{\tau}}$	$\boldsymbol{W}_{\gamma}$
HGRN-2	$\mathbf{G}_t = \boldsymbol{\alpha}_t^{\top}\mathbf{1}, \quad \boldsymbol{\alpha}_t = \boldsymbol{\gamma} + (1-\boldsymbol{\gamma})\sigma(\boldsymbol{x}_t \boldsymbol{W}_{\alpha})$	$\boldsymbol{W}_{\alpha}, \boldsymbol{\gamma}$
RWKV-6	$\mathbf{G}_t = \boldsymbol{\alpha}_t^{\top}\mathbf{1}, \quad \boldsymbol{\alpha}_t = \exp\left(-\exp\left(\boldsymbol{x}_t \boldsymbol{W}_{\alpha}\right)\right)$	$\boldsymbol{W}_{\alpha}$
GLA (ours)	$\mathbf{G}_t = \boldsymbol{\alpha}_t^{\top}\mathbf{1}, \quad \boldsymbol{\alpha}_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2}\right)^{\frac{1}{\tau}}$	$\boldsymbol{W}_{\alpha_1}, \boldsymbol{W}_{\alpha_2}$

Gated linear attention $\subset$ State-space models

Gated Linear Attention Transformers **are** Scalable State-Space Models!

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

Model	Parameterization	Parameters
Mamba	$\mathbf{G}_t = \exp(-(\mathbf{1}^\top \boldsymbol{\alpha}_t) \odot \exp(\boldsymbol{A})), \quad \boldsymbol{\alpha}_t = \mathrm{softplus}(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2})$	$\boldsymbol{A}, \boldsymbol{W}_{\alpha_1}, \boldsymbol{W}_{\alpha_2}$
Mamba-2	$\mathbf{G}_t = \gamma_t \mathbf{1}^\top \mathbf{1}, \quad \gamma_t = \exp\left(-\mathrm{softplus}\left(\boldsymbol{x}_t \boldsymbol{W}_\gamma\right) \exp\left(a\right)\right)$	$\boldsymbol{W}_\gamma, a$
mLSTM	$\mathbf{G}_t = \gamma_t \mathbf{1}^\top \mathbf{1}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_\gamma\right)$	$\boldsymbol{W}_\gamma$
Gated RetNet	$\mathbf{G}_t = \gamma_t \mathbf{1}^\top \mathbf{1}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_\gamma\right)^{\frac{1}{\tau}}$	$\boldsymbol{W}_\gamma$
HGRN-2	$\mathbf{G}_t = \boldsymbol{\alpha}_t^\top \mathbf{1}, \quad \boldsymbol{\alpha}_t = \boldsymbol{\gamma} + (1 - \boldsymbol{\gamma})\sigma(\boldsymbol{x}_t \boldsymbol{W}_\alpha)$	$\boldsymbol{W}_\alpha, \boldsymbol{\gamma}$
RWKV-6	$\mathbf{G}_t = \boldsymbol{\alpha}_t^\top \mathbf{1}, \quad \boldsymbol{\alpha}_t = \exp\left(-\exp\left(\boldsymbol{x}_t \boldsymbol{W}_\alpha\right)\right)$	$\boldsymbol{W}_\alpha$
GLA (ours)	$\mathbf{G}_t = \boldsymbol{\alpha}_t^\top \mathbf{1}, \quad \boldsymbol{\alpha}_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2}\right)^{\frac{1}{\tau}}$	$\boldsymbol{W}_{\alpha_1}, \boldsymbol{W}_{\alpha_2}$

Scalable state-space models ⊂ Gated linear attention

Gated Linear Attention Transformers **are** Scalable State-Space Models!

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

Model	Parameterization	Parameters
Mamba	$\mathbf{G}_t = \exp(-(\mathbf{1}^\top \boldsymbol{\alpha}_t) \odot \exp(\boldsymbol{A})), \quad \boldsymbol{\alpha}_t = \text{softplus}(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2})$	$\boldsymbol{A}, \boldsymbol{W}_{\alpha_1}, \boldsymbol{W}_{\alpha_2}$
Mamba-2	$\mathbf{G}_t = \gamma_t \mathbf{1}^\top \mathbf{1}, \quad \gamma_t = \exp\left(-\text{softplus}\left(\boldsymbol{x}_t \boldsymbol{W}_\gamma\right) \exp\left(a\right)\right)$	$\boldsymbol{W}_\gamma, a$
mLSTM	$\mathbf{G}_t = \gamma_t \mathbf{1}^\top \mathbf{1}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_\gamma\right)$	$\boldsymbol{W}_\gamma$
Gated RetNet	$\mathbf{G}_t = \gamma_t \mathbf{1}^\top \mathbf{1}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_\gamma\right)^{\frac{1}{\tau}}$	$\boldsymbol{W}_\gamma$
HGRN-2	$\mathbf{G}_t = \boldsymbol{\alpha}_t^\top \mathbf{1}, \quad \boldsymbol{\alpha}_t = \boldsymbol{\gamma} + (1 - \boldsymbol{\gamma}) \sigma(\boldsymbol{x}_t \boldsymbol{W}_\alpha)$	$\boldsymbol{W}_\alpha, \boldsymbol{\gamma}$
RWKV-6	$\mathbf{G}_t = \boldsymbol{\alpha}_t^\top \mathbf{1}, \quad \boldsymbol{\alpha}_t = \exp\left(-\exp\left(\boldsymbol{x}_t \boldsymbol{W}_\alpha\right)\right)$	$\boldsymbol{W}_\alpha$
GLA (ours)	$\mathbf{G}_t = \boldsymbol{\alpha}_t^\top \mathbf{1}, \quad \boldsymbol{\alpha}_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2}\right)^{\frac{1}{\tau}}$	$\boldsymbol{W}_{\alpha_1}, \boldsymbol{W}_{\alpha_2}$

Scalable state-space models ⊂ Gated linear attention

Scalable here: efficient scaling of state size → recurrence has matmul form

Gated Linear Attention Transformers **are** Scalable State-Space Models!

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

Model	Parameterization	Parameters
Mamba	$\mathbf{G}_t = \exp(-(\mathbf{1}^\top \boldsymbol{\alpha}_t) \odot \exp(\boldsymbol{A})), \quad \boldsymbol{\alpha}_t = \text{softplus}(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2})$	$\boldsymbol{A}, \boldsymbol{W}_{\alpha_1}, \boldsymbol{W}_{\alpha_2}$
Mamba-2	$\mathbf{G}_t = \gamma_t \mathbf{1}^\top \mathbf{1}, \quad \gamma_t = \exp\left(-\text{softplus}\left(\boldsymbol{x}_t \boldsymbol{W}_\gamma\right) \exp\left(a\right)\right)$	$\boldsymbol{W}_\gamma, a$
mLSTM	$\mathbf{G}_t = \gamma_t \mathbf{1}^\top \mathbf{1}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_\gamma\right)$	$\boldsymbol{W}_\gamma$
Gated RetNet	$\mathbf{G}_t = \gamma_t \mathbf{1}^\top \mathbf{1}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_\gamma\right)^{\frac{1}{\tau}}$	$\boldsymbol{W}_\gamma$
HGRN-2	$\mathbf{G}_t = \boldsymbol{\alpha}_t^\top \mathbf{1}, \quad \boldsymbol{\alpha}_t = \boldsymbol{\gamma} + (1 - \boldsymbol{\gamma})\sigma(\boldsymbol{x}_t \boldsymbol{W}_\alpha)$	$\boldsymbol{W}_\alpha, \boldsymbol{\gamma}$
RWKV-6	$\mathbf{G}_t = \boldsymbol{\alpha}_t^\top \mathbf{1}, \quad \boldsymbol{\alpha}_t = \exp\left(-\exp\left(\boldsymbol{x}_t \boldsymbol{W}_\alpha\right)\right)$	$\boldsymbol{W}_\alpha$
GLA (ours)	$\mathbf{G}_t = \boldsymbol{\alpha}_t^\top \mathbf{1}, \quad \boldsymbol{\alpha}_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2}\right)^{\frac{1}{\tau}}$	$\boldsymbol{W}_{\alpha_1}, \boldsymbol{W}_{\alpha_2}$

Scalable state-space models $\subset$ Gated linear attention

G_t must be of the form $\alpha_t \beta_t^\top$ to rewrite recurrence in matmul form

Summary

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

$$\mathrm{A}\ 4\ \mathrm{B}\ 3\ \mathrm{C}\ 6\ \mathrm{F}\ 1\ \mathrm{E}\ 2 \rightarrow \mathrm{A}\ ?\ \mathrm{C}\ ?\ \mathrm{F}\ ?\ \mathrm{E}\ ?\ \mathrm{B}\ ?$$

[Example from: Arora et al. ’24]

Deficiencies of Linear Attention / State-Space Models

Multi-Query Associative Recall Task

Input

$$\mathrm{A}\ 4\ \mathrm{B}\ 3\ \mathrm{C}\ 6\ \underbrace{\mathrm{F}\ 1}_{\textbf{Key-Value}}\ \mathrm{E}\ 2 \rightarrow \mathrm{A}\ ?\ \mathrm{C}\ ?\ \underbrace{\mathrm{F}\ ?}_{\textbf{Query}}\ \mathrm{E}\ ?\ \mathrm{B}\ ?$$

[Example from: Arora et al. '24]

Deficiencies of Linear Attention / State-Space Models

Multi-Query Associative Recall Task

Input

$$\mathrm{A}\ 4\ \mathrm{B}\ 3\ \mathrm{C}\ 6\ \mathrm{F}\ 1\ \mathrm{E}\ 2 \rightarrow \mathrm{A}\ ?\ \mathrm{C}\ ?\ \mathrm{F}\ ?\ \mathrm{E}\ ?\ \mathrm{B}\ ?$$

Output

$$4,\ 6,\ 1,\ 2,\ 3$$

[Example from: Arora et al. '24]

Deficiencies of Linear Attention / State-Space Models

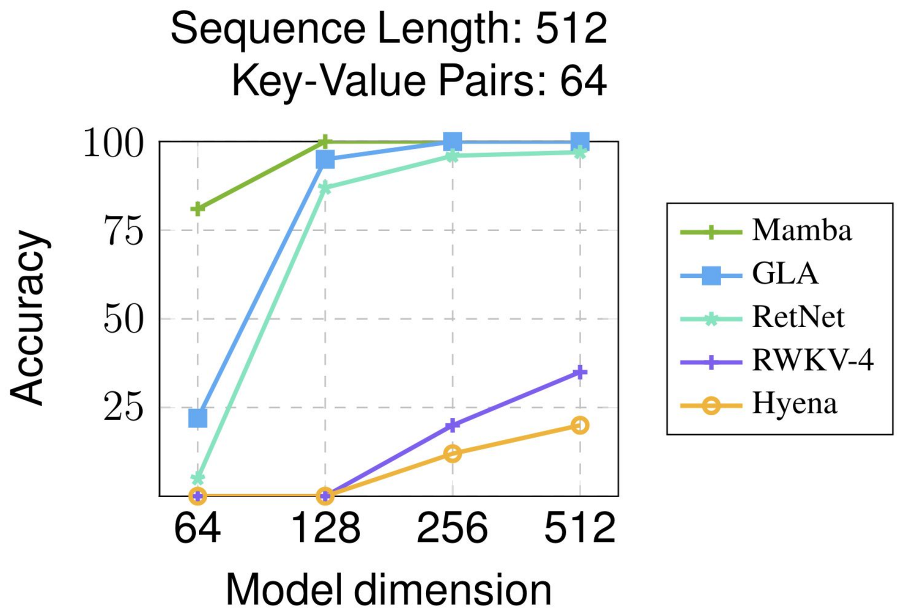
Multi-Query Associative Recall Task

Input

$$A\ 4\ B\ 3\ C\ 6\ F\ 1\ E\ 2 \rightarrow A\ ?\ C\ ?\ F\ ?\ E\ ?\ B\ ?$$

Output

$$4,\ 6,\ 1,\ 2,\ 3$$



[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 $$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{W}_Q \boldsymbol{x}_t, \boldsymbol{W}_K \boldsymbol{x}_t, \boldsymbol{W}_V \boldsymbol{x}_t$$

Retrieve old memory $$\boldsymbol{v}_t^{\text{old}} = \mathbf{S}_{t-1} \boldsymbol{k}_t$$

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 $$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{W}_Q\boldsymbol{x}_t, \boldsymbol{W}_K\boldsymbol{x}_t, \boldsymbol{W}_V\boldsymbol{x}_t$$

Retrieve old memory $$\boldsymbol{v}_t^{\text{old}} = \mathbf{S}_{t-1}\boldsymbol{k}_t$$

Combine old memory with current value vector $$\boldsymbol{v}_t^{\text{new}} = \beta_t\boldsymbol{v}_t + (1-\beta_t)\,\boldsymbol{v}_t^{\text{old}}$$

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

$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{W}_Q \boldsymbol{x}_t, \boldsymbol{W}_K \boldsymbol{x}_t, \boldsymbol{W}_V \boldsymbol{x}_t$$

Retrieve old memory

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

Combine old memory with current value vector

$$\boldsymbol{v}_t^{\text{new}} = \beta_t \boldsymbol{v}_t + (1 - \beta_t)\, \boldsymbol{v}_t^{\text{old}}$$

$$\beta_t = \sigma(\mathbf{W}_\beta \boldsymbol{x}_t) \in (0, 1)$$

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
$$\boldsymbol{q}_t,\ \boldsymbol{k}_t,\ \boldsymbol{v}_t = \boldsymbol{W}_Q \boldsymbol{x}_t, \boldsymbol{W}_K \boldsymbol{x}_t, \boldsymbol{W}_V \boldsymbol{x}_t$$

Retrieve old memory
$$\boldsymbol{v}_t^{\text{old}} = \mathbf{S}_{t-1} \boldsymbol{k}_t$$

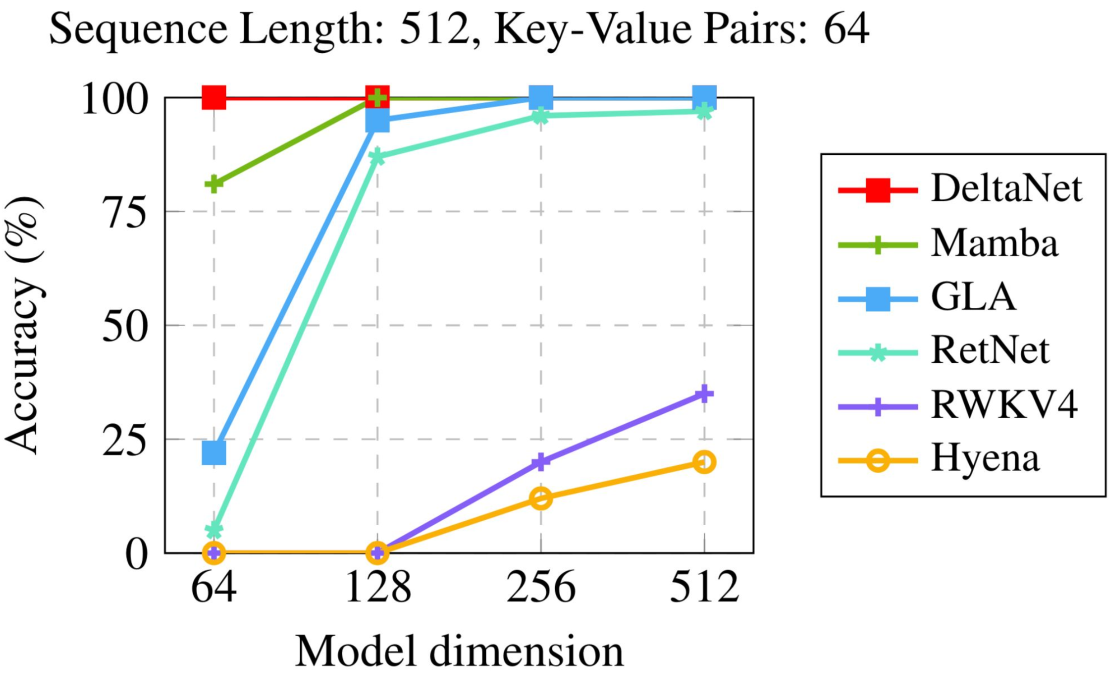
Combine old memory with current value vector
$$\boldsymbol{v}_t^{\text{new}} = \beta_t \boldsymbol{v}_t + (1 - \beta_t)\, \boldsymbol{v}_t^{\text{old}}$$

Remove old memory, write new memory
$$\mathbf{S}_t = \mathbf{S}_{t-1} \underbrace{-\boldsymbol{v}_t^{\text{old}} \boldsymbol{k}_t^{\top}}_{\text{remove}} \underbrace{+\boldsymbol{v}_t^{\text{new}} \boldsymbol{k}_t^{\top}}_{\text{write}}$$

Get output
$$\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t$$

DeltaNet Associative Recall Performance

Multi-Query Associative Recall Task



DeltaNet Associative Recall Performance

Mechanistic architecture design

Model	Compress	Fuzzy Recall	In-Context Recall	Memorize	Noisy Recall	Selective Copy	Average
Transformer	51.6	29.8	94.1	85.2	86.8	99.6	74.5
Hyena [71]	45.2	7.9	81.7	89.5	78.8	93.1	66.0
Multihead Hyena [56]	44.8	14.4	99.0	89.4	98.6	93.0	73.2
Mamba [25]	52.7	6.7	90.4	89.5	90.1	86.3	69.3
GLA [101]	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

DeltaNet Issue

$$\boldsymbol{v}_t^{\text{old}} = \mathbf{S}_{t-1}\boldsymbol{k}_t$$
$$\boldsymbol{v}_t^{\text{new}} = \beta_t \boldsymbol{v}_t + (1-\beta_t)\,\boldsymbol{v}_t^{\text{old}}$$

$\longrightarrow$

$$\boldsymbol{u}_t = \boldsymbol{v}_t^{\text{new}} - \boldsymbol{v}_t^{\text{old}}$$

$$\mathbf{S}_t = \mathbf{S}_{t-1} \underbrace{-\boldsymbol{v}_t^{\text{old}}\boldsymbol{k}_t^{\intercal}}_{\text{remove}} \underbrace{+\boldsymbol{v}_t^{\text{new}}\boldsymbol{k}_t^{\intercal}}_{\text{write}}$$

$\longrightarrow$

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{u}_t\boldsymbol{k}_t^{\intercal}$$

DeltaNet Issue

$$\boldsymbol{v}_t^{\text{old}} = \mathbf{S}_{t-1}\boldsymbol{k}_t$$
$$\boldsymbol{v}_t^{\text{new}} = \beta_t \boldsymbol{v}_t + (1-\beta_t)\,\boldsymbol{v}_t^{\text{old}}$$

$$\boldsymbol{u}_t = \boldsymbol{v}_t^{\text{new}} - \boldsymbol{v}_t^{\text{old}}$$

$$\mathbf{S}_t = \mathbf{S}_{t-1} \underbrace{-\boldsymbol{v}_t^{\text{old}}\boldsymbol{k}_t^\intercal}_{\text{remove}} \underbrace{+\boldsymbol{v}_t^{\text{new}}\boldsymbol{k}_t^\intercal}_{\text{write}}$$

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{u}_t\boldsymbol{k}_t^\intercal$$

$$\mathbf{O} = \left(\mathbf{Q}\mathbf{K}^\intercal \odot \mathbf{M}\right)\mathbf{U}$$

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

DeltaNet Issue

$$\boldsymbol{v}_t^{\text{old}} = \mathbf{S}_{t-1}\boldsymbol{k}_t$$
$$\boldsymbol{v}_t^{\text{new}} = \beta_t \boldsymbol{v}_t + (1-\beta_t)\,\boldsymbol{v}_t^{\text{old}}$$

$$\rightarrow \quad \boldsymbol{u}_t = \boldsymbol{v}_t^{\text{new}} - \boldsymbol{v}_t^{\text{old}}$$

$$\mathbf{S}_t = \mathbf{S}_{t-1} \underbrace{-\boldsymbol{v}_t^{\text{old}}\boldsymbol{k}_t^{\intercal}}_{\text{remove}} \underbrace{+\boldsymbol{v}_t^{\text{new}}\boldsymbol{k}_t^{\intercal}}_{\text{write}}$$

$$\rightarrow \quad \mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{u}_t\boldsymbol{k}_t^{\intercal}$$

$$\mathbf{O} = \left(\mathbf{Q}\mathbf{K}^{\intercal} \odot \mathbf{M}\right)\mathbf{U}$$

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

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

Parallelizing DeltaNet

$$\boldsymbol{v}_t^{\text{old}} = \mathbf{S}_{t-1}\boldsymbol{k}_t$$
$$\boldsymbol{v}_t^{\text{new}} = \beta_t \boldsymbol{v}_t + (1-\beta_t)\,\boldsymbol{v}_t^{\text{old}}$$

$$\rightarrow \quad \boldsymbol{u}_t = \boldsymbol{v}_t^{\text{new}} - \boldsymbol{v}_t^{\text{old}}$$

$$\mathbf{S}_t = \mathbf{S}_{t-1} \underbrace{-\boldsymbol{v}_t^{\text{old}}\boldsymbol{k}_t^{\top}}_{\text{remove}} \underbrace{+\boldsymbol{v}_t^{\text{new}}\boldsymbol{k}_t^{\top}}_{\text{write}}$$

$$\rightarrow \quad \mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{u}_t\boldsymbol{k}_t^{\top}$$

$$\mathbf{O} = \left(\mathbf{Q}\mathbf{K}^{\top} \odot \mathbf{M}\right)\mathbf{U}$$

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

If there is an efficient way to compute U, we would be good to go!

Parallelizing DeltaNet: A Simple Reparameterization

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \boldsymbol{v}_t^{\text{old}} \boldsymbol{k}_t^{\intercal} + \boldsymbol{v}_t^{\text{new}} \boldsymbol{k}_t^{\intercal}$$

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

Parallelizing DeltaNet: A Simple Reparameterization

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

Parallelizing DeltaNet: A Simple Reparameterization

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \boldsymbol{v}_t^{\text{old}} \boldsymbol{k}_t^\intercal + \boldsymbol{v}_t^{\text{new}} \boldsymbol{k}_t^\intercal$$

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

$$= \sum_{i=1}^{t} \beta_i (\boldsymbol{v}_i \boldsymbol{k}_i^\intercal) \left(\boxed{\prod_{j=i+1}^{t} (\mathbf{I} - \beta_j \boldsymbol{k}_j \boldsymbol{k}_j^\intercal)} \right)$$

Product of generalized Householder matrices.

Parallelizing DeltaNet: Memory-efficient Representation

THE WY REPRESENTATION FOR PRODUCTS OF HOUSEHOLDER MATRICES*

CHRISTIAN BISCHOF† AND CHARLES VAN LOAN†

$$\mathbf{P}_n = \prod_{t=1}^{n}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) \quad \longrightarrow \quad \mathbf{P}_n = \mathbf{I} - \textstyle\sum_{t=1}^{n} \boldsymbol{w}_t \boldsymbol{k}_t^\top$$

$$\mathbf{S}_n = \mathbf{S}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top) + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^\top \quad \longrightarrow \quad \mathbf{S}_n = \sum_{t=1}^{n} \boldsymbol{u}_t \boldsymbol{k}_n^\top$$

Parallelizing DeltaNet: Memory-efficient Representation

$$\mathbf{P}_n = \prod_{t=1}^{n} (\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top)$$

Parallelizing DeltaNet: Memory-efficient Representation

$$
\begin{aligned}
\mathbf{P}_n &= \prod_{t=1}^{n} (\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) \\
&= \mathbf{P}_{n-1} (\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top)
\end{aligned}
$$

Parallelizing DeltaNet: Memory-efficient Representation

$$
\begin{aligned}
\mathbf{P}_n &= \prod_{t=1}^{n}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) \\
&= \mathbf{P}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top) \\
&= (\mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\top)(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top)
\end{aligned}
$$

Parallelizing DeltaNet: Memory-efficient Representation

$$\begin{aligned}
\mathbf{P}_n &= \prod_{t=1}^{n}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) \\
&= \mathbf{P}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top) \\
&= (\mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\top)(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top) \\
&= \mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\top - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top + (\sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\top)\beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top
\end{aligned}$$

Parallelizing DeltaNet: Memory-efficient Representation

$$
\begin{aligned}
\mathbf{P}_n &= \prod_{t=1}^{n}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^{\top}) \\
&= \mathbf{P}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^{\top}) \\
&= (\mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^{\top})(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^{\top}) \\
&= \mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^{\top} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^{\top} + (\sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^{\top}) \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^{\top} \\
&= \mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^{\top} - \underbrace{\left(\beta_n \boldsymbol{k}_n - \beta_n \sum_{t=1}^{n-1} \left(\boldsymbol{w}_t (\boldsymbol{k}_t^{\top} \boldsymbol{k}_n)\right)\right)}_{\boldsymbol{w}_n} \boldsymbol{k}_n^{\top}
\end{aligned}
$$

Parallelizing DeltaNet: Memory-efficient Representation

$$
\begin{aligned}
\mathbf{P}_n &= \prod_{t=1}^{n}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\intercal) \\
&= \mathbf{P}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal) \\
&= (\mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal)(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal) \\
&= \mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal + (\sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal)\beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal \\
&= \mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal - \underbrace{\left(\beta_n \boldsymbol{k}_n - \beta_n \sum_{t=1}^{n-1} \left(\boldsymbol{w}_t (\boldsymbol{k}_t^\intercal \boldsymbol{k}_n)\right)\right)}_{\boldsymbol{w}_n} \boldsymbol{k}_n^\intercal \\
&= \mathbf{I} - \sum_{t=1}^{n} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal
\end{aligned}
$$

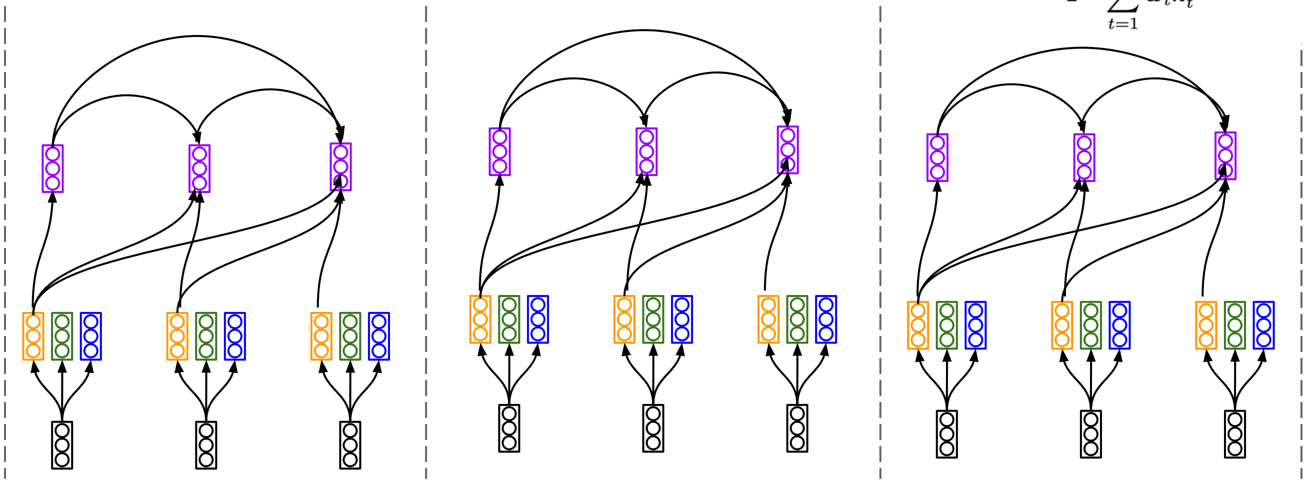
Parallelizing DeltaNet: Memory-efficient Representation

$$
\begin{aligned}
\mathbf{S}_n &= \mathbf{S}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top) + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^\top \\
&= \left(\sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\top\right)(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top) + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^\top \\
&= \sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\top - \left(\sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\top\right) \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\top + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^\top \\
&= \sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\top + \underbrace{\left(\beta_n \boldsymbol{v}_n - \beta_n \sum_{t=1}^{n-1} \boldsymbol{u}_t \left(\boldsymbol{k}_t^\top \boldsymbol{k}_n\right)\right)}_{\boldsymbol{u}_n} \boldsymbol{k}_n^\top \\
&= \sum_{t=1}^{n} \boldsymbol{u}_t \boldsymbol{k}_n^\top
\end{aligned}
$$

Parallelizing DeltaNet: Chunkwise Parallel form

Recurrent W construction

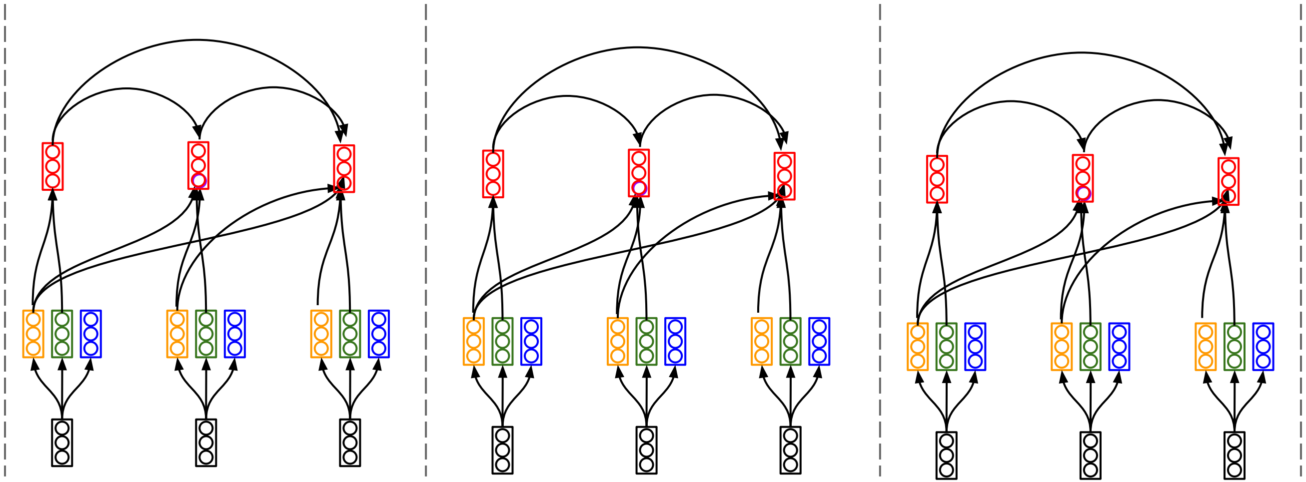
$$
\begin{aligned}
\mathbf{P}_n &= \prod_{t=1}^{n}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\intercal) \\
&= \mathbf{P}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal) \\
&= (\mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal)(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal) \\
&= \mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal + (\sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal)\beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal \\
&= \mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal - \underbrace{\left(\beta_n \boldsymbol{k}_n - \beta_n \sum_{t=1}^{n-1} \left(\boldsymbol{w}_t (\boldsymbol{k}_t^\intercal \boldsymbol{k}_n)\right)\right)}_{\boldsymbol{w}_n} \boldsymbol{k}_n^\intercal \\
&= \mathbf{I} - \sum_{t=1}^{n} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal
\end{aligned}
$$



Parallelizing DeltaNet: Chunkwise Parallel form

$$
\begin{aligned}
\mathbf{S}_n &= \mathbf{S}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal) + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^\intercal \\
&= \left(\sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\intercal\right)(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal) + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^\intercal \\
&= \sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\intercal - \left(\sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\intercal\right) \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^\intercal \\
&= \sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\intercal + \underbrace{\left(\beta_n \boldsymbol{v}_n - \beta_n \sum_{t=1}^{n-1} \boldsymbol{u}_t \left(\boldsymbol{k}_t^\intercal \boldsymbol{k}_n\right)\right)}_{\boldsymbol{u}_n} \boldsymbol{k}_n^\intercal \\
&= \sum_{t=1}^{n} \boldsymbol{u}_t \boldsymbol{k}_n^\intercal
\end{aligned}
$$

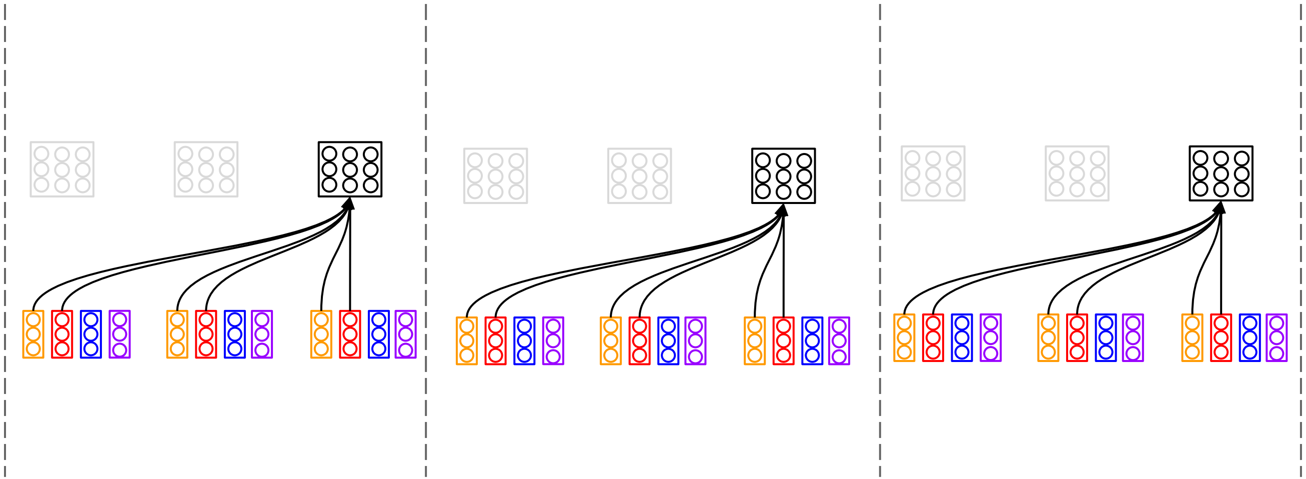
Recurrent U construction



Parallelizing DeltaNet: Chunkwise Parallel form

local state computation $\boldsymbol{U}_{[t]}^{\top} \boldsymbol{K}_{[t]}$

$$\begin{aligned}
\mathbf{S}_n &= \mathbf{S}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^{\top}) + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^{\top} \\
&= \left(\sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^{\top} \right) (\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^{\top}) + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^{\top} \\
&= \sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^{\top} - \left(\sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^{\top} \right) \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^{\top} + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^{\top} \\
&= \sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^{\top} + \underbrace{\left(\beta_n \boldsymbol{v}_n - \beta_n \sum_{t=1}^{n-1} \boldsymbol{u}_t \left(\boldsymbol{k}_t^{\top} \boldsymbol{k}_n \right) \right)}_{\boldsymbol{u}_n} \boldsymbol{k}_n^{\top} \\
&= \sum_{t=1}^{n} \boldsymbol{u}_t \boldsymbol{k}_n^{\top}
\end{aligned}$$

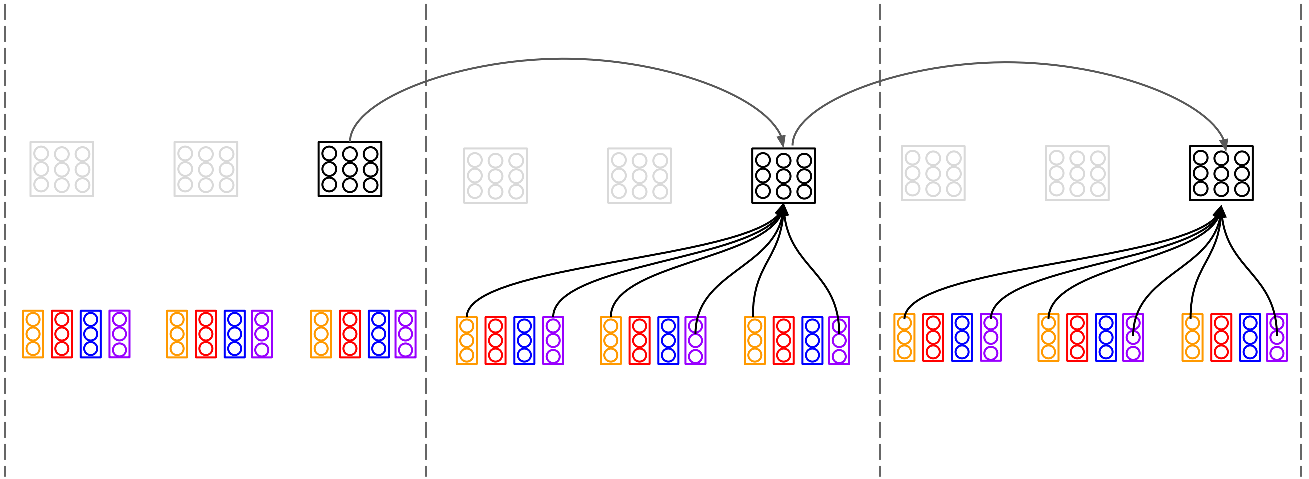


Parallelizing DeltaNet: Chunkwise Parallel form

State passing

$$S_{[2]} = S_{[1]}(I - W_{[2]}^\top K_{[2]}) + U_{[2]}^T K_{[2]}$$
$$= S_{[1]} + (U_{[2]} - S_{[1]} W_{[2]}^\top) K_{[2]}$$

$$S_{[3]} = S_{[2]}(I - W_{[3]}^\top K_{[3]}) + U_{[3]}^T K_{[3]}$$
$$= S_{[2]} + (U_{[3]} - S_{[2]} W_{[3]}^\top) K_{[3]}$$



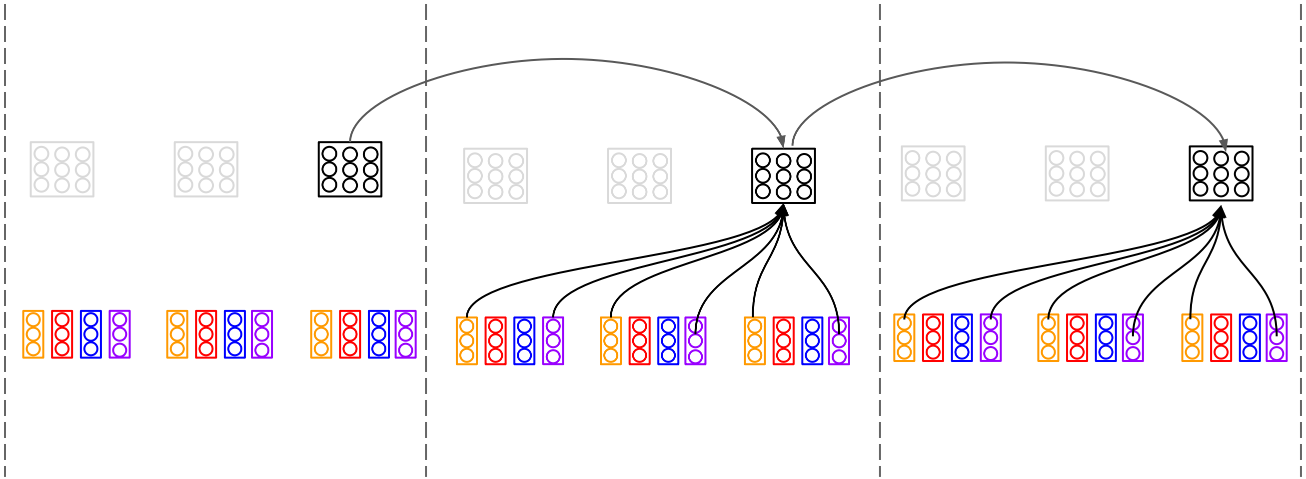
Parallelizing DeltaNet: Chunkwise Parallel form

$$V_{[i+1]}^{\text{new}} = U_{[i+1]} - S_i W_{[i+1]}^{\top}$$

Output computation is the same as vanilla linear attention with new values!

$$\begin{aligned} S_{[2]} &= S_{[1]}(I - W_{[2]}^{\top} K_{[2]}) + U_{[2]}^{T} K_{[2]} \\ &= S_{[1]} + (U_{[2]} - S_{[1]} W_{[2]}^{\top}) K_{[2]} \end{aligned}$$

$$\begin{aligned} S_{[3]} &= S_{[2]}(I - W_{[3]}^{\top} K_{[3]}) + U_{[3]}^{T} K_{[3]} \\ &= S_{[2]} + (U_{[3]} - S_{[2]} W_{[3]}^{\top}) K_{[3]} \end{aligned}$$



Parallelized DeltaNet: Speed

Dimension	Length	Speed-up (vs. recurrent)
64	2048	5.5x
	4096	7.6x
	8192	11.5x
128	2048	8.9x
	4096	13.2x
256	2048	13.7x

On a single H100

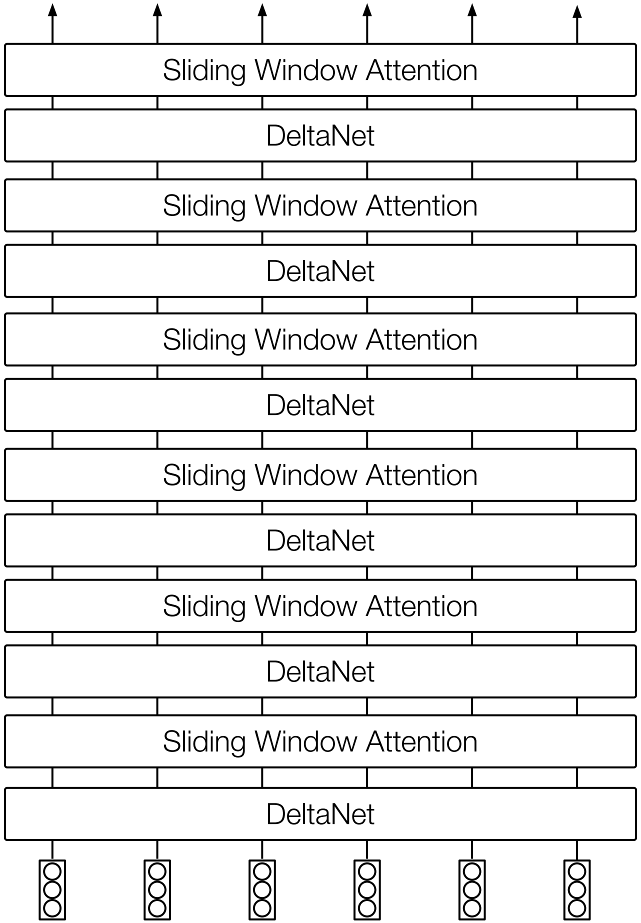
Parallelized DeltaNet: Performance

Model	PPL ↓	LM Eval ↑	Retrieval ↑
Transformer++	16.9	50.9	41.8
RetNet	18.6	48.9	30.6
Mamba	17.1	50.0	27.6
Gated Linear Attention	17.2	51.1	37.7
DeltaNet	16.9	51.6	34.7

1.3B models trained on 100B tokens

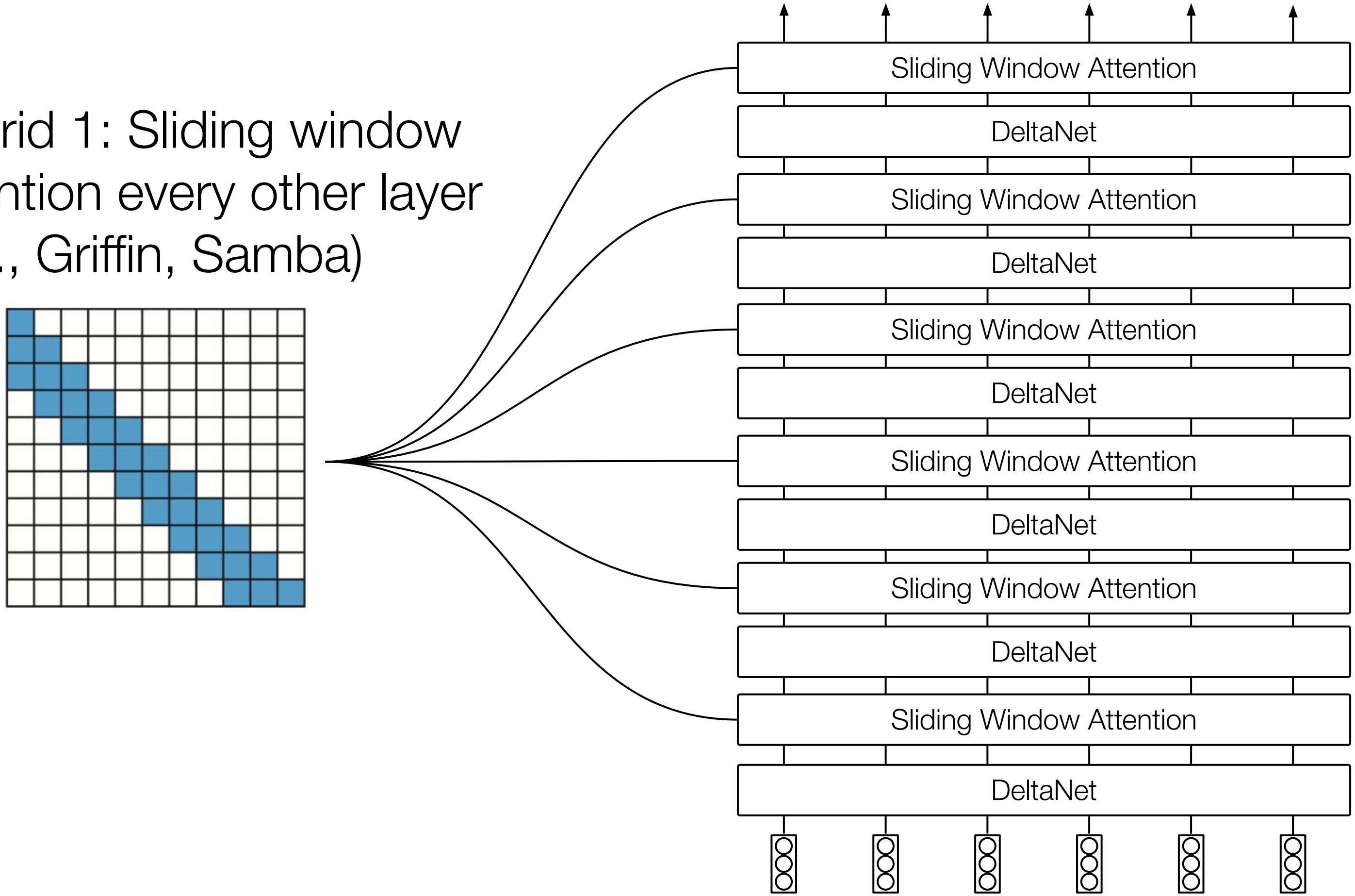
Hybridizing DeltaNet

Hybrid 1: Sliding window attention every other layer



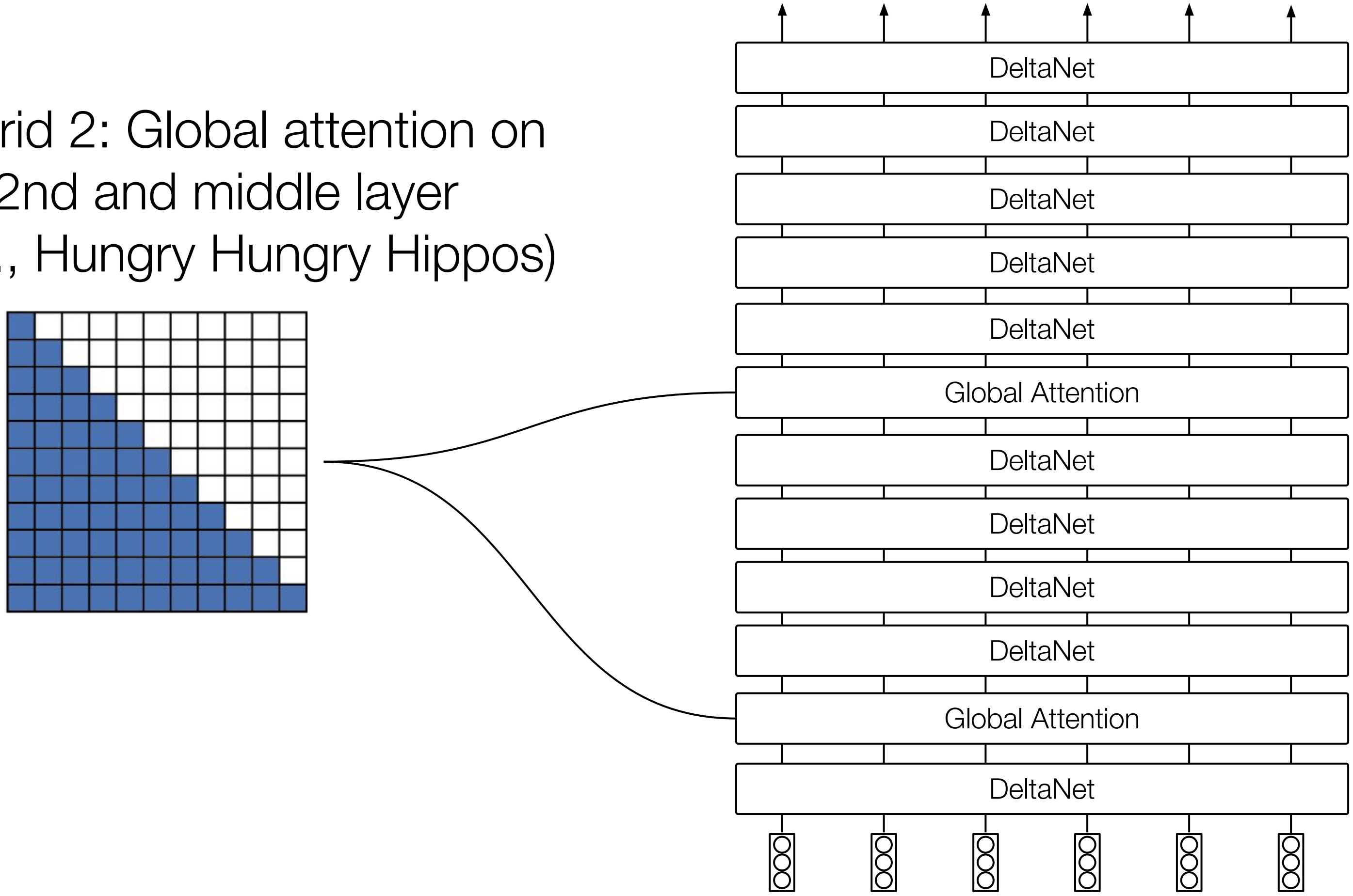
Hybridizing DeltaNet

Hybrid 1: Sliding window attention every other layer (e.g., Griffin, Samba)



Hybridizing DeltaNet

Hybrid 2: Global attention on the 2nd and middle layer (e.g., Hungry Hungry Hippos)



Hybrid DeltaNet: Performance

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

1.3B models trained on 100B tokens

Generalizing Gated Linear Attention / State-Space Models

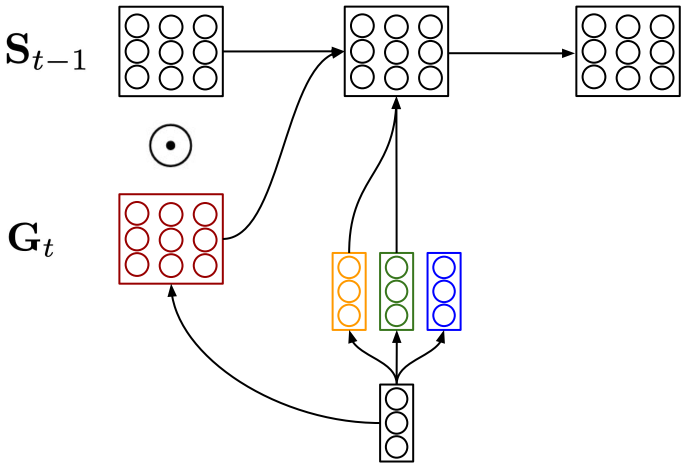
Gated Linear Attention / State-Space Models

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

Recurrence with elementwise product

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

Memory read-out



Generalizing Gated Linear Attention / State-Space Models

Gated Linear Attention / State-Space Models

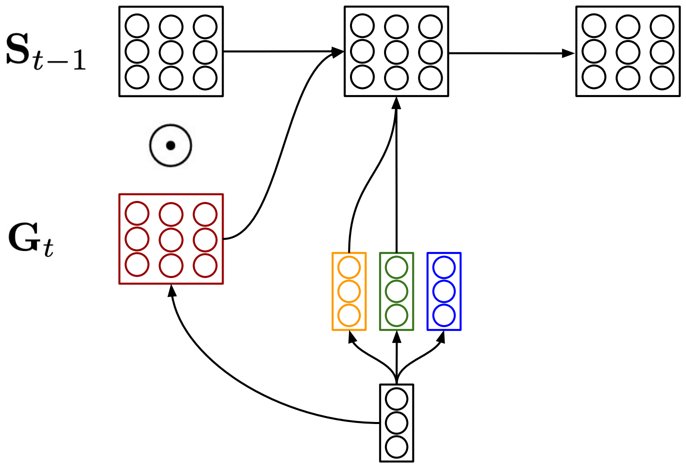
$$\mathbf{S}_t = \boxed{\mathbf{S}_{t-1} \odot \mathbf{G}_t} + \boldsymbol{v}_t \boldsymbol{k}_t^\top$$

Recurrence with elementwise product

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

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

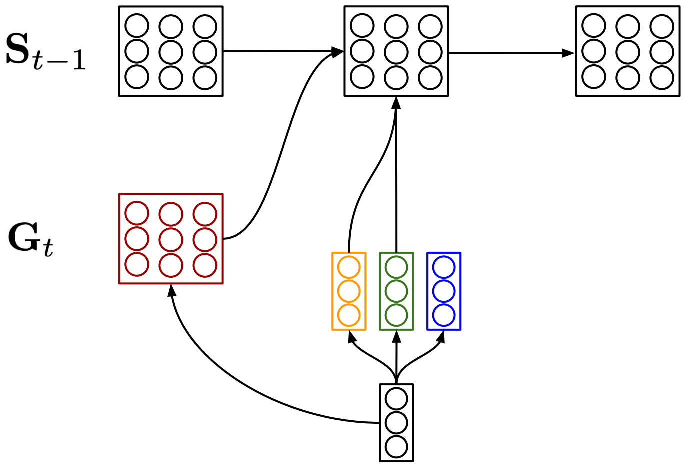
Generalized Linear Transformers

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

Recurrence with matmul

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

Memory read-out



Generalizing Gated Linear Attention / State-Space Models

Generalized Linear Transformers

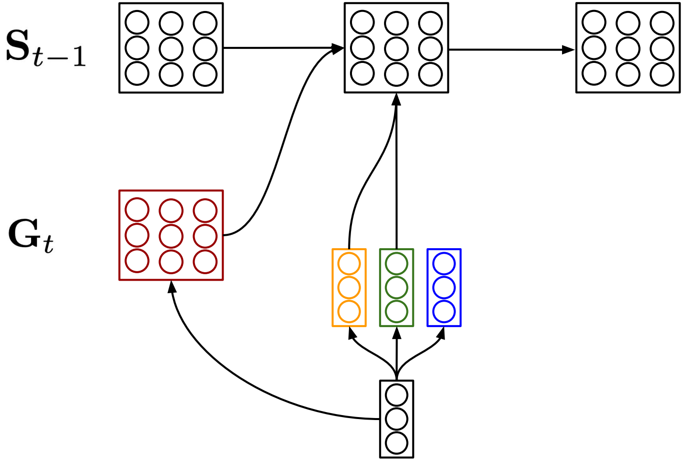
$$\mathbf{S}_t = \boxed{\mathbf{S}_{t-1}\mathbf{G}_t} + \boldsymbol{v}_t\boldsymbol{k}_t^{\top}$$

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

Recurrence with matmul

Memory read-out

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



Generalizing Gated Linear Attention / State-Space Models

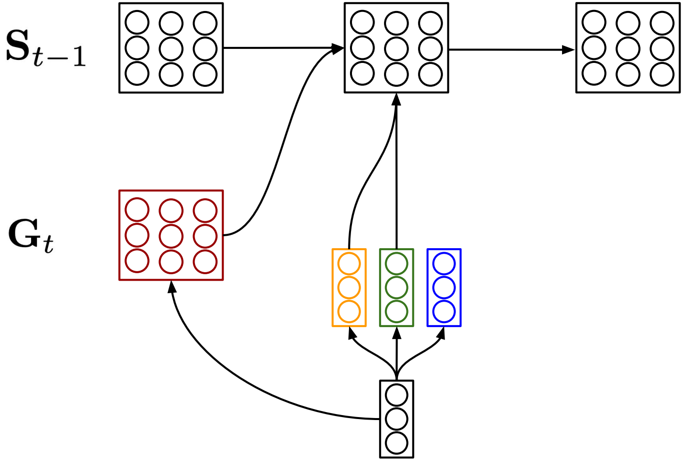
Generalized Linear Transformers with **Structured Matmuls**

$$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \boldsymbol{a}_t \boldsymbol{b}_t^\top) + \boldsymbol{v}_t \boldsymbol{k}_t^\top$$

Recurrence with identity + low-rank

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

Memory read-out



Generalizing Gated Linear Attention / State-Space Models

Generalized Linear Transformers with **Structured Matmuls**

$$\mathbf{S}_t = \boxed{\mathbf{S}_{t-1}(\mathbf{I} - \boldsymbol{a}_t \boldsymbol{b}_t^\top)} + \boldsymbol{v}_t \boldsymbol{k}_t^\top$$

Recurrence with identity + low-rank

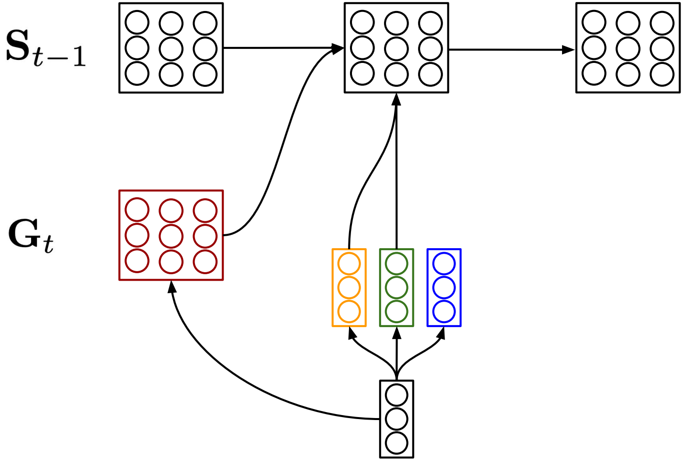
$$\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t$$

Memory read-out

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

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

and is thus a special case.



Generalizing Gated Linear Attention / State-Space Models

Open/Future Work

What about more general associative operators?

$$\mathbf{S}_t = \mathbf{S}_{t-1} \bullet \mathbf{M}_t + \boldsymbol{v}_t \boldsymbol{k}_t^{\mathsf{T}}$$

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!