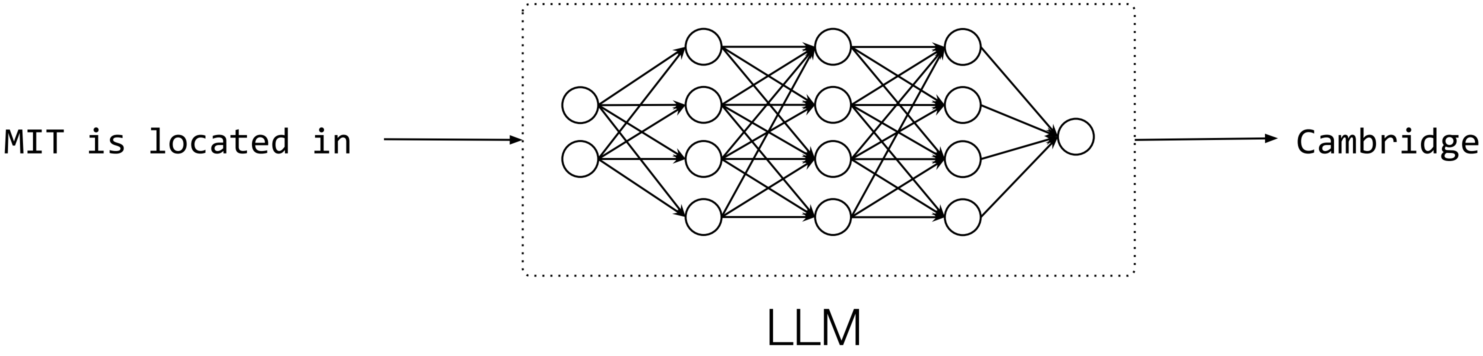
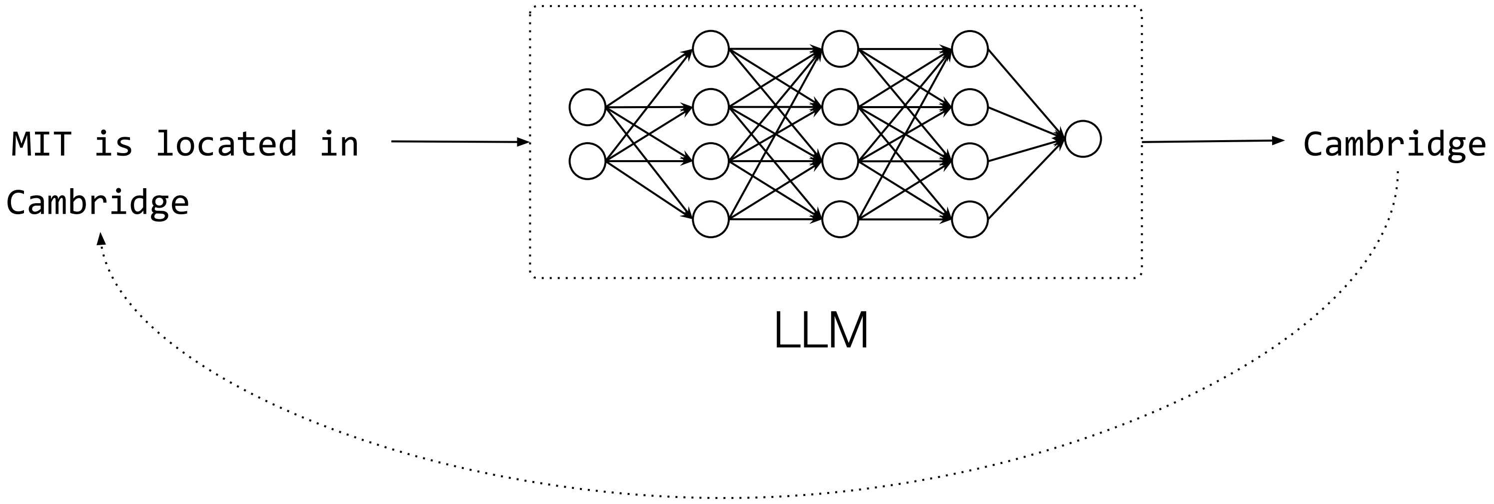
Linear Transformers for Efficient Sequence Modeling

Yoon Kim
MIT

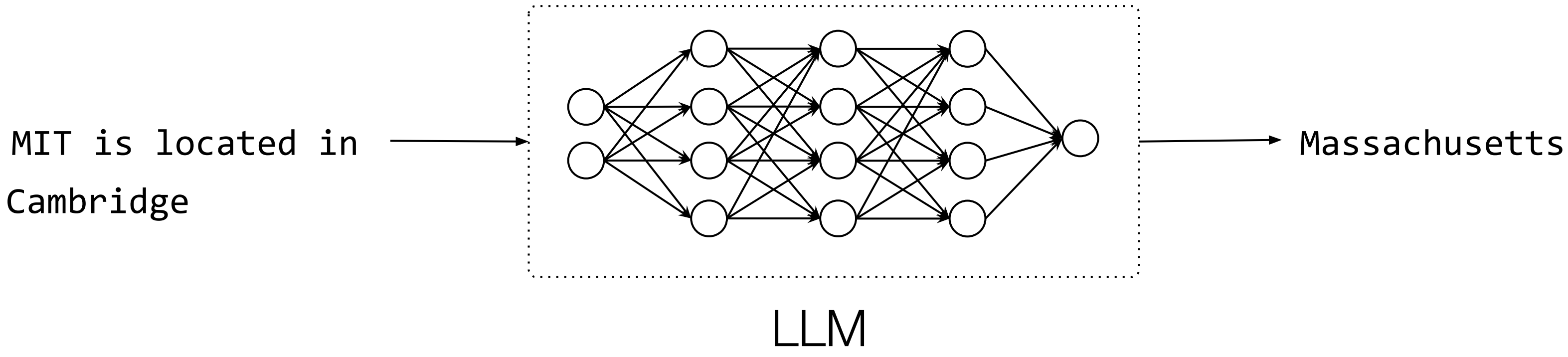
How do large language models work?



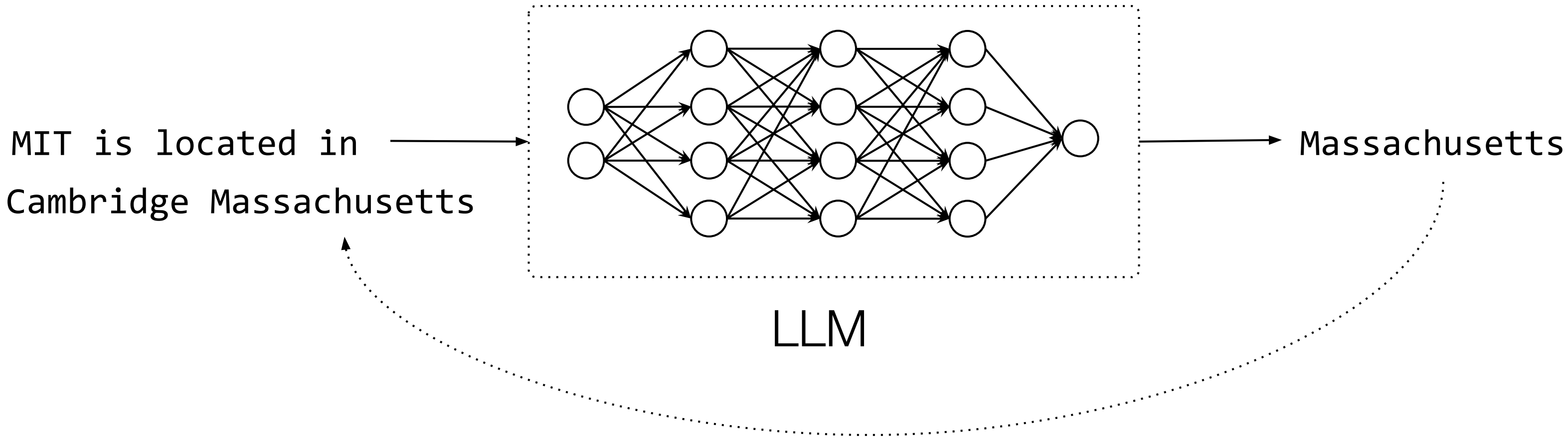
How do large language models work?



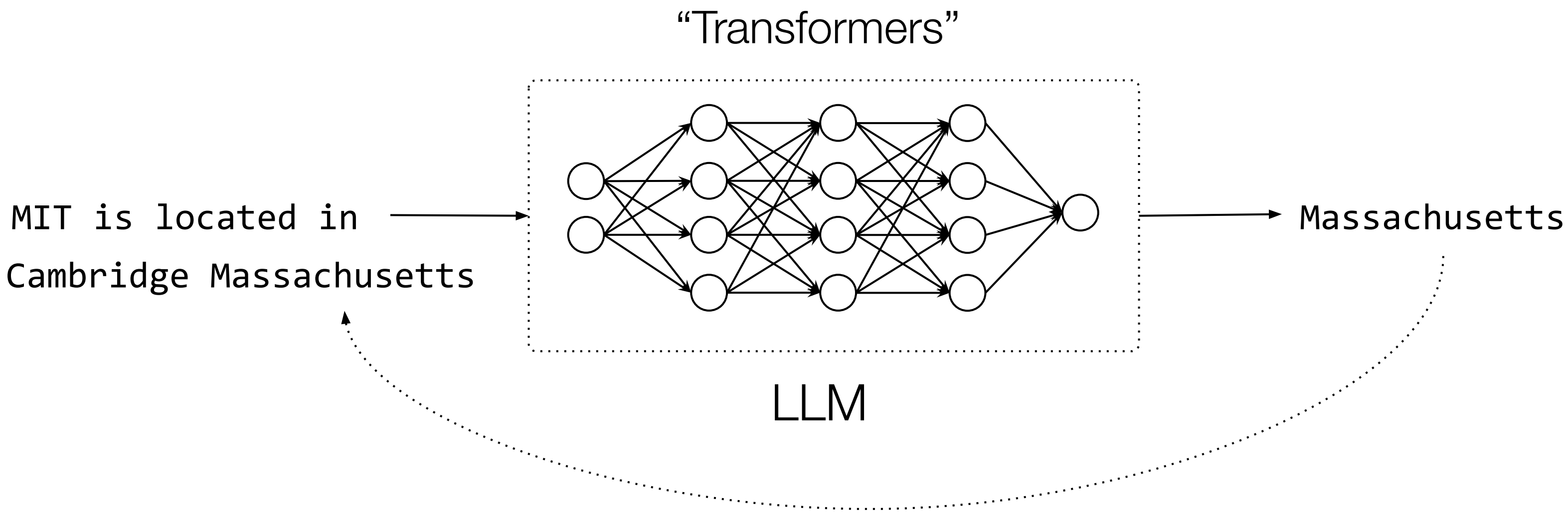
How do large language models work?



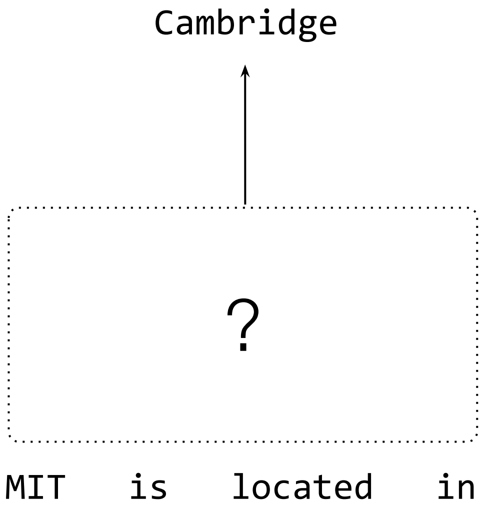
How do large language models work?



How do large language models work?

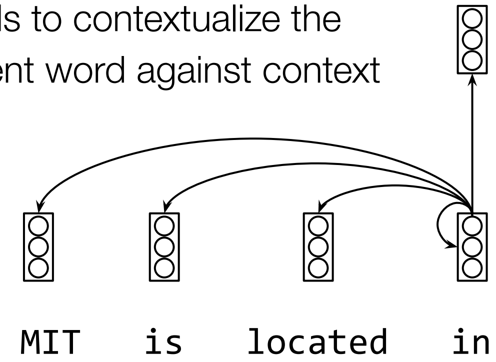


Transformers [Vaswani et al. '17]



Transformers [Vaswani et al. '17]

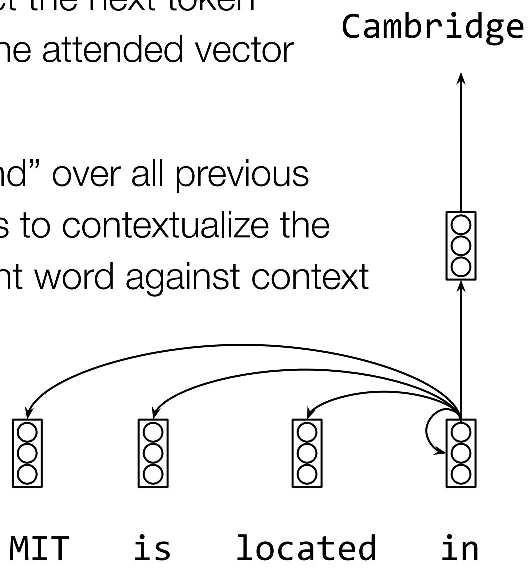
"Attend" over all previous words to contextualize the current word against context



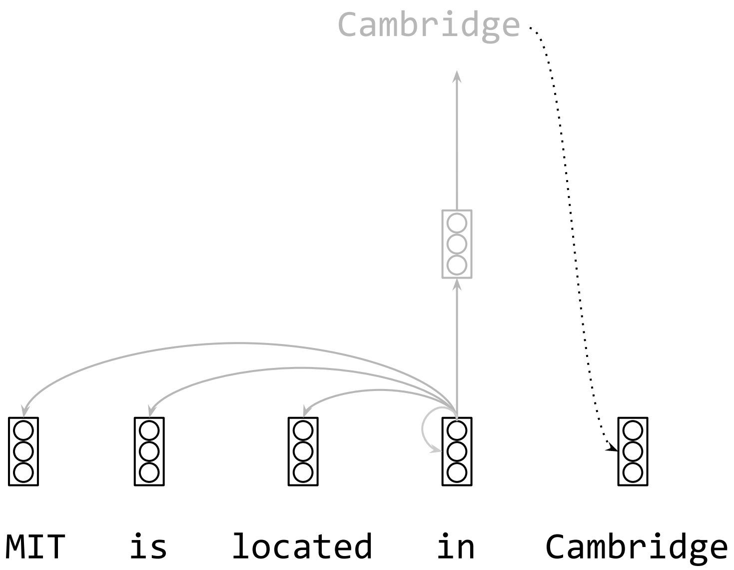
Transformers [Vaswani et al. '17]

Predict the next token
with the attended vector

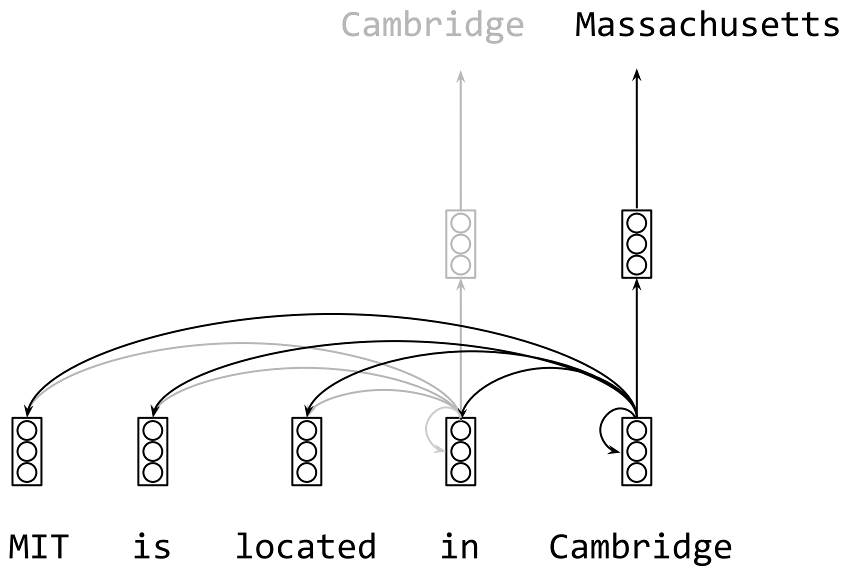
"Attend" over all previous
words to contextualize the
current word against context



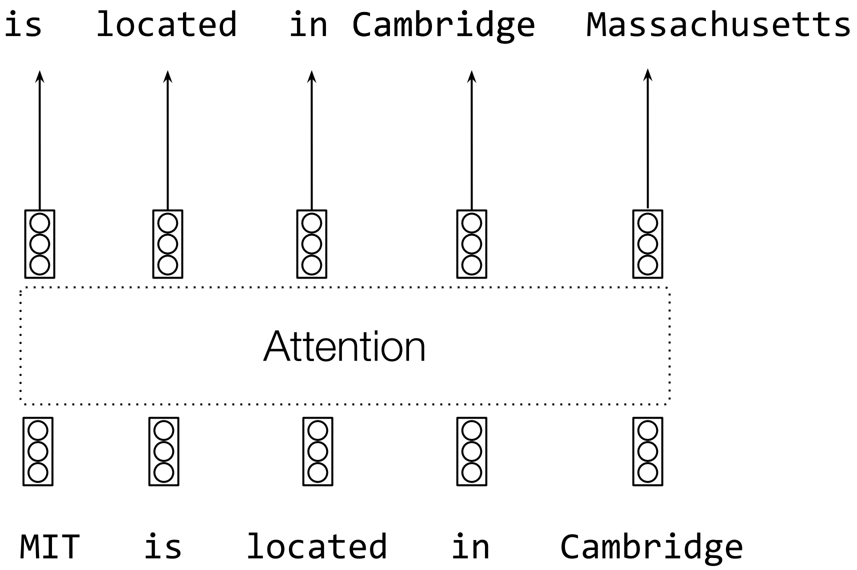
Transformers [Vaswani et al. '17]



Transformers [Vaswani et al. '17]

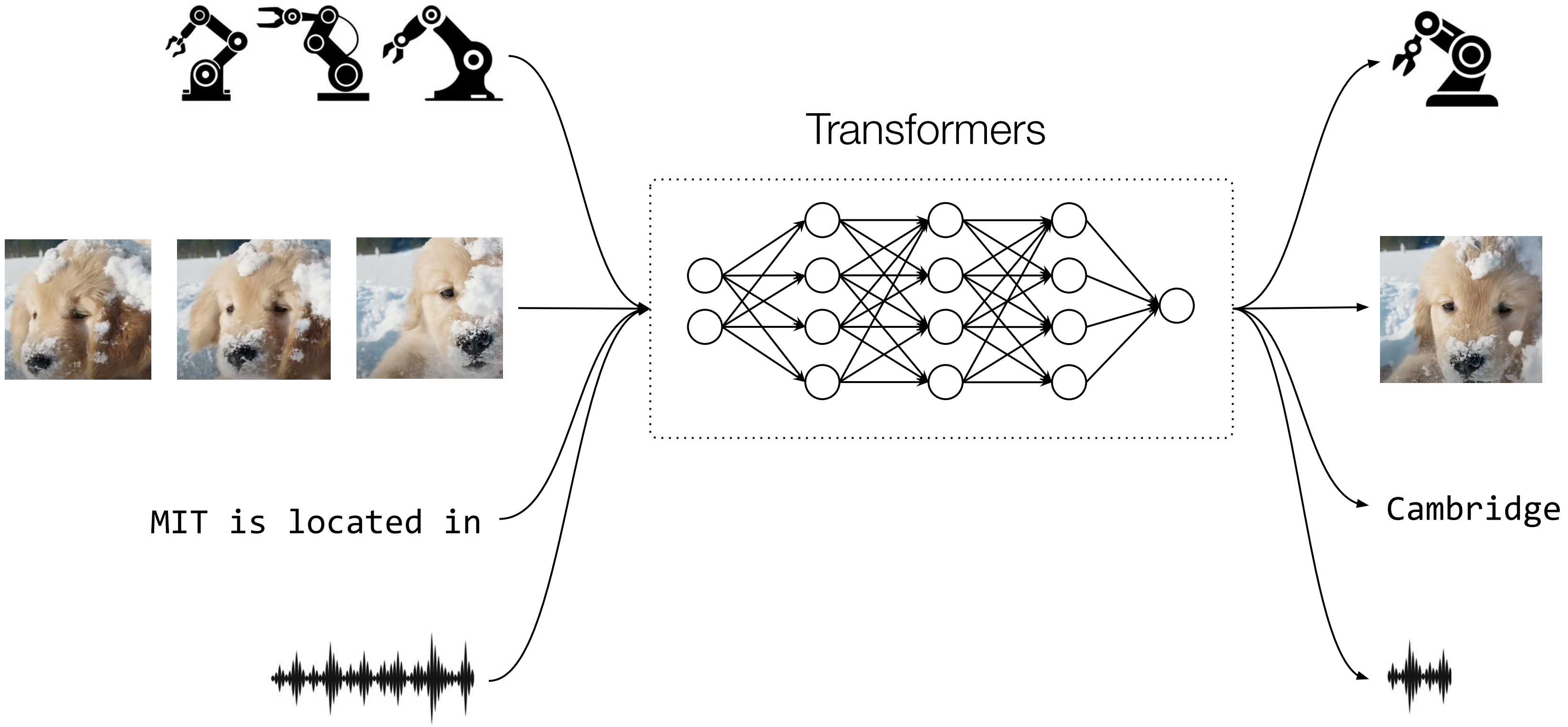


Transformers [Vaswani et al. '17]



Attention can model rich interactions among input elements
→ Important primitive for accurate sequence modeling!

Transformers for Generative AI



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

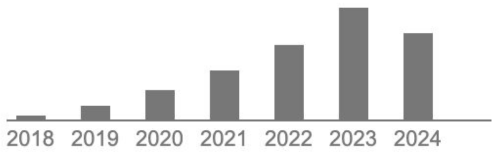
Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

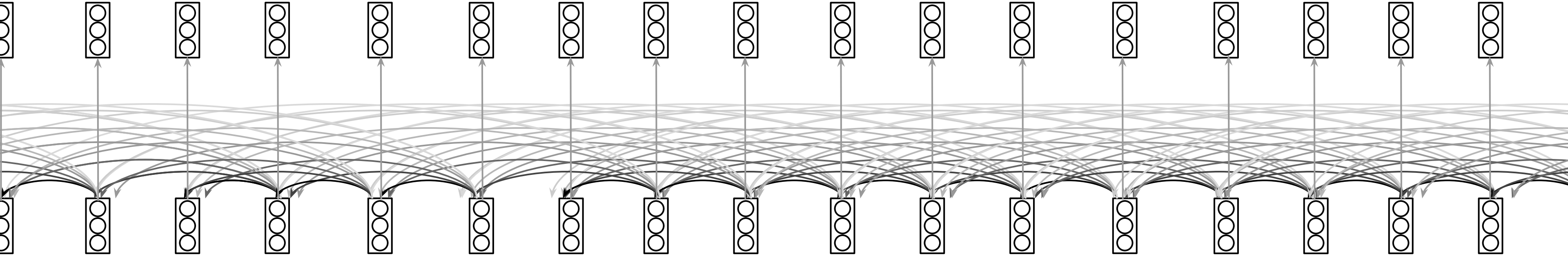
Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Total citations Cited by 136710



Transformers have difficulty scaling to long sequences



Harry Potter series: 1M words

Human DNA: 3.2B nucleotides

How can we maintain the **accuracy** of attention while enabling **efficient** training and inference?

Today: Linear Transformers for Efficient Sequence Modeling

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
NeurIPS '24

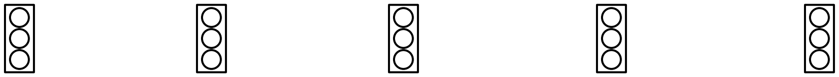
Background: Attention & Linear Attention

Attention: Training

L : sequence length

d : hidden state dimension

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



Attention: Training

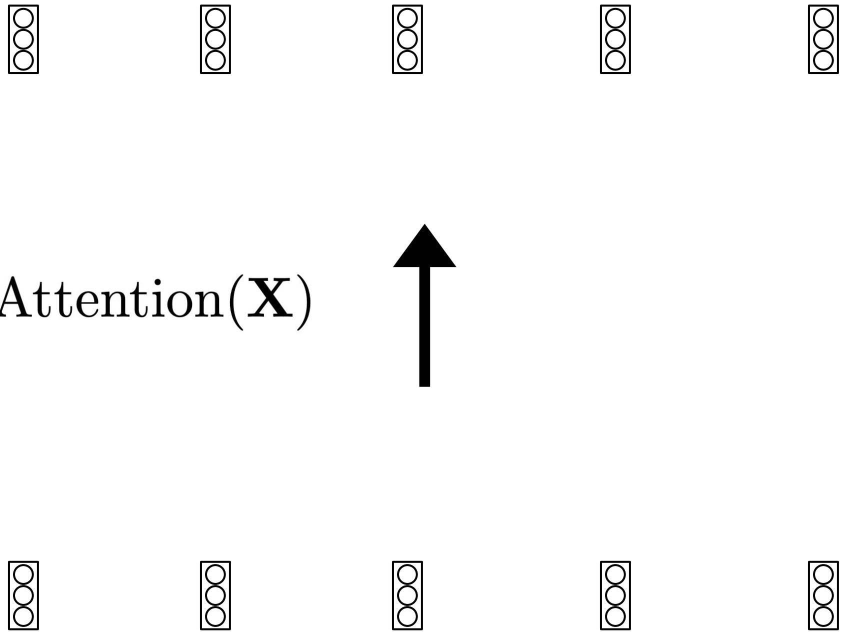
L : sequence length

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

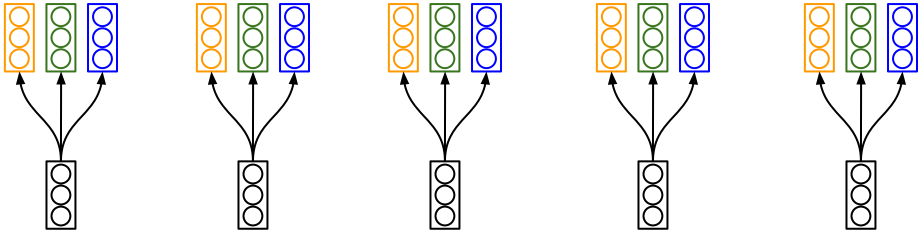
$O(Ld^2)$ $\quad$ $\mathbf{Q},\mathbf{K},\mathbf{V} = \mathbf{X}\boldsymbol{W}_Q, \mathbf{X}\boldsymbol{W}_K, \mathbf{X}\boldsymbol{W}_V$

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

Key K

Value V

Query Q



Attention: Training

L : sequence length

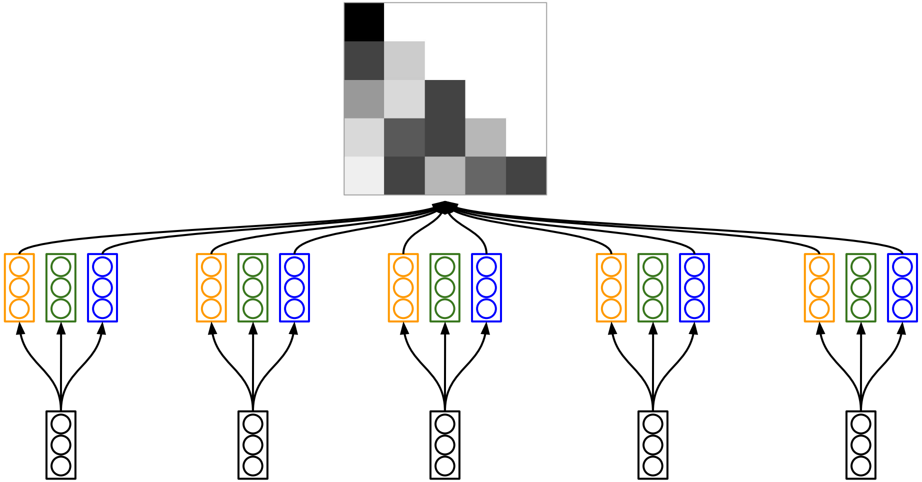
d : hidden state dimension

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

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

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

Key K
Value V
Query Q



Attention: Training

L : sequence length

d : hidden state dimension

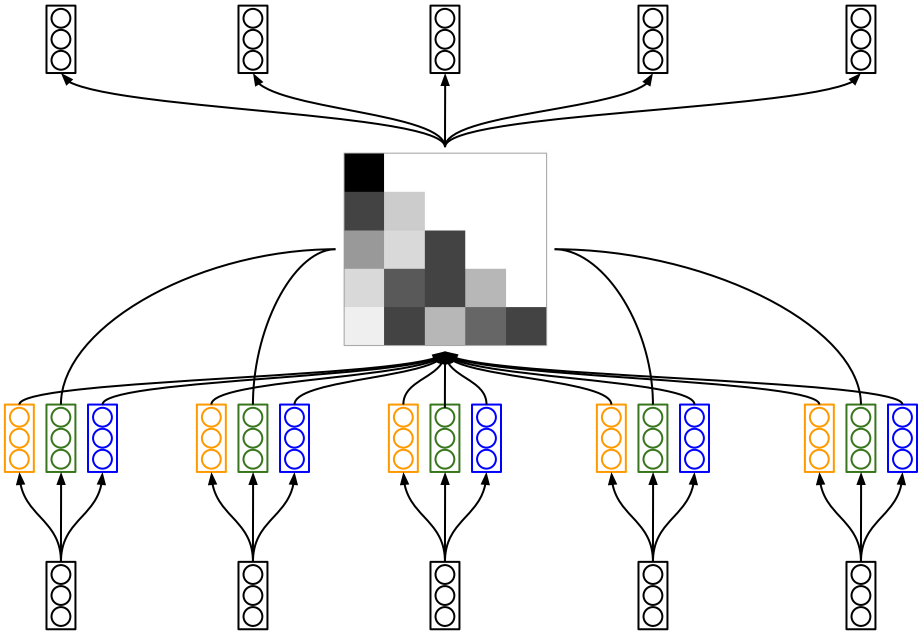
$O(L^2d)$ $\quad \mathbf{O} = \mathbf{A}\mathbf{V} \in \mathbb{R}^{L \times d}$

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

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

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

Key K
Value V
Query Q



Attention: Training

L : sequence length

d : hidden state dimension

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

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

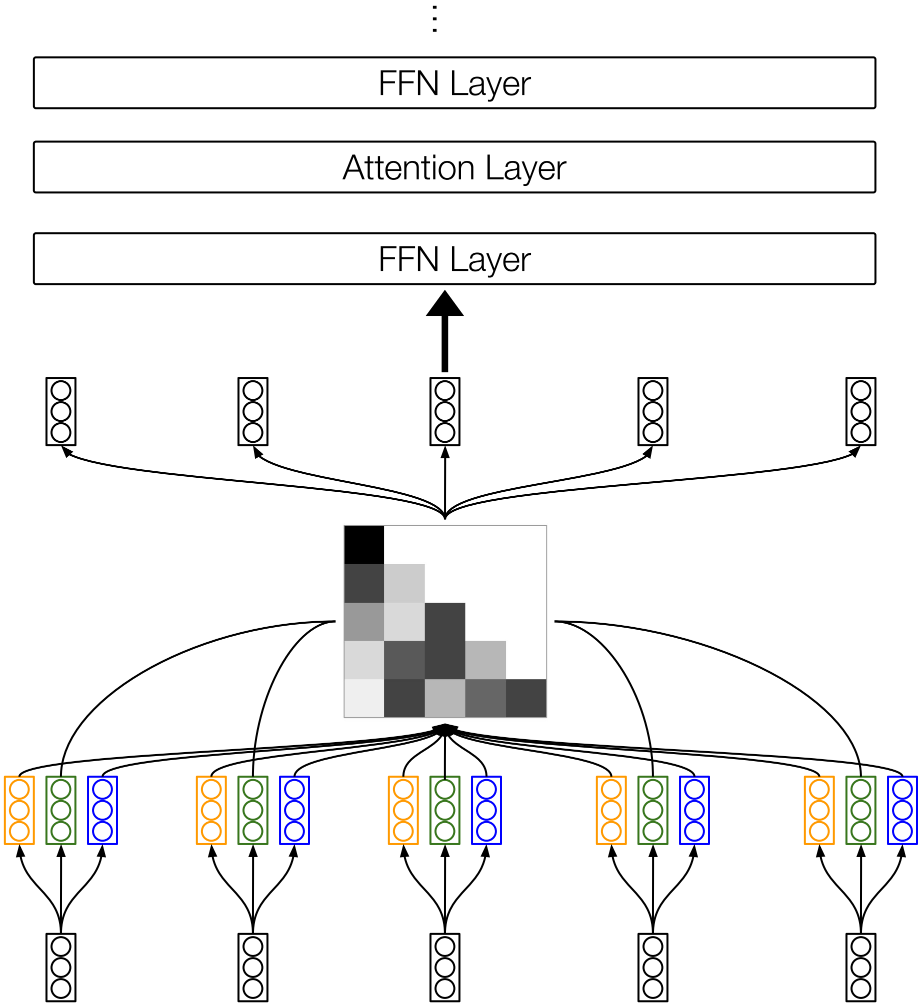
$O(Ld^2)$ $\quad \mathbf{Q}, \mathbf{K}, \mathbf{V} = \mathbf{X}\boldsymbol{W}_Q, \mathbf{X}\boldsymbol{W}_K, \mathbf{X}\boldsymbol{W}_V$

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

Key K

Value V

Query Q



Attention: Training

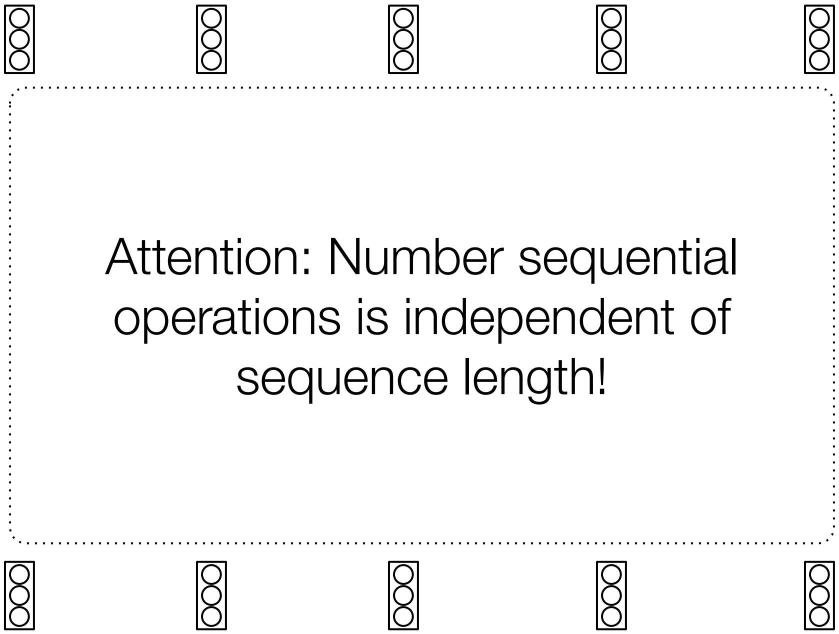
Attention requires $O(L^2 d + L d^2)$ work but can be done in $O(1)$ steps
→ Parallel training that is rich in matmuls.

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

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

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

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

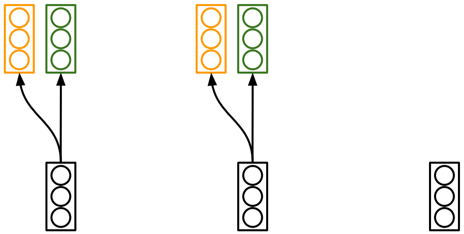


Attention: Training

	Training ("Parallel Form")	
	$\mathbf{O} = \mathrm{softmax}\left((\mathbf{Q}\mathbf{K}^\top) \odot \mathbf{M}\right)\mathbf{V}$	
Compute (Work)	$O(L^2)$	(FLOPs)
Memory	$O(L)$	(GPU memory)
Steps	$O(1)$	(Number of matmuls)

Attention: Generative Inference

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

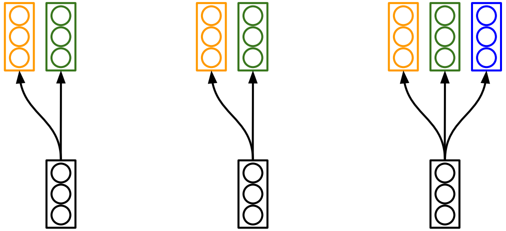


Key K
Value V
Query Q

Attention: Generative Inference

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

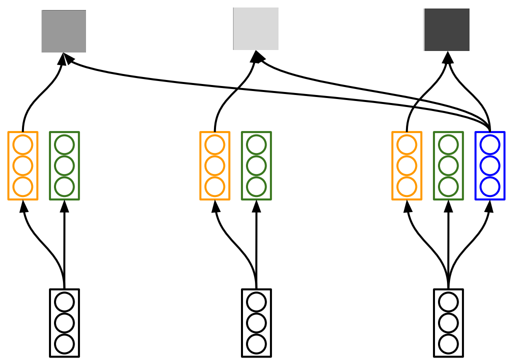
Key K
Value V
Query Q



Attention: Generative Inference

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

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



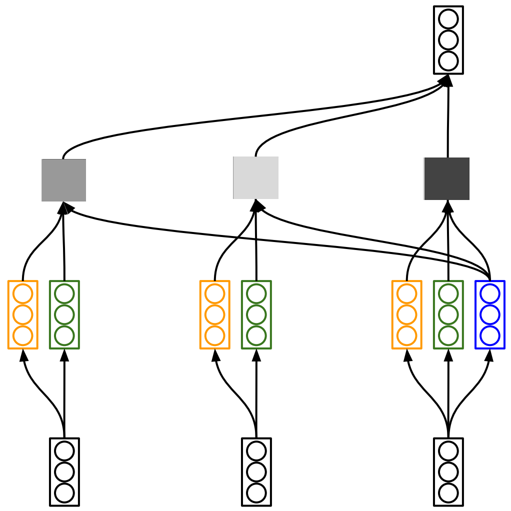
Key K
Value V
Query Q

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

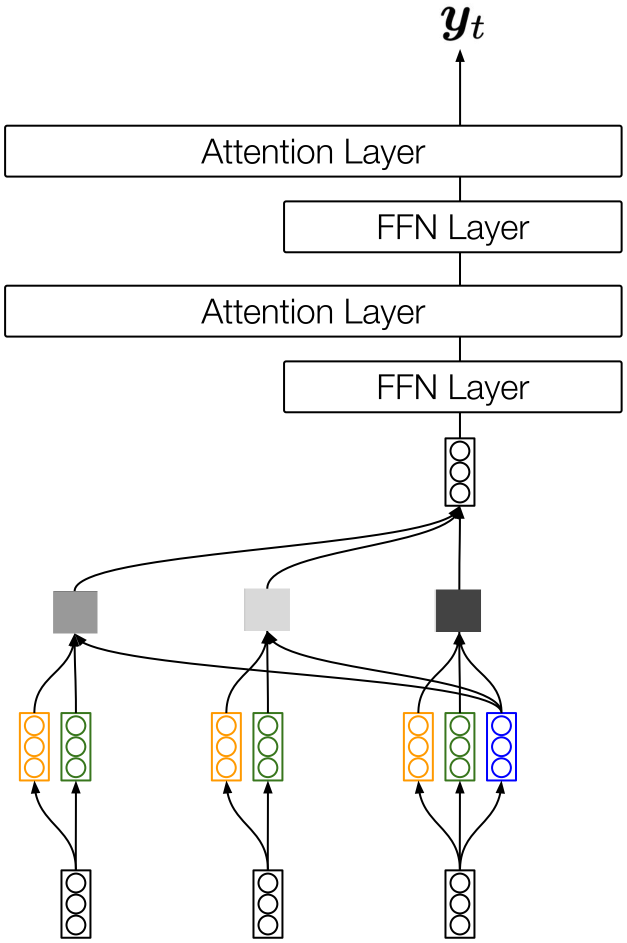


Attention: Generative Inference

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

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

Key K
Value V
Query Q

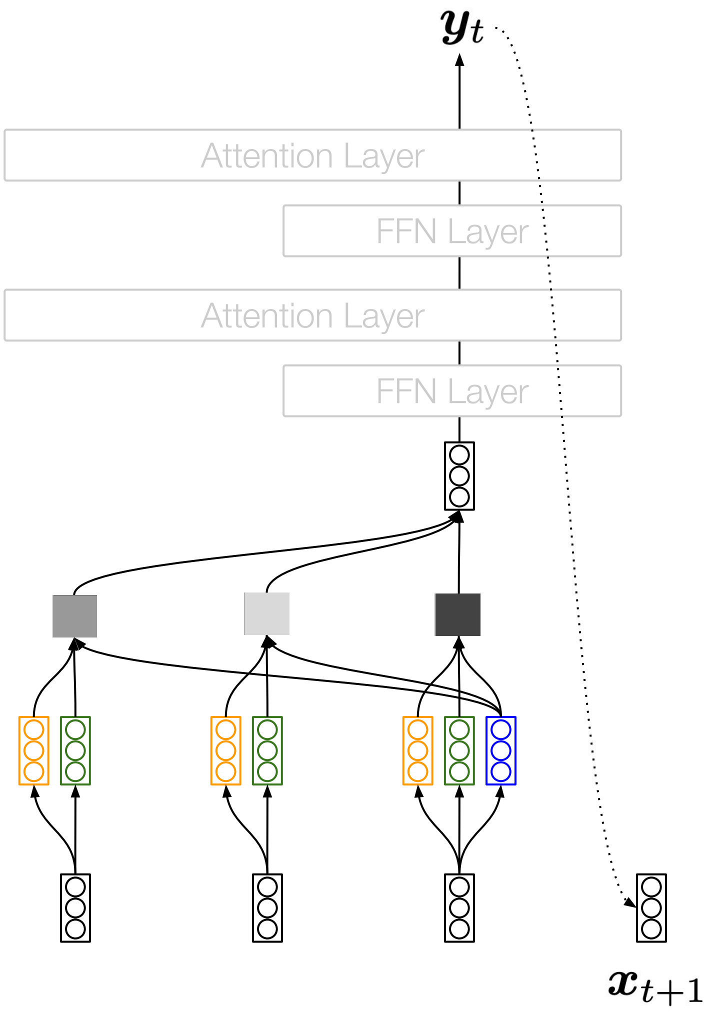


Attention: Generative Inference

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

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

Key	K
Value	V
Query	Q

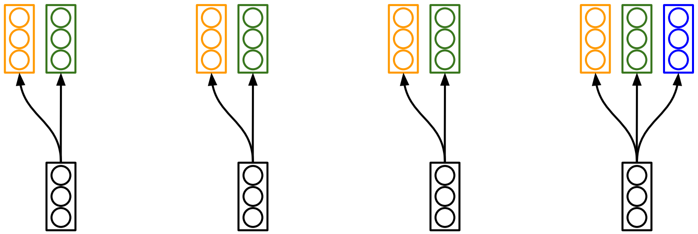


Attention: Generative Inference

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

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

Key K
Value V
Query Q

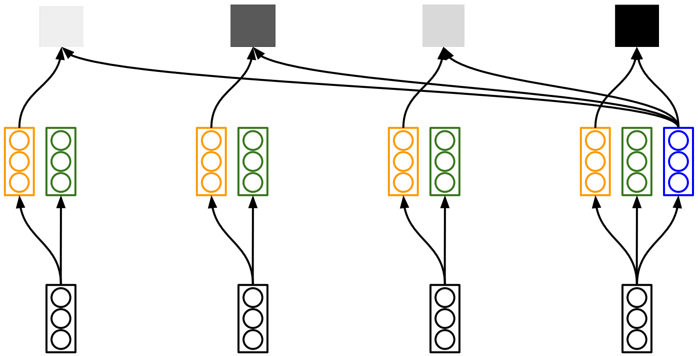


Attention: Generative Inference

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

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

Key	K
Value	V
Query	Q



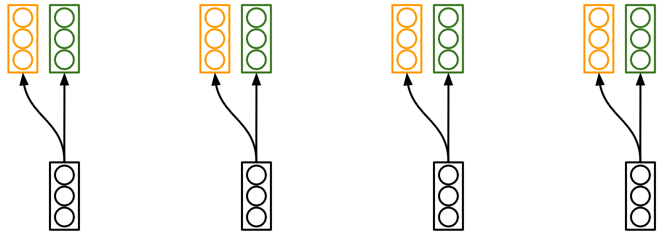
Attention: Generative Inference

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

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

Attention

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

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

- Quadratic compute (bad for training / inference).
- Linear memory (bad for inference).

Linear Attention ("Linear Transformers") [Katharopoulos et al. '20]

Softmax Attention

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

(Simple) Linear Attention

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

Linear Attention ("Linear Transformers") [Katharopoulos et al. '20]

Softmax Attention

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

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

(Simple) Linear Attention

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

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

Linear Attention ("Linear Transformers") [Katharopoulos et al. '20]

	Training ("Parallel Form")
Softmax Attention	$\mathbf{O} = \mathrm{softmax}\big((\mathbf{Q}\mathbf{K}^{\top}) \odot \mathbf{M}\big)\mathbf{V}$
(Simple) Linear Attention	$\mathbf{O} = \big((\mathbf{Q}\mathbf{K}^{\top}) \odot \mathbf{M}\big)\mathbf{V}$

Training: Haven't really gained anything (yet)...

Linear Attention ("Linear Transformers") [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 ("Linear Transformers") [Katharopoulos et al. '20]

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

Linear Attention: Inference

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

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

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

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

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

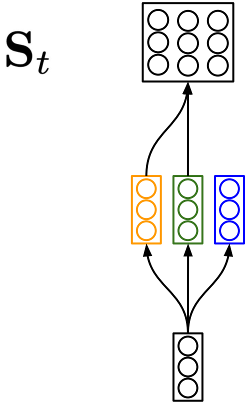
Linear Attention: Inference

Key K
Value V
Query Q

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

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

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



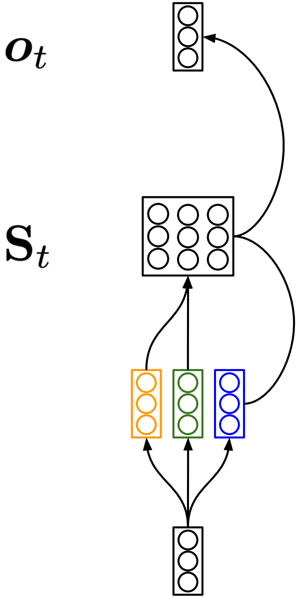
Linear Attention: Inference

Key K
Value V
Query Q

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

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

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



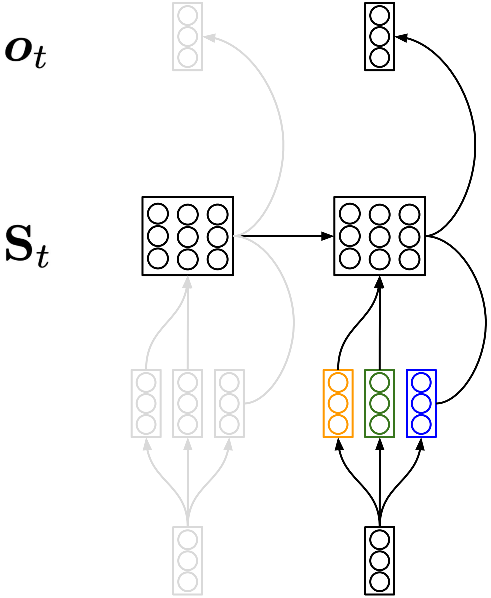
Linear Attention: Inference

Key K
Value V
Query Q

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

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

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



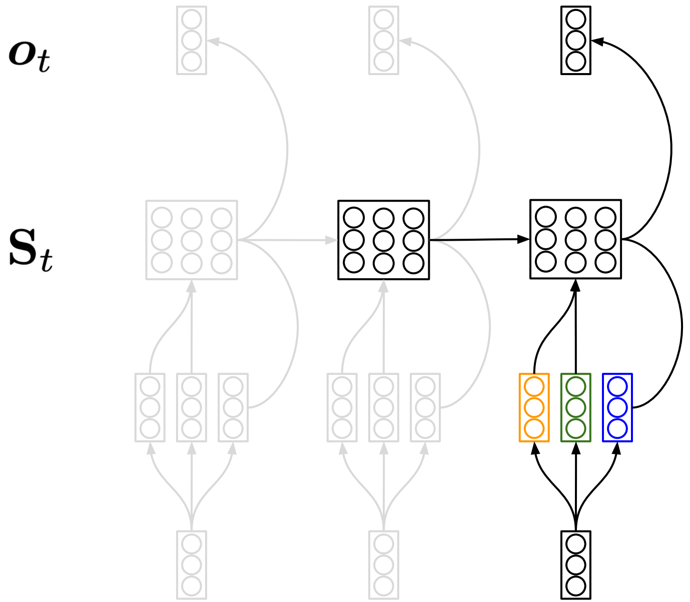
Linear Attention: Inference

Key K
Value V
Query Q

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

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

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



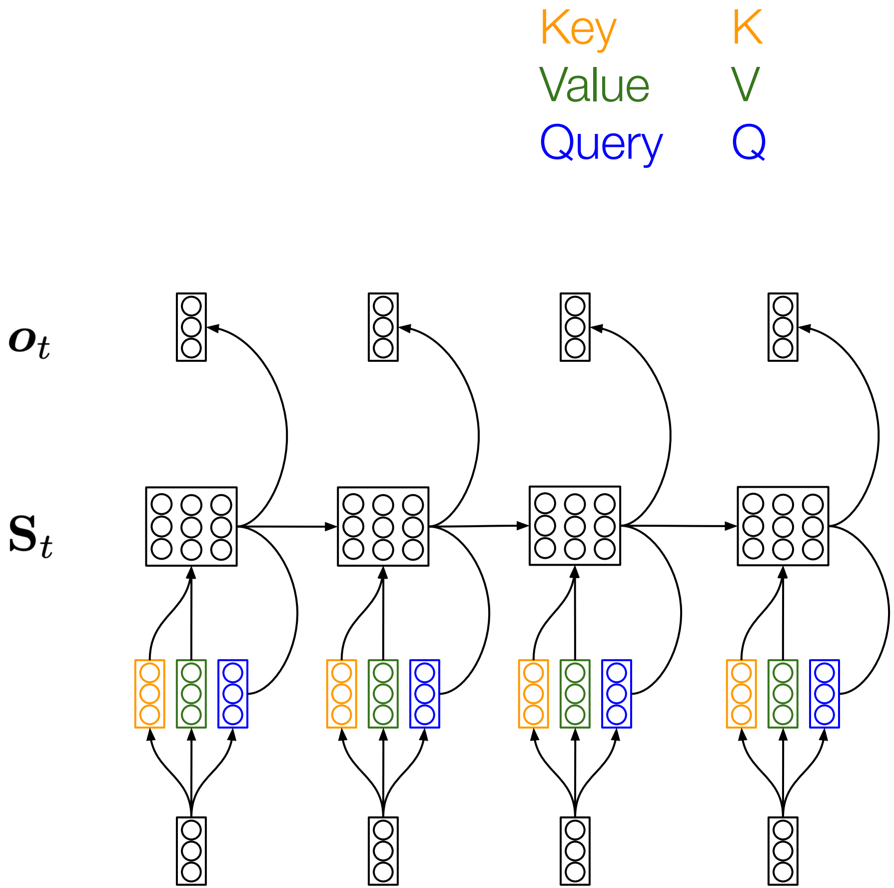
Linear Attention: Inference

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

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

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

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



Linear Transformers are “Fast Weights”!

Using Fast Weights to Deblur Old Memories

Geoffrey E. Hinton and David C. Plaut

Computer Science Department
Carnegie-Mellon University

[Hinton and Plaut ’87]

LEARNING TO CONTROL FAST-WEIGHT MEMORIES: AN ALTERNATIVE TO DYNAMIC RECURRENT NETWORKS

(*Neural Computation*, 4(1):131–139, 1992)

Jürgen Schmidhuber*
Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2, Germany
schmidhu@tumult.informatik.tu-muenchen.de

[Schmidhuber ’92]

A “slow network” changes the weights of a “fast network”

$$\boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)} = \boldsymbol{W}_a \boldsymbol{x}^{(i)}, \boldsymbol{W}_b \boldsymbol{x}^{(i)}$$

$$\boxed{\boldsymbol{W}^{(i)}} = \sigma\left(\boldsymbol{W}^{(i-1)} + \boldsymbol{a}^{(i)} \otimes \boldsymbol{b}^{(i)}\right)$$

$$\boldsymbol{y}^{(i)} = \boldsymbol{W}^{(i)} \boldsymbol{x}^{(i)}$$

Linear Attention

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

Why not use the recurrent form for training?

Linear Attention: Naive Parallel Form

	Training ("Parallel Form") $\mathbf{O} = ((\mathbf{Q}\mathbf{K}^\top) \odot \mathbf{M})\mathbf{V}$	Inference ("Recurrent Form") $\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top \quad \boldsymbol{o}_t = \boldsymbol{q}_t^\top \mathbf{S}_t$
Compute	$O(L^2)$ 😐	$O(L)$
Memory	$O(L)$	$O(1)$ 🙂
Steps	$O(1)$	$O(L)$ 😐

- Strict sequential computation (no sequence-level parallelism).

Linear Attention: Naive Parallel Form

	Training ("Parallel Form")	Inference ("Recurrent Form")	
	$\mathbf{O} = ((\mathbf{Q}\mathbf{K}^\top) \odot \mathbf{M})\mathbf{V}$	😐 $\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top$	$\boldsymbol{o}_t = \boldsymbol{q}_t^\top \mathbf{S}_t$
Compute	$O(L^2)$ 😐	$O(L)$	
Memory	$O(L)$	$O(1)$ 🙂	
Steps	$O(1)$	$O(L)$ 😐	

- Strict sequential computation (no sequence-level parallelism).
- All operations are either elementwise operations or reductions → cannot leverage tensor cores.

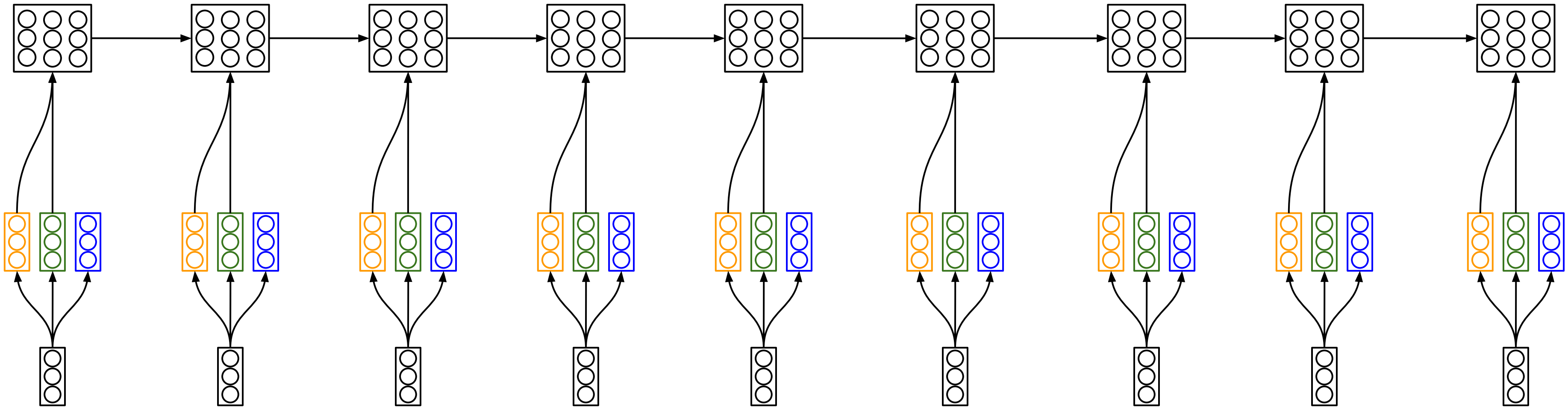
Linear Attention: Naive Parallel Form

	Training ("Parallel Form")	Inference ("Recurrent Form")
	$\mathbf{O} = ((\mathbf{Q}\mathbf{K}^{\top}) \odot \mathbf{M})\mathbf{V}$	😐 $\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^{\top}$ $\quad \boldsymbol{o}_t = \boldsymbol{q}_t^{\top}\mathbf{S}_t$ 😐
Compute	$O(L^2)$ 😐	$O(L)$
Memory	$O(L)$	$O(1)$ 🙂
Steps	$O(1)$	$O(L)$ 😐

- Strict sequential computation (no sequence-level parallelism).
- All operations are either elementwise operations or reductions → cannot leverage tensor cores.
- Materialization of each time step's hidden states → High I/O cost.

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

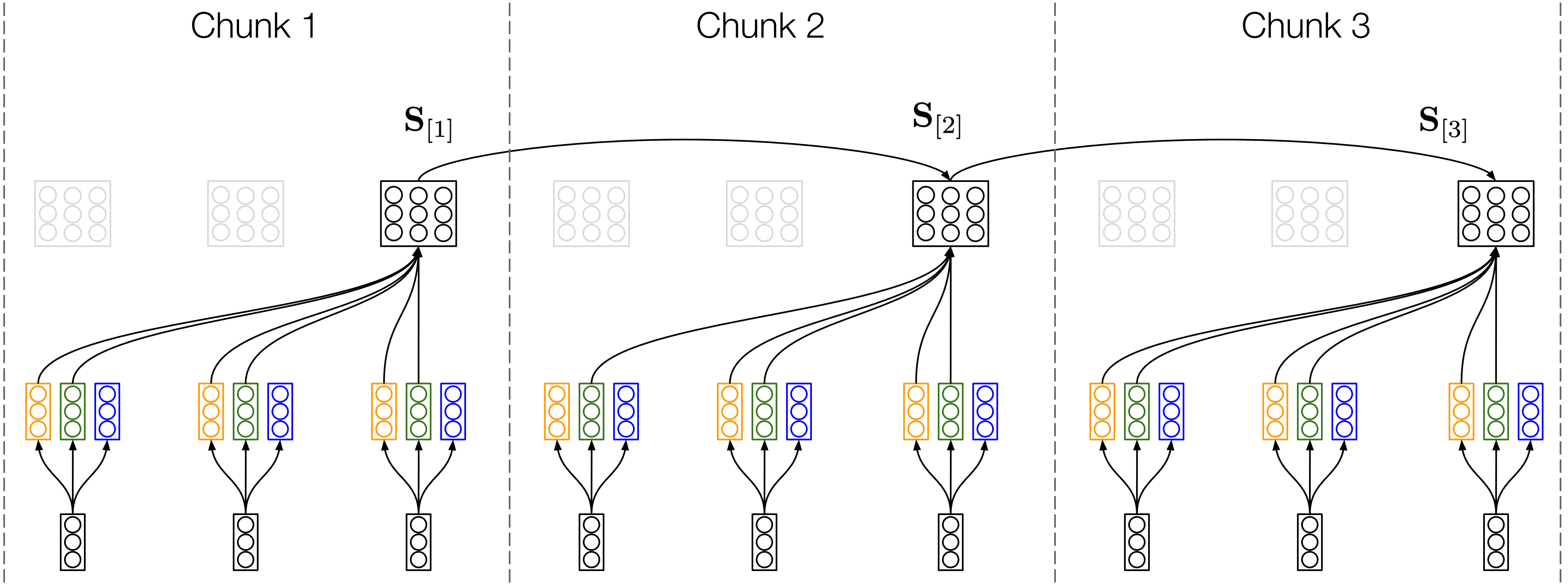
Pure RNN ·



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

Pure RNN → "Chunk-level" RNN

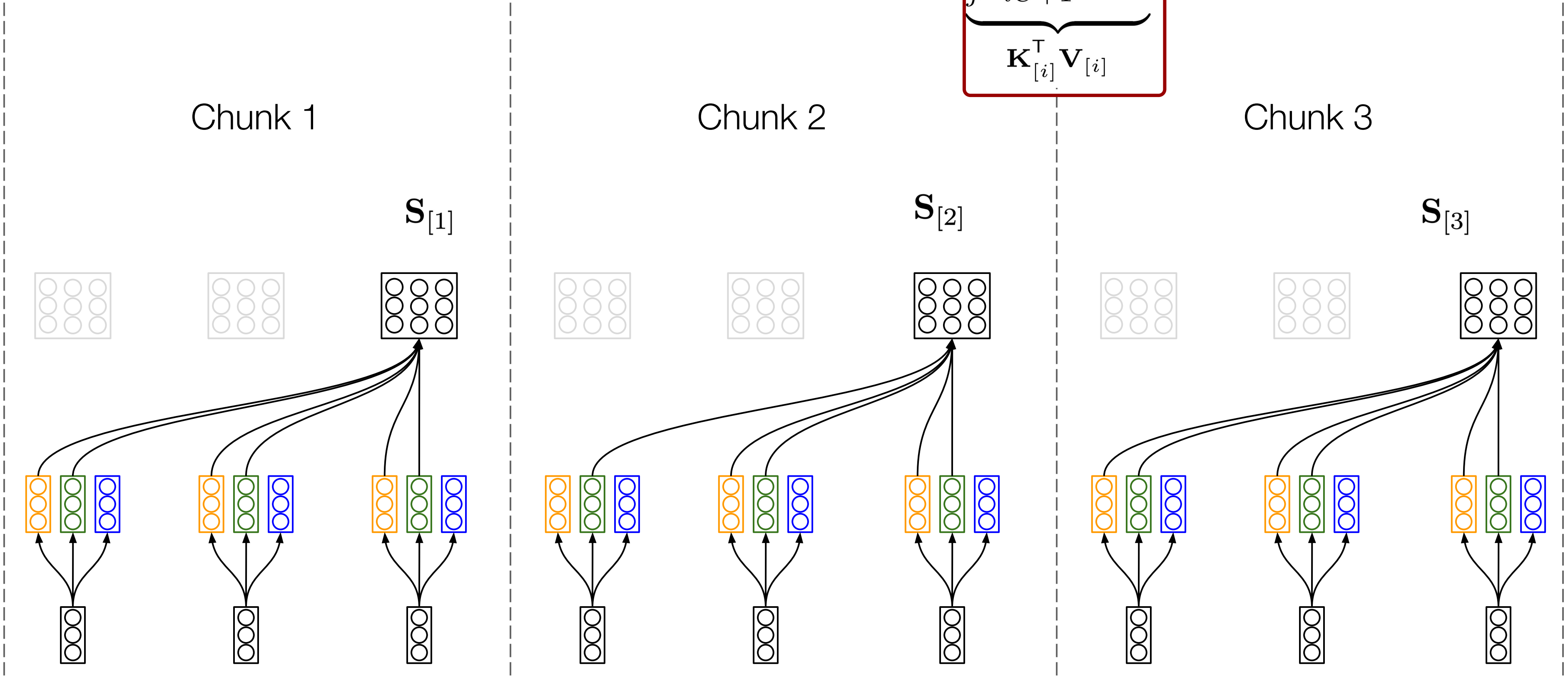
$$\mathbf{S}_{[i+1]} = \mathbf{S}_{[i]} + \underbrace{\sum_{j=iC+1}^{(i+1)C} \boldsymbol{k}_j^{\mathsf{T}} \boldsymbol{v}_j}_{\mathbf{K}_{[i]}^{\mathsf{T}} \mathbf{V}_{[i]}}$$



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

Step 1: local state computation

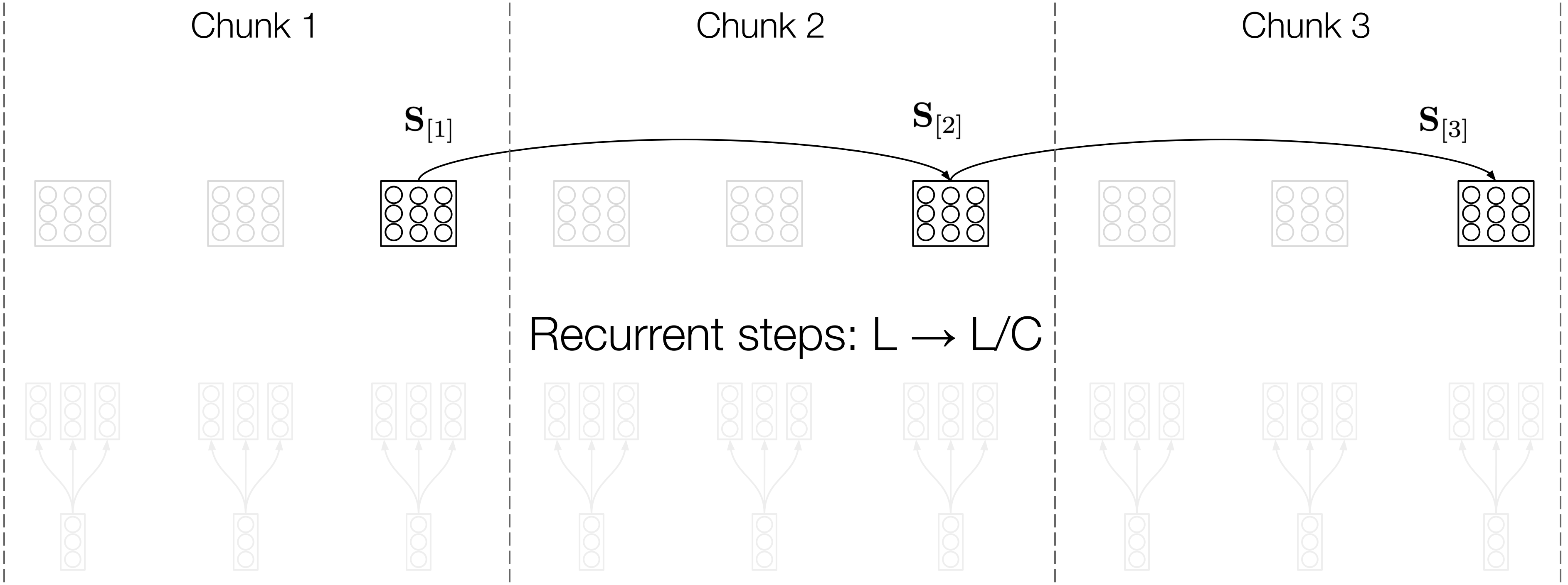
$$\mathbf{S}_{[i+1]} = \mathbf{S}_{[i]} + \underbrace{\sum_{j=iC+1}^{(i+1)C} \boldsymbol{k}_j^\top \boldsymbol{v}_j}_{\mathbf{K}_{[i]}^\top \mathbf{V}_{[i]}}$$



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

Step 2: state passing

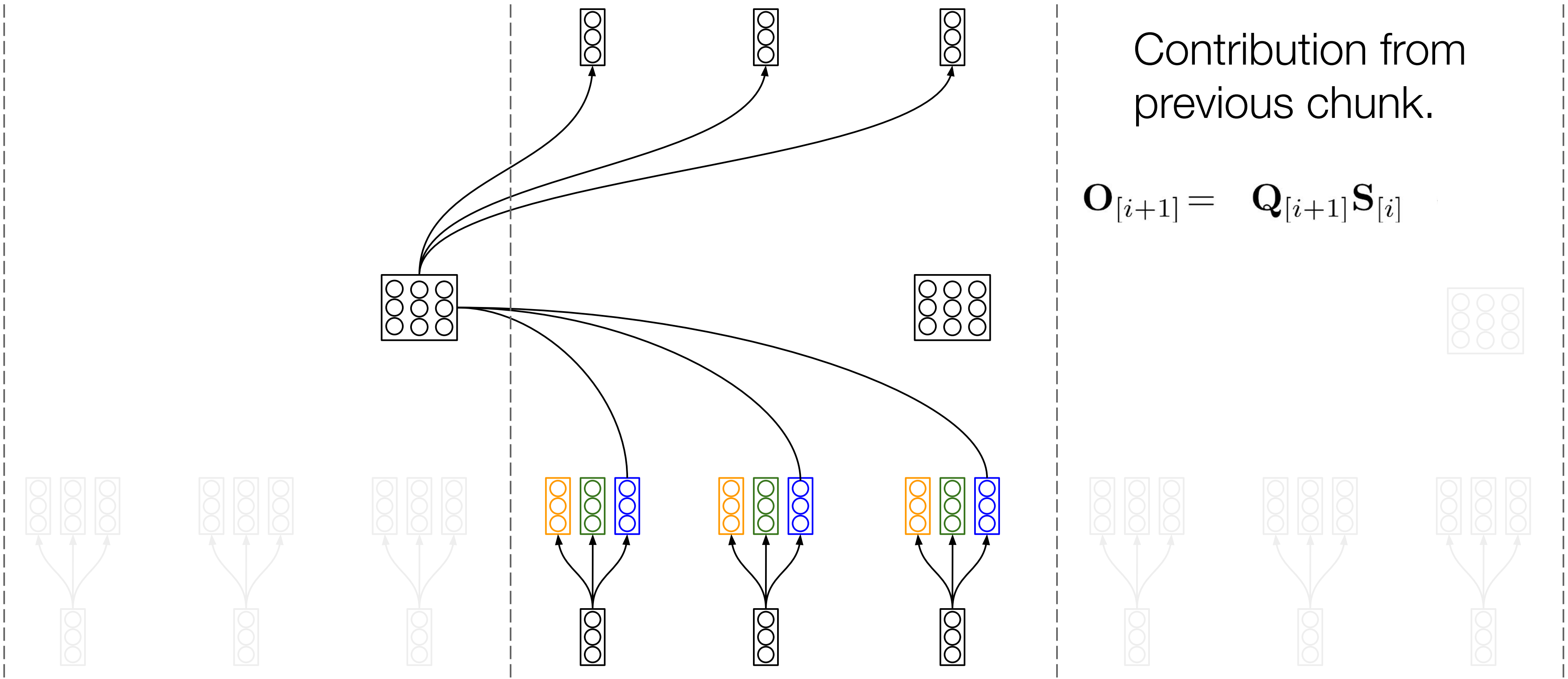
$$\mathbf{S}_{[i+1]} = \boxed{\mathbf{S}_{[i]}} + \underbrace{\sum_{j=iC+1}^{(i+1)C} \boldsymbol{k}_j^{\top} \boldsymbol{v}_j}_{\mathbf{K}_{[i]}^{\top}\mathbf{V}_{[i]}}$$



Recurrent steps: L → L/C

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

Step 3: output computation

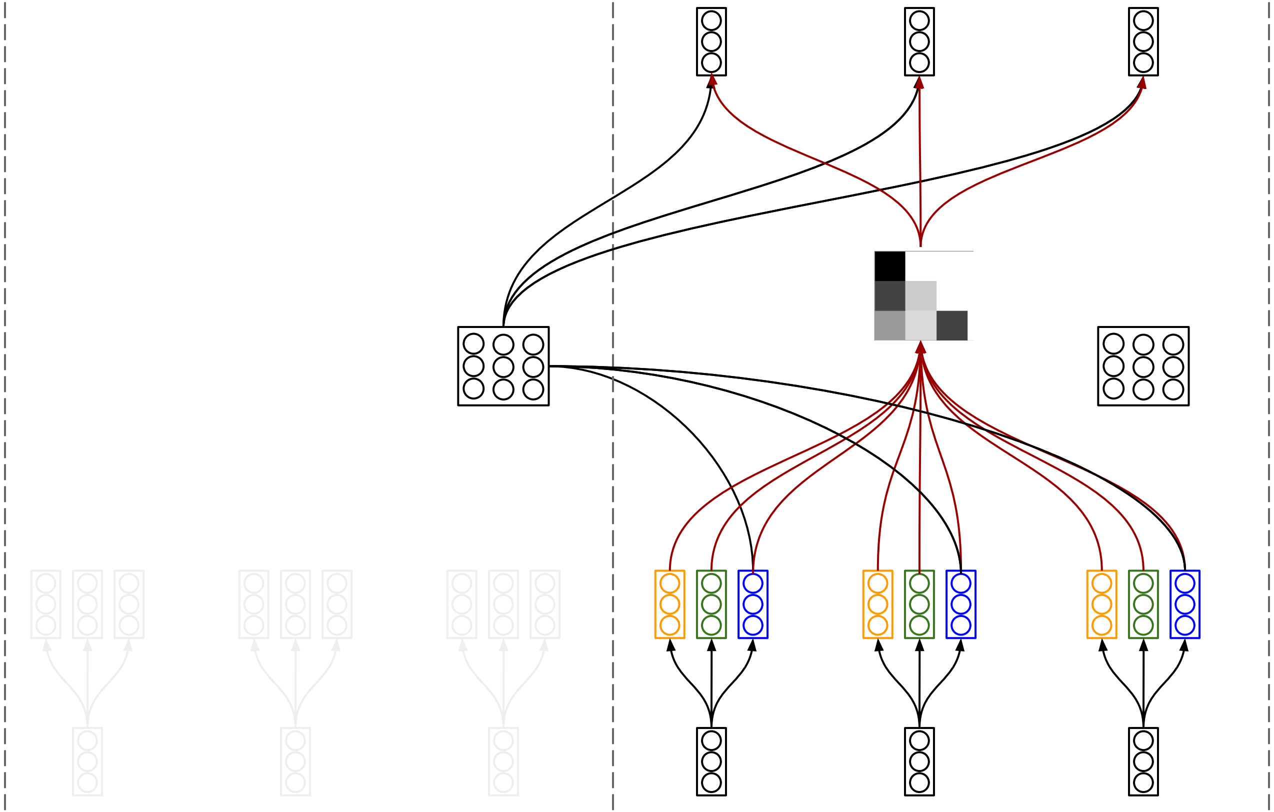


Contribution from previous chunk.

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

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

Step 3: output computation



Contribution from previous chunk.

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

Contribution from current chunk.

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

	Fully Parallel Form	Chunkwise Parallel Form	Fully Recurrent Form
Compute	$O(L^2)$	$O(LC)$	$O(L)$
Memory	$O(L)$	$O(C)$	$O(1)$
Steps	$O(1)$	$O\left(\frac{L}{C}\right)$	$O(L)$

Chunkwise parallel form interpolates between fully parallel and recurrent forms.

- C = L → Fully parallel form
- C = 1 → Fully recurrent form

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

	Fully Parallel Form	Chunkwise Parallel Form	Fully Recurrent Form
Compute	$O(L^2)$	$O(LC)$ ☺	$O(L)$
Memory	$O(L)$	$O(C)$	$O(1)$ ☺
Steps	$O(1)$	$O\left(\frac{L}{C}\right)$ ☺	$O(L)$

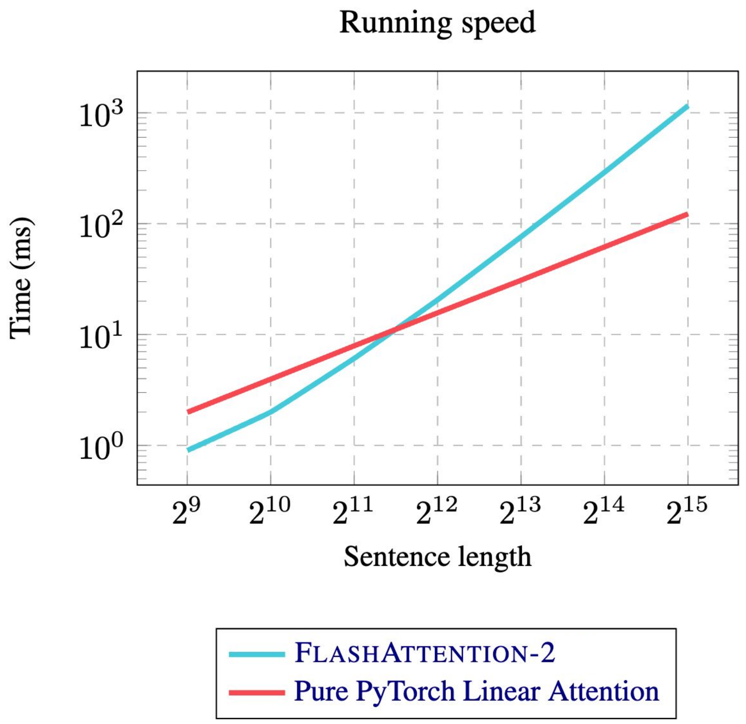
Chunkwise parallel form interpolates between fully parallel and recurrent forms.

- C = L → Fully parallel form
- C = 1 → 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.

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

Linear Transformers for Efficient Sequence Modeling

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
NeurIPS '24

Our Contributions

Issue 1:

Slower than optimized implementations of softmax attention in practice.

→

FlashLinearAttention

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

Our Contributions

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

→

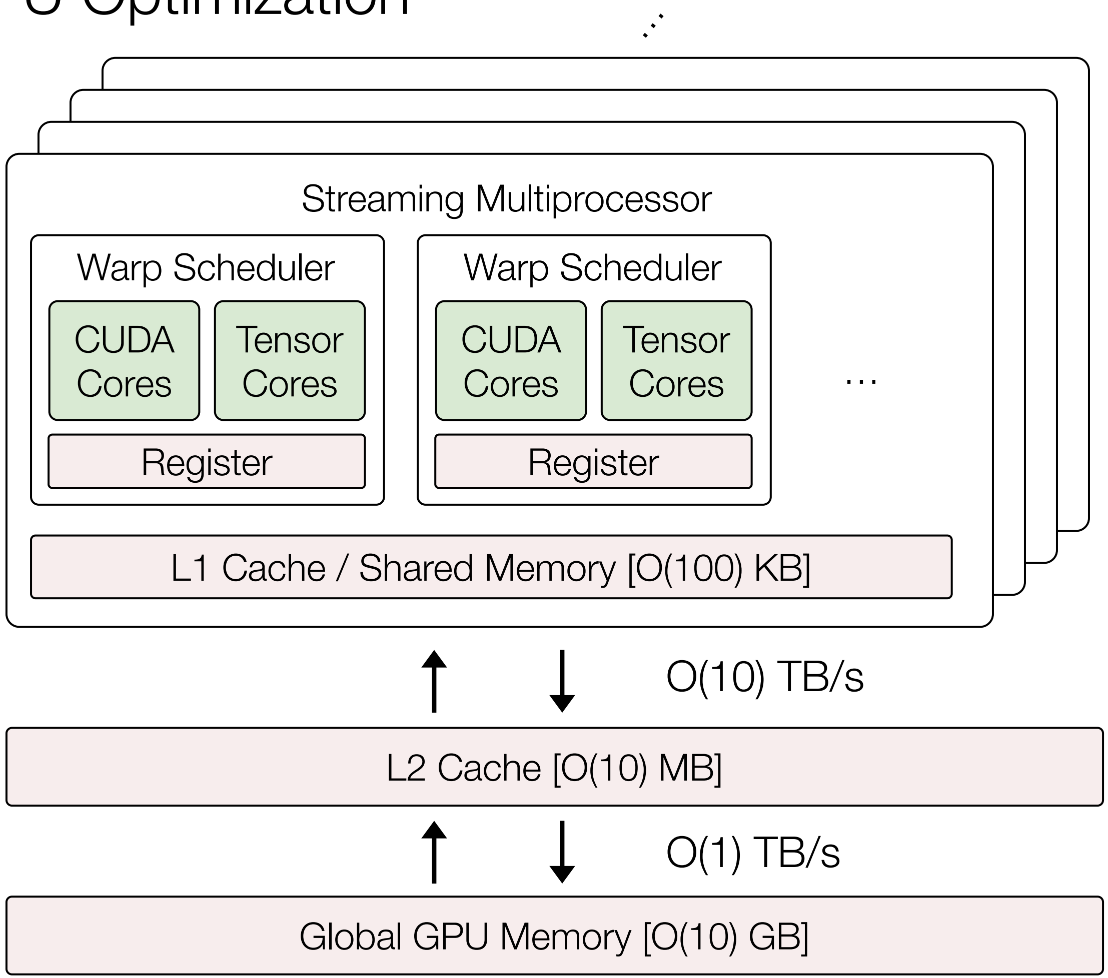
FlashLinearAttention
Hardware-efficient I/O-aware implementation of linear attention

Issue 2:
Underperforms softmax attention by a significant margin.

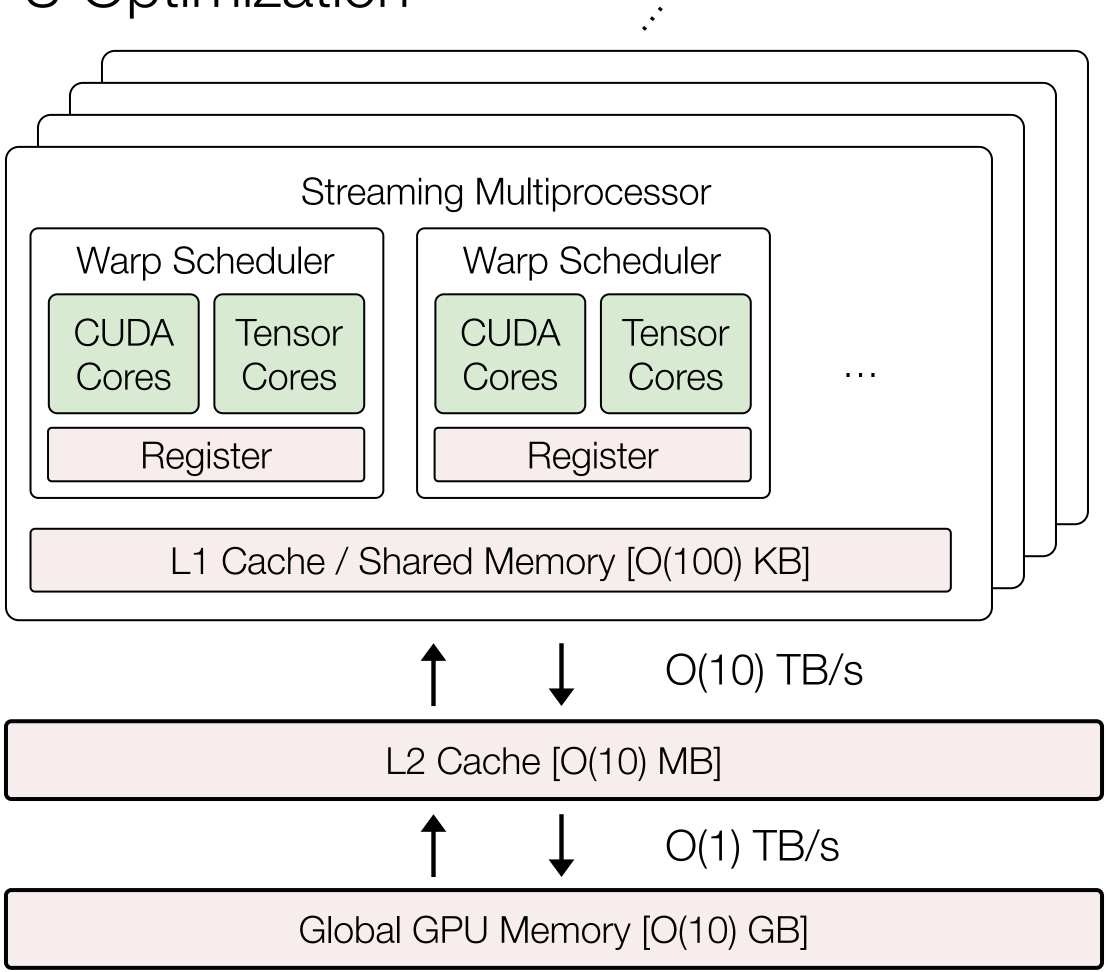
→

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

Background: Principles of GPU Optimization



Background: Principles of GPU Optimization

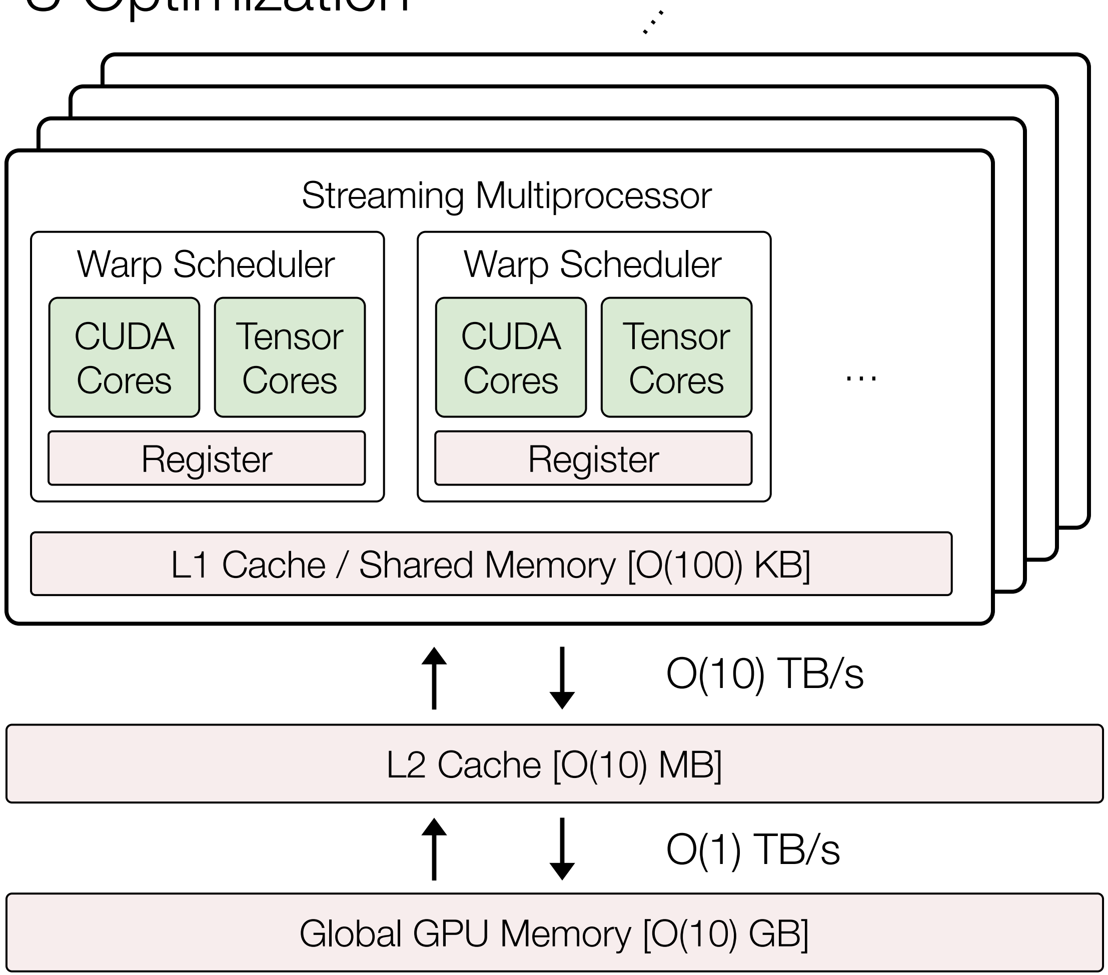


Minimize memory movement between global memory (HBM) and L2 cache (kernel fusion).

Background: Principles of GPU Optimization

Keep the streaming multiprocessors as busy as possible (parallelization).

Minimize memory movement between global memory (HBM) and L2 cache (kernel fusion).

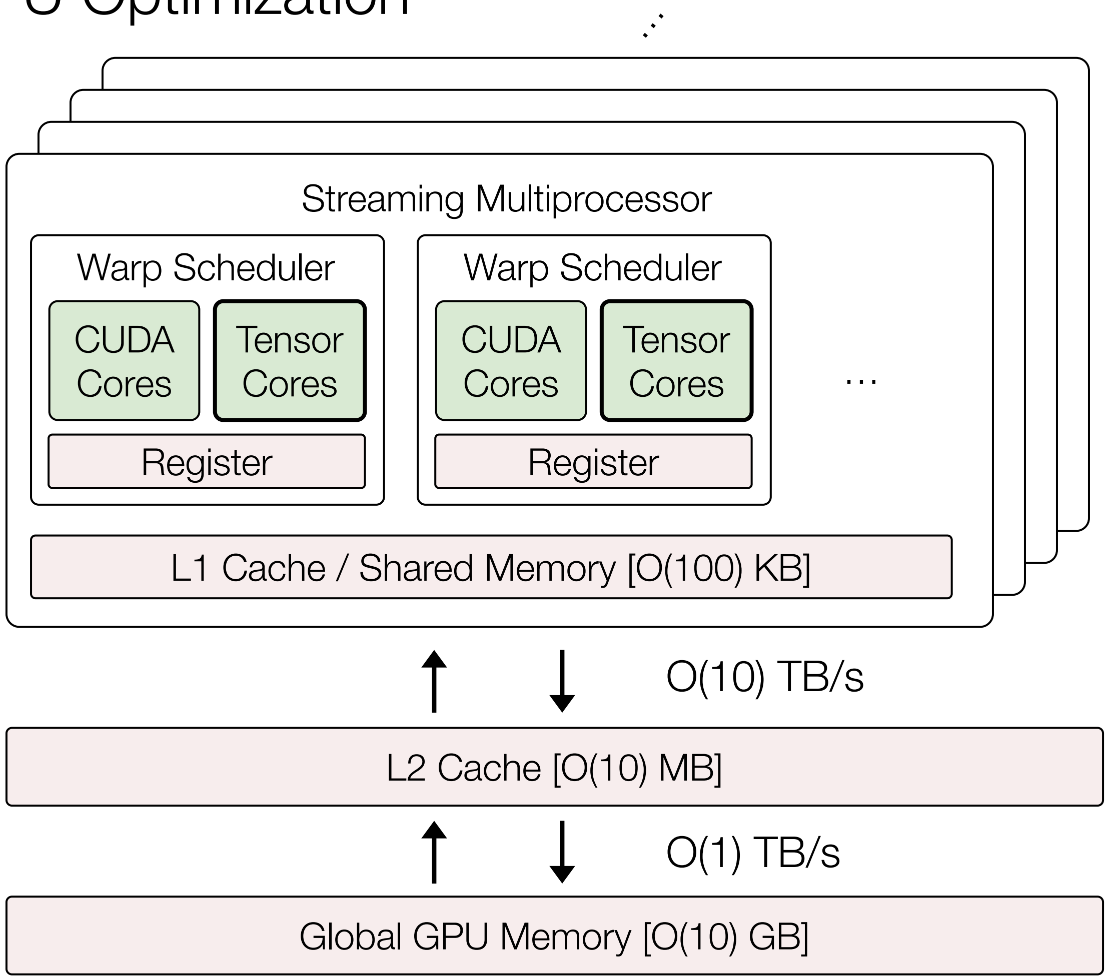


Background: Principles of GPU Optimization

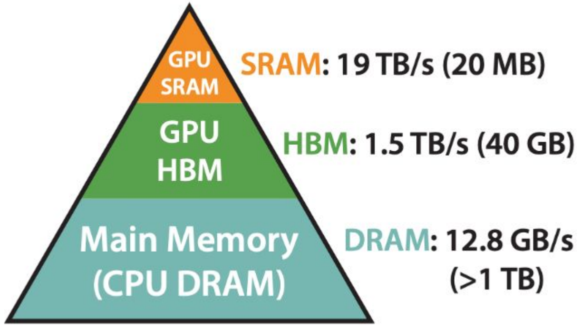
Use (half-precision) matmuls as much as possible.

Keep the streaming multiprocessors as busy as possible (parallelization).

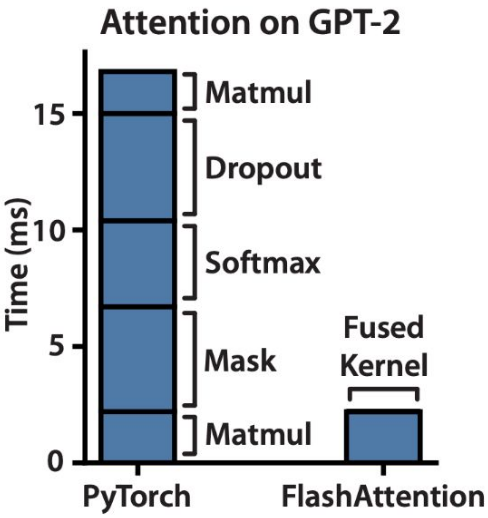
Minimize memory movement between global memory (HBM) and L2 cache (kernel fusion).



Background: FlashAttention [Dao et al. '22, Dao '23]



Memory Hierarchy with Bandwidth & Memory Size



$$\mathbf{Q},\mathbf{K},\mathbf{V} = \mathbf{X}\boldsymbol{W}_Q, \mathbf{X}\boldsymbol{W}_K, \mathbf{X}\boldsymbol{W}_V$$

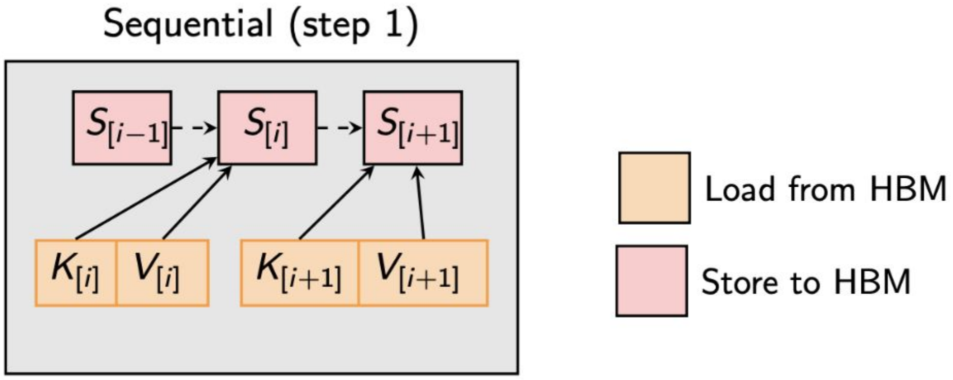
$$\boxed{\mathbf{A} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M})}$$

$$\mathbf{O} = \mathbf{A}\mathbf{V}$$

Fused attention:
Never instantitate this in slower HBM.

[Image credit: Dao et al. '22]

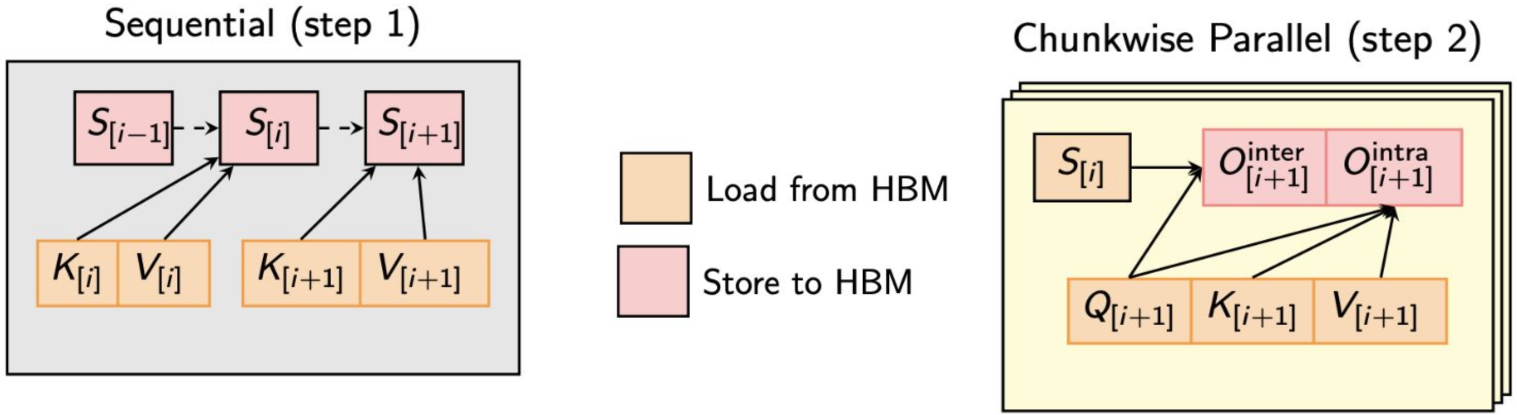
FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention



Step 1: Sequential state computation

Fuse local state computation and state passing in a single kernel to minimize I/O cost.

FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention



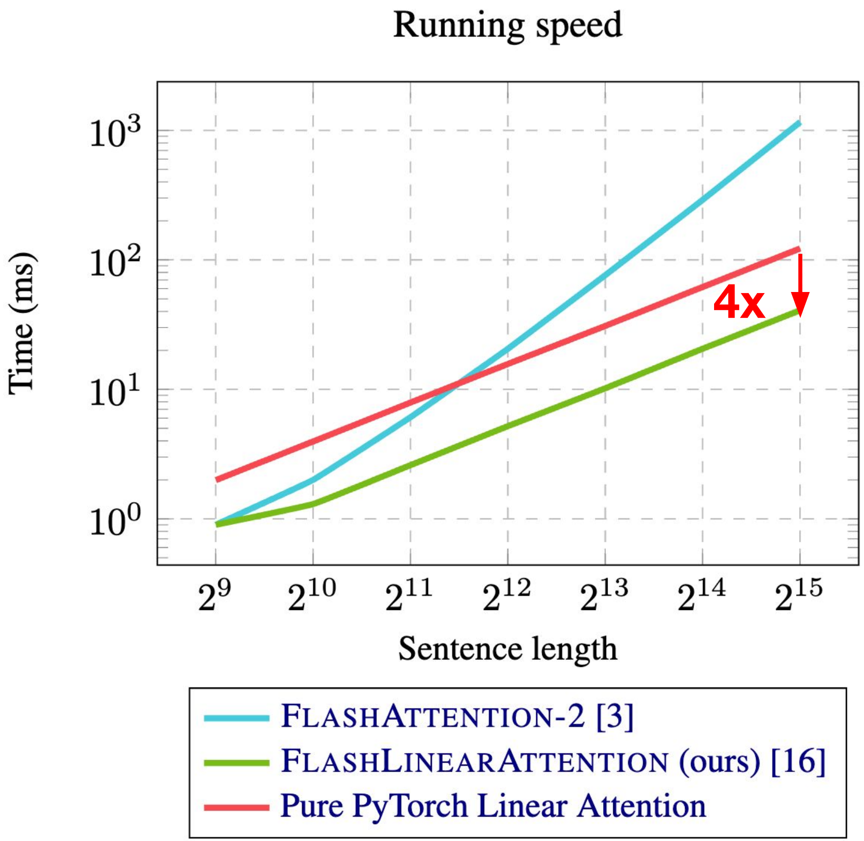
Step 1: Sequential state computation

Fuse local state computation and state passing in a single kernel to minimize I/O cost.

Step 2: Parallel output computation

Compute all chunk outputs in parallel based on previous chunk's state and current chunk's QKV blocks.

FlashLinearAttention: Hardware-Efficient Algorithm for 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.

Date	Model	Title	Paper	Code	FLA impl
2023-07	RetNet (@MSRA@THU)	Retentive network: a successor to transformer for large language models	[arxiv]	[official] [RetNet]	code
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
2021-10	ABC (@UW)	Attention with Bounded-memory Control	arxiv		code
2023-09	HGRN	Hierarchically Gated Recurrent Neural Network for Sequence Modeling	openreview	[official]	code

Date	Model	Title	Paper	Code	FLA impl
2023-09	HGRN	Hierarchically Gated Recurrent Neural Network for Sequence Modeling	openreview	[official]	code
2024-04	HGRN2	HGRN2: Gated Linear RNNs with State Expansion	arxiv	[official]	code
2024-04	RWKV6	Eagle and Finch: RWKV with Matrix-Valued States and Dynamic Recurrence	arxiv	[official]	code
2024-06	Samba	Samba: Simple Hybrid State Space Models for Efficient Unlimited Context Language Modeling	arxiv	[official]	code
2024-05	Mamba2	Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality	arxiv	[official]	code
2024-09	GSA	Gated Slot Attention for Efficient Linear-Time Sequence Modeling	arxiv	[official]	code

Our Contributions

Issue 1:

Slower than optimized implementations of softmax attention in practice. → **FlashLinearAttention**
Hardware-efficient I/O-aware implementation of linear attention

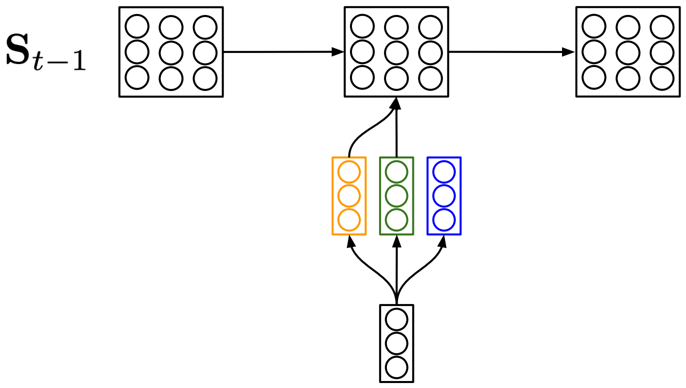
Issue 2:

Underperforms softmax attention by a significant margin. → **Gated Linear Attention**
Linear attention with data-dependent “forget” gate

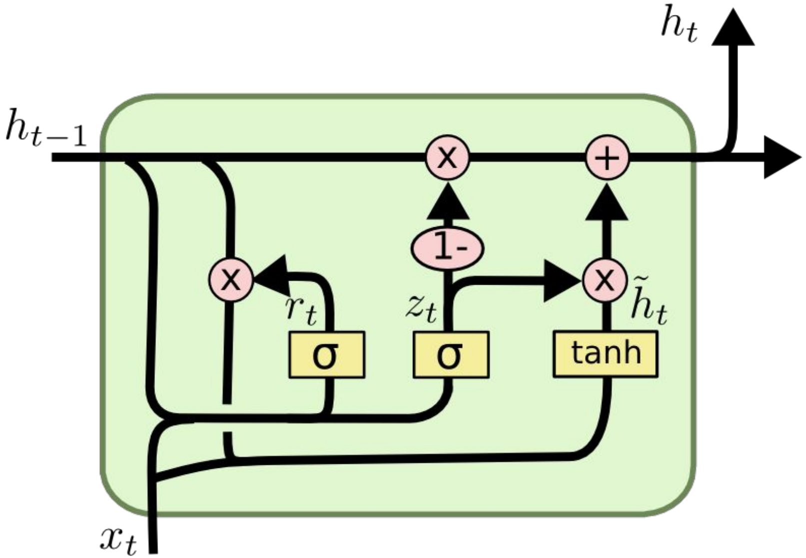
Gated Linear Attention: Data-dependent Multiplicative Gate

Simple Linear Attention

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



Gated Linear Attention: Data-dependent Multiplicative Gate



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

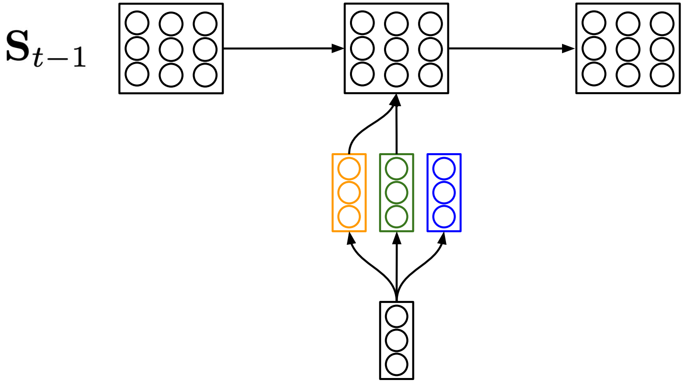
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

[Image credit: https://colah.github.io/posts/2015-08-Understanding-LSTMs/]

Gated Linear Attention: Data-dependent Multiplicative Gate

Simple Linear Attention

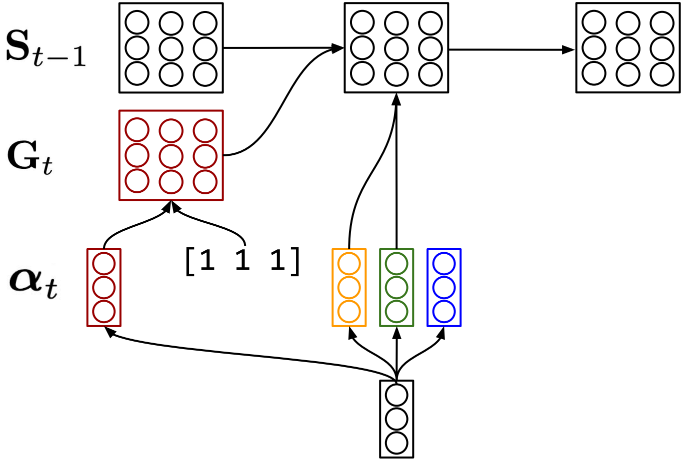
$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top$$



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

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

$$\mathbf{B}_t := \prod_{j=1}^{t} \boldsymbol{\alpha}_j$$

$$\mathbf{P}_{ij} = \sum_{k=1}^{d} \mathbf{Q}_{ik}\mathbf{K}_{jk}\exp(\log \mathbf{B}_{ik} - \log \mathbf{B}_{jk})$$

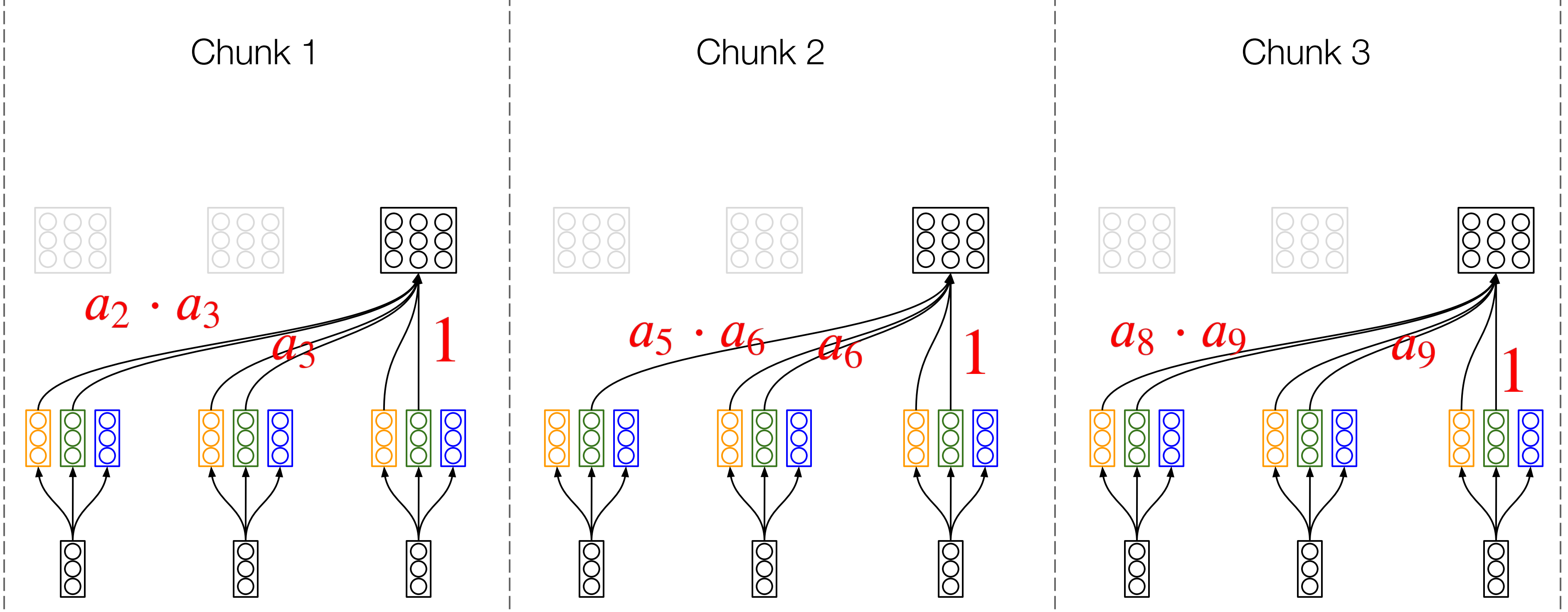
Gated Linear Attention: Decay-aware Chunkwise Parallel Form

Step 1: local state computation

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

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

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



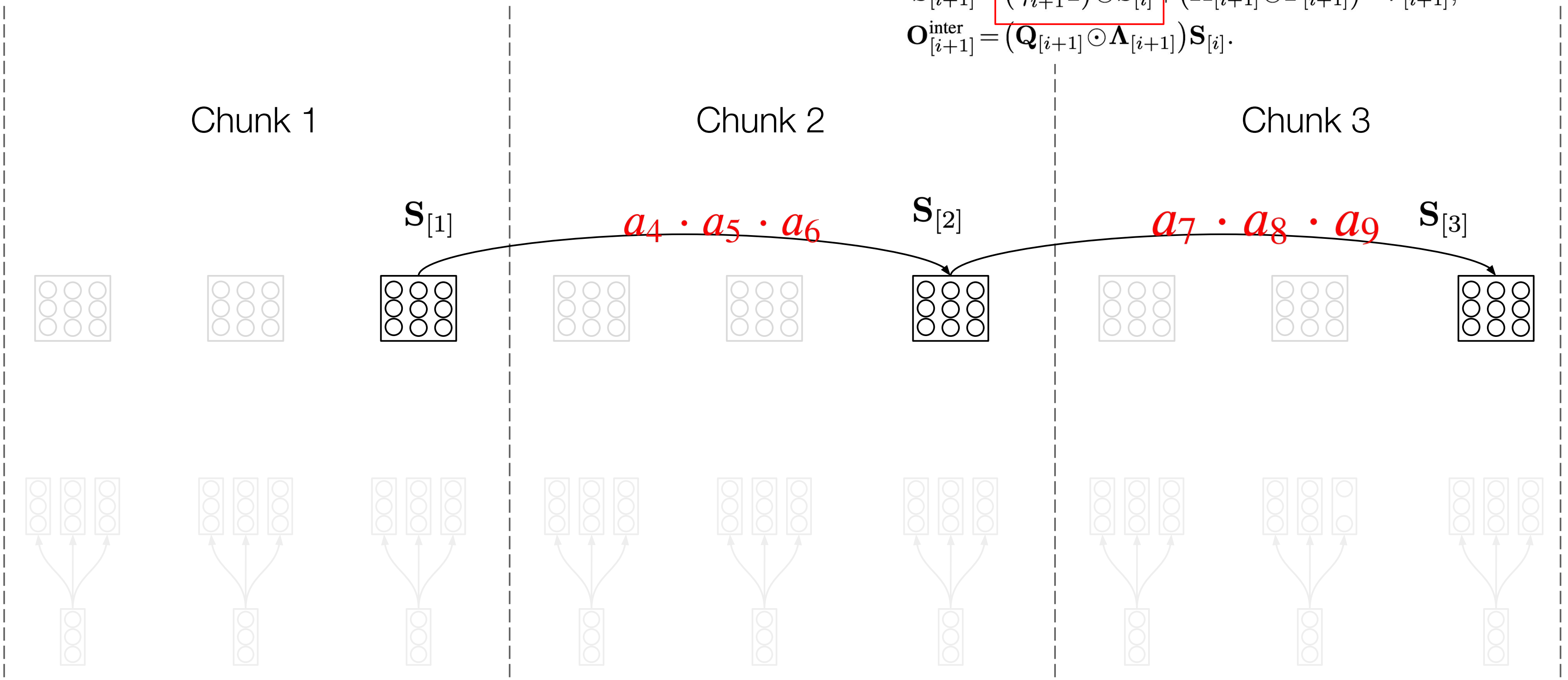
Gated Linear Attention: Decay-aware Chunkwise Parallel Form

Step 2: state passing

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

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

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



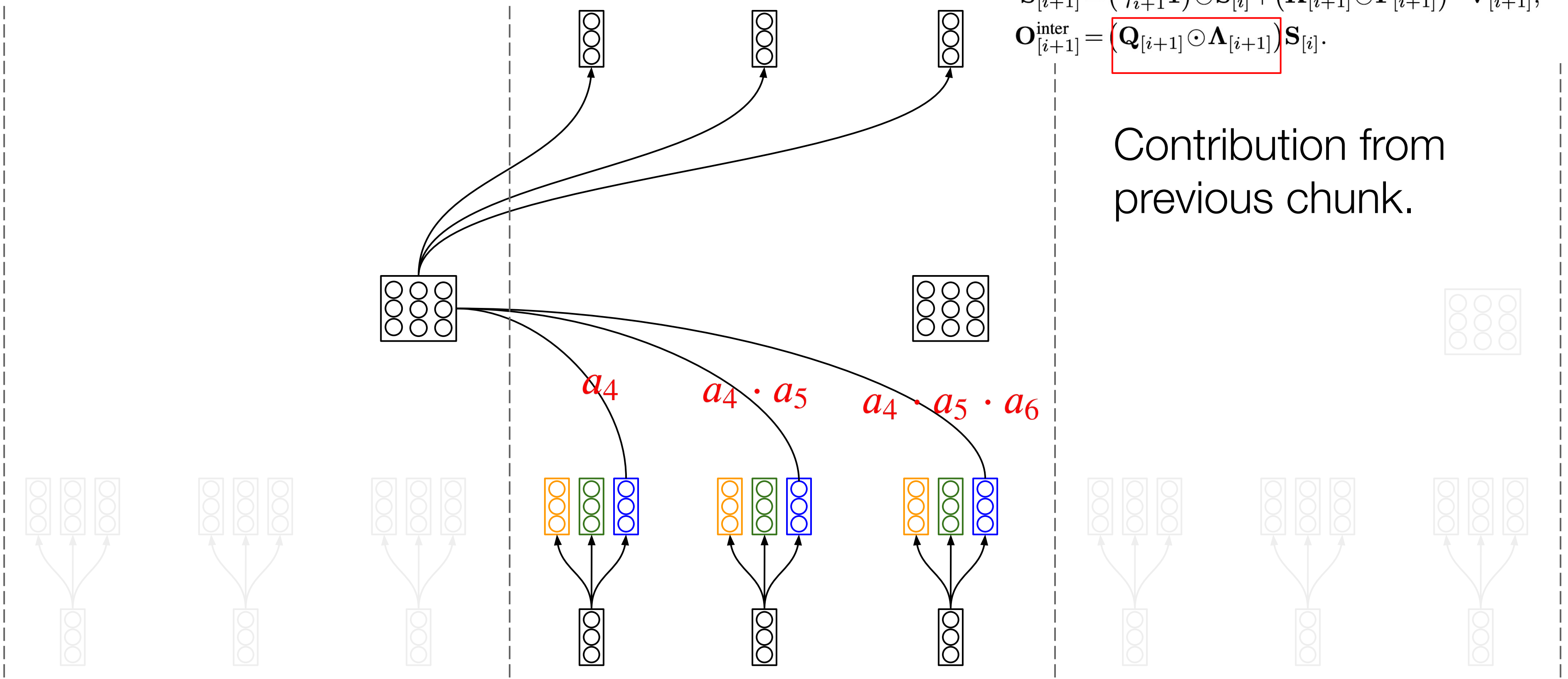
Gated Linear Attention: Decay-aware Chunkwise Parallel Form

Step 3: output computation

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

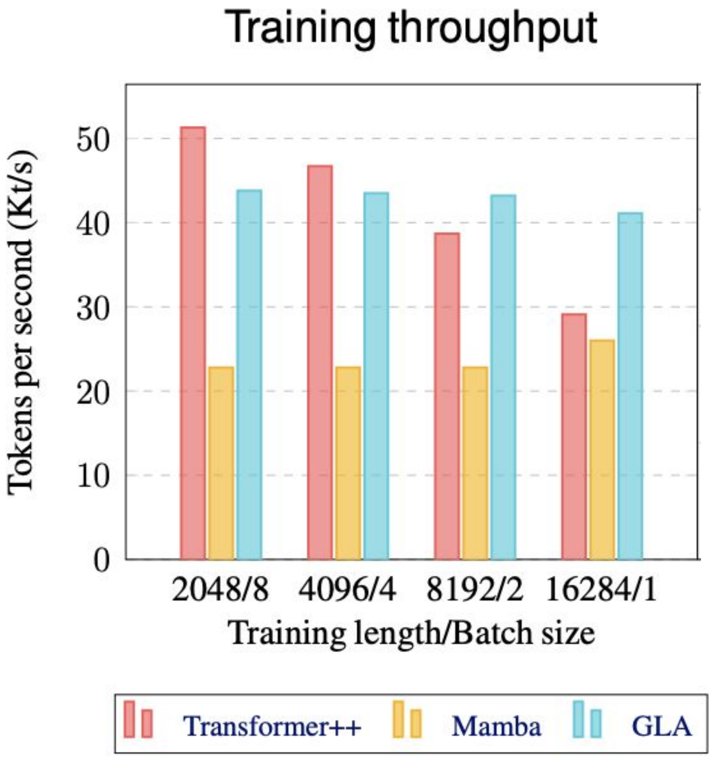
$$\mathbf{S}_{[i+1]} = \left(\boldsymbol{\gamma}_{i+1}^{\top}\mathbf{1}\right) \odot \mathbf{S}_{[i]} + \left(\mathbf{K}_{[i+1]} \odot \boldsymbol{\Gamma}_{[i+1]}\right)^{\top} \mathbf{V}_{[i+1]},$$

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



Contribution from previous chunk.

Gated Linear Attention: Throughput



Gated Linear Attention: Performance

Model	PPL ↓	LM Eval ↑
Transformer++	16.9	50.9
RetNet (Linear Attention with Decay)	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…
```

[Arora et al. ’24]

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)

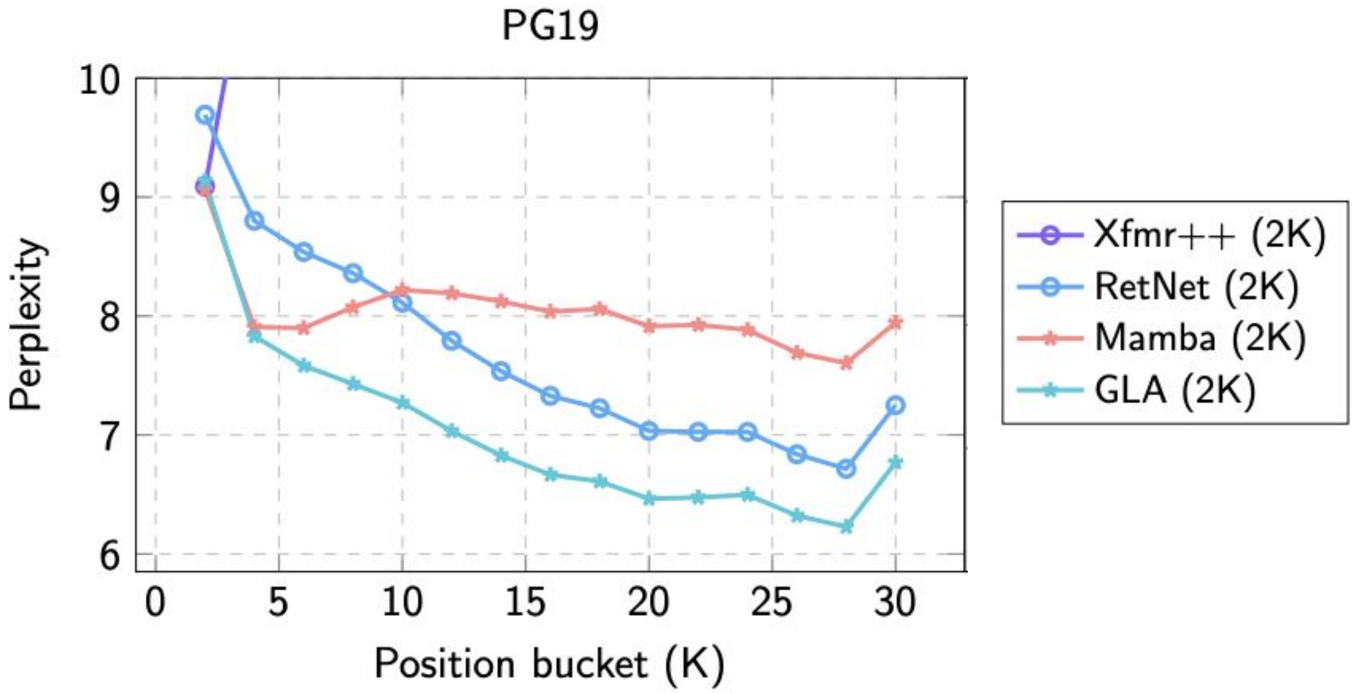
[Arora et al. '24]

Gated Linear Attention: Recall-oriented Tasks

Model	PPL ↓	LM Eval ↑	Retrieval ↑
Transformer++	16.9	50.9	41.8
RetNet (Linear Attention with Decay)	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 [Gu and Dao '23]

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

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

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

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

Algorithm 1 SSM (S4)	**Algorithm 2** SSM + Selection (S6)
Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$	**Input:** $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$	**Output:** $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter ▷ Represents structured $N \times N$ matrix	$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter ▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter	$\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter	$\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
$\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter})$	$\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$	$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{B}, \mathsf{L}, \mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$ ▷ Time-invariant: recurrence or convolution	$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$ ▷ Time-varying: recurrence (*scan*) only
return y	**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
Mamba [Gu & Dao 2023]	$\mathbf{G}_t = \exp(-(\mathbf{1}\boldsymbol{\alpha}_t^{\top}) \odot \exp(\boldsymbol{A})), \quad \boldsymbol{\alpha}_t = \text{softplus}(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2})$
Mamba-2 [Dao & Gu 2024]	$\mathbf{G}_t = \gamma_t \mathbf{1}\mathbf{1}^{\top}, \quad \gamma_t = \exp\left(-\text{softplus}\left(\boldsymbol{x}_t \boldsymbol{W}_{\gamma}\right) \exp\left(a\right)\right)$
xLSTM [Beck et al. 2024]	$\mathbf{G}_t = \gamma_t \mathbf{1}\mathbf{1}^{\top}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\gamma}\right)$
GLA [Yang et al. 2023]	$\mathbf{G}_t = \boldsymbol{\alpha}_t \mathbf{1}^{\top}, \quad \boldsymbol{\alpha}_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\alpha_1} \boldsymbol{W}_{\alpha_2}\right)^{\frac{1}{\tau}}$
Gated RetNet [Sun et al. 2024]	$\mathbf{G}_t = \gamma_t \mathbf{1}\mathbf{1}^{\top}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\gamma}\right)^{\frac{1}{\tau}}$
HGRN-2 [Qin et al. 2024]	$\mathbf{G}_t = \boldsymbol{\alpha}_t \mathbf{1}^{\top}, \quad \boldsymbol{\alpha}_t = \boldsymbol{\gamma} + (1 - \boldsymbol{\gamma})\sigma(\boldsymbol{x}_t \boldsymbol{W}_{\alpha})$
RWKV-6 [Peng et al. 2024]	$\mathbf{G}_t = \boldsymbol{\alpha}_t \mathbf{1}^{\top}, \quad \boldsymbol{\alpha}_t = \exp\left(-\exp\left(\boldsymbol{x}_t \boldsymbol{W}_{\alpha}\right)\right)$
Gated RFA [Peng et al. 2021]	$\mathbf{G}_t = \gamma_t \mathbf{1}\mathbf{1}^{\top}, \quad \gamma_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\gamma}\right)$
Decaying FW [Mao et al. 2022]	$\mathbf{G}_t = \boldsymbol{\alpha}_t \boldsymbol{\beta}_t^{\top}, \quad \boldsymbol{\alpha}_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\alpha}\right), \boldsymbol{\beta}_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{W}_{\beta}\right)$

Takeaways

Linear attention removes the nonlinearity in softmax attention → RNN with matrix-valued hidden states.

Chunkwise-parallel algorithm enables wallclock-efficient linear attention.

Data-dependent gating factor improves performance of linear Transformers

Gated linear attention Transformers are (scalable) SSMs.

Linear Transformers for Efficient Sequence Modeling

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
NeurIPS '24

Deficiencies of Linear Transformers / State-Space Models

Multi-Query Associative Recall Task

Input

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

[Example from: Arora et al. ’24]

Deficiencies of Linear Transformers / State-Space Models

Multi-Query Associative Recall Task

Input

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

[Example from: Arora et al. '24]

Deficiencies of Linear Transformers / 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 Transformers / State-Space Models

Multi-Query Associative Recall Task

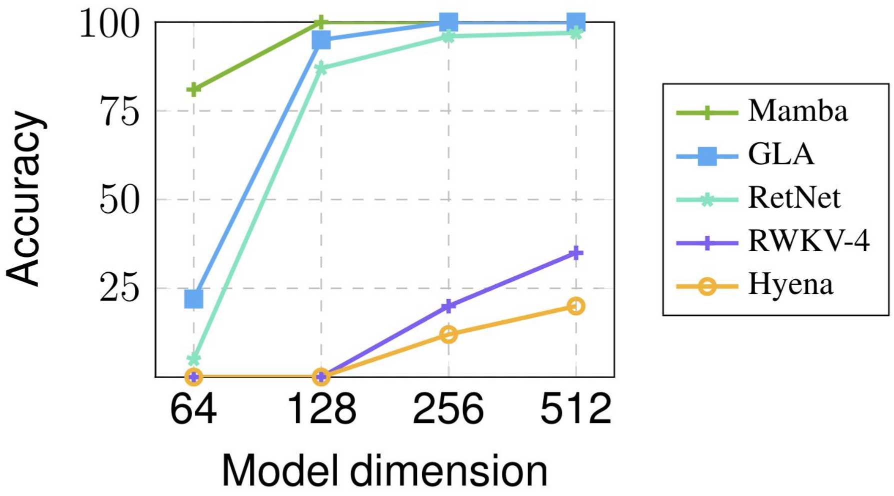
Input

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

Output

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

(Transformers get 100% even with small model dimensions)

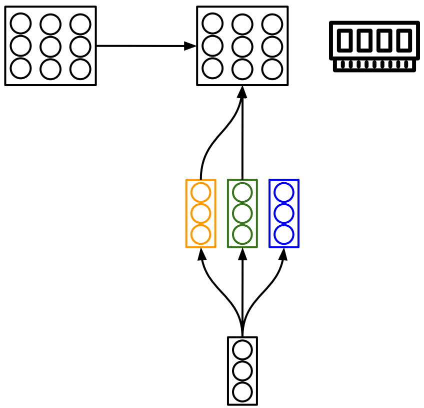


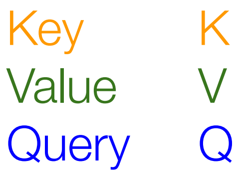
[Example from: Arora et al. '24]

Associative Memory Perspective of Linear Attention

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^{\top}$$ Store "value" $\boldsymbol{v}_t$ associated with "key" $\boldsymbol{k}_t$ into "memory" $\mathbf{S}_{t-1}$.

$\mathbf{S}_t$



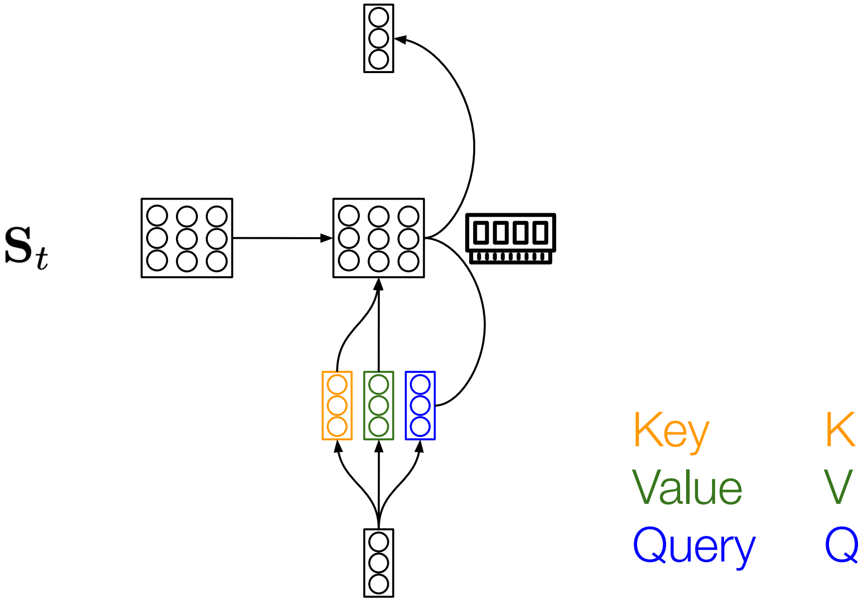


Associative Memory Perspective of Linear Attention

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top$$ Store "value" $\boldsymbol{v}_t$ associated with "key" $\boldsymbol{k}_t$ into "memory" $\mathbf{S}_{t-1}$.

$$\boldsymbol{o}_t = \boldsymbol{q}_t^\top \mathbf{S}_t$$ Look up value associated with "query" $\boldsymbol{q}_t$.

(Reading to and writing from memory)



Associative Memory Perspective of Linear Attention

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^{\mathsf{T}}$$ Store "value" $\boldsymbol{v}_t$ associated with "key" $\boldsymbol{k}_t$ into "memory" $\mathbf{S}_{t-1}$.

$$\boldsymbol{o}_t = \boldsymbol{q}_t^{\mathsf{T}} \mathbf{S}_t$$ Look up value associated with "query" $\boldsymbol{q}_t$.

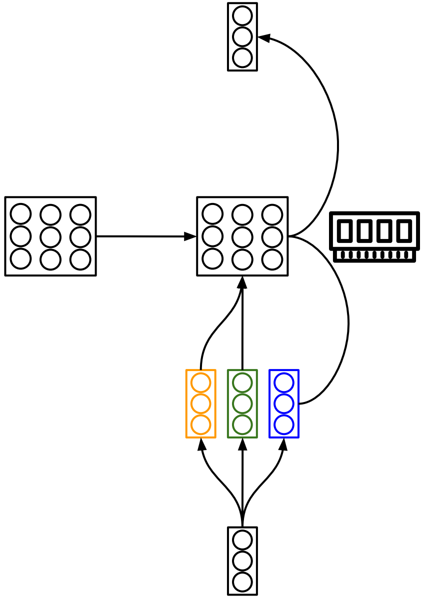
(Reading to and writing from memory)

Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems

Paul Smolensky
Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, CO 80309-0430, USA

Tensor Product Variable Binding [Smolensky '90]

$\mathbf{S}_t$



Key K
Value V
Query Q

Associative Memory Perspective of Linear Attention

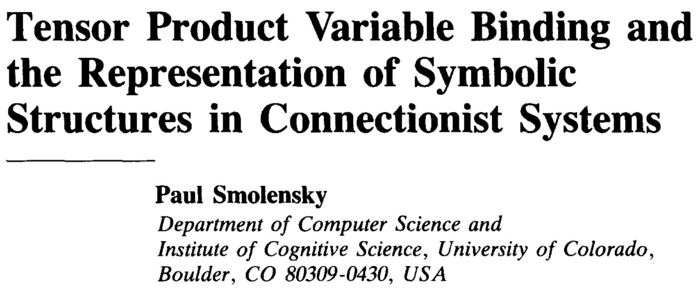
$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^{\mathsf{T}}$$

Store "value" $\boldsymbol{v}_t$ associated with "key" $\boldsymbol{k}_t$ into "memory" $\mathbf{S}_{t-1}$.

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

Look up value associated with "query" $\boldsymbol{q}_t$.

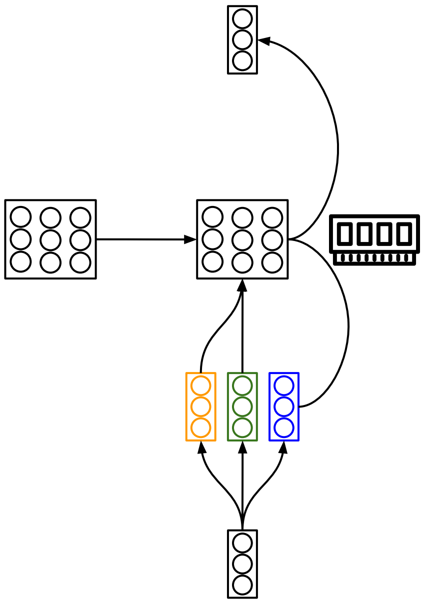
(Reading to and writing from memory)


Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems

Paul Smolensky
Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, CO 80309-0430, USA

Tensor Product Variable Binding [Smolensky '90]

$\mathbf{S}_t$



Issue: There is no way to remove/update the memory!

Key K
Value V
Query Q

DeltaNet: Linear Transformers with the Delta Rule [Schlag et al. ’21]

Idea: Allow the values associated with keys to be removed/updated.

DeltaNet: Linear Transformers with the Delta Rule [Schlag et al. '21]

Idea: Allow the values associated with keys to be removed/updated.

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

DeltaNet: Linear Transformers with the Delta Rule [Schlag et al. '21]

Idea: Allow the values associated with keys to be removed/updated.

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

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

DeltaNet: Linear Transformers with the Delta Rule [Schlag et al. '21]

Idea: Allow the values associated with keys to be removed/updated.

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

DeltaNet: Linear Transformers with the Delta Rule [Schlag et al. '21]

Idea: Allow the values associated with keys to be removed/updated.

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

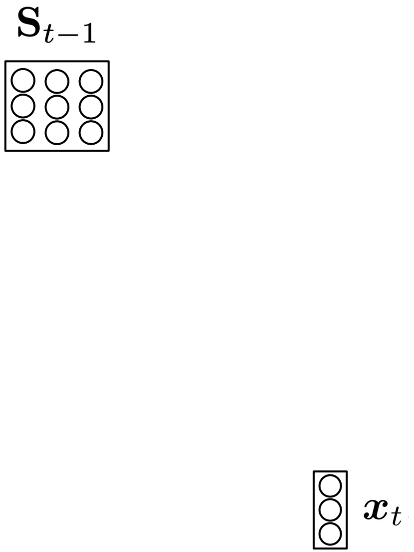
Remove old memory, write new memory $\mathbf{S}_t = \mathbf{S}_{t-1} \underbrace{- \boldsymbol{v}_t^{\text{old}} \boldsymbol{k}_t^{\intercal}}_{\text{remove}} \underbrace{+ \boldsymbol{v}_t^{\text{new}} \boldsymbol{k}_t^{\intercal}}_{\text{write}}$

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

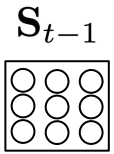
$$\mathbf{S}_t = \mathbf{S}_{t-1} + \beta_t (\boldsymbol{v}_t - \boldsymbol{v}_t^{\text{old}}) \boldsymbol{k}_t^{\intercal}$$

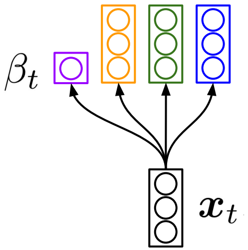
An application of Delta update rule [Widrow & Hoff '60]

DeltaNet [Schlag et al. ’21]



DeltaNet [Schlag et al. '21]





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

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

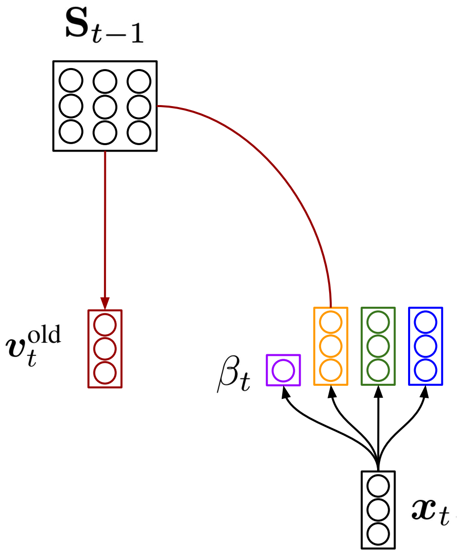
Key K
Value V
Query Q

DeltaNet [Schlag et al. '21]

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

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

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



Key K
Value V
Query Q

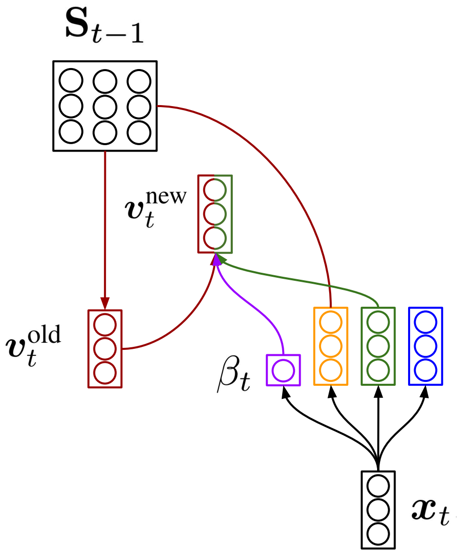
DeltaNet [Schlag et al. '21]

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

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

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

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



Key K
Value V
Query Q

DeltaNet [Schlag et al. '21]

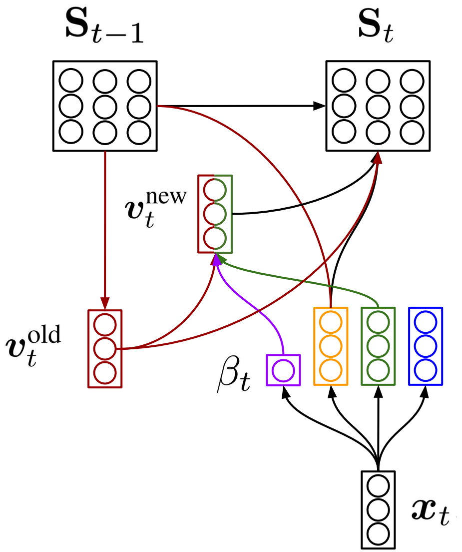
$$\mathbf{S}_t = \mathbf{S}_{t-1} \underbrace{- \boldsymbol{v}_t^{\text{old}} \boldsymbol{k}_t^{\top}}_{\text{remove}} \underbrace{+ \boldsymbol{v}_t^{\text{new}} \boldsymbol{k}_t^{\top}}_{\text{write}}$$

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

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

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

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



Key K
Value V
Query Q

DeltaNet [Schlag et al. '21]

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

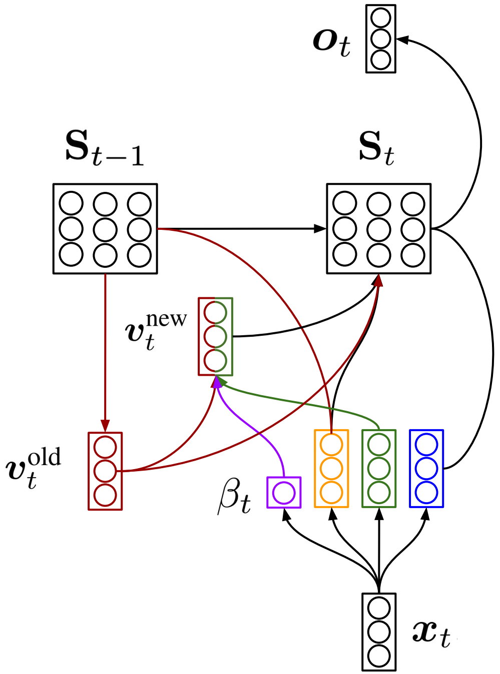
$$\mathbf{S}_t = \mathbf{S}_{t-1} \underbrace{-\boldsymbol{v}_t^{\text{old}} \boldsymbol{k}_t^{\top}}_{\text{remove}} \underbrace{+\boldsymbol{v}_t^{\text{new}} \boldsymbol{k}_t^{\top}}_{\text{write}}$$

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

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

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

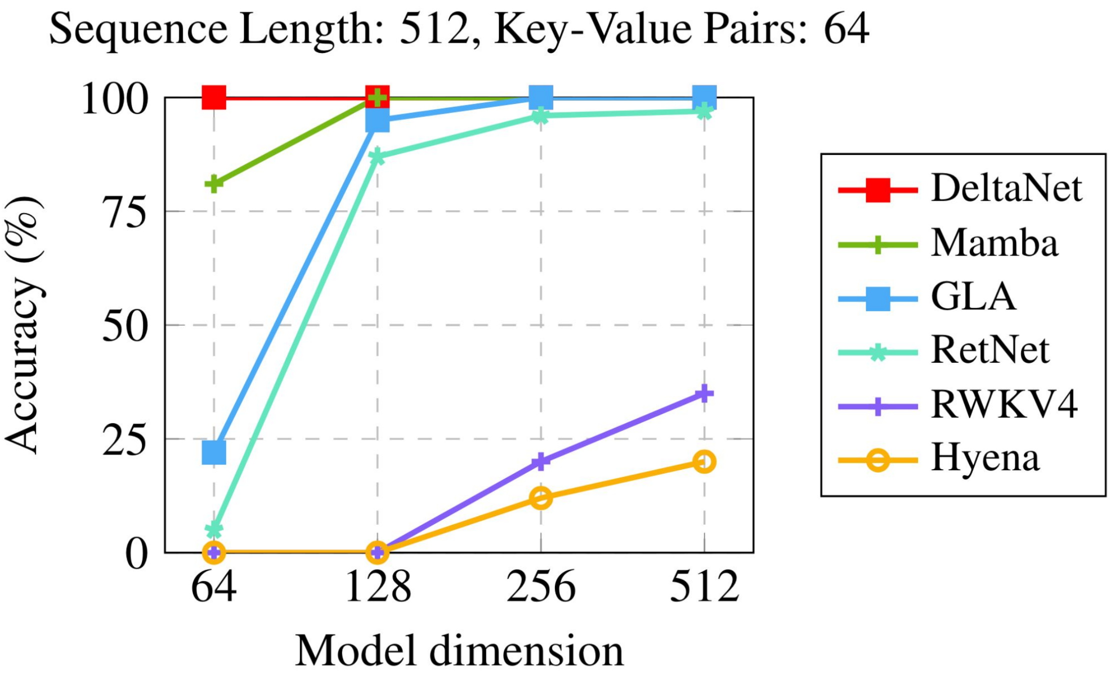
$$\beta_t = \sigma(\mathbf{W}_\beta \boldsymbol{x}_t) \in (0, 1)$$



Key K
Value V
Query Q

DeltaNet Associative Recall Performance

Multi-Query Associative Recall Task



DeltaNet Issue

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

→

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

DeltaNet Issue

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

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

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

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

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

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

DeltaNet Issue

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Product of generalized Householder matrices.

Parallelizing DeltaNet: Memory-efficient Representation

THE WY REPRESENTATION FOR PRODUCTS OF HOUSEHOLDER MATRICES*

CHRISTIAN BISCHOF† AND CHARLES VAN LOAN†

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

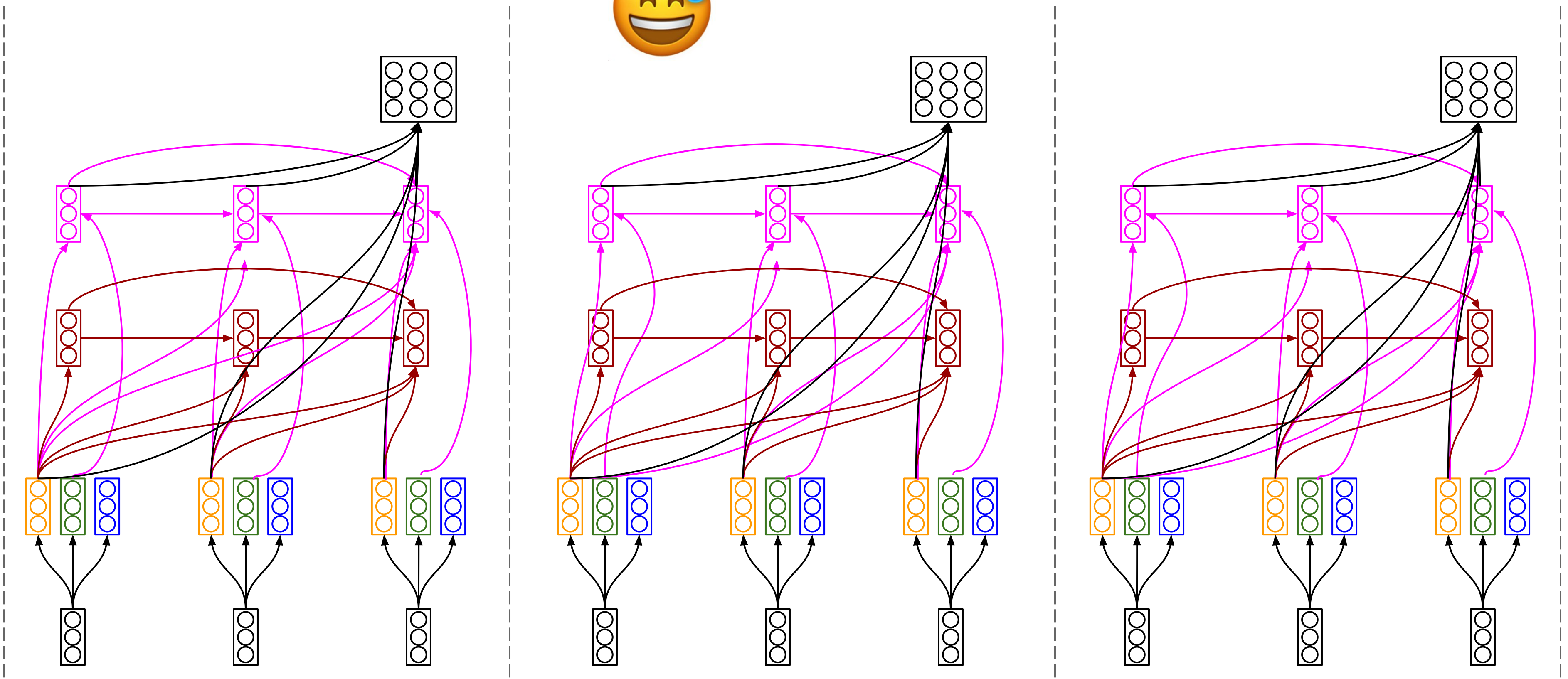
$$\mathbf{S}_n = \mathbf{S}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^{\top}) + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^{\top} \quad \longrightarrow \quad \mathbf{S}_n = \sum_{t=1}^{n} \boxed{\boldsymbol{u}_t} \boldsymbol{k}_n^{\top}$$

Idea: Compute the pseudo-value vectors and then just run regular linear attention.

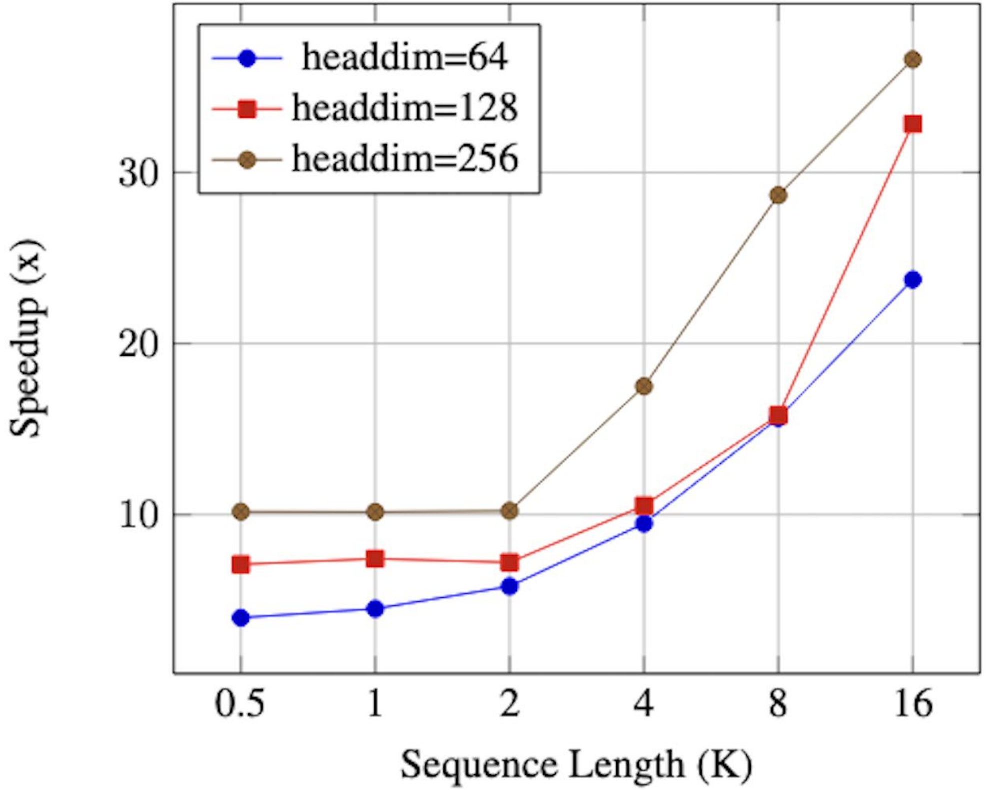
Chunkwise Parallel Form of DeltaNet

Recurrent W/U construction

$$\mathbf{P}_n = \mathbf{I} - \sum_{t=1}^{n} \boxed{\boldsymbol{w}_t} \boldsymbol{k}_t^\top \qquad \mathbf{S}_n = \sum_{t=1}^{n} \boxed{\boldsymbol{u}_t} \boldsymbol{k}_n^\top$$

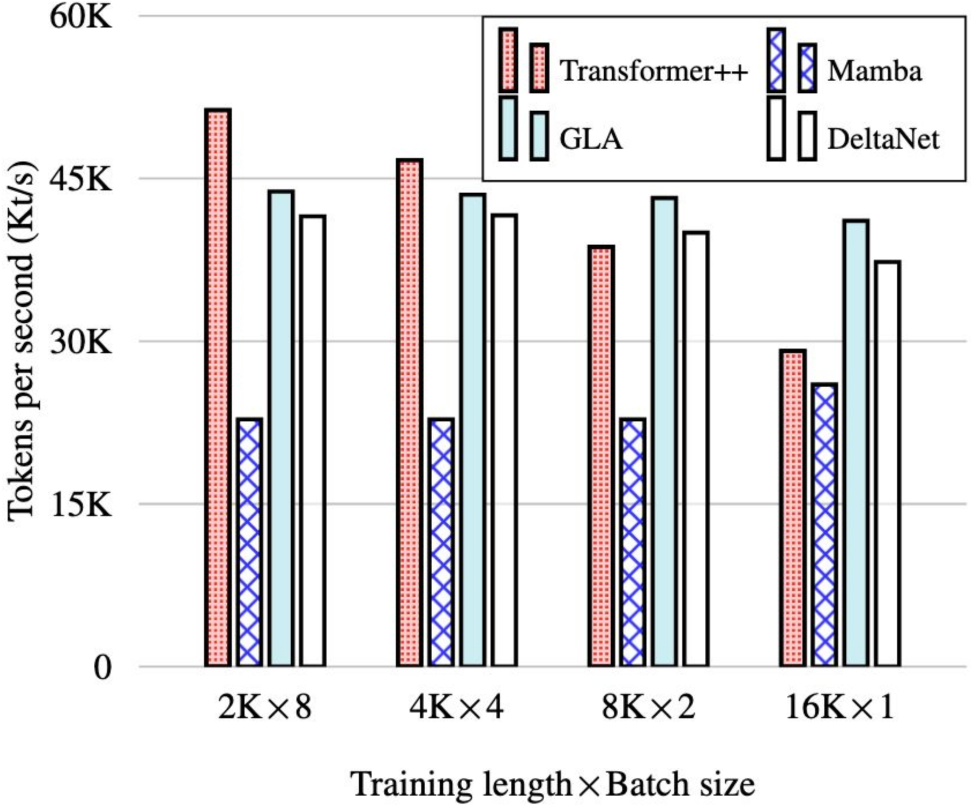


Parallelized DeltaNet: Speed



On a single H100

Parallelized DeltaNet: Speed



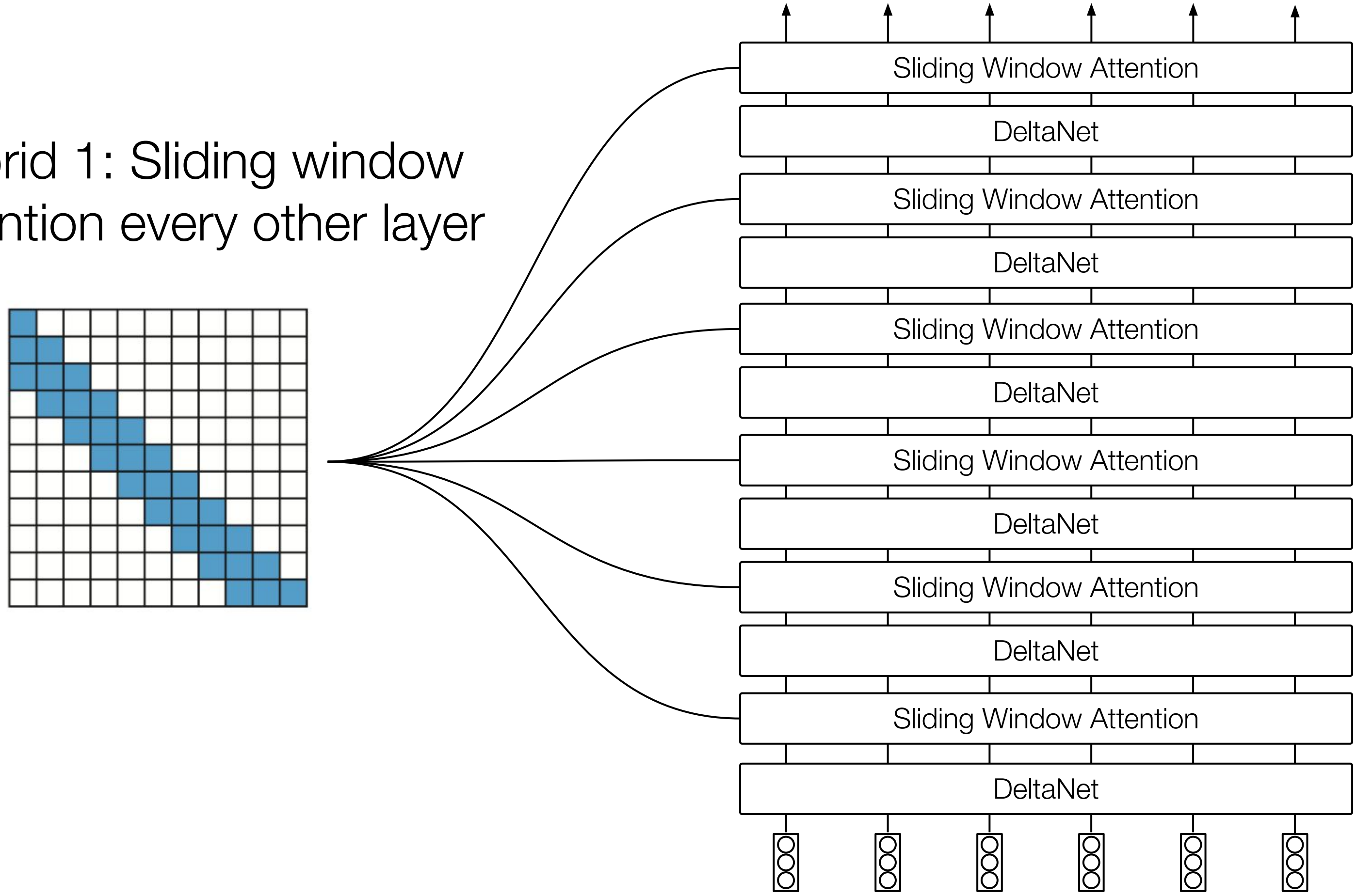
Parallelized DeltaNet: Performance

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

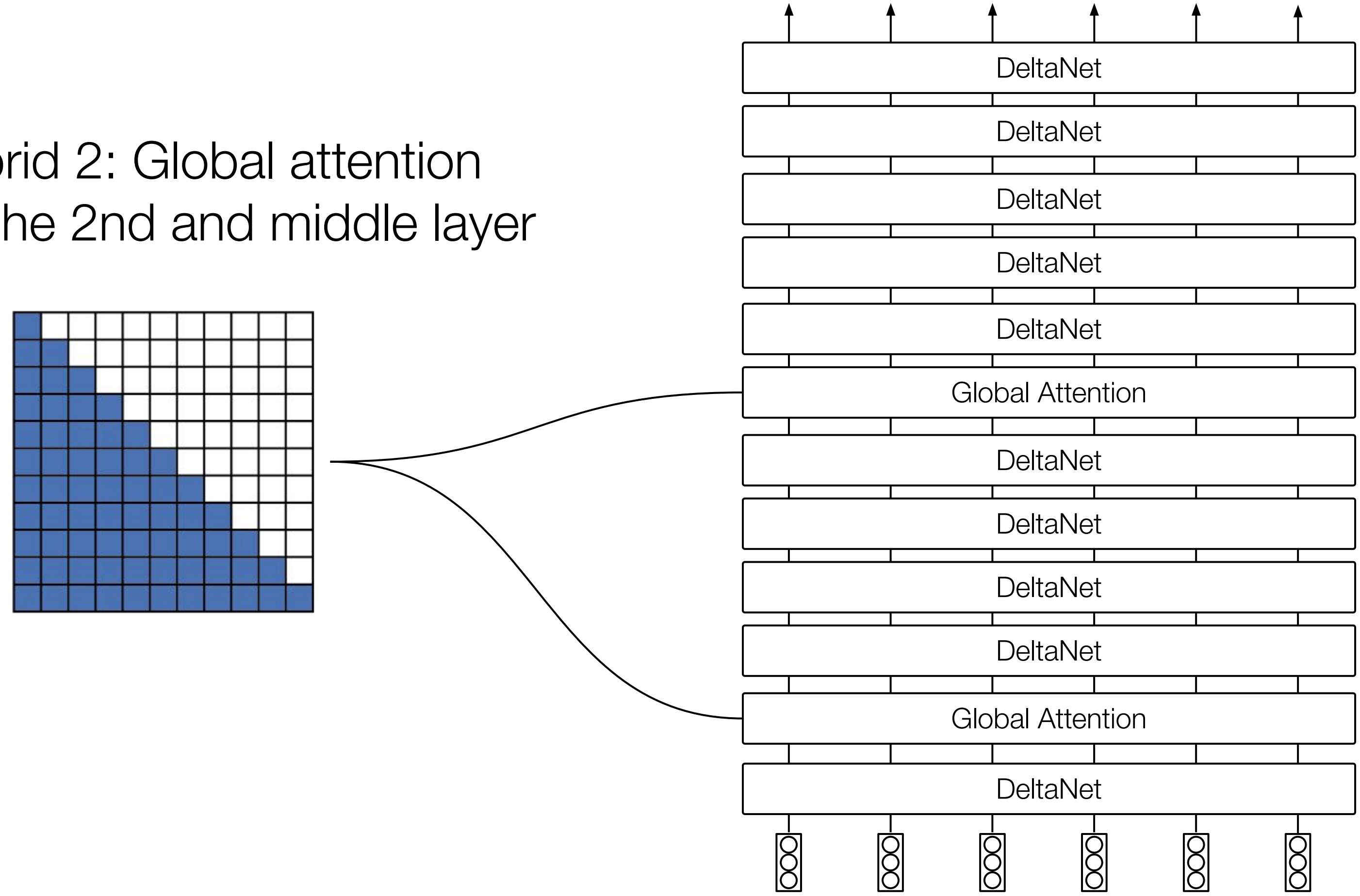
Hybridizing DeltaNet

Hybrid 1: Sliding window attention every other layer



Hybridizing DeltaNet

Hybrid 2: Global attention on the 2nd and middle layer

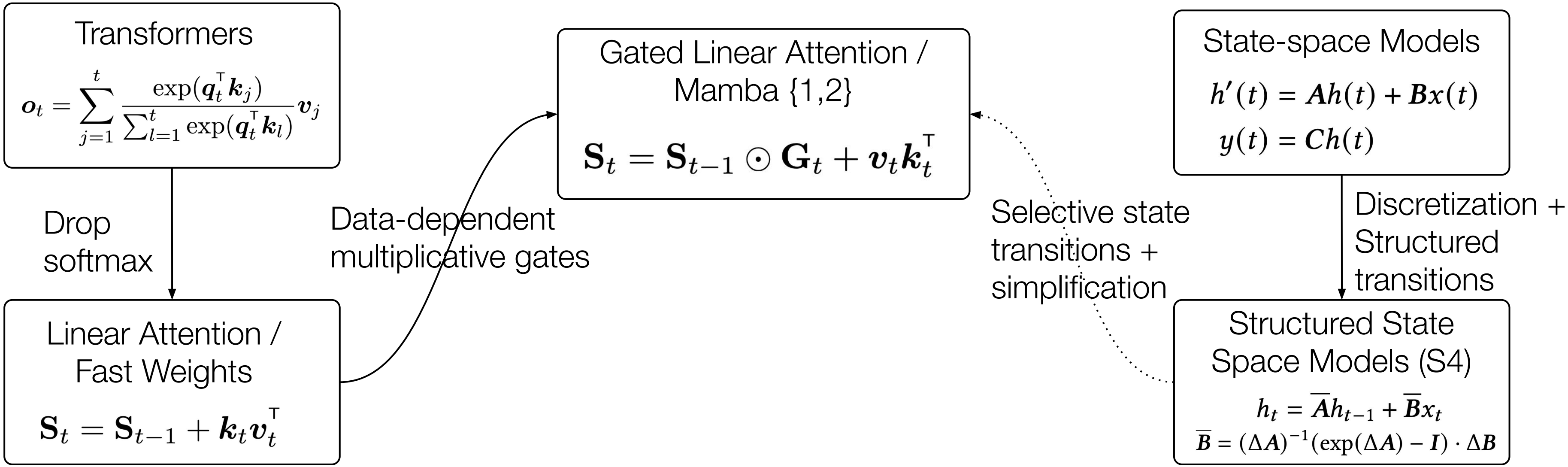


Hybrid DeltaNet: Performance

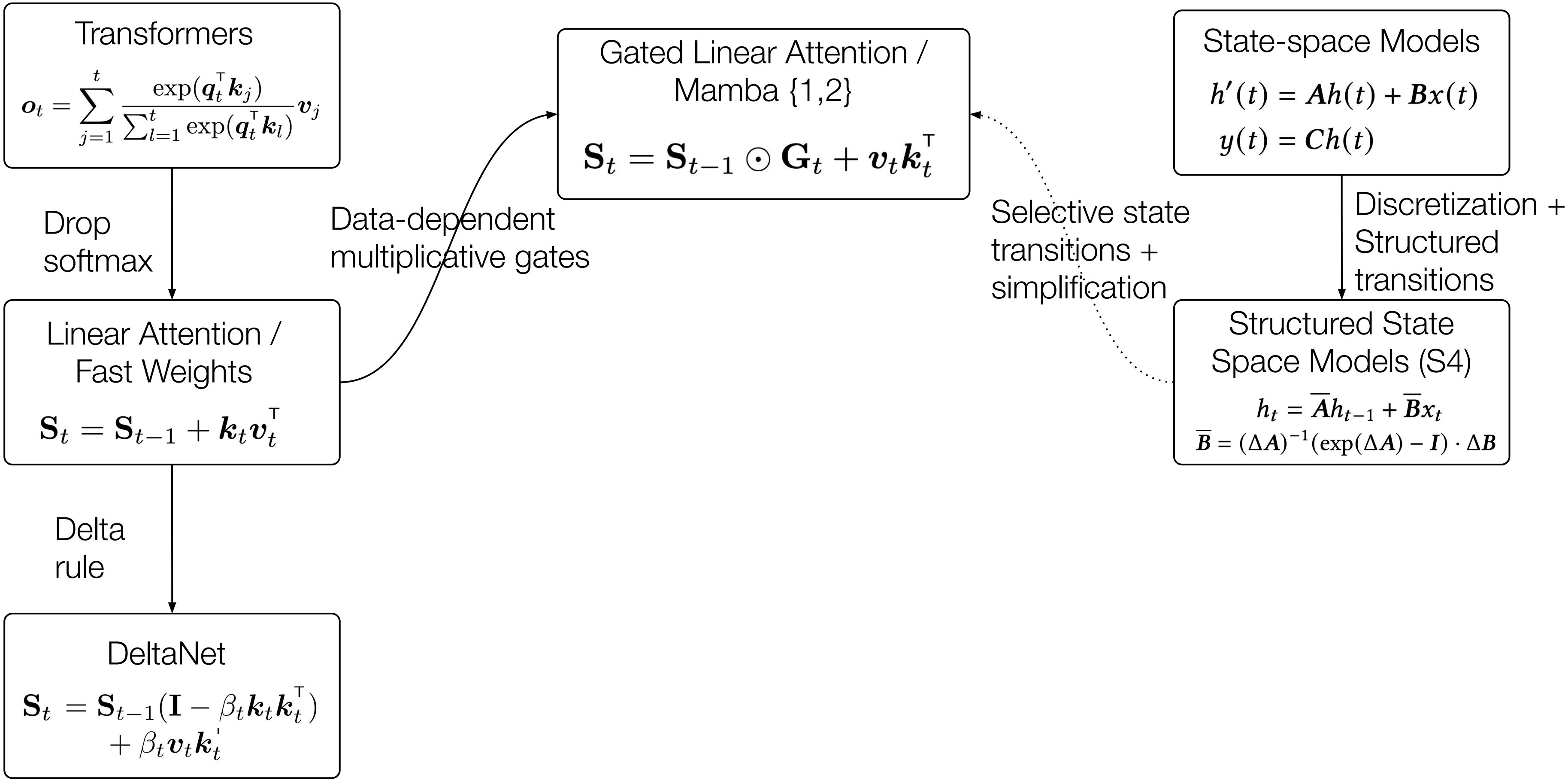
Model	PPL↓	LM Eval↑	Retrieval↑
Transformer++	16.9	50.9	41.8
RetNet	18.6	48.9	30.6
Mamba	17.1	50.0	27.6
Gated Linear Attention	17.2	51.1	37.7
DeltaNet	16.9	51.6	34.7
Hybrid 1: DeltaNet + Sliding window attention	16.6	52.1	40.0
Hybrid 2: DeltaNet + Global attention on 2 layers	16.6	51.8	47.9

1.3B models trained on 100B tokens

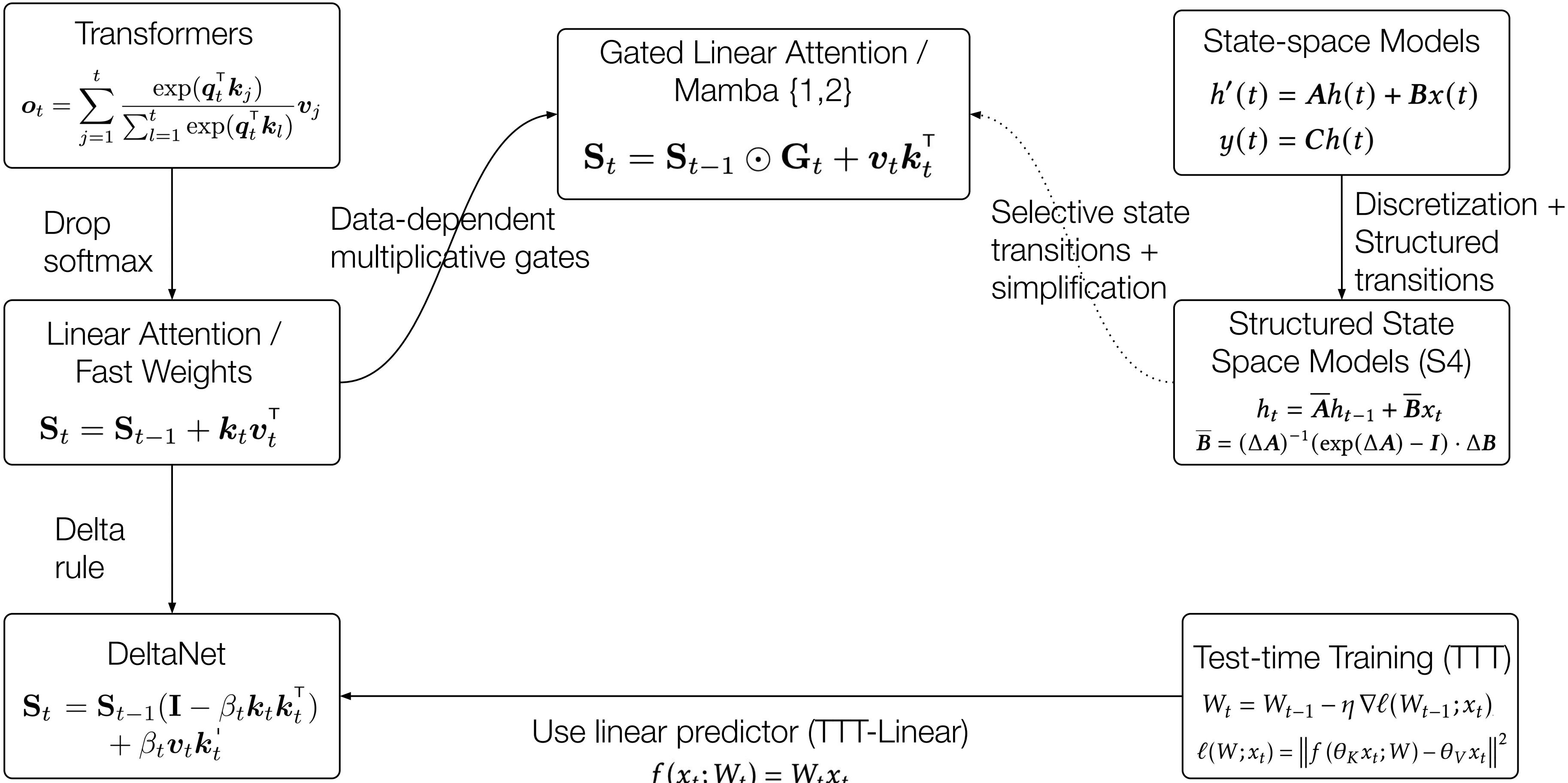
That which we call an SSM by any other name would perform just as well…



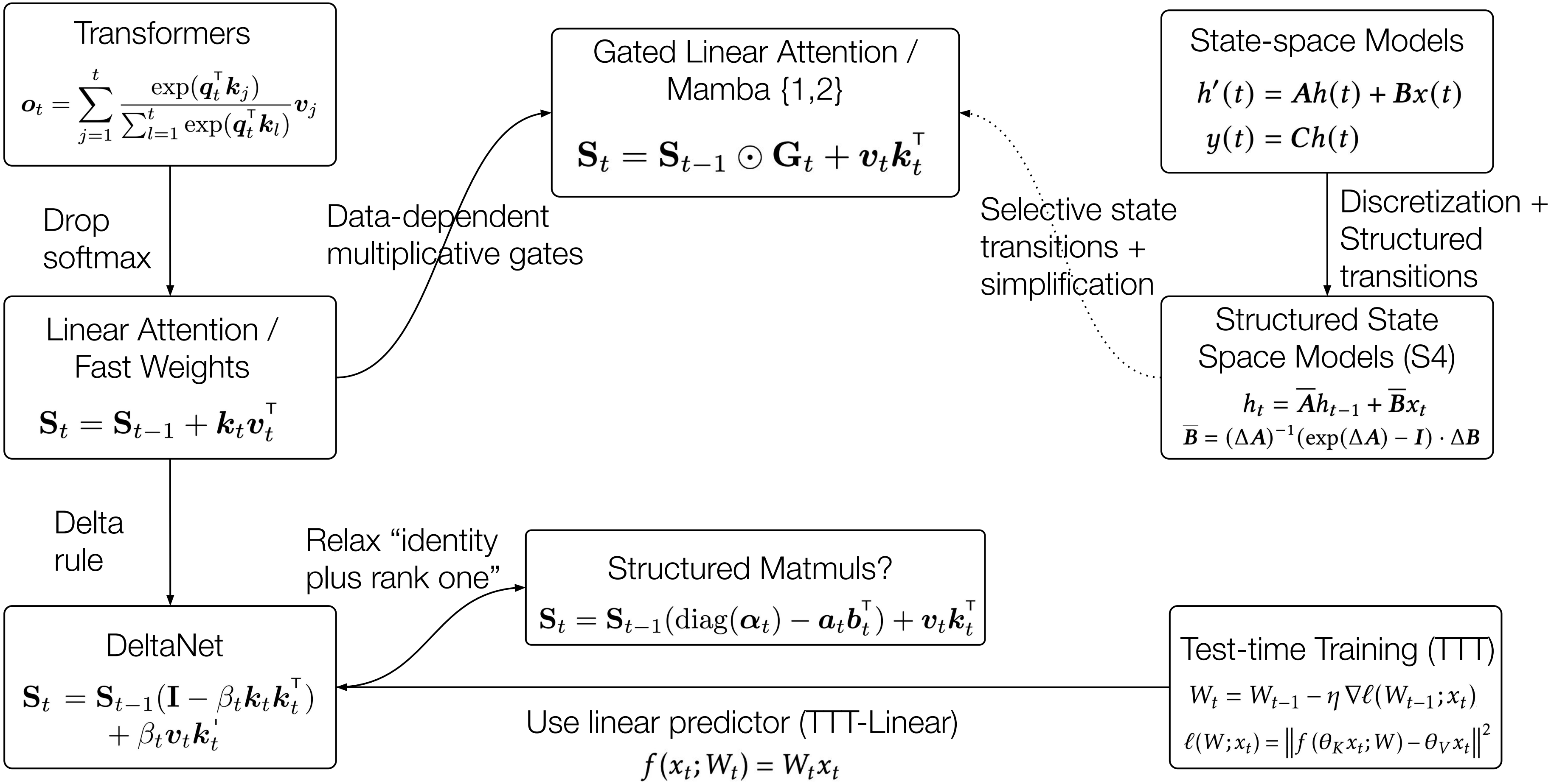
That which we call an SSM by any other name would perform just as well…



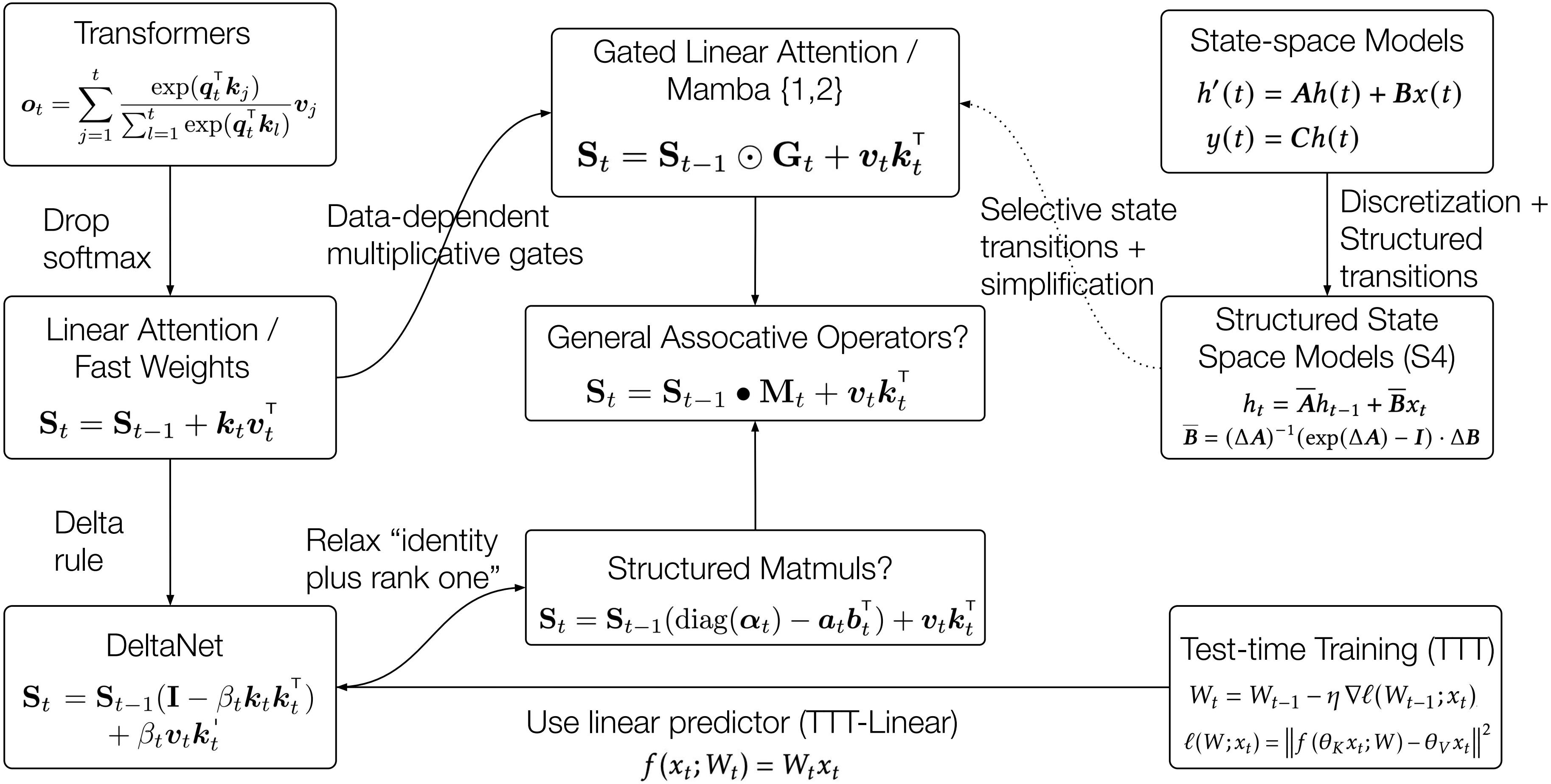
That which we call an SSM by any other name would perform just as well…



That which we call an SSM by any other name would perform just as well…



That which we call an SSM by any other name would perform just as well…



Takeways

Linear attention and SSMs have trouble with recall-oriented tasks → DeltaNet operationalizes a key-value retrieval/update mechanism.

Reparameterizing DeltaNet can enable parallelization via memory-efficient representations of Householder matrices.

Hybrid token strategies work well.

Parting thoughts

Some type of attention-like retrieval mechanism is likely necessary for the capabilities we want in our LLMs.

Language is still probably not the most impactful domain in which to explore subquadratic models.

Thanks!