Mamba Selective State Space Sequence Model

Ratish Puduppully

IT University of Copenhagen

Partly based on tutorial by Sasha Rush (https://www.youtube.com/watch?v=dVH1dRoMPBc)

Goal: Large Language Models over Long Context

Detailed task description with example

Goal: Large Language Models over Long Context

Long form generation

write me recipe for Hollandaise Over Roasted Potatoes

Here's a recipe for **Hollandaise Over Roasted Potatoes**, a delicious combination of crispy roasted potatoes topped with creamy, rich hollandaise sauce.

Ingredients:

For the Roasted Potatoes:

- 1.5 lbs (700g) baby potatoes, halved or quartered
- 2 tbsp olive oil
- 1 tsp garlic powder
- 1 tsp smoked paprika
- 1 tsp dried rosemary or thyme
- Salt and pepper to taste

For the Hollandaise Sauce:

- 3 large egg yolks
- 1 tbsp lemon juice
- ½ cup (1 stick) unsalted butter, melted and warm
- Salt to taste
- A pinch of cayenne pepper or white pepper (optional)

Instructions:

For the Roasted Potatoes:

1. **Preheat the oven** to 400°F (200°C).
2. **Prepare the potatoes**: In a large bowl, toss the halved or quartered potatoes with olive oil, garlic powder, smoked paprika, dried rosemary or thyme, salt, and pepper.
3. **Roast the potatoes**: Spread the potatoes on a baking sheet in a single layer. Roast for 25–30 minutes, turning halfway through, until golden brown and crispy on the outside.
4. **Set aside** the roasted potatoes once done.

For the Hollandaise Sauce:

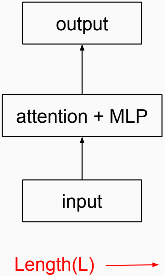
1. **Prepare a double boiler**: Place a heatproof bowl over a pot of simmering water, making sure the bottom of the bowl doesn't touch the water.
2. **Whisk the egg yolks** and lemon juice together in the bowl until thickened and lighter in color (about 1-2 minutes).
3. **Slowly add butter**: Very slowly drizzle in the warm melted butter while whisking constantly. Continue whisking until the sauce is thick and creamy.
4. **Season** the sauce with salt and a pinch of cayenne pepper (if desired).
5. **Remove from heat**: Keep the sauce warm by covering it and placing it near the stove or over warm water.

To Serve:

1. Place the roasted potatoes on a serving plate.
2. Drizzle the warm hollandaise sauce generously over the potatoes

Dominant Model: Transformer LM

- Interactions between all elements
- Highly optimized training

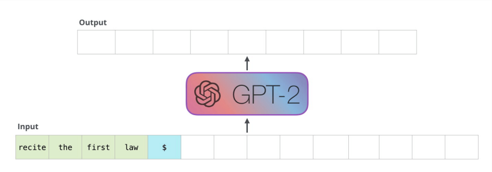


Challenge: Inference Scales Poorly

Challenges:

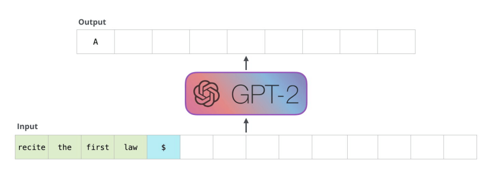
- O(L) memory scaling at inference
- KV cache grows with length

Autoregressive Generation in GPT-2



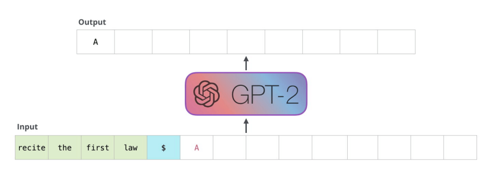
Example from `https://jalammar.github.io/illustrated-gpt2`

Autoregressive Generation in GPT-2



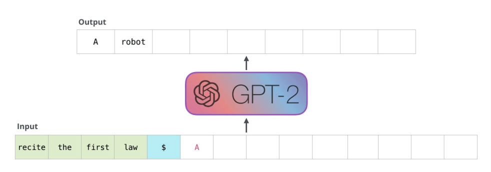
Example from https://jalammar.github.io/illustrated-gpt2

Autoregressive Generation in GPT-2



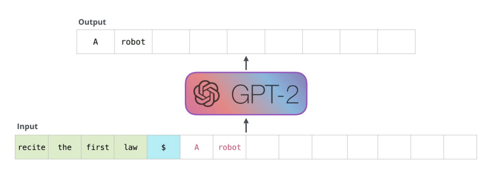
Example from `https://jalammar.github.io/illustrated-gpt2`

Autoregressive Generation in GPT-2



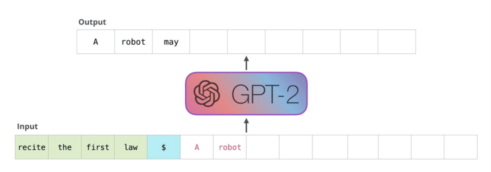
Example from https://jalammar.github.io/illustrated-gpt2

Autoregressive Generation in GPT-2



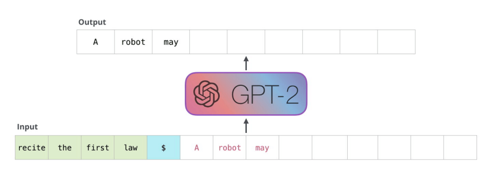
Example from https://jalammar.github.io/illustrated-gpt2

Autoregressive Generation in GPT-2



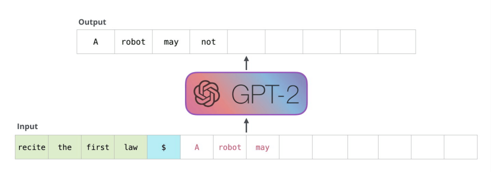
Example from https://jalammar.github.io/illustrated-gpt2

Autoregressive Generation in GPT-2



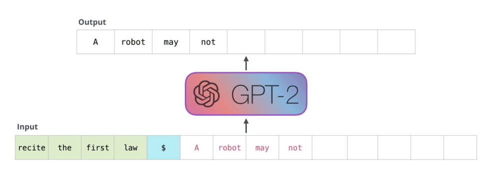
Example from https://jalammar.github.io/illustrated-gpt2

Autoregressive Generation in GPT-2



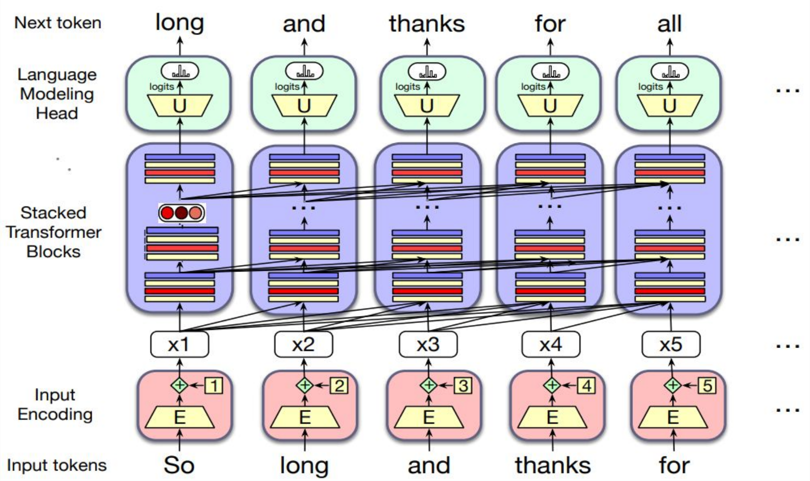
Example from `https://jalammar.github.io/illustrated-gpt2`

Autoregressive Generation in GPT-2



Example from https://jalammar.github.io/illustrated-gpt2

Transformer Blocks



SLP - Jurafsky and Martin (3rd ed)

Query, Key, and Value Projections

We use matrices to project each input vector $\mathbf{x}_i$ into representations of its role as query, key, or value:

- $\mathbf{W}^Q$ for queries
- $\mathbf{W}^K$ for keys
- $\mathbf{W}^V$ for values

Mathematically:

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q, \quad \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K, \quad \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V.$$

Self-Attention Equations

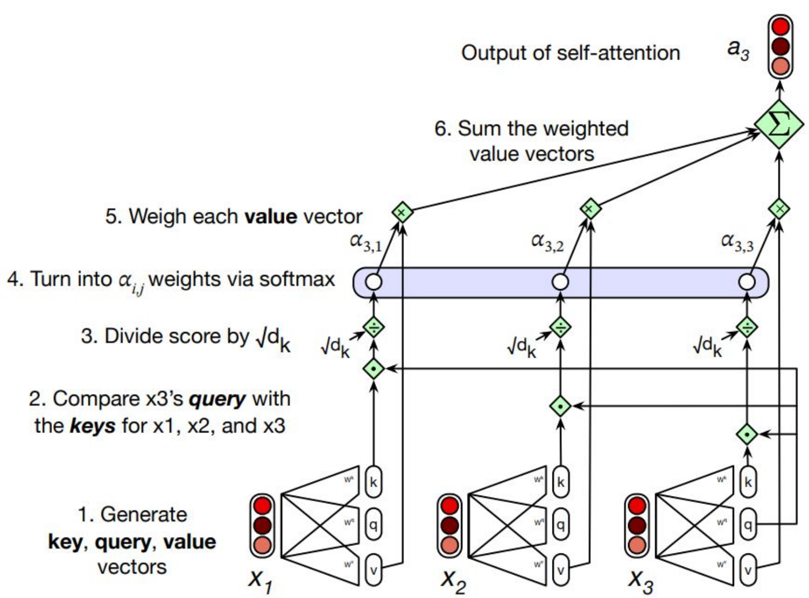
$$q_i = x_i\, W^Q, \quad k_j = x_j\, W^K, \quad v_j = x_j\, W^V$$

$$\mathsf{score}(x_i, x_j) \;=\; \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

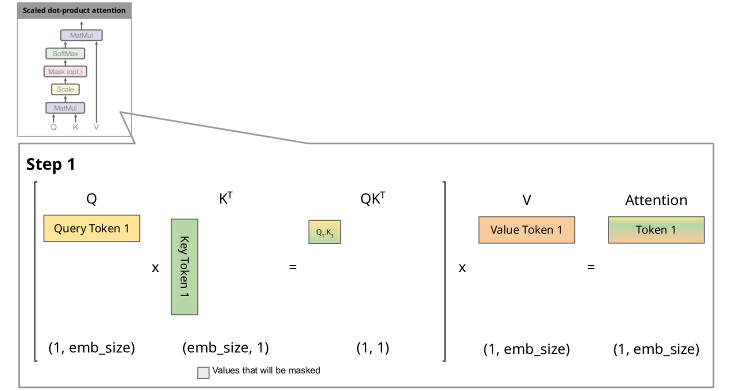
$$\alpha_{i,j} = \mathrm{softmax}\big(\mathsf{score}(x_i, x_j)\big) \quad (\forall\, j \le i)$$

$$a_i = \sum_{j \le i} \alpha_{i,j}\, v_j$$

Calculating a3



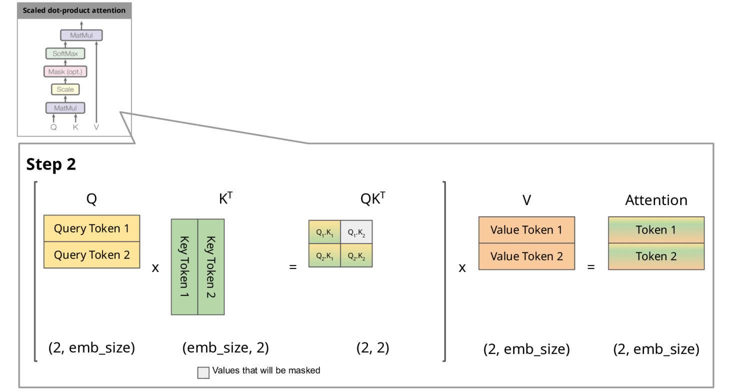
Dot-product Attention in Decoder



Zoom-in! (simplified without Scale and Softmax)

Example from https://medium.com/@joaolages/kv-caching-explained-276520203249

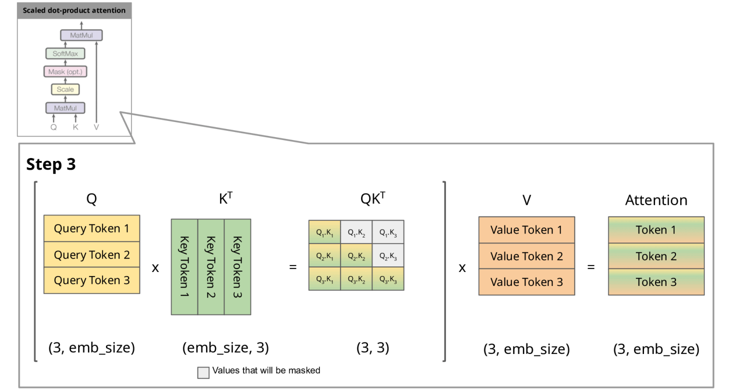
Dot-product Attention in Decoder



Zoom-in! (simplified without Scale and Softmax)

Example from https://medium.com/@joaolages/kv-caching-explained-276520203249

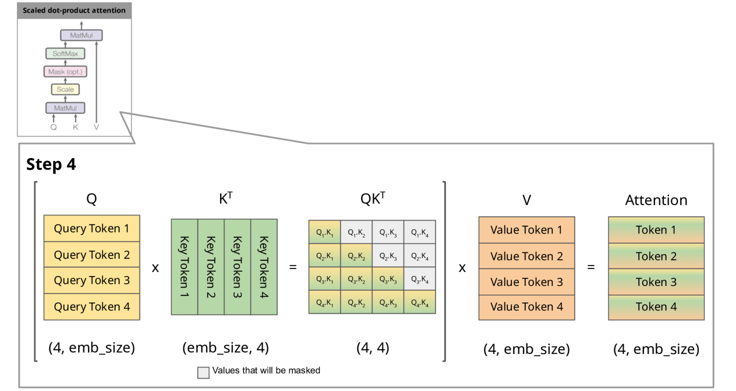
Dot-product Attention in Decoder



Zoom-in! (simplified without Scale and Softmax)

Example from https://medium.com/@joaolages/kv-caching-explained-276520203249

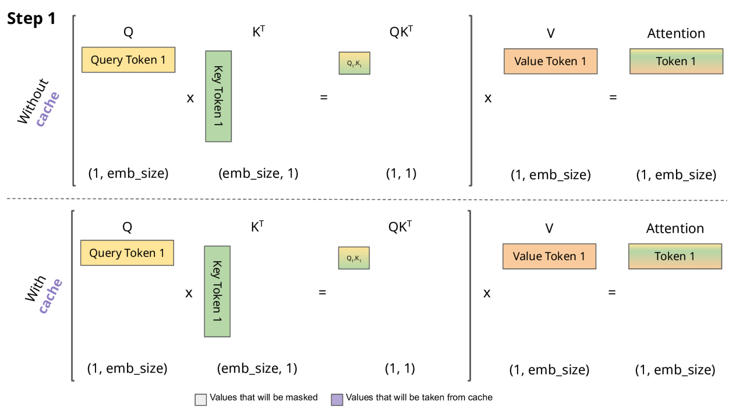
Dot-product Attention in Decoder



Zoom-in! (simplified without Scale and Softmax)

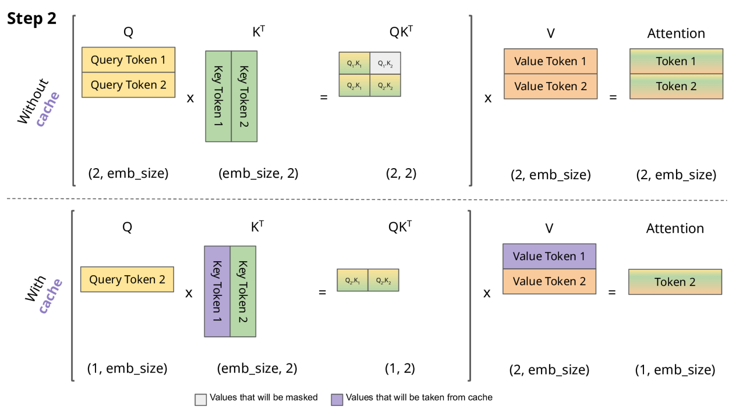
Example from https://medium.com/@joaolages/kv-caching-explained-276520203249

Dot-product Attention in Decoder with KV-cache



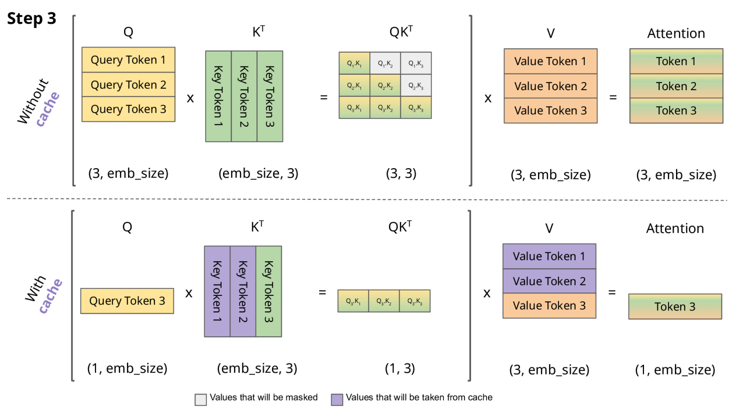
Example from https://medium.com/@joaolages/kv-caching-explained-276520203249

Dot-product Attention in Decoder with KV-cache



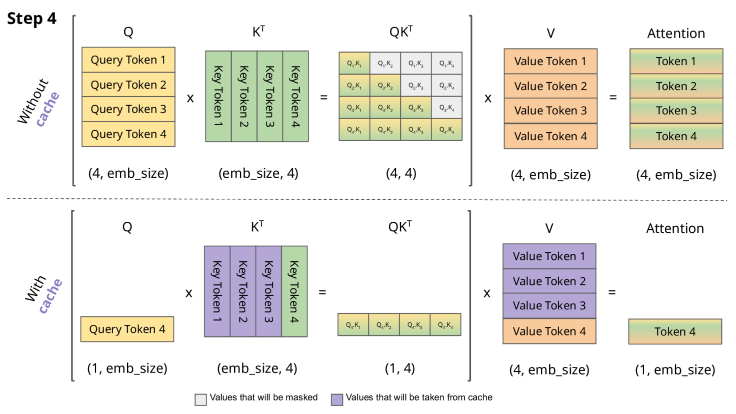
Example from https://medium.com/@joaolages/kv-caching-explained-276520203249

Dot-product Attention in Decoder with KV-cache



Example from https://medium.com/@joaolages/kv-caching-explained-276520203249

Dot-product Attention in Decoder with KV-cache



Example from https://medium.com/@joaolages/kv-caching-explained-276520203249

Dot-product Attention in Decoder with KV-cache

- Earlier: FLOPs for $QK^T \propto L \times L$

Dot-product Attention in Decoder with KV-cache

- Earlier: FLOPs for $QK^T \propto L \times L$
- With KV-cache $\propto L$

Dot-product Attention in Decoder with KV-cache

- Earlier: FLOPs for $QK^T \propto L \times L$
- With KV-cache $\propto L$
- Memory cost: $L \times \text{num\_layers} \times 2 \times d$

Challenge: Inference Scales Poorly

Challenges:

- O(L) memory scaling at inference
- KV cache grows with length

Challenge: Inference Scales Poorly

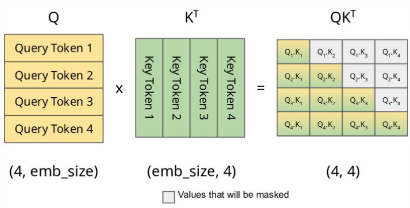
	Available params	Active params	KV cache (256K context, 16bit)
Mistral	7.2B	7.2B	32GB
Mixtral 8x7B	46.7B	12.9B	32GB
LLaMA-3.1 8B	8B	8B	32GB
Mixtral 8x22B	141B	39B	56GB
Mistral-Large-2	123B	123B	88GB
LLaMA-3.1 70B	70B	70B	80GB
LLaMA-3.1 405B	405B	405B	252GB
Jamba-1.5-Mini	52B	12B	4GB
Jamba-1.5-Large	398B	94B	9GB

Table 1: Comparison of Jamba-1.5-Mini, Jamba-1.5-Large and recent open models in terms of total available parameters, active parameters, and KV cache memory on long contexts. Jamba-1.5-Mini and Jamba-1.5-Large provide substantial reductions in the KV cache memory requirements.

Jamba 1.5 (2024)

Challenge: Training Scales Poorly

$O(L^2)$ scaling in sequence length



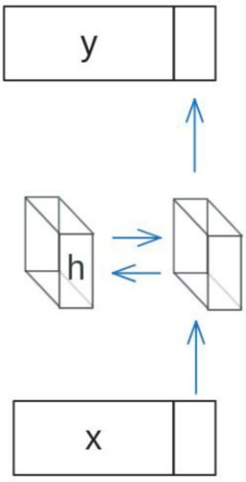
Mamba (and friends)

Newer subquadratic architectures targeting large language models

- Mamba (Gu and Dao 2023)
- S5 (Smith et al. 2022)
- Based (Arora et al. 2024)
- Griffin (De et al. 2024)
- GLA (Yang et al. 2023)
- RetNet (Sun et al. 2023)

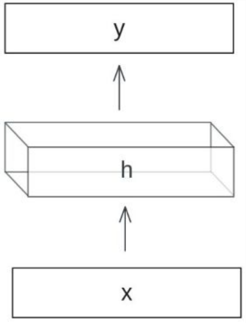
Property: Fixed-Size Memory

Constant memory at inference

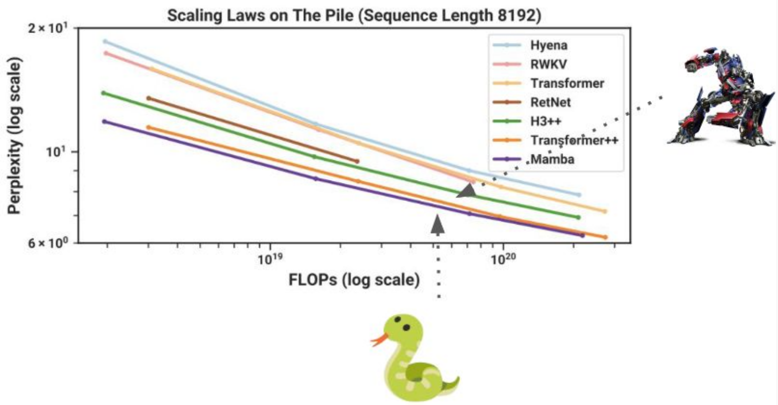


Property: Linear Training Scaling

Linear compute in length



Why is this important now?



Mamba (Gu and Dao 2023)

Questions?

Outline

How do I *understand* the model?

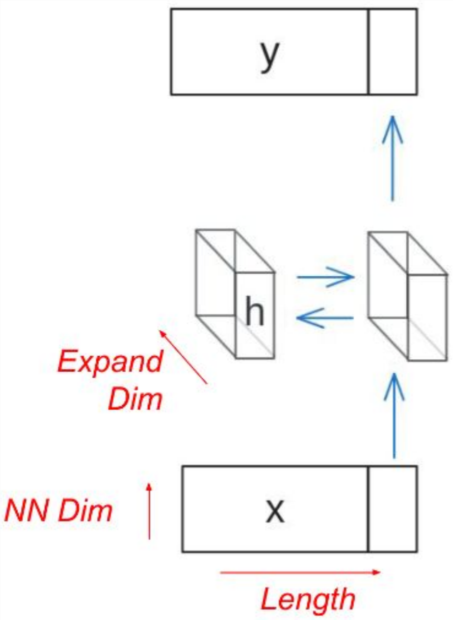
How do I *compute* the model?

How do I *design* an effective version?

How do I *scale* it to its max state?

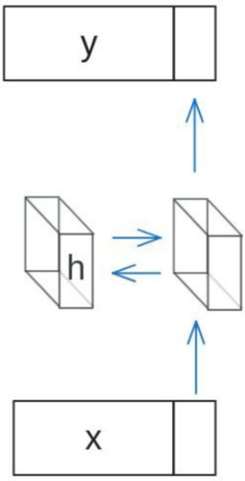
How do I understand the model?

General Form of Fixed-State Model



Prelim 1: Vanilla RNN

$$h_k = \sigma\left(\bar{A}h_{k-1} + \bar{B}x_k\right)$$
$$y_k = Ch_k$$



Challenges

- Difficult to learn well
- Inefficient to train historically

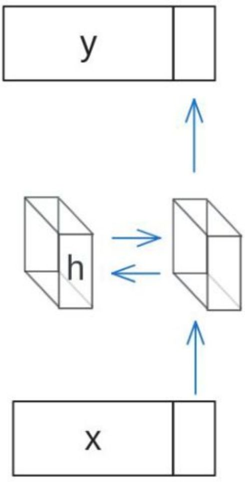
Prelim 2: Linear Time Invariant (LTI)

$$h_k = \left(\bar{A}h_{k-1} + \bar{B}x_k\right)$$
$$y_k = Ch_k$$

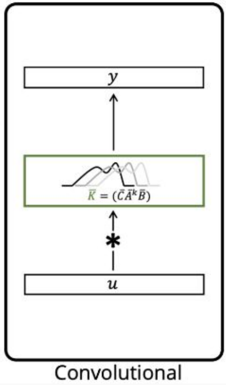
Challenges

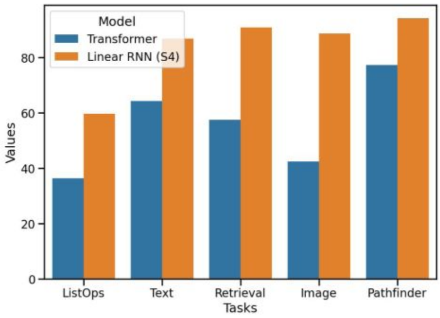
- Thought to be hard to learn

LSSL (Gu et al. 2021)



Context (S4): LTI is fast and relatively effective





S4 (Gu et al. 2021)

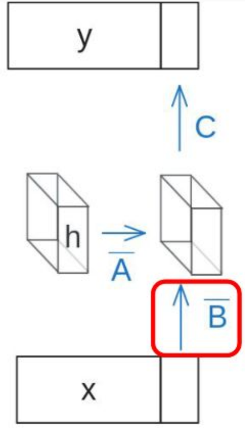
Roadblock: LTI is not great at LM

	Model	Param (M)	TFLOPs	Overall
	Attention	125	2.46	**11.01** (2.40)
LTI	Long Conv	128	1.74	16.98 (2.83)
LTI	H3	168	2.55	12.06 (2.49)
LTI	Hyena	158	2.41	11.60 (2.45)
LTI	RWKV	169	2.08	11.64 (2.45)
	Attention	360	6.23	**9.44** (2.25)
LTI	Long Conv	360	4.08	13.13 (2.57)
LTI	H3	357	4.85	10.38 (2.34)
LTI	Hyena	358	5.03	10.07 (2.31)
LTI	RWKV	351	4.31	9.79 (2.28)

Based (Arora et al. 2024)

Failure Case 1: Filtering

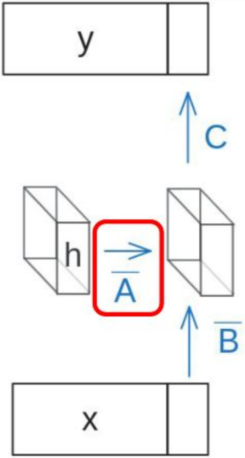
$$h_k = (\bar{A}h_{k-1} + \bar{B}x_k)$$
$$y_k = Ch_k$$



LTI cannot ignore tokens!
Example: Junk text on the web
(copyright, ad copy)

Failure Case 2: Reset

$$h_k = (\bar{A} h_{k-1} + \bar{B} x_k)$$
$$y_k = C h_k$$

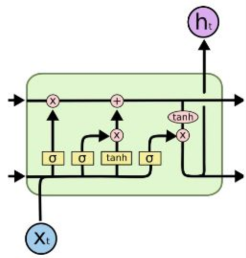


LTI cannot reset history!
Example: Start of a new article, chapter in a long document

Historical Parallel: RNN → LSTM to Allow Gating

$$h_k = \sigma\left(\bar{A}h_{k-1} + \bar{B}x_k\right)$$
$$y_k = Ch_k$$

RNN



LSTM

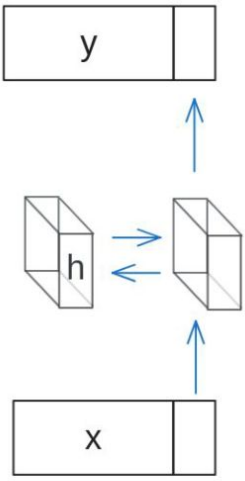
Linear Time Varying (LTV) Model

$$h_k = \left(\bar{A}_k h_{k-1} + \bar{B}_k x_k\right)$$
$$y_k = C_k h_k$$

Contribution: Let parameters change based on position:

$$\text{Reset} \quad \rightarrow \quad \bar{A}_k = 0$$

$$\text{Filter} \quad \rightarrow \quad \bar{B}_k = 0$$

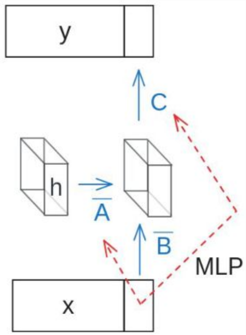


Generating Linear Time Varying (LTV)

$$h_k = \left(\bar{A}_{\color{red}k} h_{k-1} + \bar{B}_{\color{red}k} x_k\right)$$
$$y_k = C_{\color{red}k} h_k$$

How to obtain $\bar{A}_{\color{red}k}, \bar{B}_{\color{red}k}, C_{\color{red}k}$:

- *Produce as a function of x*

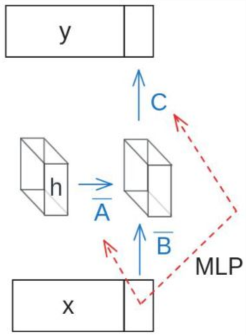


Results: LTV is a Promising Approach For LMs

✓Fixes central issues with LTI
✓Maintains fixed-sized state

But ....

How do you run it efficiently?



Questions?

How do I compute the model?

Background: LTI

$$h_k = \left(\bar{A}h_{k-1} + \bar{B}x_k\right)$$
$$y_k = Ch_k$$

$$h_1 = \bar{B}x_1 \quad y_1 = C\bar{B}x_1$$

Background: LTI

$$
\begin{aligned}
h_k &= \left(\bar{A}h_{k-1} + \bar{B}x_k\right) \\
y_k &= Ch_k
\end{aligned}
$$

$$h_1 = \bar{B}x_1 \quad y_1 = C\bar{B}x_1$$

$$h_2 = \bar{A}\bar{B}x_1 + \bar{B}x_2 \quad y_2 = C\bar{A}\bar{B}x_1 + C\bar{B}x_2$$

Background: LTI

$$h_k = \left(\bar{A}h_{k-1} + \bar{B}x_k\right)$$
$$y_k = Ch_k$$

$$h_1 = \bar{B}x_1 \quad y_1 = C\bar{B}x_1$$

$$h_2 = \bar{A}\bar{B}x_1 + \bar{B}x_2 \quad y_2 = C\bar{A}\bar{B}x_1 + C\bar{B}x_2$$

$$\vdots$$

$$y_{k+1} = C\bar{A}^k\bar{B}x_1 + C\bar{A}^{k-1}\bar{B}x_2 + \cdots + C\bar{A}\bar{B}x_k + C\bar{B}x_{k+1}$$

Background: LTI

$$h_k = \left(\bar{A}h_{k-1} + \bar{B}x_k\right)$$
$$y_k = Ch_k$$

$$h_1 = \bar{B}x_1 \quad y_1 = C\bar{B}x_1$$

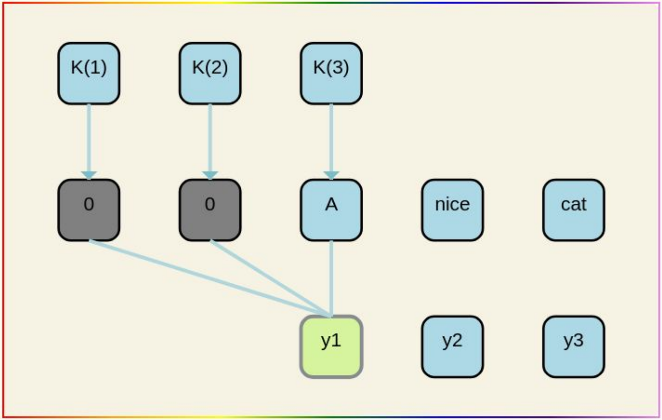
$$h_2 = \bar{A}\bar{B}x_1 + \bar{B}x_2 \quad y_2 = C\bar{A}\bar{B}x_1 + C\bar{B}x_2$$

$$\vdots$$

$$y_{k+1} = C\bar{A}^k\bar{B}x_1 + C\bar{A}^{k-1}\bar{B}x_2 + \cdots + C\bar{A}\bar{B}x_k + C\bar{B}x_{k+1}$$

$$\bar{y} = \bar{K} * x$$
$$\bar{K} \in \mathbb{R}^{L+1} = (C\bar{B}, C\bar{A}\bar{B}, \ldots, C\bar{A}^L\bar{B})$$

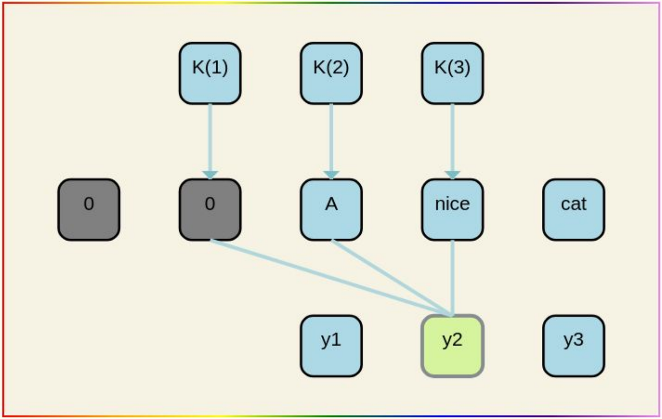
Background: LTI



- Convolution
- Computed in parallel

Example from `https://chus.space/blog/2024/ssm_2_networks/`

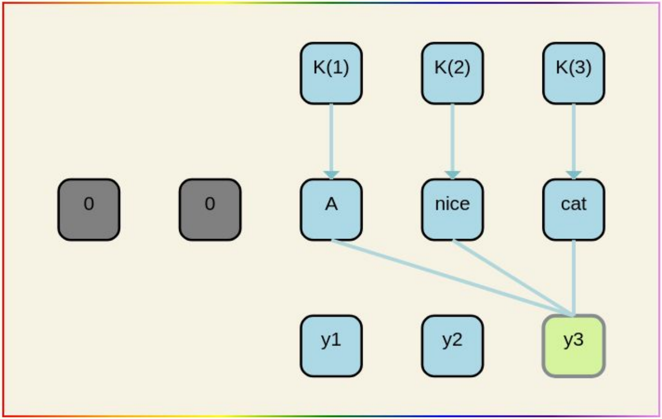
Background: LTI



- Convolution
- Computed in parallel

Example from `https://chus.space/blog/2024/ssm_2_networks/`

Background: LTI



- Convolution
- Computed in parallel

Example from `https://chus.space/blog/2024/ssm_2_networks/`

In LTV?

$$h_k = \left(\bar{A}_k h_{k-1} + \bar{B}_k x_k\right)$$
$$y_k = C_k h_k$$

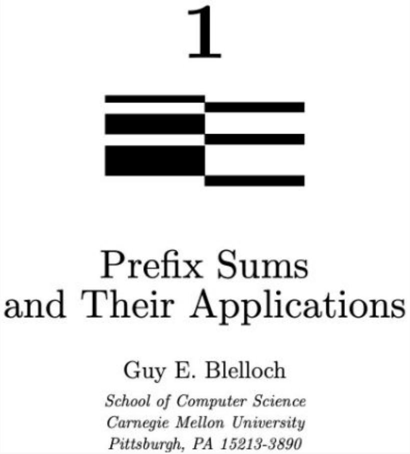
- No convolutional form (time-varying parameters).
- Recurrent form required.
- Sequential bottleneck?

In LTV?

$$h_k = \left(\bar{A}_{k} h_{k-1} + \bar{B}_{k} x_k\right)$$
$$y_k = C_{k} h_k$$

- No convolutional form (time-varying parameters).
- Recurrent form required.
- Sequential bottleneck?
- Solution: Parallel scan.

Associative Scan



1

Prefix Sums
and Their Applications

Guy E. Blelloch

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

See also Martin et al. 2017, Smith et al. 2022 (S5)

Associative Scan

- Parallelizes a sequential operation to achieve faster computation.

Associative Scan

- Parallelizes a sequential operation to achieve faster computation.
- Reduces complexity from $O(L)$ to $O(\log L)$.

Associative Scan

- Parallelizes a sequential operation to achieve faster computation.
- Reduces complexity from $O(L)$ to $O(\log L)$.
- Requires an associative operator (e.g., matrix multiplication, matrix addition).
  - Example: Matrix Addition $(A + B) + C = A + (B + C)$

“Hello world” of Parallel Scans: Cumulative Sum

$$y_k = \sum_{i=1}^{k} x_i$$

$$\begin{bmatrix} 3 & 1 & 7 & 0 & 4 & 1 & 6 & 3 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \end{bmatrix}$$

$$y_k = \sum_{i=1}^{k} x_i$$

$$\begin{bmatrix} 3 & 1 & 7 & 0 & 4 & 1 & 6 & 3 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \end{bmatrix}$$

$$y_k = \sum_{i=1}^{k} x_i$$

$$\begin{bmatrix} 3 & 4 & 11 & 11 & 15 & 16 & 22 & 25 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 \end{bmatrix}$$

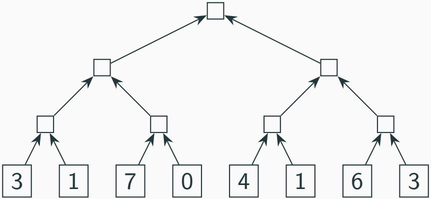
$$\begin{bmatrix} 3 & 1 & 7 & 0 & 4 & 1 & 6 & 3 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \end{bmatrix}$$

$$\begin{aligned} h_k &= h_{k-1} + x_k \\ y_k &= h_k \end{aligned}$$

$$\begin{bmatrix} 3 & 4 & 11 & 11 & 15 & 16 & 22 & 25 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 \end{bmatrix}$$

$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3] \quad \longrightarrow \quad [3 \quad 4 \quad 11 \quad 11 \quad 15 \quad 16 \quad 22 \quad 25]$

Up Sweep



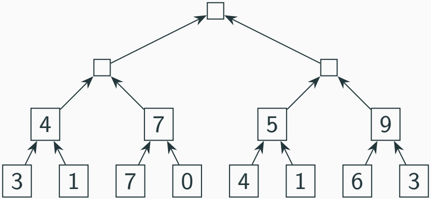
Down Sweep

sum[v] = sum[L[v]] + sum[R[v]]

prescan[L[v]] = prescan[v]
prescan[R[v]] = sum[L[v]] + prescan[v]

[3 1 7 0 4 1 6 3] $\longrightarrow$ [3 4 11 11 15 16 22 25]

Up Sweep

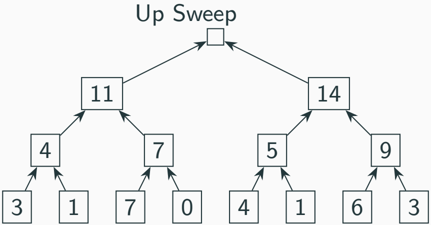


Down Sweep

sum[v] = sum[L[v]] + sum[R[v]]

prescan[L[v]] = prescan[v]
prescan[R[v]] = sum[L[v]] + prescan[v]

$$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3] \longrightarrow [3 \quad 4 \quad 11 \quad 11 \quad 15 \quad 16 \quad 22 \quad 25]$$



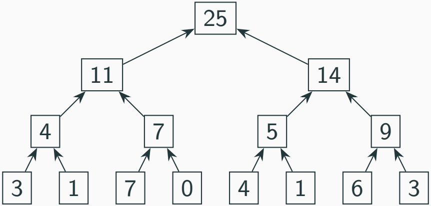
Down Sweep

sum[v] = sum[L[v]] + sum[R[v]]

prescan[L[v]] = prescan[v]
prescan[R[v]] = sum[L[v]] + prescan[v]

$$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3] \quad \longrightarrow \quad [3 \quad 4 \quad 11 \quad 11 \quad 15 \quad 16 \quad 22 \quad 25]$$

Up Sweep



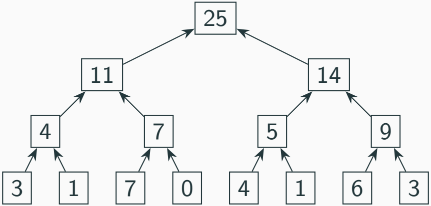
Down Sweep

sum[v] = sum[L[v]] + sum[R[v]]

prescan[L[v]] = prescan[v]
prescan[R[v]] = sum[L[v]] + prescan[v]

$$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3] \longrightarrow [3 \quad 4 \quad 11 \quad 11 \quad 15 \quad 16 \quad 22 \quad 25]$$

Up Sweep

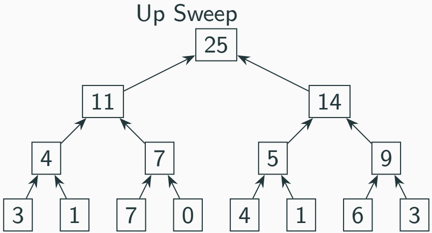


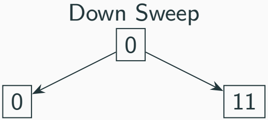
Down Sweep

sum[v] = sum[L[v]] + sum[R[v]]

prescan[L[v]] = prescan[v]
prescan[R[v]] = sum[L[v]] + prescan[v]

[3 1 7 0 4 1 6 3] $\longrightarrow$ [3 4 11 11 15 16 22 25]

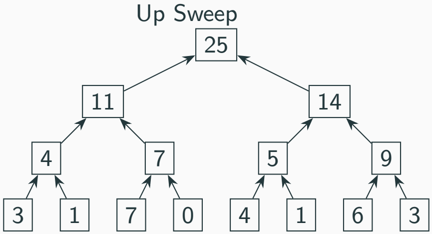


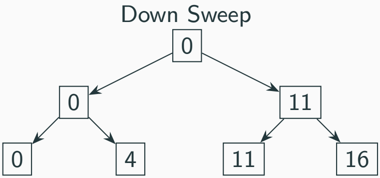


sum[v] = sum[L[v]] + sum[R[v]]

prescan[L[v]] = prescan[v]
prescan[R[v]] = sum[L[v]] + prescan[v]

$$[3\quad 1\quad 7\quad 0\quad 4\quad 1\quad 6\quad 3] \longrightarrow [3\quad 4\quad 11\quad 11\quad 15\quad 16\quad 22\quad 25]$$

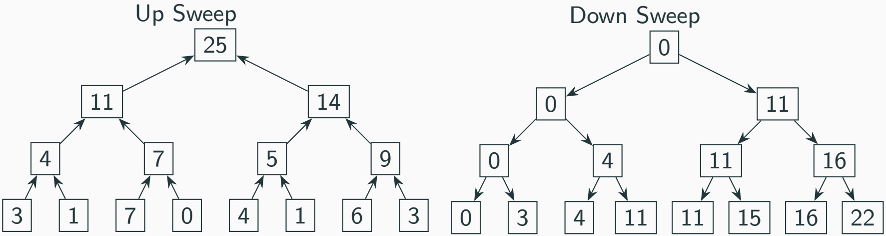




sum[v] = sum[L[v]] + sum[R[v]]

prescan[L[v]] = prescan[v]
prescan[R[v]] = sum[L[v]] + prescan[v]

$$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3] \quad \longrightarrow \quad [3 \quad 4 \quad 11 \quad 11 \quad 15 \quad 16 \quad 22 \quad 25]$$



sum[v] = sum[L[v]] + sum[R[v]]

prescan[L[v]] = prescan[v]
prescan[R[v]] = sum[L[v]] + prescan[v]

$$h_k = \bar{A}_k h_{k-1} + \bar{B}_k x_k$$

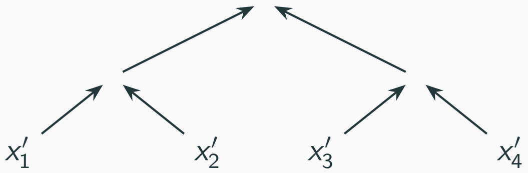
$$y_k = C_k h_k$$

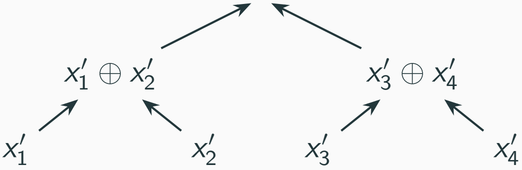
Linear recurrence?

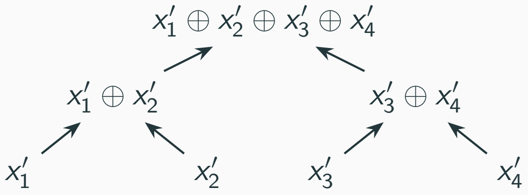
New primitives

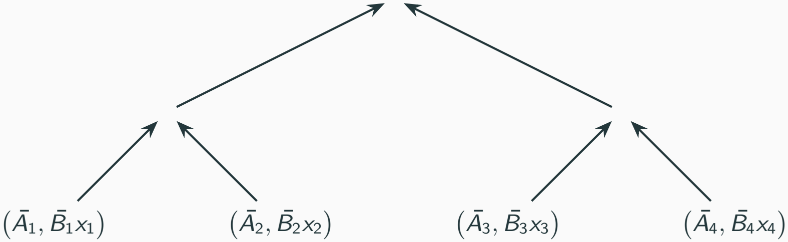
$$x_k' := \left(\bar{A}_k, \bar{B}_k x_k\right)$$

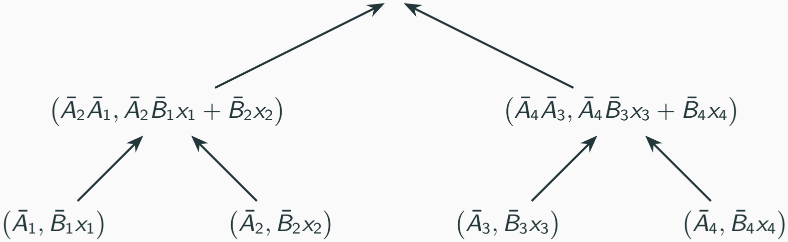
$$(a_1, b_1) \oplus (a_2, b_2) := (a_2 a_1, a_2 b_1 + b_2)$$


x_1'
x_2'
x_3'
x_4'


$x'_1 \oplus x'_2$
$x'_3 \oplus x'_4$
x'_1
x'_2
x'_3
x'_4


$x_1' \oplus x_2' \oplus x_3' \oplus x_4'$
$x_1' \oplus x_2'$
$x_3' \oplus x_4'$
x_1'
x_2'
x_3'
x_4'


$(\bar{A}_1, \bar{B}_1 x_1)$
$(\bar{A}_2, \bar{B}_2 x_2)$
$(\bar{A}_3, \bar{B}_3 x_3)$
$(\bar{A}_4, \bar{B}_4 x_4)$

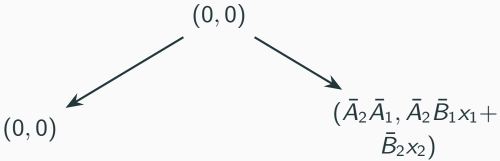

$(\bar{A}_2\bar{A}_1, \bar{A}_2\bar{B}_1x_1 + \bar{B}_2x_2)$
$(\bar{A}_4\bar{A}_3, \bar{A}_4\bar{B}_3x_3 + \bar{B}_4x_4)$
$(\bar{A}_1, \bar{B}_1x_1)$
$(\bar{A}_2, \bar{B}_2x_2)$
$(\bar{A}_3, \bar{B}_3x_3)$
$(\bar{A}_4, \bar{B}_4x_4)$

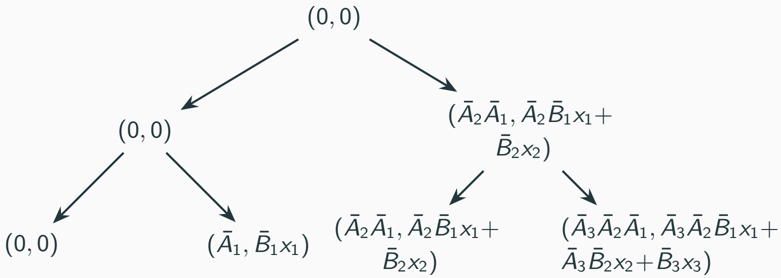
$$(\bar{A}_4\bar{A}_3\bar{A}_2\bar{A}_1, \bar{A}_4\bar{A}_3\bar{A}_2\bar{B}_1x_1 + \bar{A}_4\bar{A}_3\bar{B}_2x_2 + \bar{A}_4\bar{B}_3x_3 + \bar{B}_4x_4)$$

$$(\bar{A}_2\bar{A}_1, \bar{A}_2\bar{B}_1x_1 + \bar{B}_2x_2) \qquad (\bar{A}_4\bar{A}_3, \bar{A}_4\bar{B}_3x_3 + \bar{B}_4x_4)$$

$$(\bar{A}_1, \bar{B}_1x_1) \qquad (\bar{A}_2, \bar{B}_2x_2) \qquad (\bar{A}_3, \bar{B}_3x_3) \qquad (\bar{A}_4, \bar{B}_4x_4)$$

$$(0, 0)$$


$(0, 0)$
$(0, 0)$
$(\bar{A}_2\bar{A}_1, \bar{A}_2\bar{B}_1x_1 + \bar{B}_2x_2)$


$(0, 0)$
$(0, 0)$
$(\bar{A}_2\bar{A}_1, \bar{A}_2\bar{B}_1x_1 + \bar{B}_2x_2)$
$(0, 0)$
$(\bar{A}_1, \bar{B}_1x_1)$
$(\bar{A}_2\bar{A}_1, \bar{A}_2\bar{B}_1x_1 + \bar{B}_2x_2)$
$(\bar{A}_3\bar{A}_2\bar{A}_1, \bar{A}_3\bar{A}_2\bar{B}_1x_1 + \bar{A}_3\bar{B}_2x_2 + \bar{B}_3x_3)$

Results: Fast algorithms for LTV

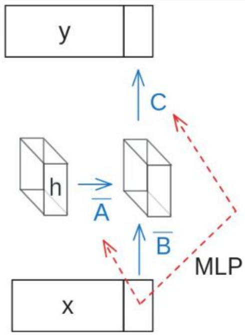
✓Can run LTV in parallel
✓Needs A, B, C to run
But ...
How do you produce A, B, C that work in practice?

Questions?

How do I design an effective version?

Reminder: LTV

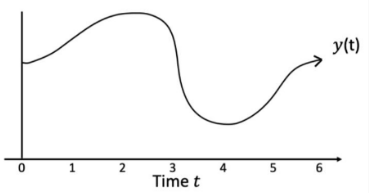
$$h_k = \bar{A}_{\color{red}k} h_{k-1} + \bar{B}_{\color{red}k} x_k$$
$$y_k = C_{\color{red}k} h_k$$



Continuous-Time State-Space Model

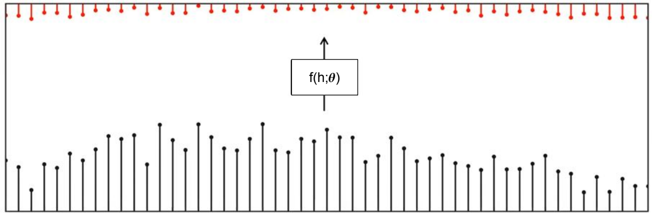
$$h'(t) = Ah(t) + B(t)x(t)$$
$$y(t) = C(t)h(t)$$

Imagine x, y was in continuous time, how do we model its dynamics?



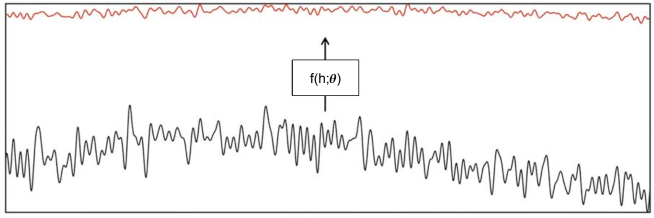
A New Approach to Linear Filtering and Prediction Problems (Kalman 1960)
Mamba (Gu & Dao 2023)

Goal: Sequence Model Maps Sequence to Sequence



Map 1D sequence to 1D sequence

SSMs: Map 1D function to 1D function



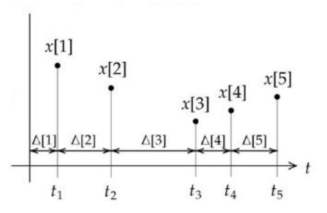
Discretization at Selected Ranges

Continuous:

$$h'(t) = Ah(t) + B(t)x(t)$$
$$y(t) = C(t)h(t)$$

Discrete:

$$h_k = \bar{A}_k h_{k-1} + \bar{B}_k x_k$$
$$y_k = C_k h_k$$



$\Delta_1 \ldots \Delta_L$ predicted from x

Discretization

$$\begin{aligned} h'(t) &= Ah(t) + B(t)x(t) \\ y(t) &= C(t)h(t) \end{aligned}$$

Euler Discretization

$$h(t + \Delta) \approx h(t) + \Delta h'(t)$$

Discretization

$$\begin{aligned} h'(t) &= Ah(t) + B(t)x(t) \\ y(t) &= C(t)h(t) \end{aligned}$$

Euler Discretization

$$\begin{aligned} h(t+\Delta) &\approx h(t) + \Delta h'(t) \\ &= h(t) + \Delta \left(Ah(t) + Bx(t)\right) \end{aligned}$$

Discretization

$$
\begin{aligned}
h'(t) &= Ah(t) + B(t)x(t) \\
y(t) &= C(t)h(t)
\end{aligned}
$$

Euler Discretization

$$
\begin{aligned}
h(t+\Delta) &\approx h(t) + \Delta h'(t) \\
&= h(t) + \Delta\left(Ah(t) + Bx(t)\right) \\
&= \left(I + \Delta A\right) h(t) + \Delta Bx(t)
\end{aligned}
$$

Discretization

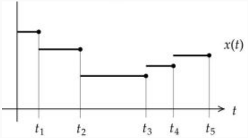
$$h'(t) = Ah(t) + B(t)x(t)$$
$$y(t) = C(t)h(t)$$

Euler Discretization

$$\begin{aligned} h(t+\Delta) &\approx h(t) + \Delta h'(t) \\ &= h(t) + \Delta\left(Ah(t) + Bx(t)\right) \\ &= \left(I + \Delta A\right)h(t) + \Delta Bx(t) \\ &= \bar{A}h(t) + \bar{B}x(t) \end{aligned}$$

Discretization Formula: Zero Order Hold

$$\bar{A}_k = \exp(\Delta_k A)$$
$$\bar{B}_k = \left(\bar{A}_k - 1\right)(B/A)$$



- A is learned weight
- $\bar{B}_k$ and Δ_k are input dependent

Complete Process

- Predict continuous values A, B, and intervals Δ for inputs x_1 to x_L.

Complete Process

- Predict continuous values A, B, and intervals Δ for inputs x_1 to x_L.
- Discretize A and B to $\bar{A}$ and $\bar{B}$.

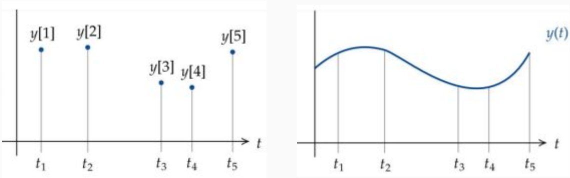
Complete Process

- Predict continuous values A, B, and intervals Δ for inputs x_1 to x_L.
- Discretize A and B to $\bar{A}$ and $\bar{B}$.
- Run parallel scan in discrete time for output values.

Complete Process

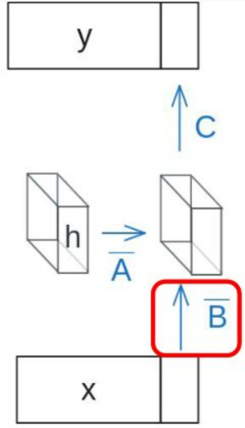
- Predict continuous values A, B, and intervals Δ for inputs x_1 to x_L.
- Discretize A and B to $\bar{A}$ and $\bar{B}$.
- Run parallel scan in discrete time for output values.

Output values approximate the continuous model at sample points of x.



Recall Failure Case 1: Filtering

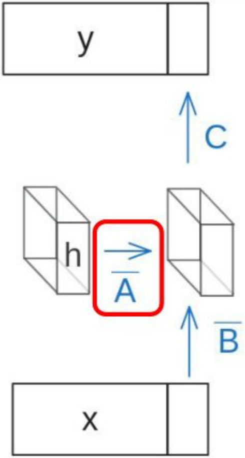
$$h_k = (\bar{A}h_{k-1} + \bar{B}x_k)$$
$$y_k = Ch_k$$



LTI cannot ignore tokens!
Example: Junk text on the web
(copyright, ad copy)

Recall Failure Case 2: Reset

$$h_k = (\bar{A} h_{k-1} + \bar{B} x_k)$$
$$y_k = C h_k$$



LTI cannot reset history!
Example: Start of a new article, chapter in a long document

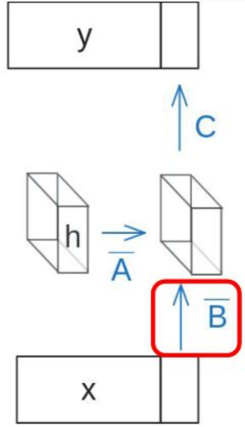
Fixed Case 1: Filtering

$\Delta_k \to 0$

$$\bar{A}_k = \exp(\Delta_k A) \to 1$$
$$\bar{B}_k = \left(\bar{A}_k - 1\right)(B/A) \to 0$$

$$h_k = \bar{A}_k h_{k-1} + \bar{B}_k x_k$$

Delta can filter tokens



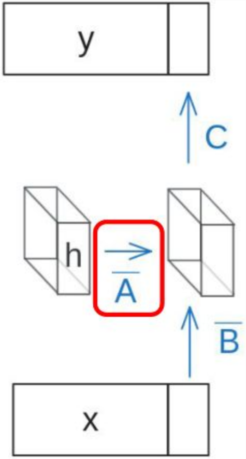
Fixed Case 2: Reset

$\Delta_k \to \infty$

$$\bar{A}_k = \exp(\Delta_k A) \to 0$$

$$h_k = \bar{A}_k h_{k-1} + \bar{B}_k x_k$$

Delta can reset state

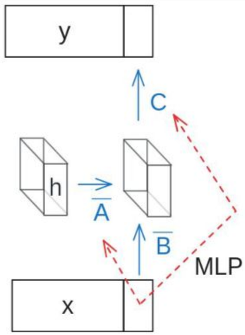


Results: Parameterization for LTV

✓Fixes central issues with LTI
✓Maintains fixed-sized state
✓Able to learn A, B, C

But ....

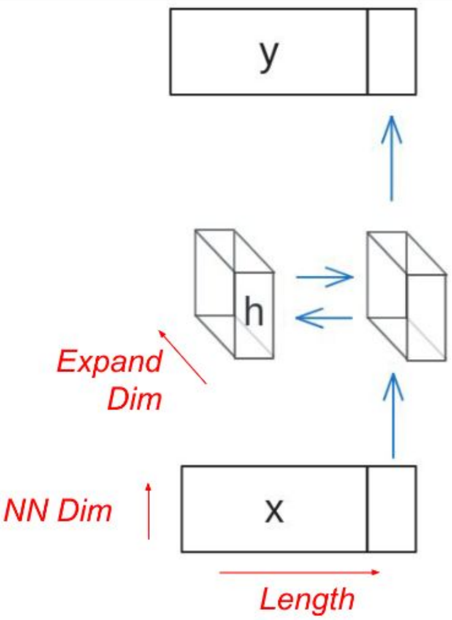
This seems inherently slower than LTI?

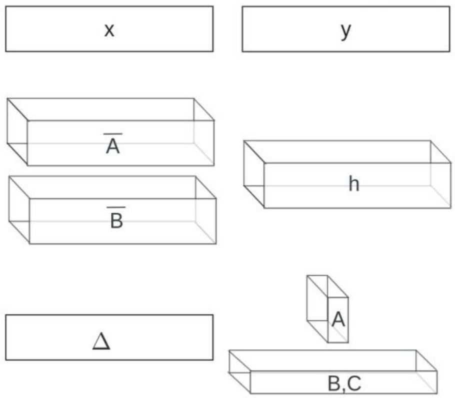


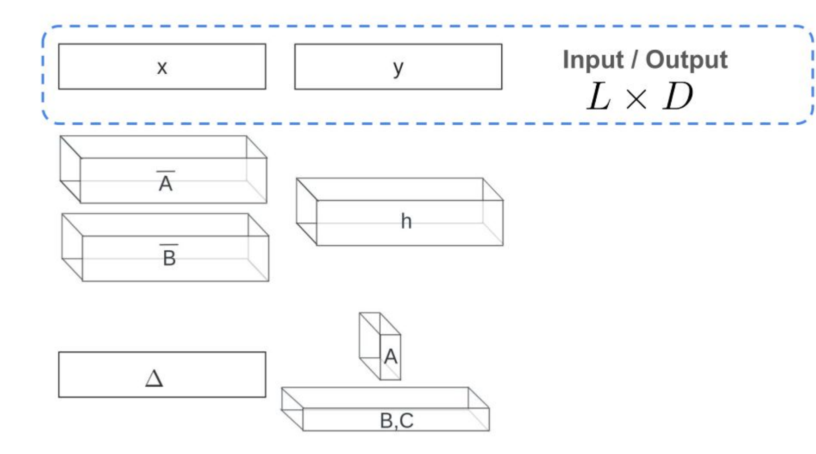
Questions?

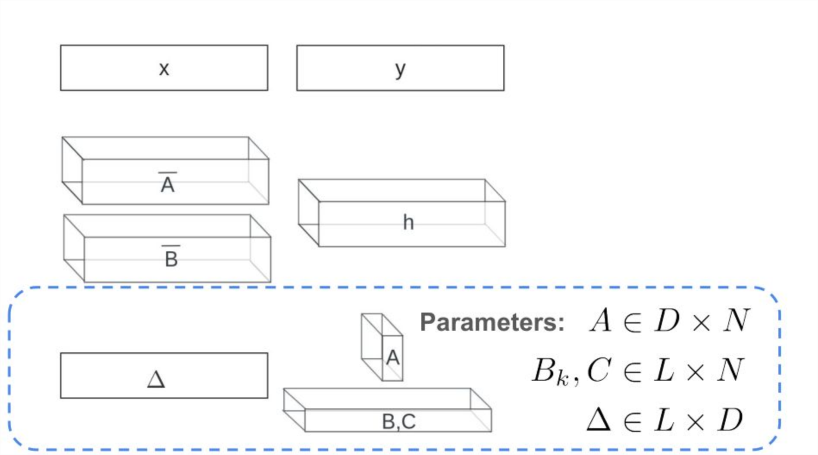
How do I scale it to its max state?

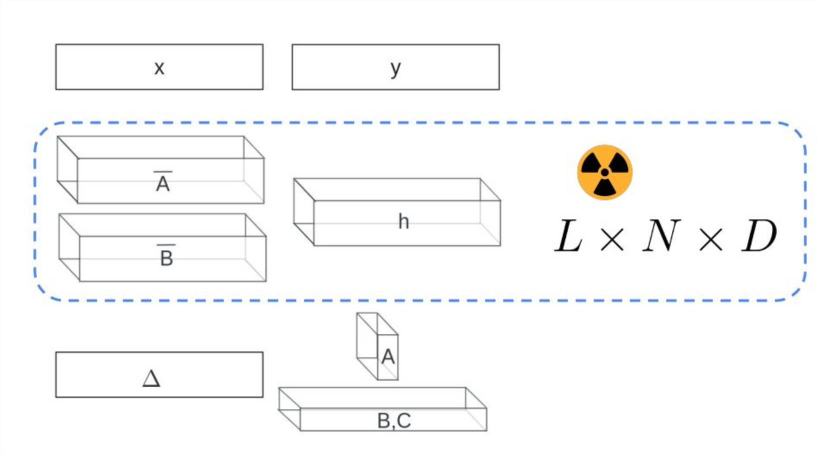
Recall Dimensions



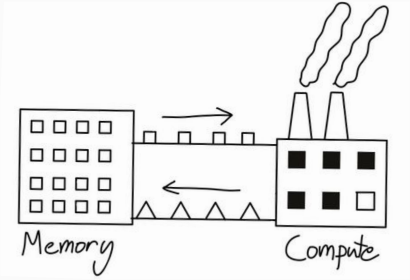

x
y
$\overline{A}$
$\overline{B}$
h
Δ
A
B,C


x
y
Input / Output
$L \times D$
$\overline{A}$
$\overline{B}$
h
Δ
A
B,C


x
y
$\bar{A}$
$\bar{B}$
h
Δ
A
B,C
Parameters: $A \in D \times N$
$B_k, C \in L \times N$
$\Delta \in L \times D$

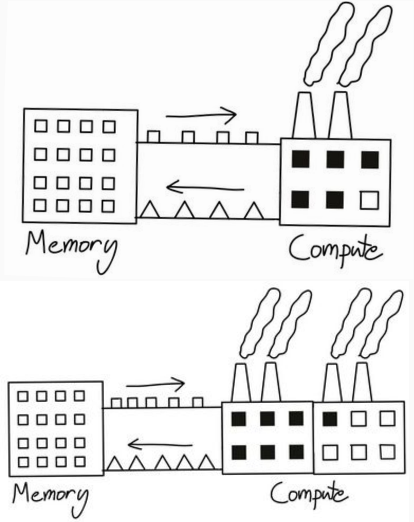

x
y
$\overline{A}$
$\overline{B}$
h
$L \times N \times D$
Δ
A
B,C

Memory bound vs Compute bound



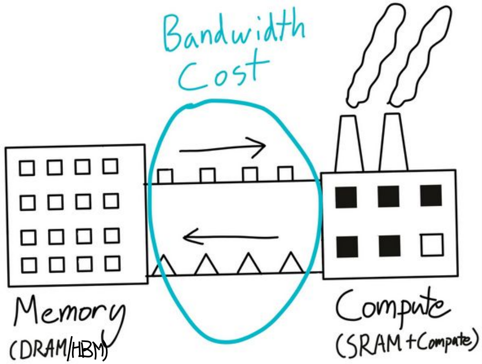
Making Deep Learning Go Brrrr From First Principles (Horace He 2022)

Memory bound vs Compute bound



Making Deep Learning Go Brrrr From First Principles (Horace He 2022)

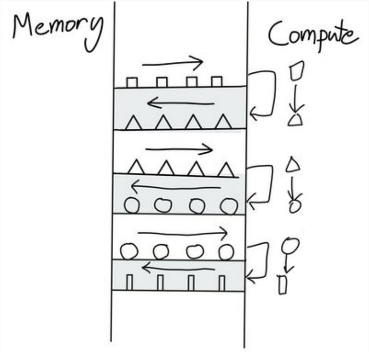
Memory-bandwidth Cost



Making Deep Learning Go Brrrr From First Principles (Horace He 2022)

Operator Fusion - Motivation

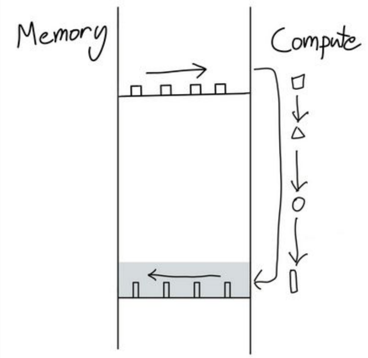
```
x1 = x.cos()
x2 = x1.cos()
x3 = x2.sin()
```



Making Deep Learning Go Brrrr From First Principles (Horace He 2022)

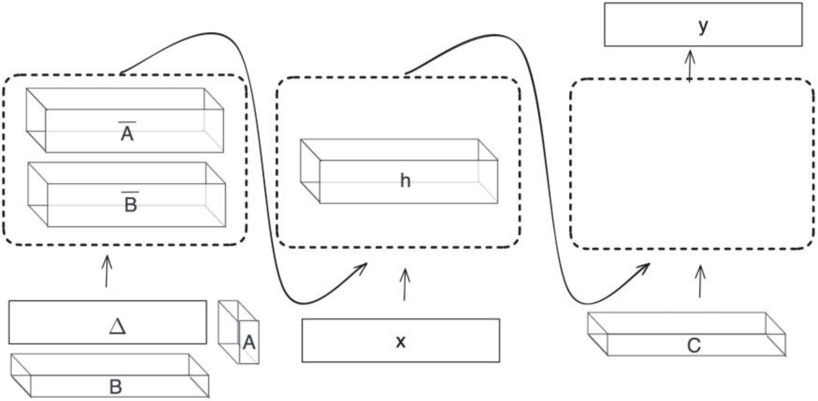
Operator Fusion

```
x1 = x.cos().cos().sin()
```

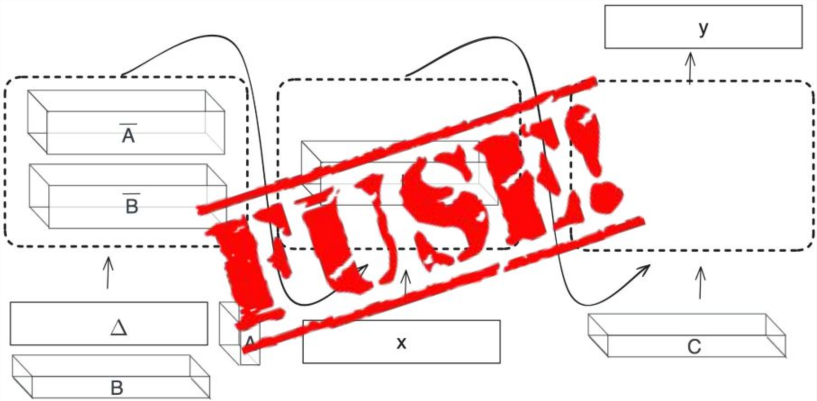


Making Deep Learning Go Brrrr From First Principles (Horace He 2022)

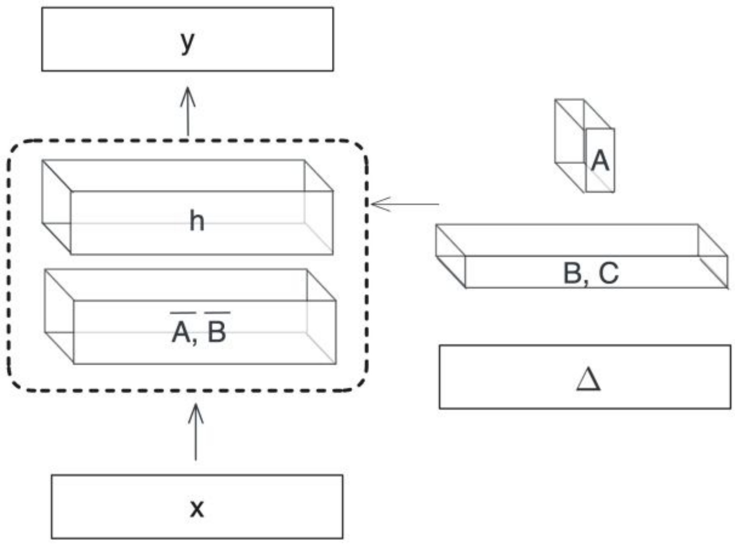
“Naive” Computation



Operator Fusion

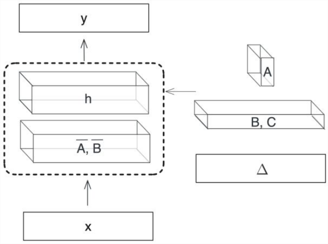


Operator Fusion

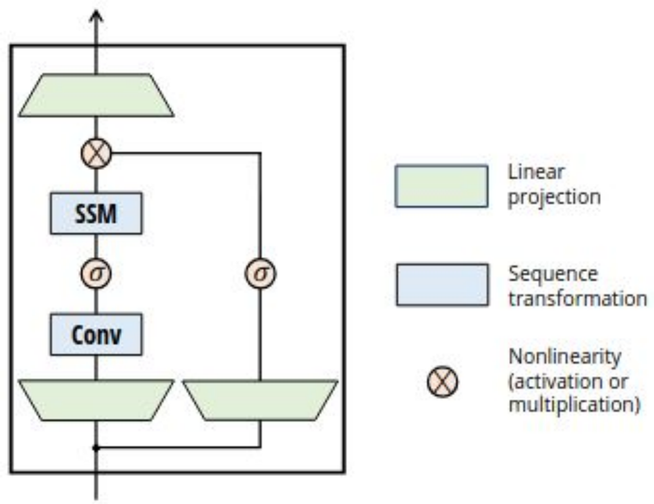


Recomputation

- Fusion requires recomputation for backpropagation.
- Reduces memory costs and speeds up by avoiding HBM reads of $h, \bar{A}, \bar{B}$.

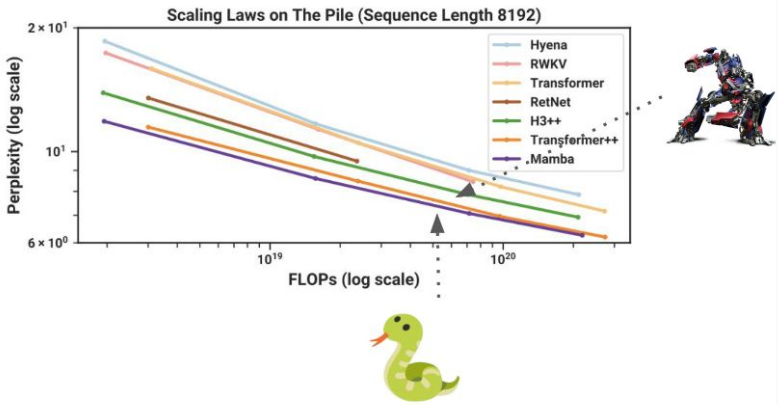


Mamba Architecture



Conclusion

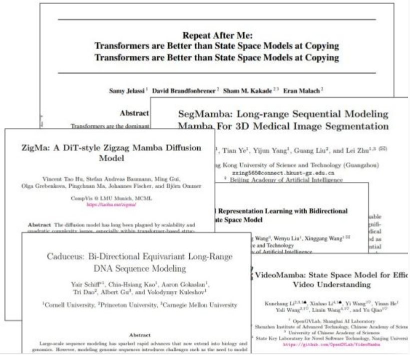
Does it work?



Mamba (Gu and Dao 2023)

What next?

- Lots of interest in different applications.
- Images, video, graphs, interpretability



Questions?

Thank You!