

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

Attention Is All You Need			
Ashish Vaswani ¹ Google Brain avaswani@google.com	Vasishth Shrivastava ² Google Brain vshrivastava@google.com	Niki Parmar ¹ Google Research nikipar@google.com	Jakob Uszkoreit ¹ Google Research uszkoreit@google.com
Llion Jones ¹ Google Research llionj@google.com	Akshay Venkatesh ¹ University of Toronto akshay@cs.toronto.edu	Adam P. Gouws ¹ Google Brain adampgouws@google.com	Illia Polosukhin ¹ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. The model achieves the BLEU score of the WMT 2014 English-to-French translation task, surpassing the existing best results. On the WMT 2014 English-to-German translation task, the model achieves a new state-of-the-art BLEU score of 40.3 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer performs well on other tasks by applying it successfully to English-to-German machine translation both with long and short training data.

¹Equal contribution. Listing order is arbitrary. Initial preprint replacing [Vaswani et al. \(2017\)](#), the other to replace the site. Author, with this, described and implemented the Transformer model and the first results described in this paper. The other authors contributed to the model architecture, implementation, and the parallel for the previous presentation and for the other parts described in this paper. All authors contributed to the model architecture, implementation, and the parallel for the previous presentation and for the other parts described in this paper. All authors contributed to the model architecture, implementation, and the parallel for the previous presentation and for the other parts described in this paper.

²Work performed while at Google Brain.

1st Conference on Neural Information Processing Systems (NIPS), Long Beach, CA, USA.

1 Introduction

Recent neural networks, long short-term networks (LSTMs) and gated recurrent units (GRUs), have been widely considered as one of the approaches to sequence modeling and transduction problems such as language modeling and machine translation (Sutskever et al., 2014). These models have been shown to be effective in a wide range of tasks.

Recent models typically factor computation along the encoded position of the input and output sequences. Modeling the position is important here, since the position of the input and output sequences is, in a sense, a function of the previous hidden state h_{t-1} , and the input for position t . This inherently sequential nature provides parallelization while training, which becomes critical at longer sequence lengths, as recurrent computation (long backtracking across examples). Recent work has advanced sequence-to-sequence models by introducing attention mechanisms (Vaswani et al., 2017) and self-attention mechanisms (Vaswani et al., 2017), which also improve model performance in cases of the latter. The fundamental constraint of sequential computation, however, remains.

Attention mechanisms have become an integral part of computing sequence modeling and transduction results in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences (Vaswani et al., 2017). In fact, in the long run, attention mechanisms are used in conjunction with a recurrent network.

In this work we propose the Transformer, a model architecture exclusively recurrent and neural network entirely, an attention mechanism to share global dependencies between inputs and outputs. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for a while on machine translation on eight GPUs.

2 Background

The goal of training sequential computation often forms the foundation of the Embedded Neural GPU (EN-GPU) and Conv-GPU (CG), all of which are convolutional neural networks in form, building on top of the recurrent hidden representations as parallel for all input and output positions. In this model, the number of operations required to update inputs from two arbitrary inputs or output positions grows as the distance between positions, linearly for Conv-GPU and quadratically for EN-GPU. This makes it more difficult to learn dependencies between distant positions (Vaswani et al., 2017). In the Transformer this is achieved by a constant number of operations, either at the cost of reduced effective resolution due to averaging attention-weighted positions, or at the cost of increased model size (Vaswani et al., 2017).

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence, in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, neural machine translation and independent sentence representations (Vaswani et al., 2017).

Summarize

The "Attention Is All You Need" paper, published by Vaswani et al. in 2017, introduced the Transformer model, which revolutionized the field of NLP by relying entirely on self-attention mechanisms, removing the need for recurrent or convolutional architectures. The Transformer consists of an encoder-decoder structure where both the encoder and decoder are made up of layers that utilize self-attention to capture dependencies between words regardless of their distance in a sequence. This architecture allowed for more efficient parallelization, making it faster to train on large datasets. The key innovation was the self-attention mechanism, which assigns different weights to each word in a sequence based on its relevance to other words, allowing the model to capture complex linguistic relationships. The Transformer significantly improved performance across various tasks such as machine translation and sequence-to-sequence problems, laying the foundation for models like BERT and GPT.

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:

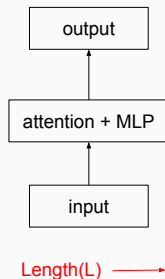
1. **Prepare a double boiler:** Place a heatproof bowl over a pot of simmering water, making sure the bottom of the bowl doesn't touch the water.
2. **Whisk the egg yolks** and lemon juice together in the bowl until thickened and lighter in color (about 1–2 minutes).
3. **Slowly add butter:** Very slowly 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

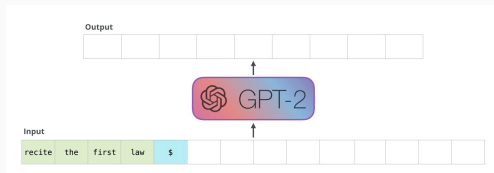


Challenge: Inference Scales Poorly

Challenges:

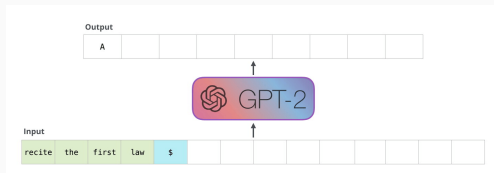
- $O(L)$ memory scaling at inference
- KV cache grows with length

Autoregressive Generation in GPT-2



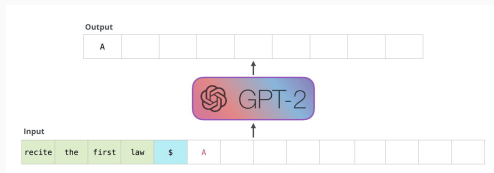
Example from <https://jalammr.github.io/illustrated-gpt2>

Autoregressive Generation in GPT-2



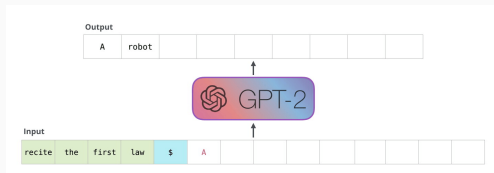
Example from <https://jalammar.github.io/illustrated-gpt2>

Autoregressive Generation in GPT-2



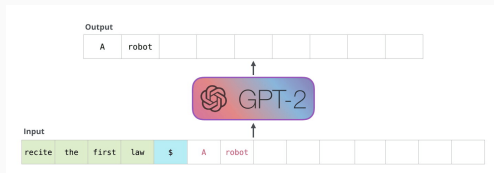
Example from <https://jalammarm.github.io/illustrated-gpt2>

Autoregressive Generation in GPT-2



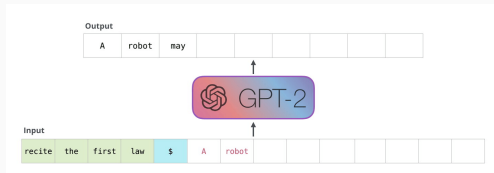
Example from <https://jalammarr.github.io/illustrated-gpt2>

Autoregressive Generation in GPT-2



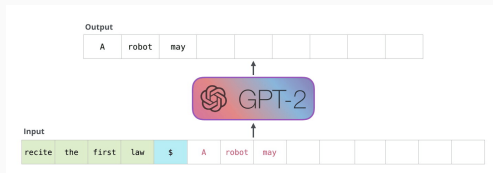
Example from <https://jalammarm.github.io/illustrated-gpt2>

Autoregressive Generation in GPT-2



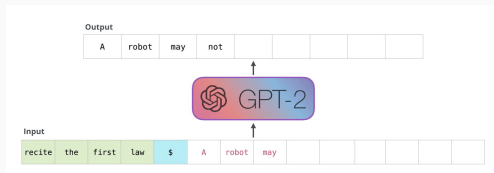
Example from <https://jalammarm.github.io/illustrated-gpt2>

Autoregressive Generation in GPT-2



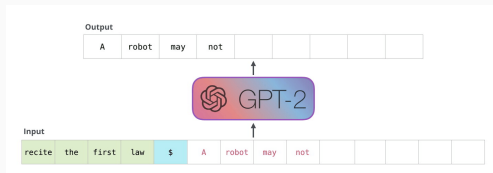
Example from <https://jalammarm.github.io/illustrated-gpt2>

Autoregressive Generation in GPT-2



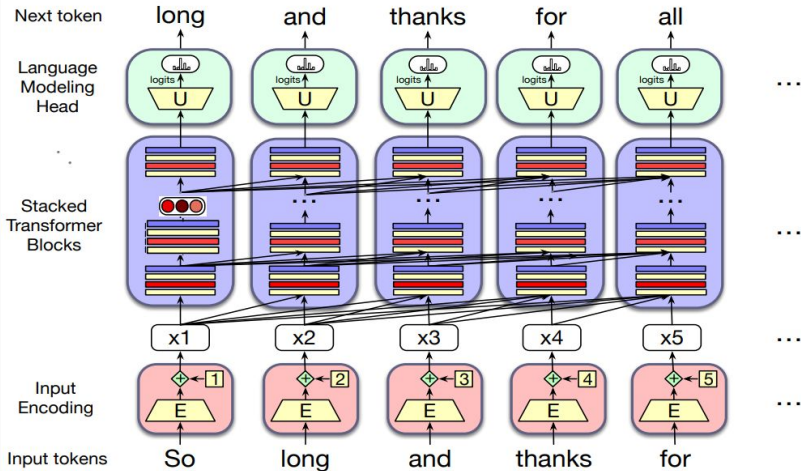
Example from <https://jalammarr.github.io/illustrated-gpt2>

Autoregressive Generation in GPT-2



Example from <https://jalammarm.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

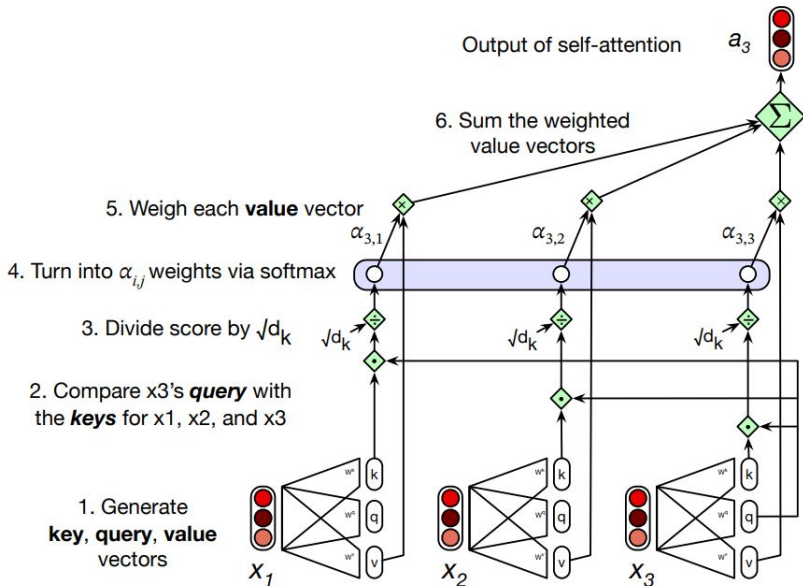
$$q_i = x_i W^Q, \quad k_j = x_j W^K, \quad v_j = x_j W^V$$

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

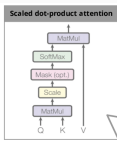
$$\alpha_{i,j} = \text{softmax}(\text{score}(x_i, x_j)) \quad (\forall j \leq i)$$

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

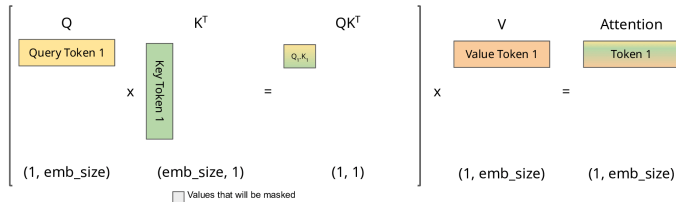
Calculating a_3



Dot-product Attention in Decoder



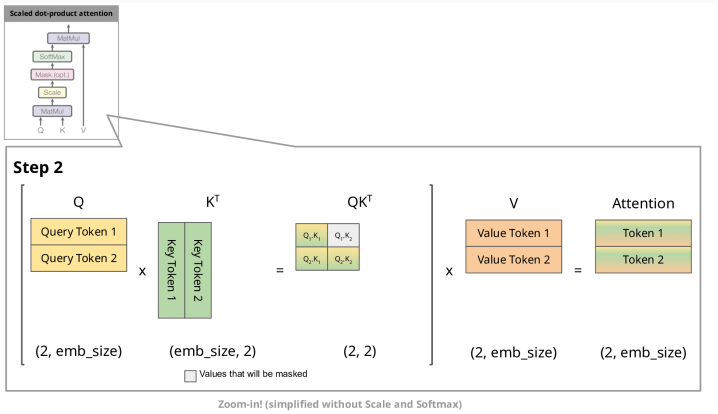
Step 1



Zoom-in! (simplified without Scale and Softmax)

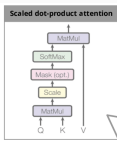
Example from <https://medium.com/@joaolages/kv-caching-explained-276520203249>

Dot-product Attention in Decoder

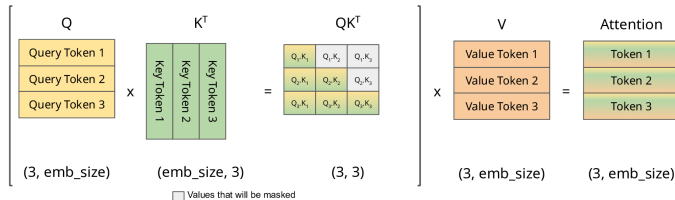


Example from <https://medium.com/@joaolages/kv-caching-explained-276520203249>

Dot-product Attention in Decoder



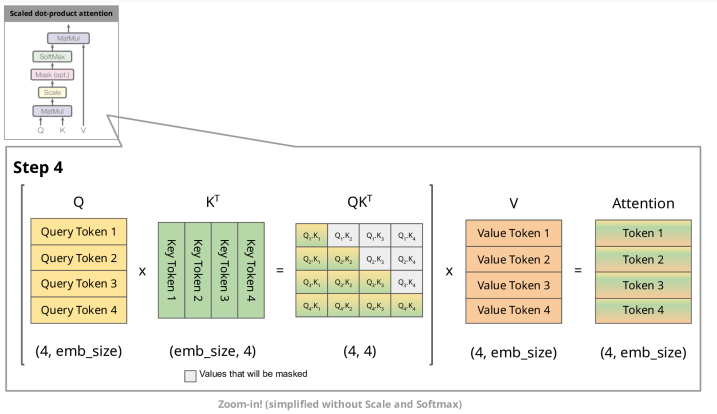
Step 3



Zoom-in! (simplified without Scale and Softmax)

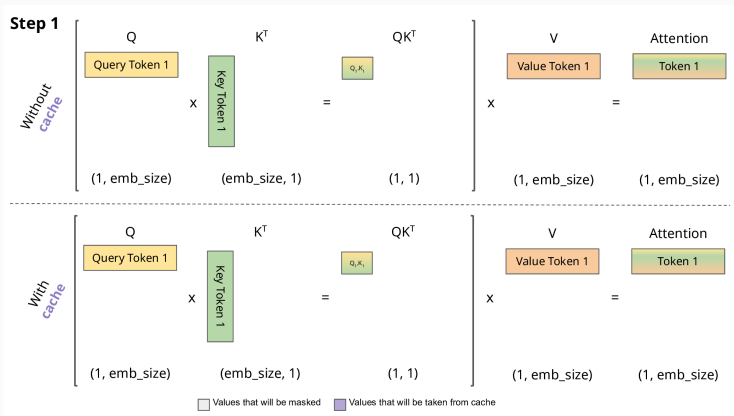
Example from <https://medium.com/@joaolages/kv-caching-explained-276520203249>

Dot-product Attention in Decoder



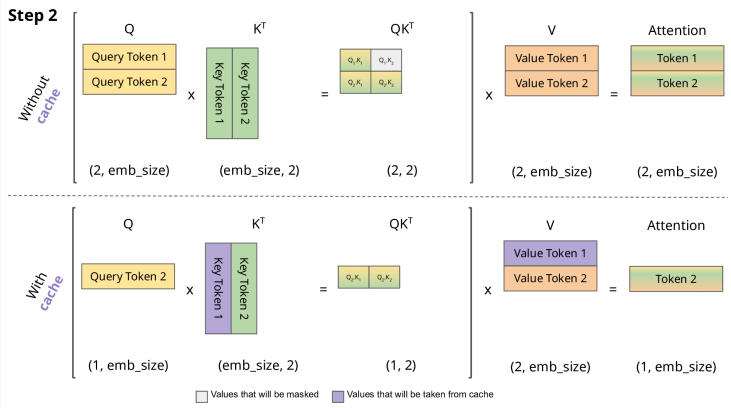
Example from <https://medium.com/@joaolages/kv-caching-explained-276520203249>

Dot-product Attention in Decoder with KV-cache



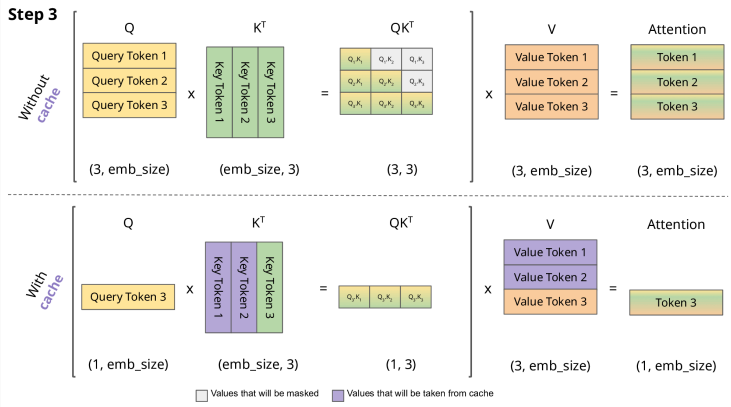
Example from <https://medium.com/@joaolages/kv-caching-explained-276520203249>

Dot-product Attention in Decoder with KV-cache



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

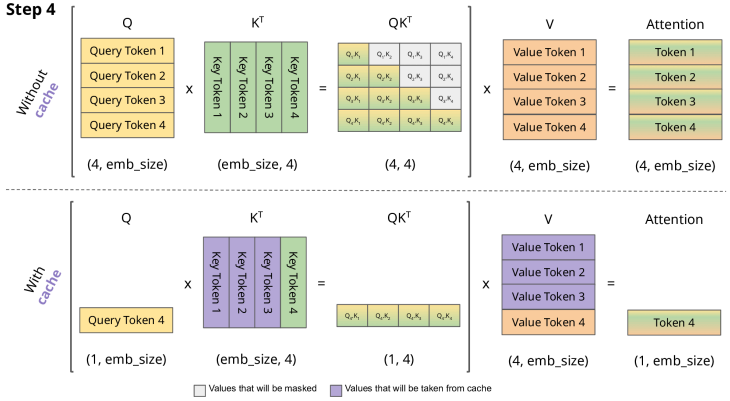
Dot-product Attention in Decoder with KV-cache



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

Dot-product Attention in Decoder with KV-cache

Step 4



Example from <https://medium.com/@joalages/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

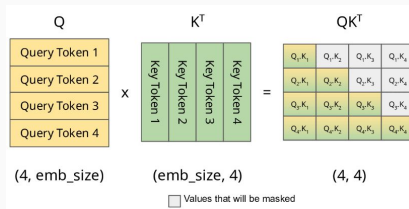
	Available params	Active params	KV cache (256K context, 16bit)
Mistral	7.2B	7.2B	32GB
Mixtral 8x7B	46.7B	12.9B	32GB
LLaMA-3.1 8B	8B	8B	32GB
Mixtral 8x22B	141B	39B	56GB
Mistral-Large-2	123B	123B	88GB
LLaMA-3.1 70B	70B	70B	80GB
LLaMA-3.1 405B	405B	405B	252GB
Jamba-1.5-Mini	52B	12B	4GB
Jamba-1.5-Large	398B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

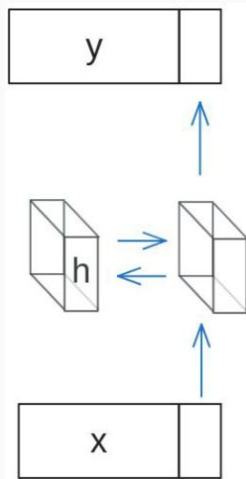


Newer subquadratic architectures targeting large language models

- 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)

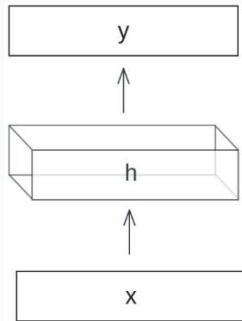
Property: Fixed-Size Memory

Constant memory at inference

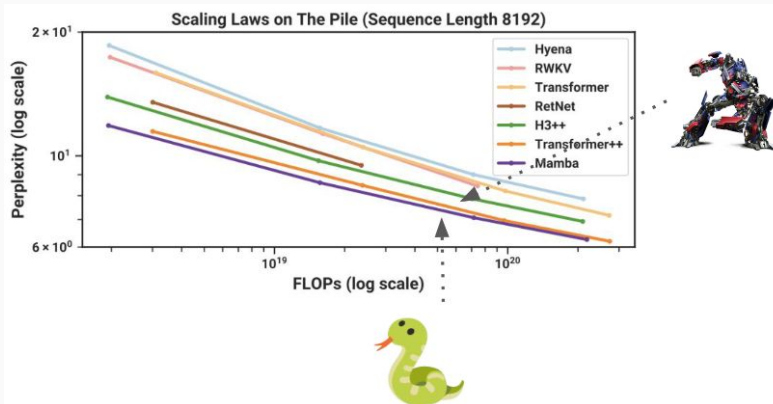


Property: Linear Training Scaling

Linear compute in length



Why is this important now?



Mamba (Gu and Dao 2023)

Questions?

How do I *understand* the model?

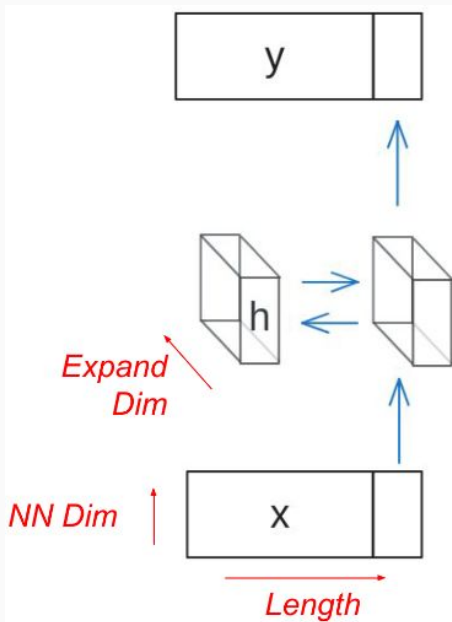
How do I *compute* the model?

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



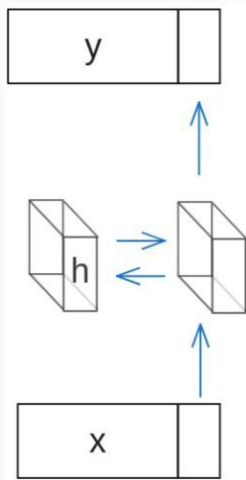
Prelim 1: Vanilla RNN

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

$$y_k = Ch_k$$

Challenges

- Difficult to learn well
- Inefficient to train historically



Prelim 2: Linear Time Invariant (LTI)

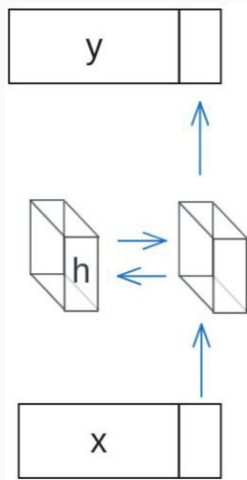
$$h_k = (\bar{A}h_{k-1} + \bar{B}x_k)$$

$$y_k = Ch_k$$

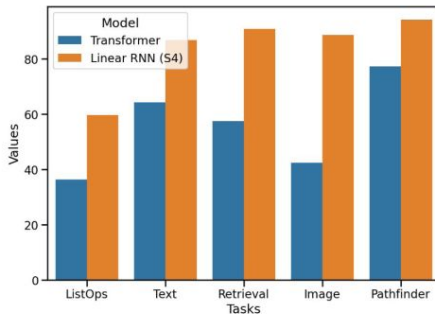
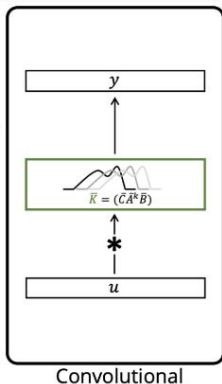
Challenges

- 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

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

Based (Arora et al. 2024)

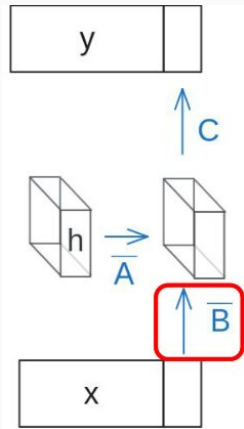
Failure Case 1: Filtering

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

$$y_k = Ch_k$$

LTI cannot ignore tokens!

Example: Junk text on the web
(copyright, ad copy)



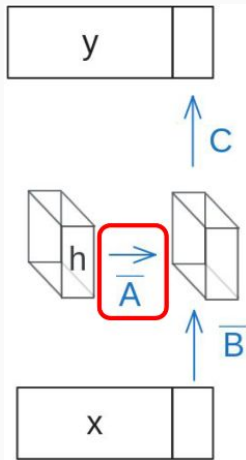
Failure Case 2: Reset

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

$$y_k = Ch_k$$

LTI cannot reset history!

Example: Start of a new article,
chapter in a long document

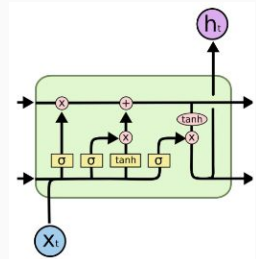


Historical Parallel: RNN \rightarrow LSTM to Allow Gating

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

$$y_k = Ch_k$$

RNN



LSTM

Linear Time Varying (LTV) Model

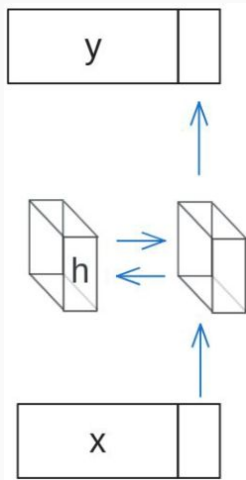
$$h_k = (\bar{A}_{\textcolor{red}{k}} h_{k-1} + \bar{B}_{\textcolor{red}{k}} x_k)$$

$$y_k = C_{\textcolor{red}{k}} h_k$$

Contribution: Let parameters change based on position:

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

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



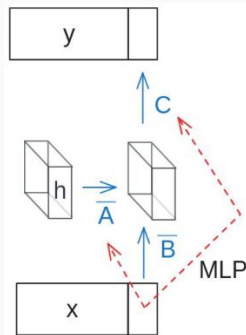
Generating Linear Time Varying (LTV)

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

$$y_k = C_k h_k$$

How to obtain $\bar{A}_k, \bar{B}_k, C_k$:

- Produce as a function of x

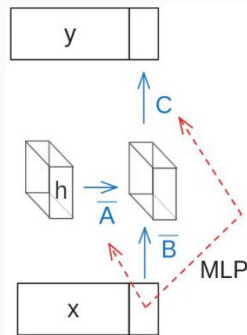


Results: LTV is a Promising Approach For LMs

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

But

How do you run it efficiently?



Questions?

How do I compute the model?

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

$$y_k = Ch_k$$

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

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

$$y_k = Ch_k$$

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

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

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

$$y_k = Ch_k$$

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

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

$$\vdots$$

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

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

$$y_k = Ch_k$$

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

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

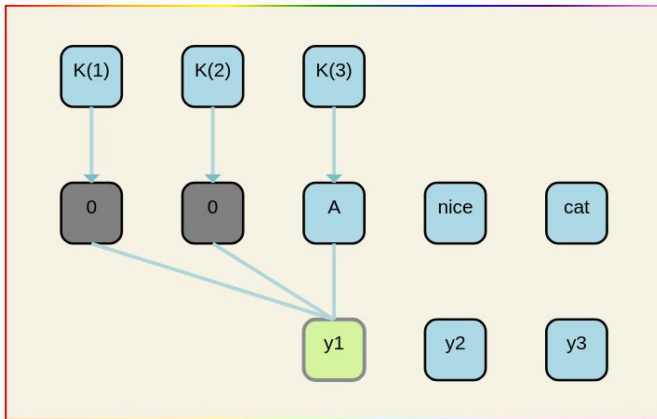
$$\vdots$$

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

$$\bar{y} = \bar{K} * x$$

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

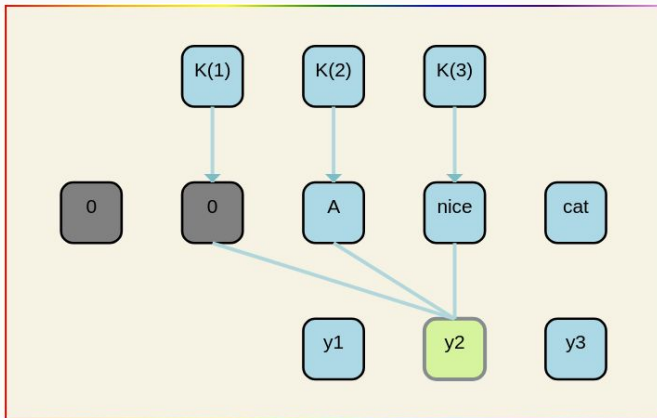
Background: LTI



- Convolution
- Computed in parallel

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

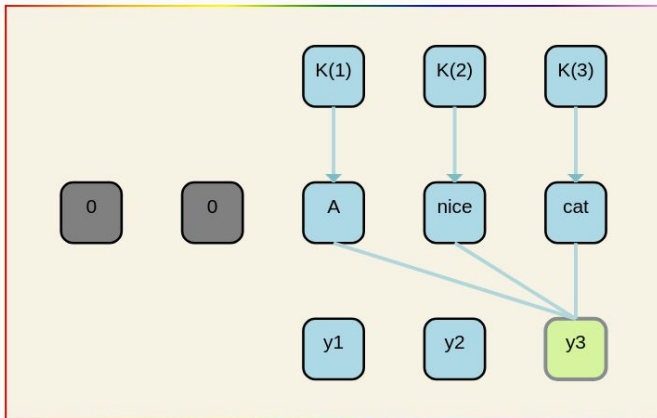
Background: LTI



- 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/

$$h_k = (\bar{A}_{\textcolor{red}{k}} h_{k-1} + \bar{B}_{\textcolor{red}{k}} x_k)$$

$$y_k = C_{\textcolor{red}{k}} h_k$$

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

$$h_k = (\bar{A}_{\textcolor{red}{k}} h_{k-1} + \bar{B}_{\textcolor{red}{k}} x_k)$$

$$y_k = C_{\textcolor{red}{k}} h_k$$

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

- Parallelizes a sequential operation to achieve faster computation.

Associative Scan

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

- 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

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

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

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

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

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

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

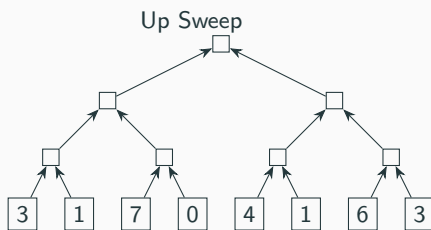
$$\begin{bmatrix} 3 & 1 & 7 & 0 & 4 & 1 & 6 & 3 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \end{bmatrix}$$

$$h_k = h_{k-1} + x_k$$

$$y_k = h_k$$

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

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



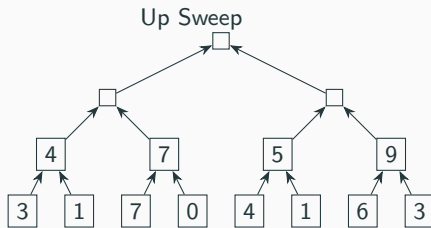
Down Sweep

$$\text{sum}[v] = \text{sum}[L[v]] + \text{sum}[R[v]]$$

$$\text{prescan}[L[v]] = \text{prescan}[v]$$

$$\text{prescan}[R[v]] = \text{sum}[L[v]] + \text{prescan}[v]$$

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



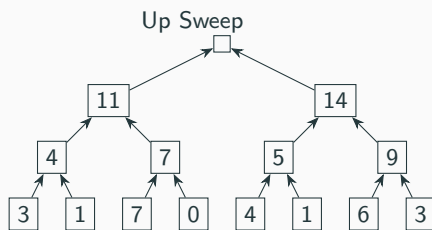
Down Sweep

$$\text{sum}[v] = \text{sum}[L[v]] + \text{sum}[R[v]]$$

$$\text{prescan}[L[v]] = \text{prescan}[v]$$

$$\text{prescan}[R[v]] = \text{sum}[L[v]] + \text{prescan}[v]$$

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



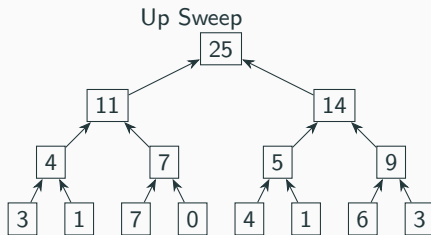
Down Sweep

$$\text{sum}[v] = \text{sum}[L[v]] + \text{sum}[R[v]]$$

$$\text{prescan}[L[v]] = \text{prescan}[v]$$

$$\text{prescan}[R[v]] = \text{sum}[L[v]] + \text{prescan}[v]$$

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



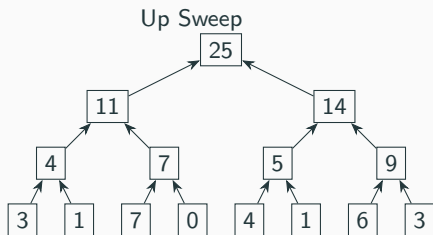
Down Sweep

$$\text{sum}[v] = \text{sum}[L[v]] + \text{sum}[R[v]]$$

$$\text{prescan}[L[v]] = \text{prescan}[v]$$

$$\text{prescan}[R[v]] = \text{sum}[L[v]] + \text{prescan}[v]$$

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



Down Sweep

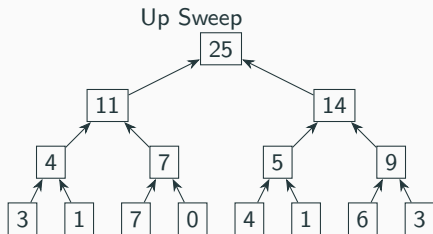
0

$$\text{sum}[v] = \text{sum}[L[v]] + \text{sum}[R[v]]$$

$$\text{prescan}[L[v]] = \text{prescan}[v]$$

$$\text{prescan}[R[v]] = \text{sum}[L[v]] + \text{prescan}[v]$$

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



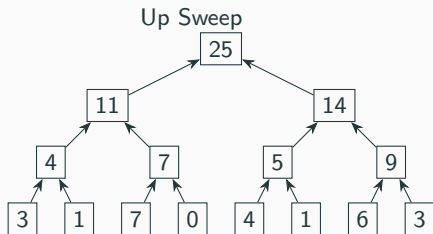
$$\text{sum}[v] = \text{sum}[L[v]] + \text{sum}[R[v]]$$



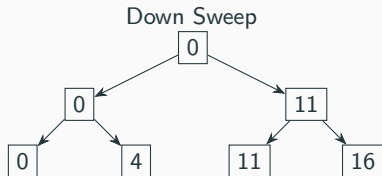
$$\text{prescan}[L[v]] = \text{prescan}[v]$$

$$\text{prescan}[R[v]] = \text{sum}[L[v]] + \text{prescan}[v]$$

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

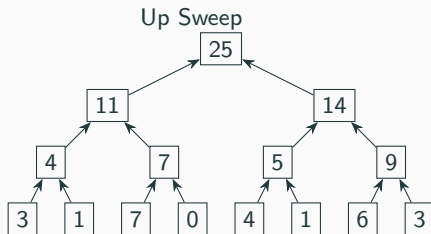


$$\text{sum}[v] = \text{sum}[L[v]] + \text{sum}[R[v]]$$

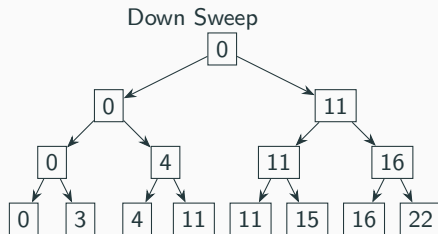


$$\begin{aligned} \text{prescan}[L[v]] &= \text{prescan}[v] \\ \text{prescan}[R[v]] &= \text{sum}[L[v]] + \text{prescan}[v] \end{aligned}$$

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



$$\text{sum}[v] = \text{sum}[L[v]] + \text{sum}[R[v]]$$



$$\text{prescan}[L[v]] = \text{prescan}[v]$$

$$\text{prescan}[R[v]] = \text{sum}[L[v]] + \text{prescan}[v]$$

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

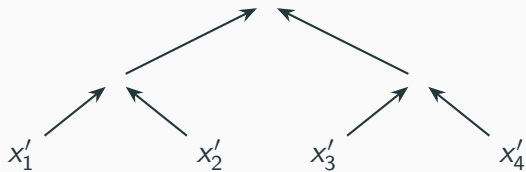
$$y_k = C_k h_k$$

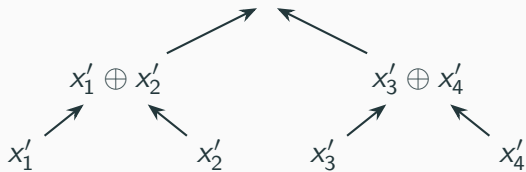
Linear recurrence?

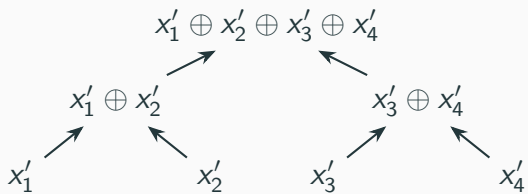
New primitives

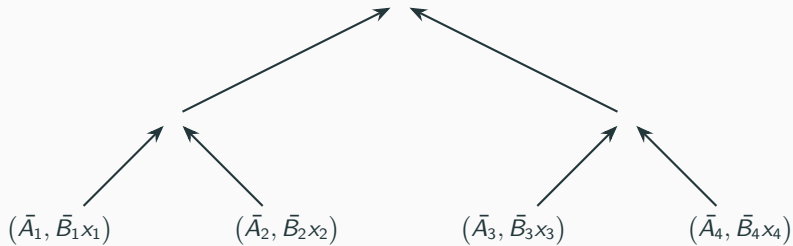
$$x'_k := (\bar{A}_k, \bar{B}_k x_k)$$

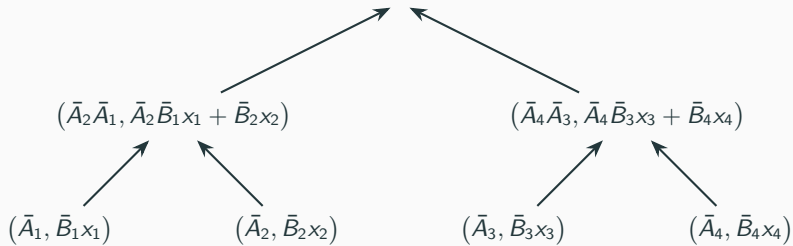
$$(a_1, b_1) \oplus (a_2, b_2) := (a_2 a_1, a_2 b_1 + b_2)$$

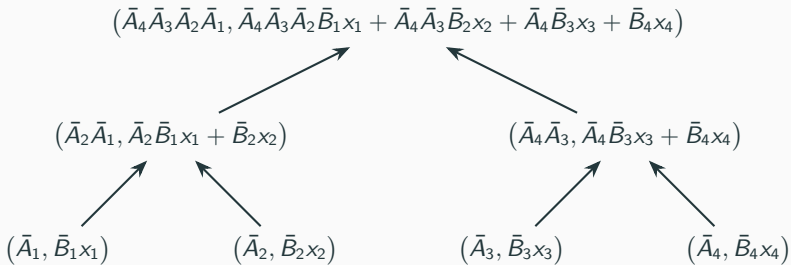




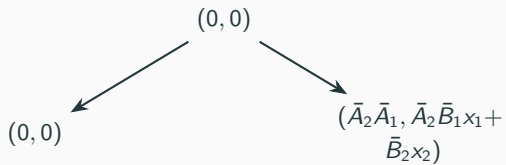


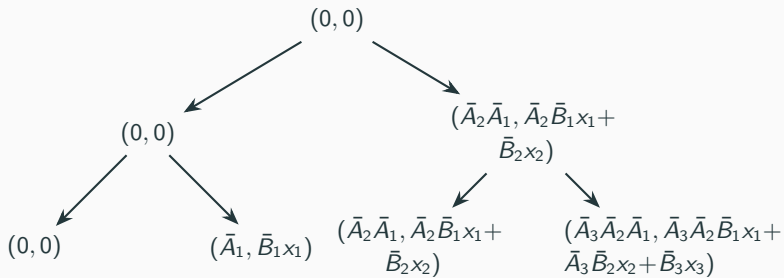






$(0, 0)$





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?

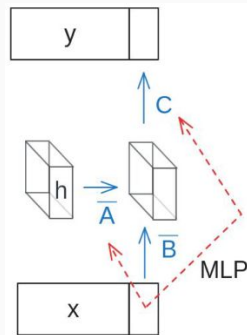
Questions?

How do I design an effective version?

Reminder: LTV

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

$$y_k = C_k h_k$$

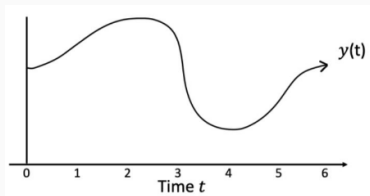


Continuous-Time State-Space Model

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

$$y(t) = C(t)h(t)$$

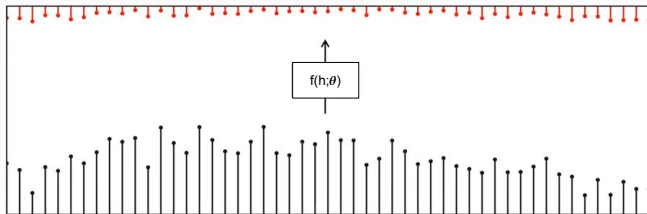
Imagine x , y was in continuous time,
how do we model its dynamics?



A New Approach to Linear Filtering and Prediction
Problems (Kalman 1960)

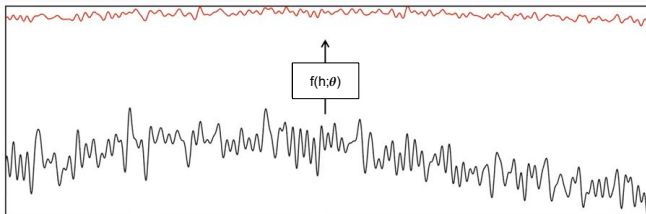
Mamba (Gu & Dao 2023)

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:

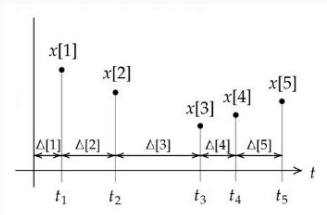
$$h'(t) = Ah(t) + B(t)x(t)$$

$$y(t) = C(t)h(t)$$

Discrete:

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

$$y_k = C_k h_k$$



$\Delta_1 \dots \Delta_L$ predicted from x

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

$$y(t) = C(t)h(t)$$

Euler Discretization

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

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

$$y(t) = C(t)h(t)$$

Euler Discretization

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

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

$$y(t) = C(t)h(t)$$

Euler Discretization

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

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

$$y(t) = C(t)h(t)$$

Euler Discretization

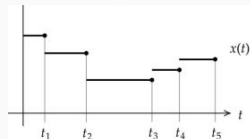
$$\begin{aligned}h(t + \Delta) &\approx h(t) + \Delta h'(t) \\&= h(t) + \Delta (Ah(t) + Bx(t)) \\&= (I + \Delta A) h(t) + \Delta Bx(t) \\&= \bar{A}h(t) + \bar{B}x(t)\end{aligned}$$

Discretization Formula: Zero Order Hold

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

$$\bar{B}_k = (\bar{A}_k - 1) (B/A)$$

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



- 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} .

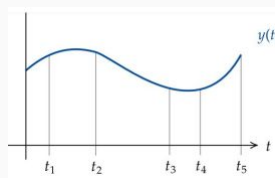
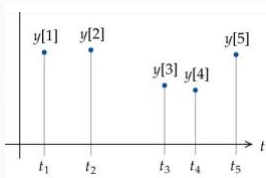
Complete Process

- 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 .



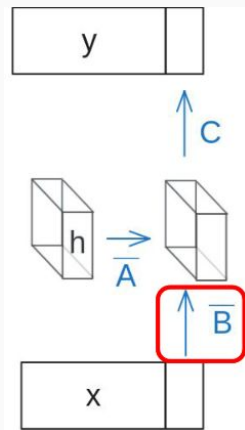
Recall Failure Case 1: Filtering

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

$$y_k = Ch_k$$

LTI cannot ignore tokens!

Example: Junk text on the web
(copyright, ad copy)



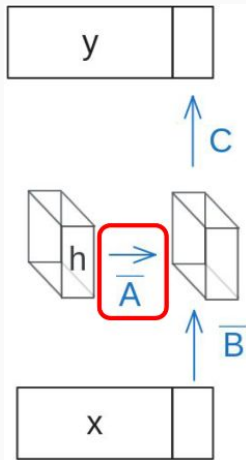
Recall Failure Case 2: Reset

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

$$y_k = Ch_k$$

LTI cannot reset history!

Example: Start of a new article,
chapter in a long document



Fixed Case 1: Filtering

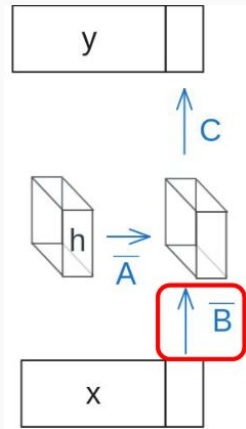
$$\Delta_k \rightarrow 0$$

$$\bar{A}_k = \exp(\Delta_k A) \rightarrow 1$$

$$\bar{B}_k = (\bar{A}_k - 1) (B/A) \rightarrow 0$$

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

Delta can filter tokens



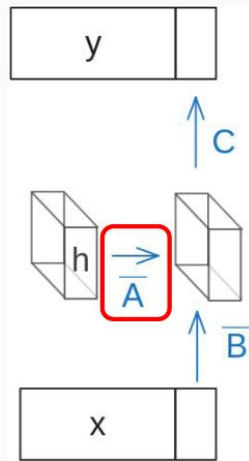
Fixed Case 2: Reset

$$\Delta_k \rightarrow \infty$$

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

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

Delta can reset state

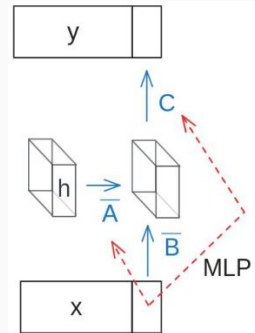


Results: Parameterization for LTV

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

But

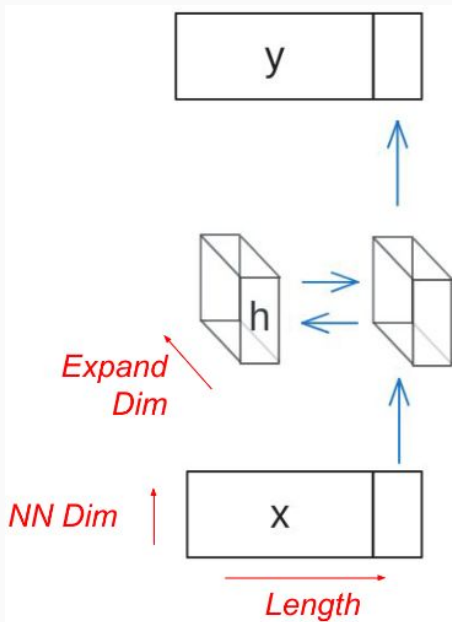
This seems inherently slower than LTI?

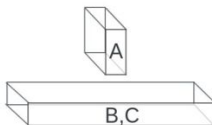
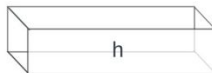
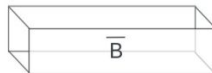
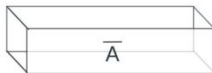
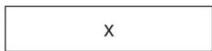


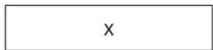
Questions?

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

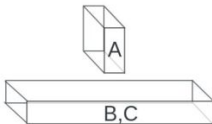
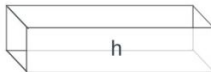
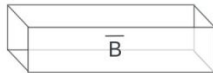
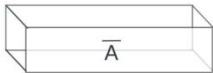
Recall Dimensions

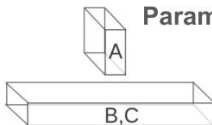
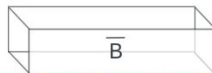
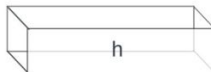
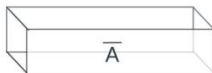
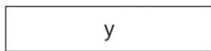
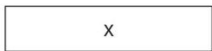






Input / Output
 $L \times D$

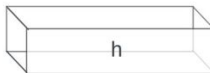
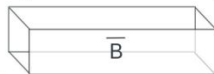
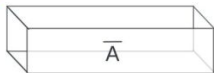




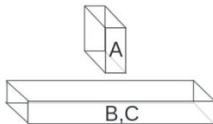
Parameters: $A \in D \times N$

$B_k, C \in L \times N$

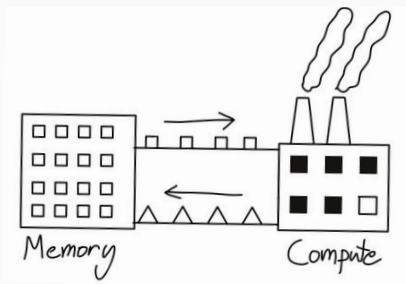
$\Delta \in L \times D$



$$L \times N \times D$$

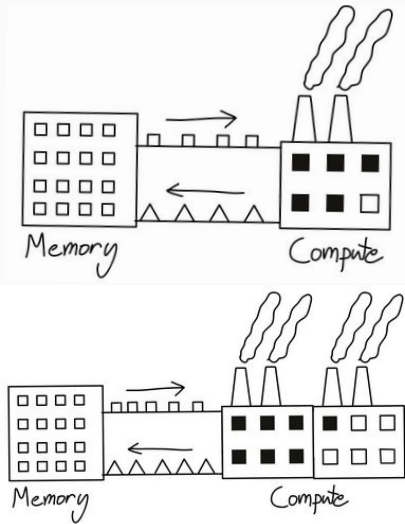


Memory bound vs Compute bound



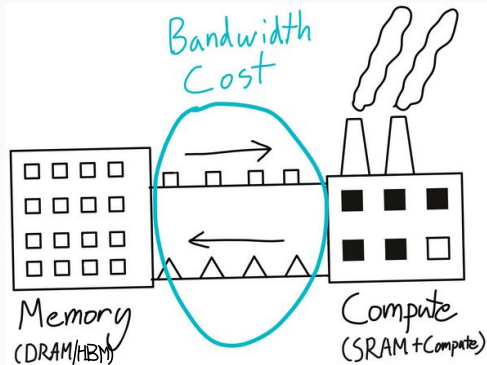
Making Deep Learning Go Brrrr From First Principles (Horace He 2022)

Memory bound vs Compute bound



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

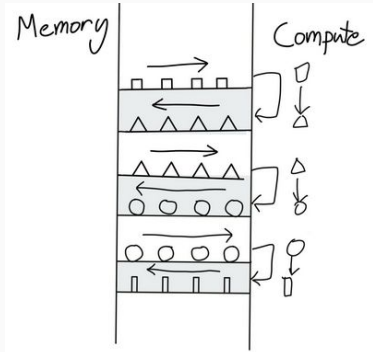
Memory-bandwidth Cost



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

Operator Fusion - Motivation

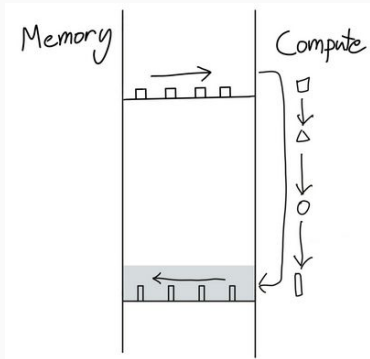
```
x1 = x.cos()  
x2 = x1.cos()  
x3 = x2.sin()
```



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

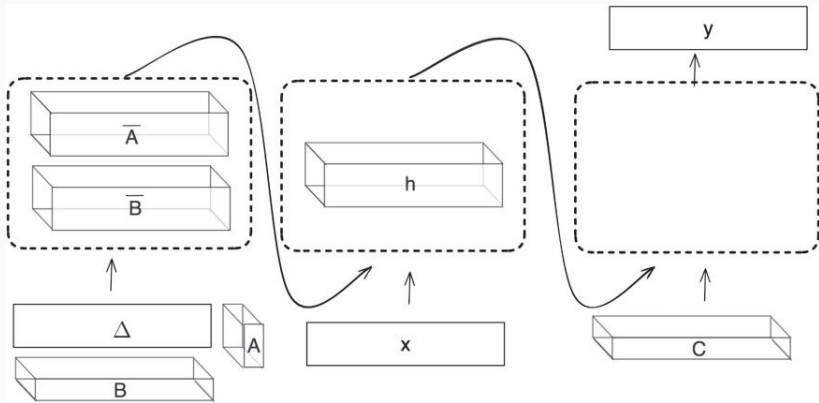
Operator Fusion

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

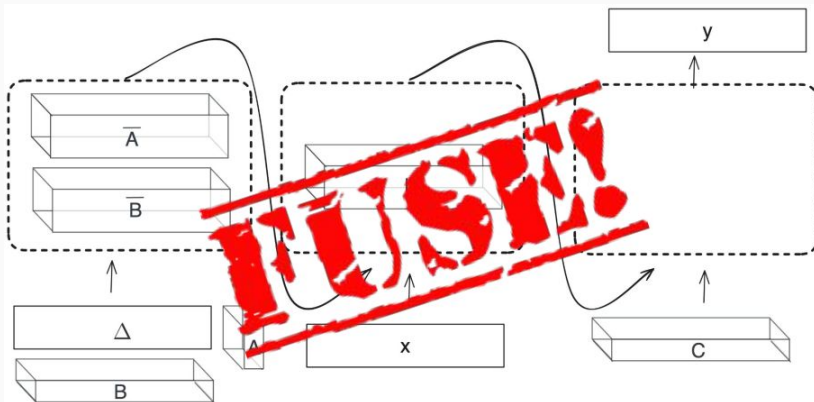


Making Deep Learning Go Brrrr From First Principles (Horace He 2022)

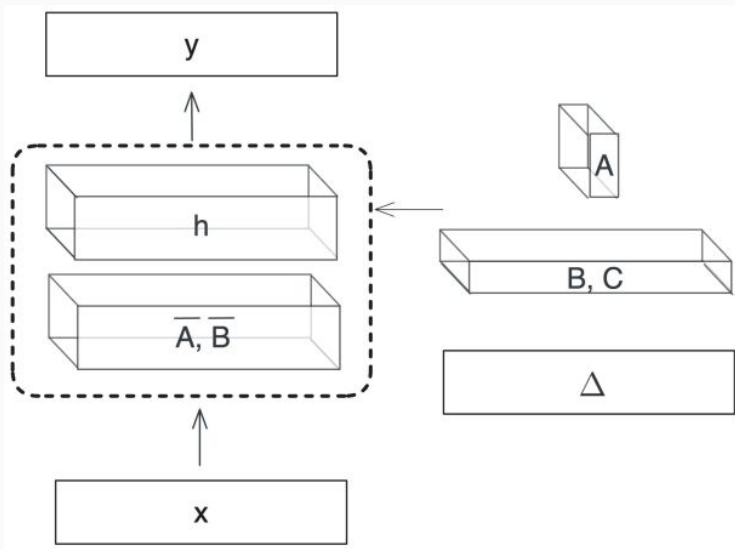
“Naive” Computation



Operator Fusion

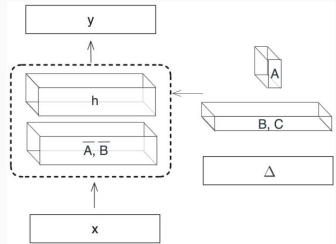


Operator Fusion

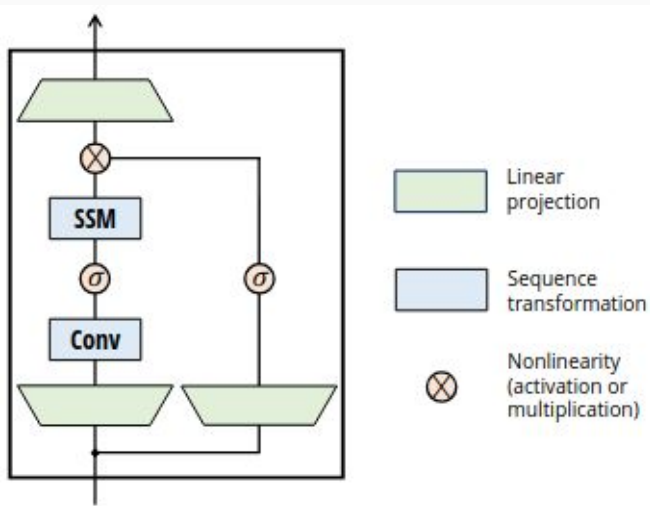


Recomputation

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

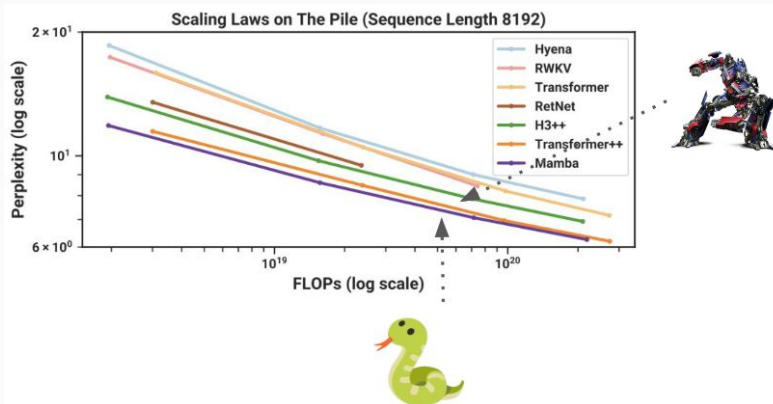


Mamba Architecture



Conclusion

Does it work?



Mamba (Gu and Dao 2023)

What next?

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



Questions?

Thank You!