Mamba

Nasy

May 03, 2024

Table of Contents

Introduction

Motivation:

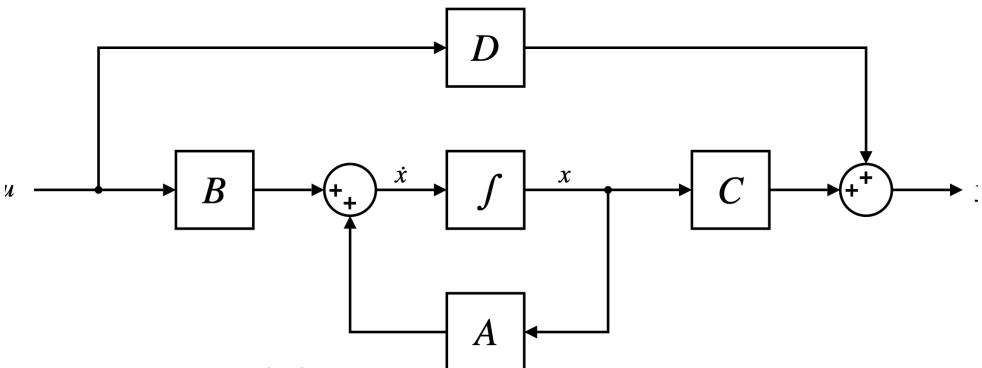
- Most current works ared based on Transformer architecture.
- However, transformers is inefficiency on long sequences.
- Current efficiency works not perform as well as transformers.
- Thus, they propose a new model, mamba, to achieve the modeling power of Transformers while scaling linearly in sequence length.
  - **New Selection Mechanism, Hardware-aware Algorithm, Simpler Architecture** on the prior works.

SSM

Content

- What is SSM?
- Why SSM?
- Motivation
- SSM to NN Layer – s4 Model
- SSM to NN Architectures – H3 Layer

What Is State Space Model (SSM)?



$$x'(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx'(t) + Du(t) \quad (1)$$

- $u(t)$: Input
- $y(t)$: Ouput
- $x(t)$: State
- $x(t)' = \frac{d}{dt}x(t))$: The differential of the state

- A: State matrix
- B: Input matrix
- C: Output matrix
- D: Feedforward matrix

Why SSM?

Deep sequence models meet three challenges (at the time paper was proposed):

- **Generalization**
  - RNN, CNN, Transformer, Neural Differential Equations (NDEs) in different field
  - At least, at the time SSM were proposed
- **Computational Efficiency**
  - CNNs and Transformers are not efficient autoregressive inference
- **Long range dependencies**
  - Gradient vanishing/exploding problem
  - Limitation of the length of the context.

Motivation

A simple and base model to archive the above challenges is the State Space Model (SSM).

- SSM is continuous (differential equations)
- SSM is recurrent (after discretization, if time invariant)
- SSM is convolutional (time invariant)

SSM in Deep Learning

In a continuous system, how to map a function $x(t) \in \mathbb{R}$ to anoher function $y(t) \in \mathbb{R}$ through an implicit latent state $h(t) \in \mathbb{R}^N$?

- **Input**: $x(t) \in \mathbb{R}^{1\times D}$
- **Output**: $y(t) \in \mathbb{R}^{1\times D}$
- **Latent State**: $h(t) \in \mathbb{R}^{N\times D}$
- **Derivative of** $h(t)$: $h'(t)$
- $\boldsymbol{\Delta}$: Time step. for discretization

$$
\begin{aligned}
h'(t) &= \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \\
y(t) &= \boldsymbol{C}h'(t) + \boldsymbol{D}x(t)
\end{aligned}
\tag{2}
$$

where, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are the learnable parameters.

S4 Model – Structure State Space Sequence Model

$$\text{SSM} \xrightarrow{\text{discrete}} \begin{cases} \text{Recurrent representation} \\ \text{Convolutional representation} \end{cases}$$

Discretization? Recurrent? Convolutional?

$$
\begin{aligned}
h'(t) &= \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \\
y(t) &= \boldsymbol{C}h'(t) + \boldsymbol{D}x(t)
\end{aligned} \tag{3}
$$

Here, $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ are continuous parameters.

We can discretize them to discrete parameters $\bar{\boldsymbol{A}}$, $\bar{\boldsymbol{B}}, \bar{\boldsymbol{C}}, \bar{\boldsymbol{D}}$ by the step parameter Δ:

$$
\begin{aligned}
h_t &= \bar{\boldsymbol{A}}h_{t-1} + \bar{\boldsymbol{B}}x_t \\
y_t &= \bar{\boldsymbol{C}}h_t + \bar{\boldsymbol{D}}x_t
\end{aligned} \tag{4}
$$

where $\bar{\boldsymbol{A}} = \exp(\boldsymbol{\Delta A})$,
$\bar{\boldsymbol{B}} = (\boldsymbol{\Delta A})^{-1}(\exp(\boldsymbol{\Delta A} - \boldsymbol{I}) \cdot \boldsymbol{\Delta B})$,
$\bar{\boldsymbol{C}} = \boldsymbol{C}, \bar{\boldsymbol{D}} = \boldsymbol{D}$.

Calculation

$$
\begin{aligned}
h_0 &= \bar{\boldsymbol{B}}x_0;\ h_1 = \bar{\boldsymbol{A}}\bar{\boldsymbol{B}}x_0 + \bar{\boldsymbol{B}}x_1; \\
h_2 &= \bar{\boldsymbol{B}}x_2 = \bar{\boldsymbol{A}}^2\bar{\boldsymbol{B}}x_0 + \bar{\boldsymbol{A}}\bar{\boldsymbol{B}}x_1 + \bar{\boldsymbol{B}}x_2 \\
&\cdots \\
y_0 &= \bar{\boldsymbol{C}}\bar{\boldsymbol{B}}x_0 + \bar{\boldsymbol{D}}x_0 \\
y_1 &= \bar{\boldsymbol{C}}\bar{\boldsymbol{A}}\bar{\boldsymbol{B}}x_0 + \bar{\boldsymbol{C}}\bar{\boldsymbol{B}}x_1 + \bar{\boldsymbol{D}}x_1 \\
y_2 &= \bar{\boldsymbol{C}}\bar{\boldsymbol{A}}^2\bar{\boldsymbol{B}}x_0 + \bar{\boldsymbol{C}}\bar{\boldsymbol{A}}\bar{\boldsymbol{B}}x_1 + \bar{\boldsymbol{C}}\bar{\boldsymbol{B}}x_2 \\
&\quad + \bar{\boldsymbol{D}}x_2 \\
&\cdots \\
\bar{\boldsymbol{K}} &= \left(\bar{\boldsymbol{C}}\bar{\boldsymbol{A}}^i\bar{\boldsymbol{B}}\right)_{i\in[L]} \\
&= \left(\bar{\boldsymbol{C}}\bar{\boldsymbol{B}}, \bar{\boldsymbol{C}}\bar{\boldsymbol{A}}\bar{\boldsymbol{B}}, ..., \bar{\boldsymbol{C}}\bar{\boldsymbol{A}}^{L-1}\bar{\boldsymbol{B}}\right) \\
y &= \bar{\boldsymbol{K}} * x
\end{aligned}
$$

Structure and Dimensions

$$
\begin{aligned}
h'(t) &= \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \\
y(t) &= \boldsymbol{C}h'(t) + \boldsymbol{D}x(t)
\end{aligned}
\tag{5}
$$

Input: $x(t) \in \mathbb{R}^{1\times D}$
Output: $y(t) \in \mathbb{R}^{1\times D}$
Latent State: $h(t) \in \mathbb{R}^{N\times D}$

where, N is the number of dimensions in the latent state, can be any numbers you want.
D is the length of the input and output.

Structure and Dimensions

$$
\begin{aligned}
h'(t) &= \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \\
y(t) &= \boldsymbol{C}h'(t) + \boldsymbol{D}x(t)
\end{aligned}
\tag{6}
$$

Input: $x(t) \in \mathbb{R}^{1\times D}$

Output: $y(t) \in \mathbb{R}^{1\times D}$

Latent State: $h(t) \in \mathbb{R}^{N\times D}$

$\boldsymbol{A} \in \mathbb{R}^{N\times N}$

$\boldsymbol{B}, \boldsymbol{D} \in \mathbb{R}^{N\times 1}, \boldsymbol{C} \in \mathbb{R}^{1\times N}$

where, N is the number of dimensions in the latent state, can be any numbers you want.

D is the length of the input and output.

Structure and Dimensions

$$
\begin{aligned}
h'(t) &= \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \\
y(t) &= \boldsymbol{C}h'(t) + \boldsymbol{D}x(t)
\end{aligned}
\tag{7}
$$

Input: $x(t) \in \mathbb{R}^{1\times D}$
Output: $y(t) \in \mathbb{R}^{1\times D}$
Latent State: $h(t) \in \mathbb{R}^{N\times D}$

$\boldsymbol{A} \in \mathbb{R}^{N\times N}$
$\boldsymbol{B}, \boldsymbol{D} \in \mathbb{R}^{N\times 1}, \boldsymbol{C} \in \mathbb{R}^{1\times N}$

where, N is the number of dimensions in the latent state, can be any numbers you want.
D is the length of the input and output.

For computing efficiently, the $\boldsymbol{A}$ matrix should have imposing structure (diagonal).

SSM Architectures – H3 Layer

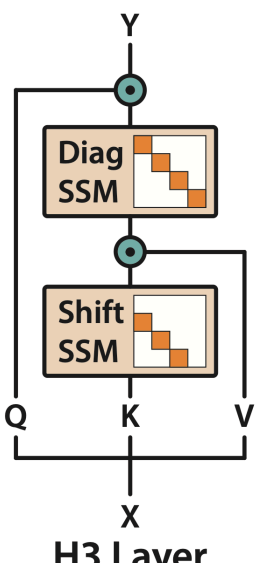
A single SSM is a standalone sequence transformation

Some basic transformation for nn layers:

- **Linear**: Matrix multiplication, $y = wx + b$
- **CNN**: Convolutional and correlation, $y = k * x$
- **RNN**: Recurrent Cell, $h_t = ux_t + wh_{t-1}$

H3 Layer

- Two SSM transformation with two gated connections
- $Q \odot \mathrm{SSM}_{\mathrm{diag}}(\mathrm{SSM}_{\mathrm{shift}}(K) \odot V)$
- $\odot$ is point-wise multiplication



Mamba

Mamba Three Contributions

1. Selection Mechanism
2. Hardware-aware Algorithm
3. Simpler Architecture

Selection Mechanism – Problem

The fundamental problem of sequence modeling is compressing context into a smaller state

Selection Mechanism – Problem

The fundamental problem of sequence modeling is compressing context into a smaller state

Transformer: Great, but not efficient.

Selection Mechanism – Problem

The fundamental problem of sequence modeling is compressing context into a smaller state

Transformer: Great, but not efficient.

RNN: Efficient, but limited by the context.

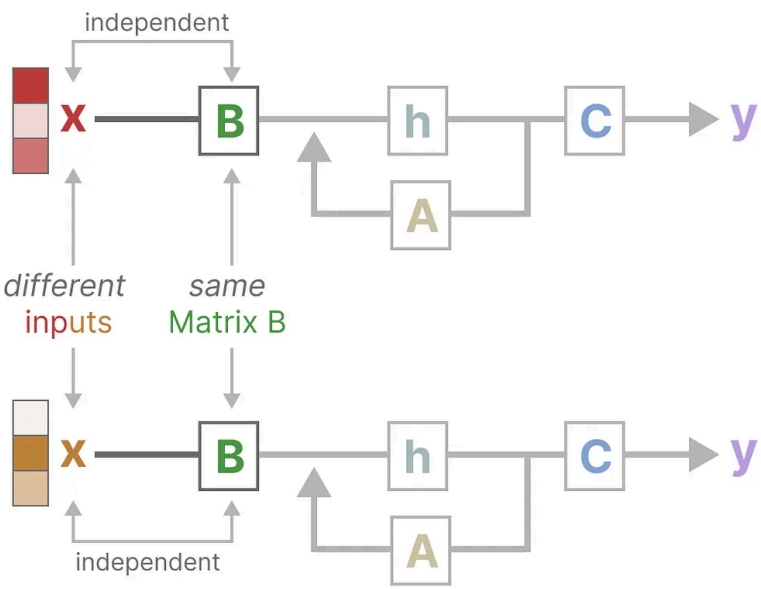
Selection Mechanism – Problem

The fundamental problem of sequence modeling is compressing context into a smaller state

Transformer: Great, but not efficient.

RNN: Efficient, but limited by the context.

SSM: Fixed the matrix A, B, C, D, also limited by the context.



Selection Mechanism

Algorithm 1 SSM (S4)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
3: $\boldsymbol{C} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
4: $\Delta : (\mathtt{D}) \leftarrow \tau_\Delta$(Parameter)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow$ discretize$(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \mathrm{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
3: $\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
4: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta$(Parameter$+s_\Delta(x)$)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow$ discretize$(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \mathrm{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

Algorithms 1 and 2 illustrates the main selection mechanism that we use. The main difference is simply making several parameters $\boldsymbol{\Delta}, \boldsymbol{B}, \boldsymbol{C}$ functions of the input, along with the associated changes to tensor shapes throughout. In particular, we highlight that these parameters now have a length dimension L, meaning that the model has changed from time-invariant to time-varying. (Note that shape annotations were described in Section 2). This loses the equivalence to convolutions (3) with implications for its efficiency, discussed next.

$$s_{B(x)} = \mathrm{Linear}_N(x); s_{C(x)} = \mathrm{Linear}_N(x); s_{\Delta(x)} = \mathrm{Broadcast}_D(\mathrm{Linear}_1(x))$$

Hardware-aware Algorithm

Since BCD are dynamic, the kernel of SSM convolution is dynamic. Thus, the parallel convolution during the training is not possible.

Hardware-aware Algorithm

Since BCD are dynamic, the kernel of SSM convolution is dynamic. Thus, the parallel convolution during the training is not possible.

Mamba proposed **parallel scan** to solve this problem.

$$\begin{aligned} \text{scan}(\text{func}, \text{init}, \text{list}) &\rightarrow \text{list} \\ \text{scan}(\text{add}, 0, [1..4]) &= [1, \\ &\quad \text{add}(1, 2), \\ &\quad \text{add}(\text{add}(1, 2), 3), \\ &\quad \text{add}(\text{add}(\text{add}(1, 2), 3), 4)] \end{aligned} \tag{9}$$

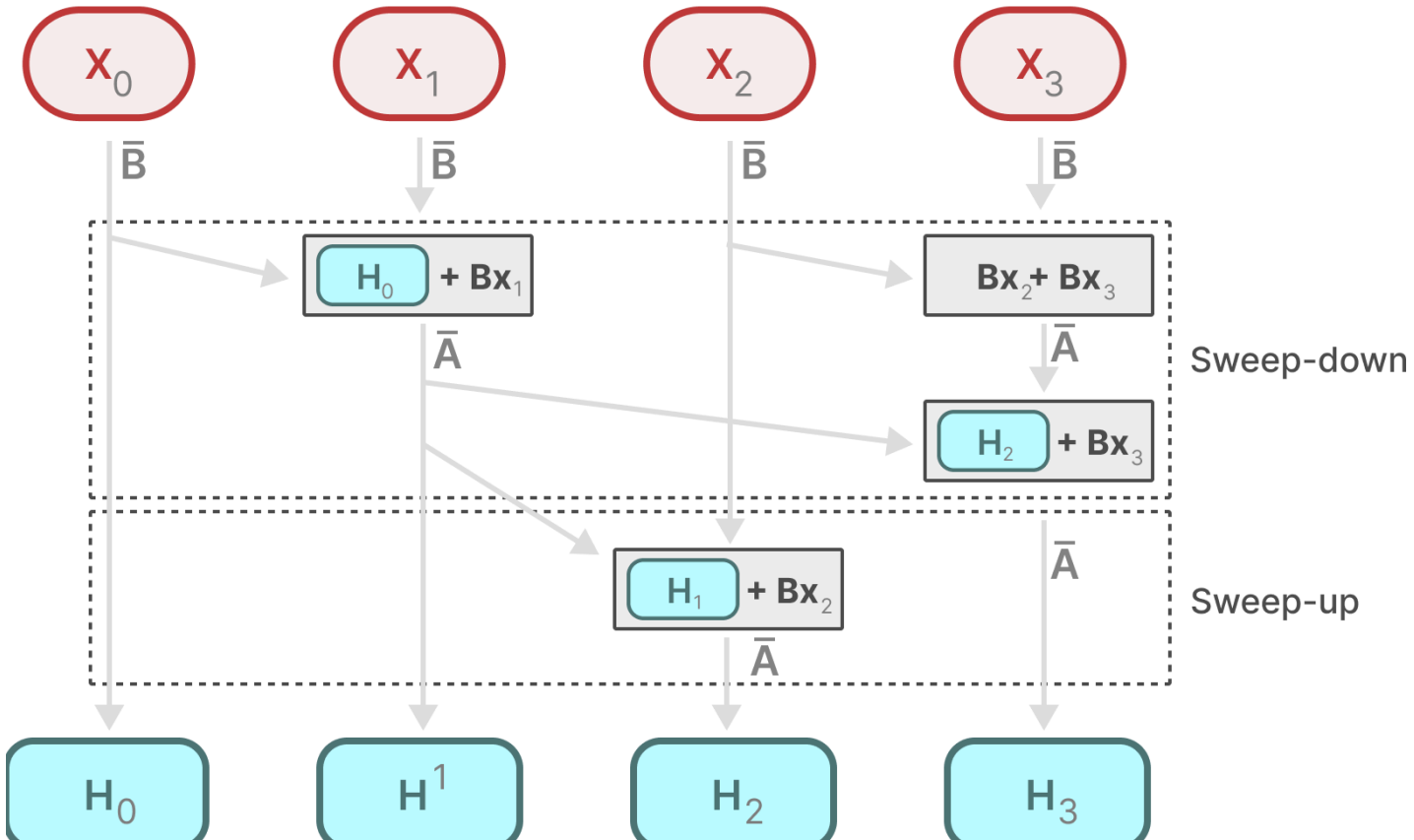
Mamba Parallel Scan I

$$h_t = \bar{\boldsymbol{A}} h_{t-1} + \bar{\boldsymbol{B}} x_t \tag{10}$$

$$\begin{aligned} h_1 &= \bar{\boldsymbol{A}} h_0 + \bar{\boldsymbol{B}} x_1 \\ h_2 &= \bar{\boldsymbol{A}} h_1 + \bar{\boldsymbol{B}} x_2 = \bar{\boldsymbol{A}}\left(\bar{\boldsymbol{A}} h_0 + \bar{\boldsymbol{B}} x_1\right) + \bar{\boldsymbol{B}} x_2 \\ h_3 &= \bar{\boldsymbol{A}} h_2 + \bar{\boldsymbol{B}} x_3 = \bar{\boldsymbol{A}}\left(\bar{\boldsymbol{A}}\left(\bar{\boldsymbol{A}} h_0 + \bar{\boldsymbol{B}} x_1\right) + \bar{\boldsymbol{B}} x_2\right) + \bar{\boldsymbol{B}} x_3 \end{aligned} \tag{11}$$

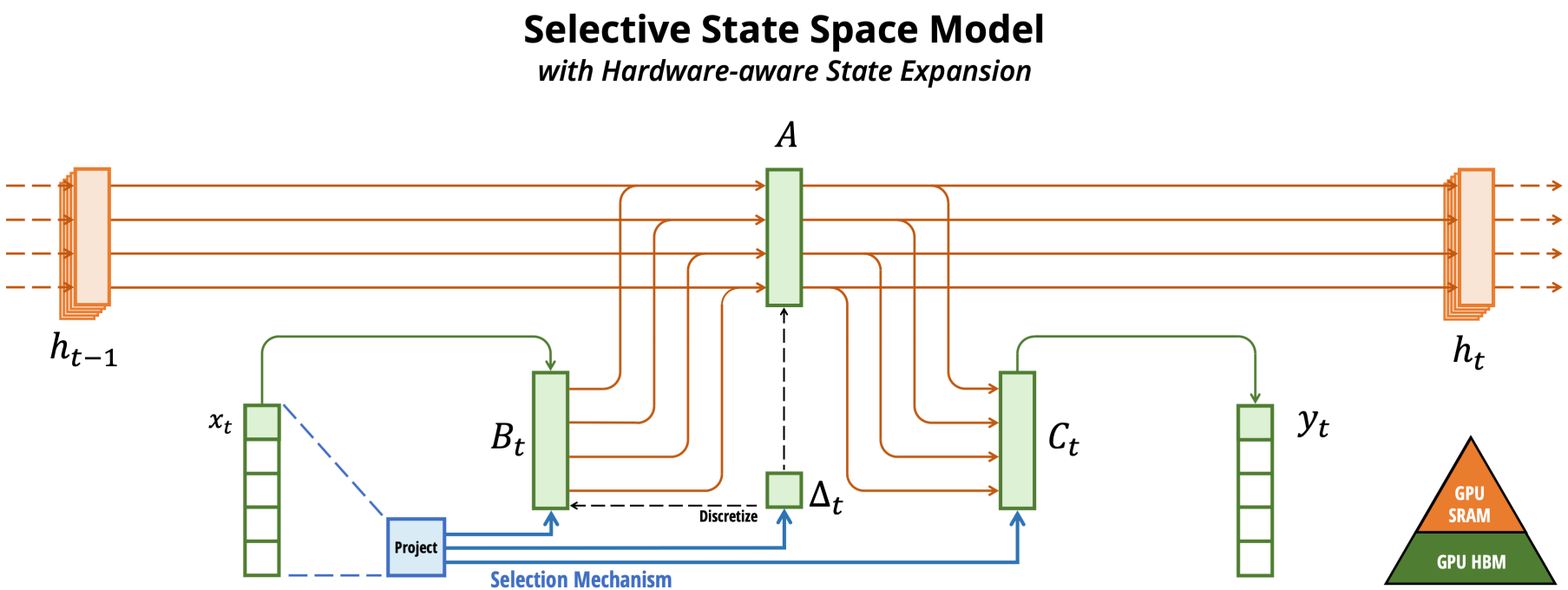
E.g.,

when calculating $\bar{\boldsymbol{A}} h_0 + \bar{\boldsymbol{B}} x_1$, we can parallel calculate the $\bar{\boldsymbol{B}} x_2 + \bar{\boldsymbol{B}} x_3$.

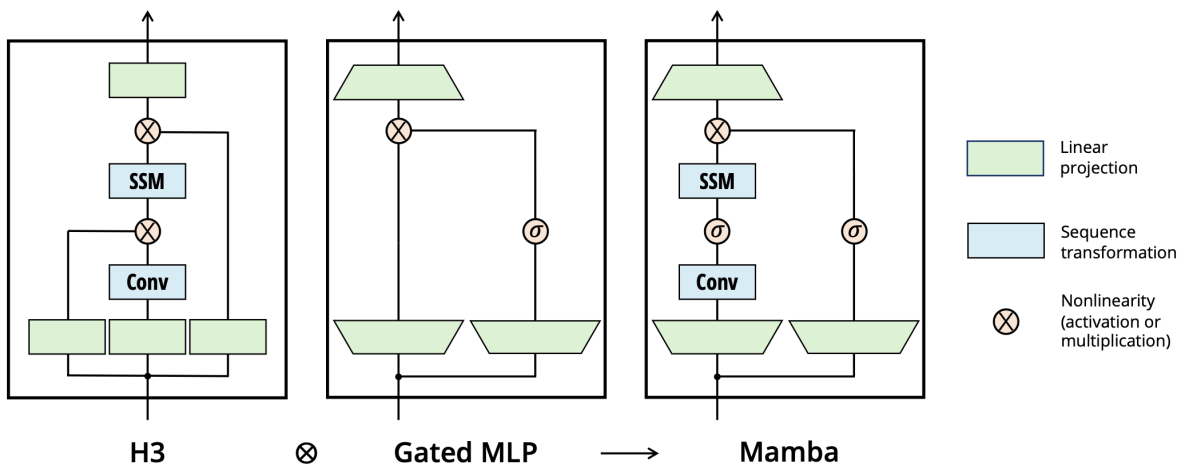


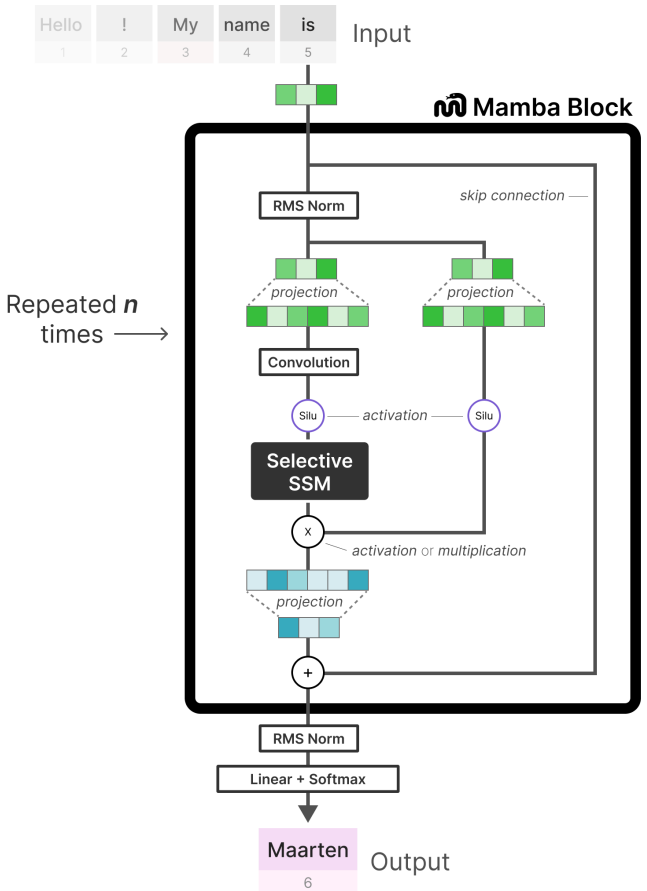
Parallel computation **O(n/t)**

Hardware-aware Scan



Simpler Architecture





Combined the H3 layer with Gated MLP to form the Mamba model.

Experiments

Datasets

Training

- The Pile: An 800GB Dataset of Diverse Text for Language Modeling (Leo Gao et al. 2020)
- SlimPajama: A 627B token, cleaned and deduplicated version of RedPajama (Daria Soboleva et al, 2023)

Evaluation

- lm-eval-harness (Leo Gao et al, 2021)

Results I

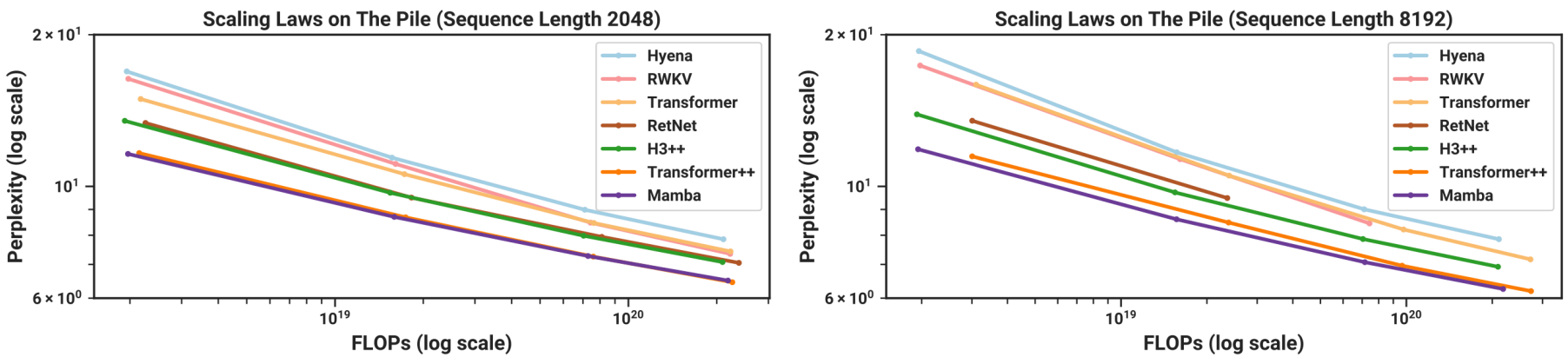


Figure 4: (**Scaling Laws.**) Models of size $\approx 125M$ to $\approx 1.3B$ parameters, trained on the Pile. Mamba scales better than all other attention-free models and is the first to match the performance of a very strong "Transformer++" recipe that has now become standard, particularly as the sequence length grows.

Results II

Table 3: (**Zero-shot Evaluations**.) Best results for each size in bold. We compare against open source LMs with various tokenizers, trained for up to 300B tokens. Pile refers to the validation split, comparing only against models trained on the same dataset and tokenizer (GPT-NeoX-20B). For each model size, Mamba is best-in-class on every single evaluation result, and generally matches baselines at twice the model size.

Model	Token.	Pile ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	Arc-E acc ↑	Arc-C acc ↑	WinoGrande acc ↑	Average acc ↑
Hybrid H3-130M	GPT2	—	89.48	25.77	31.7	64.2	44.4	24.2	50.6	40.1
Pythia-160M	NeoX	29.64	38.10	33.0	30.2	61.4	43.2	24.1	**51.9**	40.6
Mamba-130M	NeoX	**10.56**	**16.07**	**44.3**	**35.3**	**64.5**	**48.0**	**24.3**	**51.9**	**44.7**
Hybrid H3-360M	GPT2	—	12.58	48.0	41.5	68.1	51.4	24.7	54.1	48.0
Pythia-410M	NeoX	9.95	10.84	51.4	40.6	66.9	52.1	24.6	53.8	48.2
Mamba-370M	NeoX	**8.28**	**8.14**	**55.6**	**46.5**	**69.5**	**55.1**	**28.0**	**55.3**	**50.0**
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	51.9
Mamba-790M	NeoX	**7.33**	**6.02**	**62.7**	**55.1**	**72.1**	**61.2**	**29.5**	**56.1**	**57.1**
GPT-Neo 1.3B	GPT2	—	7.50	57.2	48.9	71.1	56.2	25.9	54.9	52.4
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	53.0
OPT-1.3B	OPT	—	6.64	58.0	53.7	72.4	56.7	29.6	59.5	55.0
Pythia-1.4B	NeoX	7.51	6.08	61.7	52.1	71.0	60.5	28.5	57.2	55.2
RWKV-1.5B	NeoX	7.70	7.04	56.4	52.5	72.4	60.5	29.4	54.6	54.3
Mamba-1.4B	NeoX	**6.80**	**5.04**	**64.9**	**59.1**	**74.2**	**65.5**	**32.8**	**61.5**	**59.7**
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8	72.1	61.1	30.2	57.6	56.5
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	58.0
OPT-2.7B	OPT	—	5.12	63.6	60.6	74.8	60.8	31.3	61.0	58.7
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3	74.0	64.1	32.9	59.7	59.1
RWKV-3B	NeoX	7.00	5.24	63.9	59.6	73.7	67.8	33.1	59.6	59.6
Mamba-2.8B	NeoX	**6.22**	**4.23**	**69.2**	**66.1**	**75.2**	**69.7**	**36.3**	**63.5**	**63.3**
GPT-J-6B	GPT2	–	4.10	68.3	66.3	75.4	67.0	36.6	64.1	63.0
OPT-6.7B	OPT	–	4.25	67.7	67.2	76.3	65.6	34.9	65.5	62.9
Pythia-6.9B	NeoX	6.51	4.45	67.1	64.0	75.2	67.3	35.5	61.3	61.7
RWKV-7.4B	NeoX	6.31	4.38	67.2	65.5	76.1	67.8	37.5	61.0	62.5

Results III

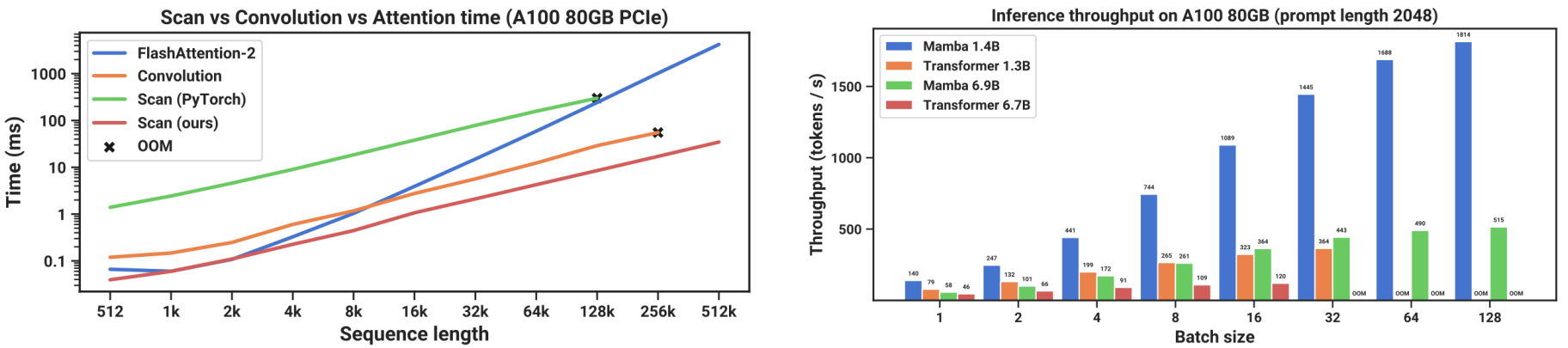


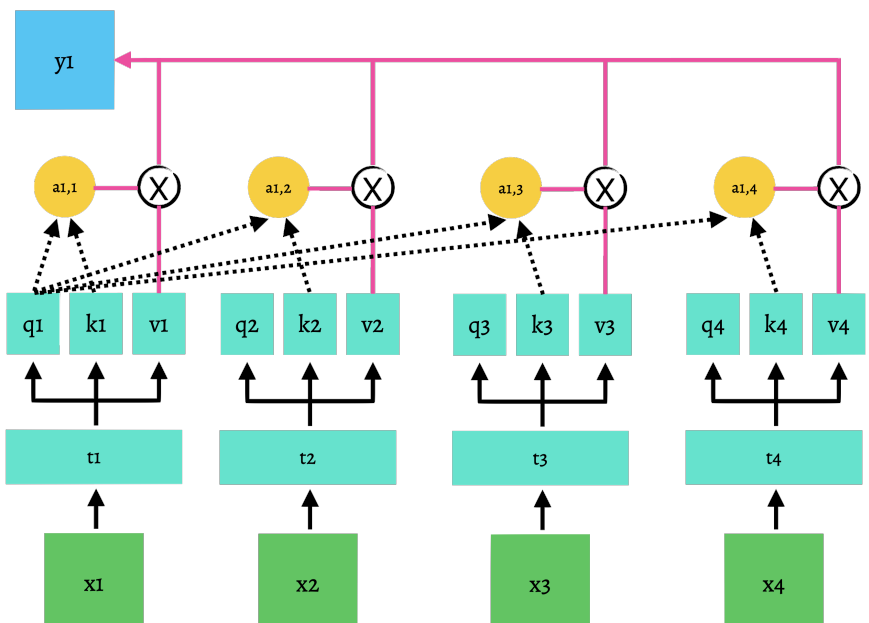
Figure 8: (**Efficiency Benchmarks**.) (*Left*) Training: our efficient scan is 40× faster than a standard implementation. (*Right*) Inference: as a recurrent model, Mamba can achieve 5× higher throughput than Transformers.

More Results And Ablation Studies

For the complete results and ablation studies,
please see the Paper: https://arxiv.org/abs/2312.00752

Linear Attention

Transformer



$$a_{1,i} = \text{softmax}\left(\frac{q_1 \cdot k_i}{\sqrt{d}}\right)$$

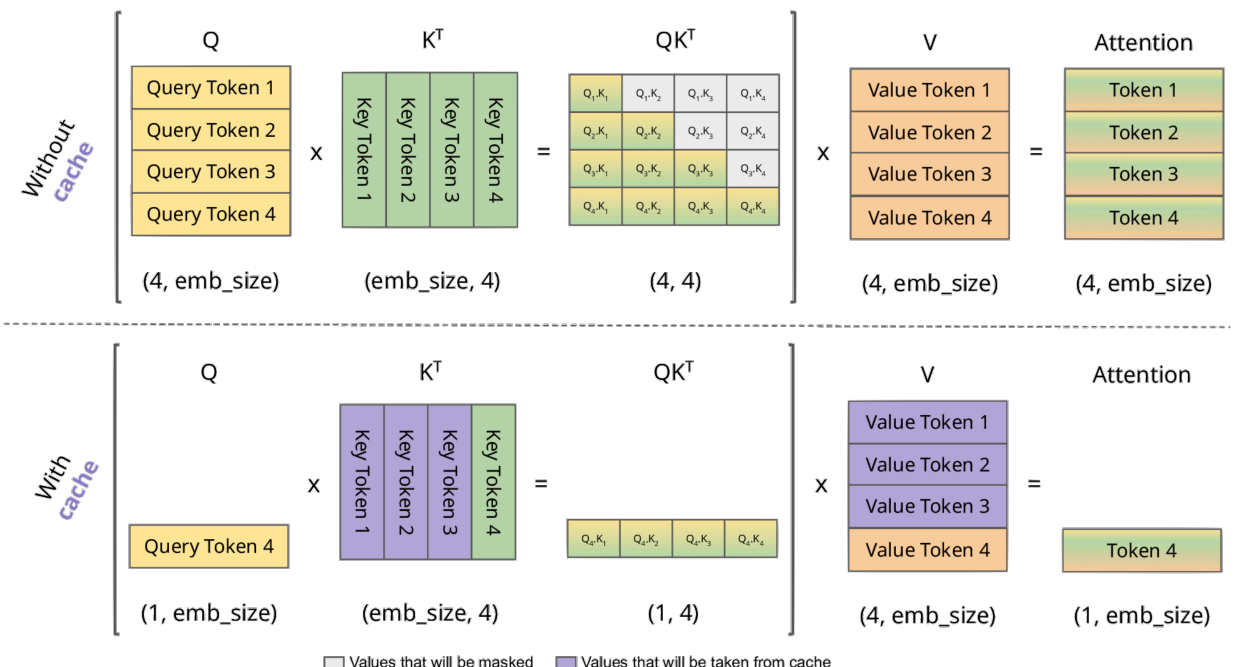
$$y_1 = \sum_i a_{1,i} v_i$$

$$QK \rightarrow O(n^2 d)$$

$$\text{softmax} \rightarrow O(n^2)$$

$$\sum \rightarrow O(n^2 d)$$

KV Cache


Without cache
Q
K^T
QK^T
V
Attention
Query Token 1
Query Token 2
Query Token 3
Query Token 4
Key Token 1
Key Token 2
Key Token 3
Key Token 4
Value Token 1
Value Token 2
Value Token 3
Value Token 4
Token 1
Token 2
Token 3
Token 4
(4, emb_size)
(emb_size, 4)
(4, 4)
(4, emb_size)
(4, emb_size)
With cache
Q
K^T
QK^T
V
Attention
Query Token 4
Key Token 1
Key Token 2
Key Token 3
Key Token 4
Value Token 1
Value Token 2
Value Token 3
Value Token 4
Token 4
(1, emb_size)
(emb_size, 4)
(1, 4)
(4, emb_size)
(1, emb_size)
Values that will be masked
Values that will be taken from cache

Convolutions

SSM

- What is SSM.
- Why? Continuous, recurrent, convolutional.
- Common SSM Layers – S4 and H3

Mamba

- Selection Mechanism
- Hardware-aware Algorithm
- Simpler Architecture

Linear Attention

- KV Cache