Mamba: Linear-Time Sequence Modeling with Selective State Spaces

(Mamba Theory)

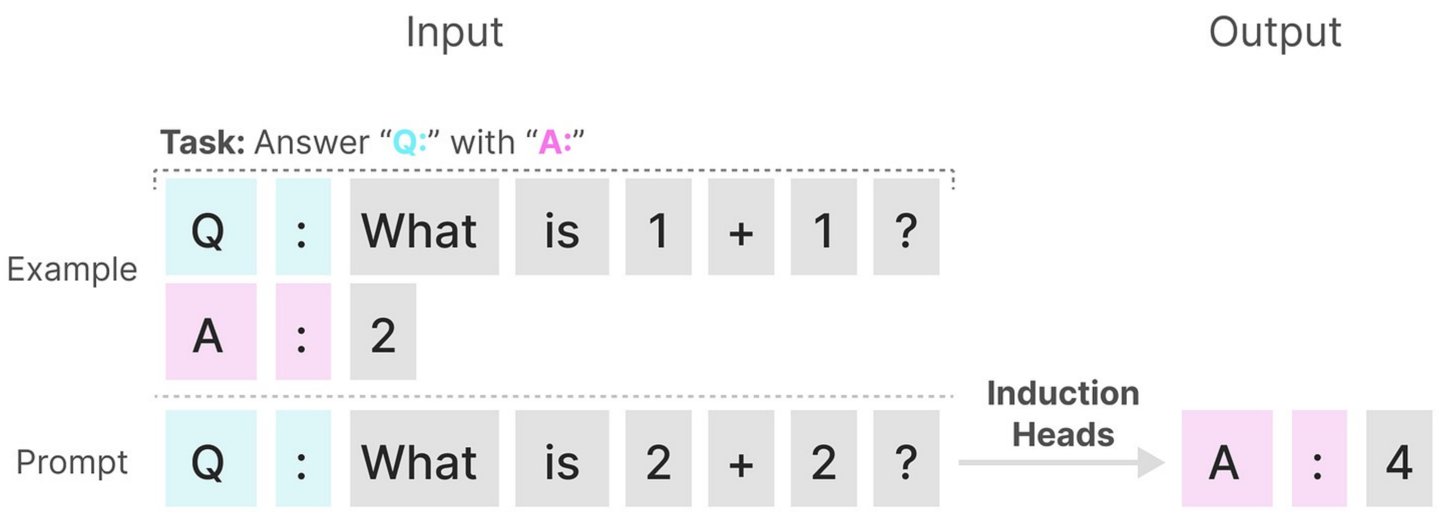
Mamba is not a Snake🐍!

Chapter

- 1. Problems with S4
- 2. Why S4 performs poorly on Language tasks
- 3. Why Mamba: Selective Scan
- 4. Why Mamba: Hardware-aware algorithm
- 5. Mamba Block
- 6. Code (Mamba 2.8b Inference)

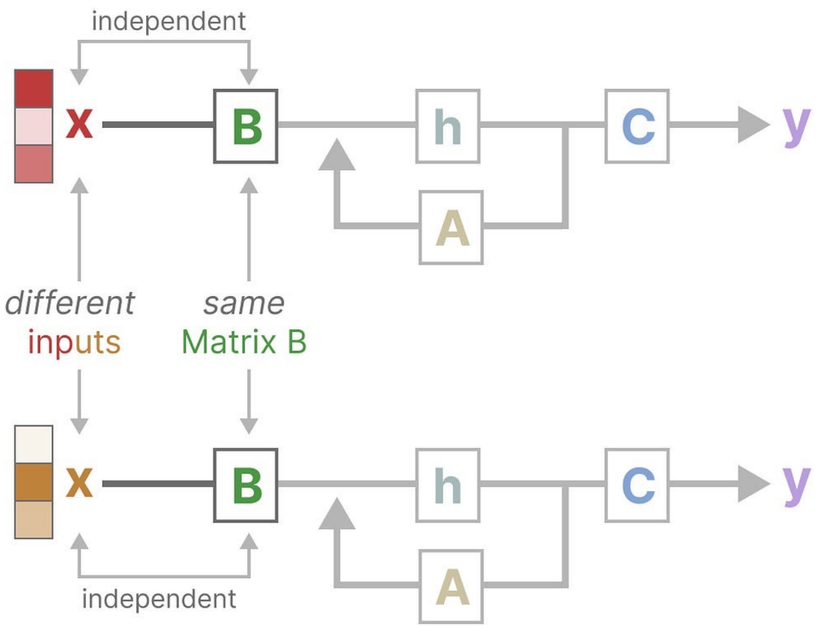
Problems with S4

- ***Key Problem***: State Space Models, and even the S4 (Structured State Space Model), perform poorly on certain tasks that are vital in language modeling and generation, namely *the ability to focus on or ignore particular inputs*.
- E.g.



S4 cannot get the result since it is time invariant and thus cannot select info from history

Why S4 performs poorly on Language tasks



Constant regardless of the input

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

$$y_k = Ch_k$$

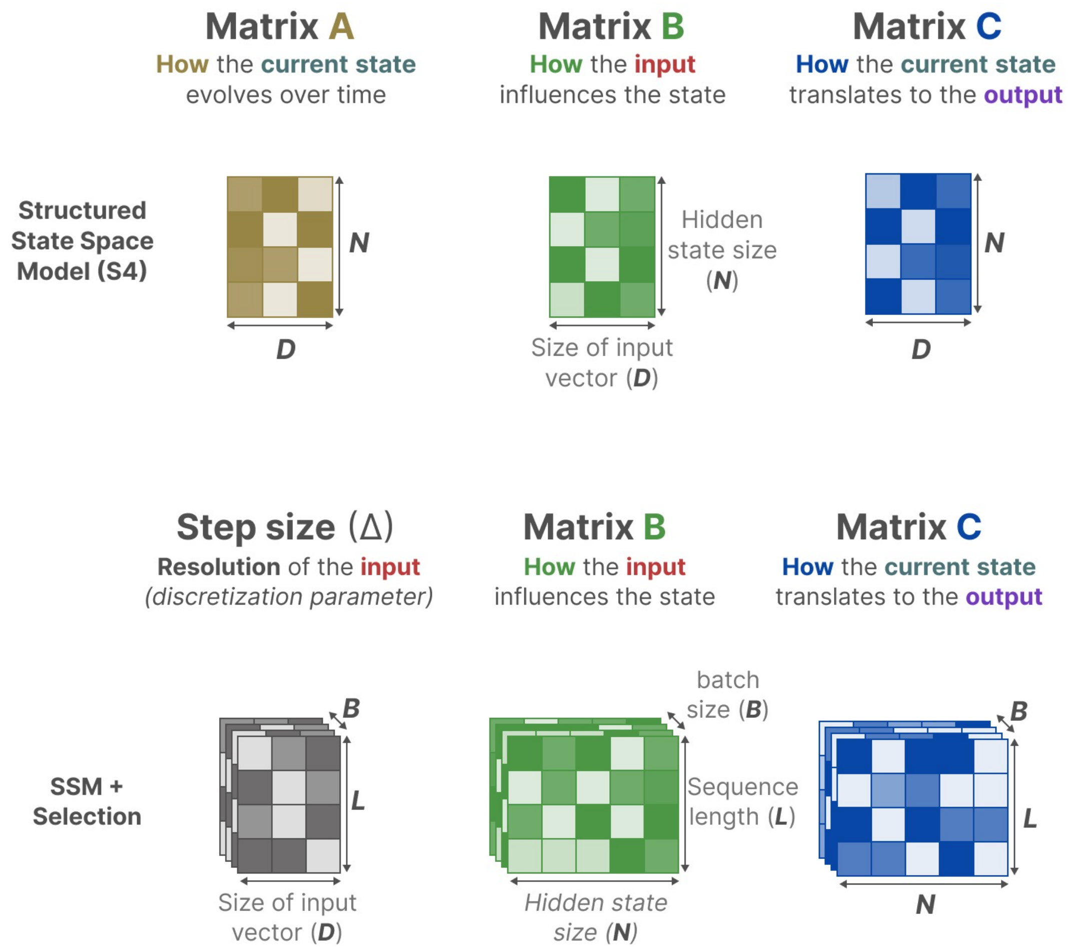
Discretized matrix **A** $$\bar{A} = \exp(\Delta A)$$

Discretized matrix **B** $$\bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B$$

Matrices A, B, and C are fixed after training; they do not change regardless of the input.
This results in an inability to focus on specific parts of the input.

Ref:[A Visual Guide to Mamba and State Space Models]

Why Mamba: Selective Scan



S4(A, B, C are fixed)

Mamba:
(B, C, Δ comes from the input x)

B = Linear1(x)
C = Linear2(x)
Δ=Linear3(x)

A is still fixed after training. (See the paper Ablation part for more details

Why Mamba: Selective Scan

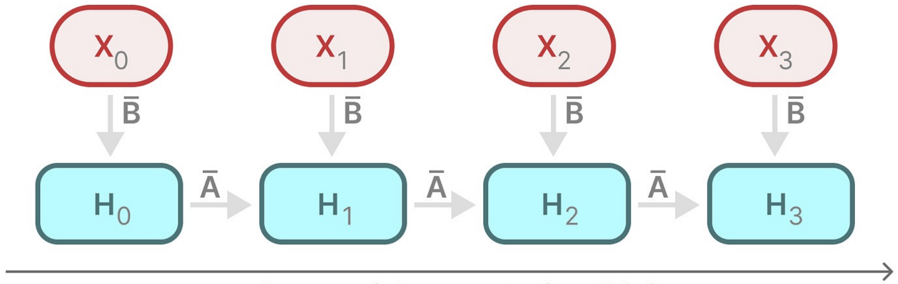
Interpretation of A. We remark that while the $\boldsymbol{A}$ parameter could also be selective, it ultimately affects the model only through its interaction with Δ via $\overline{\boldsymbol{A}} = \exp(\Delta \boldsymbol{A})$ (the discretization (4)). Thus selectivity in Δ is enough to ensure selectivity in $(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}})$, and is the main source of improvement. We hypothesize that making $\boldsymbol{A}$ selective in addition to (or instead of) Δ would have similar performance, and leave it out for simplicity.

Table 8: (**Ablations: Parameterization of A.**) The more standard initializations based on S4D-Lin (Gu, Gupta, et al. 2022) perform worse than S4D-Real or a random initialization, when the SSM is selective.

$\boldsymbol{A}_n$ Initialization	Field	Perplexity
$\boldsymbol{A}_n = -\frac{1}{2} + ni$	Complex	9.16
$\boldsymbol{A}_n = -1/2$	Real	8.85
$\boldsymbol{A}_n = -(n+1)$	Real	8.71
$\boldsymbol{A}_n \sim \exp(\mathcal{N}(0,1))$	Real	8.71

Selective Δ	Selective $\boldsymbol{B}$	Selective $\boldsymbol{C}$	Perplexity
✗	✗	✗	10.93
✗	✓	✗	10.15
✗	✗	✓	9.98
✓	✗	✗	9.81
✓	✓	✓	8.71

Why Mamba: Selective Scan

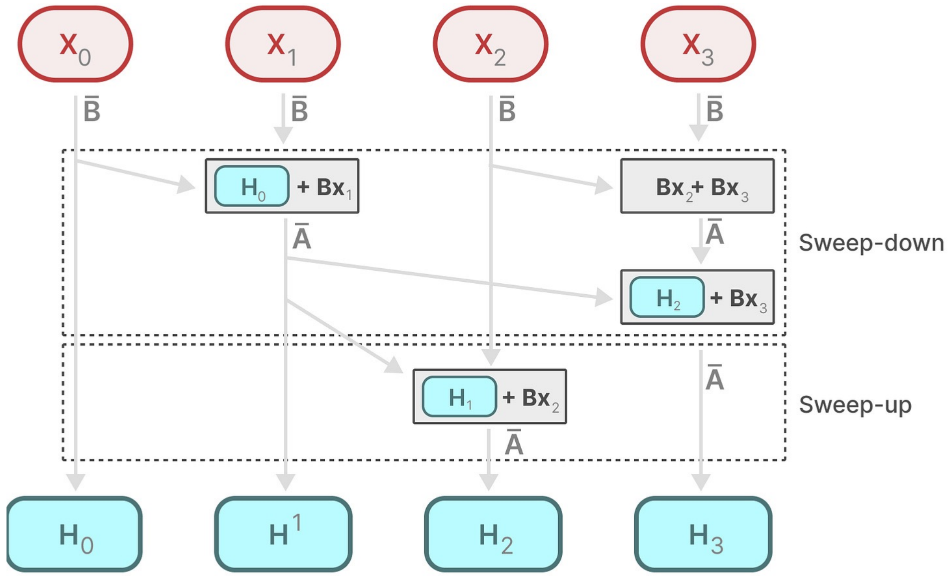


Sequential computation **O(n)**

Before: O(n)

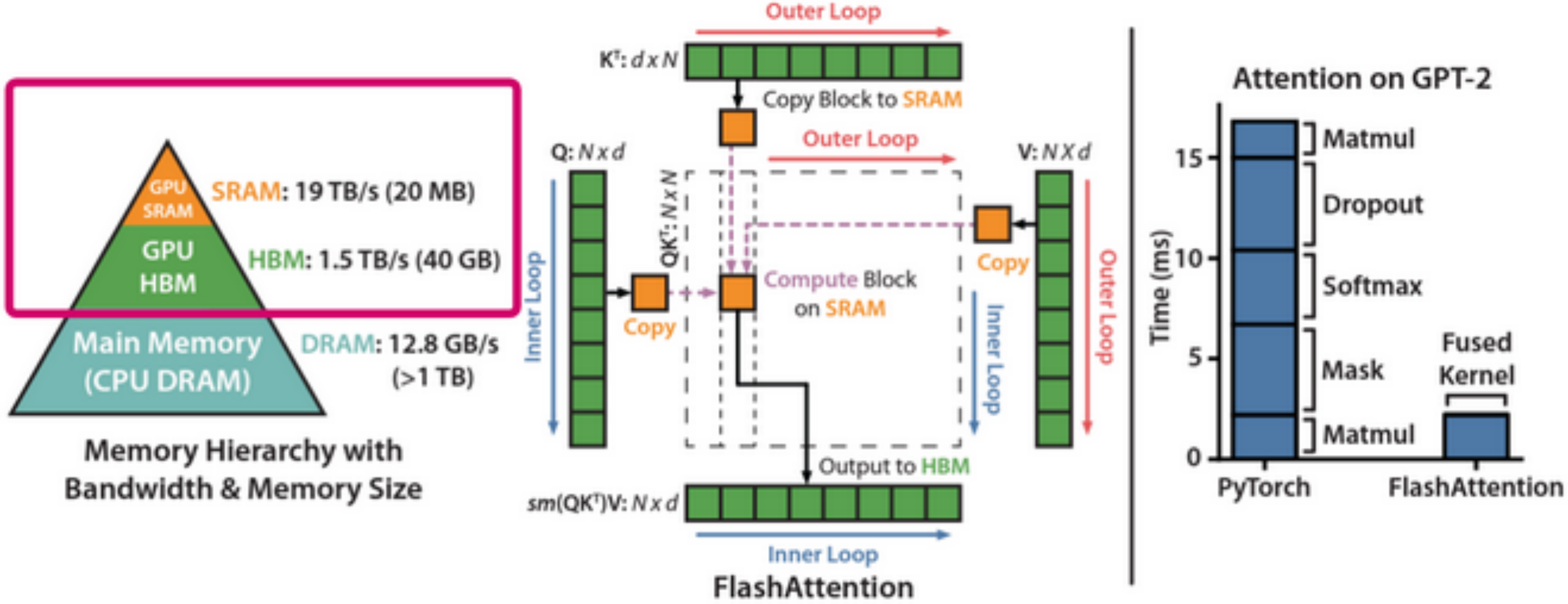
After: O(n/t)

$$\mathrm{H}_1 = H_0 + \bar{B}x_1$$
$$\mathrm{H}_2 = H_1 + \bar{B}x_2$$
$$= (H_0 + \bar{B}x_1) + \bar{B}x_2$$



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

Why Mamba: Hardware-aware algorithm

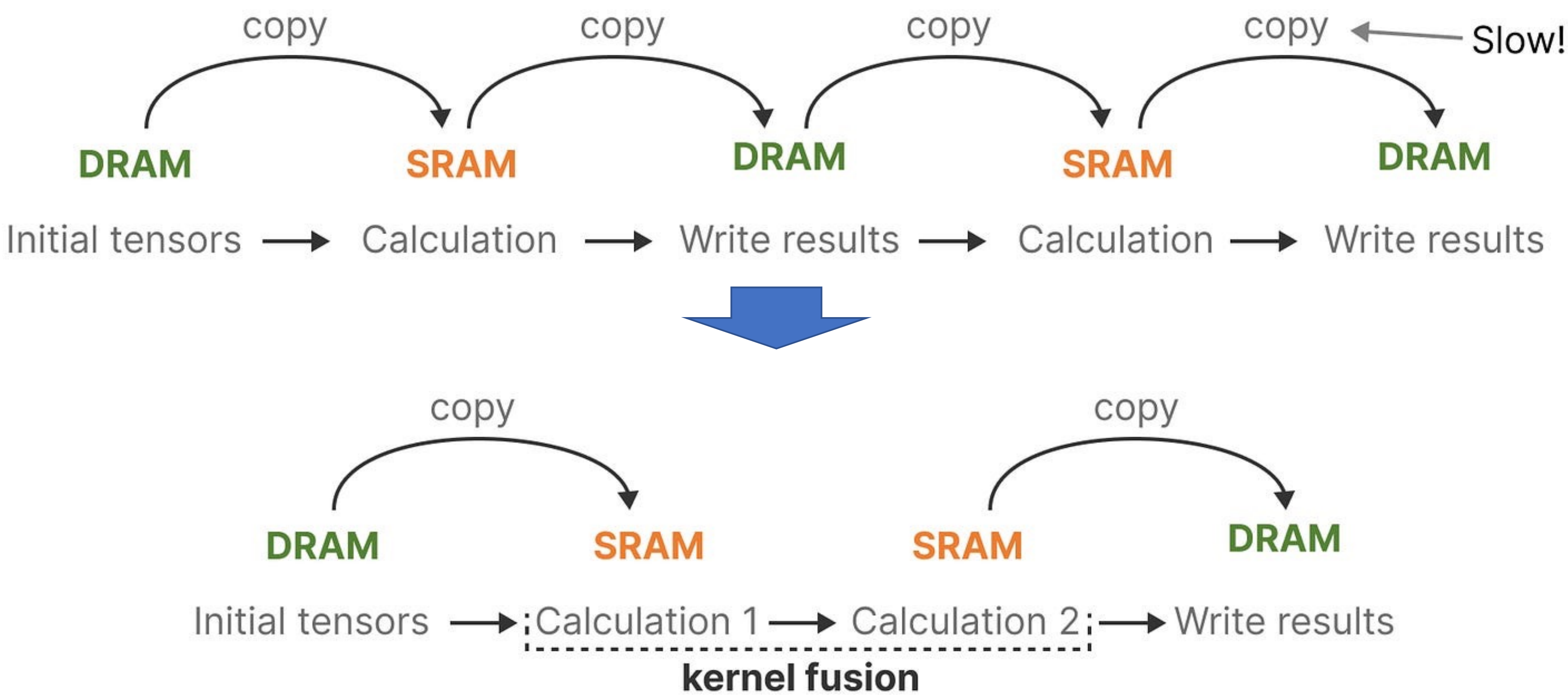


Data1(CPU) -> HBM(data.to("cuda:0") -> SRAM -> Computation Core

Global Local

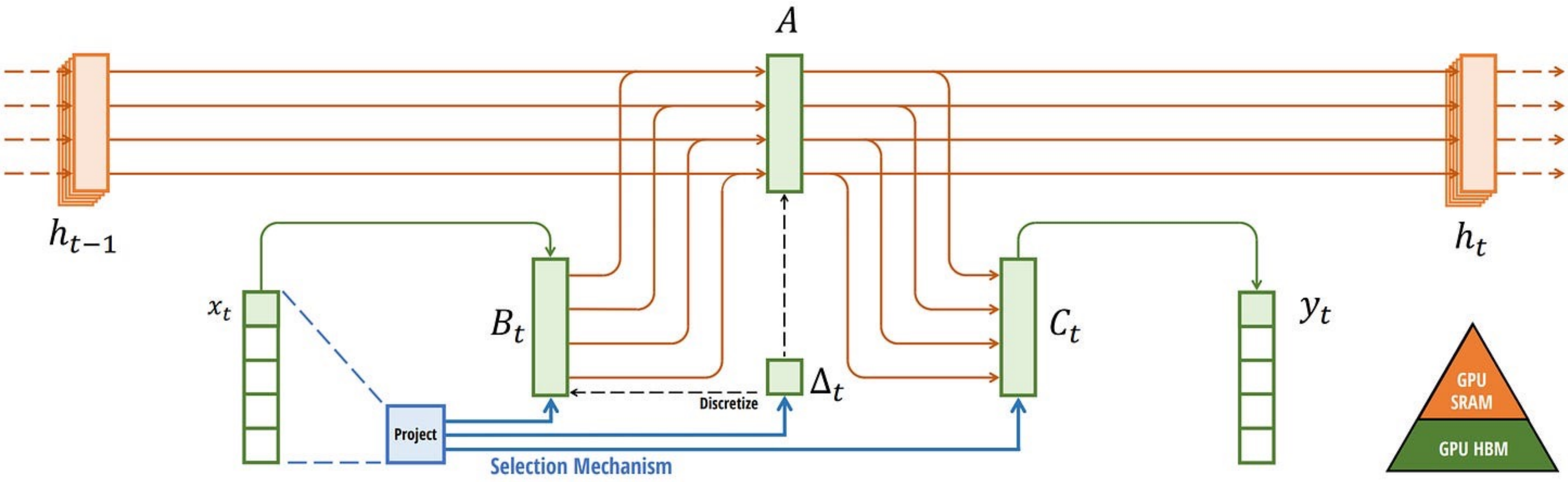
Ref: [GPU存储的结构]

Why Mamba: Hardware-aware algorithm



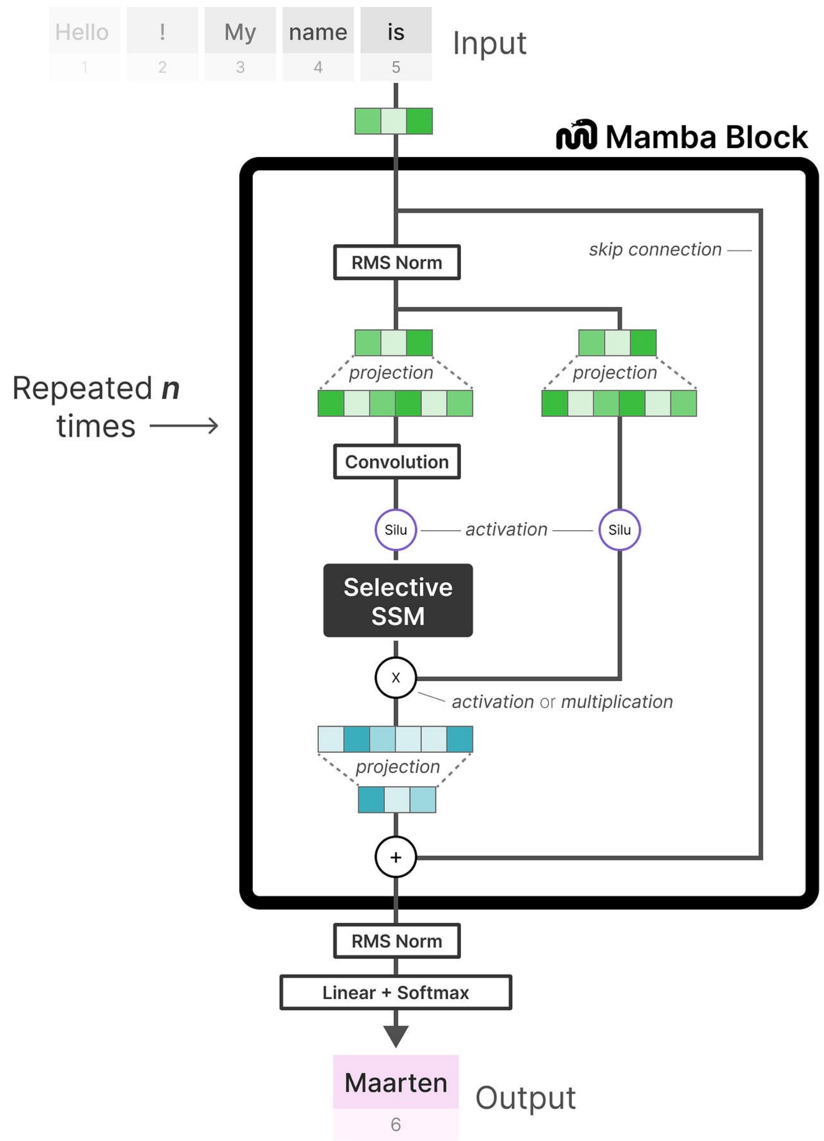
Ref: [GPU存储的结构]

Why Mamba: Hardware-aware algorithm



Ref: [GPU存储的结构]

Mamba Block



Hello
!
My
name
is
1
2
3
4
5
Input
Mamba Block
Repeated *n* times
RMS Norm
skip connection
projection
projection
Convolution
Silu
activation
Silu
Selective SSM
X
activation or multiplication
projection
+
RMS Norm
Linear + Softmax
Maarten
6
Output

Mamba Block How LLM Inference?

Sample:
Prefill:
6 tokens -> ?
[-1] last token distribution -> sample
a[1, 6, 5120] -> [1, 1, 5120] -> sample
a[:,-1,:]

Decode:
1 token -> 1 distribution -> sample

Input: Prefill
Q: Hello! My name is

(tokenizer)

Hello | ! | My | name | is -> (id)

(Embedding Layer) vocab_size x hidden_dim(100)
Hello 1 (row 1 -> Hello vector)

(Transformer Block) | (Mamba Block)
(KV cache) (Mamba cache)

Output:

Distribution
(vocab_size = 200)
Output: 1x200

Sample:
Greedy (argmax)
T, sample
B S… -> (4 max)
(vocab 4: Tom)

Input: Decode
Tom

(tokenizer)

Tom (id)

(Embedding Layer)
Tom (vector)

(Transformer / Mamba Block)

Output:

Distribution
Output: 1x200

Sample
.
.
.
.
(Stop: 1. token=stop(EOS)
Stop: 2. max length)

Embedding Layer

Input: Token id
Output: Embedding states

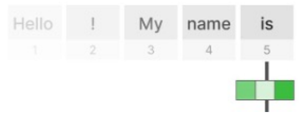
E.g.

Input sentence: Hello !
Tokenizer results: [1, 2]

Results (shape: [2,3]) :

0.1	0.1	0.1
0.2	0.2	0.2

Embedding



Input

Embedding Layer:
nn.Parameters(vocab_size, embed_dim)

Hello	!	My	name	is	Tom
1	2	3	4	5	6

Vocab

Embedding Layer (weight shape: [6,3]) :

0.1	0.1	0.1
0.2	0.2	0.2
0.3	0.3	0.3
0.4	0.4	0.4
0.5	0.5	0.5
0.6	0.6	0.6

Mamba Block

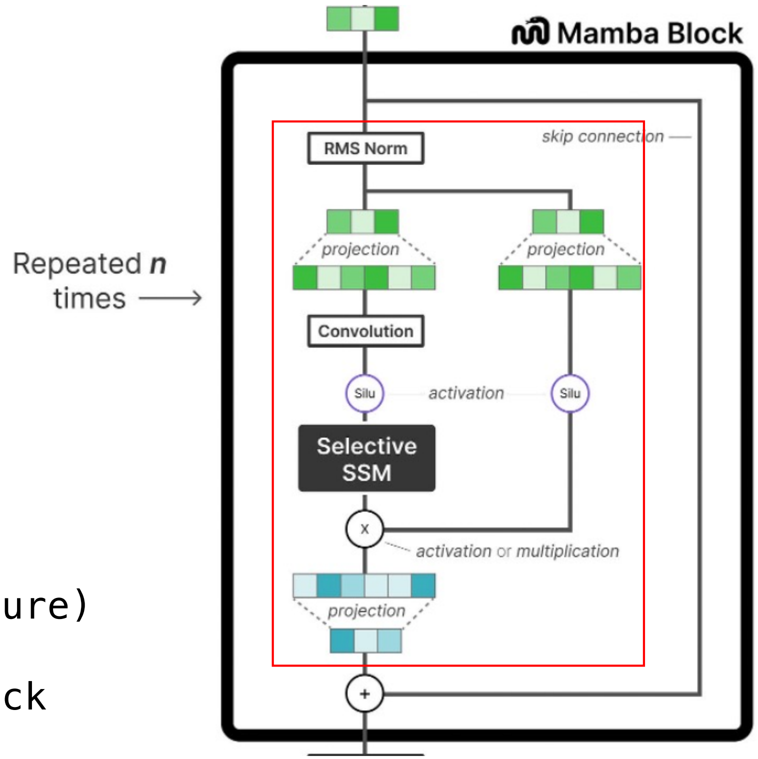
See Codes:
Modeling_mamba.py(HF: mamba 2.8b)

Class:
`MambaCache:` caches for each Mamba Block (ssm hidden states & conv states)

`MambaMixer:` 👉(Red Box)

`MambaRMSNorm:` Normalization(See mamba block figure)

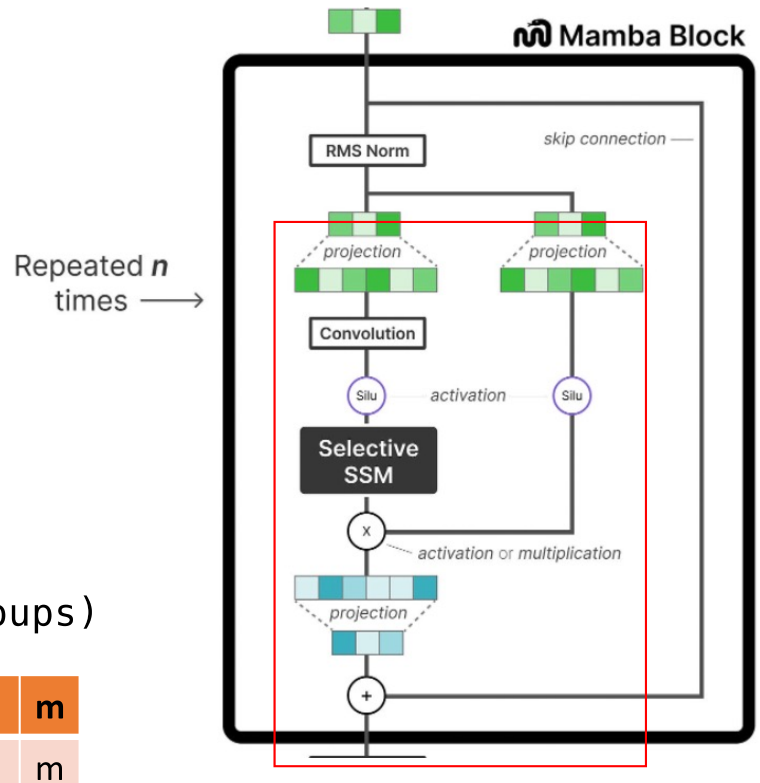
`MambaPreTrainedModel:` MambaMixer + Residual(Black box)



Mamba Block

```
Conv1d Layer:
```

ResNet
ResNeXt



(take mamba-2.8b for example)

```
Input: (bs, hidden_dim=5120, seq_len)
Conv: Conv1d(5120, 5120, kernel_size=(4,),
stride=(1,), padding=(3,), groups=5120)
```

```
data:
```

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

.
.
.

```
(w/o groups)
```

a	b	c	m
d	e	f	n
g	h	i	q

.
.
.

```
(w/ groups)
```

a	b	c	m
a	b	c	m
a	b	c	m

.
.
.

Output shape:

$$W' = floor(\frac{W + padding_w - kernel_w}{stride}) + 1$$