AI VIET NAM
@aivietnam.edu.vn

Text Classification with Mamba - Project

TA Minh-Duc Bui
STA Khai-Xuan Trinh

Outline

1. Motivation
2. State Space Models
3. Mamba
4. Coding

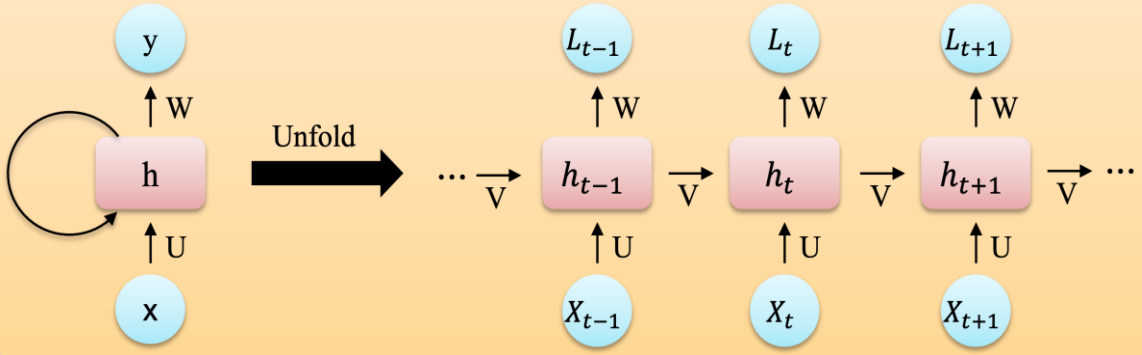
Outline

1. Motivation
2. State Space Models
3. Mamba
4. Coding

RNNs vs. Transformers

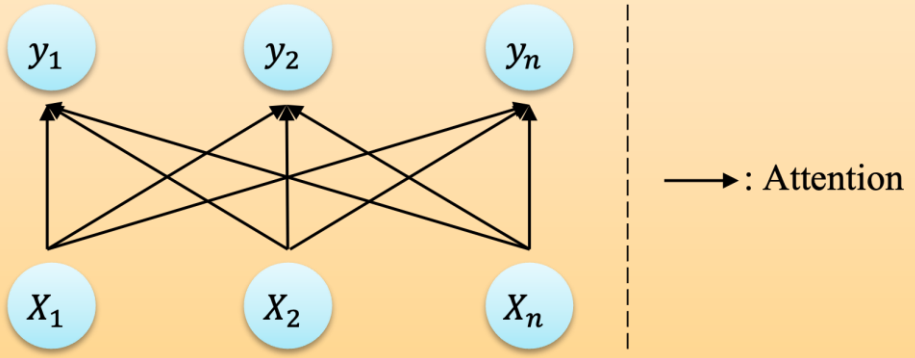
RNNs

- train stage: $O(N)$, **non-parallelable**
- inference stage: $O(N)$
- $O(1)$ per token



Transformers

- train stage: $O(N^2)$, **parallelable**
- inference stage: $O(N^2)$
- $O(N)$ per token

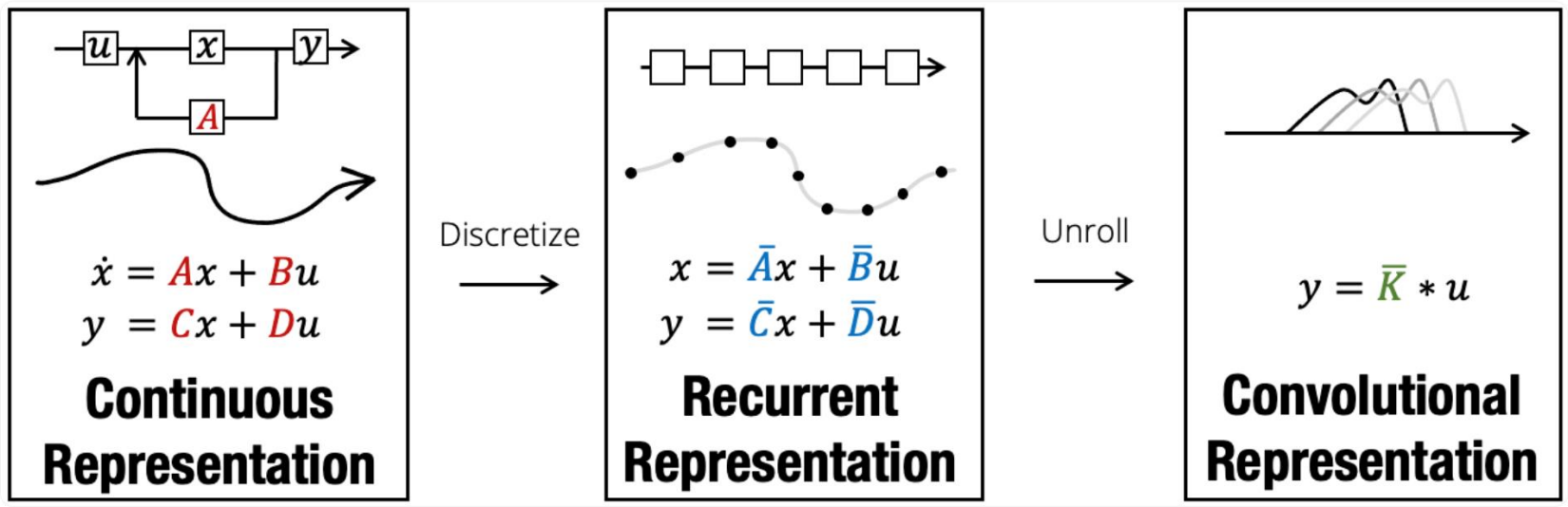


Ideal Models

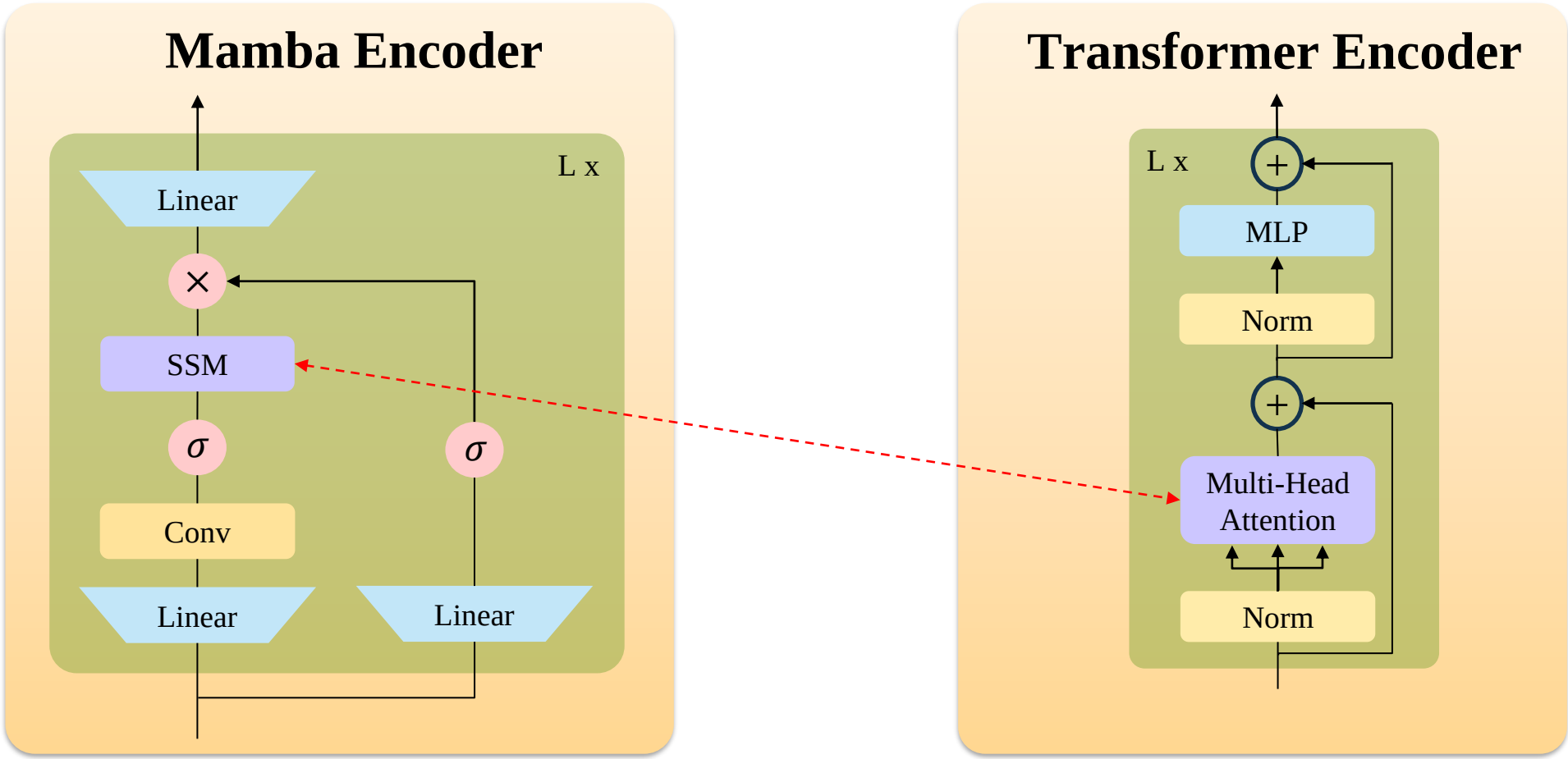
- train stage: $O(N)$, **parallelable**
- inference stage: $O(N)$
- $O(1)$ per token

RNNs vs. Transformers

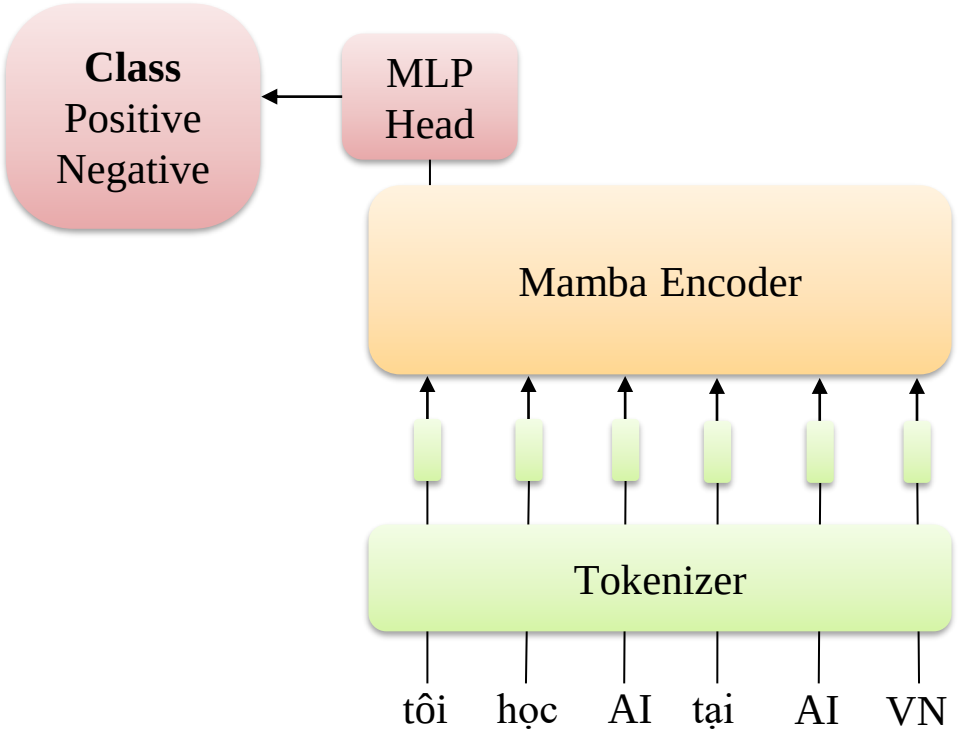
	RNN	Transformer	Ideal Model
Parallelizable	no	yes	yes
Training Stage	$O(n)$	$O(n^2)$	$O(n)$
Inference Stage	$O(n)$	$O(n^2)$	$O(n)$
Inference per Token	$O(1)$	$O(n)$	$O(1)$

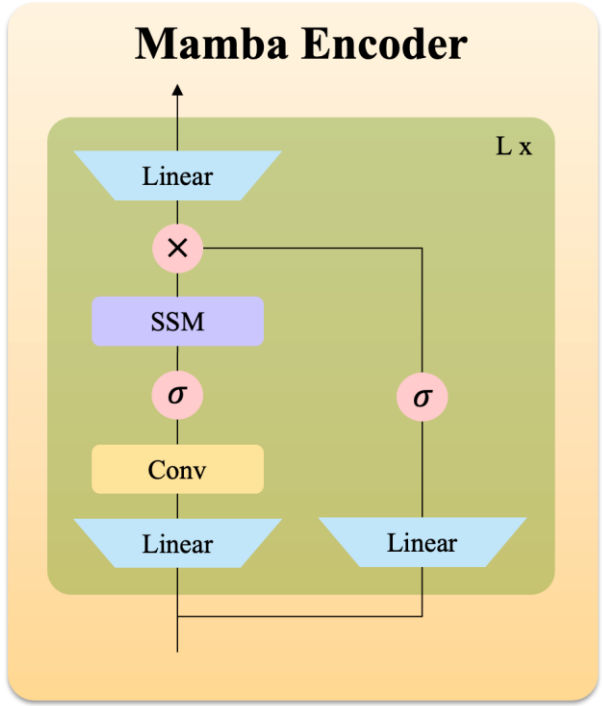


Mamba Block


Mamba Encoder
L x
Linear
SSM
Conv
Linear
Linear
Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm

Mamba Architecture





Outline

1. Motivation
2. State Space Models
3. Mamba
4. Coding

State Space Models (SSMs)

What

The SSMs are traditionally used in control theory to model a dynamic system via state variables.

How

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t)$$
$$y(t) = \mathbf{C}h(t) + \mathbf{D}x(t)$$

Why

- train stage: $O(N)$, **parallelable**
- inference stage: $O(N)$
- $O(1)$ per token

Benefits

- suitable for long-range tasks
- faster training and inferencing
- low computational cost
- less memory

Discretize SSMs

Continuous

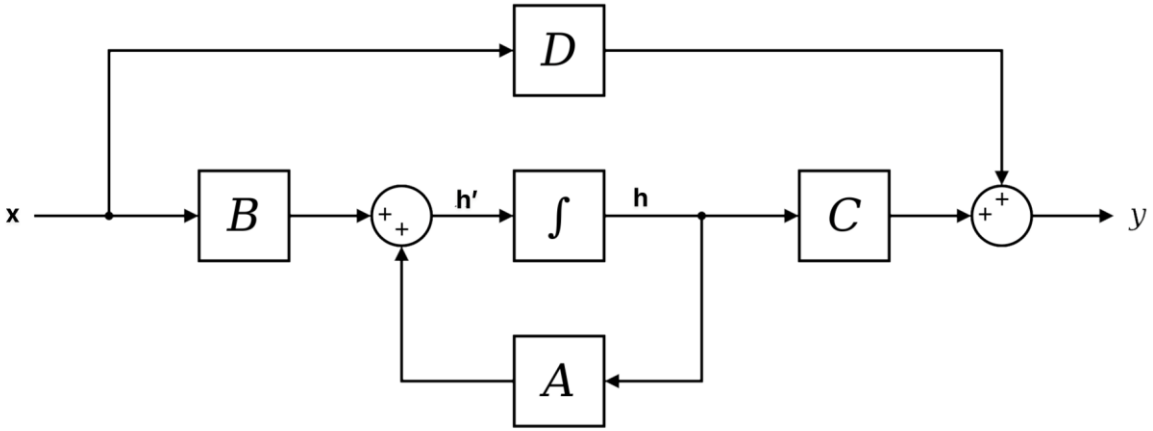
$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t)$$
$$y(t) = \mathbf{C}h(t) + \mathbf{D}x(t)$$

Discretize

Recurrent

$$h_t = \overline{\mathbf{A}}h_{t-1} + \overline{\mathbf{B}}x_t$$
$$y_t = \mathbf{C}h_t + \mathbf{D}x_t$$

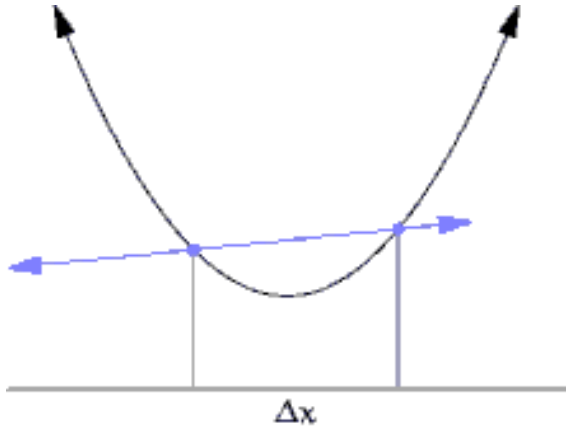
$$\overline{\mathbf{A}}, \overline{\mathbf{B}} = \text{Bilinear}(\mathbf{A}, \mathbf{B}, \Delta) \quad \text{hoặc} \quad \overline{\mathbf{A}}, \overline{\mathbf{B}} = \text{ZOH}(\mathbf{A}, \mathbf{B}, \Delta)$$



Discretize SSMs

1

$$h'(t) = \lim_{\Delta \to 0} \frac{h(t+\Delta) - h(t)}{\Delta}$$



$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t)$$

when Δ is small

2

$$h'(t) \approx \frac{h(t+\Delta) - h(t)}{\Delta}$$
$$h'(t)\Delta \approx h(t+\Delta) - h(t)$$
$$h(t+\Delta) \approx h'(t)\Delta + h(t).$$

3

$$h_{t+\Delta} \approx (\mathbf{A}h_t + \mathbf{B}x_t)\Delta + h_t$$
$$h_{t+\Delta} \approx \Delta\mathbf{A}h_t + \Delta\mathbf{B}x_t + h_t$$
$$h_{t+\Delta} \approx (\Delta\mathbf{A} + \mathbf{I})h_t + \Delta\mathbf{B}x_t$$
$$h_{t+\Delta} \approx \overline{\mathbf{A}}h_t + \overline{\mathbf{B}}x_t,$$

Example of Recurrent Representation

Recurrent Representation

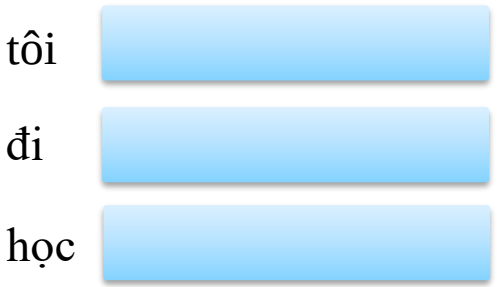
$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$

$$\overline{\mathbf{A}}, \overline{\mathbf{B}} = \text{Bilinear}(\mathbf{A}, \mathbf{B}, \Delta) \quad \text{hoặc} \quad \overline{\mathbf{A}}, \overline{\mathbf{B}} = \text{ZOH}(\mathbf{A}, \mathbf{B}, \Delta)$$

$$(\textbf{ZOH}) \quad \overline{\mathbf{A}} = \exp(\Delta \mathbf{A})$$
$$\overline{\mathbf{B}} = (\Delta \mathbf{A})^{-1}(\exp(\Delta \cdot \mathbf{A}) - \mathbf{I}) \cdot \Delta \mathbf{B}.$$

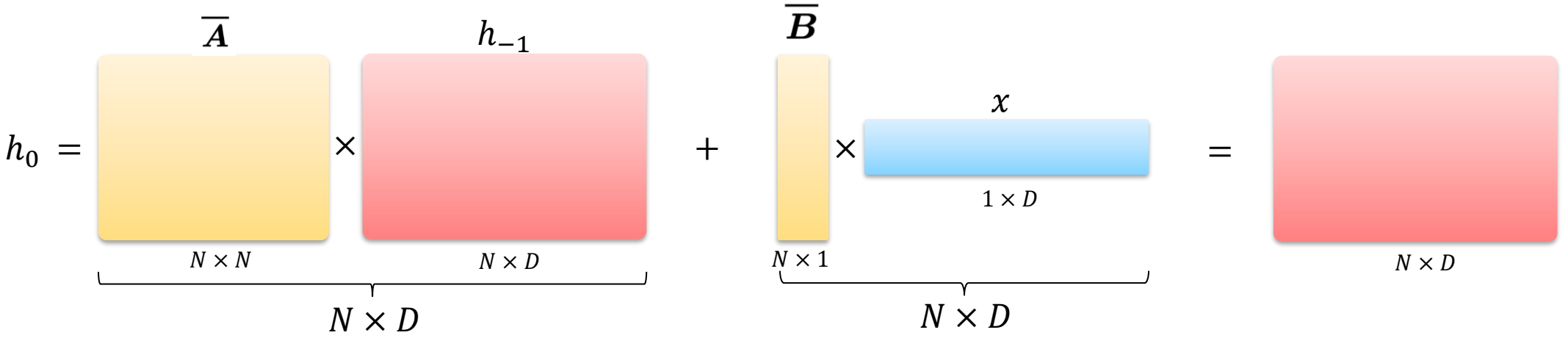
$$(\textbf{Bilinear}) \quad \overline{\mathbf{A}} = (\mathbf{I} - \Delta/2\mathbf{A})^{-1}(\mathbf{I} + \Delta/2\mathbf{A})$$
$$\overline{\mathbf{B}} = (\mathbf{I} - \Delta/2\mathbf{A})^{-1} \cdot \Delta \mathbf{B}$$

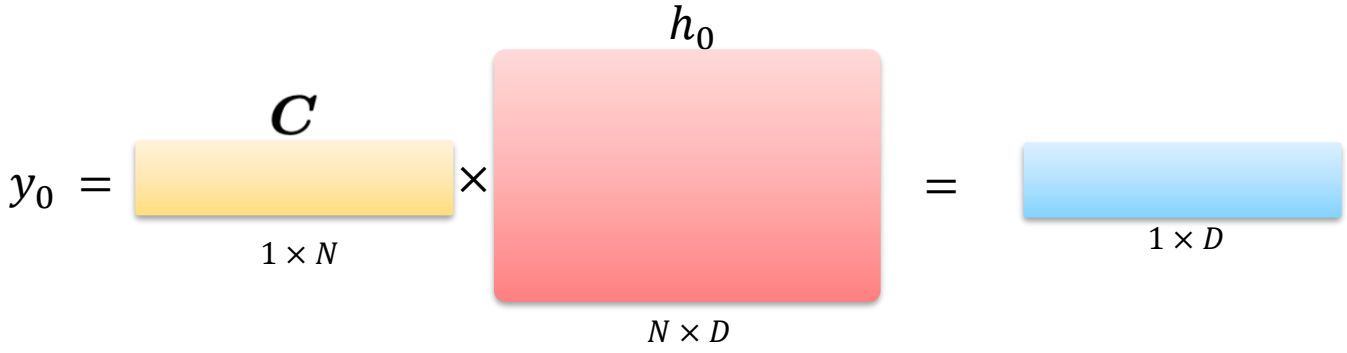
Example of Recurrent Representation



L: sequence len
D: hidden state
N: SSM dimension

$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$





Example of Recurrent Representation

tôi	1	2	1	4
đi	5	1	7	3
học	4	3	2	4

input: 3×4

L: sequence len
D: hidden state
N: SSM dimension

$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$

A

0.5	1	6
1	2	3
2	5	4

3×3

B

1
5
3

3×1

Δ 1

C

1	3	5

1×3

(ZOH)
$$\overline{\mathbf{A}} = \exp(\Delta \mathbf{A})$$
$$\overline{\mathbf{B}} = (\Delta \mathbf{A})^{-1}(\exp(\Delta \cdot \mathbf{A}) - \mathbf{I}) \cdot \Delta \mathbf{B}.$$

(Bilinear)
$$\overline{\mathbf{A}} = (\mathbf{I} - \Delta/2\mathbf{A})^{-1}(\mathbf{I} + \Delta/2\mathbf{A})$$
$$\overline{\mathbf{B}} = (\mathbf{I} - \Delta/2\mathbf{A})^{-1} \cdot \Delta \mathbf{B}$$

$\overline{A}$

1	2	1
2	3	2
3	4	1

3×3

$\overline{B}$

2
3
1

3×1

C

1	3	5

1×3

Example of Recurrent Representation

tôi	1	2	1	4
đi	5	1	7	3
học	4	3	2	4

input: 3×4

L: sequence len
D: hidden state
N: SSM dimension

$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$

$$h_0 = \underbrace{\underbrace{\overline{A}}_{\begin{bmatrix}1&2&1\\2&3&2\\3&4&1\end{bmatrix}_{3\times3}} \times \underbrace{h_{-1}}_{\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}_{3\times4}}}_{3\times4} + \underbrace{\underbrace{\overline{B}}_{\begin{bmatrix}2\\3\\1\end{bmatrix}_{3\times1}} \times \underbrace{x_0\ (tôi)}_{\begin{bmatrix}1&2&1&4\end{bmatrix}_{1\times4}}}_{3\times4} = \underbrace{\begin{bmatrix}2&4&2&8\\3&6&3&12\\1&2&1&4\end{bmatrix}}_{3\times4}$$

$$y_0 = \underbrace{C}_{\begin{bmatrix}1&3&5\end{bmatrix}_{1\times3}} \times \underbrace{h_0}_{\begin{bmatrix}2&4&2&8\\3&6&3&12\\1&2&1&4\end{bmatrix}_{3\times4}} = \underbrace{\begin{bmatrix}16&32&16&64\end{bmatrix}}_{1\times4}$$

AI VIET NAM
@aivietnam.edu.vn

Example of Recurrent Representation

tôi	1	2	1	4
đi	5	1	7	3
học	4	3	2	4

input: 3×4

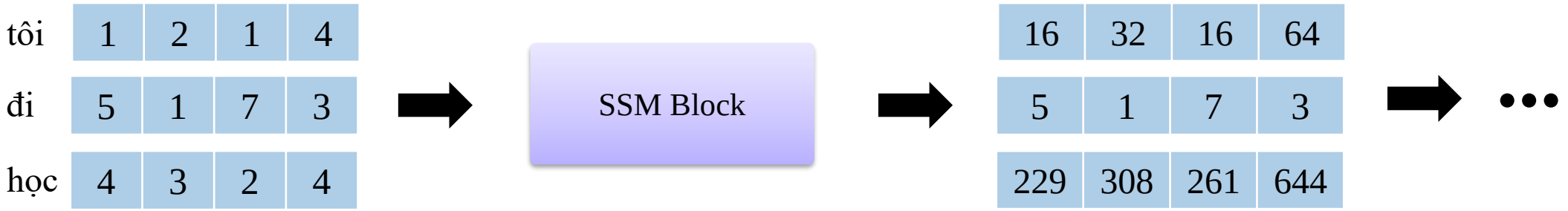
L: sequence len
D: hidden state
N: SSM dimension

$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$

$$h_1 = \underbrace{\overbrace{\begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 3 & 4 & 1 \end{bmatrix}}^{\overline{A}\ (3\times 3)} \times \overbrace{\begin{bmatrix} 2 & 4 & 2 & 8 \\ 3 & 6 & 3 & 12 \\ 1 & 2 & 1 & 4 \end{bmatrix}}^{h_0\ (3\times 4)}}_{3\times 4} + \underbrace{\overbrace{\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}}^{\overline{B}\ (3\times 1)} \times \overbrace{\begin{bmatrix} 5 & 1 & 7 & 3 \end{bmatrix}}^{x_1\ (đi)\ (1\times 4)}}_{3\times 4} = \underbrace{\begin{bmatrix} 19 & 20 & 23 & 42 \\ 30 & 31 & 36 & 69 \\ 24 & 39 & 26 & 79 \end{bmatrix}}_{3\times 4}$$

$$y_1 = \overbrace{\begin{bmatrix} 1 & 3 & 5 \end{bmatrix}}^{C\ (1\times 3)} \times \overbrace{\begin{bmatrix} 19 & 20 & 23 & 42 \\ 30 & 31 & 36 & 69 \\ 24 & 39 & 26 & 79 \end{bmatrix}}^{h_1\ (3\times 4)} = \underbrace{\begin{bmatrix} 229 & 308 & 261 & 644 \end{bmatrix}}_{1\times 4}$$

Example of Recurrent Representation



Discretize SSMs

Recurrent Representation

$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$

$$\overline{\mathbf{A}}, \overline{\mathbf{B}} = \text{Bilinear}(\mathbf{A}, \mathbf{B}, \Delta) \quad \text{hoặc} \quad \overline{\mathbf{A}}, \overline{\mathbf{B}} = \text{ZOH}(\mathbf{A}, \mathbf{B}, \Delta)$$

$$\textbf{(ZOH)} \quad \overline{\mathbf{A}} = \exp(\Delta \mathbf{A})$$
$$\overline{\mathbf{B}} = (\Delta \mathbf{A})^{-1}(\exp(\Delta \cdot \mathbf{A}) - \mathbf{I}) \cdot \Delta \mathbf{B}.$$

$$\textbf{(Bilinear)} \quad \overline{\mathbf{A}} = (\mathbf{I} - \Delta/2\mathbf{A})^{-1}(\mathbf{I} + \Delta/2\mathbf{A})$$
$$\overline{\mathbf{B}} = (\mathbf{I} - \Delta/2\mathbf{A})^{-1} \cdot \Delta \mathbf{B}$$

Learnable Params

- A, B, C, D, Δ

	Ideal Model
Parallelizable	yes
Training Stage	$O(n)$
Inference Stage	$O(n)$
Inference per Token	$O(1)$

Convolutional Representation of SSMs

$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$

t=0

$$h_0 = \overline{\mathbf{B}} x_0$$
$$y_0 = \mathbf{C} h_0 = \mathbf{C}\overline{\mathbf{B}} x_0$$

t=1

$$h_1 = \overline{\mathbf{A}} h_0 + \overline{\mathbf{B}} x_1 = \overline{\mathbf{AB}} x_0 + \overline{\mathbf{B}} x_1$$
$$y_1 = \mathbf{C} h_1 = \mathbf{C}(\overline{\mathbf{AB}} x_0 + \overline{\mathbf{B}} x_1) = \mathbf{C}\overline{\mathbf{AB}} x_0 + \mathbf{C}\overline{\mathbf{B}} x_1$$

t=2

$$h_2 = \overline{\mathbf{A}} h_1 + \overline{\mathbf{B}} x_2 = \overline{\mathbf{A}}(\overline{\mathbf{AB}} x_0 + \overline{\mathbf{B}} x_1) + \overline{\mathbf{B}} x_2 = \overline{\mathbf{A}}^2 \overline{\mathbf{B}} x_0 + \overline{\mathbf{AB}} x_1 + \overline{\mathbf{B}} x_2$$
$$y_2 = \mathbf{C} h_2 = \mathbf{C}(\overline{\mathbf{A}}^2 \overline{\mathbf{B}} x_0 + \overline{\mathbf{AB}} x_1 + \overline{\mathbf{B}} x_2) = \mathbf{C}\overline{\mathbf{A}}^2 \overline{\mathbf{B}} x_0 + \mathbf{C}\overline{\mathbf{AB}} x_1 + \mathbf{C}\overline{\mathbf{B}} x_2$$

... ...

t=k

$$y_k = \mathbf{C}\overline{\mathbf{A}}^k \overline{\mathbf{B}} x_0 + \mathbf{C}\overline{\mathbf{A}}^{k-1} \overline{\mathbf{B}} x_1 + \ldots + \mathbf{C}\overline{\mathbf{AB}} x_{k-1} + \mathbf{C}\overline{\mathbf{B}} x_k$$

Convolutional Representation of SSMs

t=k

$$y_k = \mathbf{C}\overline{\mathbf{A}}^k\overline{\mathbf{B}}x_0 + \mathbf{C}\overline{\mathbf{A}}^{k-1}\overline{\mathbf{B}}x_1 + \ldots + \mathbf{C}\overline{\mathbf{A}\mathbf{B}}x_{k-1} + \mathbf{C}\overline{\mathbf{B}}x_k$$

$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}),$$
$$x = (x_0, x_1, \ldots, x_{L-1}),$$
$$y = \overline{\mathbf{K}} * x.$$

Example of Convolutional Representation

tôi
đi
học

input: $L \times D$

L: sequence len
D: hidden state
N: SSM dimension

$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}),$$
$$x = (x_0, x_1, \ldots, x_{L-1}),$$
$$y = \overline{\mathbf{K}} * x.$$

	$C\overline{A}^2\overline{B}$	$C\overline{A}\overline{B}$	$C\overline{B}$

Kernel

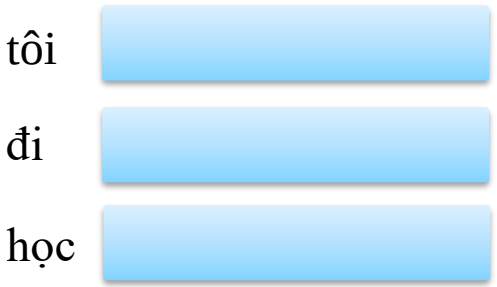
Padding

0	**0**	x_0	x_1	x_2

Input

Output

Example of Convolutional Representation

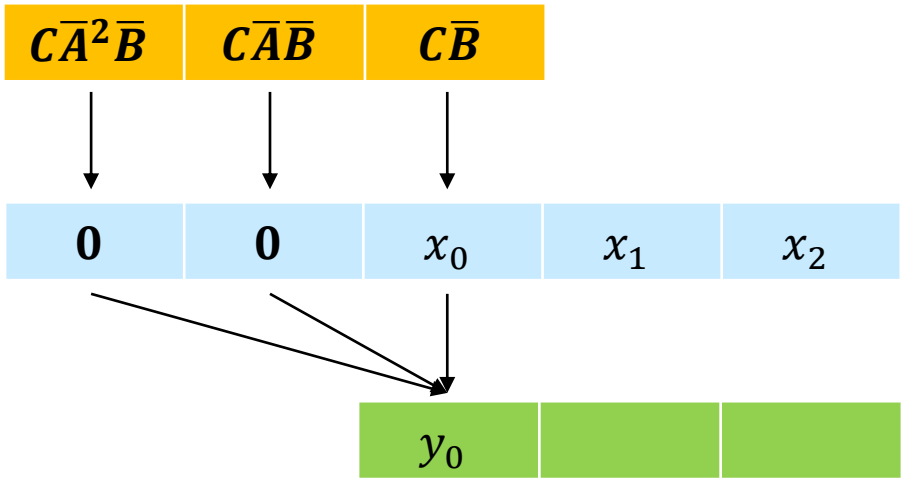


input: $L \times D$

L: sequence len
D: hidden state
N: SSM dimension

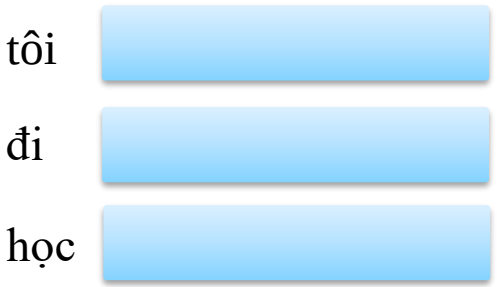
$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}),$$
$$x = (x_0, x_1, \ldots, x_{L-1}),$$
$$y = \overline{\mathbf{K}} * x.$$

Step 1



$$y_0 = C\bar{B}x_0$$

Example of Convolutional Representation

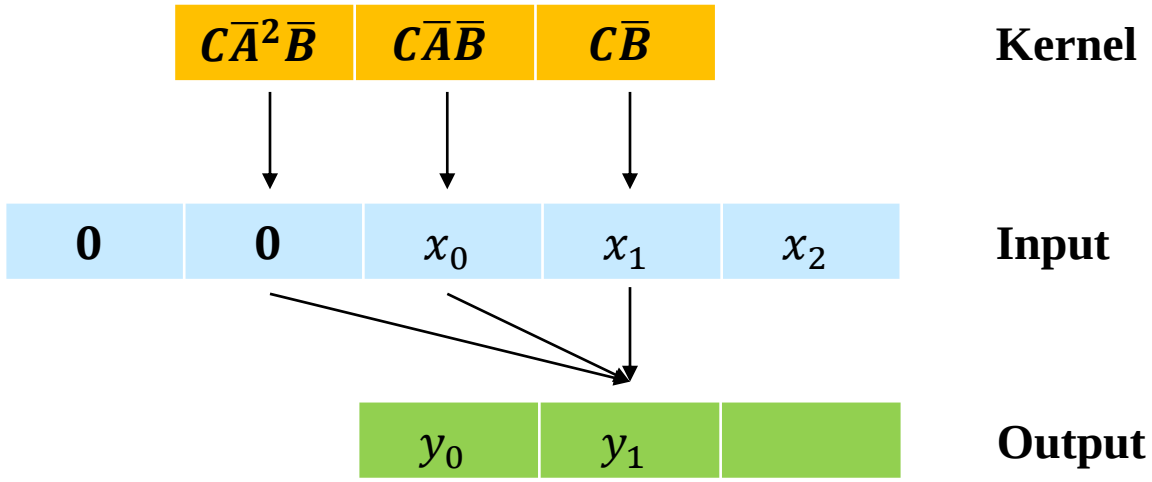


input: $L \times D$

L: sequence len
D: hidden state
N: SSM dimension

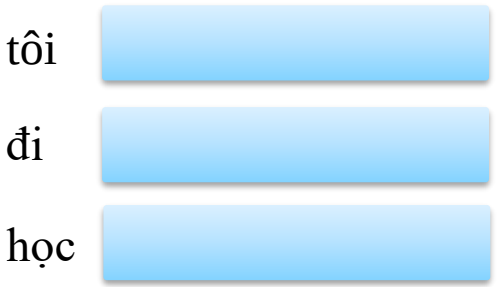
$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}),$$
$$x = (x_0, x_1, \ldots, x_{L-1}),$$
$$y = \overline{\mathbf{K}} * x.$$

Step 2



$$y_1 = C\overline{AB}x_0 + C\bar{B}x_1$$

Example of Convolutional Representation

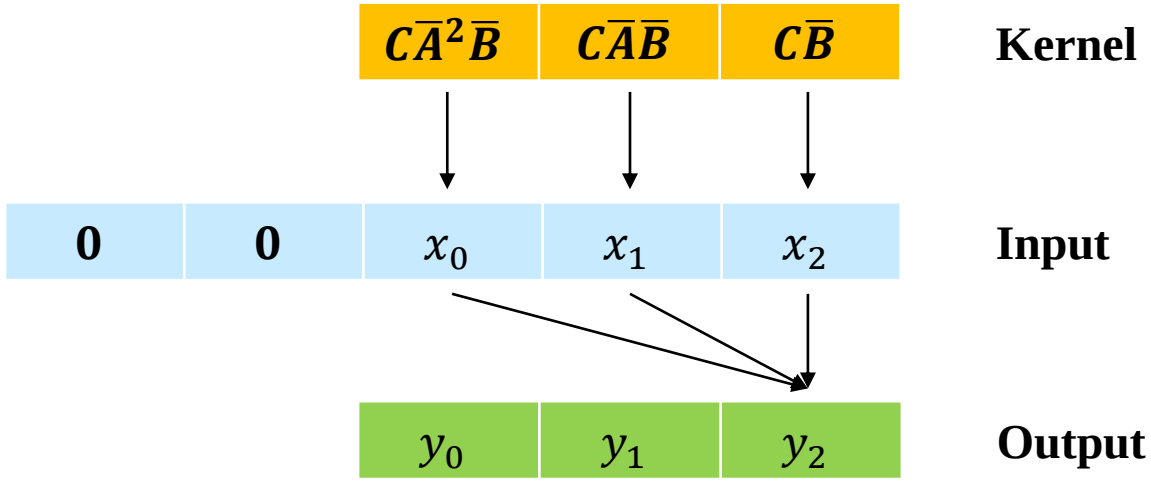


input: $L \times D$

L: sequence len
D: hidden state
N: SSM dimension

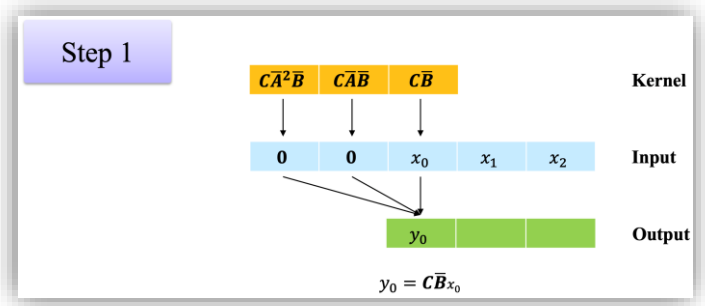
$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}),$$
$$x = (x_0, x_1, \ldots, x_{L-1}),$$
$$y = \overline{\mathbf{K}} * x.$$

Step 3

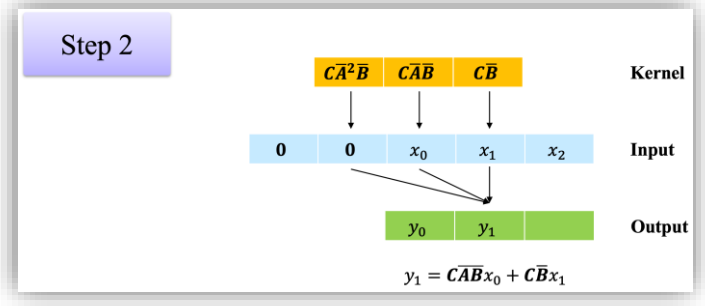


$$y_2 = C\bar{A}^2\bar{B}x_0 + C\bar{A}\bar{B}x_1 + C\bar{B}x_2$$

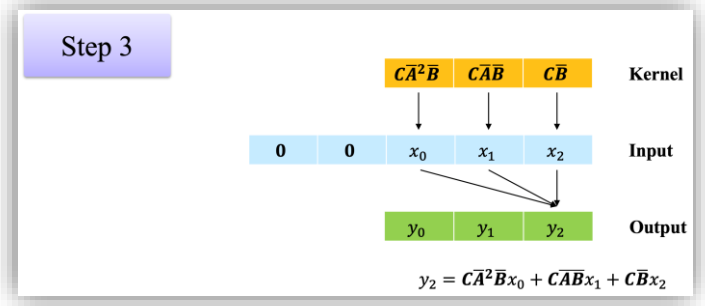
Example of Convolutional Representation



worker 1 → y_0



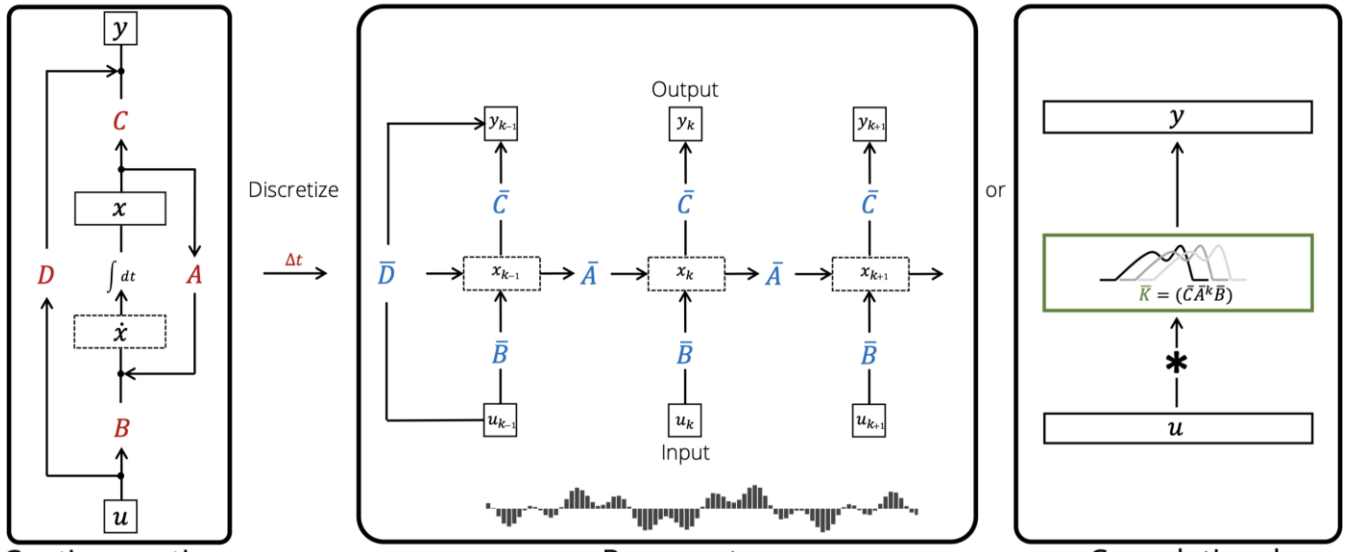
worker 2 → y_1



worker 3 → y_2

Can be computed in parallel!

SSMs is Ideal Models



✓ *continuous data*
✓ *irregular sampling*

✓ *unbounded context*
✓ *efficient inference*

✓ *local information*
✓ *parallelizable training*

Ideal Model	
Parallelizable	yes
Training Stage	$O(n)$
Inference Stage	$O(n)$
Inference per Token	$O(1)$

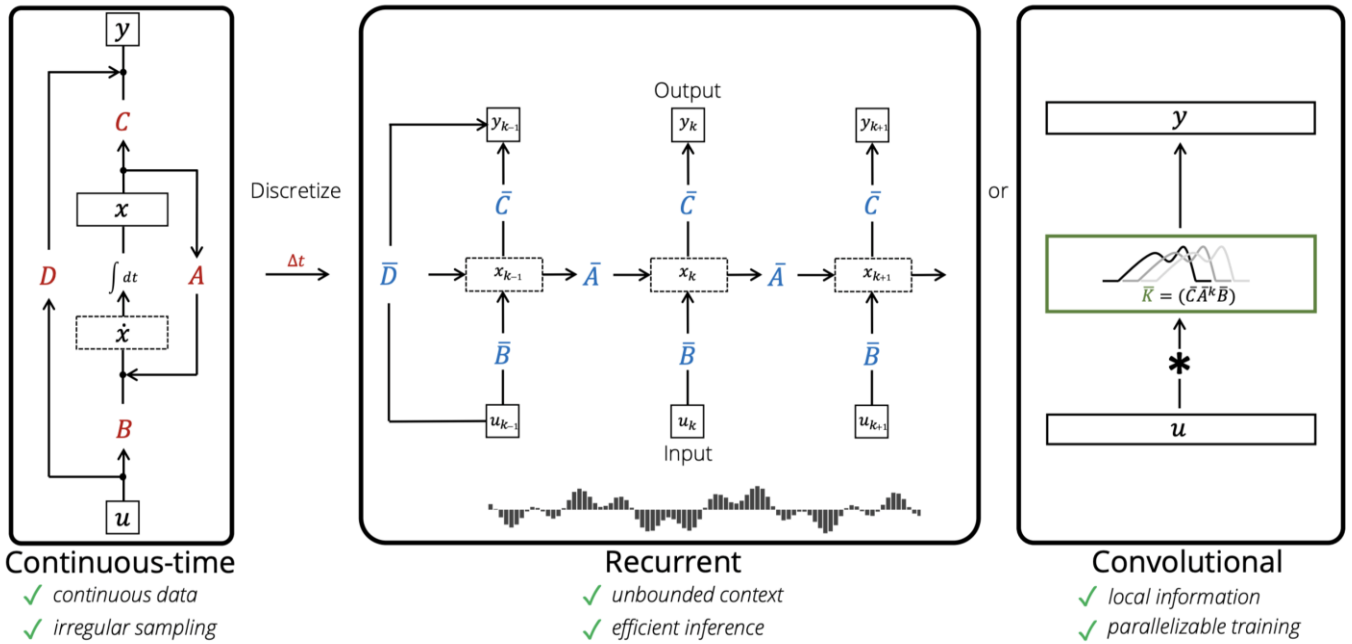
Recurrent

$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$

Convolutional

$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}),$$
$$x = (x_0, x_1, \ldots, x_{L-1}),$$
$$y = \overline{\mathbf{K}} * x.$$

SSMs is Ideal Models



Recurrent

$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$

Convolutional

$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}),$$
$$x = (x_0, x_1, \ldots, x_{L-1}),$$
$$y = \overline{\mathbf{K}} * x.$$

The importance of the A matrix

SSMs still have some of the same issues as RNNs, like vanishing/exploding gradients.

solution

(**HiPPO Matrix**) $$A_{nk} = -\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

The performance in the Sequencial MNIST increased from 60% to 98%.

Recurrent

$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$

Convolutional

$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}),$$
$$x = (x_0, x_1, \ldots, x_{L-1}),$$
$$y = \overline{\mathbf{K}} * x.$$

AI VIET NAM

@aivietnam.edu.vn

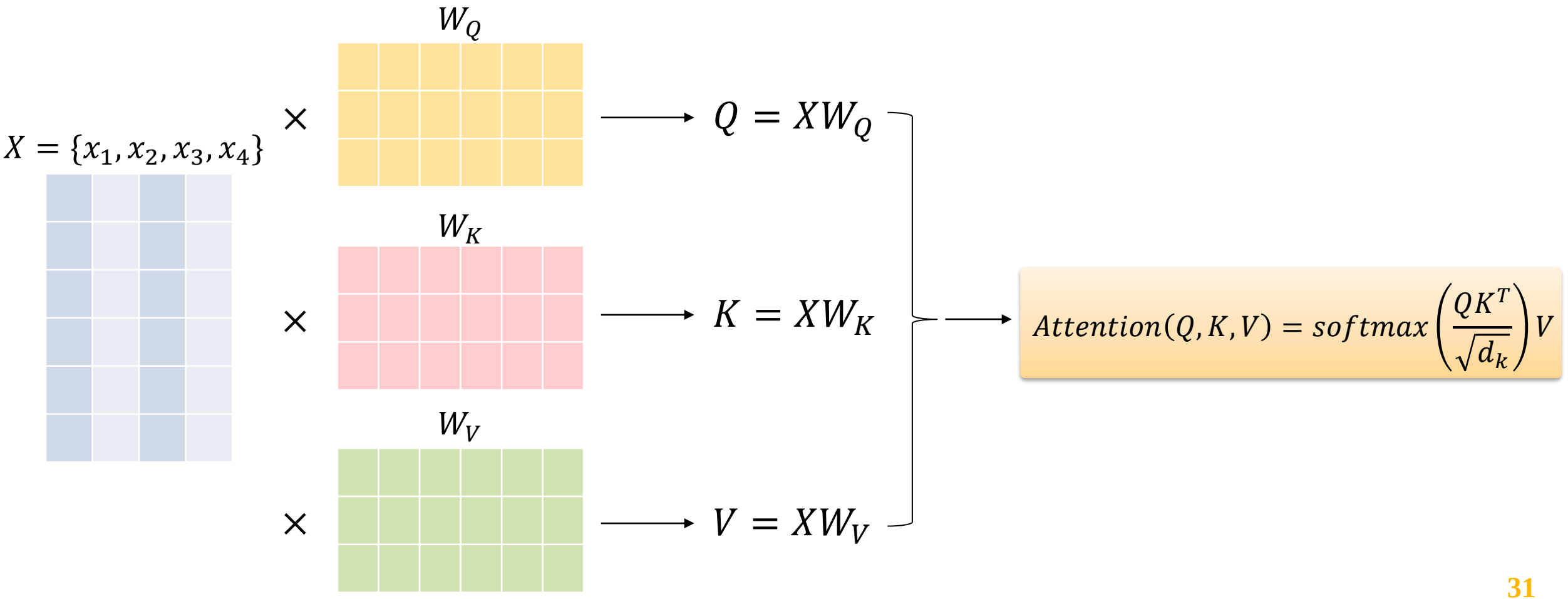
QUIZ! TIME

Outline

1. Motivation
2. State Space Models
3. **Mamba**
4. Coding

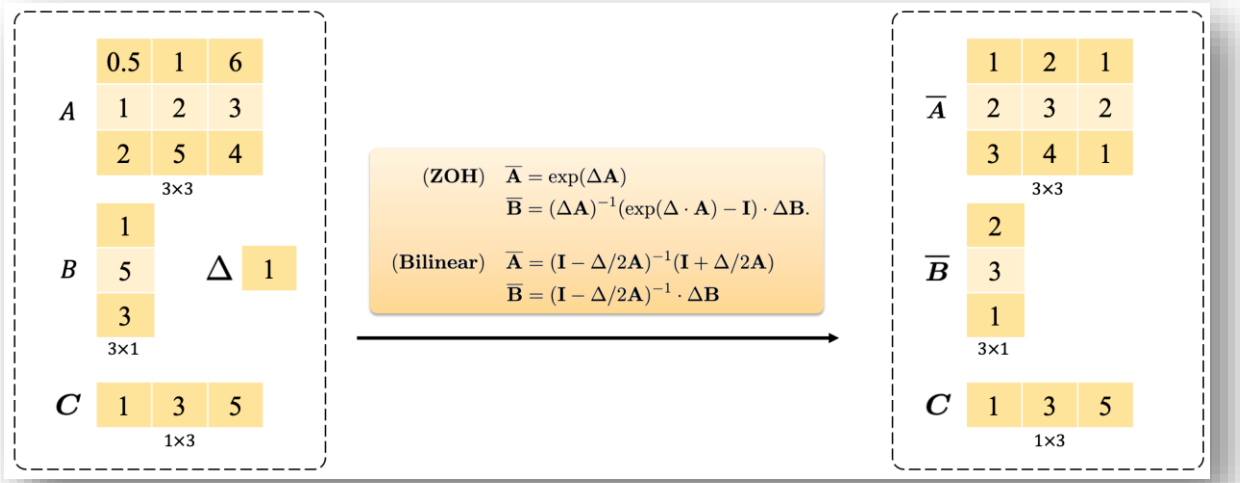
Mamba Motivation

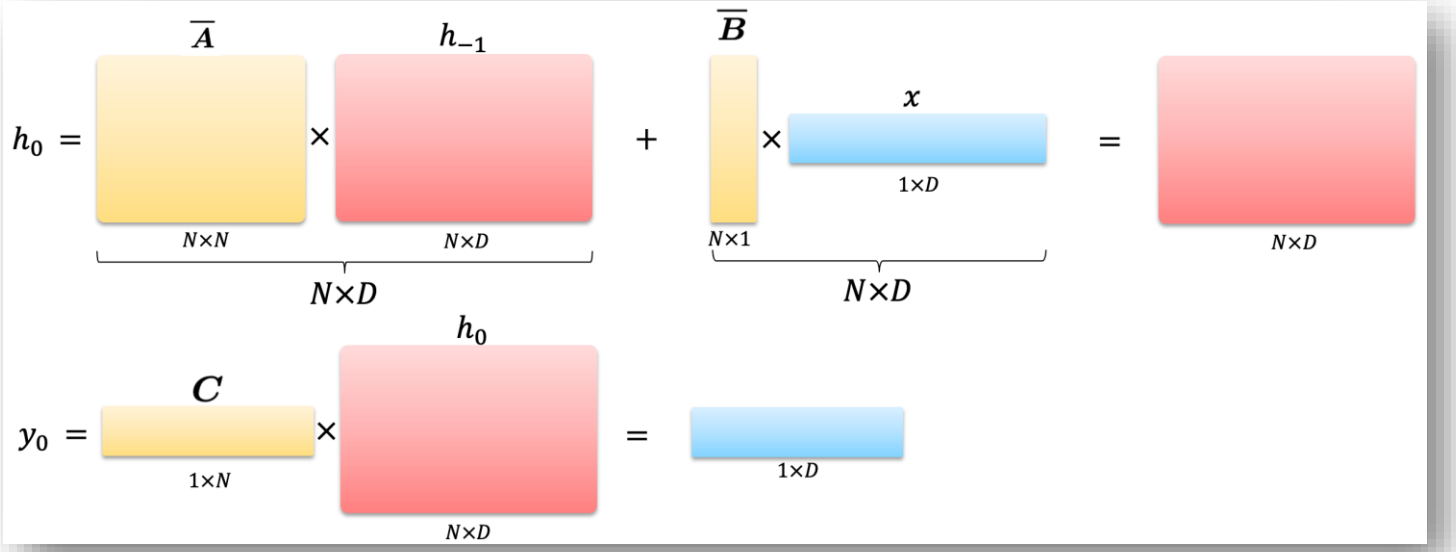
A step back: Attention in Transformer



Mamba Motivation

A step back: S4





Mamba Contribution #1: Selection Mechanism

A

0.5	1	6
1	2	3
2	5	4

$B = Linear_B$

$C = Linear_C$

$\Delta = Softplus(\Delta + Linear_\Delta(\Delta))$

(ZOH)

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

(Bilinear)

$$\overline{\mathbf{A}} = (\mathbf{I} - \Delta/2\mathbf{A})^{-1}(\mathbf{I} + \Delta/2\mathbf{A})$$
$$\overline{\mathbf{B}} = (\mathbf{I} - \Delta/2\mathbf{A})^{-1} \cdot \Delta\mathbf{B}$$

$\overline{A}$

1	2	1
2	3	2
3	4	1

3×3

$\overline{B}$

2
3
1

3×1

C

1	3	5

1×3

Mamba Contribution #1: Selection Mechanism

Compare S4 and Mamba (S6)

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: $\Delta : (\mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter})$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

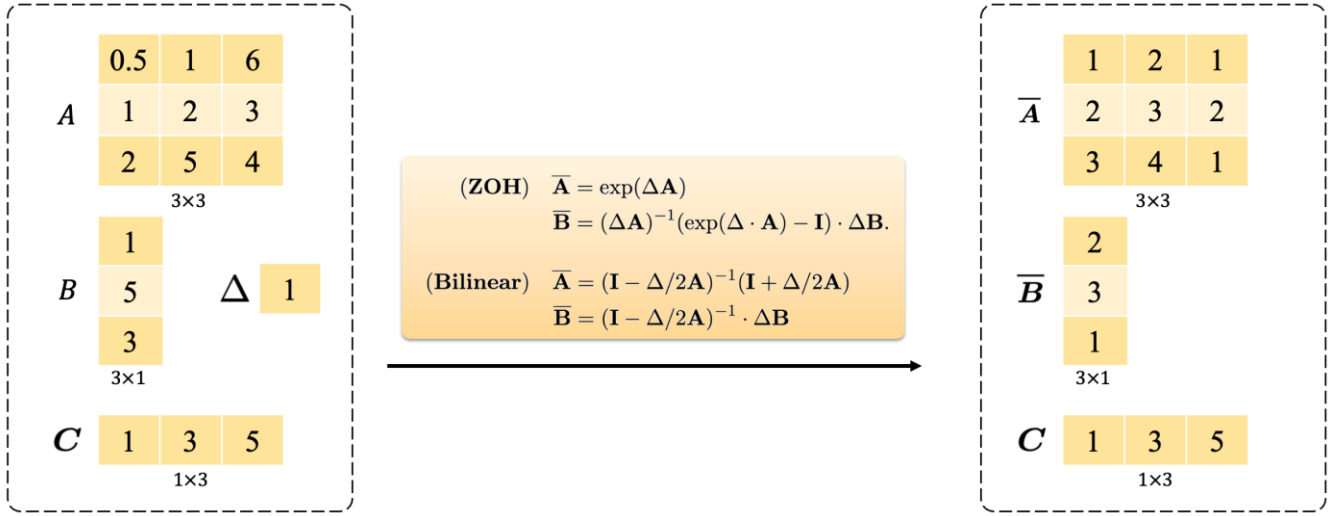
Algorithm 2 SSM + Selection (S6)

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: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

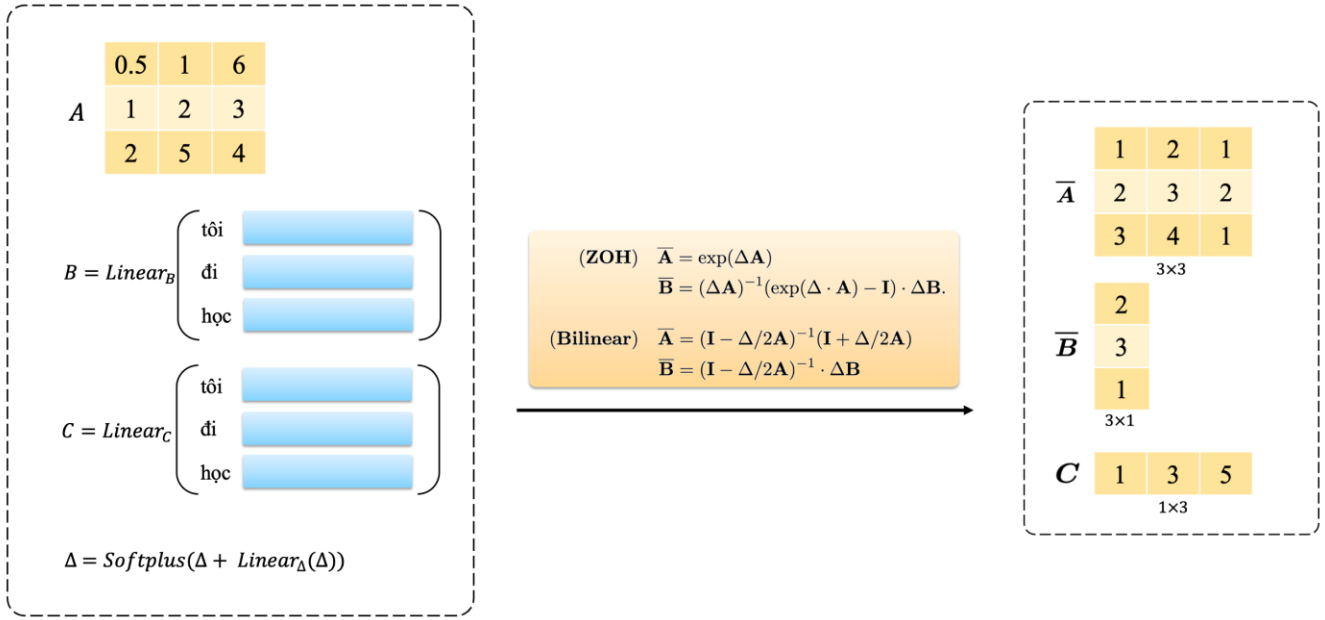
AI VIET NAM
@aivietnam.edu.vn

Mamba Contribution #1: Selection Mechanism

S4



Mamba



Mamba Contribution #1: Selection Mechanism

Mamba becomes time-varying

t=0

$$h_0 = \overline{\mathbf{B}}x_0$$
$$y_0 = \mathbf{C}h_0 = \mathbf{C}\overline{\mathbf{B}}x_0$$

t=1

$$h_1 = \overline{\mathbf{A}}h_0 + \overline{\mathbf{B}}x_1 = \overline{\mathbf{AB}}x_0 + \overline{\mathbf{B}}x_1$$
$$y_1 = \mathbf{C}h_1 = \mathbf{C}(\overline{\mathbf{AB}}x_0 + \overline{\mathbf{B}}x_1) = \mathbf{C}\overline{\mathbf{AB}}x_0 + \mathbf{C}\overline{\mathbf{B}}x_1$$

t=2

$$h_2 = \overline{\mathbf{A}}h_1 + \overline{\mathbf{B}}x_2 = \overline{\mathbf{A}}(\overline{\mathbf{AB}}x_0 + \overline{\mathbf{B}}x_1) + \overline{\mathbf{B}}x_2 = \overline{\mathbf{A}}^2\overline{\mathbf{B}}x_0 + \overline{\mathbf{AB}}x_1 + \overline{\mathbf{B}}x_2$$
$$y_2 = \mathbf{C}h_2 = \mathbf{C}(\overline{\mathbf{A}}^2\overline{\mathbf{B}}x_0 + \overline{\mathbf{AB}}x_1 + \overline{\mathbf{B}}x_2) = \mathbf{C}\overline{\mathbf{A}}^2\overline{\mathbf{B}}x_0 + \mathbf{C}\overline{\mathbf{AB}}x_1 + \mathbf{C}\overline{\mathbf{B}}x_2$$

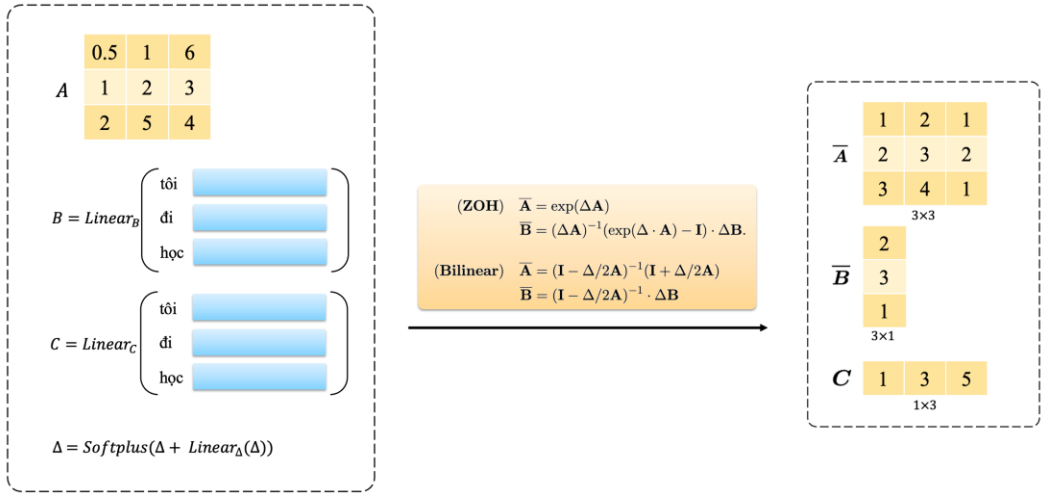
...

t=k

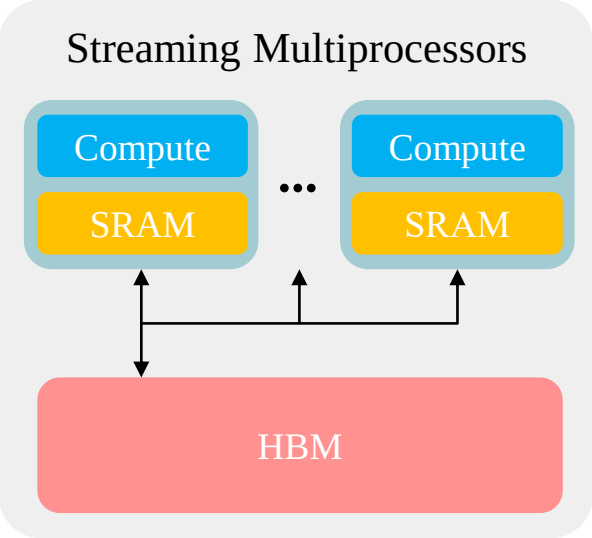
$$y_k = \mathbf{C}\overline{\mathbf{A}}^k\overline{\mathbf{B}}x_0 + \mathbf{C}\overline{\mathbf{A}}^{k-1}\overline{\mathbf{B}}x_1 + \ldots + \mathbf{C}\overline{\mathbf{AB}}x_{k-1} + \mathbf{C}\overline{\mathbf{B}}x_k$$

Convolutional

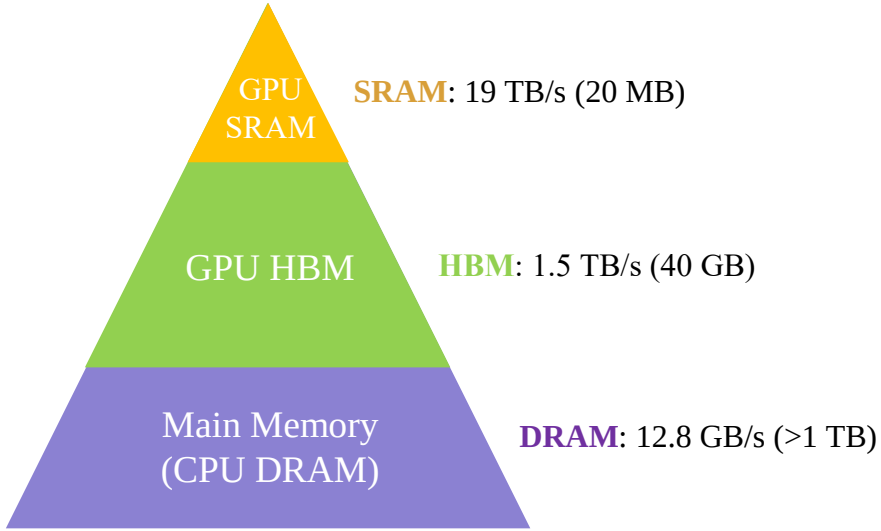
$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{AB}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}),$$
$$x = (x_0, x_1, \ldots, x_{L-1}),$$
$$y = \overline{\mathbf{K}} * x.$$



How does GPU work?



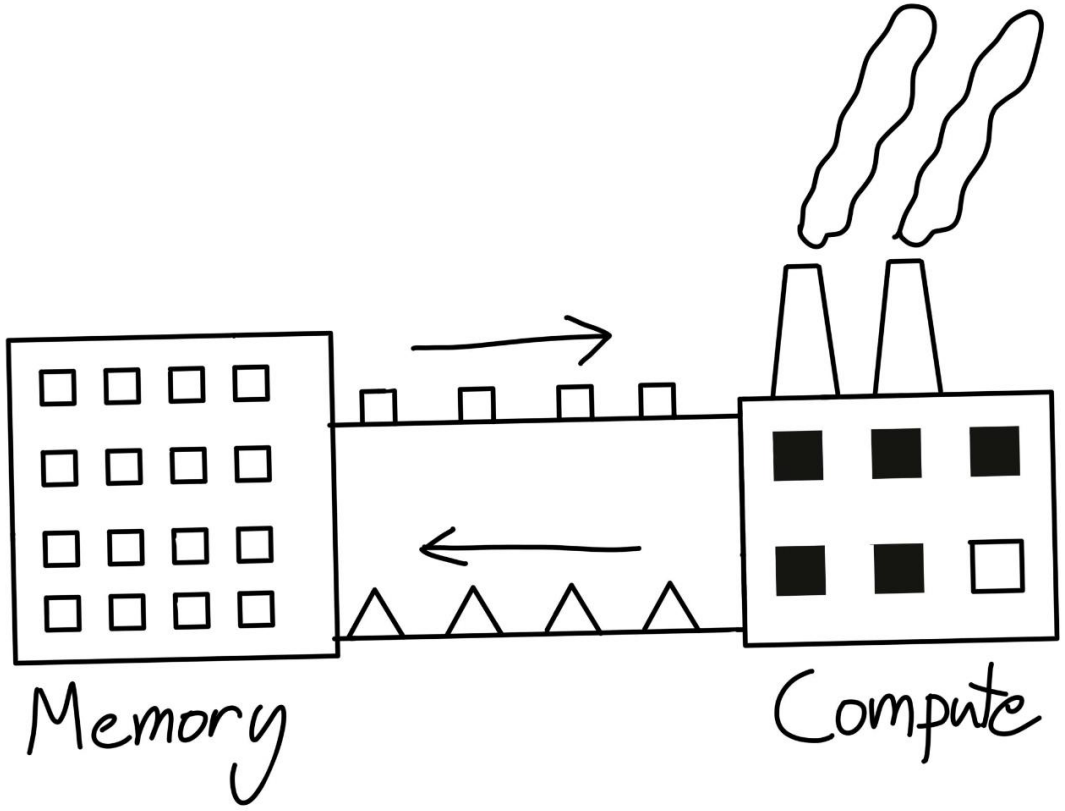
GPU

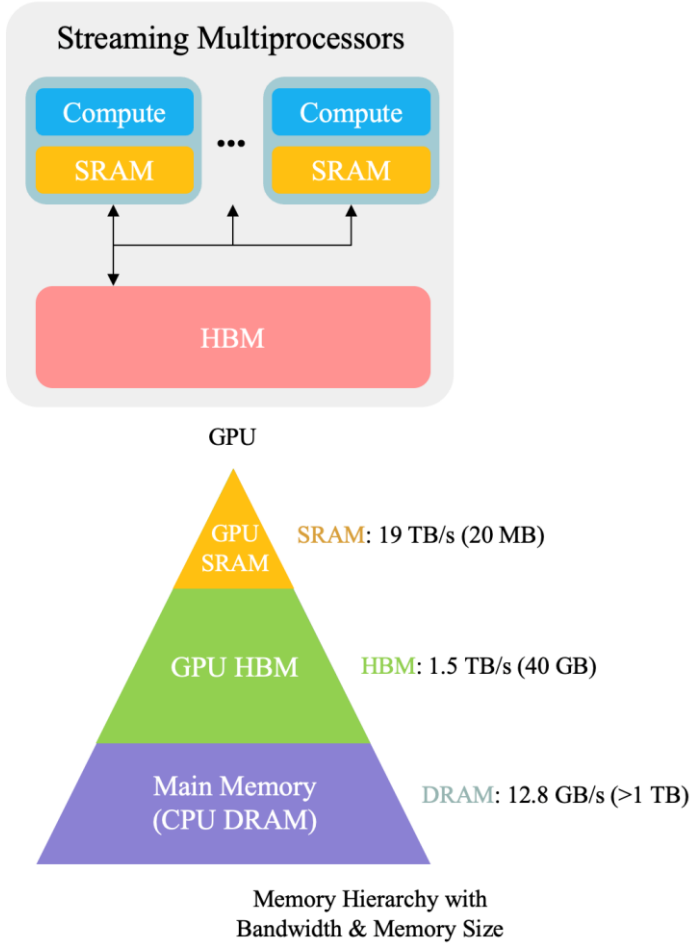


Memory Hierarchy with
Bandwidth & Memory Size

How does GPU work?

Memory & Compute



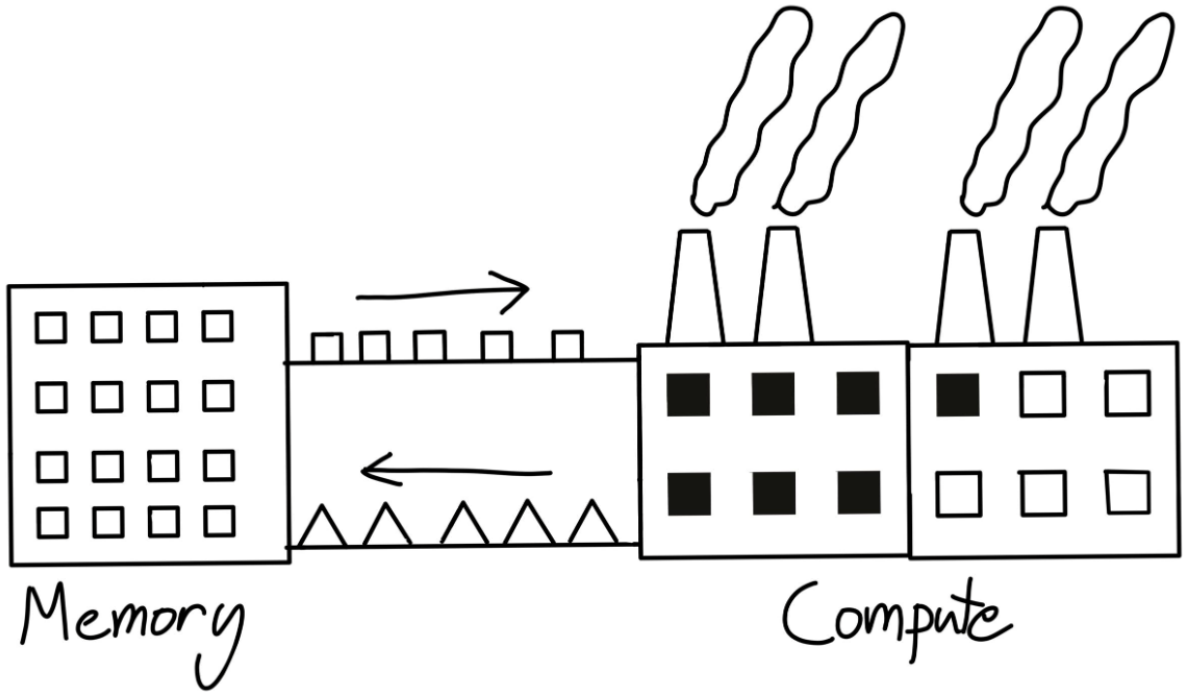


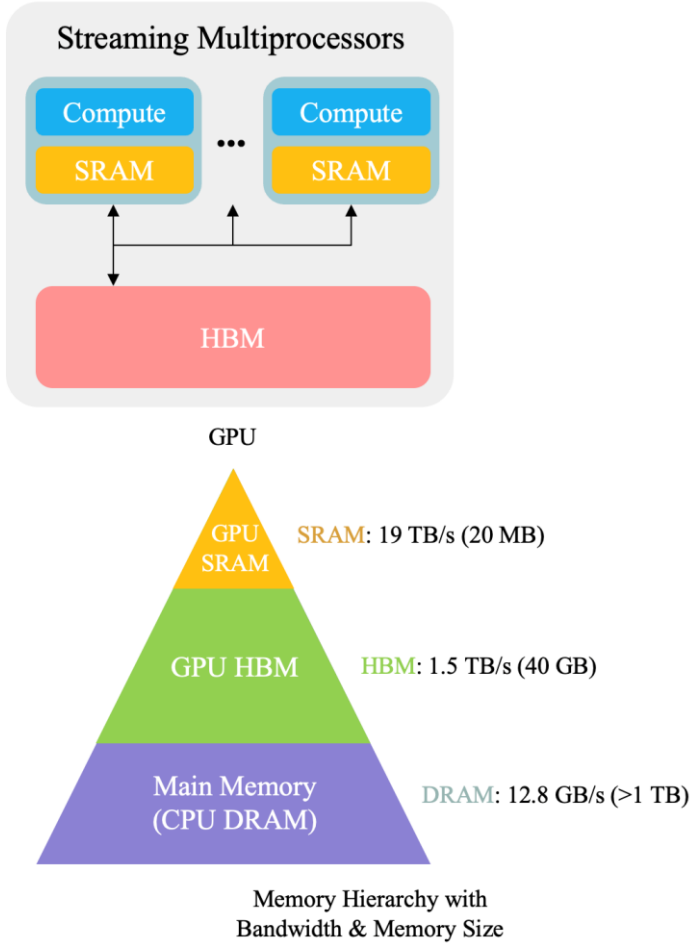
Memory Hierarchy with Bandwidth & Memory Size

Blog: Making Deep Learning Go Brrrr From First Principles

How does GPU work?

Memory & Compute





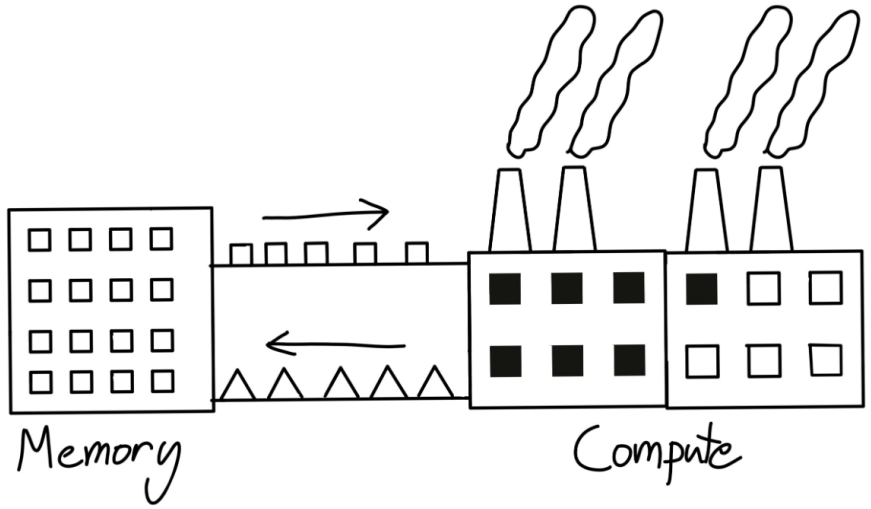
Memory Hierarchy with Bandwidth & Memory Size

Blog: Making Deep Learning Go Brrrr From First Principles

AI VIET NAM
@aivietnam.edu.vn

How does GPU work?

Memory & Compute

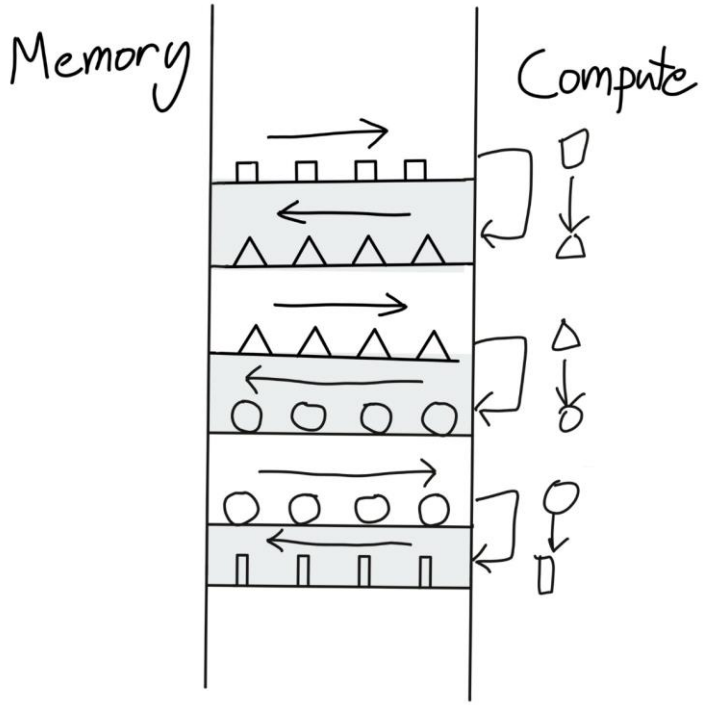


	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
INT8 Tensor Core	624 TOPS | 1248 TOPS*	
GPU Memory	80GB HBM2e	80GB HBM2e
GPU Memory Bandwidth	1,935 GB/s	2,039 GB/s
Max Thermal Design Power (TDP)	300W	400W ***

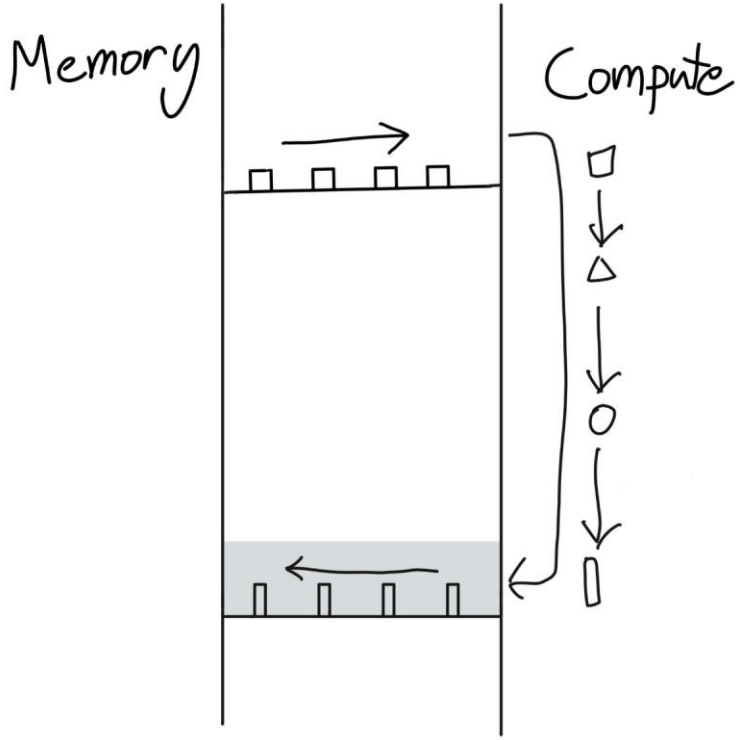
Blog: Making Deep Learning Go Brrrr From First Principles
NVIDIA A100 Tensor Core GPU

Mamba Contribution #2: Hardware-aware Algorithm

Kernel Fusion




```
x1 = x.cos()  # Read from x  in global memory, write to x1
x2 = x1.cos() # Read from x1 in global memory, write to x2
x3 = x2.cos() # Read from x2 in global memory, write to x3
```

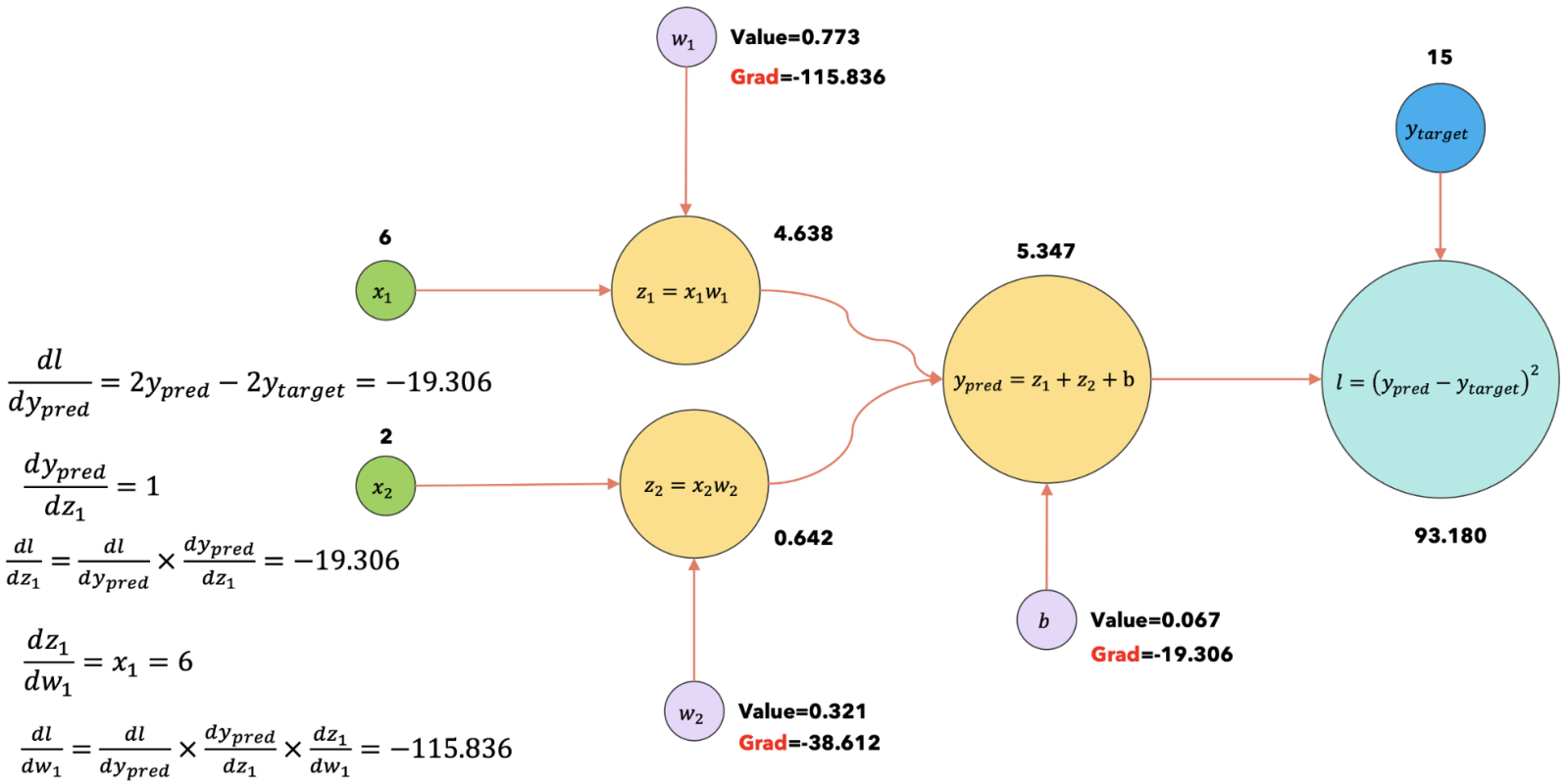



```
x3 = x.cos().cos().cos() # Read from x in global memory, write to x3
```


Blog: Making Deep Learning Go Brrrr From First Principles

Mamba Contribution #2: Hardware-aware Algorithm

Recomputation



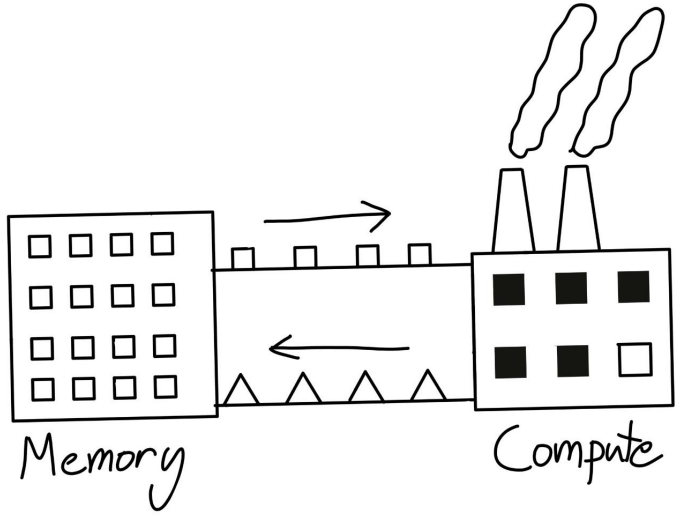
$$\frac{dl}{dy_{pred}} = 2y_{pred} - 2y_{target} = -19.306$$

$$\frac{dy_{pred}}{dz_1} = 1$$

$$\frac{dl}{dz_1} = \frac{dl}{dy_{pred}} \times \frac{dy_{pred}}{dz_1} = -19.306$$

$$\frac{dz_1}{dw_1} = x_1 = 6$$

$$\frac{dl}{dw_1} = \frac{dl}{dy_{pred}} \times \frac{dy_{pred}}{dz_1} \times \frac{dz_1}{dw_1} = -115.836$$



Mamba Contribution #2: Hardware-aware Algorithm

Parallel Scan

Prefix-sum

Initial array	9	6	7	10	8	7
Pre-fix sum	9	15	22	32	40	47

Parallel Scan

Model input	x_0	x_1	x_2	x_3	x_4	x_5
Scan output	h_0	h_1	h_2	h_3	h_4	h_5

Recurrent

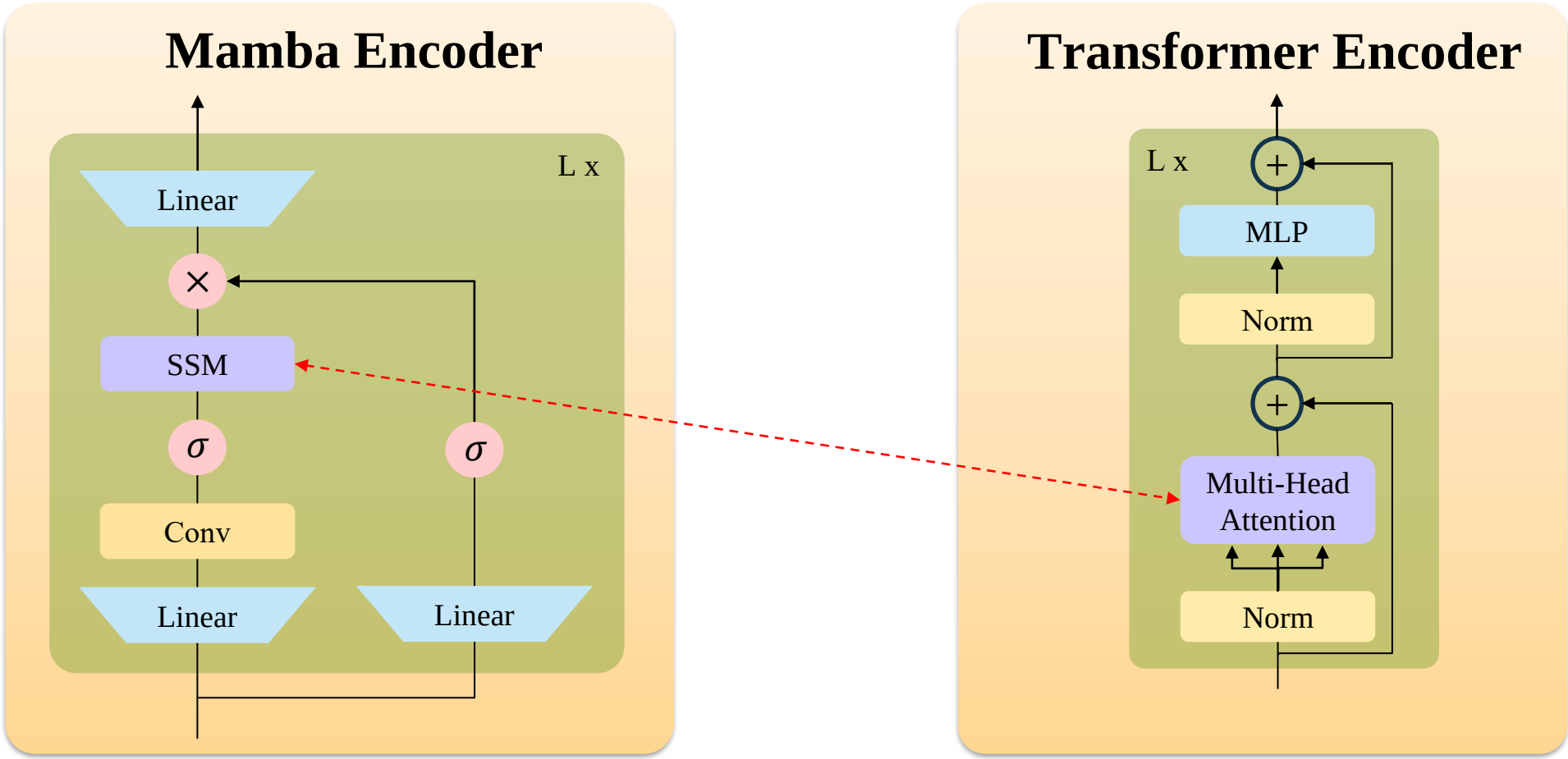
$$h_t = \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t$$
$$y_t = \mathbf{C} h_t + \mathbf{D} x_t$$

Mamba Contribution #2: Hardware-aware Algorithm

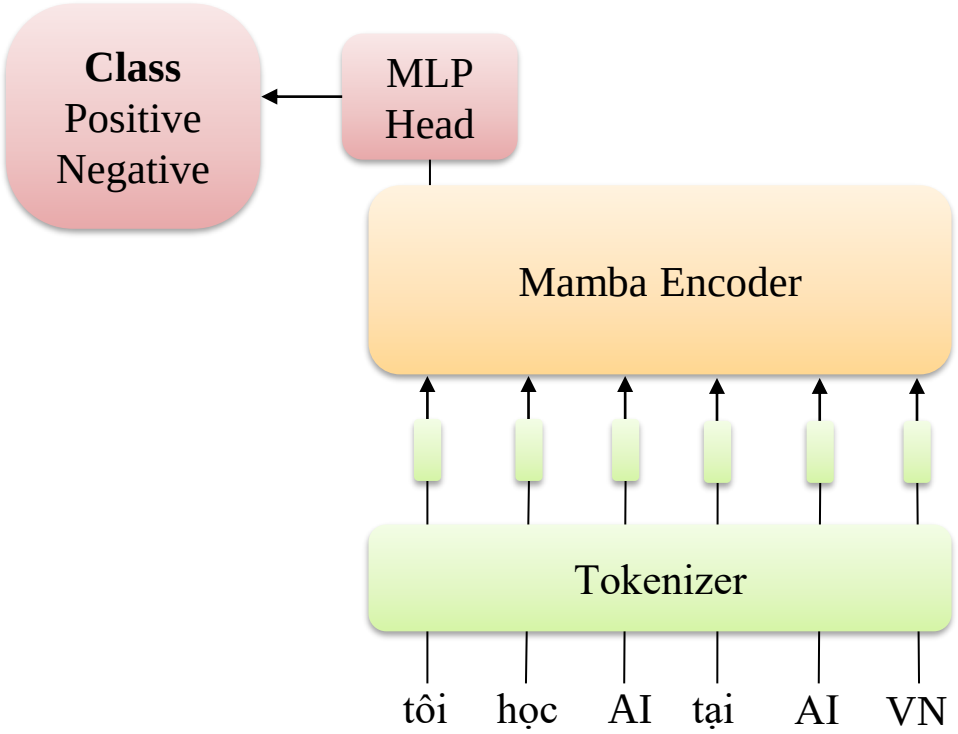
Parallel Scan

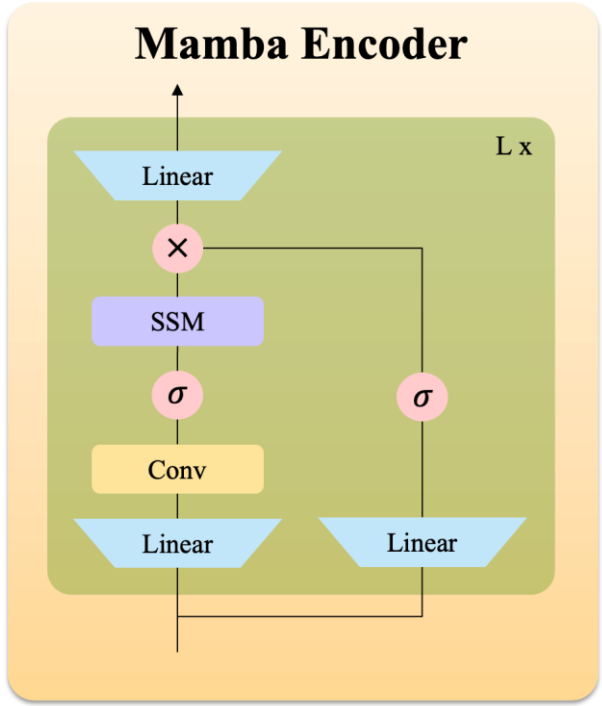
Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Sequential Prefix-Sum (what we expect as output from the parallel computation)	1	5	11	14	24	26	33	34	41	50	58	66	76	85	91	101
Array of values	1	4	6	3	10	2	7	1	7	9	8	8	10	9	6	10
		5		9		12		8		16		16		19		16
				14				20				32				35
								34								67
																101
												66				
						26				50				85		
			11		24		33		41		58		76		91	
Parallel Prefix-Sum (output of parallel computation)	1	5	11	14	24	26	33	34	41	50	58	66	76	85	91	101
Do the two computations match?	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES

Mamba Block


Mamba Encoder
L x
Linear
SSM
Conv
Linear
Linear
Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm

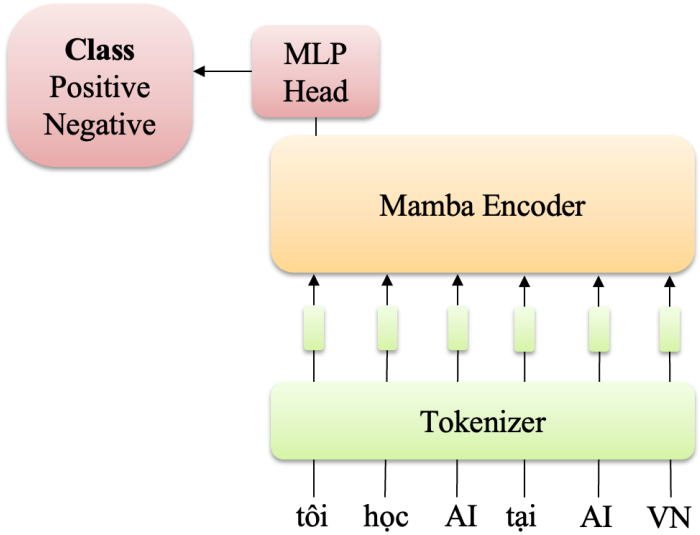
Mamba Architecture

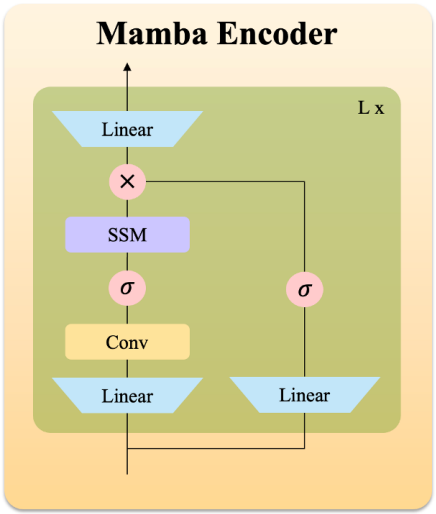




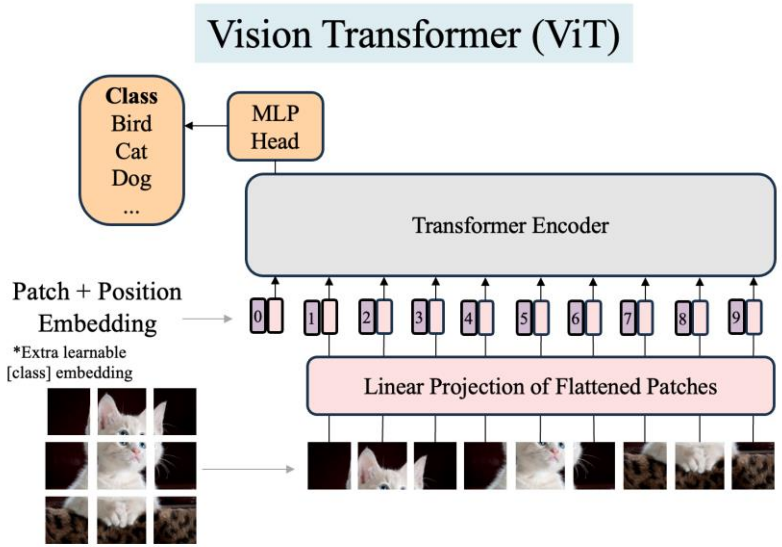
Mamba Arch. & Transformer Arch.

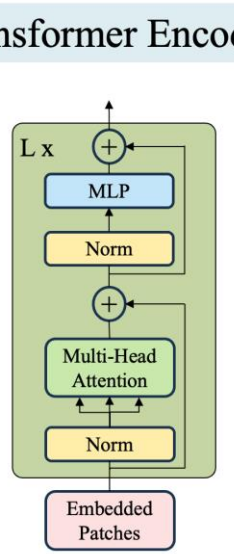
Mamba





ViT

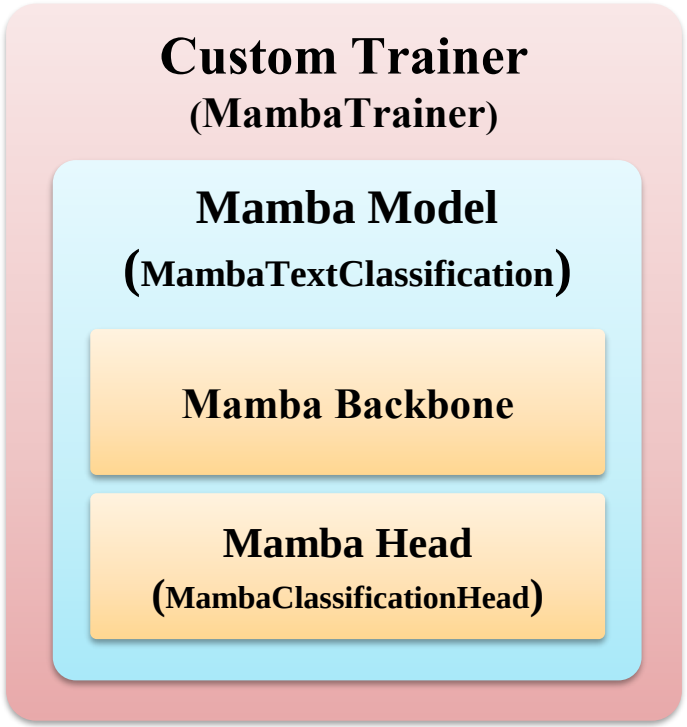




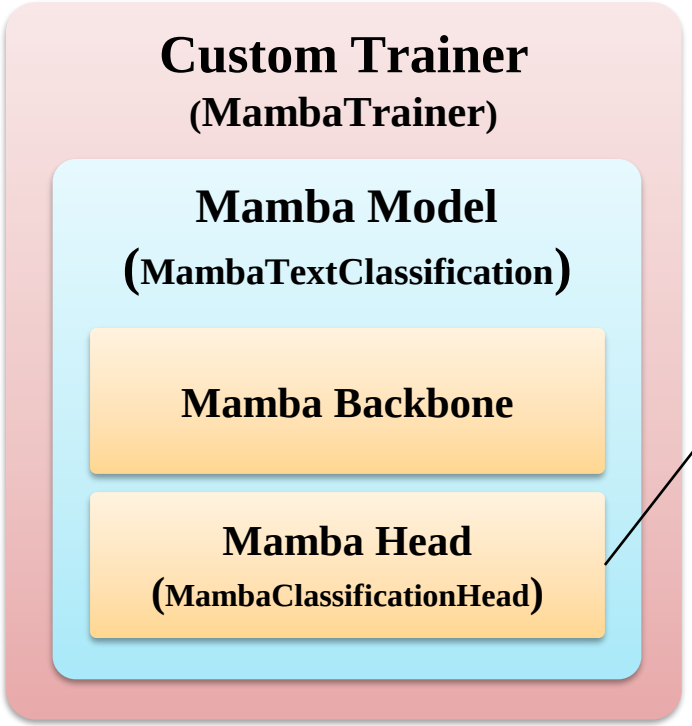
Outline

1. Motivation
2. State Space Models
3. Mamba
4. **Coding**

Coding

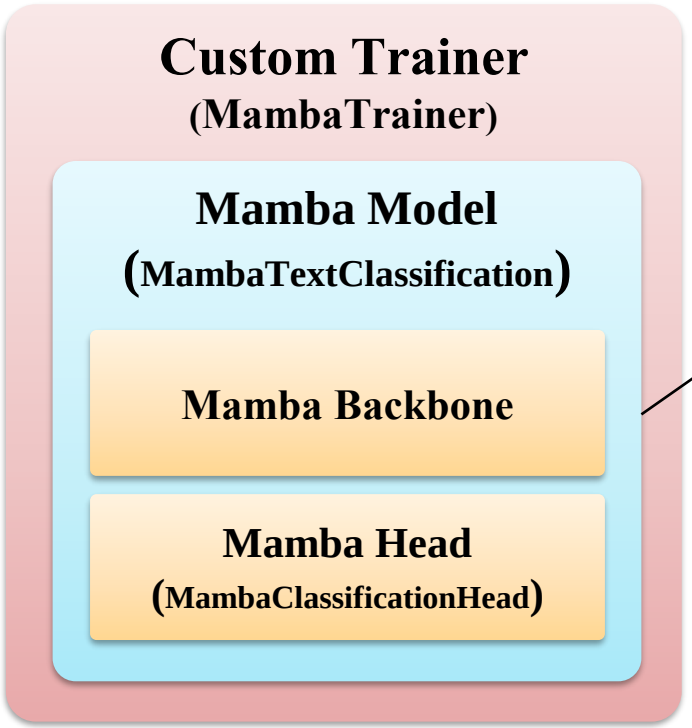

Custom Trainer
(MambaTrainer)
Mamba Model
(MambaTextClassification)
Mamba Backbone
Mamba Head
(MambaClassificationHead)

Coding



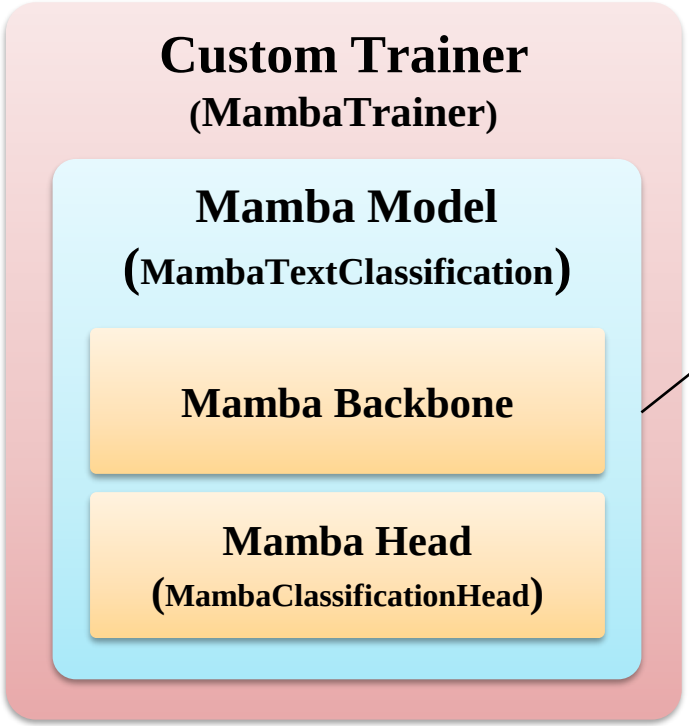

```python
# Định nghĩa lớp head để phân loại
class MambaClassificationHead(nn.Module):
    def __init__(self, d_model, num_classes, **kwargs):
        super(MambaClassificationHead, self).__init__()
        # Sử dụng một lớp tuyến tính để thực hiện phân loại dựa trên đầu vào
        # có kích thước d_model và num_classes cần phân loại.
        self.classification_head = nn.Linear(d_model, num_classes, **kwargs)

    def forward(self, hidden_states):
        return self.classification_head(hidden_states)
```

Coding



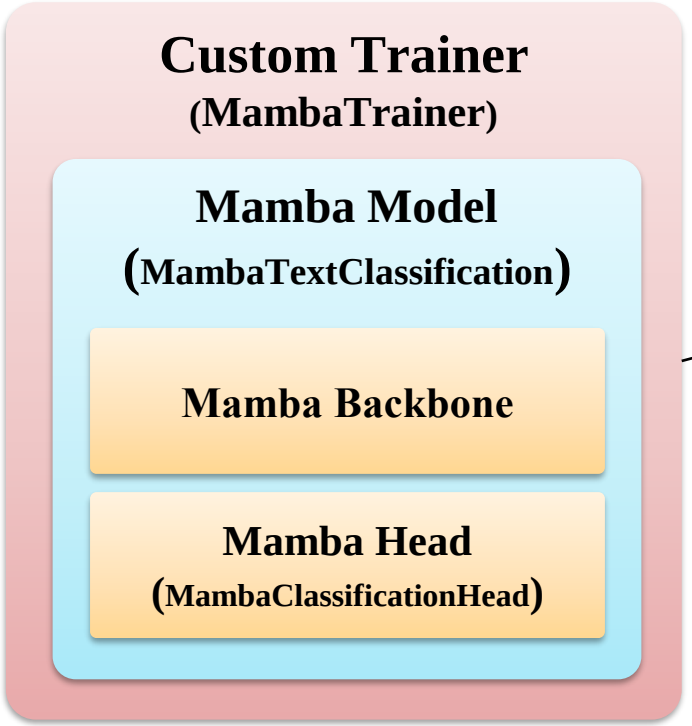

```python
class MambaTextClassification(MambaLMHeadModel):
    def __init__(
        self,
        config: MambaConfig,
        initializer_cfg=None,
        device=None,
        dtype=None,
    ) -> None:
        super().__init__(config, initializer_cfg, device, dtype)

        # Tạo một đầu phân loại sử dụng MambaClassificationHead với kích thước đầu vào là d_model và số lớp là 2.
        self.classification_head = MambaClassificationHead(d_model=config.d_model, num_classes=2)

        del self.lm_head

    def forward(self, input_ids, attention_mask=None, labels=None):
        # Truyền input_ids qua mô hình gốc để nhận hidden_states.
        hidden_states = self.backbone(input_ids)

        # Lấy trung bình của hidden_states theo chiều thứ 2 để tạo ra [CLS] feature đại diện
        mean_hidden_states = hidden_states.mean(dim=1)

        # Đưa mean_hidden_states qua đầu phân loại để nhận logits.
        logits = self.classification_head(mean_hidden_states)

        if labels is None:
          ClassificationOutput = namedtuple("ClassificationOutput", ["logits"])
          return ClassificationOutput(logits=logits)
        else:
          ClassificationOutput = namedtuple("ClassificationOutput", ["loss", "logits"])

          # Sử dụng hàm mất mát CrossEntropyLoss để tính loss.
          loss_fct = nn.CrossEntropyLoss()
          loss = loss_fct(logits, labels)

          return ClassificationOutput(loss=loss, logits=logits)
```

Coding




```python
class MambaTextClassification(MambaLMHeadModel):
    ...
    def predict(self, text, tokenizer, id2label=None):
        input_ids = torch.tensor(tokenizer(text)['input_ids'], device='cuda')[None]
        with torch.no_grad():
          logits = self.forward(input_ids).logits[0]
          label = np.argmax(logits.cpu().numpy())

        if id2label is not None:
          return id2label[label]
        else:
          return label

    @classmethod
    def from_pretrained(cls, pretrained_model_name, device=None, dtype=None, **kwargs):
        # Tải pretrained từ mô hình đã được huấn luyện trước đó.
        config_data = load_config_hf(pretrained_model_name)
        config = MambaConfig(**config_data)

        # Khởi tạo mô hình từ cấu hình và chuyển nó đến thiết bị và kiểu dữ liệu mong muốn.
        model = cls(config, device=device, dtype=dtype, **kwargs)

        # Tải trạng thái mô hình đã được huấn luyện trước đó.
        model_state_dict = load_state_dict_hf(pretrained_model_name, device=device, dtype=dtype)
        model.load_state_dict(model_state_dict, strict=False)

        # In ra các tham số embedding mới được khởi tạo.
        print("Newly initialized embedding:", set(model.state_dict().keys()) - set(model_state_dict.keys()))
        return model
```

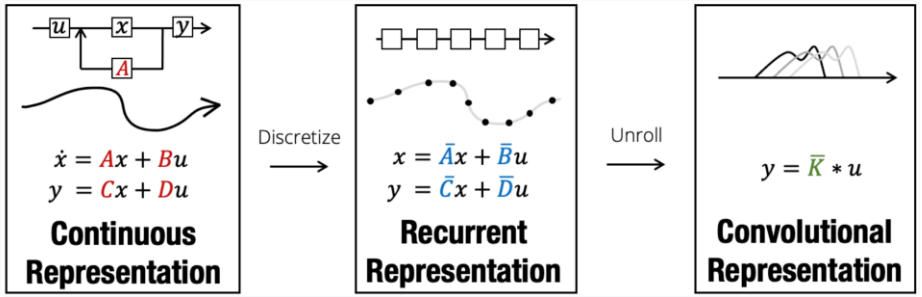
Coding

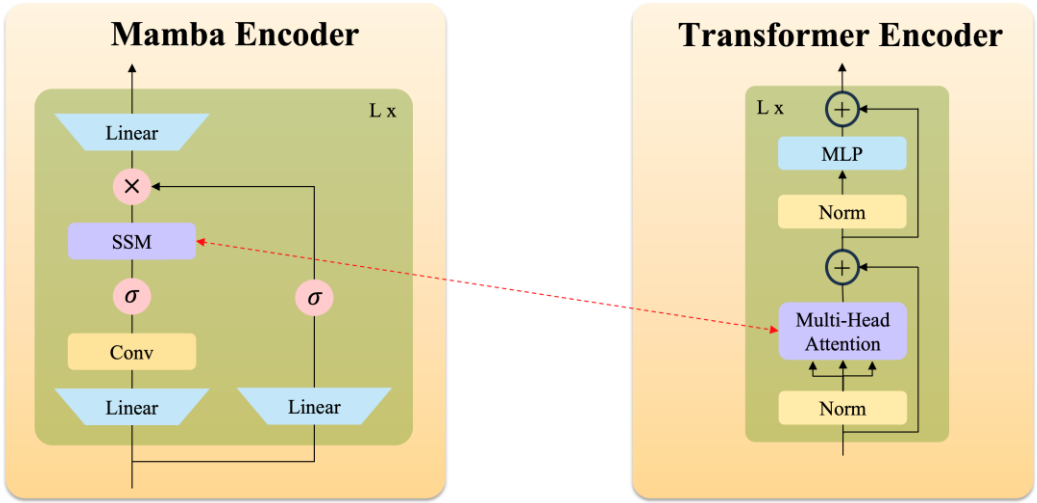


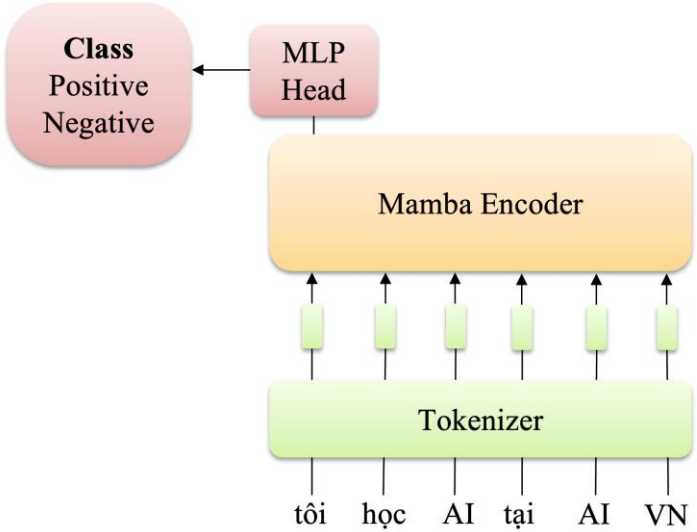

```python
# Định nghĩa một lớp con MambaTrainer kế thừa từ lớp Trainer.
class MambaTrainer(Trainer):

    # Định nghĩa hàm compute_loss để tính toán hàm mất mát trong quá trình huấn luyện.
    def compute_loss(self, model, inputs, return_outputs=False):
        # Lấy giá trị input_ids và labels từ inputs.
        input_ids = inputs.pop("input_ids")
        labels = inputs.pop('labels')

        # Gọi hàm forward của mô hình với input_ids và labels để nhận các kết quả.
        outputs = model(input_ids=input_ids, labels=labels)

        # Lấy giá trị loss từ kết quả của mô hình.
        loss = outputs.loss

        # Trả về cả loss và outputs nếu return_outputs là True, ngược lại chỉ trả về loss.
        return (loss, outputs) if return_outputs else loss

    # Định nghĩa hàm save_model để lưu model trong quá trình huấn luyện.
    def save_model(self, output_dir = None, _internal_call = False):
        # Kiểm tra nếu thư mục lưu trữ không được chỉ định, sử dụng thư mục mặc định từ đối số 'args'.
        if output_dir is None:
            output_dir = self.args.output_dir

        # Nếu thư mục đầu ra không tồn tại, tạo mới nó.
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)

        # Lưu trạng thái của mô hình PyTorch vào file 'pytorch_model.bin' trong thư mục đầu ra.
        torch.save(self.model.state_dict(), f"{output_dir}/pytorch_model.bin")

        # Lưu trạng thái của tokenizer vào thư mục đầu ra.
        self.tokenizer.save_pretrained(output_dir)

        # Lưu cấu hình của mô hình vào file 'config.json' trong thư mục đầu ra.
        with open(f'{output_dir}/config.json', 'w') as f:
            json.dump(self.model.config.to_dict(), f)
```

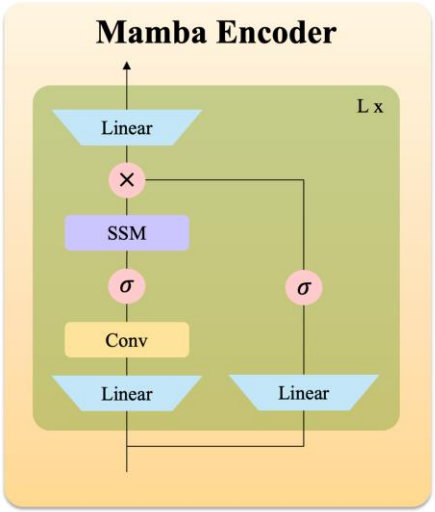
Summary

	RNN	Transformer	Ideal Model
Parallelizable	no	yes	yes
Training Stage	$O(n)$	$O(n^2)$	$O(n)$
Inference Stage	$O(n)$	$O(n^2)$	$O(n)$
Inference per Token	$O(1)$	$O(n)$	$O(1)$









AI VIET NAM

@aivietnam.edu.vn

Thanks!

Any questions?