Mamba: Linear-Time Sequence Modeling with Selective State Spaces

May 27, 2026

UNIVERSITY of WASHINGTON

State Space Model

General Sense of State Space Model

State Space Models (SSMs) are sequence models that:

- Maintain a hidden state over time
- Update this state with each new input
- Generate outputs based on the state

Key idea: Compress past information into a fixed-size state.

Intuition Example

Sentence:

"The cat that chased the mouse was hungry."

I

- You don't remember every word equally
- You keep a compressed summary

SSMs do the same:

- Compress past into a fixed state
- Update it step-by-step

What is an SSM?

- Maps input sequence $x(t)$ to output $y(t)$
- Uses latent state $h(t)$

$$x(t) \in \mathbb{R} \quad \rightarrow \quad h(t) \in \mathbb{R}^N \quad \rightarrow \quad y(t) \in \mathbb{R}$$

- Inspired by classical dynamical systems
- Combines ideas from:
  - RNNs
  - CNNs
  - Control theory

Core Equations(Discrete Time Formulation)

$$h_t = Ah_{t-1} + Bx_t$$
$$y_t = Ch_t$$

- x_t: input
- h_t: hidden state (memory)
- y_t: output
- Equivalent to a linear RNN
- Efficient sequential computation

Interpretation

- A: controls how memory evolves
- B: controls how input affects memory
- C: maps memory to output

Intuition:

- The model summarizes past inputs into h_t
- Updates this summary as new inputs arrive

Continuous-Time SSM

Defined by linear differential equations:

$$h'(t) = Ah(t) + Bx(t)$$

$$y(t) = Ch(t)$$

- A: state transition
- B: input projection
- C: output projection

In this case

$$h(t) = e^{At}h(0) + \int_0^t e^{A(t-\tau)}Bx(\tau)\,d\tau$$

Why Are SSMs Useful?

- Linear-time complexity
- Efficient memory usage
- Strong ability to model long sequences

Goal: Combine efficiency of RNNs with power of attention.

Convolutional View

SSM can also be written as a convolution:

$$K = (CB, CAB, CA^2B, \dots)$$

$$y = x * K$$

- Equivalent to recurrence
- Enables parallel training

Matrix expression

$$\overline{K}_j = C\overline{A}^j\overline{B}$$

$$y_t = \sum_{j=0}^{t} \overline{K}_j x_{t-j}$$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{L-1} \end{bmatrix} = \begin{bmatrix} C\overline{B} & 0 & 0 & \cdots & 0 \\ C\overline{AB} & C\overline{B} & 0 & \cdots & 0 \\ C\overline{A}^2\overline{B} & C\overline{AB} & C\overline{B} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C\overline{A}^{L-1}\overline{B} & C\overline{A}^{L-2}\overline{B} & C\overline{A}^{L-3}\overline{B} & \cdots & C\overline{B} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{L-1} \end{bmatrix}$$

Two Computation Modes

- **Recurrent mode**
  - Sequential
  - Used for inference
- **Convolution mode**
  - Parallelizable
  - Used for training

Key insight: Same model, two computation paths

Discretization

Convert continuous system to discrete:

$$\bar{A} = \exp(\Delta A)$$

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

- Δ: step size
- Known as Zero-Order Hold (ZOH)

Why Discretization Matters

- Connects continuous and discrete models
- Provides:
  - Stability
  - Normalization
  - Resolution invariance
- Links to RNN gating mechanisms

Linear Time Invariance (LTI)

- Parameters do not change over time:

$$(A, B, C, \Delta) \text{ constant across } t$$

- Enables:
  - Convolution representation
  - Efficient computation

Limitations of LTI

- Cannot adapt to input content
- Same dynamics for all tokens
- Weak for:
  - Language
  - Discrete data

Key motivation for Mamba

Structured SSMs

To be efficient, impose structure on A:

- Typically diagonal

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

- Reduces parameter count
- Enables fast computation

Multi-Channel Inputs

For sequence with:

- Batch size B
- Length L
- Channels D
- Apply SSM independently per channel
- Total state size:

$$D \times N$$

Efficiency Challenge

Full state tensor:

$$(B, L, D, N)$$

- Very large
- Expensive in memory and compute

Core problem: How to scale SSMs efficiently

General Definition of SSMs

SSMs broadly include:

- Kalman filters
- Hidden Markov Models
- Linear dynamical systems
- RNNs

In this paper:

- Focus on structured SSMs (S4)

SSM Architectures

Examples built on SSMs:

- Linear Attention
- H3
- Hyena
- RetNet
- RWKV

- Different ways to integrate SSM layers
- Trade-offs in efficiency and performance

Key Takeaways

- SSM = sequence model with latent state
- Dual view:
  - Recurrence (RNN-like)
  - Convolution (CNN-like)
- LTI enables efficiency but limits expressivity
- Structure is crucial for scalability

Next: Selective SSMs

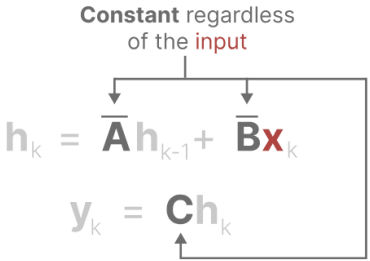
Selective SSMs

Selective SSMs

- Motivation for Selective SSMs: 'Adapting to the Input'
- Selective SSM formulation
- Selective Scan Algorithm
- Addressing Computational Inefficiency: Hardware Scan Algorithm

Motivation: 'Adapting to the Input'

- SSMs are **LTI** $\rightarrow$ same compression for every input.



- However, not all inputs should be treated equally

- We need a model that can dynamically focus on important parts of input

Key Idea: Selection

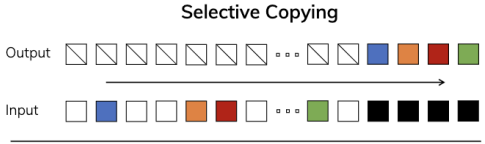
Classic SSM compress information into hidden state h, but it cannot 'select' which information to compress or filter out.
Selective SSM (aka S6 model) should:

- Focus on relevant inputs and compress them into hidden state
- Ignore irrelevant inputs and filter out of hidden state

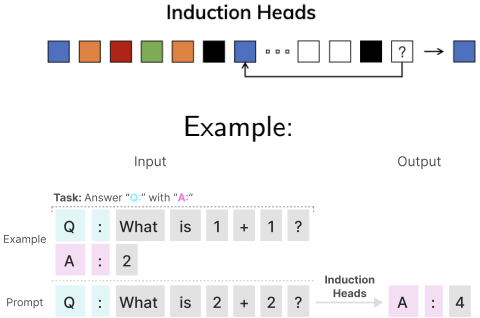
'Selection' Assessments

To assess **selectivity**, Selection can be formalized by its performance on two toy tasks:

Selective Copying - copy task that varies position of tokens to copy



Induction heads: reproduce patterns found in input



From SSM to Selective SSM

Key Idea: Make parameters of the SSM depend on the input

S4: Fixed recurrence

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

$$y_t = Ch_t$$

$\bar{A}, \bar{B}, C$ fixed for all t

S6: Dynamic recurrence

$$h_t = \bar{A}_t h_{t-1} + \bar{B}_t x_t$$

$$y_t = C_t h_t$$

$\Delta_t,\ B_t,\ C_t$ depend on x_t

Algorithm 1 SSM (S4)

Input: x : (B, L, D)
Output: y : (B, L, D)
$\boldsymbol{A}$: (D, N) ← Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B}$: (D, N) ← Parameter
$\boldsymbol{C}$: (D, N) ← Parameter
$\underline{\Delta}$: (D) ← τ_Δ(Parameter)
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$: (D, N) ← discretize($\Delta, \boldsymbol{A}, \boldsymbol{B}$)
y ← SSM($\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C}$)($x$)
▷ Time-invariant: recurrence or convolution
return y

Algorithm 2 SSM + Selection (S6)

Input: x : (B, L, D)
Output: y : (B, L, D)
$\boldsymbol{A}$: (D, N) ← Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B}$: (B, L, N) ← $s_B(x)$
$\boldsymbol{C}$: (B, L, N) ← $s_C(x)$
$\underline{\Delta}$: (B, L, D) ← τ_Δ(Parameter+$s_\Delta(x)$)
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$: (B, L, D, N) ← discretize($\Delta, \boldsymbol{A}, \boldsymbol{B}$)
y ← SSM($\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C}$)($x$)
▷ Time-varying: recurrence (*scan*) only
return y

Selective SSM: It Works!

By making SSM linear-time variant, selective SSM's perform well on selection tasks

Model	Arch.	Layer	Acc.
S4	No gate	S4	18.3
-	No gate	S6	**97.0**
H3	H3	S4	57.0
Hyena	H3	Hyena	30.1
-	H3	S6	**99.7**
-	Mamba	S4	56.4
-	Mamba	Hyena	28.4
Mamba	Mamba	S6	**99.8**

Table 1: (**Selective Copying.**) Accuracy for combinations of architectures and inner sequence layers.

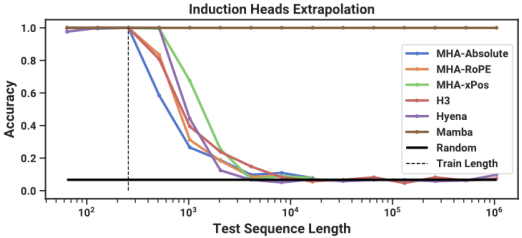


Table 2: (**Induction Heads.**) Models are trained on sequence length $2^8 = 256$, and tested on increasing sequence lengths of $2^6 = 64$ up to $2^{20} = 1048576$. Full numbers in Table 11.

Figure: Gu and Dao, 2024

Downside of Selective SSM: Computational Inefficiency

Selective SSM pressent computational problems:

1. Sequential training: convolutional representation does not work with LTV SSM due to dynamic matrices
2. Computing hidden state may take lot more FLOPS and memory

Namely, for point 2:

$$h_t \in \mathbb{R}^{B \times D \times N}$$

For each timestep t, update all states:

$$h_t = \overline{A}_t h_{t-1} + \overline{B}_t x_t$$

With diagonal/structured A,

$$O(BLDN)$$

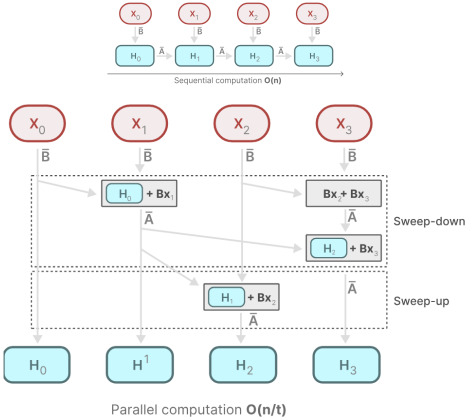
Naively storing all hidden states also costs:

$$O(BLDN)$$

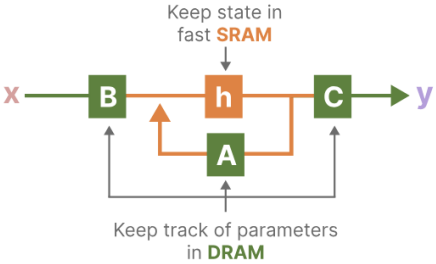
Mamba's Hardware-Aware Algorithm - An Efficient Implementation for Selective SSM

Mamba implements kernel fusion, parallel scan, and recomputation to address the computational inefficiencies of selective scan:

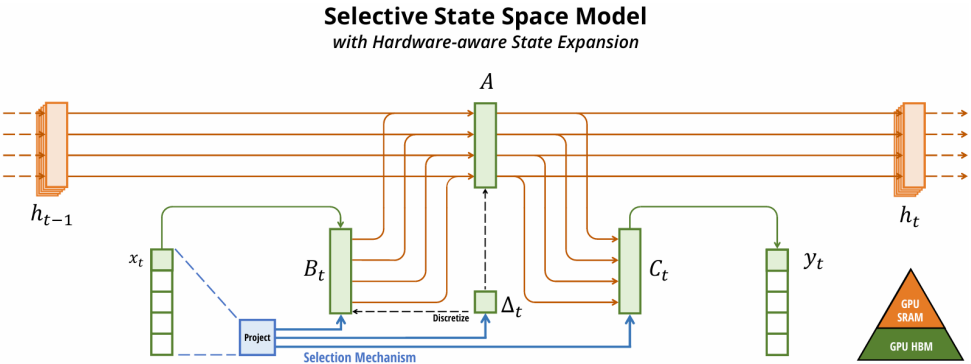
'Parallel' Scan



Kernel fusion



Result: Select SSM with Hardware-Aware State Expansion



Mamba

Mamba Architecture

- Combine:
  - SSM block
  - MLP block
- Single unified block
- Repeated across layers
- Use SiLU activation
- Residual + normalization
- Similar parameter count to Transformer

Architecture Details

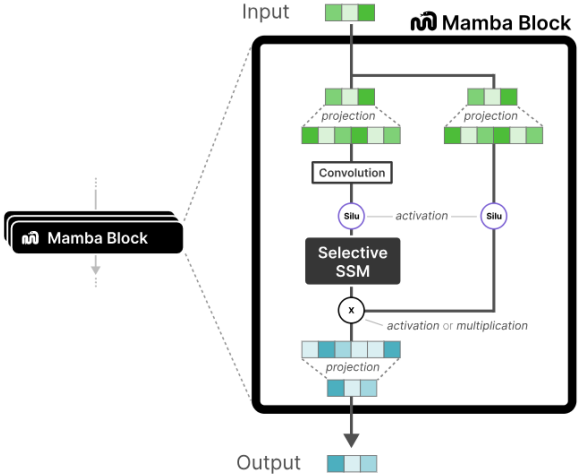


Diagram of Selective SSM

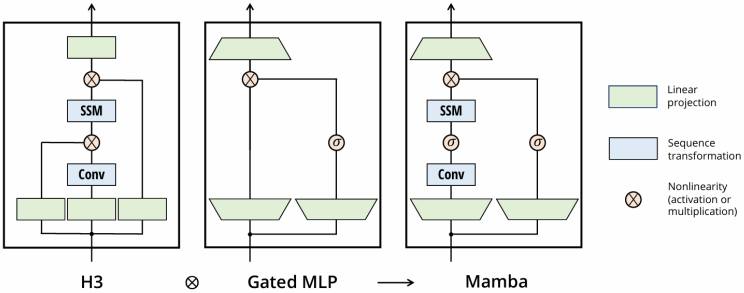


Figure: Mamba Structure

Connection to Gating

Special case:

$$g_t = \sigma(\text{Linear}(x_t))$$

$$h_t = (1 - g_t)h_{t-1} + g_t x_t$$

- Equivalent to gated RNN

Insight: selection generalizes gating

Interpretation of Δ

- Controls memory vs input
- Large Δ:
  - Reset state
  - Focus on input
- Small Δ:
  - Keep memory
  - Ignore input

Gated MLP

Input: $x \in \mathbb{R}^{B \times L \times D}$

Step 1: Linear projections

$$u = W_u x = \Sigma_{i=0}^{D} (W_u)_{i,j} x_{b,t,j}, \quad v = W_v x = \Sigma_{i=0}^{D} (W_v)_{i,j} x_{b,t,j}$$

Step 2: Activation (gate)

$$g = \sigma(u) = u \cdot sigmoid(u) = \frac{u}{1 + e^{-u}}$$

Step 3: Elementwise gating

$$h = g \odot v \quad h_{b,t,i} = g_{b,t,i} \cdot v_{b,t,i}$$

Step 4: Output projection

$$y = W_o h$$

Final form:

$$y = W_o \big(\sigma(W_u x) \odot W_v x\big)$$

H3 Block

Input: $x \in \mathbb{R}^{B \times L \times D}$

Step 1: Linear projections

$$u = W_u x, \quad v = W_v x$$

Step 2: First gating

$$h_1 = u \odot v$$

Step 3: Local convolution

$$h_2 = \text{Conv}(h_1) = \Sigma_{k=0}^{K-1} w_k^{(i)} \cdot h_{1,b,t-k,i} \quad \text{K as the kernel size}$$

Step 4: SSM transformation

$$h_3 = \text{SSM}(h_2) \quad (s_{b,t,i} = \bar{A}_i s_{b,t-1,i} + \bar{B}_i h_{2,b,t,i}, \quad h_3 = C_i s_{b,t,i})$$

Step 5: Second gating

$$h_4 = h_3 \odot v$$

Step 6: Output projection

$$y = W_o h_4$$

Final form:

$$y = W_o \Big(\mathsf{SSM}(\mathsf{Conv}(W_u x \odot W_v x)) \odot W_v x \Big)$$

Mamba Block

Input: $x \in \mathbb{R}^{B \times L \times D}$

Step 1: Linear projections

$$u = W_u x, \qquad v = W_v x$$

Step 2: Gate activation

$$g = \sigma(u)$$

Step 3: Gated interaction

$$h_1 = g \odot v$$

Step 4: Local convolution

$$h_2 = \mathrm{Conv}(h_1)$$

Step 5: Selective SSM

$$h_3 = \mathsf{SSM}_{\mathsf{selective}}(h_2) \quad (s_B(x) = W_B x + b_B \quad s_C(x) = W_C x + b_C)$$

Step 6: Output activation

$$h_4 = \sigma(h_3)$$

Step 7: Output projection

$$y = W_o h_4$$

Final form:

$$y = W_o\big(\sigma\big(\mathrm{SSM}_{\mathrm{sel}}\big(\mathrm{Conv}(\sigma(W_u x) \odot W_v x)\big)\big)\big)$$

Interpretation of B and C

- B_t: controls input $\rightarrow$ state
- C_t: controls state $\rightarrow$ output
- Enables fine-grained filtering

Why Selection Works

- Filters irrelevant tokens
- Adapts to content
- Enables long-context reasoning

Key benefit:

- Efficient + expressive

Key Takeaways

- LTI models are limited
- Selection introduces input-dependent dynamics
- Requires new computation (Selective Scan)
- Leads to Mamba architecture

Core idea: efficient content-based reasoning