勉強会　240606
～状態空間モデルとMamba～

M2 上野

目次

- ✓ 概要
  - イメージ
  - アーキテクチャ
  - 性能
- ✓ 状態空間モデル
  - 状態空間モデル：SSM: State Space Model
  - 構造化状態空間モデル：S4: Structured State Space Model
  - H3: Hungry Hungry HiPPOs
- ✓ 選択的状態空間モデル：Mamba
  - Selection Mechanism
  - Hardware-aware Algorithm
  - Vision Mamba（≠VMamba），MambaOut

～ 概要 ～

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

概要

- ✓ Mambaは**選択的状態空間モデル**を採用した新しいアーキテクチャ
  - 高度な推論能力，シーケンス長に対する性能のスケーリング，
    推論時間の線形増加が特徴
- ✓ Transformerベースのモデルと比べて，同程度のパラメータ数
  で優れた性能を示す

参考文献：

S4	https://arxiv.org/abs/2111.00396
H3	https://arxiv.org/abs/2212.14052
Mamba	https://arxiv.org/abs/2312.00752
ViM	https://arxiv.org/abs/2401.09417
MambaOut	https://arxiv.org/abs/2405.07992
Qiita解説	https://qiita.com/peony_snow/items/649ecb307cd3b5c10aa7
HiPPO導出	https://zenn.dev/izmyon/articles/8374a11d272602
図を引用	https://zenn.dev/kotoba_tech/articles/3eb0984d8fdfb8

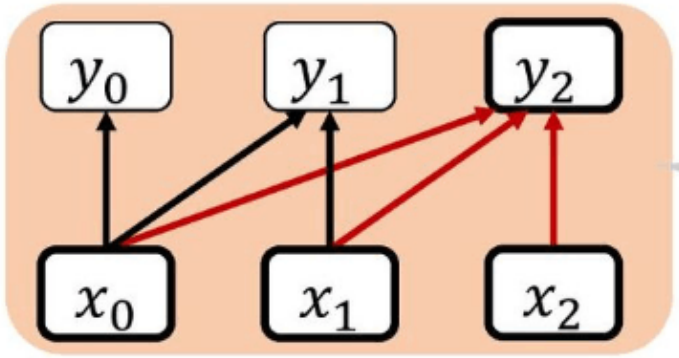
状態空間モデルのイメージ

状態空間モデルはTransformerとRNNの良いとこ取りを目指す

- Transformer (Attention に注目)
  - 推論：😫 $O(L^2)$
  - 訓練：😊 並列化可能
- RNN
  - 推論：😊 $O(L)$
  - 訓練：😫 並列化不可

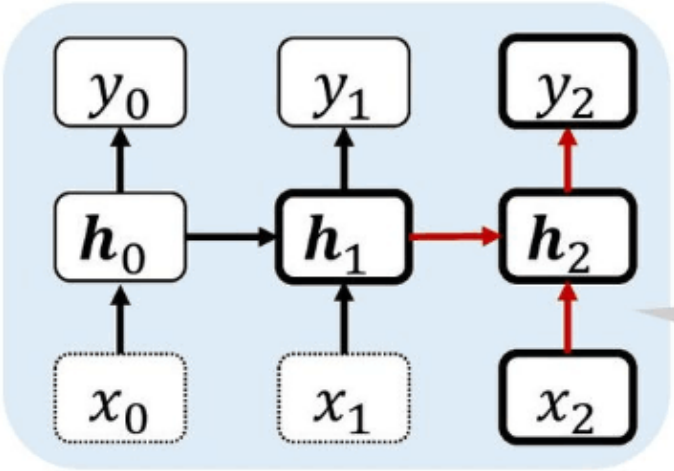
⬇

- SSM
  - 推論：😊 $O(L)$
  - 訓練：😊 並列化可能



過去入力を
全て見に行く

Attention



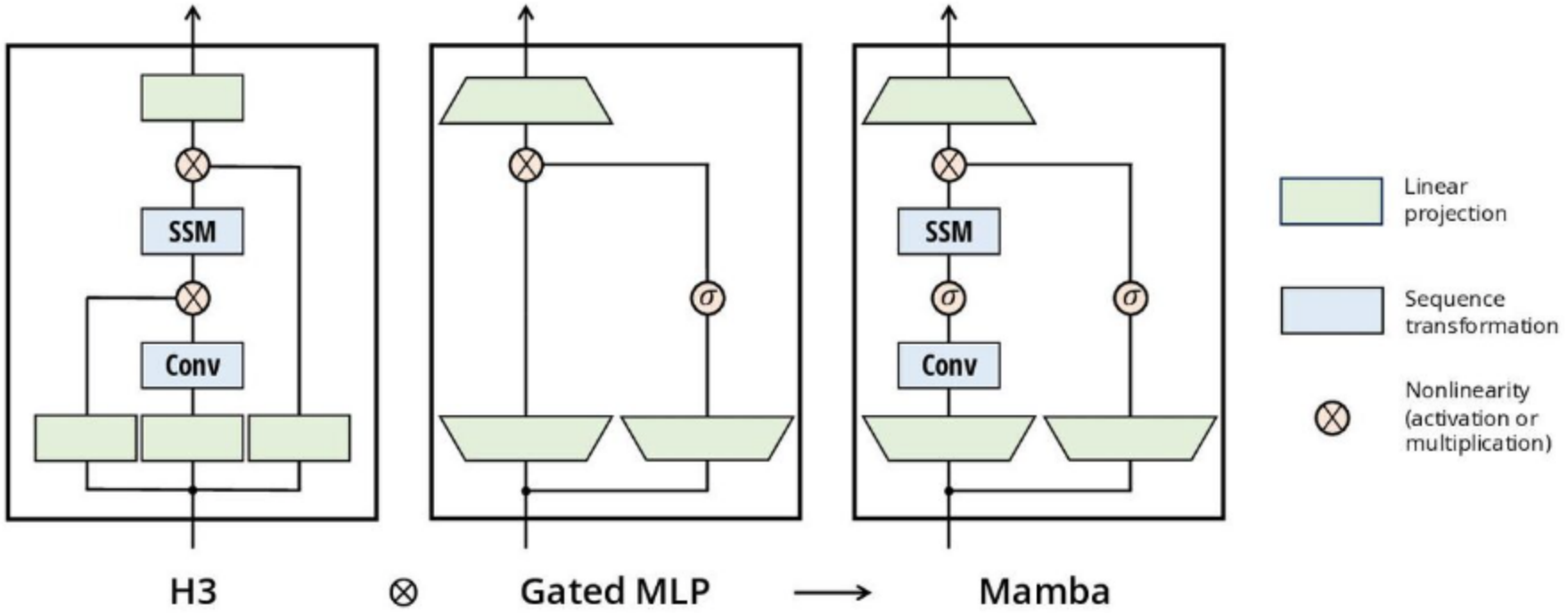
現在の入力と
1つ前の"状態"
のみを見る

SSM (≈ Linear RNN)

Mambaの概要｜アーキテクチャ

✓状態空間モデルH3に対してMLPを導入

– **MambaはTransformerなどと同様にブロックを重ねて実装**



SSM：　状態空間モデル
σ：　　活性化（SiLU，Swish）
正規化：Layer Normalization
残差ブロックを導入（RetNetと同様）

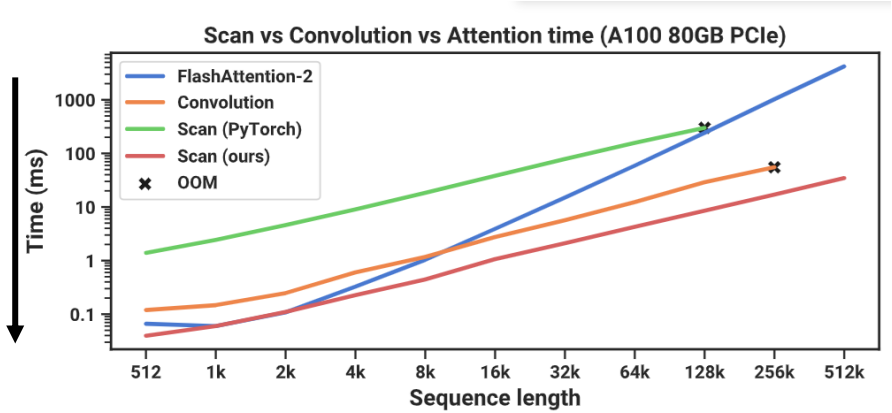
Mambaの概要｜性能

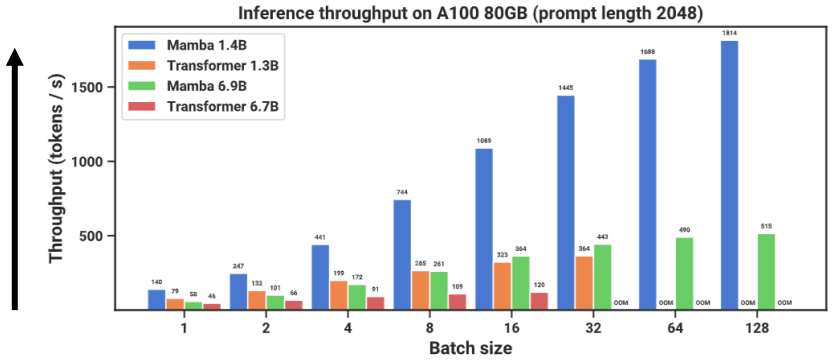
✓パラメータ数が同程度のモデル
より優れた性能

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

✓系列長やバッチサイズに対する
性能が顕著

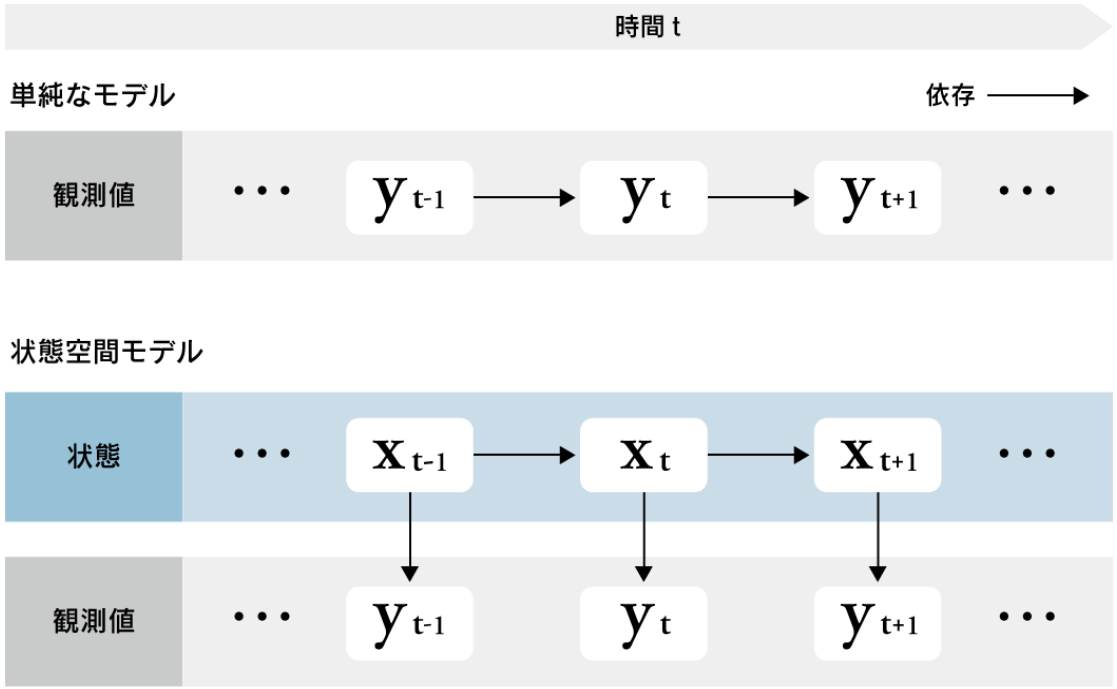




～ 状態空間モデル ～

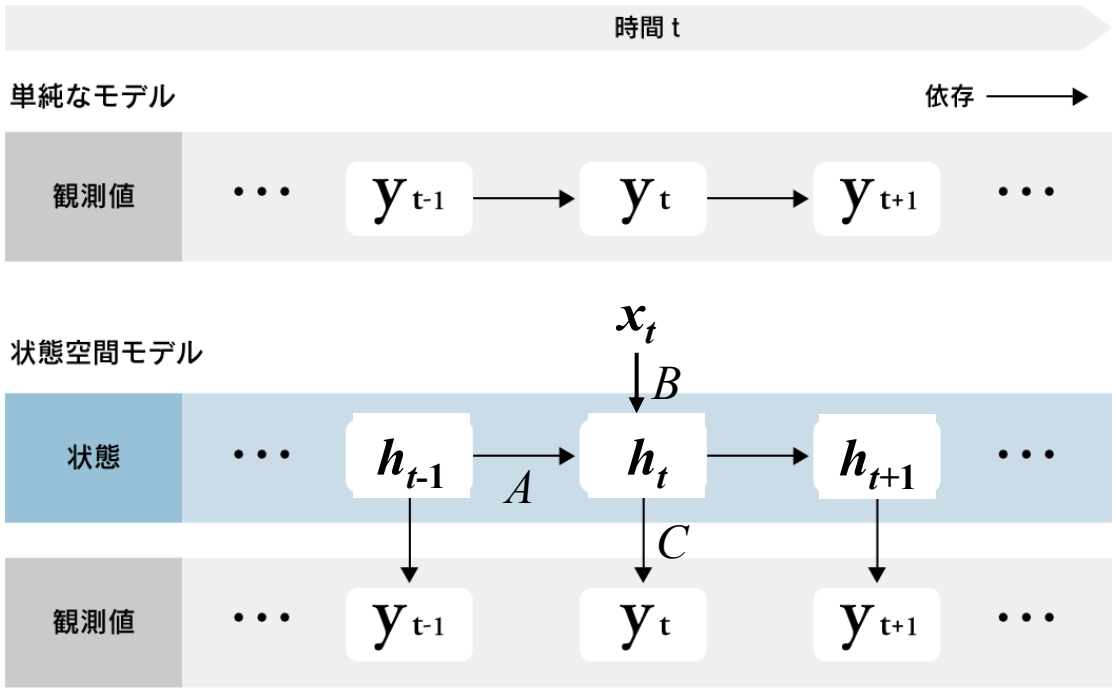
状態空間モデル｜概要

- ✓状態空間モデルは系列から系列への変換器（写像）
  - Transformerと同様の並列処理学習が可能
  - **$O(L)$で推論可能**
    - TransformerはAttentionが$O(L^2)$



状態空間モデル（オリジナル）

状態空間モデル｜定義



$t \in \mathbb{R}$

$$\begin{cases} h'(t) = Ah(t) + Bx(t) \\ y(t) = Ch(t) \end{cases}$$

$t \in \{\ldots, -1, 0, 1, \ldots\}$

$$\begin{cases} h_t = \bar{A}h_{t-1} + \bar{B}x_t \\ y_t = \bar{C}h_t \end{cases}$$

A，B，Cは入力$x(t)$に対して不変

離散状態空間モデル｜定式化

$$\begin{cases} h_t = \bar{A}h_{t-1} + \bar{B}x_t \\ y_t = \bar{C}h_t \end{cases}$$

離散状態空間モデルの定義

✓ h_{-1} = 0のとき，yは以下のように定式化可能

$$h_0 = \bar{A}h_{-1} + \bar{B}x_0 = \bar{B}x_0,\ y_0 = \bar{C}h_0 = \bar{C}\bar{B}x_0$$
$$h_1 = \bar{A}h_0 + \bar{B}x_1 = \bar{A}\bar{B}x_0 + \bar{B}x_1,\ y_1 = \bar{C}h_1 = \bar{C}\bar{A}\bar{B}x_0 + \bar{C}\bar{B}x_1$$

$$y_k = \bar{C}h_k = \bar{A}^k\bar{B}\bar{C}x_0 + \bar{A}^{k-1}\bar{B}\bar{C}x_1 + \cdots + \bar{B}\bar{C}x_k$$

✓ まとめると以下の**畳み込みで表現可能**

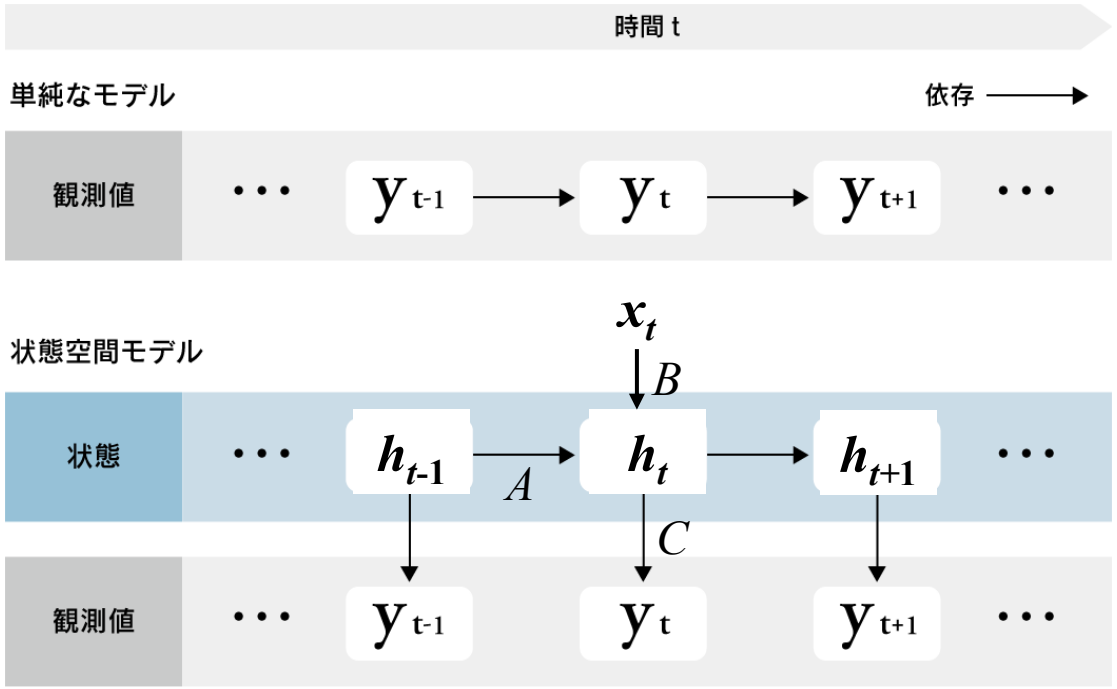
$$y = x * K \quad K = \left(\bar{A}^k\bar{B}\bar{C}, \ldots, \bar{A}\bar{B}\bar{C}, \bar{B}\bar{C}\right)$$

✓ xは入力時点で確定＆Kは不変(事前計算可能) ➡並列計算が可能

離散状態空間モデル｜課題

✓状態空間モデルは状態hが過去の情報をすべて持ち，
Aが系列間のやり取りを担う

➡ ランダムな初期値では計算不可 ＆ kが高次元の場合に計算量が爆発



$$y = x * K \quad K = (\bar{B}\bar{C}, \bar{A}\bar{B}\bar{C}, \ldots, \bar{A}^k\bar{B}\bar{C}, \ldots)$$

構造化状態空間モデルS4

- ✓ S4：連続変数（Δ，A，B）を離散変数（A，B）に変換する段階を導入
  - ΔがRNNのゲーティング機構の役割を果たす
    - Δが大きいとき：状態をリセットする
    - Δが小さいとき：状態を保持する

$$h'(t) = Ah(t) + Bx(t)$$
$$y(t) = Ch(t)$$
$$h_t = \overline{A}h_{t-1} + \overline{B}x_t$$
$$y_t = Ch_t$$
$$K = (CB, C\overline{A}B, \ldots, C\overline{A}^kB, \ldots)$$
$$y = x * K$$

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

構造化状態空間モデルS4｜係数行列の初期化

✓ 係数行列Aを以下のように初期化（HiPPO行列）

$$\bar{A}_{nk} = \begin{cases} \sqrt{(2n+1)} * \sqrt{(2k+1)} & (n > k) \\ n+1 & (n = k) \\ 0 & (n < k) \end{cases}$$

導出過程は↓
https://zenn.dev/izmyon/articles/8374a11d272602#hippo-(high-order-polynomial-projection-operator%3B-%E9%AB%98%E6%AC%A1%E5%A4%9A%E9%A0%85%E5%BC%8F%E6%8A%95%E5%BD%B1%E6%BC%94%E7%AE%97%E5%AD%90)

※ このn，kは行列のインデックスなので注意

✓ 係数行列を上三角行列へ変換し，計算量の削減に成功
- 系列の記憶・系列間の比較が困難 → H3

Hungry Hungry Hippos： H3

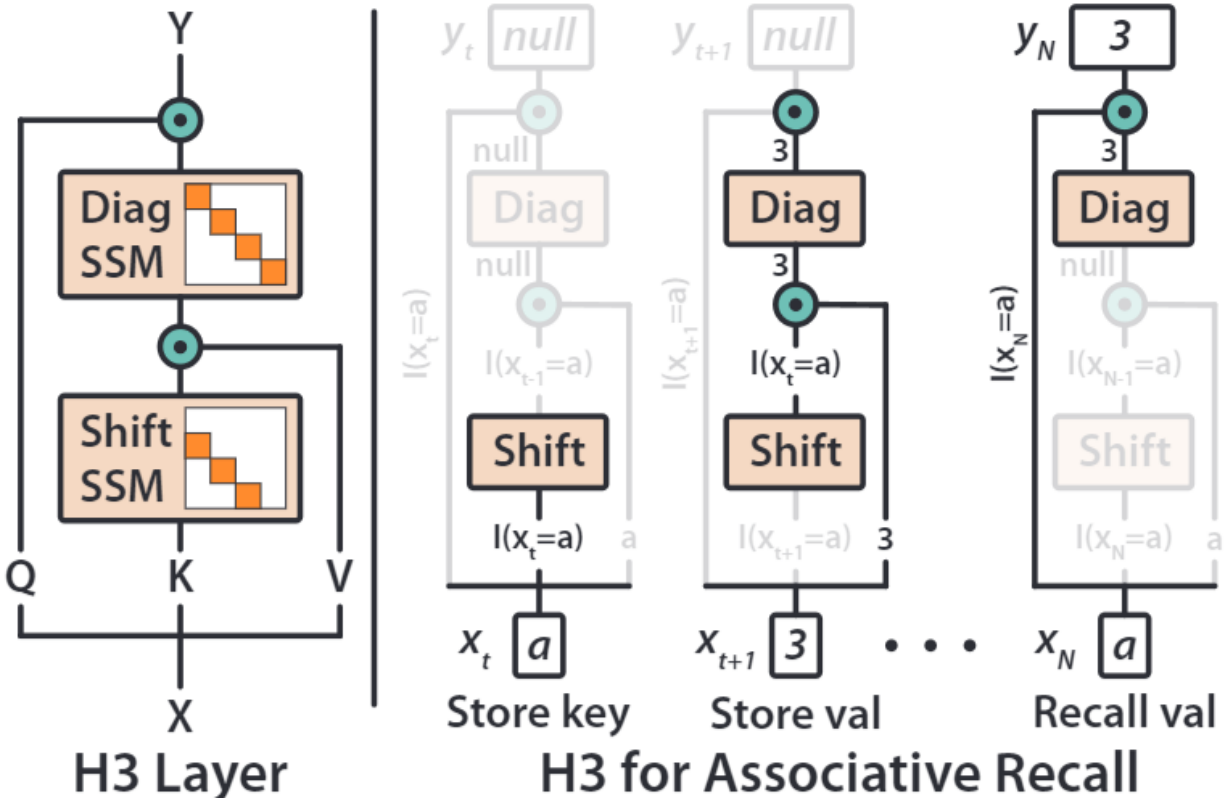
- ✓ Induction HeadとAssociative Recall性能を改善した状態空間モデル
  - – Induction Head：　　現在の系列以前から，次の系列を予測
    - – 過去の系列情報の保持
  - – Associative Recall：　アルファベット→数字 の辞書を学習
    - – 系列間の比較

Task	Input	Output	Sequence Length	Vocab Size
Induction Head	$a\ b\ c\ d\ e \vdash f\ g\ h\ i \ldots x\ y\ z \vdash$	f	30	20
Associative Recall	$a\ 2\ c\ 4\ b\ 3\ d\ 1\ a$	2	20	10

Task	Random	S4D	Gated State Spaces	H3	Attention
Induction Head	5.0	35.6	6.8	**100.0**	**100.0**
Associative Recall	25.0	86.0	78.0	99.8	**100.0**

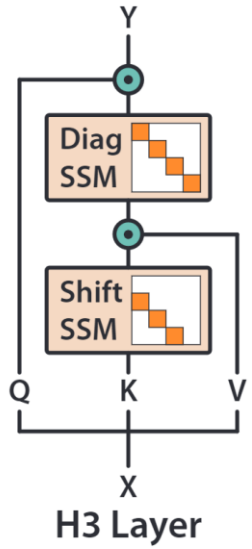
H3 Layer

✓ 構造化状態空間モデルS4に対して線形Attentionを追加



$$\mathbf{Q} \circ \mathrm{SSM}_{\mathrm{diag}}(\mathrm{SSM}_{\mathrm{shift}}(\mathbf{K}) \circ \mathbf{V})$$

H3｜DiagSSM・ShiftSSM



$$A_{i,j} = \begin{cases} 1 & \text{if } i-1 = j \\ 0 & \text{otherwise} \end{cases}$$

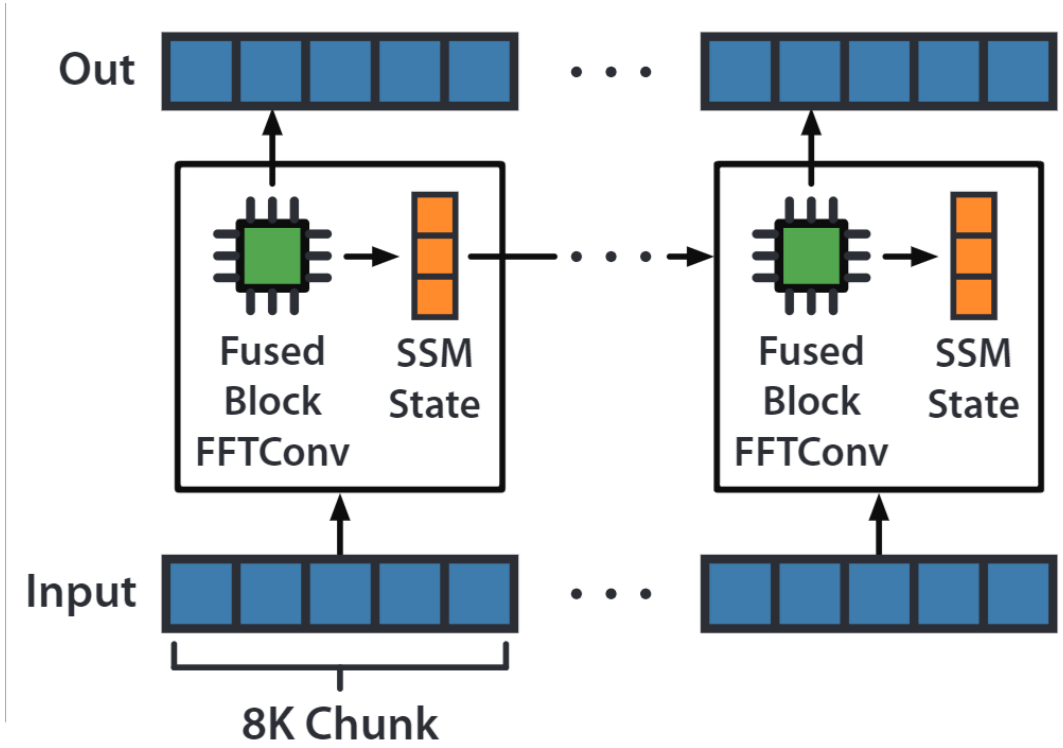
ShiftSSM
状態x_iをシフトし前の入力u_{i-1}を次の状態に保存

$$A = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$$

DiagonalSSM
対角行列Aについて，各成分を
HiPPOプロジェクションで初期化

H3｜FlashConv

- ✓FFT，ポイントワイズ乗算，逆FFTを融合し，
  Tensor Coresで効率的に演算
  - 8Kまでの効率化を確認
  - 8Kを超える場合は状態遷移アルゴリズムでチャンクに分割



FlashConv

既存の状態空間モデルの課題

✓係数行列が入力に非依存な為，動的推論が不可

$$y = x * K \quad K = (\bar{B}\bar{C}, \bar{A}\bar{B}\bar{C}, \ldots, \bar{A}^k\bar{B}\bar{C}, \ldots)$$

✓A，B，Cを入力依存にすると・・・

– 入力に応じてそれぞれの形が変化

➡ Kの事前計算ができない

➡ 並列化計算ができない

➡ k乗の行列計算＋畳み込みで計算量爆発

➡ Mambaではこの問題を解消

～選択的状態空間モデル：Mamba～

選択的状態空間モデル：Mamba

✓状態空間モデル最大の欠点（LTI: Linear Time Invariance）を解消する為,
Selection MechanismとHardware-aware Algorithmを提案

目次

✓アーキテクチャ

✓Selection Mechanism

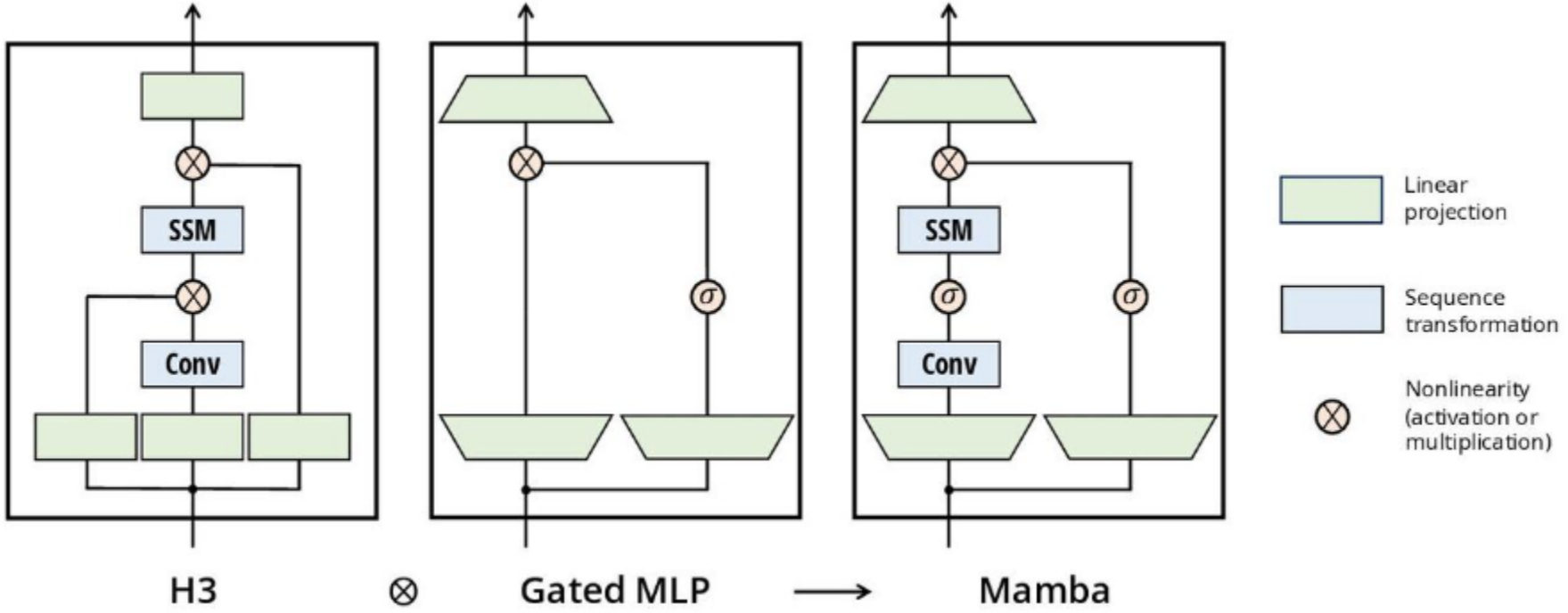
- Selective Copying ← **Mambaのキモ**
- Induction Heads ← H3

✓Hardware-aware Algorithm ← 実装上の工夫

- Parallel Scan
- Kernel Fusion

Mamba｜アーキテクチャ

✓状態空間モデルH3に対してGated MLPを導入したMambaBlockの積層



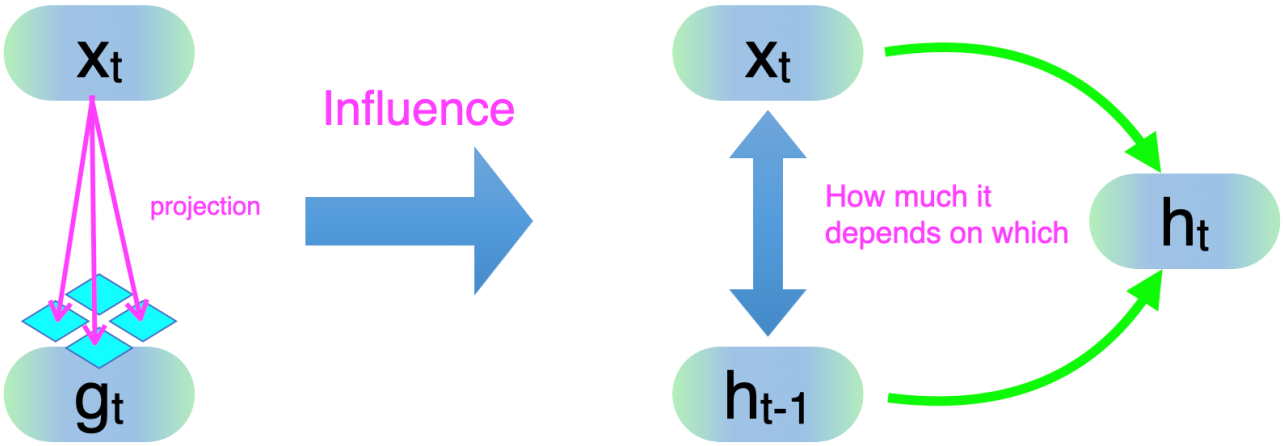
SSM：　状態空間モデル
σ：　　活性化（SiLU，Swish）
正規化：Layer Normalization
残差ブロックを導入（RetNetと同様）
SSMの部分で先の計算が行われる

Selection Mechanism｜Selective Copying

✓無関係な系列をフィルタリングし，選択的に内容をコピー

$$g_t = \sigma(\text{Linear}(x_t))$$ ← Parameterized projection depending on x

$$h_t = (1 - g_t)h_{t-1} + g_t x_t$$ ← Inference depending on g

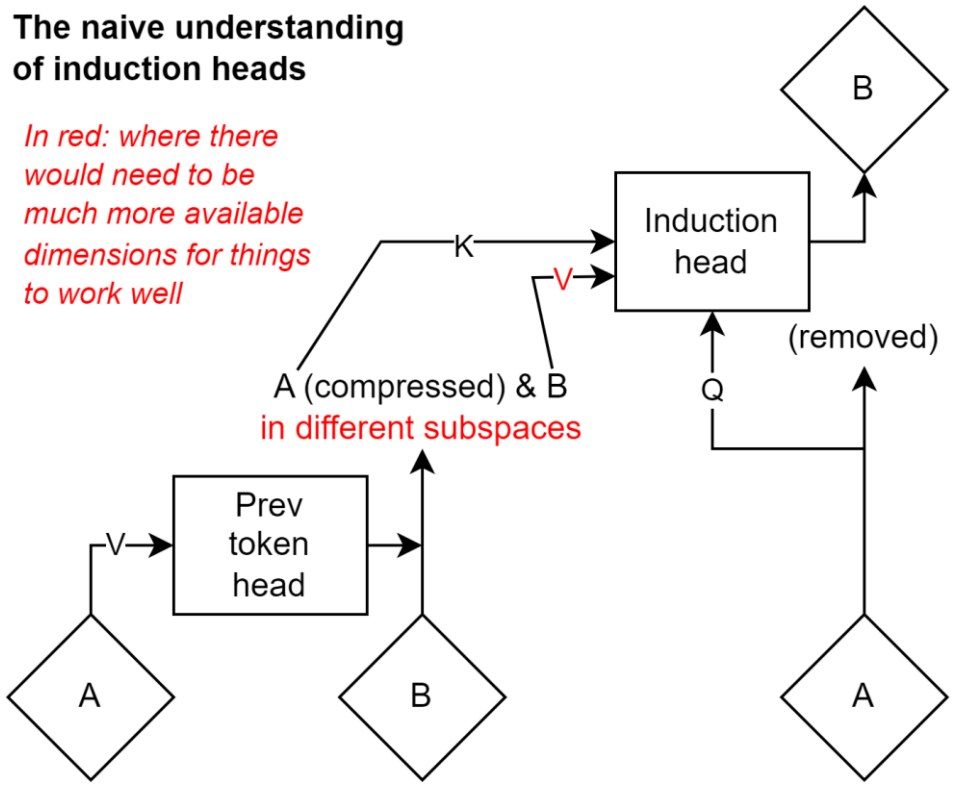


σ：シグモイド関数

Selection Mechanism｜Induction Heads

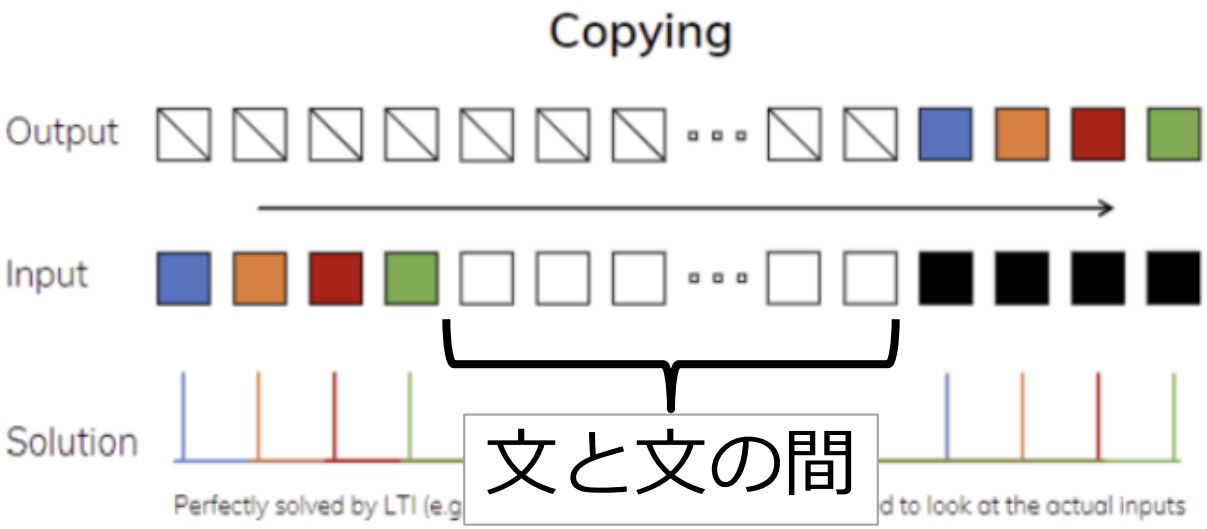
- ✓現在の系列以前のシーケンスを学習し，
  同じパターンが出現するか否かを予測するアルゴリズム
  - [A][B] ・・・[A] → [B]

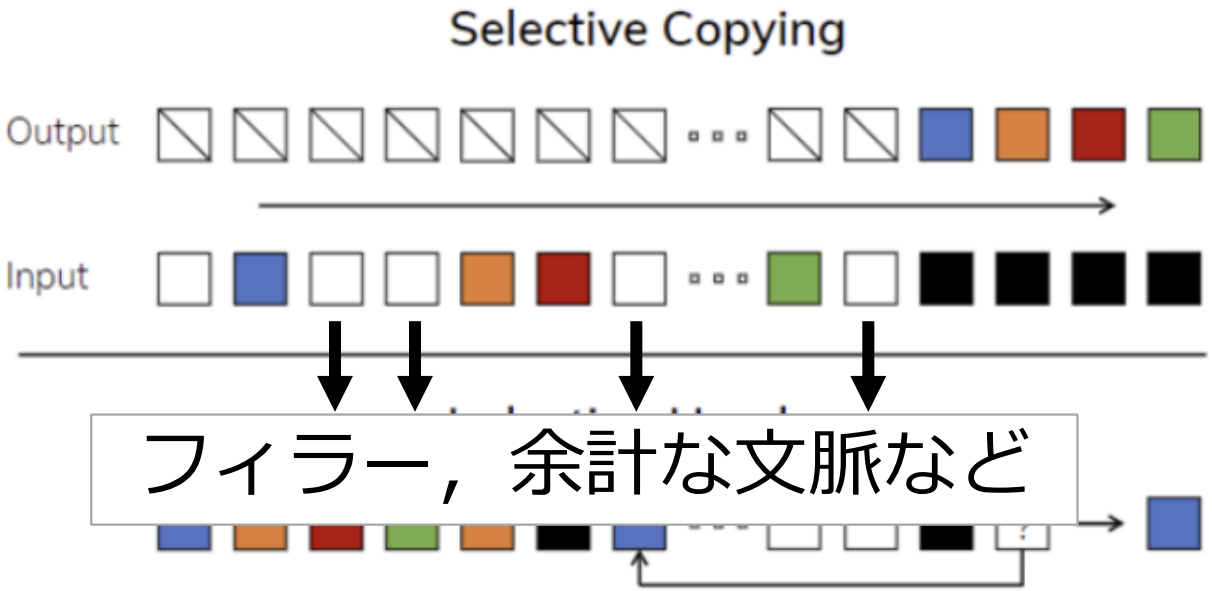
In-Context Learningとも関係しているらしい



Selection Mechanism

✓ Selection Mechanismは動的な文脈のフィルタリングが可能な為，
コンテキスト長に対してスケーリングする





S4とS6の違い

Algorithm 1 SSM (S4)

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

Algorithm 2 SSM + Selection (S6)

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

N：次元
B：バッチサイズ
L：系列長
D：チャネル
τ： softplus

s_B, s_C： xの線形射影
s_Δ： xの線形射影をbroadcast

Selective Copyingの性能

- ✓既存の状態空間モデルにSelective Copyingを導入することで性能向上
  - S4：構造化状態空間モデル
  - S6：構造化状態空間モデル＋Selective Copying

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.

Selection Mechanismの性能

- ✓ Mambaは系列長が極端に大きい場合にも性能劣化を起こさない
  - – Selection Mechanismが不要な系列を処理できる

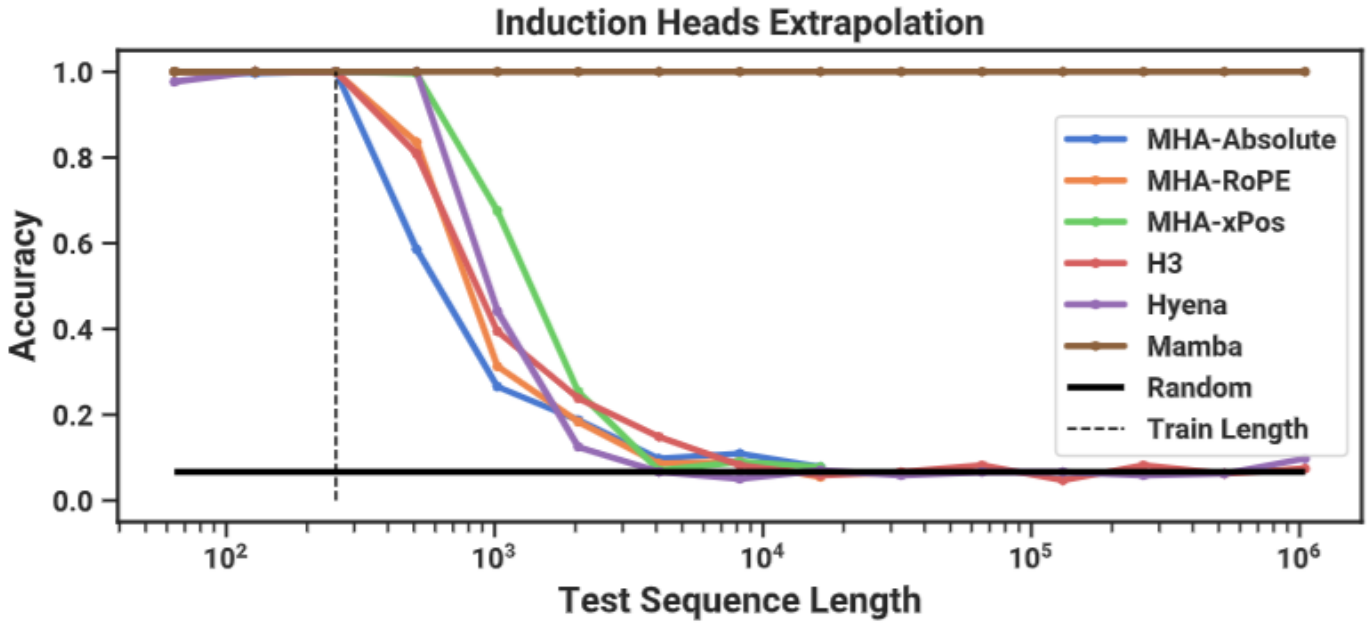


Table 2: (**Induction Heads.**) Models are trained on sequence length $2^8 = 256$, and tested on increasing sequence lengths of $2^6 = 64$ up to $2^{20} = 1048576$. Full numbers in Table 11.

Hardware-aware Algorithm｜Parallel Scan（1/2）

- ✓状態空間モデルの再帰計算をscanに代替して並列処理を実現

scan

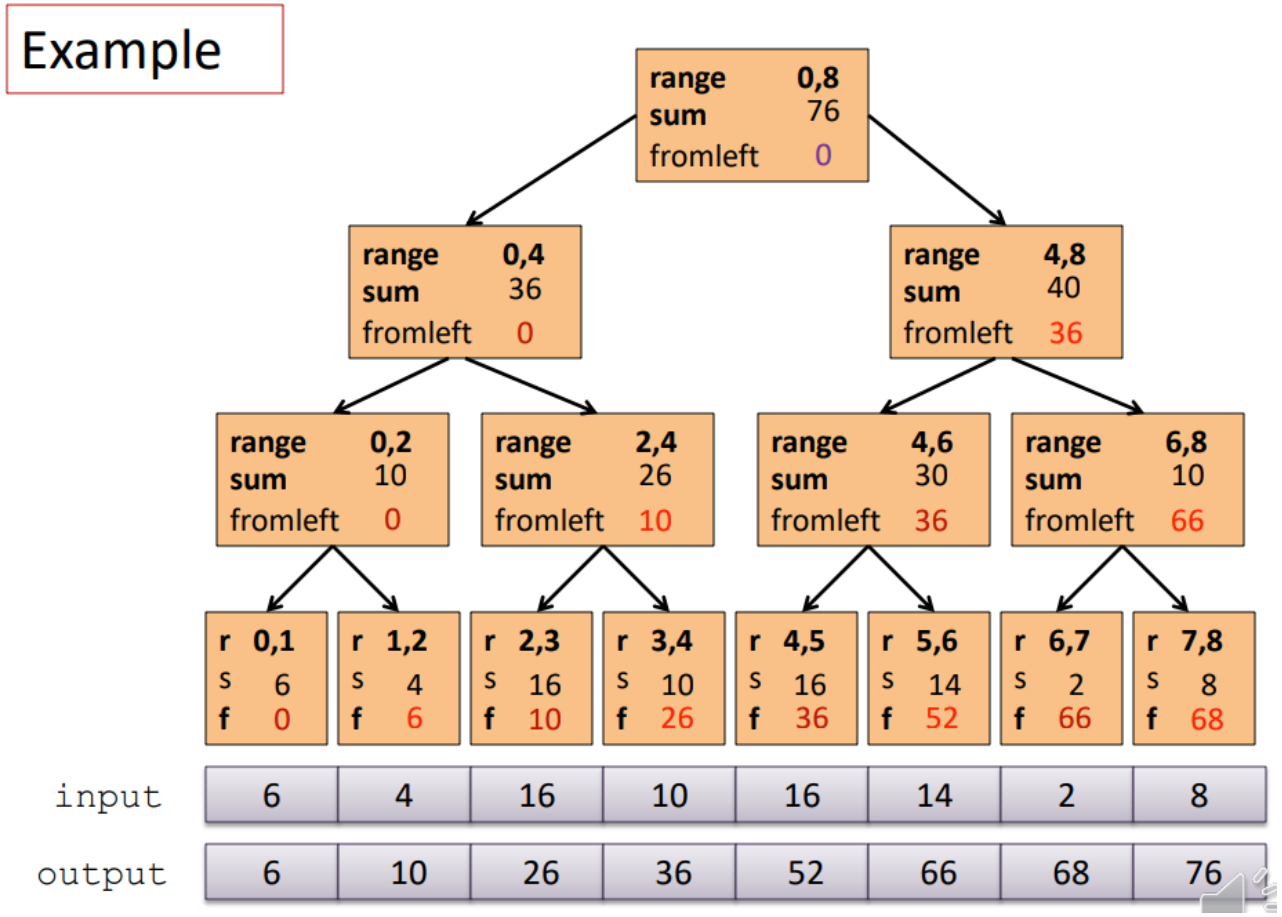
- ✓元はPostgreSQLの機能
  - – クエリを受け取り，どのテーブルからデータを処理するか解析
  - – テーブルの最初のブロックから順に読み込み，条件が一致するかを調査

Parallel Scan

- ✓元はPostgreSQL10の機能
  - – テーブル読み込みをマルチプロセスで行うためにデータの分割や割り当てを行う

Hardware-aware Algorithm｜Parallel Scan（2/2）

✓ MambaにおけるParallel Scanでは，入力をペアに分割して二分木で計算



※同じ階層は並列処理が可能

Hardware-aware Algorithm｜Kernel Fusionの前提

- ✓ モデル推論・学習のボトルネックはHBM，SRAM間のデータの移動
  - – HBM：　　DRAMの積層 ＝ 一般のストレージ
  - – SRAM：　　フリップフロップ回路 ＝ キャッシュメモリ
    - – 計算速度：　HBM　＜＜＜　SRAM
    - – 容量：　　HBM　＞＞＞　SRAM
  - ➡ 基本はHBMにデータを置き，頻繁に呼び出されるデータはSRAMで処理

- ✓ 通常のモデルの計算手順：入力 ➡ HBM ➡ SRAM ➡ HBM
  - – **QNAP2にデータを置いて計算しているようなもの**

Hardware-aware Algorithm｜Kernel Fusion

- ✓ Kernelを統合し呼び出し回数を削減
  - – Kernel：ソフトとハードをつなぐOS
  - ➡ メモリ節約

- ✓ 同時に，活性値の計算をSRAMで行うようにアーキテクチャを設計
  - – HBM，SRAM間の通信を最小限まで削減
  - ➡計算時間削減

Mambaの全体像

✓状態と係数行列AはSRAM上で演算を行うことで計算量削減

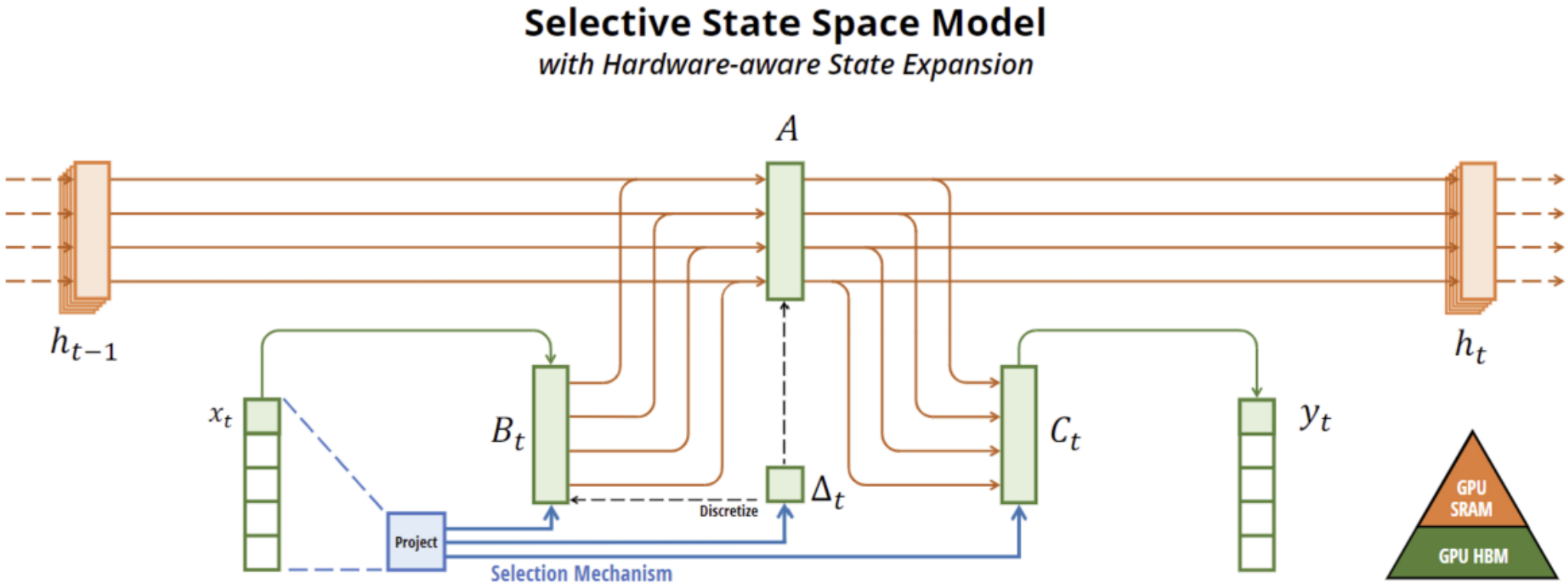


Figure 1: (**Overview.**) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate computation paths requiring time-invariance: the $(\Delta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ parameters are constant across time. Our selection mechanism adds back input-dependent dynamics, which also requires a careful hardware-aware algorithm to only materialize the expanded states in more efficient levels of the GPU memory hierarchy.

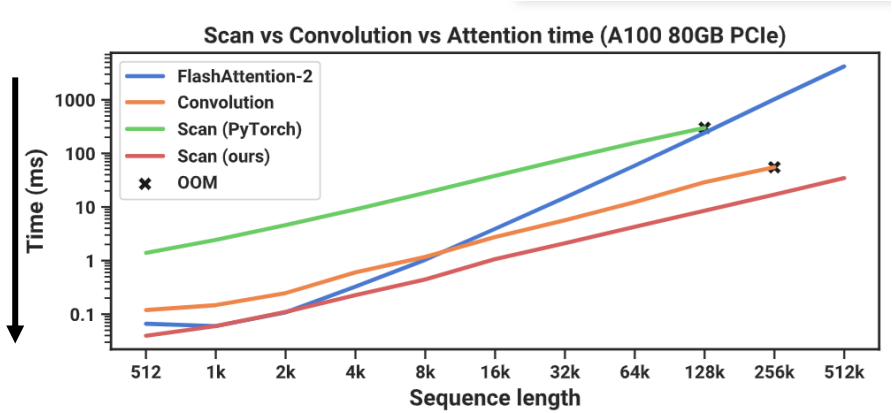
Mamba｜性能

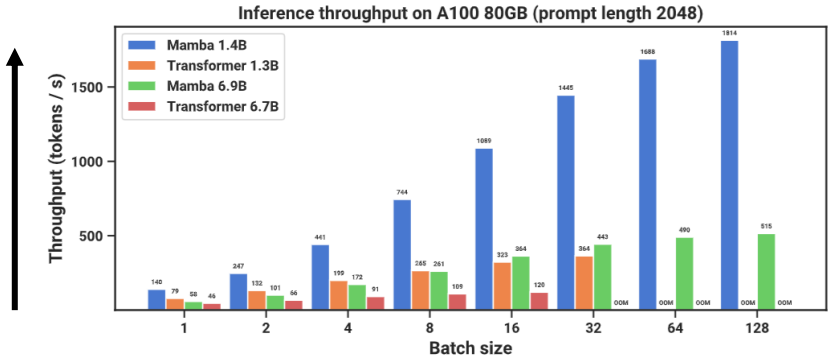
✓パラメータ数が同程度のモデル
より優れた性能

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

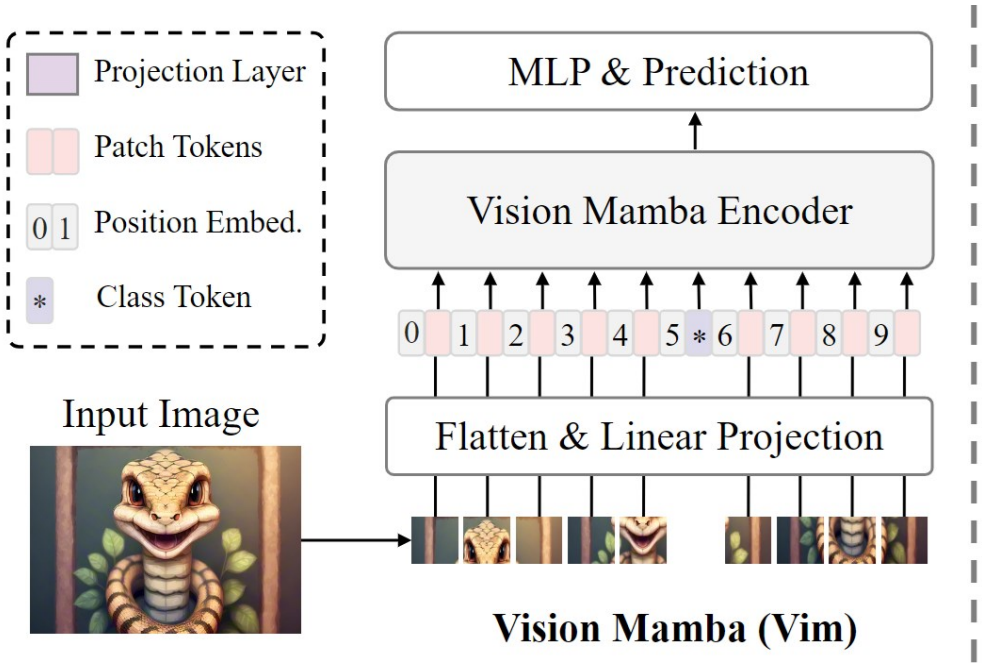
✓推論速度はTransformerの
4~5倍高速
特に系列長が大きい場合に顕著

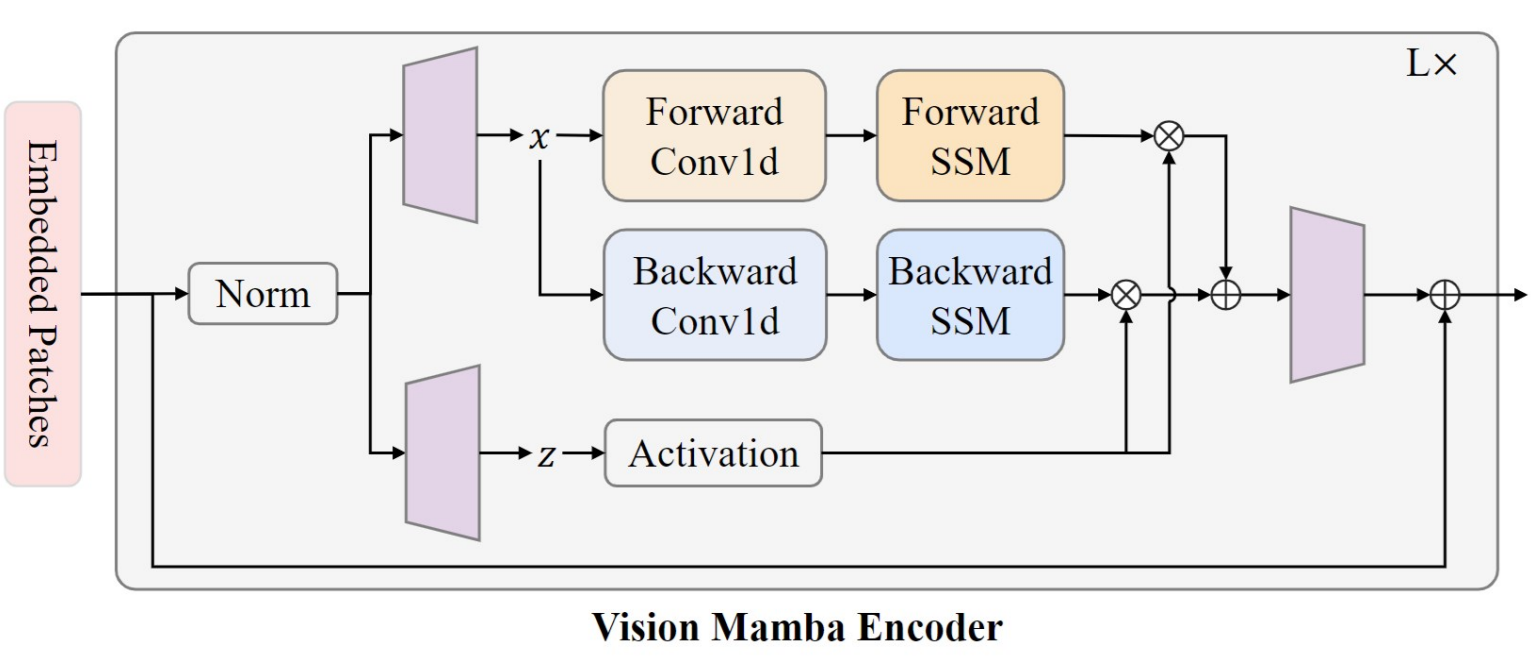




Vision Mamba: Efficient Visual Representation Learning with Bidirectional State Space Model

✓双方向状態空間モデルを活用して表現学習を行うことで,
高解像度画像の効率的な処理を可能に





Backward：後ろから前への状態遷移

Vision Mamba｜双方向状態空間モデル

- ✓前方向，後方向の状態遷移を行うことで前後の情報を考慮可能
  - 前方向，後方向の演算は同様の処理

前方向の状態遷移

$$\vec{h}_{t+1} = F\vec{h}_t + \vec{w}_t$$
$$\vec{y}_t = H\vec{h}_t + \vec{v}_t$$

F：前方向の状態遷移行列
H：前方向の観測行列
w：プロセスノイズ
v：観測ノイズ

後方向の状態遷移

$$\overleftarrow{h}_{t-1} = G\overleftarrow{h}_t + \overleftarrow{w}_t$$
$$\overleftarrow{y}_t = I\overleftarrow{h}_t + \overleftarrow{v}_t$$

G：後方向の状態遷移行列
I：後方向の観測行列
w：プロセスノイズ
v：観測ノイズ

Vision Mamba｜アルゴリズム

Algorithm 1 Vim Block Process

Require: token sequence $\mathbf{T}_{l-1}$: $(\mathtt{B},\mathtt{M},\mathtt{D})$
Ensure: token sequence $\mathbf{T}_{l}$: $(\mathtt{B},\mathtt{M},\mathtt{D})$

/* normalize the input sequence $\mathbf{T}'_{l-1}$ */
$\mathbf{T}'_{l-1}$: $(\mathtt{B},\mathtt{M},\mathtt{D}) \leftarrow \mathbf{Norm}(\mathbf{T}_{l-1})$ ← 正規化
$\mathbf{x}$: $(\mathtt{B},\mathtt{M},\mathtt{E}) \leftarrow \mathbf{Linear}^{\mathbf{x}}(\mathbf{T}'_{l-1})$
$\mathbf{z}$: $(\mathtt{B},\mathtt{M},\mathtt{E}) \leftarrow \mathbf{Linear}^{\mathbf{z}}(\mathbf{T}'_{l-1})$ ← 線形変換
/* process with different direction */
for o in {forward, backward} **do**
  $\mathbf{x}'_o$: $(\mathtt{B},\mathtt{M},\mathtt{E}) \leftarrow \mathbf{SiLU}(\mathbf{Conv1d}_o(\mathbf{x}))$
  $\mathbf{B}_o$: $(\mathtt{B},\mathtt{M},\mathtt{N}) \leftarrow \mathbf{Linear}^{\mathbf{B}}_o(\mathbf{x}'_o)$
  $\mathbf{C}_o$: $(\mathtt{B},\mathtt{M},\mathtt{N}) \leftarrow \mathbf{Linear}^{\mathbf{C}}_o(\mathbf{x}'_o)$ ← 1次元畳み込み，線形変換
  /* softplus ensures positive $\mathbf{\Delta}_o$ */
  $\mathbf{\Delta}_o$: $(\mathtt{B},\mathtt{M},\mathtt{E}) \leftarrow \log(1+\exp(\mathbf{Linear}^{\mathbf{\Delta}}_o(\mathbf{x}'_o)+\mathbf{Parameter}^{\mathbf{\Delta}}_o))$
  /* shape of $\mathbf{Parameter}^{\mathbf{A}}_o$ is $(\mathtt{E},\mathtt{N})$ */
  $\overline{\mathbf{A}}_o$: $(\mathtt{B},\mathtt{M},\mathtt{E},\mathtt{N}) \leftarrow \mathbf{\Delta}_o \bigotimes \mathbf{Parameter}^{\mathbf{A}}_o$
  $\overline{\mathbf{B}}_o$: $(\mathtt{B},\mathtt{M},\mathtt{E},\mathtt{N}) \leftarrow \mathbf{\Delta}_o \bigotimes \mathbf{B}_o$
  $\mathbf{y}_o$: $(\mathtt{B},\mathtt{M},\mathtt{E}) \leftarrow \mathbf{SSM}(\overline{\mathbf{A}}_o,\overline{\mathbf{B}}_o,\mathbf{C}_o)(\mathbf{x}'_o)$ ← SSM
end for
/* get gated $\mathbf{y}_o$ */
$\mathbf{y}'_{forward}$: $(\mathtt{B},\mathtt{M},\mathtt{E}) \leftarrow \mathbf{y}_{forward} \odot \mathbf{SiLU}(\mathbf{z})$
$\mathbf{y}'_{backward}$: $(\mathtt{B},\mathtt{M},\mathtt{E}) \leftarrow \mathbf{y}_{backward} \odot \mathbf{SiLU}(\mathbf{z})$ ← ゲーティング
/* residual connection */
$\mathbf{T}_l$: $(\mathtt{B},\mathtt{M},\mathtt{D}) \leftarrow \mathbf{Linear}^{\mathbf{T}}(\mathbf{y}'_{forward}+\mathbf{y}'_{backward})+\mathbf{T}_{l-1}$ ← 残差接続
Return: $\mathbf{T}_l$

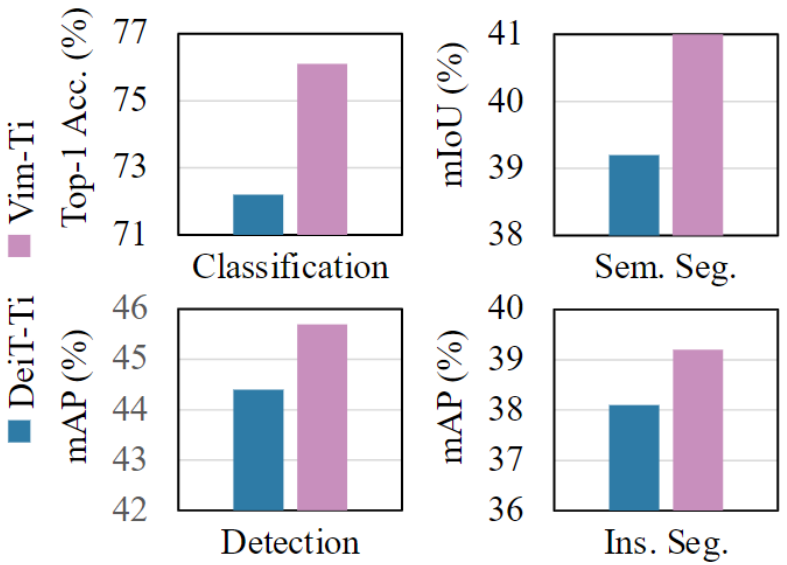
D：隠れ層の次元，E：拡張状態次元，N：SSMの次元

目的

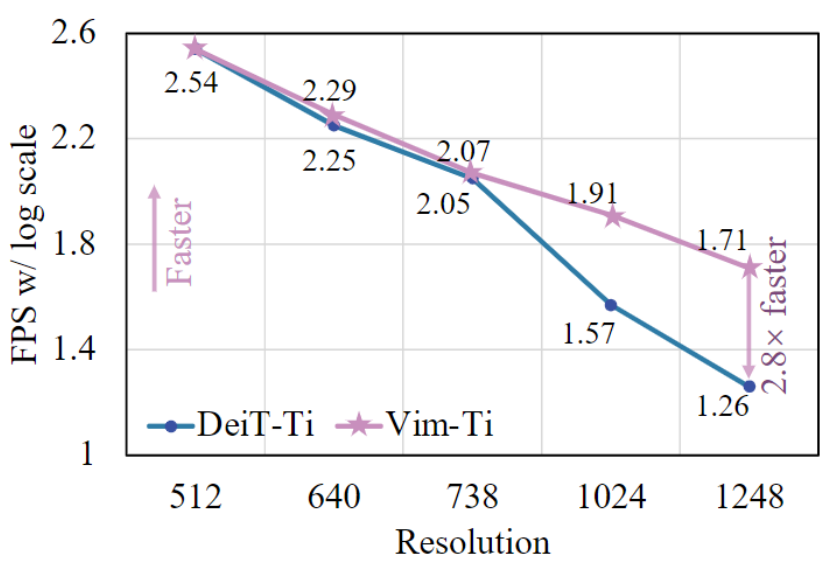
- ✓位置認識の強化
  - 5-16行目で双方向の系列情報を処理し，統合
- ✓ゲーティング
  - 17-21行目でSiLUを用いて不要な系列情報を排除
    - $\mathrm{SiLU}(x) = x \times \sigma(x)$
      σ：シグモイド関数
- ✓前方向，後方向の状態遷移は線形和＋残差接続で統合

Vision Mamba｜性能

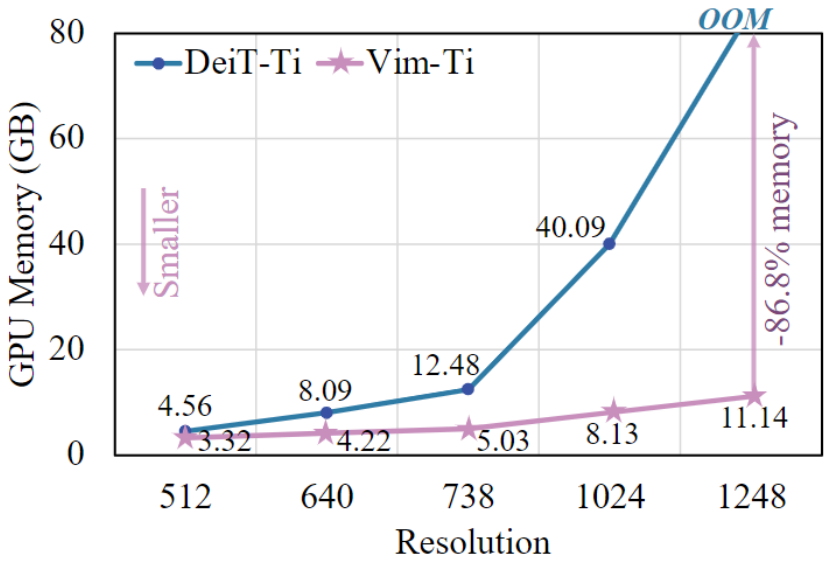
✓DeiT(Data-efficient image Transformer)と比べて高性能かつ高効率



(a) Accuracy Comparison



(b) Speed Comparison

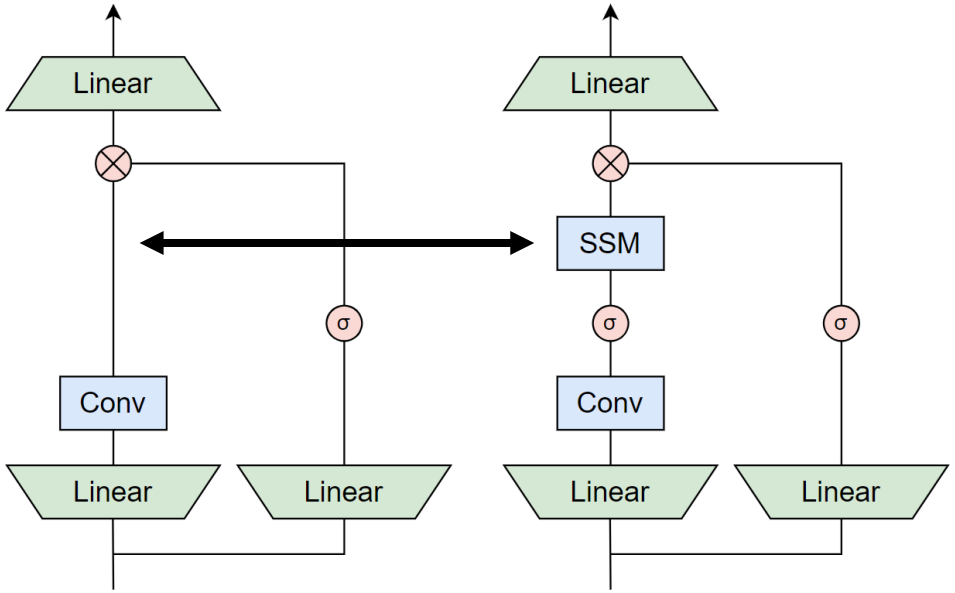


(c) GPU Memory Comparison

Vision Mambaは
巨大物体も認識

Mambaが求められる場面（1/2）

- ✓ MambaからSSMを外して性能比較を行う（**MambaOut**）
  - クラス認識はSSMが**ない**方が性能が高い
  - 物体検出やセグメンテーションではSSMが**ある**方が性能が高い



Model	Token Mixing Type	Param (M)	Test@224^2 MAC (G)	Test@224^2 Acc (%)
VAN-B0 [28]	Conv	4	0.9	75.4
MogaNet-T [45]	Conv	5	1.1	**79.0**
FasterNet-T1 [7]	Conv	8	0.9	76.2
InceptionNeXt-A [93]	Conv	4	0.5	75.3
DeiT-Ti [73]	Attn	6	1.3	72.2
T2T-ViT-7 [94]	Attn	4	1.1	71.7
PVTv2-B0 [80]	Conv + Attn	3	0.6	70.5
MobileViTv3-XS [77]	Conv + Attn	3	0.9	76.7
EMO-6M [101]	Conv + Attn	6	1.0	**79.0**
Vim-Ti [104]	Conv + SSM	7	1.5	76.1
LocalVim-T [37]	Conv + SSM	8	1.5	76.2
EfficientVMamba-T [58]	Conv + SSM	6	0.8	76.5
EfficientVMamba-S [58]	Conv + SSM	11	1.3	78.7
MambaOut-Femto	Conv	7	1.2	**78.9**

ImageNet

Backbone	1× sche AP^b	1× sche AP^m
ConvNeXt-T [49]	44.2	40.1
FocalNet-T [89]	46.1	41.5
Swin-T [51]	42.7	39.3
ViT-Adapter-S [10]	44.7	39.9
CSWin-T [22]	46.7	42.2
PVTv2-B2 [80]	45.3	41.2
SG-Former-S [65]	47.4	42.6
TransNeXt-Tiny [69]	**49.9**	**44.6**
VMamba-T [50]	46.5	42.1
LocalVMamba-T [37]	46.7	42.2
EfficientVMamba-B [58]	43.7	40.2
VMambaV9-T [50]	**47.4**	**42.7**
PlainMamba-L1 [88]	44.1	39.1
MambaOut-Tiny	45.1	41.0

COCO（Detection）

Mambaが求められる場面（2/2）

クラス認識で性能が上がらない理由

- ✓ 系列長が小さく，自己回帰が不要の為
  - – 無駄に計算量が増える

Mambaが求められる場面

- ✓ 系列長が大きい場合
  - – 複雑なCVタスクはピクセル単位の理解が必要 → 系列長が大きいとみなせる
- ✓ 自己回帰タスク
  - – Decoder型の言語生成など

現状，Transformerのように“とりあえず”で使うものではない

まとめ

- ✓ Mambaは選択的状態空間モデルを用いた新しいアーキテクチャ
  - 状態を動的に取捨選択できるため**系列長のスケーリング性能が顕著**
  - 同サイズのTransformerと比べると高速で高精度

- ✓ 様々な派生が登場中
  - Swin-UMamba：　UNet
  - Graph-Mamba：　GNN
  - BlackMamba：　MoE（Mixture of Experts）
  - Cobra：　マルチモーダル
  - Gamba：　3D Gaussian Splatting