State-space Models for Deep Sequence Modeling

손형욱

2024-02-02

hyounguk.shon@kaist.ac.kr

KAIST

Statistical Inference and Information Theory Laboratory

RNNs are beating Transformers

- SSM models are competing SOTA **without attention mechanism**

Survey: Progress on Attention Alternatives

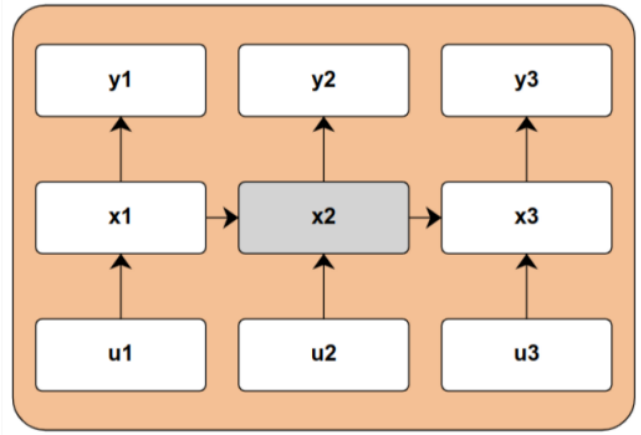
Recent research has made significant progress.

S4 [Gu et al., 2022a]
DSS [Gupta, 2022]
GSS [Mehta et al., 2022]
S4D [Gu et al., 2022b]
H3 [Dao et al., 2022]
S5 [Smith et al., 2022]
BiGS [Wang et al., 2022]

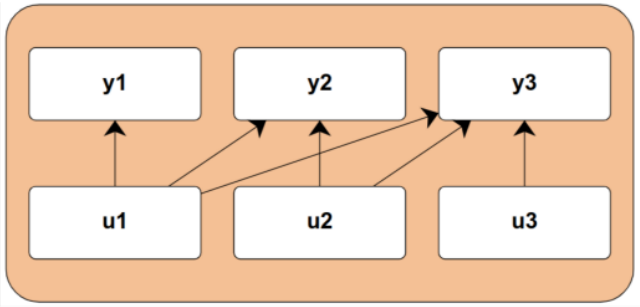
QRNN [Bradbury et al., 2016]
Mega [Ma et al., 2022]
SGConv [Li et al., 2022]
Hyena [Poli et al., 2023]
LRU [Orvieto et al., 2023]
RWKV [Peng et al., 2023]
MultiRes [Shi et al., 2023]

RNNs are beating Transformers

- Attention can’t scale to longer sequences
- RNNs scale to long sequence due to constant memory

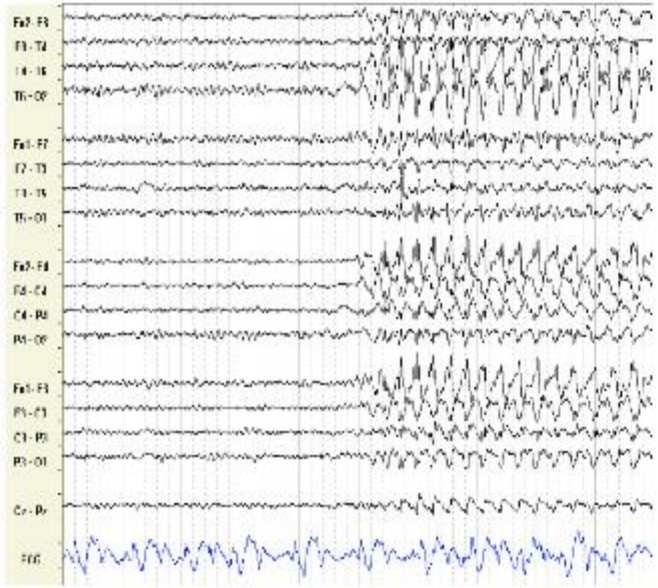


attention does not compress the context

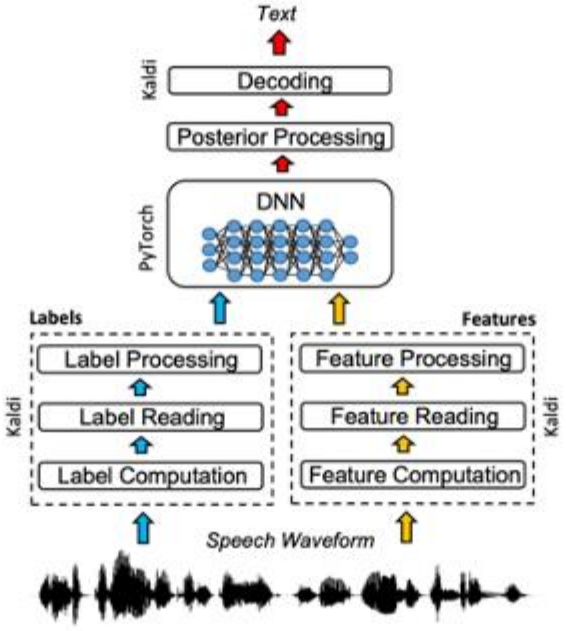


- Training Speed: Slow (Serial bottleneck)
- Generation Speed: Fast (constant-time per step)

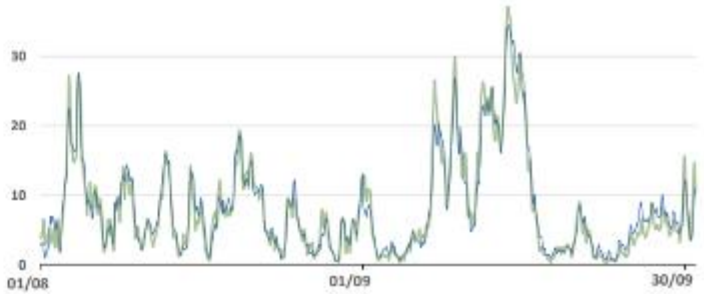
RNNs are beating Transformers



EEG/ECG



Audio



Energy Forecasting

Recent history of SSM models

- [HiPPO] Gu, Albert, et al. "Hippo: Recurrent memory with optimal polynomial projections." *NeurIPS 2020*
- [LSSL] Gu, Albert, et al. "Combining recurrent, convolutional, and continuous-time models with linear state space layers." *NeurIPS 2020*
- [S4] Gu, Albert, Karan Goel, and Christopher Re. "Efficiently Modeling Long Sequences with Structured State Spaces." *ICLR* 2021
- [S4D] Gu, Albert, et al. "On the parameterization and initialization of diagonal state space models." *NeurIPS 2022*
- [H3] Fu, Daniel Y., et al. "Hungry Hungry Hippos: Towards Language Modeling with State Space Models." *ICLR* 2022
- [Mamba] Gu, Albert, and Tri Dao. "Mamba: Linear-time sequence modeling with selective state spaces." *arXiv preprint arXiv:2312.00752* (2023).

S4 = LSSL + HiPPO + structured matrix

S4D = S4 + diagonal approximation

H3 = S4D + gating + FlashConv

Mamba = H3 + parallel scan

1. Efficient Models

2. Effective Long-Range Parameterizations

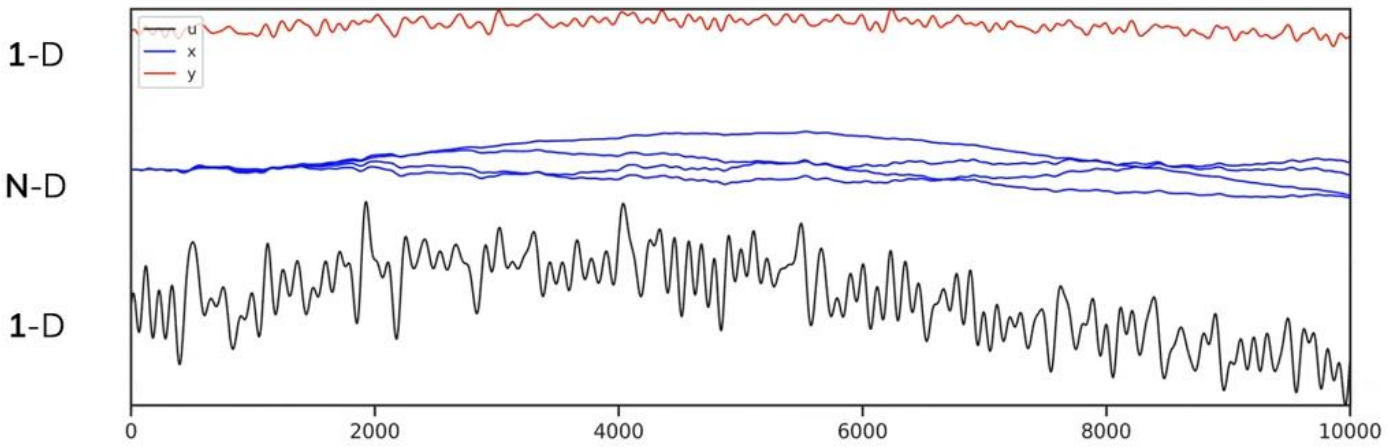
State-space Models (SSM)

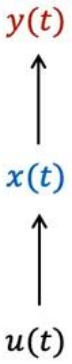
- 제어공학 (control engineering), Kalman filter
- 상태 변수 state variable x(t)
- 입력 변수 input variable u(t)
- 출력 변수 output variable y(t)

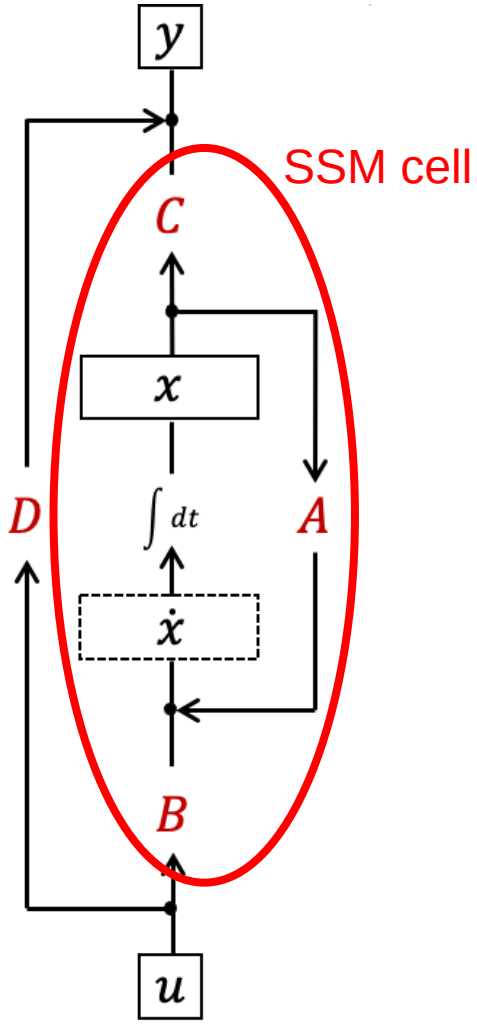
SSM

$$x' = \boldsymbol{A}x + \boldsymbol{B}u$$

$$y = \boldsymbol{C}x + \boldsymbol{D}u$$







State-space Models (SSM)

Continuous-time form

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

Discrete-time form

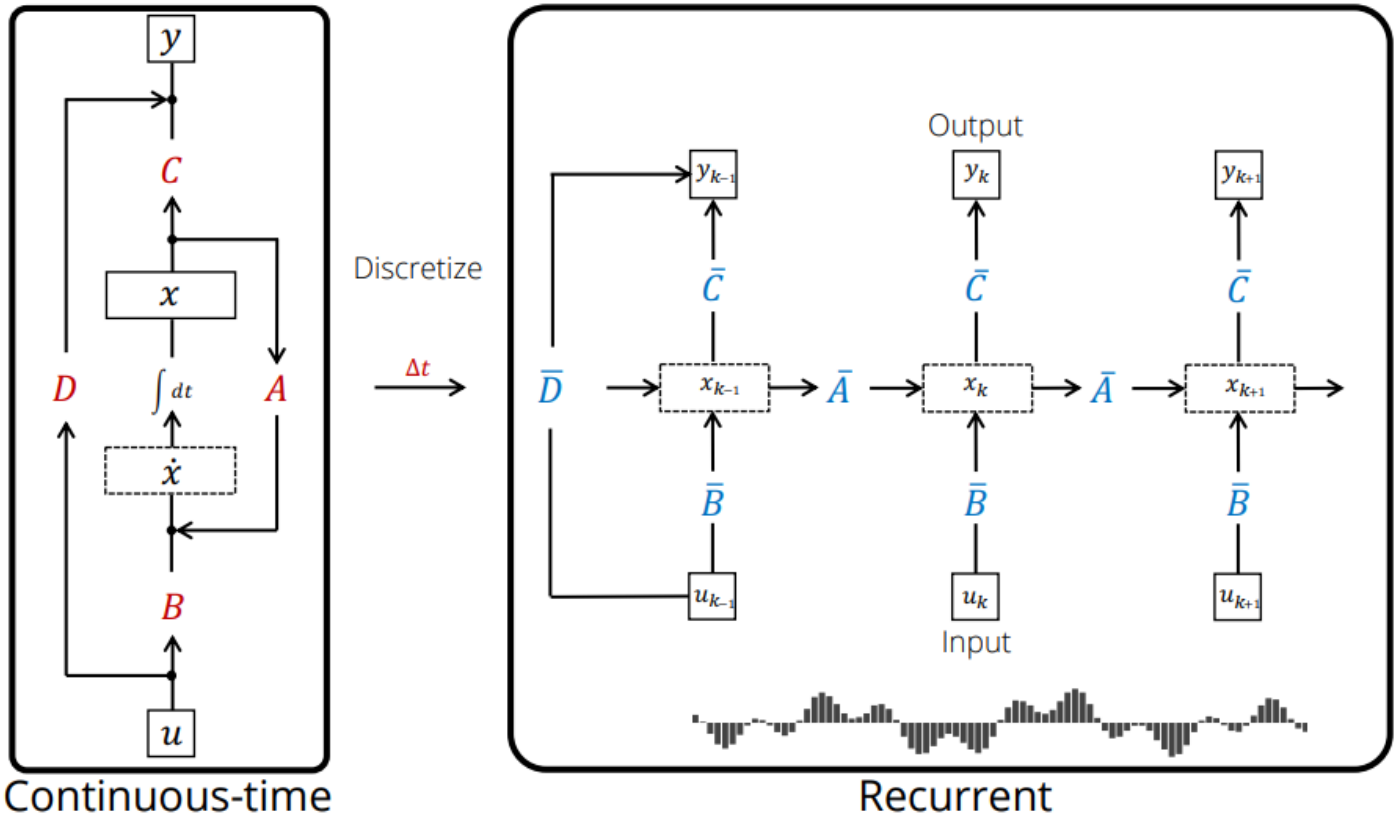
$$x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k$$
$$y_k = \overline{\boldsymbol{C}} x_k + \overline{\boldsymbol{D}} u_k$$

$$\overline{\boldsymbol{A}} = (\boldsymbol{I} - \frac{\Delta}{2} \cdot \boldsymbol{A})^{-1}(\boldsymbol{I} + \frac{\Delta}{2} \cdot \boldsymbol{A})$$
$$\overline{\boldsymbol{B}} = (\boldsymbol{I} - \frac{\Delta}{2} \cdot \boldsymbol{A})^{-1}\Delta \boldsymbol{B}$$
$$\overline{\boldsymbol{C}} = \boldsymbol{C}$$
$$\overline{\boldsymbol{D}} = \boldsymbol{D}$$

Learnable parameters: A, B, C, D, Δ

State-space Models (SSM)

- Discrete-time SSM is a (time-linear) RNN



SSM

$$x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k$$
$$y_k = \overline{\boldsymbol{C}} x_k$$

RNN

$$x_k = \sigma(\overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k)$$
$$y_k = \overline{\boldsymbol{C}} x_k$$

High-order Polynomial Projection (HiPPO)

- HiPPO is a special memory mechanism that efficiently compresses a long signal $u(t)$
- Key idea: Approximate the signal by a combination of Legendre polynomials.
- The state vector $x(t)$ memorizes the coefficients of basis functions.

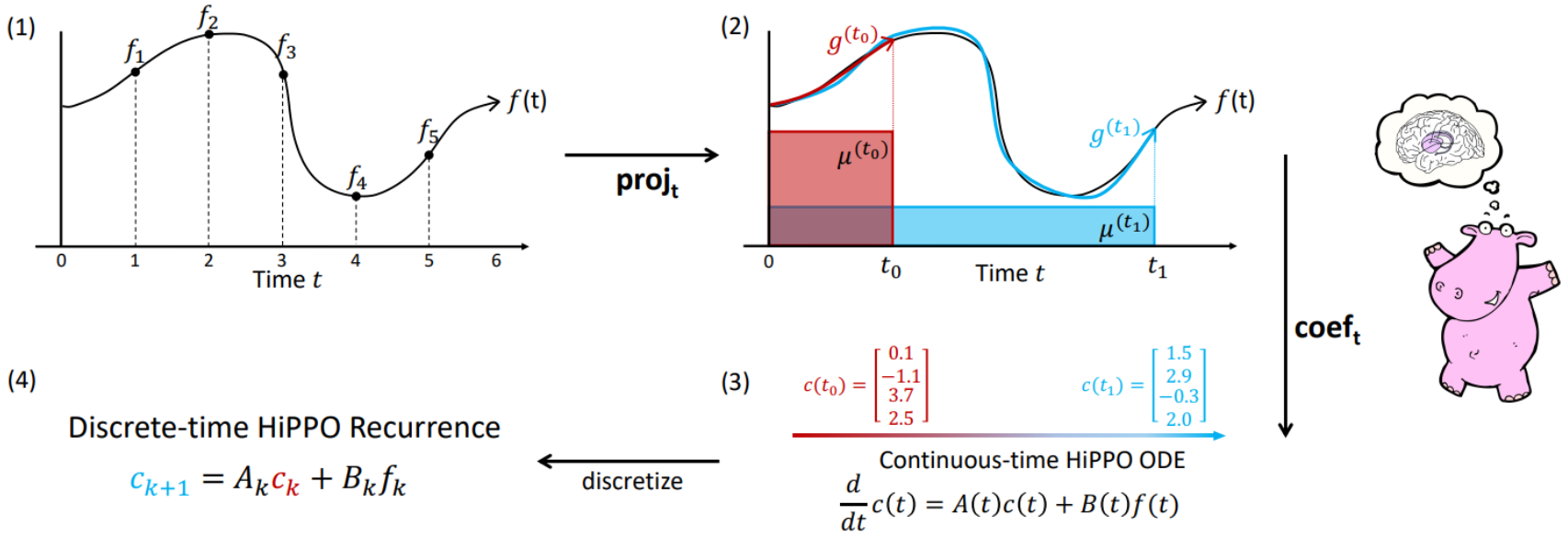


Figure 1: **Illustration of the HiPPO framework.** (1) For any function f, (2) at every time t there is an optimal projection $g^{(t)}$ of f onto the space of polynomials, with respect to a measure $\mu^{(t)}$ weighing the past. (3) For an appropriately chosen basis, the corresponding coefficients $c(t) \in \mathbb{R}^N$ representing a compression of the history of f satisfy linear dynamics. (4) Discretizing the dynamics yields an efficient closed-form recurrence for online compression of time series $(f_k)_{k \in \mathbb{N}}$.

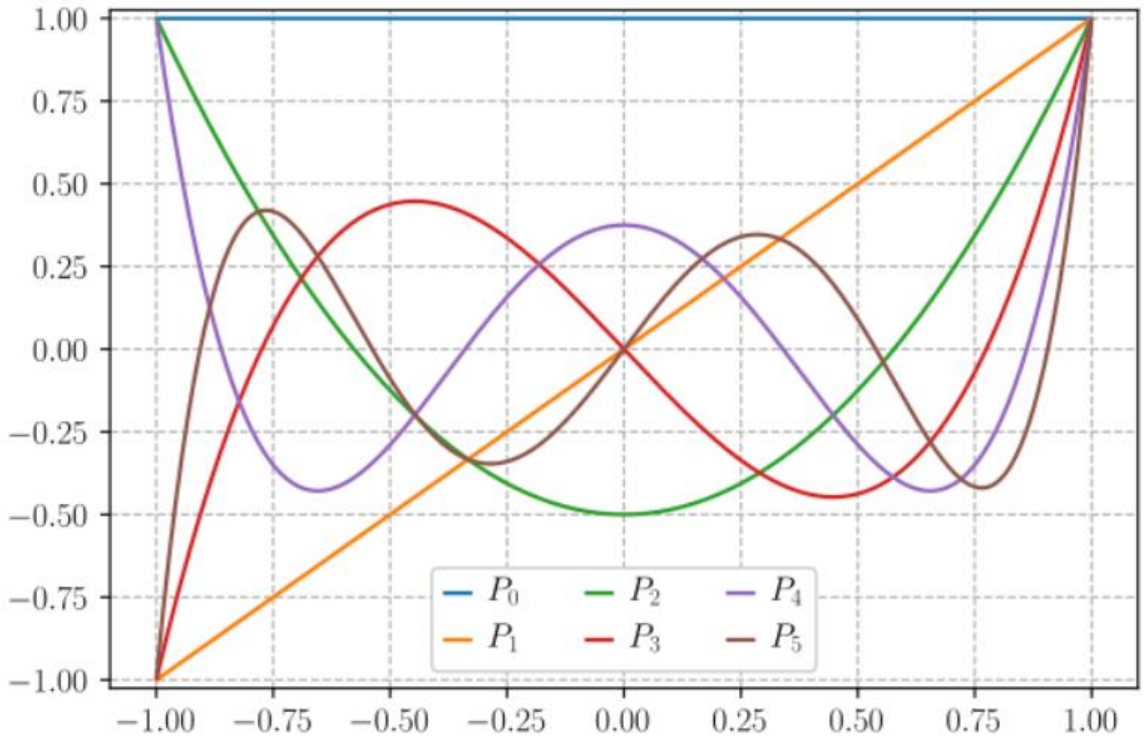
High-order Polynomial Projection (HiPPO)

- Legendre polynomials as orthogonal basis functions
- Use the state variable $x(t)$ as a memory of the coefficients

The first few Legendre polynomials are:

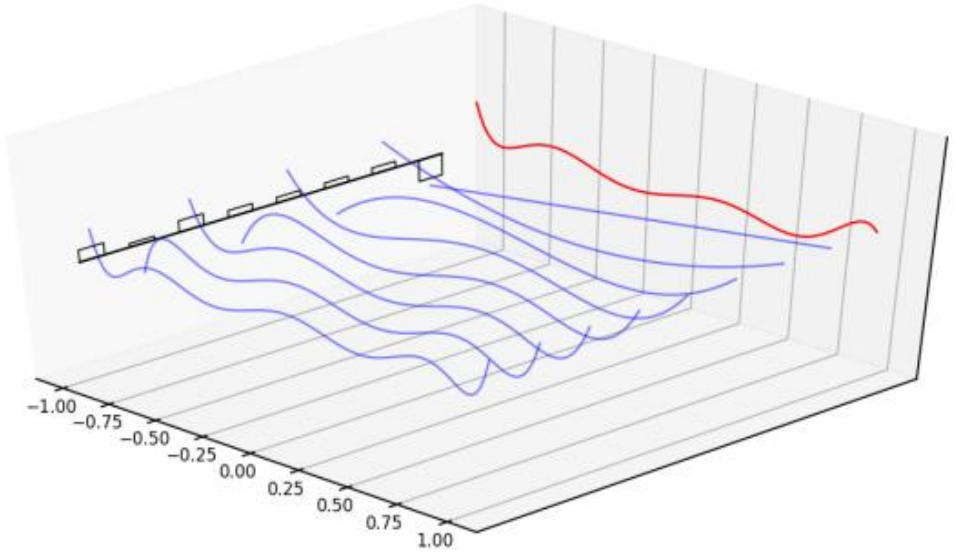
n	$P_n(x)$
0	1
1	x
2	$\frac{1}{2}\left(3x^2-1\right)$
3	$\frac{1}{2}\left(5x^3-3x\right)$
4	$\frac{1}{8}\left(35x^4-30x^2+3\right)$
5	$\frac{1}{8}\left(63x^5-70x^3+15x\right)$
6	$\frac{1}{16}\left(231x^6-315x^4+105x^2-5\right)$
7	$\frac{1}{16}\left(429x^7-693x^5+315x^3-35x\right)$
8	$\frac{1}{128}\left(6435x^8-12012x^6+6930x^4-1260x^2+35\right)$
9	$\frac{1}{128}\left(12155x^9-25740x^7+18018x^5-4620x^3+315x\right)$
10	$\frac{1}{256}\left(46189x^{10}-109395x^8+90090x^6-30030x^4+3465x^2-63\right)$

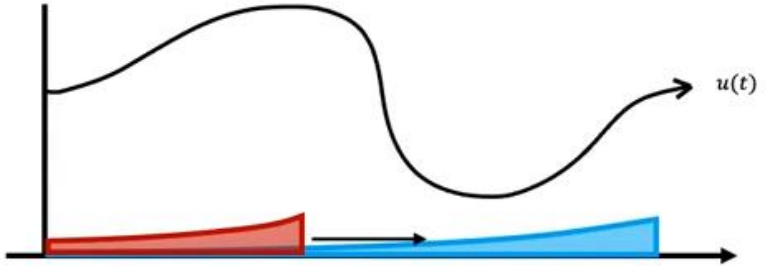
The graphs of these polynomials (up to $n=5$) are shown below:



High-order Polynomial Projection (HiPPO)

- HiPPO is the on-line update formula for the coefficient vector $x(t)$
- vector $x(t)$ stores the compressed representation of the input context





HiPPO operator

$$x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t)$$

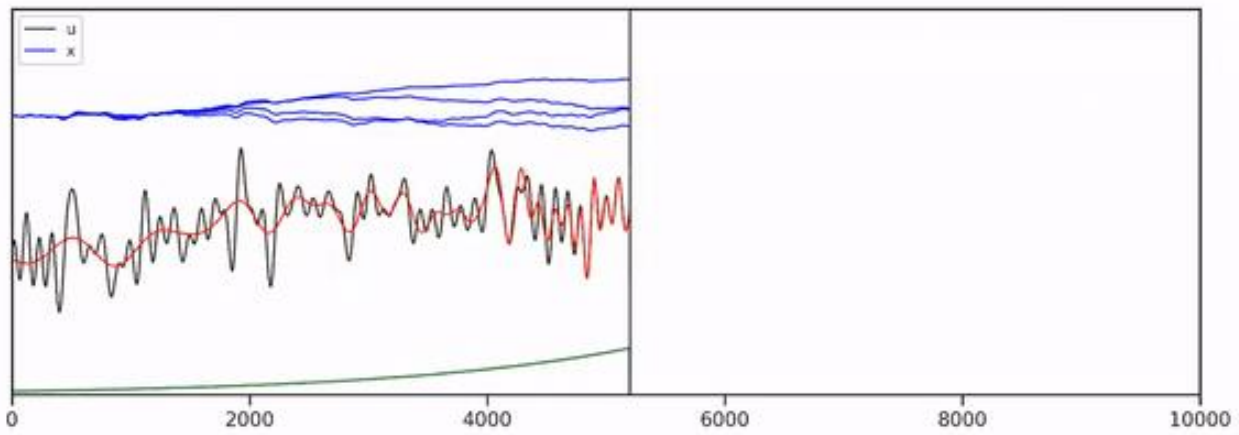
HiPPO matrix

$$\boldsymbol{A}_{nk} = \begin{cases} 0 & n < k \\ n+1 & n = k \\ 2n+1 & n > k \end{cases}$$

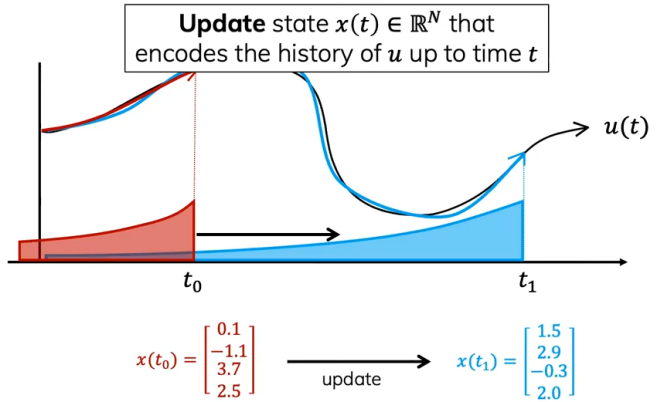
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 3 & 3 & 0 \\ 1 & 3 & 5 & 4 \end{bmatrix}$$

High-order Polynomial Projection (HiPPO)

HIPPO: Online Function Reconstruction



Encode history of length **10000** sequence by state of size **64**

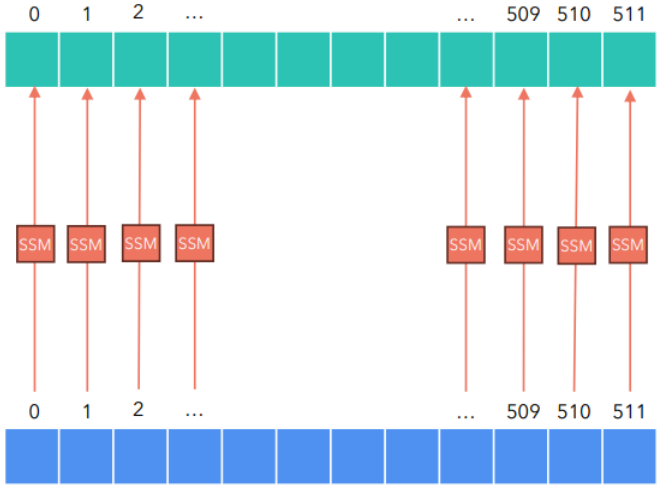


Structured State-space Model (S4)

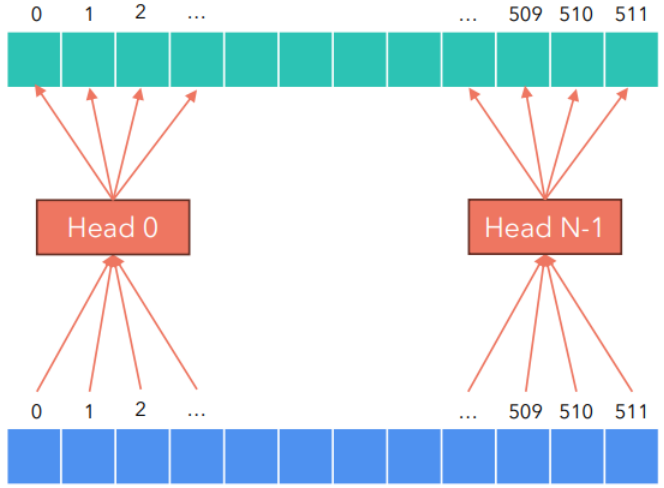
From 1 dimension to multiple dimensions

The state space model that we have studied so far, only computes one output (so only one number) for each input token (which is also represented by a single number). How can we work when the input/output signal is a vector?

State Space Model: Each dimension is managed by an independent state space model!



Transformer: Each group of dimensions dimension is managed by a different head of the multi-head attention!



Output

Input

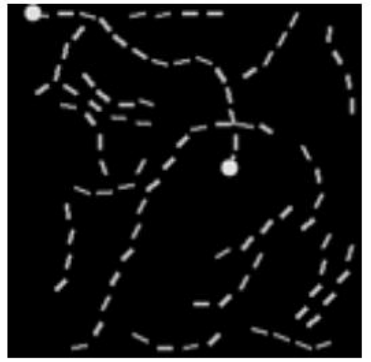
Of course we can parallelize all these operations by working on batches of input. This way, **the parameters A, B, C, D and the input x(t) and y(t) become vectors and matrices**. This way, the computation will be done in parallel for all the dimensions.

Umar Jamil - https://github.com/hkproj/mamba-notes

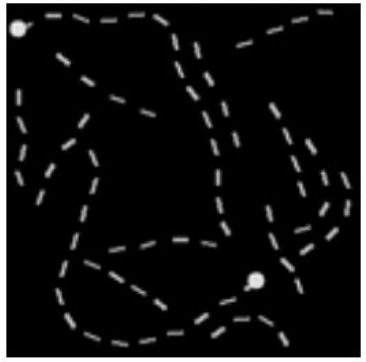
Structured State-space Model (S4)

Table 10: Full results for the Long Range Arena (LRA) benchmark for long-range dependencies in sequence models. (Top): Original Transformer variants in LRA. (Bottom): Other models reported in the literature.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Random	10.00	50.00	50.00	10.00	50.00	50.00	36.67
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Local Attention	15.82	52.98	53.39	41.46	66.63	✗	46.71
Sparse Trans.	17.07	63.58	59.59	44.24	71.71	✗	51.03
Longformer	35.63	62.85	56.89	42.22	69.71	✗	52.88
Linformer	35.70	53.94	52.27	38.56	76.34	✗	51.14
Reformer	37.27	56.10	53.40	38.07	68.50	✗	50.56
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	✗	51.23
Synthesizer	36.99	61.68	54.67	41.61	69.45	✗	52.40
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	65.90	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	77.80	✗	54.42
Nyströmformer	37.15	65.52	79.56	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	47.38	77.72	✗	59.37
S4 (original)	58.35	76.02	87.09	87.26	86.05	88.10	80.48
S4 (updated)	**59.60**	**86.82**	**90.90**	**88.65**	**94.20**	**96.35**	**86.09**



(a) A positive example.



(b) A negative example.

Figure 1: Samples of the Pathfinder task.

Structured State-space Model (S4)

- General sequence tasks

Table 5: (**SC10 classification**) Transformer, CTM, RNN, CNN, and SSM models. (*MFCC*) Standard preprocessed MFCC features (length 161). (*Raw*) Unprocessed signals (length 16000). (*0.5×*) Frequency change at test time. ✗ denotes not applicable or computationally infeasible on single GPU. *Please read Appendix D.3 before citing this table.*

	MFCC	Raw	0.5×
Transformer	90.75	✗	✗
Performer	80.85	30.77	30.68
ODE-RNN	65.9	✗	✗
NRDE	89.8	16.49	15.12
ExpRNN	82.13	11.6	10.8
LipschitzRNN	88.38	✗	✗
CKConv	**95.3**	71.66	65.96
WaveGAN-D	✗	96.25	✗
LSSL	93.58	✗	✗
S4	93.96	**98.32**	**96.30**

Table 6: (**Pixel-level 1-D image classification**) Comparison against reported test accuracies from prior works (Transformer, RNN, CNN, and SSM models). Extended results and citations in Appendix D.

	sMNIST	pMNIST	sCIFAR
Transformer	98.9	97.9	62.2
LSTM	98.9	95.11	63.01
r-LSTM	98.4	95.2	72.2
UR-LSTM	99.28	96.96	71.00
UR-GRU	99.27	96.51	74.4
HiPPO-RNN	98.9	98.3	61.1
LMU-FFT	-	98.49	-
LipschitzRNN	99.4	96.3	64.2
TCN	99.0	97.2	-
TrellisNet	99.20	98.13	73.42
CKConv	99.32	98.54	63.74
LSSL	99.53	**98.76**	84.65
S4	**99.63**	98.70	**91.13**

Table 7: (**CIFAR-10 density estimation**) As a generic sequence model, S4 is competitive with previous autoregressive models (in bits per dim.) while incorporating no 2D inductive bias, and has fast generation through its recurrence mode.

Model	bpd	2D bias	Images / sec
Transformer	3.47	**None**	0.32 (1×)
Linear Transf.	3.40	**None**	17.85 (56×)
PixelCNN	3.14	2D conv.	-
Row PixelRNN	3.00	2D BiLSTM	-
PixelCNN++	2.92	2D conv.	19.19 (59.97×)
Image Transf.	2.90	2D local attn.	0.54 (1.7×)
PixelSNAIL	2.85	2D conv. + attn.	0.13 (0.4×)
Sparse Transf.	**2.80**	2D sparse attn.	-
S4 (base)	2.92	**None**	**20.84 (65.1×)**
S4 (large)	2.85	**None**	3.36 (10.5×)

Table 8: (**WikiText-103 language modeling**) S4 approaches the performance of Transformers with much faster generation. (*Top*) Transformer baseline which our implementation is based on, with attention replaced by S4. (*Bottom*) Attention-free models (RNNs and CNNs).

Model	Params	Test ppl.	Tokens / sec
Transformer	247M	**20.51**	0.8K (1×)
GLU CNN	229M	37.2	-
AWD-QRNN	151M	33.0	-
LSTM + Hebb.	-	29.2	-
TrellisNet	180M	29.19	-
Dynamic Conv.	255M	25.0	-
TaLK Conv.	240M	23.3	-
S4	249M	**20.95**	**48K (60×)**

Mamba

- Token selection mechanism (gating)
- Fast GPU-friendly implementation (~FlashAttention)

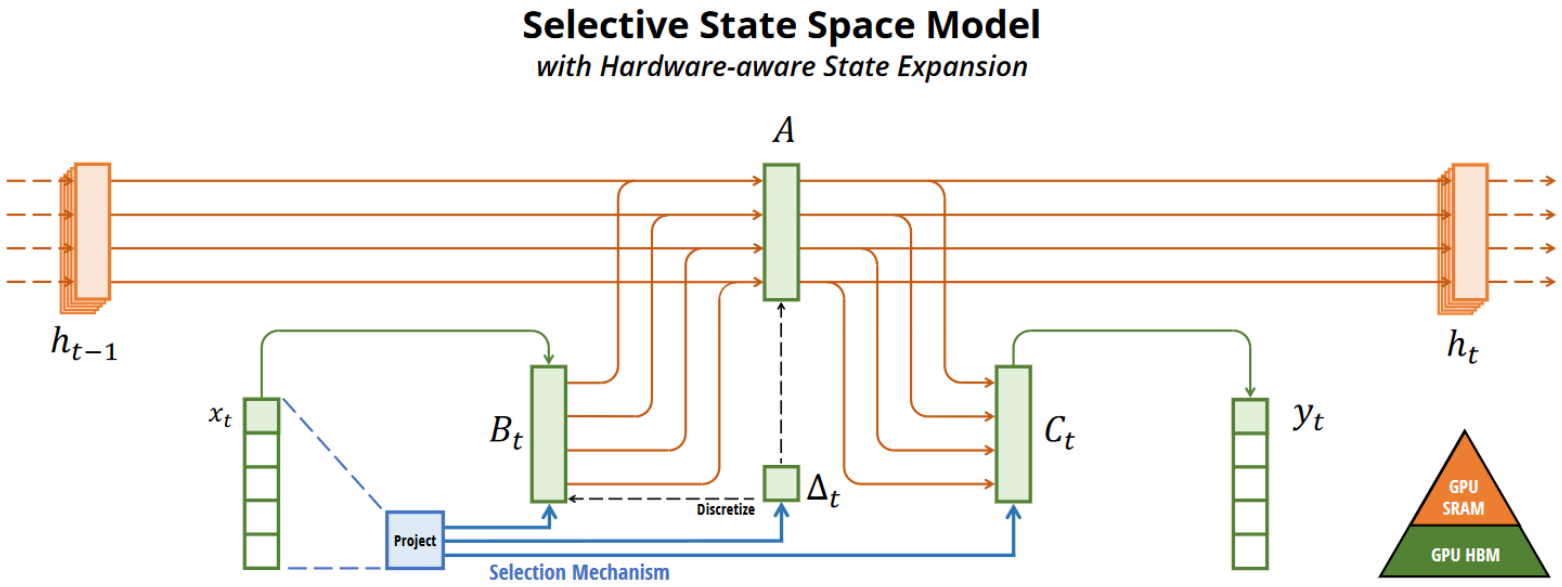


Figure 1: (**Overview**.) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate 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.

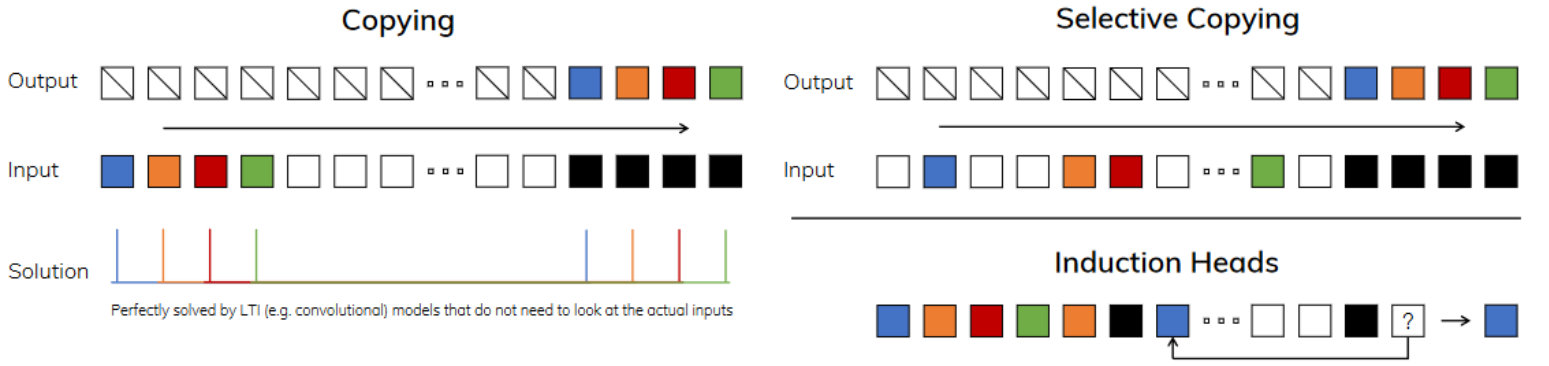
Mamba

Table 1: Synthetic language modeling tasks.

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

Correlates to in-context learning ability

- Selection mechanism
  - Solving the “Induction Heads” task
  - Make the SSM matrices dependent to x_t. (time-varying)
  - Cannot use the “convolution trick” because now the SSM is not LTI



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

*표기법이 바뀜 (x=인풋, h=state)

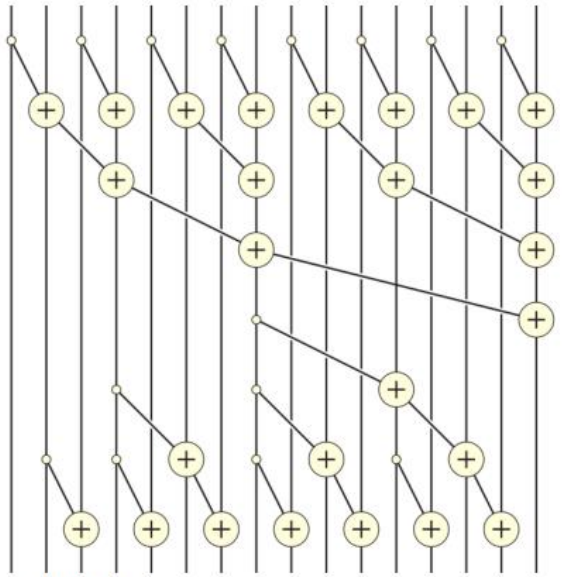
B: batch size
L: seq length
D: input channels
N: state size

Mamba

- Fast state computation using Parallel scan operation
- $x_t \rightarrow h_t$ is a scan-like operation: $h_t = \overline{\boldsymbol{A}} h_{t-1} + \overline{\boldsymbol{B}} x_t$

Initial array x_t

x_1	x_2	x_3	x_4	x_5	x_6
9	6	7	10	8	7



Source: Wikipedia

Prefix-Sum h_t

9	15	22	32	40	47
h_1	h_2	h_3	h_4	h_5	h_6

$$x_0 = \overline{\boldsymbol{B}} u_0 \quad x_1 = \overline{\boldsymbol{AB}} u_0 + \overline{\boldsymbol{B}} u_1 \quad x_2 = \overline{\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{AB}} u_1 + \overline{\boldsymbol{B}} u_2 \quad \dots$$

$$y_0 = \overline{\boldsymbol{CB}} u_0 \quad y_1 = \overline{\boldsymbol{CAB}} u_0 + \overline{\boldsymbol{CB}} u_1 \quad y_2 = \overline{\boldsymbol{CA}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CAB}} u_1 + \overline{\boldsymbol{CB}} u_2 \quad \dots$$

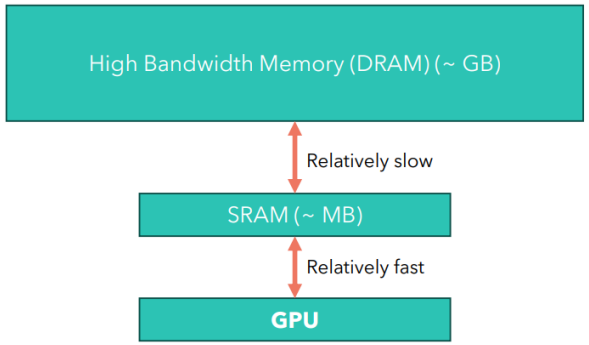
$$y_k = \overline{\boldsymbol{CA}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CA}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{CAB}} u_{k-1} + \overline{\boldsymbol{CB}} u_k$$

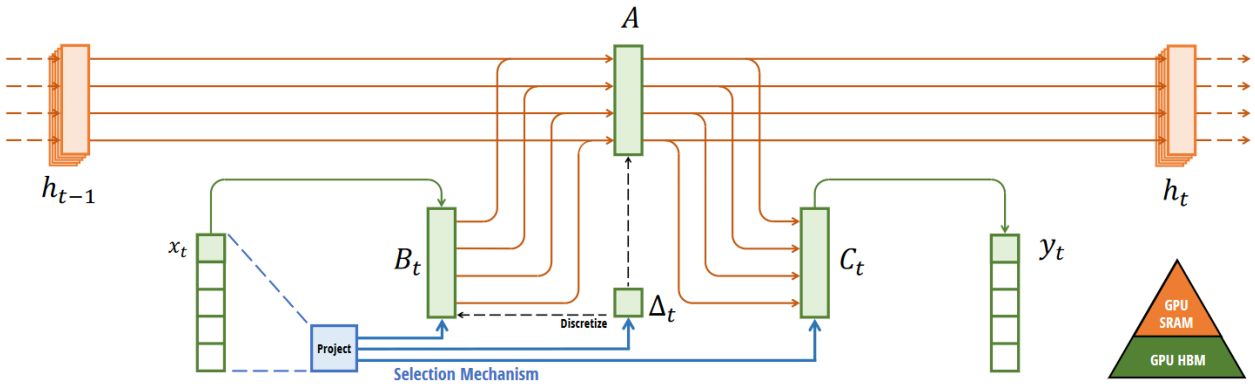
We can spawn multiple threads to perform the binary operation in parallel, synchronizing at each step.
The time complexity instead of being $O(N)$ is reduced to $O(N/T)$ where T is the number of parallel threads.

Umar Jamil - https://github.com/hkproj/mamba-notes

Mamba

- Keep and update the SSM state in fast cache memory. (SRAM)
- The state sequence is not recorded. Instead, we simply re-compute the states during backprop.





3.3.2 Overview of Selective Scan: Hardware-Aware State Expansion

Concretely, instead of preparing the scan input $(\overline{A}, \overline{B})$ of size $(\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N})$ in GPU HBM (high-bandwidth memory), we load the SSM parameters (Δ, A, B, C) directly from slow HBM to fast SRAM, perform the discretization and recurrence in SRAM, and then write the final outputs of size $(\mathtt{B}, \mathtt{L}, \mathtt{D})$ back to HBM.

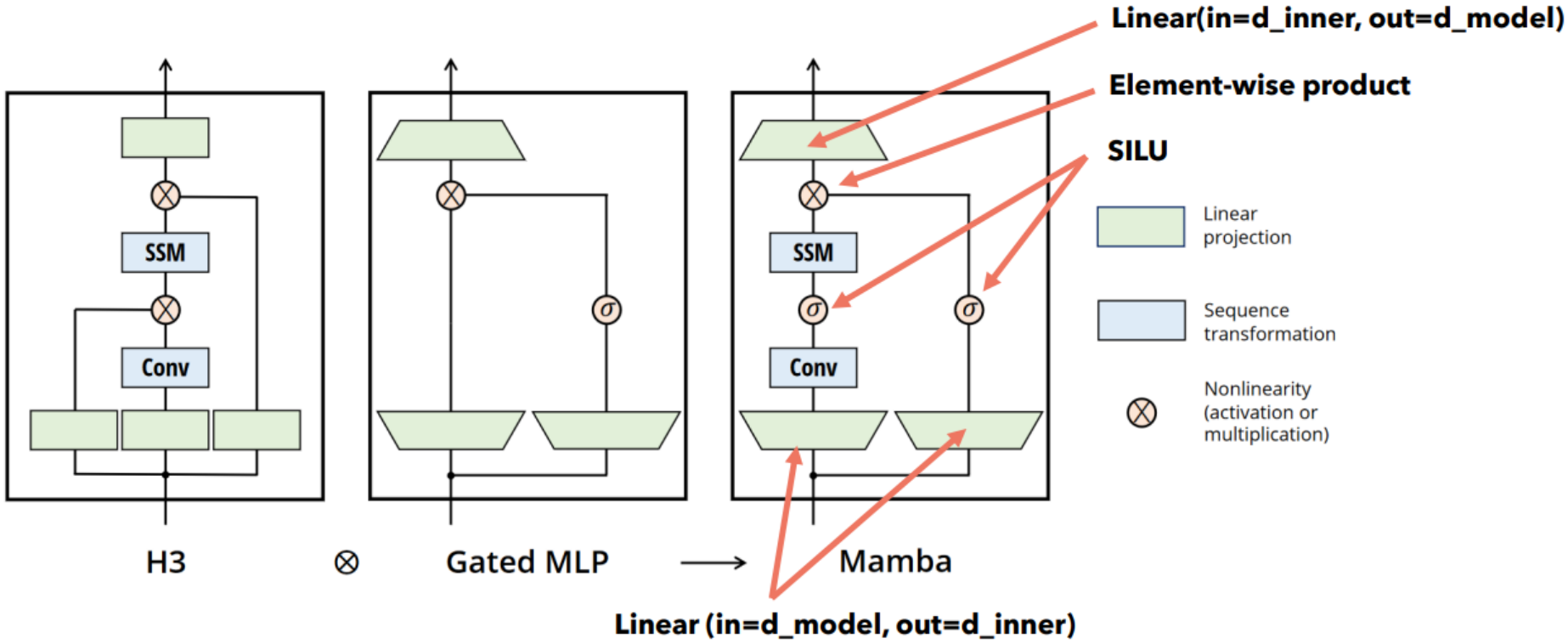
To avoid the sequential recurrence, we observe that despite not being linear it can still be parallelized with a work-efficient parallel scan algorithm (Blelloch 1990; Martin and Cundy 2018; Smith, Warrington, and Linderman 2023).

Finally, we must also avoid saving the intermediate states, which are necessary for backpropagation. We carefully apply the classic technique of recomputation to reduce the memory requirements: the intermediate states are not stored but recomputed in the backward pass when the inputs are loaded from HBM to SRAM. As a result, the fused selective scan layer has the same memory requirements as an optimized transformer implementation with FlashAttention.

Mamba

- Mamba block = H3 block + nonlinear gates

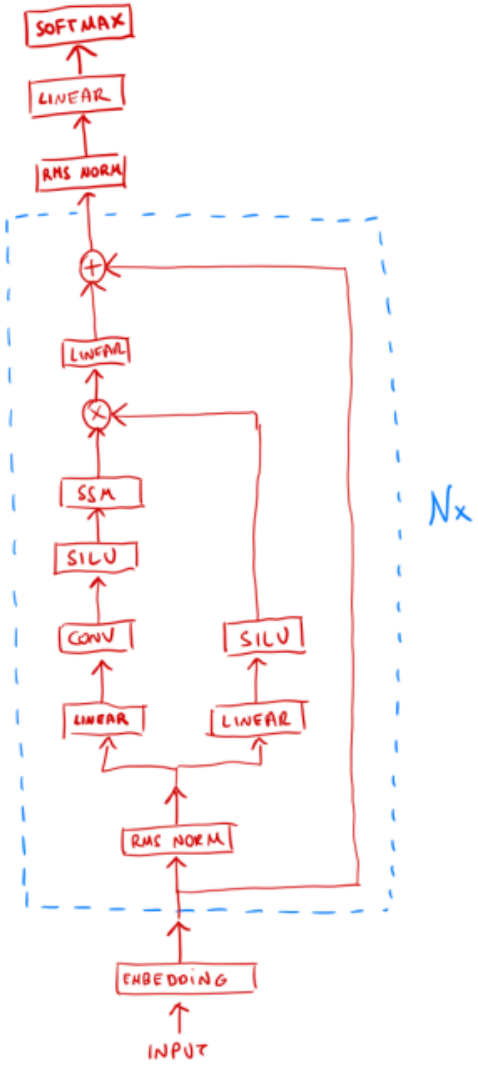
Mamba is built by stacking multiple layers of the Mamba block, shown below. This is very similar to the stacked layers of the Transformer model. The Mamba architecture derives from the *Hungry Hungry Hippo* (H3) architecture.



Now let's see the entire architecture of Mamba!

Umar Jamil - https://github.com/hkproj/mamba-notes

Mamba



3.4 A Simplified SSM Architecture

This architecture involves expanding the model dimension D by a controllable expansion factor E. For each block, most of the parameters ($3ED^2$) are in the linear projections ($2ED^2$ for input projections, ED^2 for output projection) while the inner SSM contributes less. The number of SSM parameters (projections for $\Delta, \boldsymbol{B}, \boldsymbol{C}$, and the matrix $\boldsymbol{A}$) are much smaller in comparison. We repeat this block, interleaved with standard normalization and residual connections, to form the Mamba architecture. We always fix to $E = 2$ in our experiments and use two stacks of the block to match the $12D^2$ parameters of a Transformer's interleaved MHA (multi-head attention) and MLP blocks. We use the SiLU / Swish activation function (Hendrycks and Gimpel 2016; Ramachandran, Zoph, and Quoc V Le 2017), motivated so that the Gated MLP becomes the popular "SwiGLU" variant (Chowdhery et al. 2023; Shazeer 2020; Touvron et al. 2023). Finally, we additionally use an optional normalization layer (we choose LayerNorm (J. L. Ba, Kiros, and Hinton 2016)), motivated by RetNet's usage of a normalization layer in a similar location (Y. Sun et al. 2023).

Umar Jamil - https://github.com/hkproj/mamba-notes

Mamba

- Language modeling

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

Mamba

- DNA sequence modeling

Results. Figure 5 (Right) shows that Mamba is able to make use of longer context even up to extremely long sequences of length 1M, and its pretraining perplexity improves as the context increases. On the other hand, the HyenaDNA model gets worse with sequence length. This is intuitive from the discussion in Section 3.5 on properties of the selection mechanism. In particular, LTI models cannot selectively ignore information; from a convolutional perspective, a very long convolution kernel is aggregating all information across a long sequence

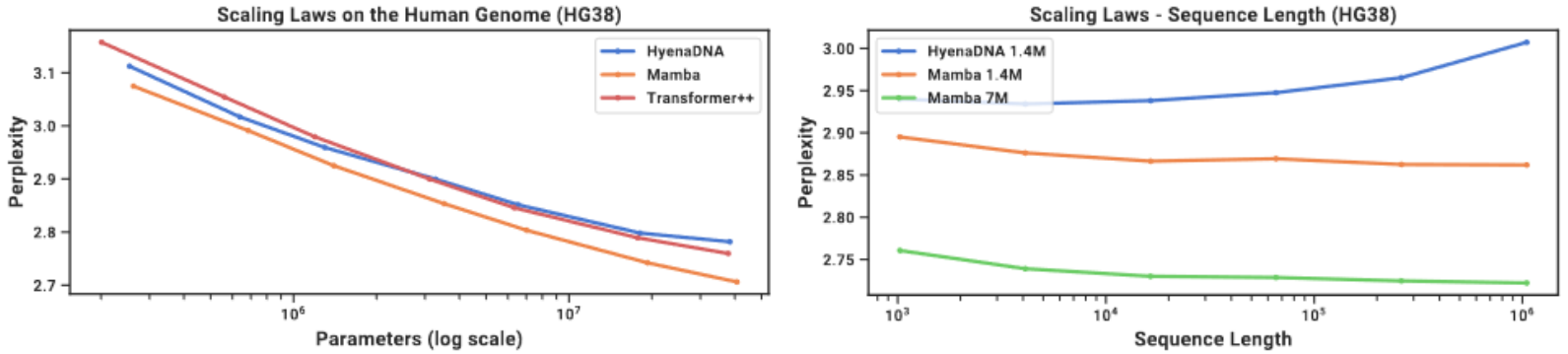


Figure 5: (**DNA Scaling Laws.**) Pretraining on the HG38 (human genome) dataset. (*Left*) Fixing short context length $2^{10} = 1024$ and increasing size from $\approx 200K$ to $\approx 40M$ parameters, Mamba scales better than baselines. (*Right*) Fixing model size and increasing sequence lengths while keeping tokens/batch and total training tokens fixed. Unlike baselines, the selection mechanism of Mamba facilitates better performance with increasing context length.

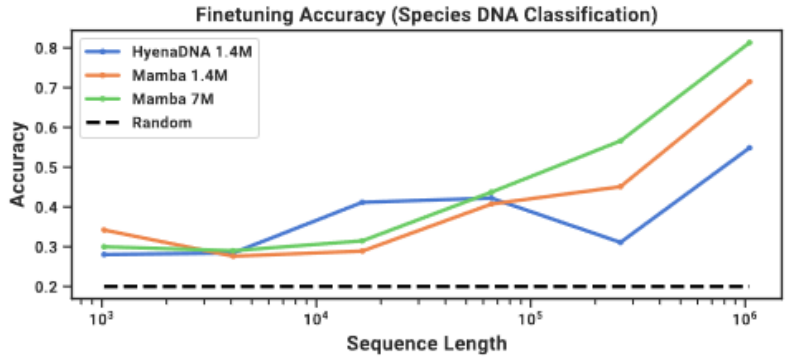


Figure 6: (**Great Apes DNA Classification.**) Accuracy after fine-tuning on sequences of length $2^{10} = 1024$ up to $2^{20} = 1048576$ using pretrained models of the same context length. Numerical results in Table 13.

Mamba

- Audio modeling and generation

4.4.1 Long-Context Autoregressive Pretraining

We evaluate pretraining quality (autoregressive next-sample prediction) on YouTubeMix (DeepSound 2017), a standard piano music dataset used by prior work consisting of 4 hours of solo piano music, sampled at a rate of 16000 Hz Pretraining details largely follow the standard language modeling setup (Section 4.2). Figure 7 evaluates the effect of increasing training sequence lengths from $2^{13} = 8192$ to $2^{20} \approx 10^6$, while keeping computation fixed. (There are some slight edge cases to the way the data is curated, which may lead to kinks in the scaling curves. For example, only minute-long clips were available so the maximum sequence length is actually bounded by $60s \cdot 16000Hz = 960000$.)

Table 4: (**SC09**) Automated metrics for unconditional generation on a challenging dataset of fixed-length speech clips. (*Top to Bottom*) Autoregressive baselines, non-autoregressive baselines, Mamba, and dataset metrics.

Model	Params	NLL ↓	FID ↓	IS ↑	mIS ↑	AM ↓
SampleRNN	35.0M	2.042	8.96	1.71	3.02	1.76
WaveNet	4.2M	1.925	5.08	2.27	5.80	1.47
SaShiMi	5.8M	1.873	1.99	5.13	42.57	0.74
WaveGAN	19.1M	-	2.03	4.90	36.10	0.80
DiffWave	24.1M	-	1.92	5.26	51.21	0.68
+ SaShiMi	23.0M	-	1.42	5.94	69.17	0.59
Mamba	6.1M	**1.852**	0.94	6.26	88.54	0.52
Mamba	24.3M	1.860	**0.67**	**7.33**	**144.9**	**0.36**
Train	-	-	0.00	8.56	292.5	0.16
Test	-	-	0.02	8.33	257.6	0.19

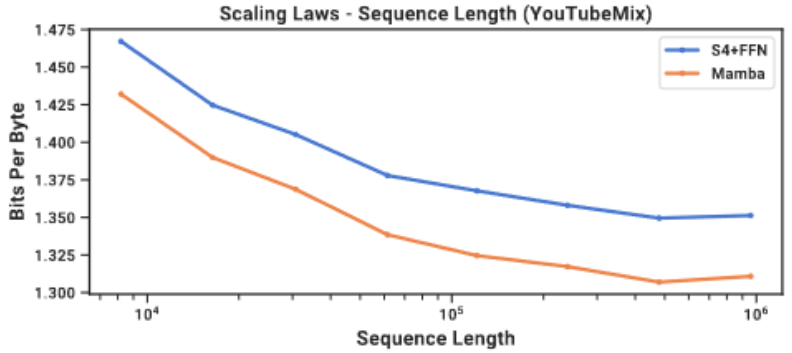


Figure 7: (**Audio Pretraining.**) Mamba improves performance over prior state-of-the-art (Sashimi) in autoregressive audio modeling, while improving up to minute-long context or million-length sequences (controlling for computation).

Mamba

- Speed and memory compared to attention

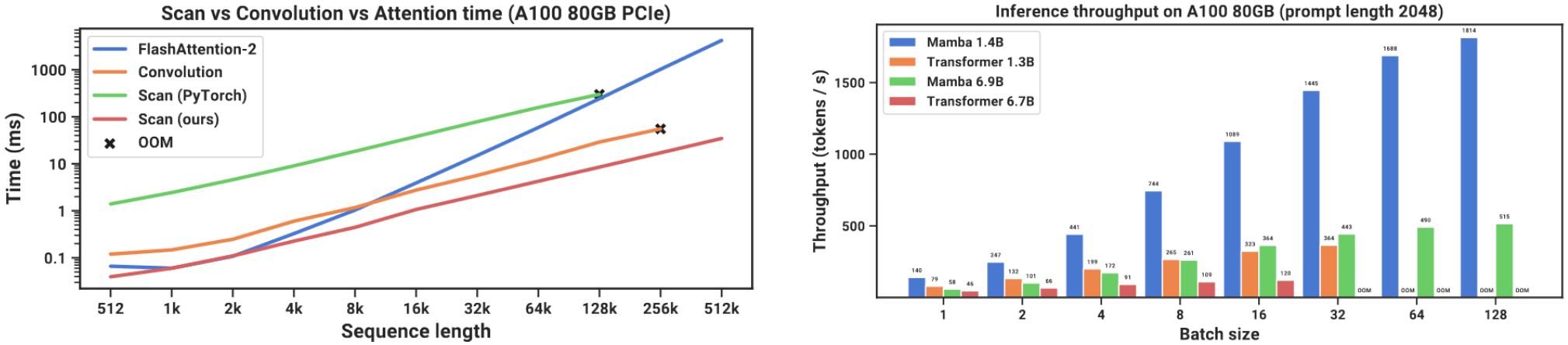


Figure 8: (**Efficiency Benchmarks.**) (*Left*) Training: our efficient scan is 40× faster than a standard implementation. (*Right*) Inference: as a recurrent model, Mamba can achieve 5× higher throughput than Transformers.

Vision Mamba (Vim)

Zhu, Lianghui, et al. "Vision Mamba: Efficient Visual Representation Learning with Bidirectional State Space Model." *arXiv preprint arXiv:2401.09417* (2024).

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

Lianghui Zhu[1*], Bencheng Liao[1*], Qian Zhang[2], Xinlong Wang[3], Wenyu Liu[1], Xinggang Wang[1] ✉

[1] Huazhong University of Science and Technology

[2] Horizon Robotics [3] Beijing Academy of Artificial Intelligence

Code & Models: hustvl/Vim

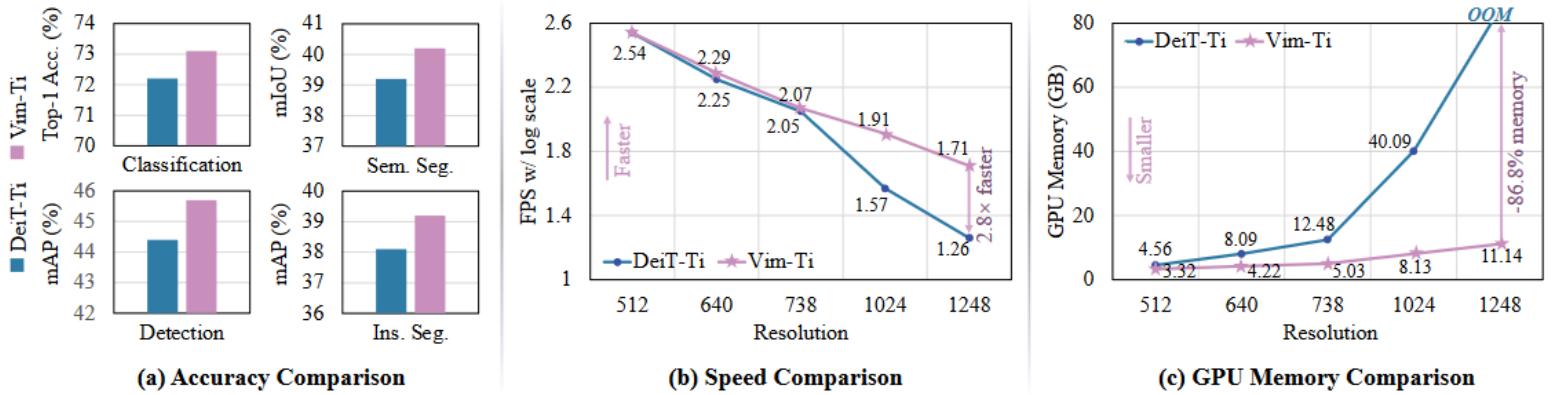


Figure 1. Performance and efficiency comparisons between DeiT [60] and our Vim model. For the accuracy comparison, we first pretrain DeiT and Vim on IN1K classification dataset [10], then we finetune the generic backbones on different downstream dense prediction tasks, *i.e.*, semantic segmentation, object detection, instance segmentation. Results show that the proposed Vim outperforms DeiT on both pretraining and finetuning tasks. For the speed comparison, the proposed Vim is significantly faster than DeiT when dealing with large images because of its subquadratic-time computation. For the GPU memory comparison, Vim requires less GPU memory than DeiT for inferring large images by its linear memory complexity. With not only superior accuracy for vision tasks, the proposed Vim is also more efficient than DeiT in dealing with large images. For example, Vim is 2.8× faster than DeiT and saves 86.8% GPU memory when performing batch inference to extract features on images with a resolution of 1248×1248, *i.e.*, 6084 tokens per image.

VMamba

Liu, Yue, et al. "VMamba: Visual State Space Model." *arXiv preprint arXiv:2401.10166* (2024).

VMamba: Visual State Space Model

Yue Liu
UCAS
liuyue171@mails.ucas.ac.cn

Yunjie Tian
UCAS
tianyunjie19@mails.ucas.ac.cn

Yuzhong Zhao
UCAS
zhaoyuzhong20@mails.ucas.ac.cn

Hongtian Yu
UCAS
yuhongtian17@mails.ucas.ac.cn

Lingxi Xie
Huawei Inc.
198808xc@gmail.com

Yaowei Wang
Pengcheng Lab.
wangyw@pcl.ac.cn

Qixiang Ye
UCAS
qxye@ucas.ac.cn

Yunfan Liu
UCAS
yunfan.liu@cripac.ia.ac.cn

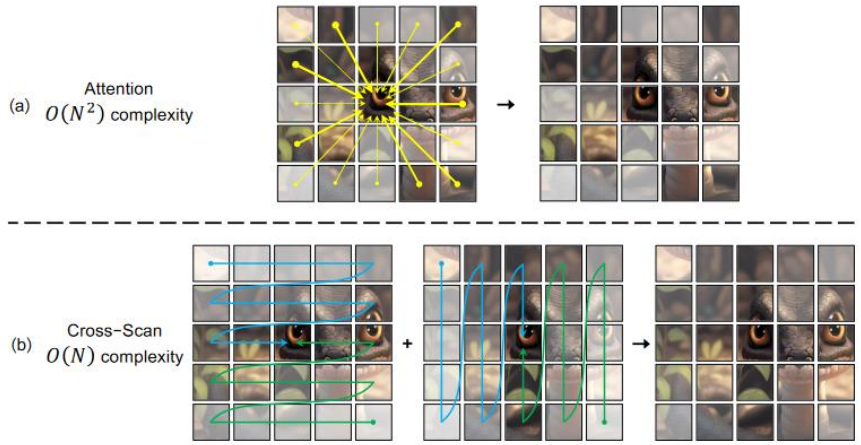


Figure 2: **Comparison of information flow: Attention vs. Cross-Scan Module (CSM).** (a) The attention mechanism uniformly integrates all pixels for the center pixel, resulting in $\mathcal{O}(N^2)$ complexity. (b) CSM integrates pixels from top-left, bottom-right, top-right, and bottom-left with $\mathcal{O}(N)$ complexity.

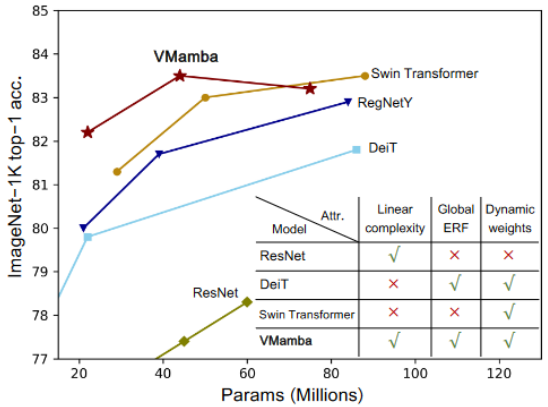


Figure 1: **Performance comparison on ImageNet-1K.** VMamba series achieves superior top-1 accuracy compared to popular counterparts. We note that the proposed VMamba has the capability of showing global effective reception field (ERF), dynamic weights with linear complexity.

method	image size	#param.	FLOPs	ImageNet top-1 acc.
RegNetY-4G [36]	224^2	21M	4.0G	80.0
RegNetY-8G [36]	224^2	39M	8.0G	81.7
RegNetY-16G [36]	224^2	84M	16.0G	82.9
EffNet-B3 [42]	300^2	12M	1.8G	81.6
EffNet-B4 [42]	380^2	19M	4.2G	82.9
EffNet-B5 [42]	456^2	30M	9.9G	83.6
EffNet-B6 [42]	528^2	43M	19.0G	84.0
ViT-B/16 [10]	384^2	86M	55.4G	77.9
ViT-L/16 [10]	384^2	307M	190.7G	76.5
DeiT-S [43]	224^2	22M	4.6G	79.8
DeiT-B [43]	224^2	86M	17.5G	81.8
DeiT-B [43]	384^2	86M	55.4G	83.1
Swin-T [28]	224^2	29M	4.5G	81.3
Swin-S [28]	224^2	50M	8.7G	83.0
Swin-B [28]	224^2	88M	15.4G	83.5
S4ND-ViT-B [33]	224^2	89M	-	80.4
VMamba-T	224^2	22M	4.5G	82.2
VMamba-S	224^2	44M	9.1G	83.5
VMamba-B	224^2	75M	15.2G	83.2†

Table 2: **Accuracy comparison across various models on ImageNet-1K.** The symbol † indicates that a bug is encountered during the training of VMamba-B, and we will update the correct number in the near future.

DiffuSSM

Yan, Jing Nathan, Jiatao Gu, and Alexander M. Rush. "Diffusion models without attention." *arXiv preprint arXiv:2311.18257* (2023).

Diffusion Models Without Attention

Jing Nathan Yan[1*], Jiatao Gu[2*], Alexander M. Rush[1]
[1]Cornell University, [2]Apple
{jy858, arush}@cornell.edu, jgu32@apple.com

Abstract

In recent advancements in high-fidelity image generation, Denoising Diffusion Probabilistic Models (DDPMs) have emerged as a key player. However, their application at high resolutions presents significant computational challenges. Current methods, such as patchifying, expedite processes in UNet and Transformer architectures but at the expense of representational capacity. Addressing this, we introduce the Diffusion State Space Model (DIFFUSSM), an architecture that supplants attention mechanisms with a more scalable state space model backbone. This approach effectively handles higher resolutions without resorting to global compression, thus preserving detailed image representation throughout the diffusion process. Our focus on FLOP-efficient architectures in diffusion training marks a significant step forward. Comprehensive evaluations on both ImageNet and LSUN datasets at two resolutions demonstrate that DiffuSSMs are on par or even outperform existing diffusion models with attention modules in FID and Inception Score metrics while significantly reducing total FLOP usage.

Figure 1. Selected samples generated by class-conditional DIFFUSSM trained on ImageNet 256 × 256 and 512 × 512 resolutions.

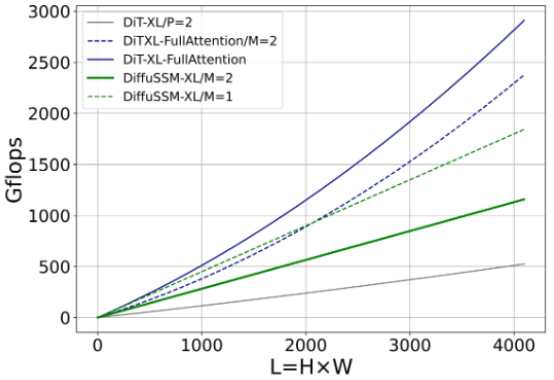


Figure 3. Comparison of Gflops of DiT and DIFFUSSM under various model architecture. DiT with patching (P=2) scales well to longer sequences, however when patching is removed it scales poorly even with hourglass (M=2). DIFFUSSM scales well, and hourglass (M=2) can be used to reduce absolute Gflops.

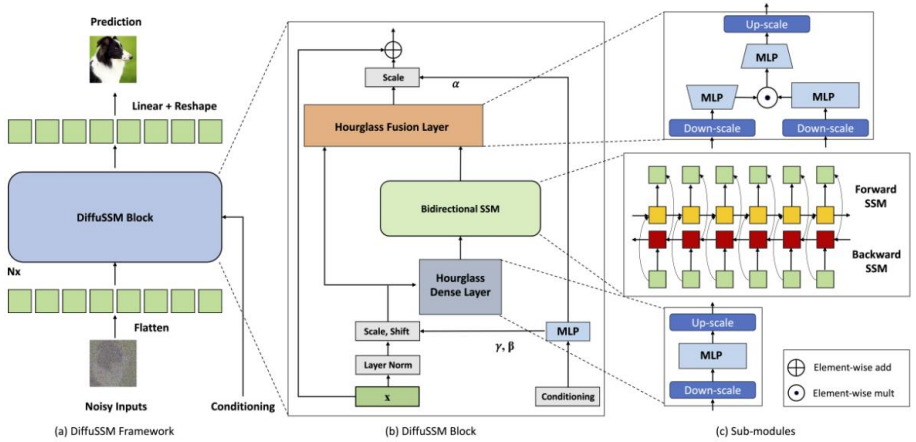


Figure 2. Architecture of DIFFUSSM. DIFFUSSM takes a noised image representation which can be a noised latent from a variational encoder, flattens it to a sequence, and applies repeated layers alternating long-range SSM cores with hour-glass feed-forward networks. Unlike with U-Nets or Transformers, there is no application of patchification or scaling for the long-range block.

MambaByte

Wang, Junxiong, et al. "MambaByte: Token-free Selective State Space Model." *arXiv preprint arXiv:2401.13660* (2024).

- Token-free language modeling
- Perhaps a step towards “any-modal” foundation models?

MambaByte: Token-free Selective State Space Model

Junxiong Wang **Tushaar Gangavarapu** **Jing Nathan Yan** **Alexander M Rush**
Cornell University
{jw2544,tg352,jy858,arush}@cornell.edu

Abstract

Token-free language models learn directly from raw bytes and remove the bias of subword tokenization. Operating on bytes, however, results in significantly longer sequences, and standard autoregressive Transformers scale poorly in such settings. We experiment with MambaByte, a token-free adaptation of the Mamba state space model, trained autoregressively on byte sequences. Our experiments indicate the computational efficiency of MambaByte compared to other byte-level models. We also find MambaByte to be competitive with and even outperform state-of-the-art subword Transformers. Furthermore, owing to linear scaling in length, MambaByte benefits from fast inference compared to Transformers. Our findings establish the viability of MambaByte in enabling token-free language modeling.

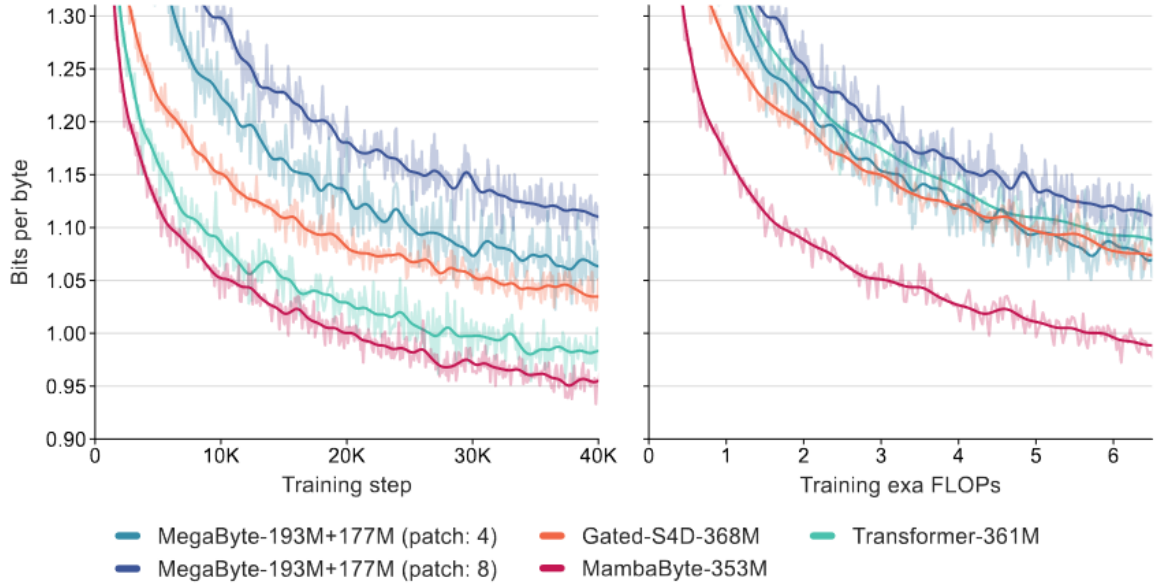


Figure 1: **Benchmarking byte-level models with a fixed parameter budget.** Language modeling results on PG19 (8, 192 consecutive bytes), comparing the standard Transformer [Vaswani et al., 2017, Su et al., 2021], MegaByte Transformer [Yu et al., 2023], gated diagonalized S4 [Mehta et al., 2023], and MambaByte. (Left) Model loss over training step. (Right) FLOP-normalized training cost. MambaByte reaches Transformer loss in less than one-third of the compute budget.