Machine Learning Department
School of Computer Science
Carnegie Mellon University

State Space Models

Matt Gormley & Pat Virtue
Lecture 22
Apr. 9, 2025

Reminders

- Project Midway Report
  - Due: Thu, Apr-17 at 11:59pm
- HW623
  - Only for students registered in 10-623
  - Due: Mon, Apr-21 at 11:59pm

Motivation

- Transformers are slow at test time: they require a KV cache that grows quadratically in size with the sequence length
- State space models (SSMs) are fast at test time: they only hold a fixed size hidden state in memory (like RNNs)
- But we'll see that SSMs can also be trained efficiently with the right tricks
- As well, they elegantly transition between different granularities of representation for the input (e.g. sound at 16Hz vs. 8Hz

Motivation

- https://www.isattentionallyouneed.com/

Is Attention All You Need?

Current Status: Yes

Time Remaining: 631d 22h 55m 11s

Proposition:

On January 1, 2027, a Transformer-like model will continue to hold the state-of-the-art position in most benchmarked tasks in natural language processing.

Motivation

- https://www.isattentionallyouneed.com/

Is Attention All You Need?

For the Motion

Jonathan Frankle
@jefrankle
Harvard Professor
Chief Scientist Mosaic ML

Against the Motion

Sasha Rush
@srush_nlp
Cornell Professor
Research Scientist Hugging Face 🤗

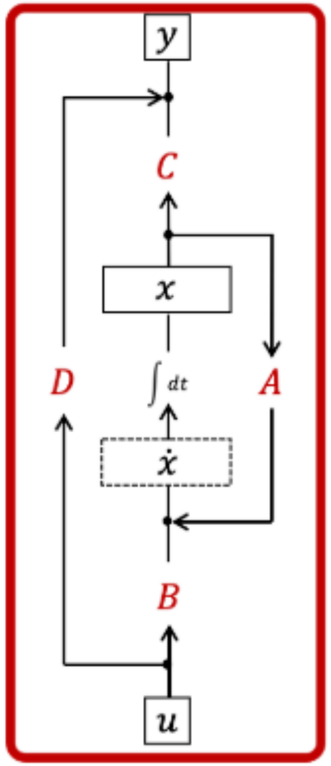
Wager

The wager is for donation of equity in Mosaic ML or Hugging Face to a charity of the winner's choice. Details to come.

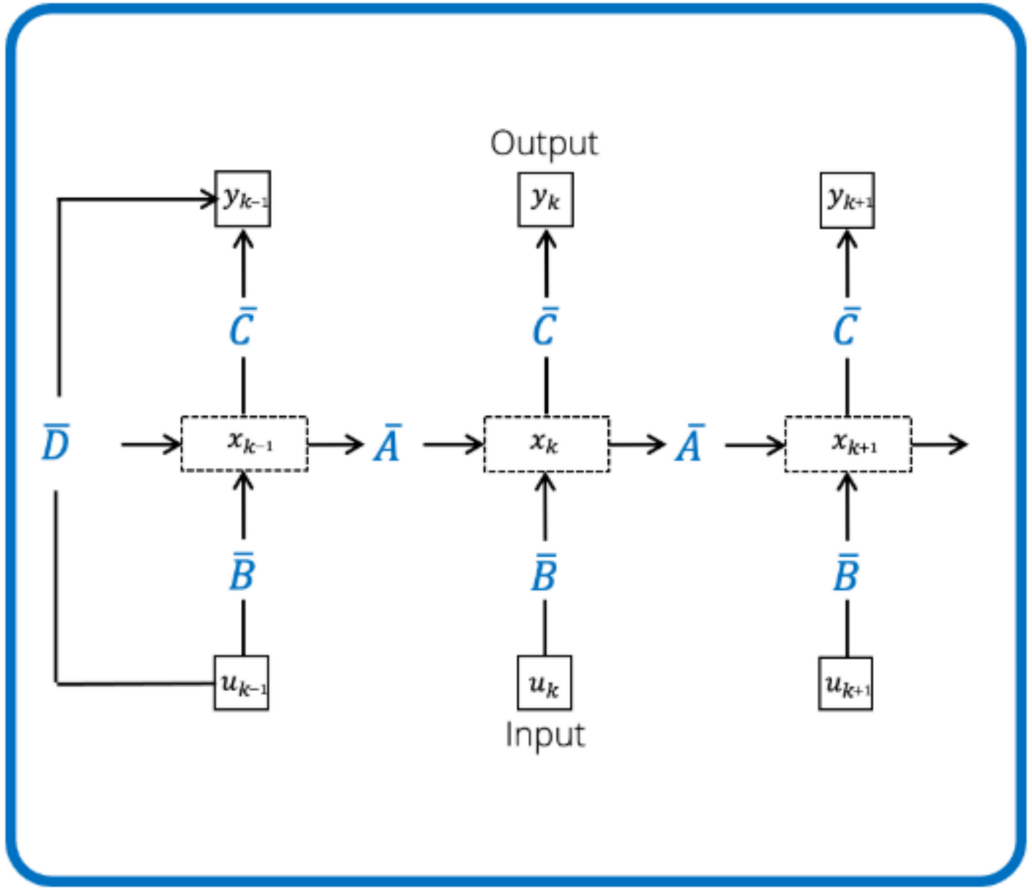
STATE SPACE MODEL (SSM)

Three Representations of a State Space Model

Continuous



Recurrent



Convolutional

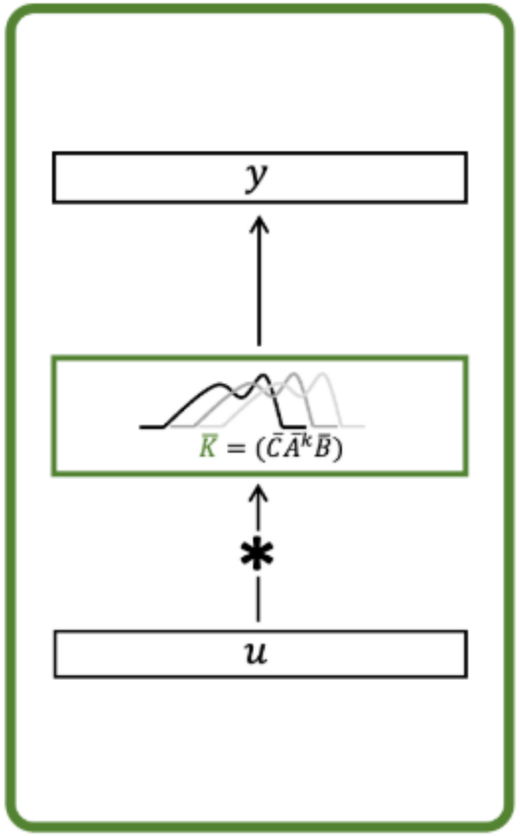
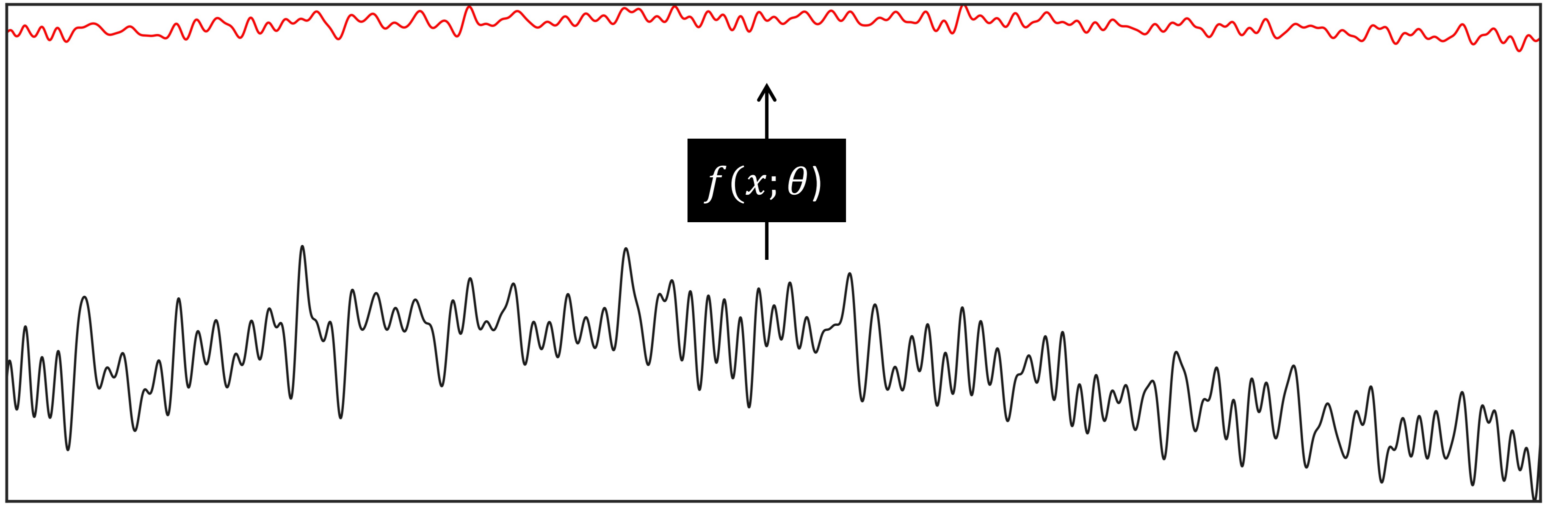
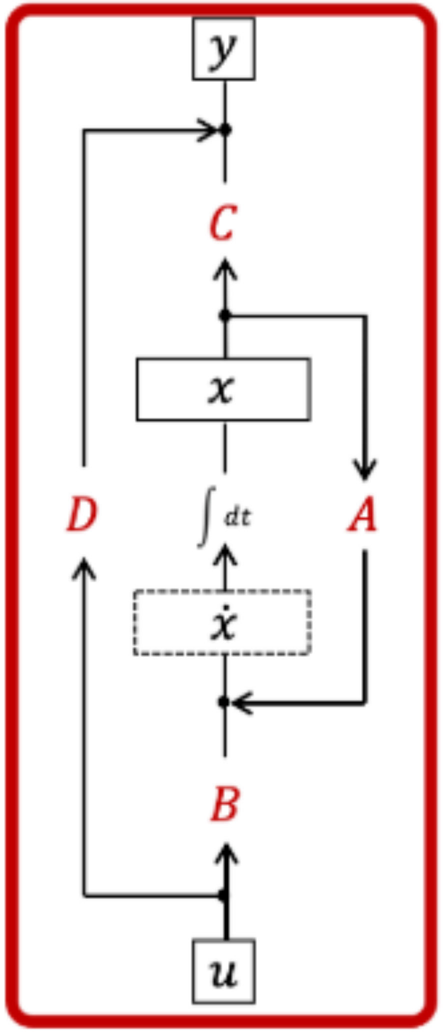


Figure from https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3

SSM: 1D Continuous Representation

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





SSMs map 1D **function** to 1D **function**

Figure from Albert Gu's F24 guest lecture in 10-423/623

Figure from https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3

SSM: 1D Continuous Representation

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

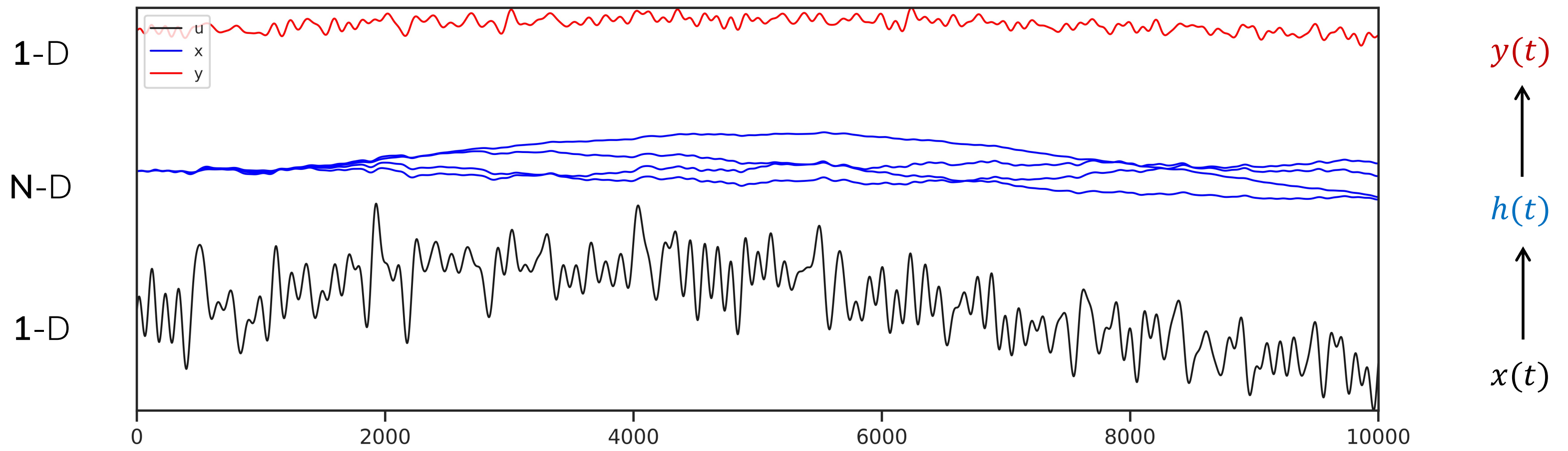


Figure from Albert Gu's F24 guest lecture in 10-423/623

SSM: 1D Continuous Representation

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

$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

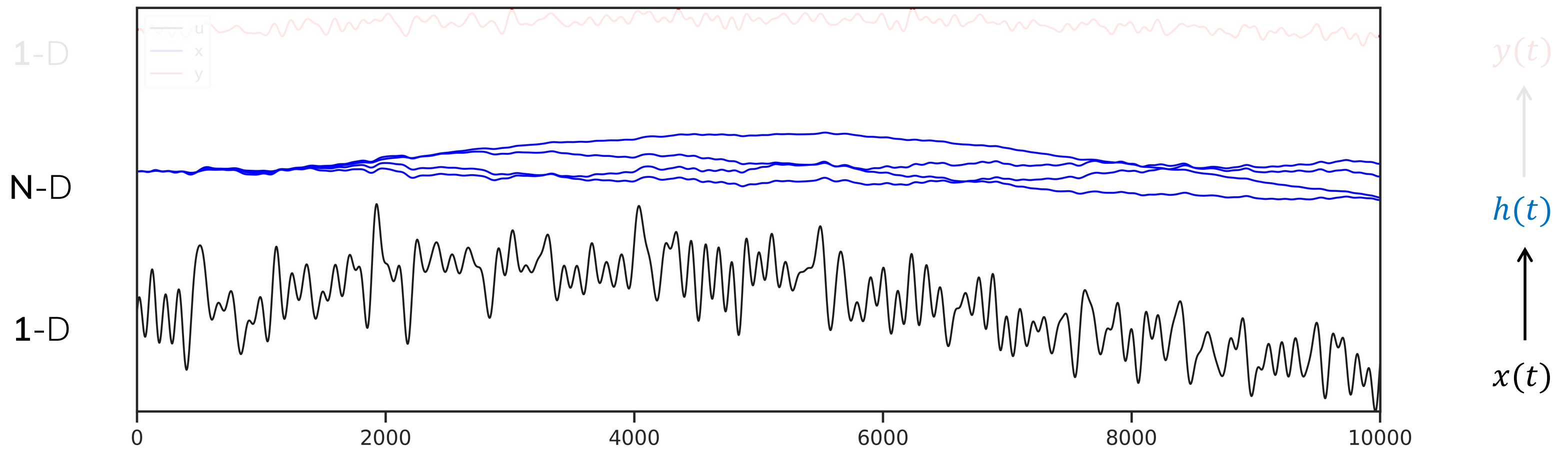


Figure from Albert Gu's F24 guest lecture in 10-423/623

SSM: 1D Continuous Representation

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

$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

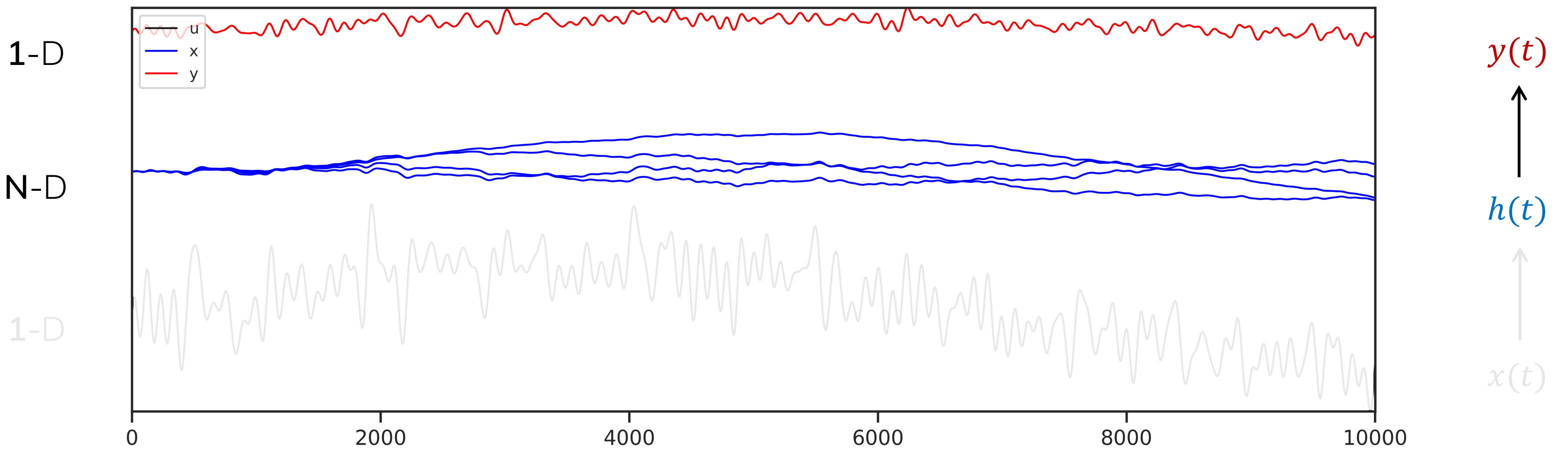


Figure from Albert Gu's F24 guest lecture in 10-423/623

SSM: 1D Continuous Representation

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

$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

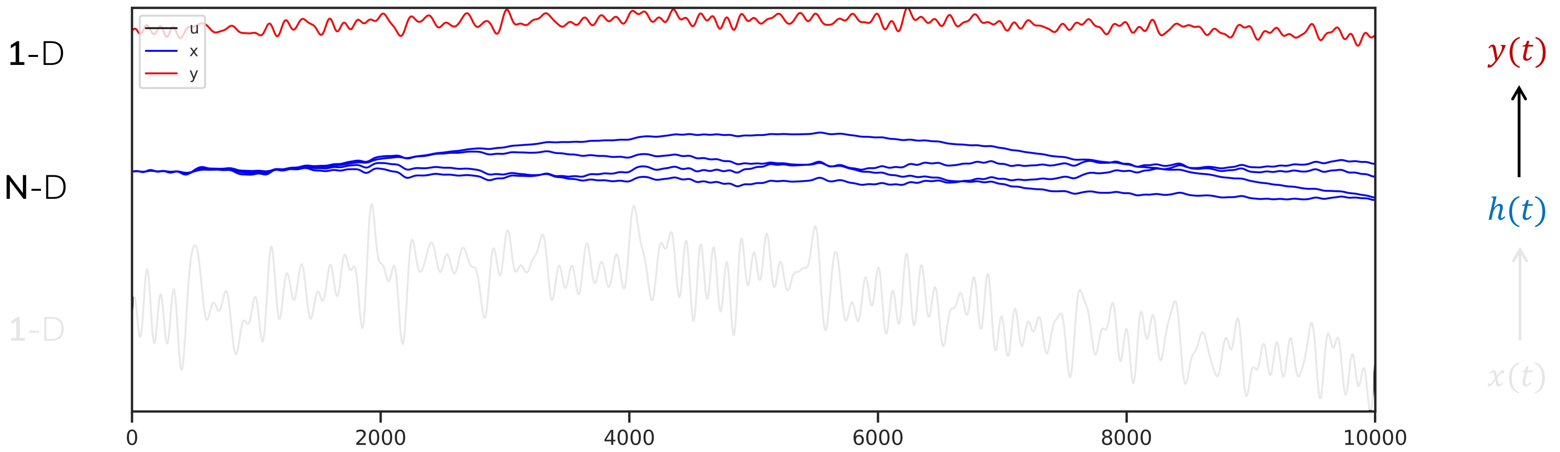


Figure from Albert Gu’s F24 guest lecture in 10-423/623

SSM: 1D Continuous Representation

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

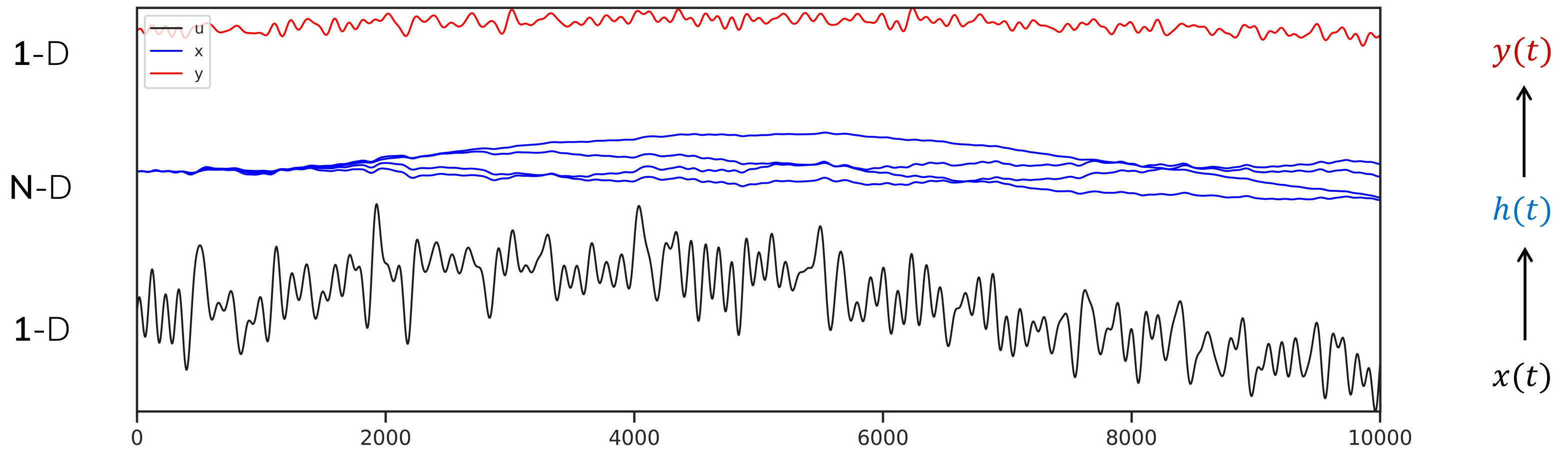


Figure from Albert Gu's F24 guest lecture in 10-423/623

SSM: 1D Discrete Recurrent Representation

Continuous Representation

- Uses parameters we will actually work with in the end
- Seamlessly represents any continuous 1D → 1D function
- Impractical for real data

Discrete Recurrent Representation

- A discrete approximation using different parameters which are **functions** of the original parameters A,B,C,D
- Allows us to work with real data

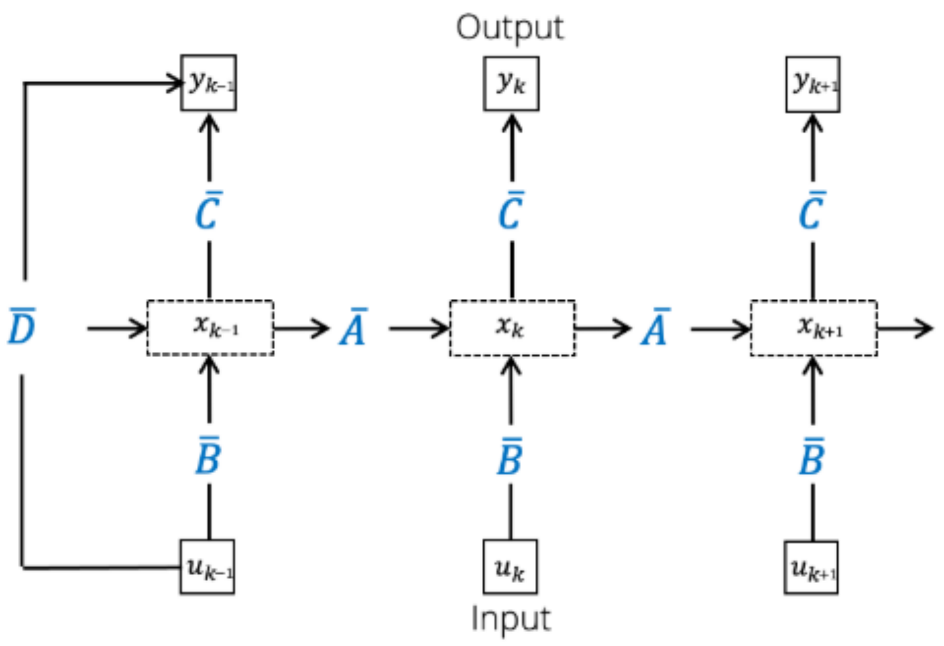
$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t)$$
$$y(t) = \mathbf{C}h(t) + \mathbf{D}x(t)$$

→

$$h_{k+1} = \mathbf{A}h_k + \mathbf{B}x_k$$
$$y_k = \mathbf{C}h_k + \mathbf{D}x_k$$

SSM: 1D Discrete Recurrent Representation

Question: How can we depict this recurrent computation?



Discrete Recurrent Representation

- A discrete approximation using different parameters which are **functions** of the original parameters A,B,C,D
- Allows us to work with real data

$$h_{k+1} = \mathbf{A}h_k + \mathbf{B}x_k$$
$$y_k = \mathbf{C}h_k + \mathbf{D}x_k$$

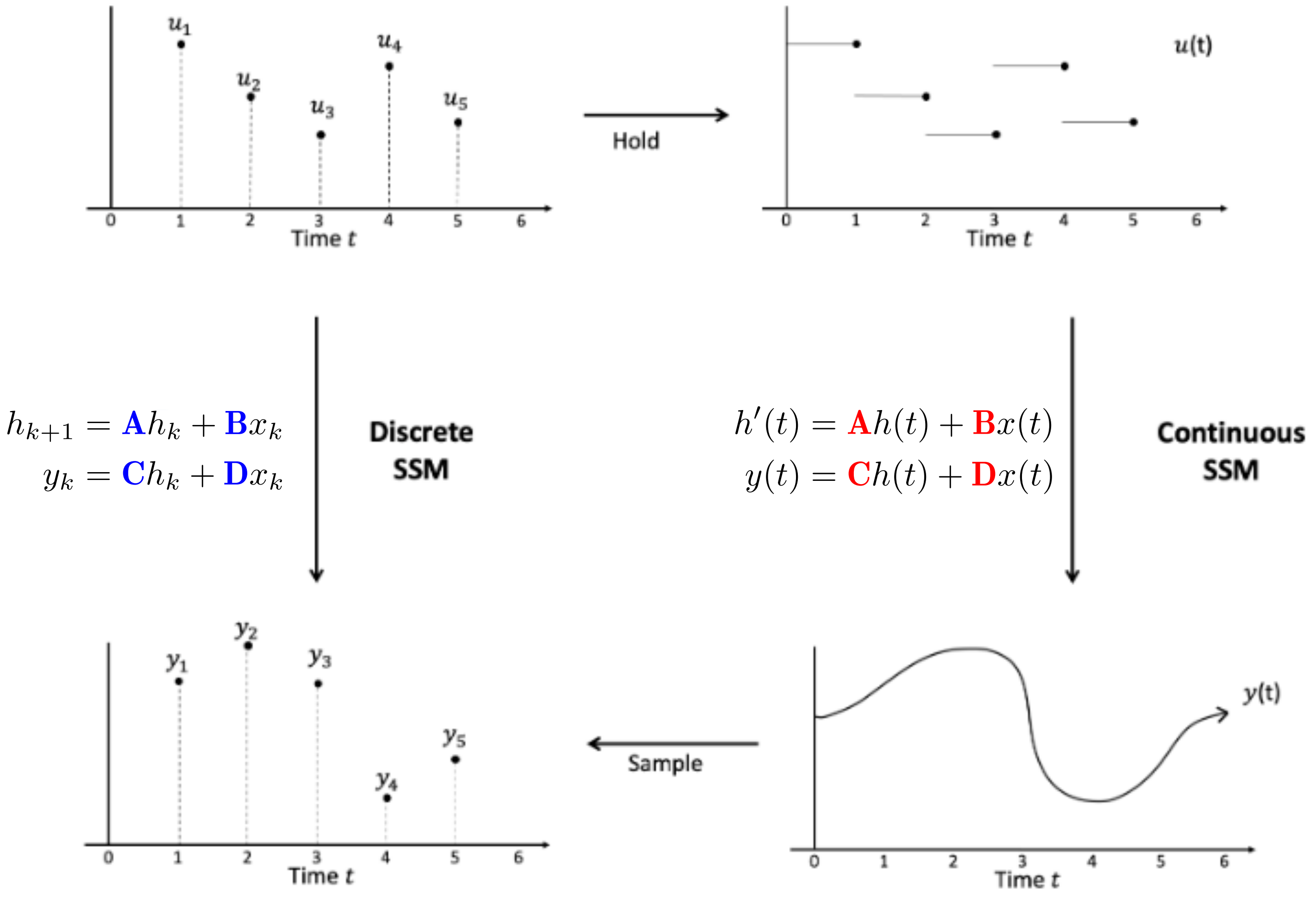
How to discretize a continuous SSM?

S4 uses a bilinear transformation to discretize the continuous SSM

$$\overline{A} = e^{\Delta A}$$
$$\overline{B} = A^{-1}(e^{\Delta A} - I)B$$
$$\overline{C} = C$$
$$\overline{D} = D$$

The bilinear transformation uses a first order Pade approximation:

$$e^x \approx \frac{1+x/2}{1-x/2}$$



The discrete-time SSM (left), a sequence-to-sequence map, is exactly equivalent to applying the continuous-time SSM (right), a function-to-function map, on the held signal.

Figure from https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3

How to discretize a continuous SSM?

S4 uses a bilinear transformation
to discretize the continuous SSM

$$\overline{\boldsymbol{A}} = e^{\Delta \boldsymbol{A}}$$
$$\overline{\boldsymbol{B}} = \boldsymbol{A}^{-1}(e^{\Delta \boldsymbol{A}} - \boldsymbol{I})\boldsymbol{B}$$
$$\overline{\boldsymbol{C}} = \boldsymbol{C}$$
$$\overline{\boldsymbol{D}} = \boldsymbol{D}$$

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

The bilinear transformation uses
a first order Pade approximation:

$$e^x \approx \frac{1+x/2}{1-x/2}$$

Figure from https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3

SSM: 1D Convolutional Representation

We unroll the recurrent computation as:
Assume a zero initial state: $h_{-1} = 0$.

$$h_0 = \bar{\mathbf{B}} x_0$$
$$h_1 = \bar{\mathbf{A}}\bar{\mathbf{B}} x_0 + \bar{\mathbf{B}} x_1$$
$$h_2 = \bar{\mathbf{A}}^2\bar{\mathbf{B}} x_0 + \bar{\mathbf{A}}\bar{\mathbf{B}} x_1 + \bar{\mathbf{B}} x_2$$
$$\vdots$$
$$y_0 = \bar{\mathbf{C}}\bar{\mathbf{B}} x_0$$
$$y_1 = \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}} x_0 + \bar{\mathbf{C}}\bar{\mathbf{B}} x_1$$
$$y_2 = \bar{\mathbf{C}}\bar{\mathbf{A}}^2\bar{\mathbf{B}} x_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}} x_1 + \bar{\mathbf{C}}\bar{\mathbf{B}} x_2$$
$$\vdots$$
$$y_k = \bar{\mathbf{C}}\bar{\mathbf{A}}^k\bar{\mathbf{B}} x_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}^{k-1}\bar{\mathbf{B}} x_1 + \cdots + \bar{\mathbf{C}}\bar{\mathbf{A}}^1\bar{\mathbf{B}} x_{k-1} + \bar{\mathbf{C}}\bar{\mathbf{A}}^0\bar{\mathbf{B}} x_k$$

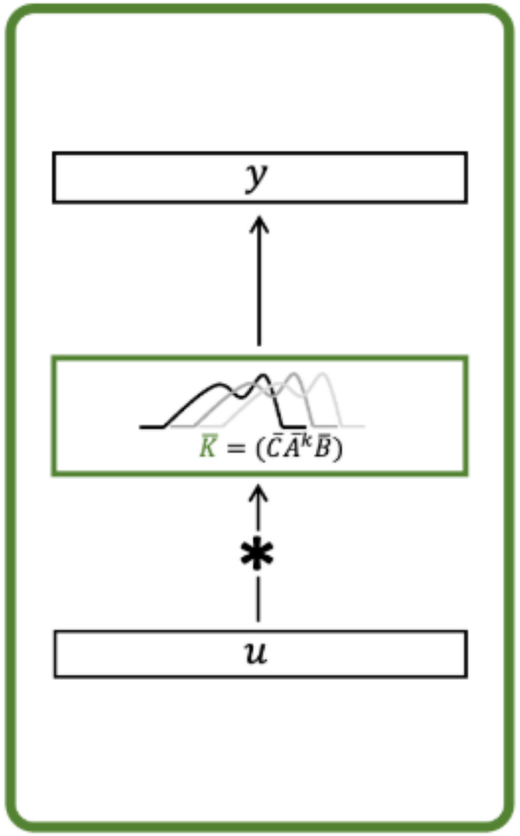
We can represent this as a *global* convolution computation:

$$\mathbf{y} = \bar{\mathbf{K}} * \mathbf{x}$$

where the SSM convolution kernel is:

$$\bar{\mathbf{K}} \in \mathbb{R}^L = \left(\bar{\mathbf{C}}\bar{\mathbf{A}}^0\bar{\mathbf{B}},\ \bar{\mathbf{C}}\bar{\mathbf{A}}^1\bar{\mathbf{B}},\ \ldots,\ \bar{\mathbf{C}}\bar{\mathbf{A}}^{L-2}\bar{\mathbf{B}}, \bar{\mathbf{C}}\bar{\mathbf{A}}^{L-1}\bar{\mathbf{B}}\right)$$

SSM: 1D Convolutional Representation



We can represent this as a *global* convolution computation:

$$\mathbf{y} = \bar{\mathbf{K}} * \mathbf{x}$$

where the SSM convolution kernel is:

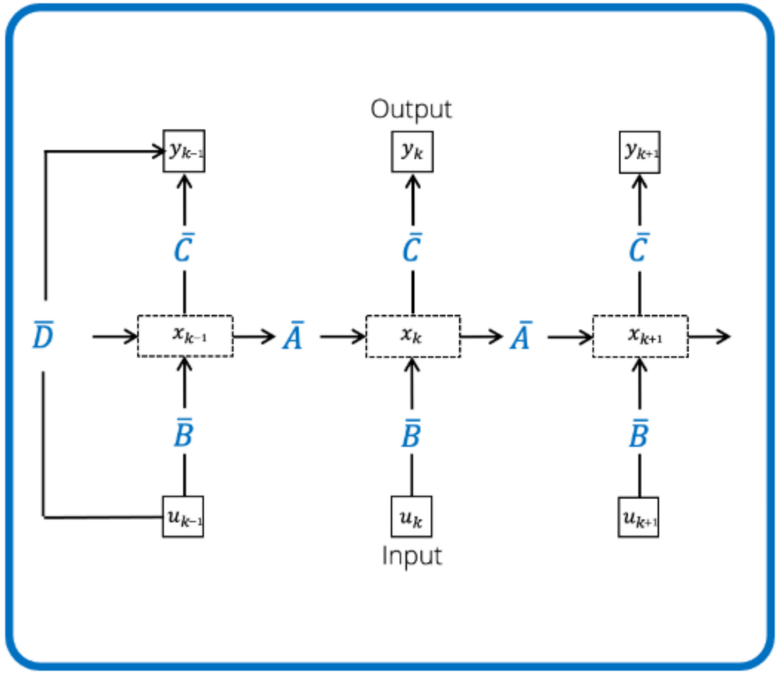
$$\bar{\mathbf{K}} \in \mathbb{R}^L = \left(\bar{\mathbf{C}}\bar{\mathbf{A}}^0\bar{\mathbf{B}},\ \bar{\mathbf{C}}\bar{\mathbf{A}}^1\bar{\mathbf{B}},\ \ldots,\ \bar{\mathbf{C}}\bar{\mathbf{A}}^{L-2}\bar{\mathbf{B}}, \bar{\mathbf{C}}\bar{\mathbf{A}}^{L-1}\bar{\mathbf{B}}\right)$$

$$y_k = \bar{\mathbf{C}}\bar{\mathbf{A}}^k\bar{\mathbf{B}}x_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}^{k-1}\bar{\mathbf{B}}x_1 + \cdots + \bar{\mathbf{C}}\bar{\mathbf{A}}^1\bar{\mathbf{B}}x_{k-1} + \bar{\mathbf{C}}\bar{\mathbf{A}}^0\bar{\mathbf{B}}x_k$$

THE STRUCTURED STATE SPACE SEQUENCE MODEL (S4)

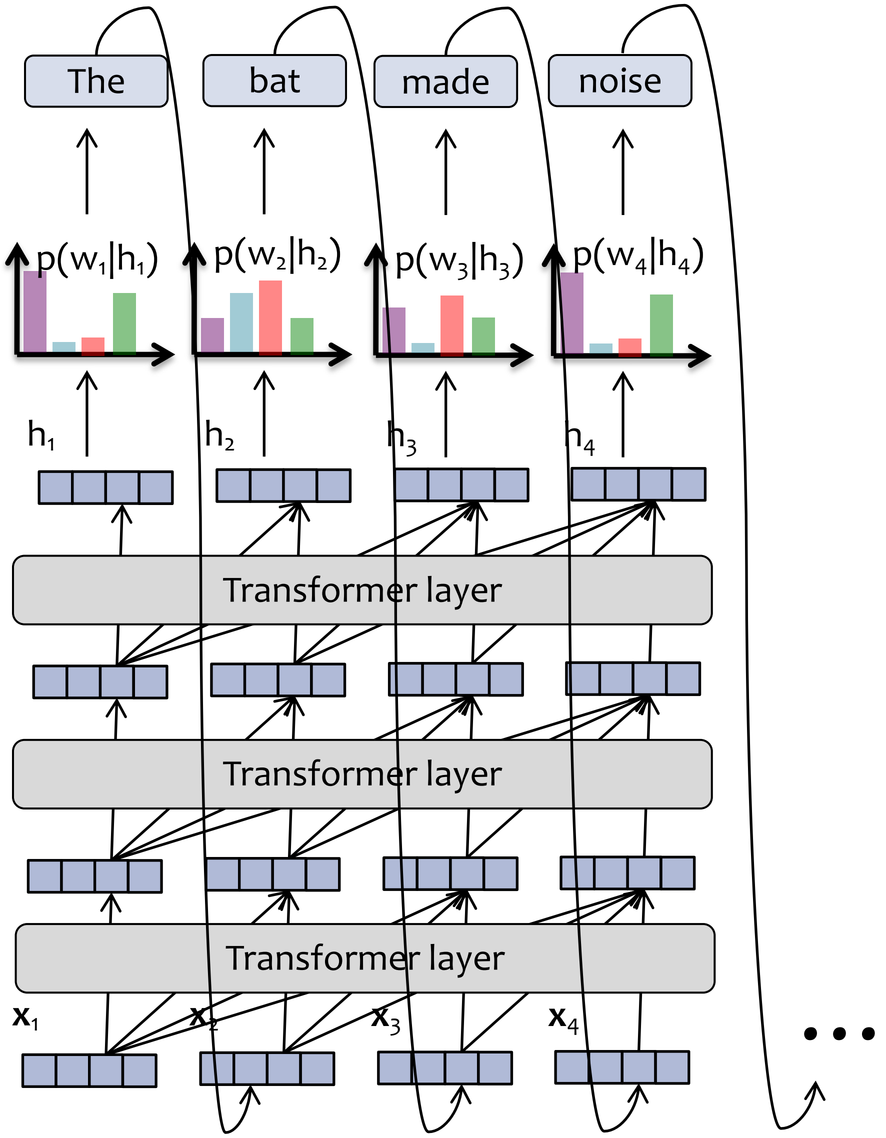
SSM as a Neural Network Layer

- We can take H copies of the 1D recurrent representation
- Let each copy have its own parameters



- This is just like multiple (indep.) heads in Attention
- And just like multiple (indep.) channels in Convolution
- So we get…

Transformer Language Model



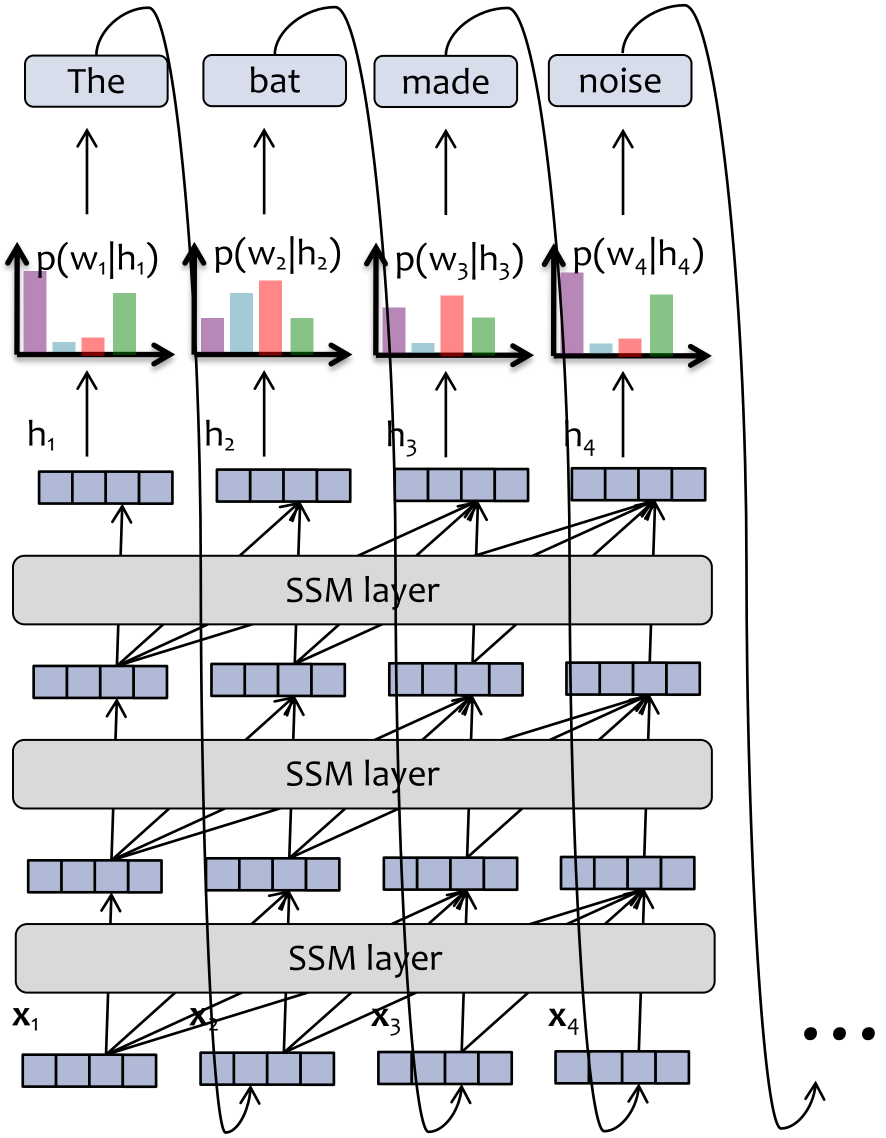
Each layer of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

The language model part is just like an RNN-LM.

SSM inside a Deep Language Model



Each layer of an S4 LM consists of several **sublayers as well** including an SSM, nonlinearity, etc.

Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

The language model part is just like an RNN-LM or Transformer-LM

Efficiency of SSM, RNN, & Transformer

For SSMs:

1. At test time, generation does NOT need a KV-cache in our **Recurrent representation**, so we can effortlessly generate truly long sequences (unlike Transformers, but just like RNNs)
2. At train time, we can use the **Convolution representation** to do fast parallel training (just like Transformers, but unlike RNNs)

	Train	Test
Recurrence		
Attention		
SSM		

S4 Model

We need several additional tricks to get training to work well:

- HiPPO Matrix
  - we initialize the matrix A very carefully
- Efficient computation
  - we decompose A so that we can compute the kernel K very efficiently and in a numerically stable way

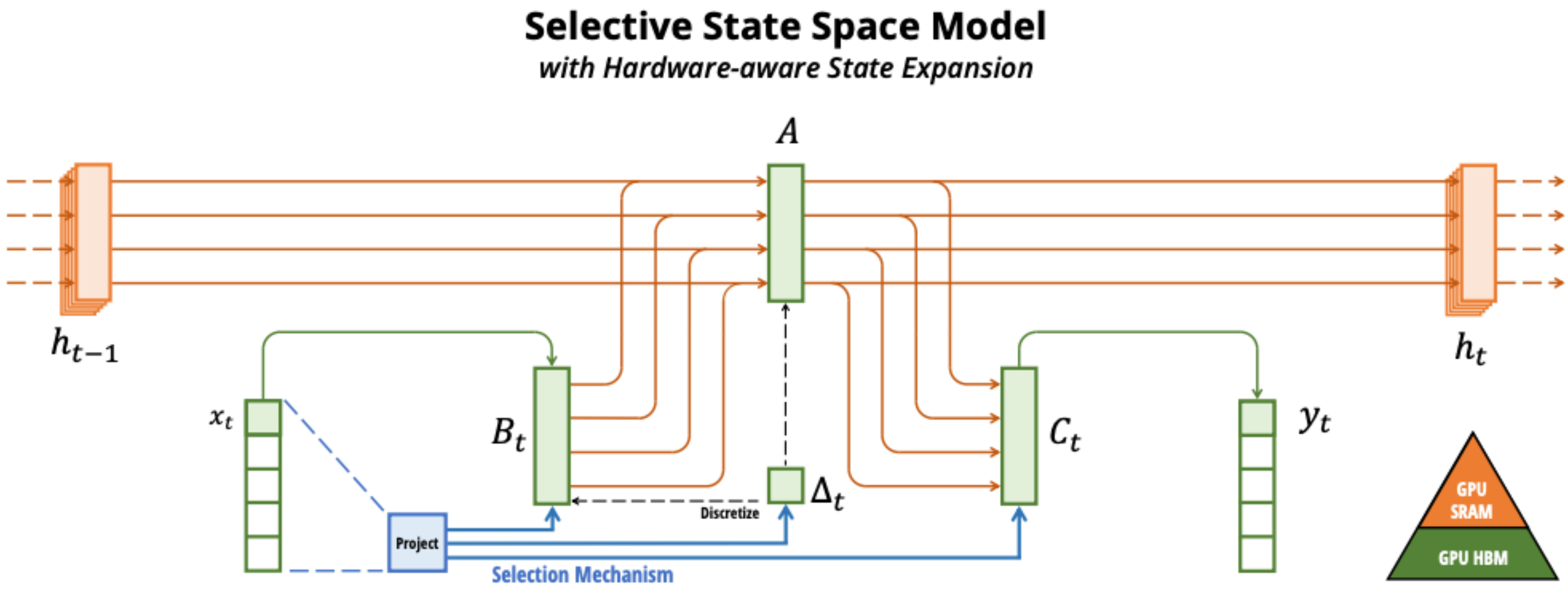
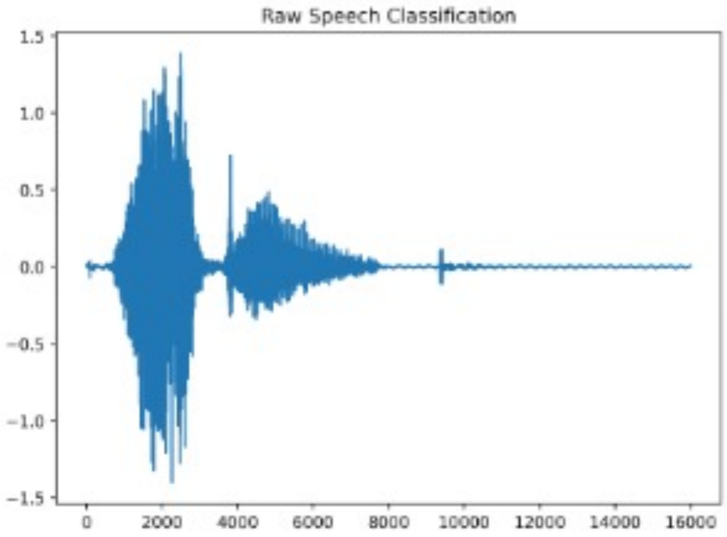


Figure 1: (**Overview**.) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate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

Figure from https://arxiv.org/pdf/2312.00752

S4 Results: Train and test on different input granularities



	MFCC	Raw (Train: 16K Hz)	0.5× (Test: 8K Hz)
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**

Figure from https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3

MAMBA

Selective State Space Models

Algorithm 1 SSM (S4)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

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

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

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

- **Selective** state space models differ from S4 in that they let the parameters B and C vary at each timestep

Selective State Space Models

Algorithm 1 SSM (S4)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

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

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

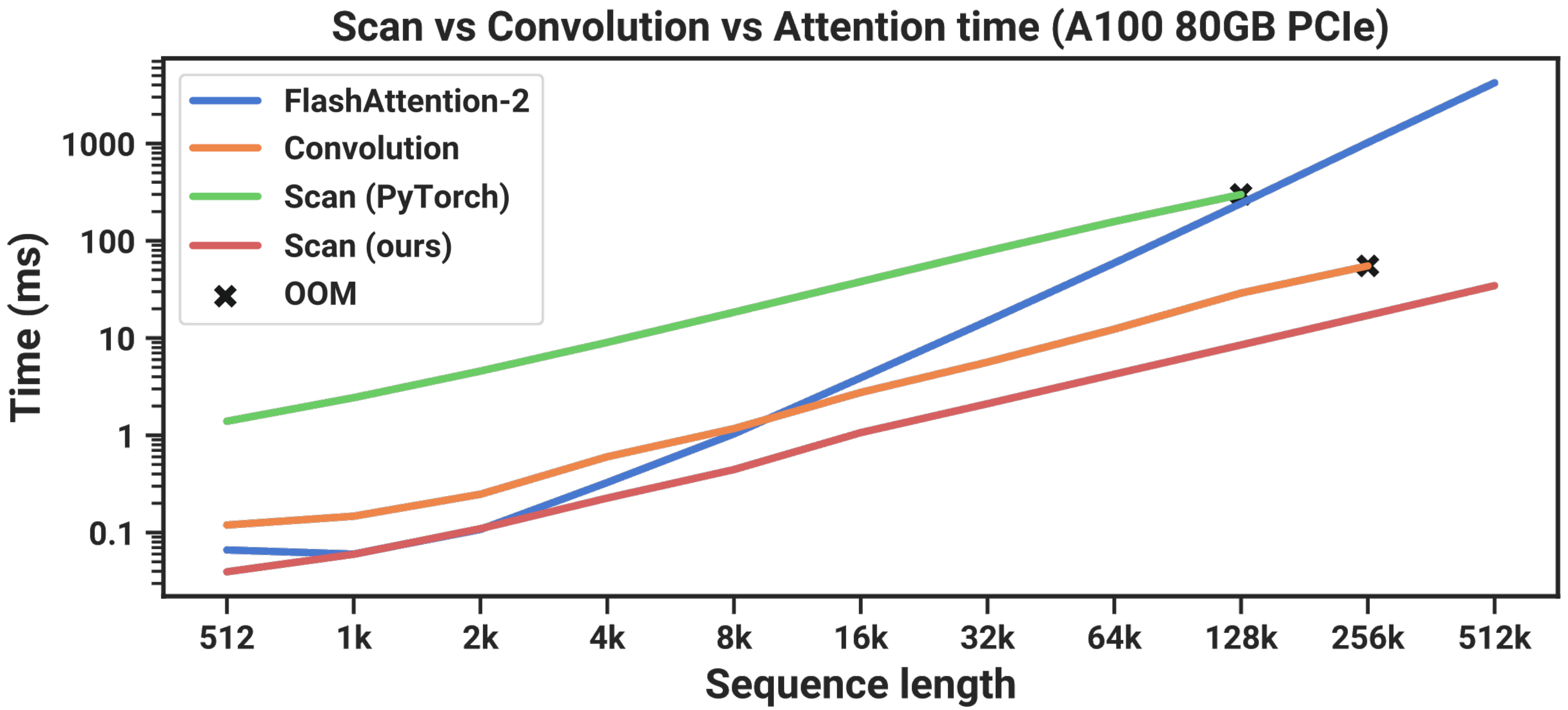
$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{Parameter}$
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
$\Delta : (\mathsf{B}, \mathsf{L},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

$$h_t = Ah_{t-1} + Bx_t$$
$$y_t = C^\top h_t$$

$$h_t = A_t h_{t-1} + B_t x_t$$
$$y_t = C_t^\top h_t$$

Mamba’s Scan Implementation

- We can no longer compute the kernel K once up front
- Instead we perform an efficient scan implementation



https://jackcook.com/2024/02/23/mamba.html

Mamba Results

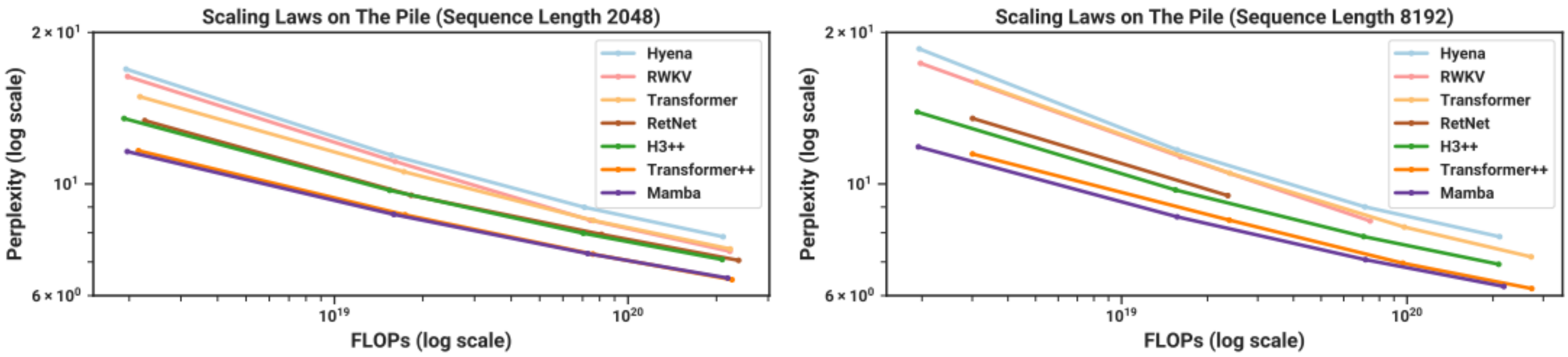
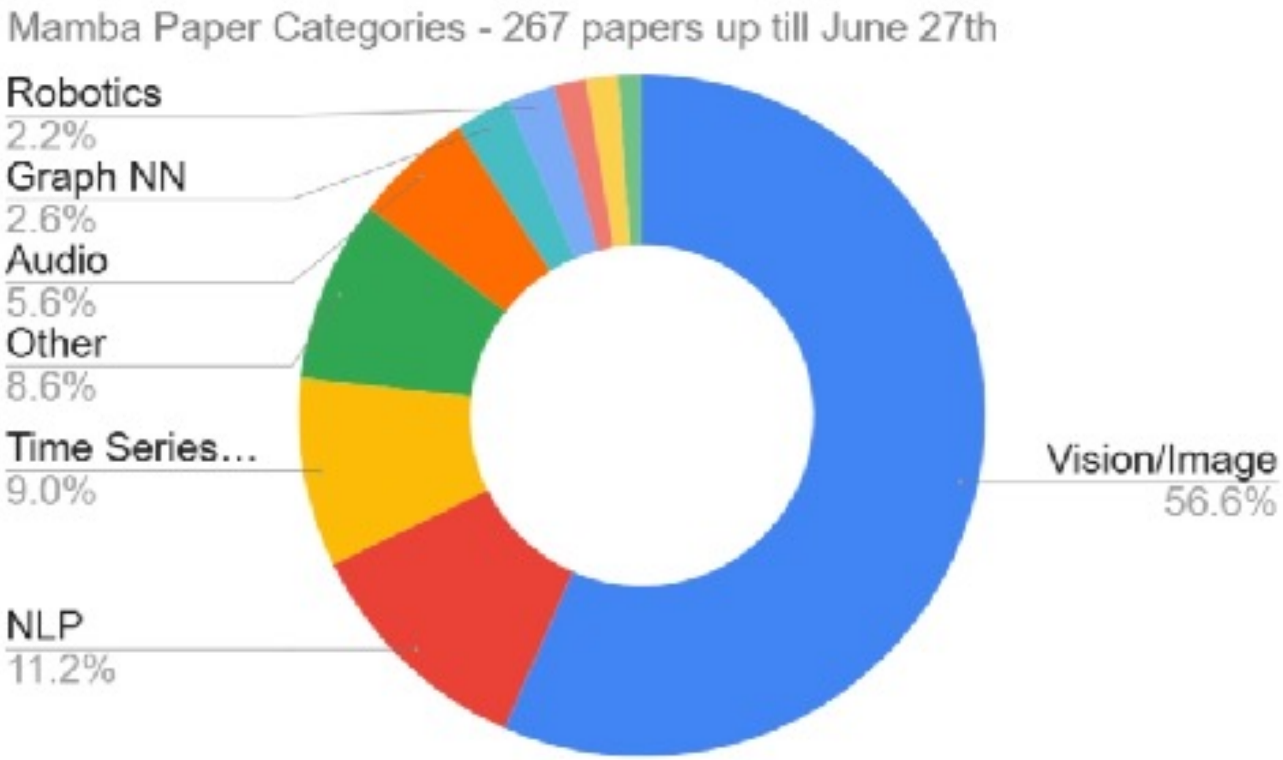


Figure 4: (**Scaling Laws.**) Models of size ≈ 125*M* to ≈ 1.3*B* parameters, trained on the Pile. Mamba scales better than all other attention-free models and is the first to match the performance of a very strong "Transformer++" recipe that has now become standard, particularly as the sequence length grows.

- Main takeaway: Mamba is the first (only?) non-attention based LM to challenge a Transformer

https://arxiv.org/pdf/2312.00752

Mamba Use in the Real World



Strong out-of-the-box on **general modalities**

(not just language!)

Figure from Albert Gu's F24 guest lecture in 10-423/623

Motivation

- https://www.isattentionallyouneed.com/

Is Attention All You Need?

Current Status: Yes

Time Remaining: 631d 22h 55m 11s

Proposition:

On January 1, 2027, a Transformer-like model will continue to hold the state-of-the-art position in most benchmarked tasks in natural language processing.