Machine Learning Department
School of Computer Science
Carnegie Mellon University

State Space Models

+

Hybrid Models

Matt Gormley & Aran Nayebi
Lecture 22
Nov. 12, 2025

Motivation

- Transformers are slow at test time: they require a KV cache that grows linearly 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 16KHz vs. 8KHz)

Motivation

- https://www.isattentionallyouneed.com/

Is Attention All You Need?

Current Status: Yes

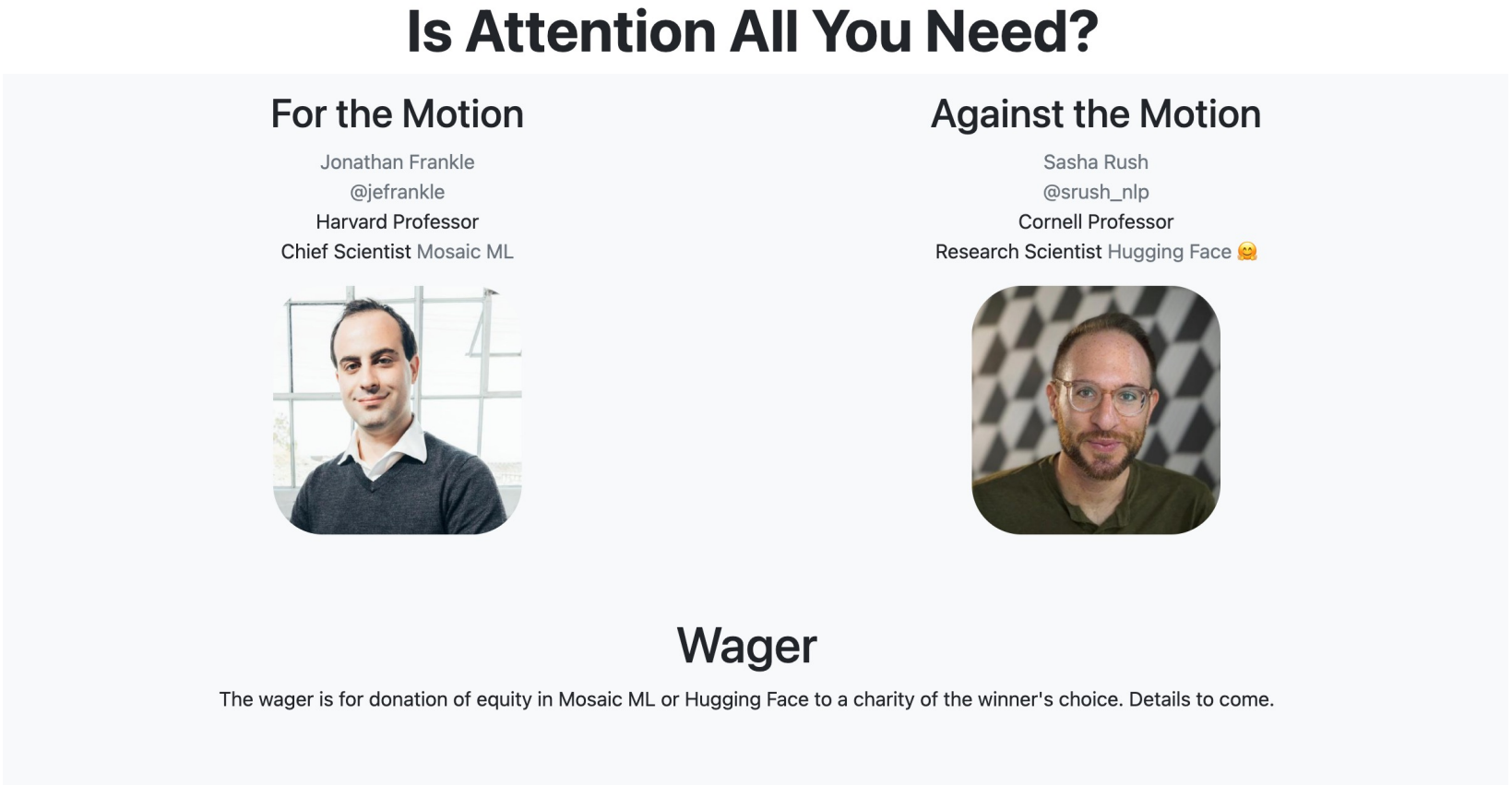
Time Remaining: 631d 22h 55m 11s

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

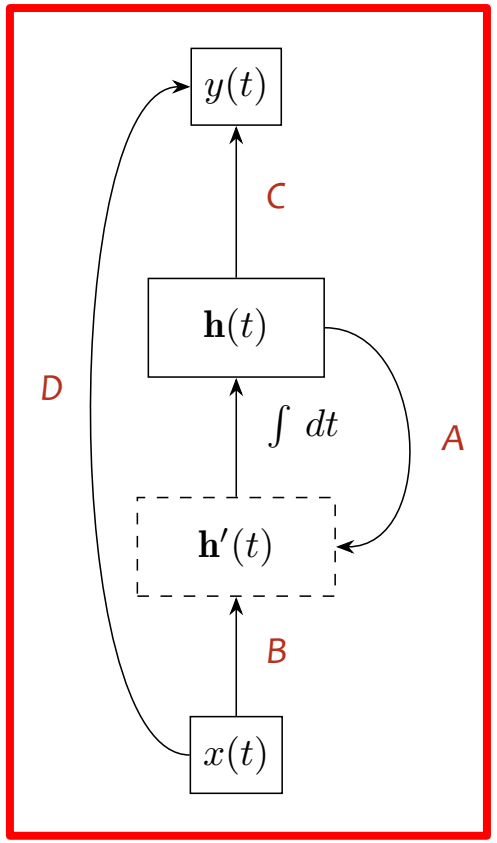
- https://www.isattentionallyouneed.com/



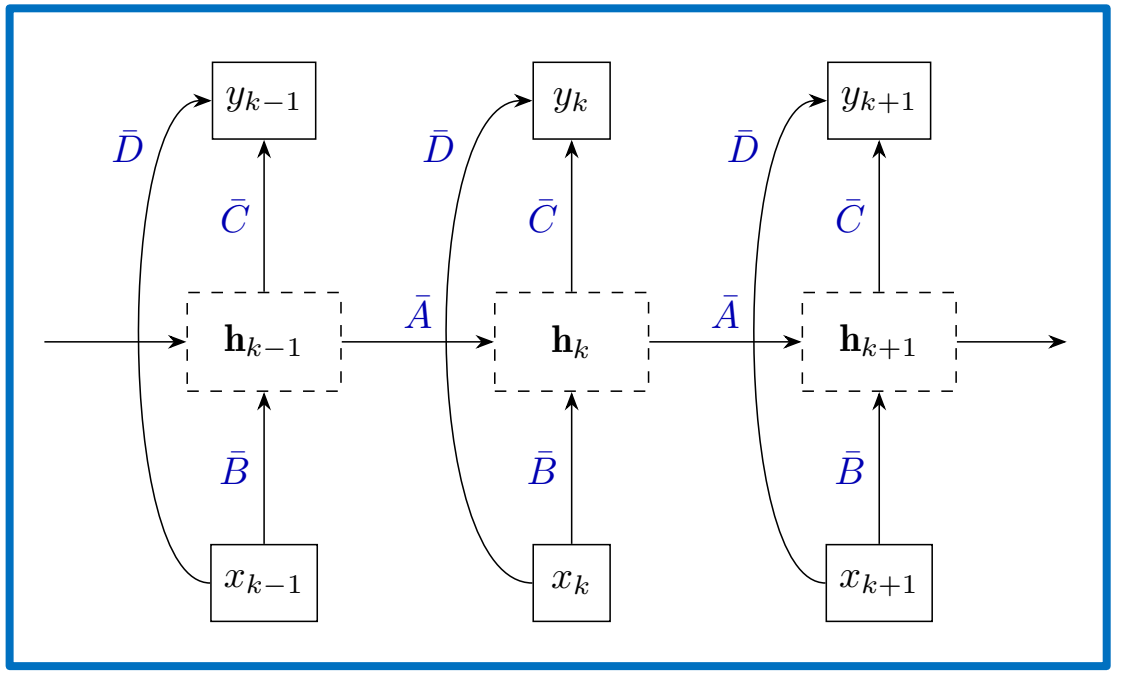
STATE SPACE MODEL (SSM)

Three Representations of a State Space Model

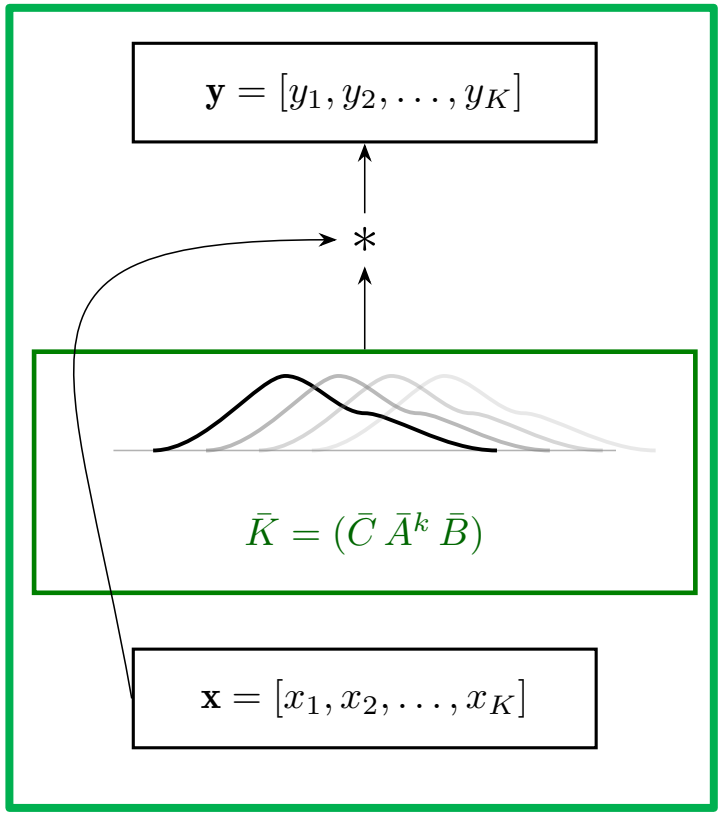
Continuous



Recurrent



Convolutional



Figures inspired by https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3

SSM: 1D Continuous Representation

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

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

$x(t) \in \mathbb{R}$

$y(t) \in \mathbb{R}$

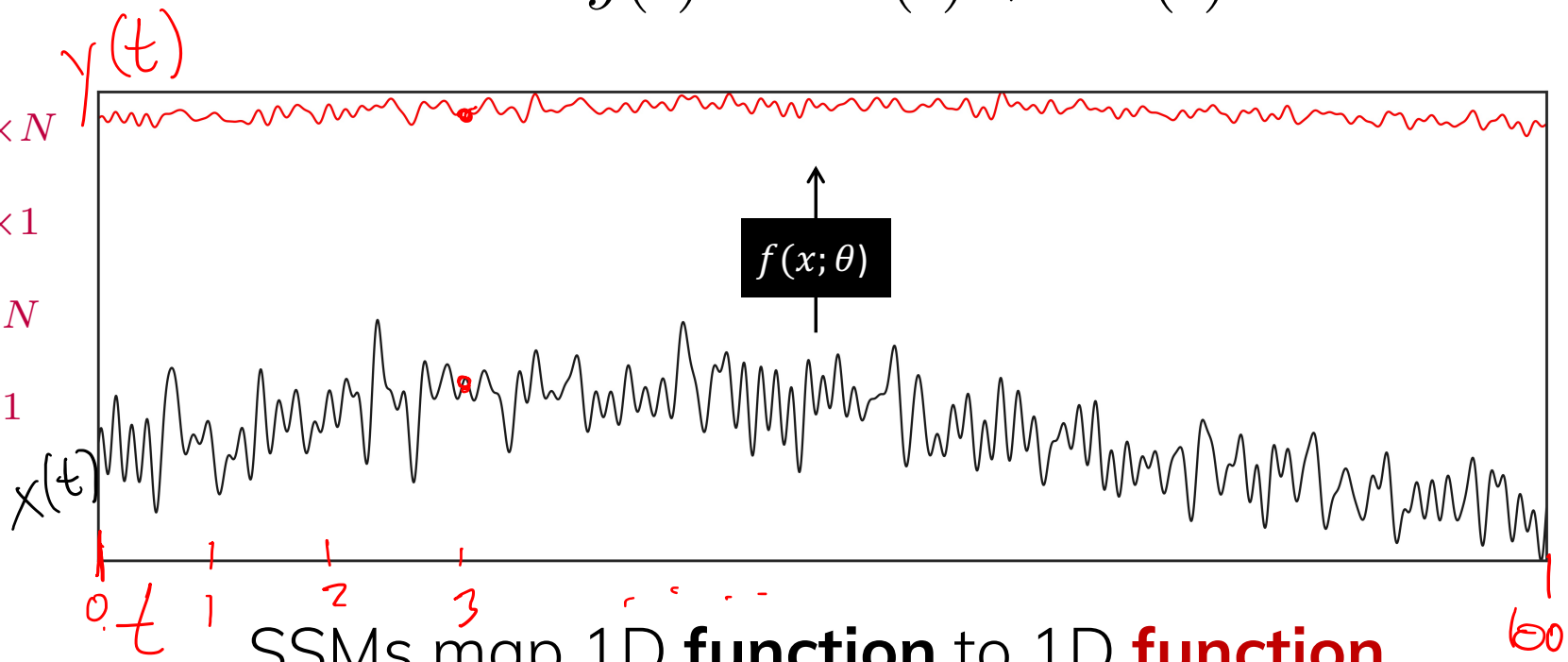
$\mathbf{h}(t) \in \mathbb{R}^N$

$A \in \mathbb{R}^{N \times N}$

$B \in \mathbb{R}^{N \times 1}$

$C \in \mathbb{R}^{1 \times N}$

$D \in \mathbb{R}^{1 \times 1}$



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

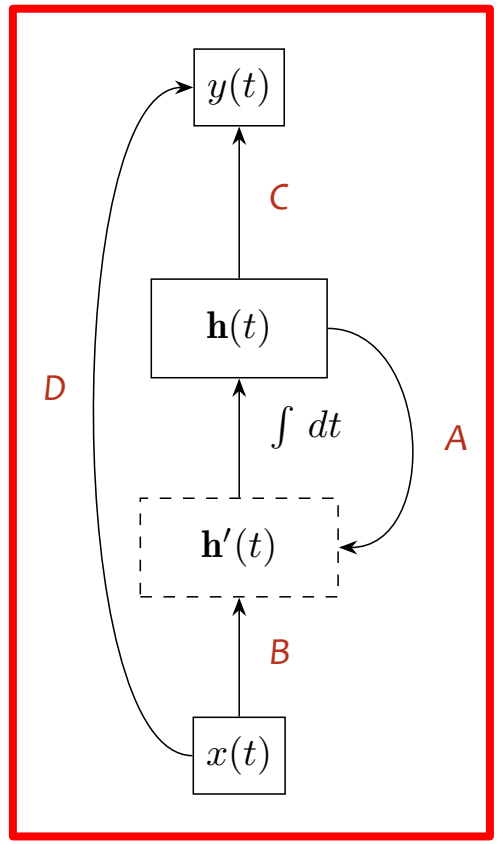


Figure from Albert Gu's F24 guest lecture in 10-423/623

SSM: 1D Continuous Representation

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

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

$x(t) \in \mathbb{R}$

$y(t) \in \mathbb{R}$

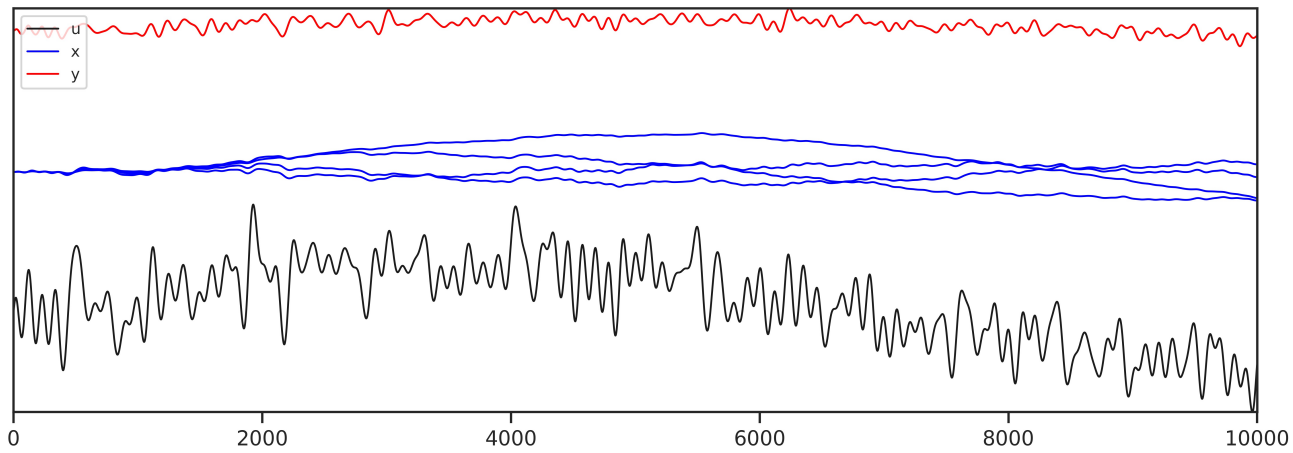
$\mathbf{h}(t) \in \mathbb{R}^N$

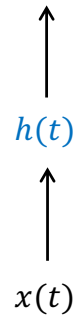
$A \in \mathbb{R}^{N \times N}$

$B \in \mathbb{R}^{N \times 1}$ 1-D

$C \in \mathbb{R}^{1 \times N}$ N-D

$D \in \mathbb{R}^{1 \times 1}$ 1-D





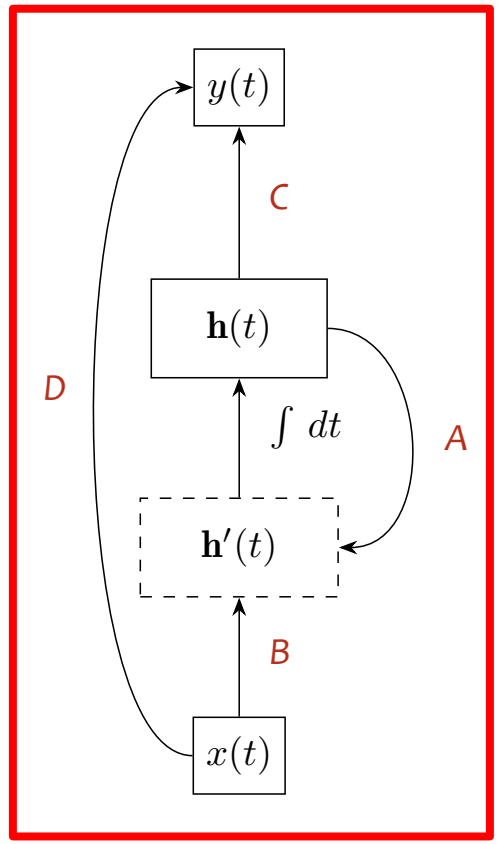


Figure from Albert Gu's F24 guest lecture in 10-423/623

SSM: 1D Continuous Representation

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

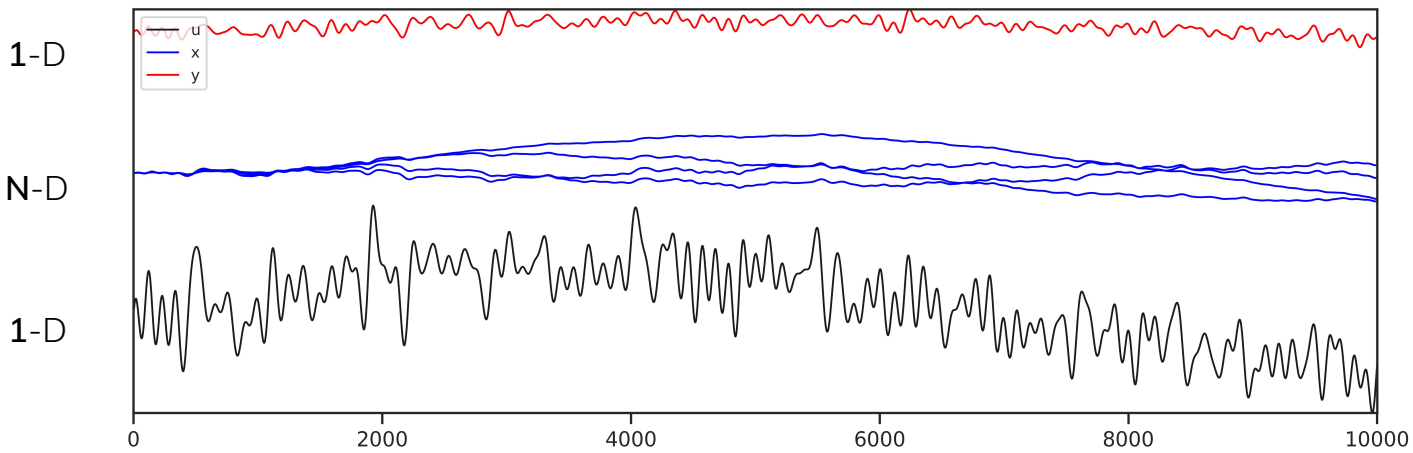
$$y(t) = \mathbf{C}\mathbf{h}(t) + \mathbf{D}x(t)$$

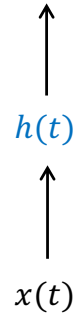
$\vec{h}(t) = \in \mathbb{R}^N$

$h_1(t) = \square$

$h_2(t) = \square$

$h_N(t) = \square$





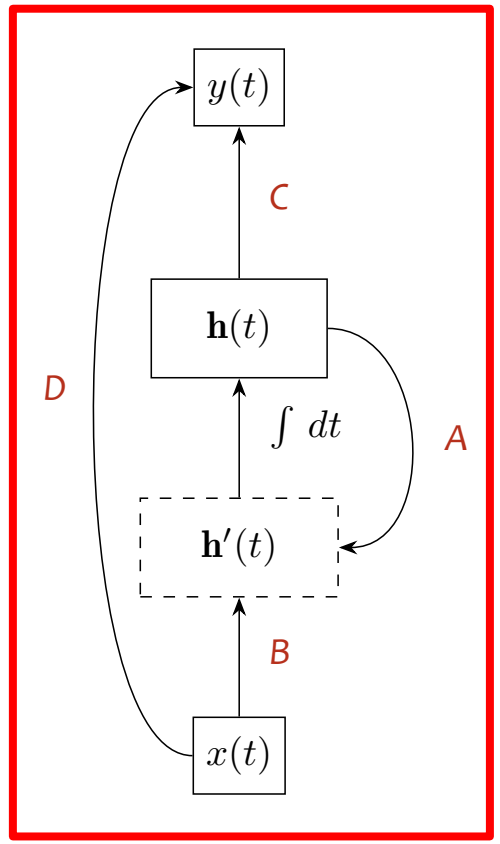


Figure from Albert Gu's F24 guest lecture in 10-423/623

SSM: 1D Discrete Recurrent Representation

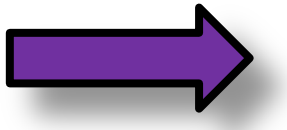
Continuous Representation

- 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

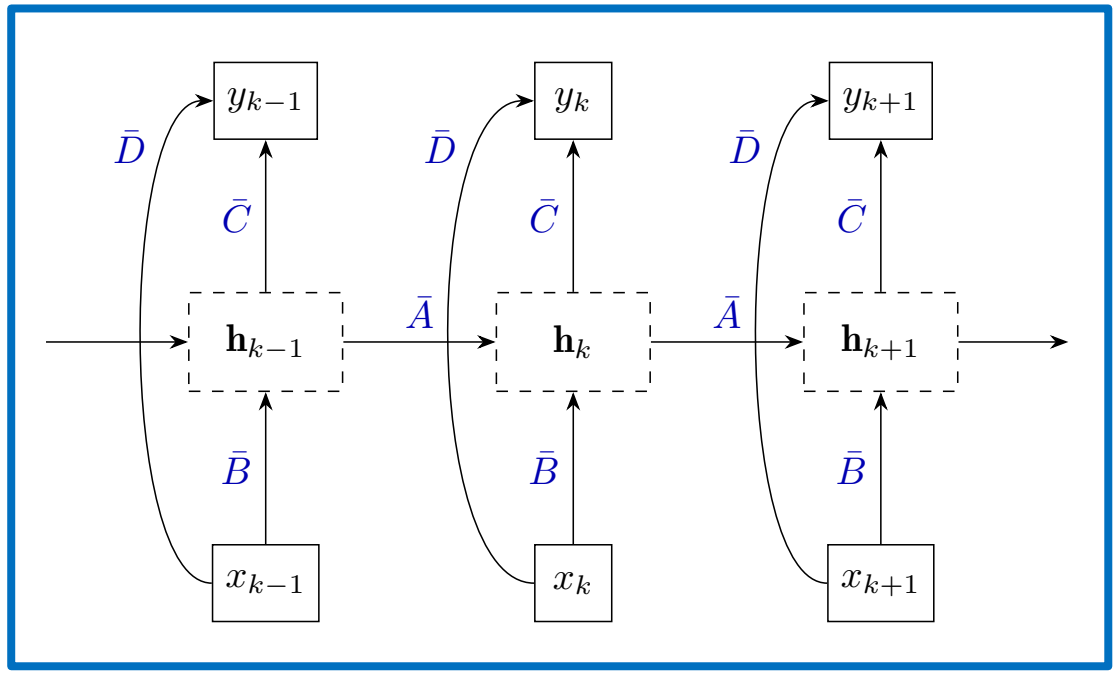
$$\mathbf{h}'(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}x(t)$$
$$y(t) = \mathbf{C}\mathbf{h}(t) + \mathbf{D}x(t)$$



$$\mathbf{h}_{k+1} = \bar{\mathbf{A}}\mathbf{h}_k + \bar{\mathbf{B}}x_k$$
$$y_k = \bar{\mathbf{C}}\mathbf{h}_k + \bar{\mathbf{D}}x_k$$

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

no activation function

no bias term

$$\mathbf{h}_{k+1} = \bar{\mathbf{A}}\mathbf{h}_k + \bar{\mathbf{B}}x_k$$

$$y_k = \bar{\mathbf{C}}\mathbf{h}_k + \bar{\mathbf{D}}x_k$$

1D

residual connection

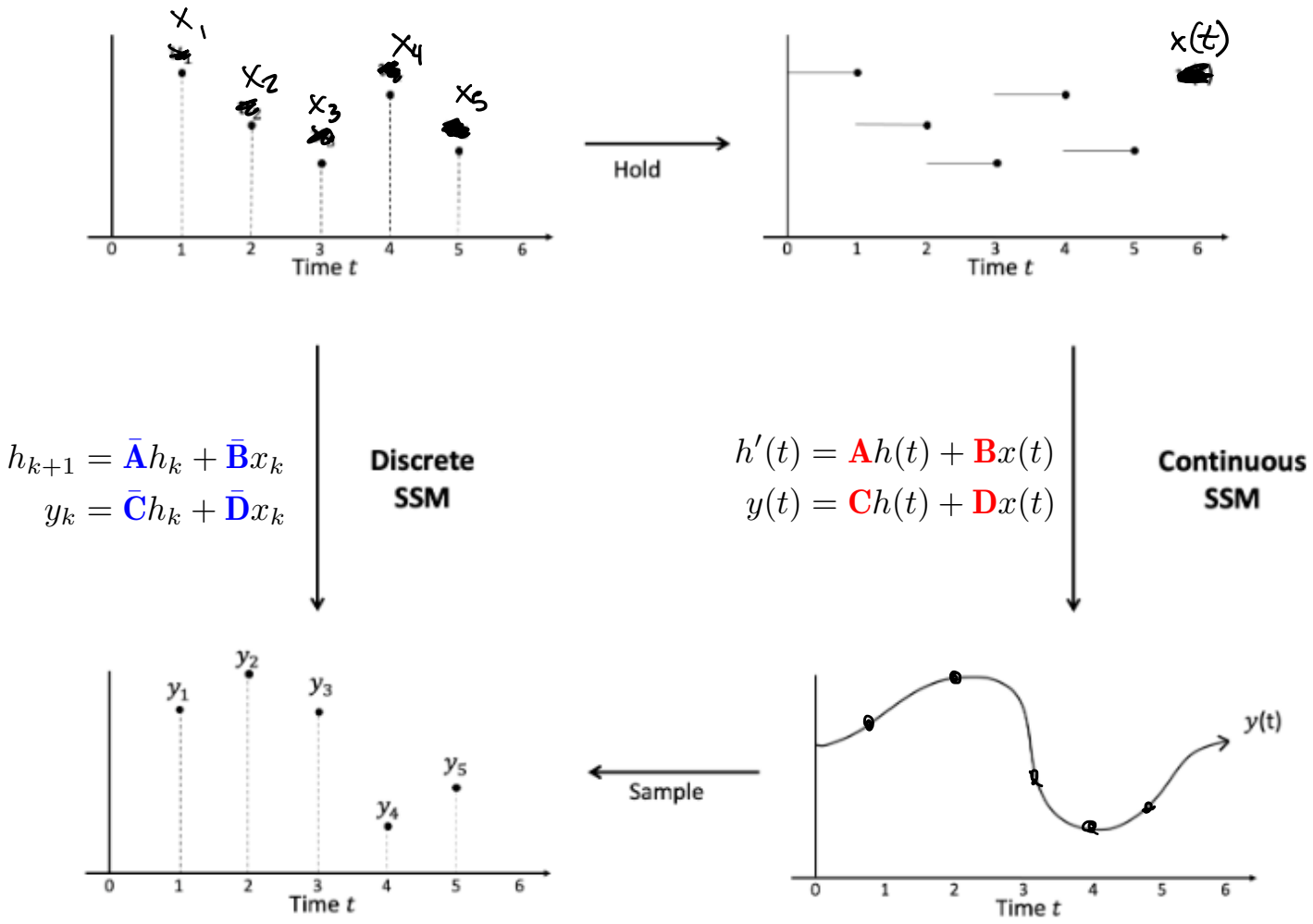
How to discretize a continuous SSM?

S4 uses a bilinear transformation to discretize the continuous SSM

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

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

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

Δ = size of the discrete timestep

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

part of a PyTorch layer where the parameters are A, B, C, D not $\bar{A}, \bar{B}, \bar{C}, \bar{D}$

The bilinear transformation uses a first order Pade approximation:

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

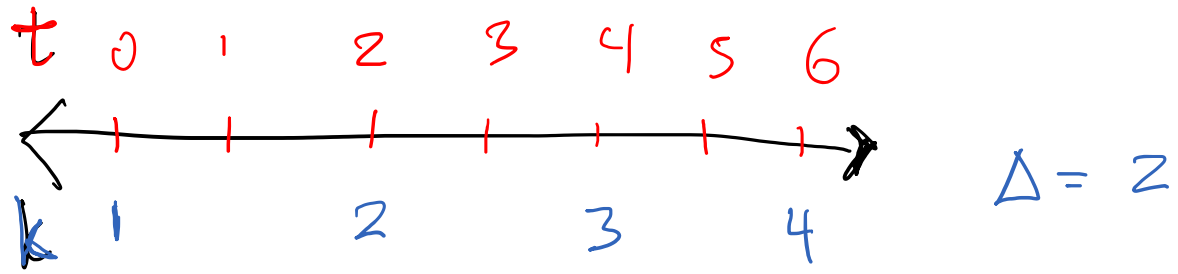


Figure from https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3

SSM: 1D Convolutional Representation

$h_{k+1} = \bar{A} h_k + \bar{B} x_k$

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

$$h_0 = \bar{\mathbf{B}} x_0$$
$$h_1 = \bar{\mathbf{A}}\bar{\mathbf{B}} x_0 + \bar{\mathbf{B}} x_1$$
$$h_2 = \bar{\mathbf{A}}^2\bar{\mathbf{B}} x_0 + \bar{\mathbf{A}}\bar{\mathbf{B}} x_1 + \bar{\mathbf{B}} x_2$$
$$\vdots$$

$y_k = \bar{C} h_k + \bar{D} x_k$ (ignore)

$$y_0 = \bar{\mathbf{C}}\bar{\mathbf{B}} x_0$$
$$y_1 = \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}} x_0 + \bar{\mathbf{C}}\bar{\mathbf{B}} x_1$$
$$y_2 = \bar{\mathbf{C}}\bar{\mathbf{A}}^2\bar{\mathbf{B}} x_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}} x_1 + \bar{\mathbf{C}}\bar{\mathbf{B}} x_2$$
$$\vdots$$

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

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

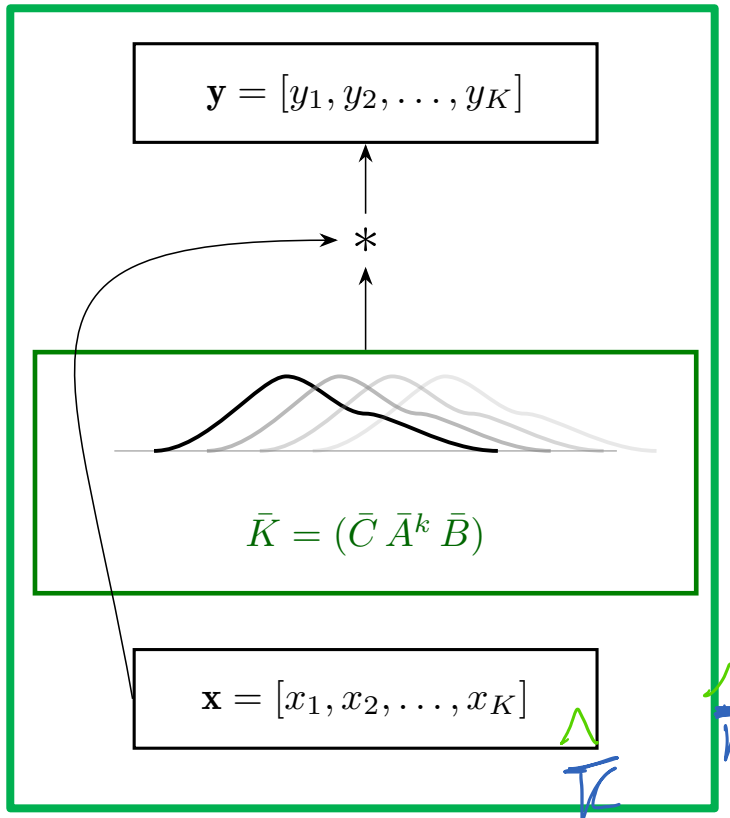
$$\mathbf{y} = \bar{\mathbf{K}} * \mathbf{x}$$

L = sequence length

where the SSM convolution kernel is:

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

SSM: 1D Convolutional Representation



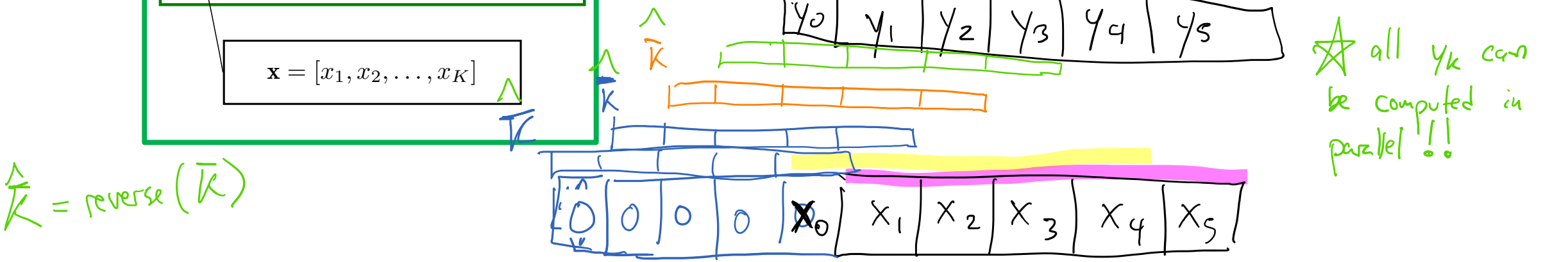
We can represent this as a *global* convolution computation:

constructor this is another PyTorch layer and a fn of A,B,C,D

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

where the SSM convolution kernel is:

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

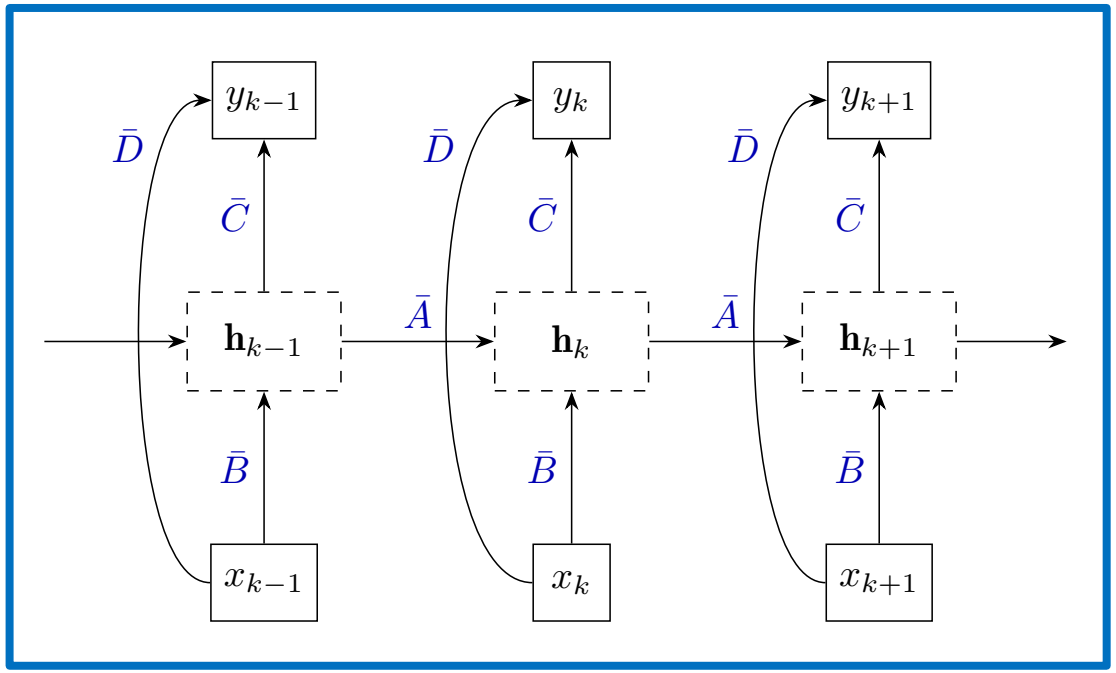


$$y_k = \bar{\mathbf{C}}\bar{\mathbf{A}}^k\bar{\mathbf{B}}x_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}^{k-1}\bar{\mathbf{B}}x_1 + \cdots + \bar{\mathbf{C}}\bar{\mathbf{A}}^1\bar{\mathbf{B}}x_{k-1} + \bar{\mathbf{C}}\bar{\mathbf{A}}^0\bar{\mathbf{B}}x_k$$

THE STRUCTURED STATE SPACE SEQUENCE MODEL (S4)

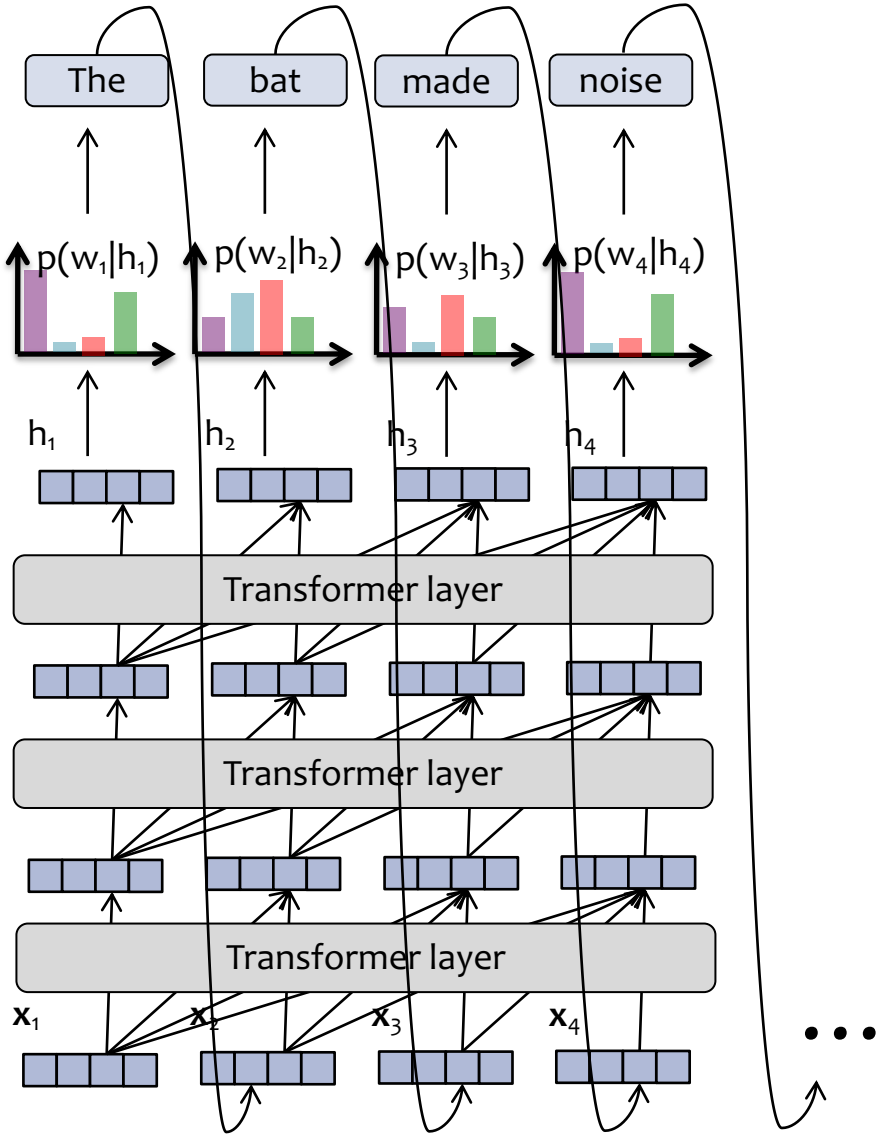
SSM as a Neural Network Layer

- 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



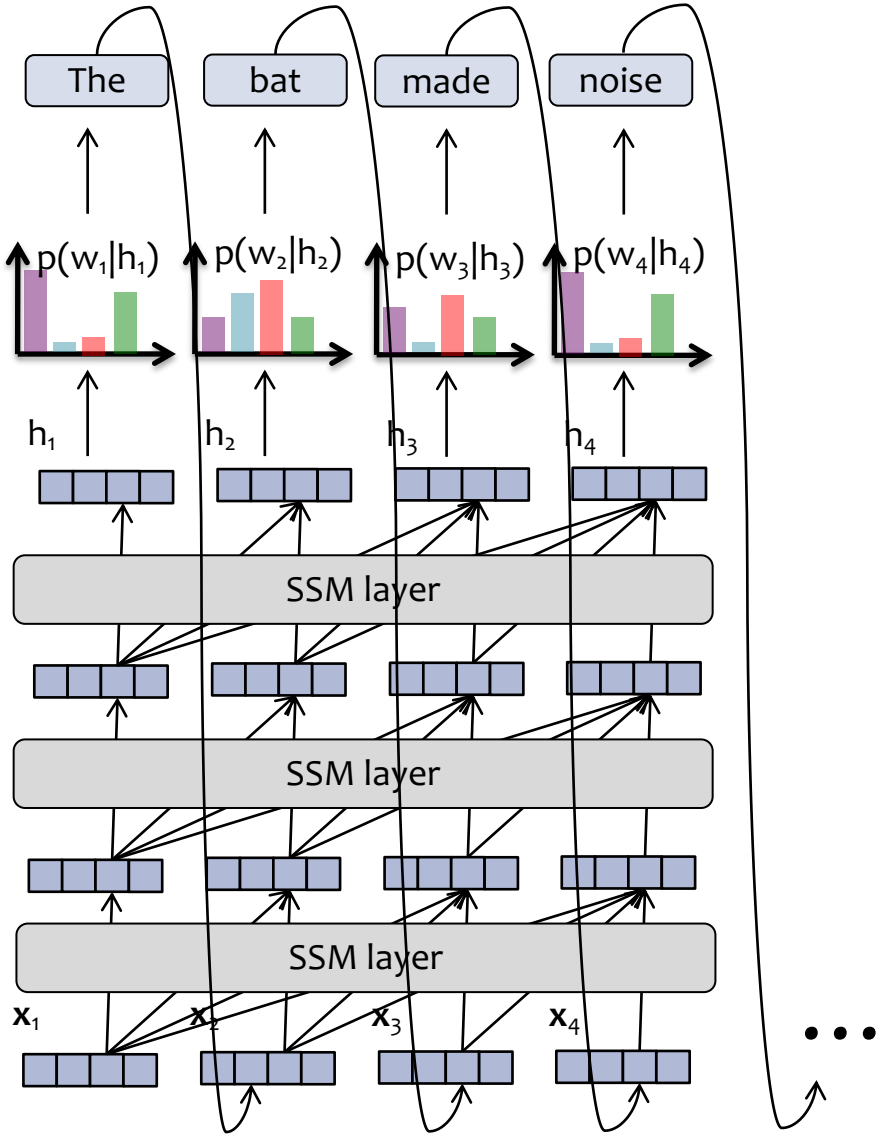
Each layer of a Transformer LM consists of several **sublayers:**

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~~ Attn.

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	slow	fast
Attention	fast	slow
SSM	fast	fast

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

$$K = \left(\ldots, \ldots, \bar{C}\bar{A}^{L-1}, \bar{B} \right)$$

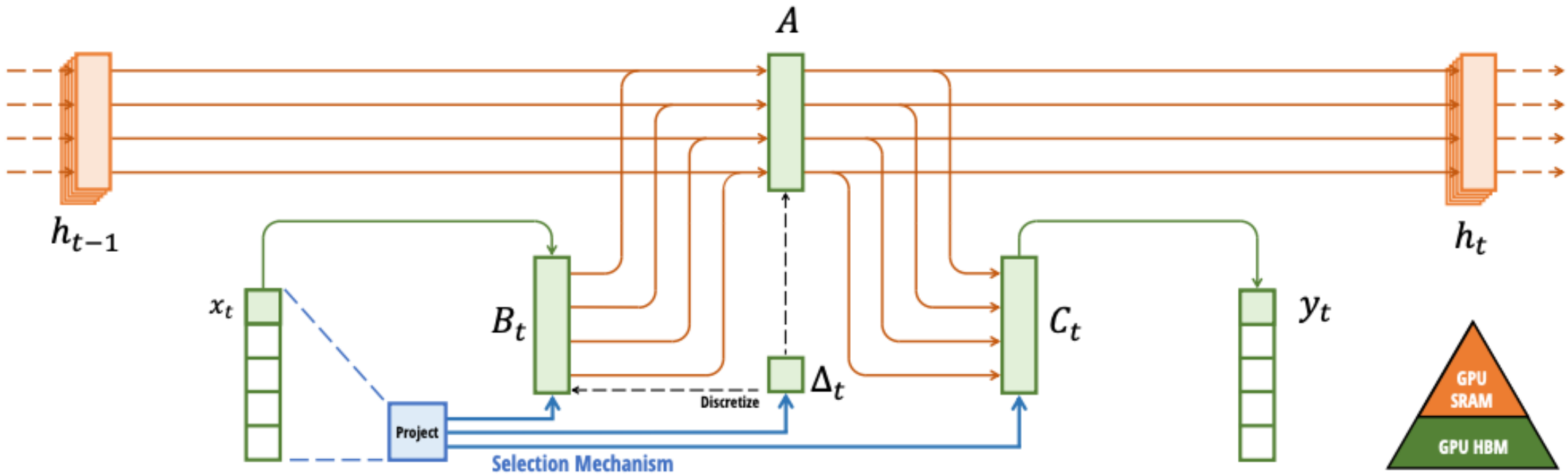
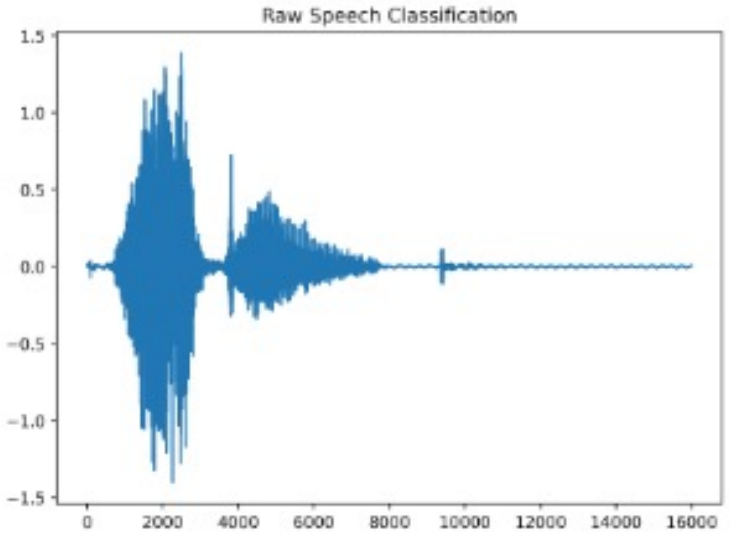


Figure 1: (**Overview.**) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate computation paths requiring time-invariance: the (Δ, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$) 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



		Train: 16K Hz	Test: 8K Hz
	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**

Learn continuous params A, B, C, D at 16KHz

Then change Δ, but keep params.

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: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
1: $\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
3: $\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
4: $\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter})$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

Algorithm 2 SSM + Selection (S6)

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

$$h'(t) = A\,h(t) + B\,x(t)$$
$$y(t) = C\,h(t)$$

- **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: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

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

Algorithm 2 SSM + Selection (S6)

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

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

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

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

Mamba's Scan Implementation

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

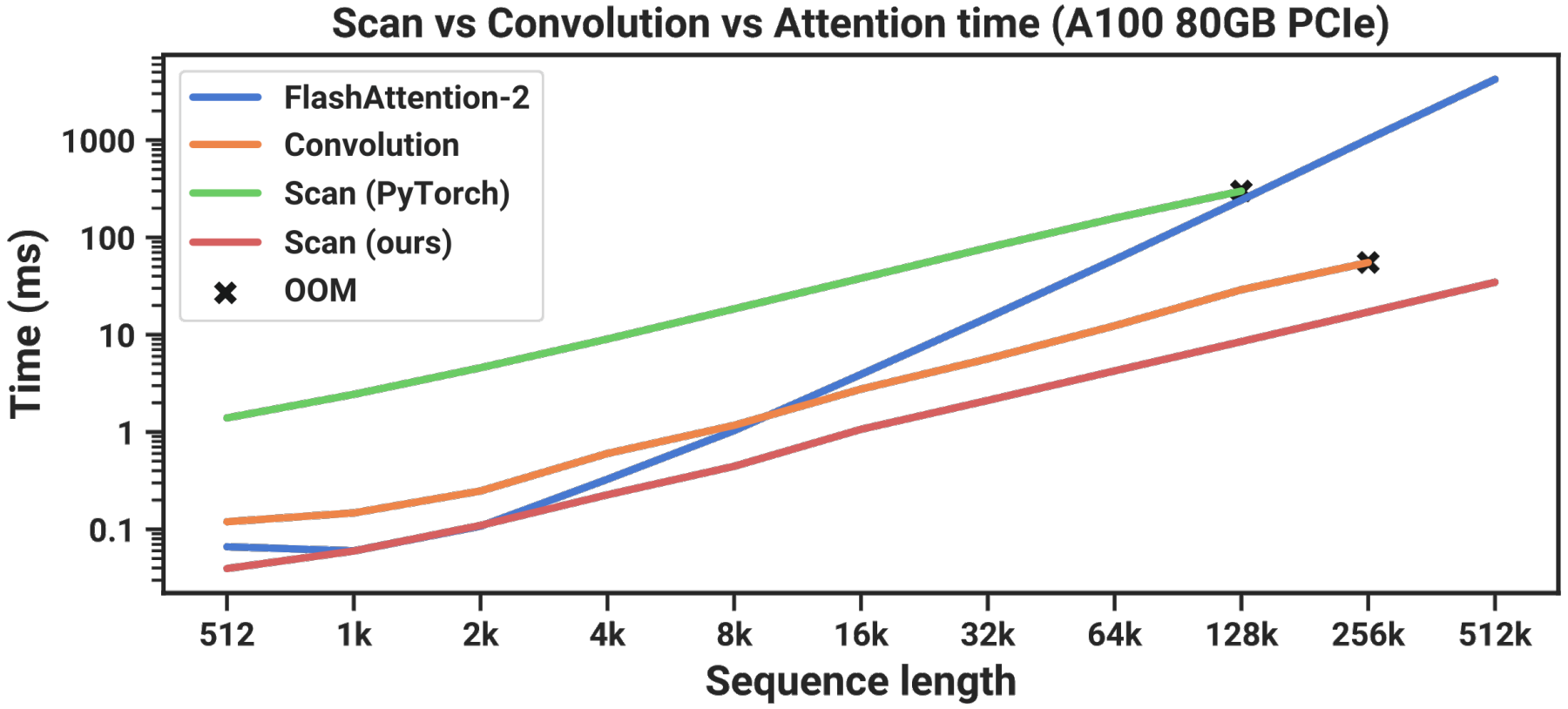


Figure from https://jackcook.com/2024/02/23/mamba.html

Mamba Results

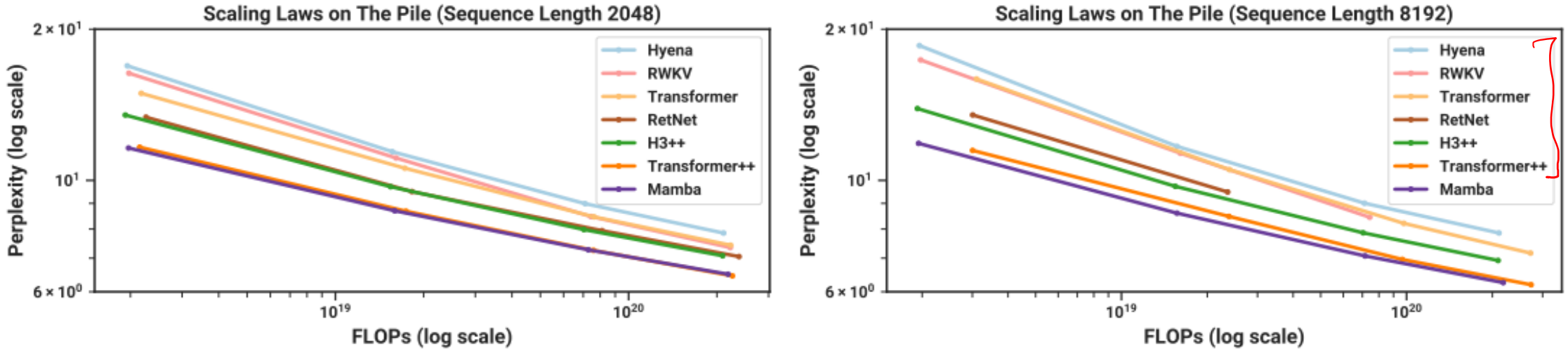
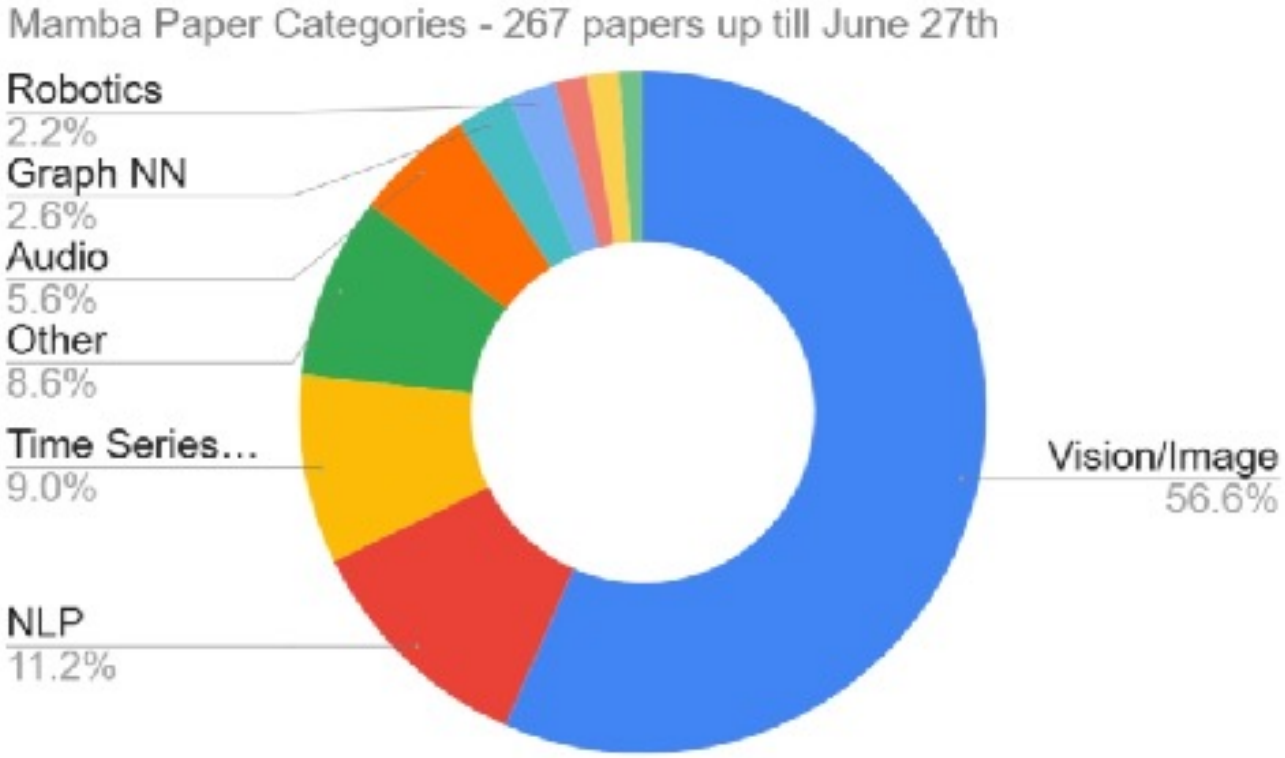


Figure 4: (**Scaling Laws.**) Models of size $\approx 125M$ to $\approx 1.3B$ parameters, trained on the Pile. Mamba scales better than all other attention-free models and is the first to match the performance of a very strong "Transformer++" recipe that has now become standard, particularly as the sequence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

- Main takeaway: Mamba was the first non-attention based LM to challenge a Transformer

Figure from 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

LINEAR ATTENTION AND SSMS

Linear Attention

- Linear attention is identical to standard attention, but drops the softmax: (below M is the causal mask)

$$\mathbf{O} = (\mathbf{Q}\mathbf{K}^\intercal \odot \mathbf{M})\mathbf{V} \in \mathbb{R}^{L \times d_v} \qquad \boldsymbol{o}_t = \sum_{i=1}^{t} (\boldsymbol{v}_i \boldsymbol{k}_i^\intercal)\boldsymbol{q}_t = \sum_{i=1}^{t} \boldsymbol{v}_i (\boldsymbol{k}_i^\intercal \boldsymbol{q}_t) \in \mathbb{R}^{d_v},$$

- And can be expressed as a recurrence:

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal \in \mathbb{R}^{d_v \times d_k}, \qquad \boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t \in \mathbb{R}^{d_v}$$

- Draws a direct connection between Transformers and SSMs

Figure from https://arxiv.org/pdf/2412.06464

Linear Attention

- Linear attention is identical to standard attention, but drops the softmax: (below M is the causal mask)

$$\mathbf{O} = (\mathbf{Q}\mathbf{K}^\intercal \odot \mathbf{M})\mathbf{V} \in \mathbb{R}^{L \times d_v} \qquad \boldsymbol{o}_t = \sum_{i=1}^{t} (\boldsymbol{v}_i \boldsymbol{k}_i^\intercal)\boldsymbol{q}_t = \sum_{i=1}^{t} \boldsymbol{v}_i (\boldsymbol{k}_i^\intercal \boldsymbol{q}_t) \in \mathbb{R}^{d_v},$$

- And can be expressed as a recurrence:

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal \in \mathbb{R}^{d_v \times d_k}, \qquad \boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t \in \mathbb{R}^{d_v}$$

- Various forms of linear attention have been proposed:

Method	Online Update
LA	$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^T$
Mamba2	$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^T$
Longhorn	$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \epsilon \boldsymbol{k}_t \boldsymbol{k}_t^T) + \epsilon_t \boldsymbol{v}_t \boldsymbol{k}_t^T, \epsilon_t = \frac{\beta_t}{1 + \beta_t \boldsymbol{k}_t^\top \boldsymbol{k}_t}$
DeltaNet	$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^T) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^T$
Gated DeltaNet	$\mathbf{S}_t = \mathbf{S}_{t-1}\left(\alpha_t(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^T)\right) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^T$

Figure from https://arxiv.org/pdf/2412.06464

HYBRID MODELS

Hybrid Models

Hybrid models combine the best of Transformers and SSMs

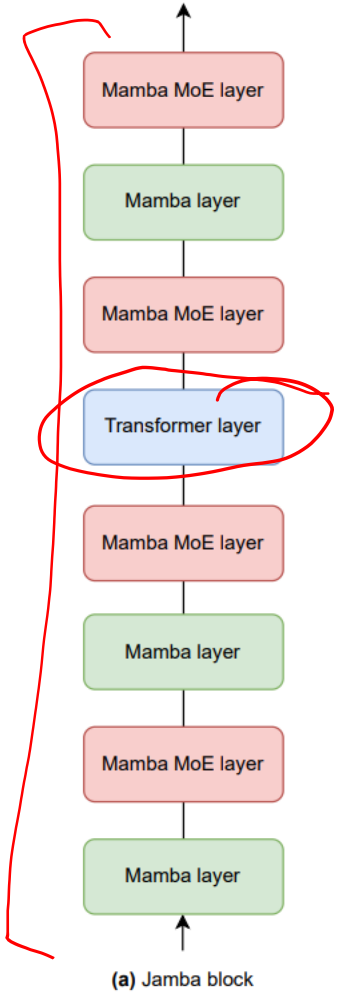
Motivation:

- *Long-context scalability:* Reduce the quadratic cost of self-attention while maintaining or improving modeling ability for sequences spanning tens or hundreds of thousands of tokens.
- *Better hardware utilization:* Exploit architectures that stream activations sequentially (like SSMs) to improve throughput and fit larger contexts on fixed-memory GPUs.
- *Robust generalization:* Mix inductive biases—e.g., attention for global dependencies, convolutional or state-space layers for local and temporal structure—to handle diverse data types (text, audio, vision).

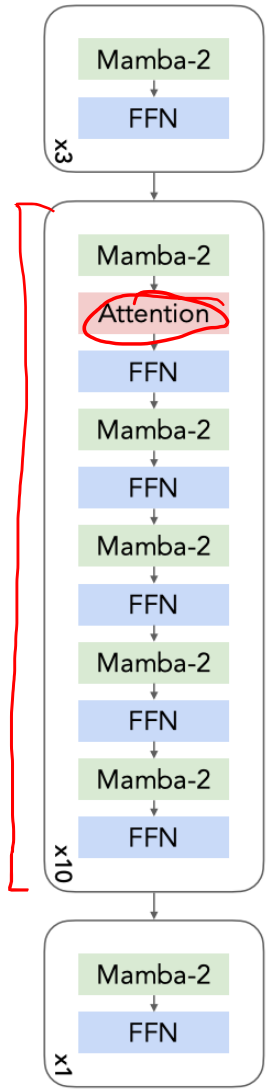
Examples: Jamba, Nemotron-H, Qwen 3 Next

Hybrid Models

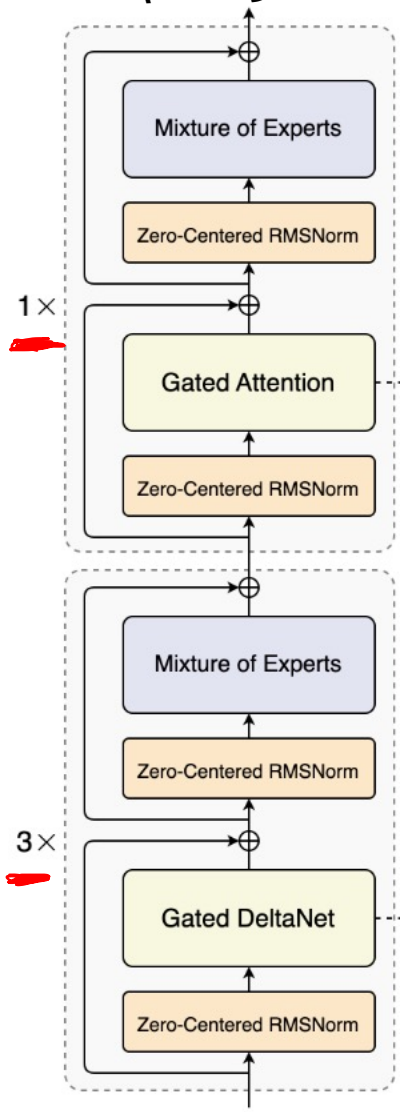
Jamba



Nemotron-H



Qwen 3 Next

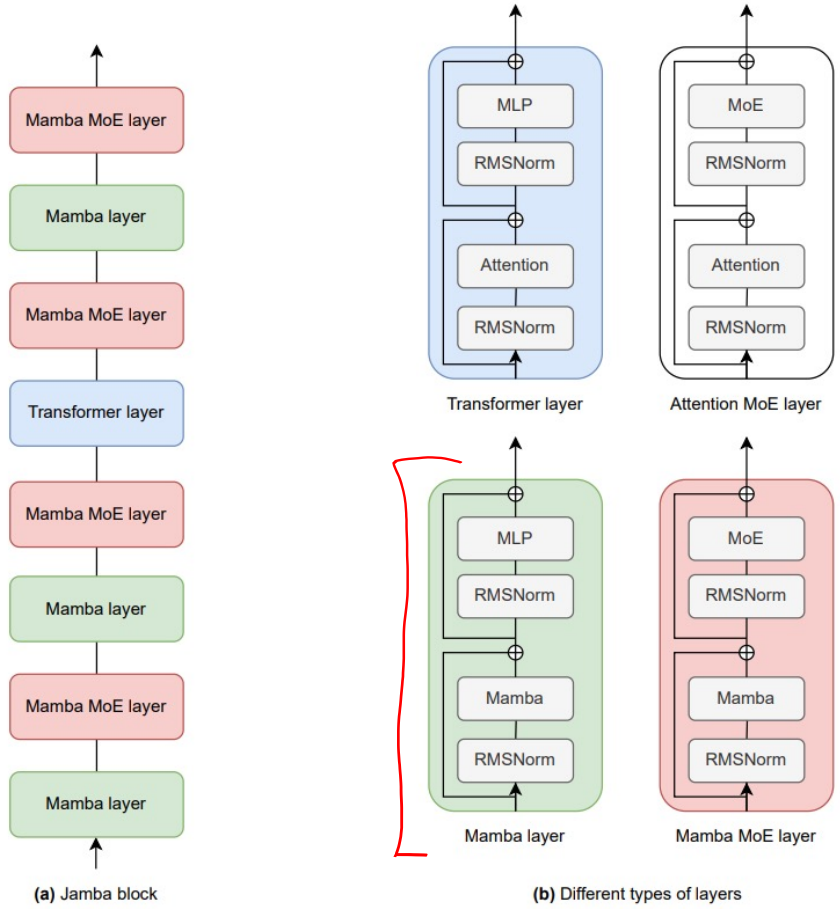


- *Common to all these models:* standard attention layers are **interspersed** with linear attention layers
- The standard attention has **quadratic complexity,** and the linear attention has **linear complexity** in the sequence length
- **Goal:** reduce memory requirements and speed up generation
  (all of these models accomplish this)

Figure from https://arxiv.org/pdf/2403.19887 Figure from https://arxiv.org/pdf/2504.03624 Figure from https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd

Jamba

- Jamba was the **first** hybrid model to combine Transformer layers with SSMs layers (2024)
- The architecture dramatically **reduces the size of the KV cache** for long contexts
- Jamba enables dramatically **longer contexts** to fit on a single GPU than traditional Tranformer LMs
- This enables much **higher throughput** (measured in tokens per second) at generation time

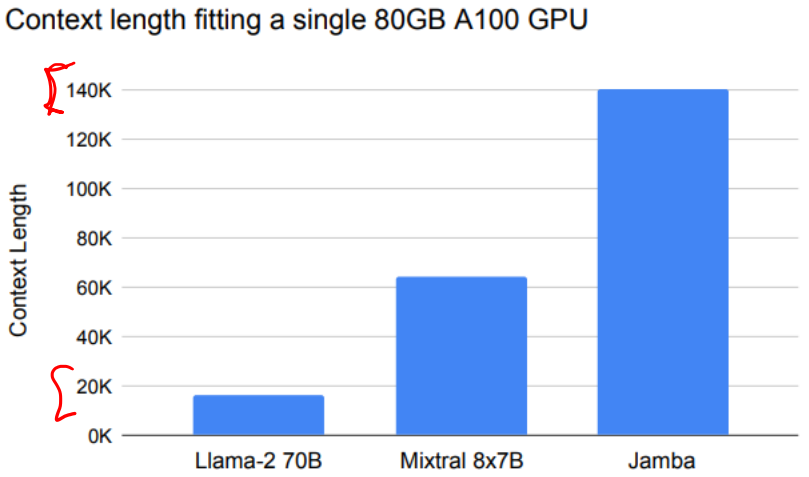


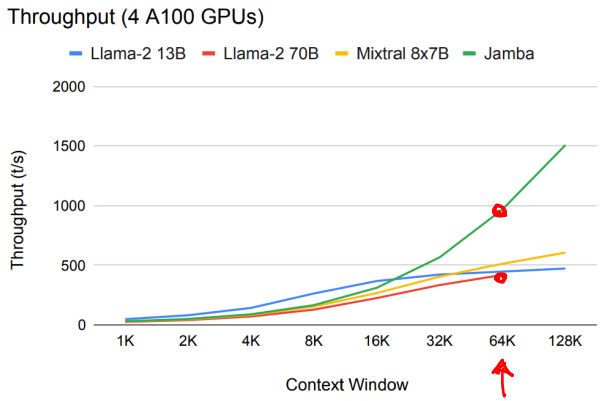
	Available params	Active params	KV cache (256K context, 16bit)
LLAMA-2	6.7B	6.7B	128GB
Mistral	7.2B	7.2B	32GB
Mixtral	46.7B	12.9B	32GB
Jamba	52B	12B	4GB

Figure from https://arxiv.org/pdf/2403.19887

Jamba

- Jamba was the **first** hybrid model to combine Transformer layers with SSMs layers
- The architecture dramatically **reduces the size of the KV cache** for long contexts
- Jamba enables dramatically **longer contexts** to fit on a single GPU than traditional Tranformer LMs
- This enables much **higher throughput** (measured in tokens per second) at generation time

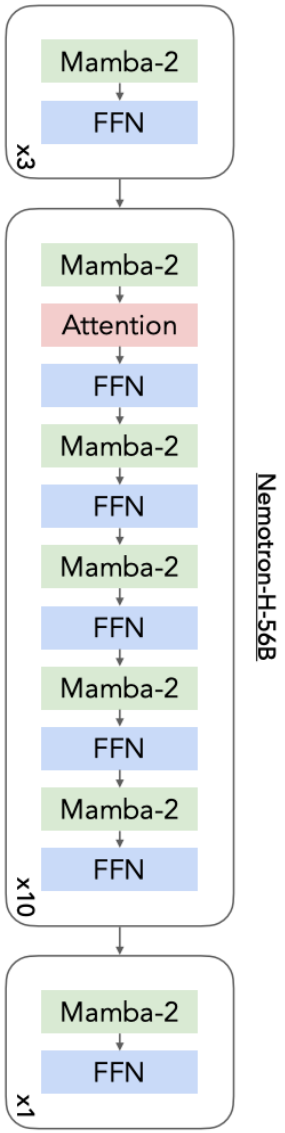




(b) Throughput at different context lengths (single batch, 4 A100 GPUs). With a context of 128K tokens, Jamba obtains 3x the throughput of Mixtral, while Llama-2-70B does not fit with this long context.

Figure from https://arxiv.org/pdf/2403.19887

Nemotron-H



- Nemotron-H demonstrated that performance of a standard Transformer (Nemotron-T) could be retained while gaining dramatic throughput improvements
- Also introduced a vision-language model (VLM) variant

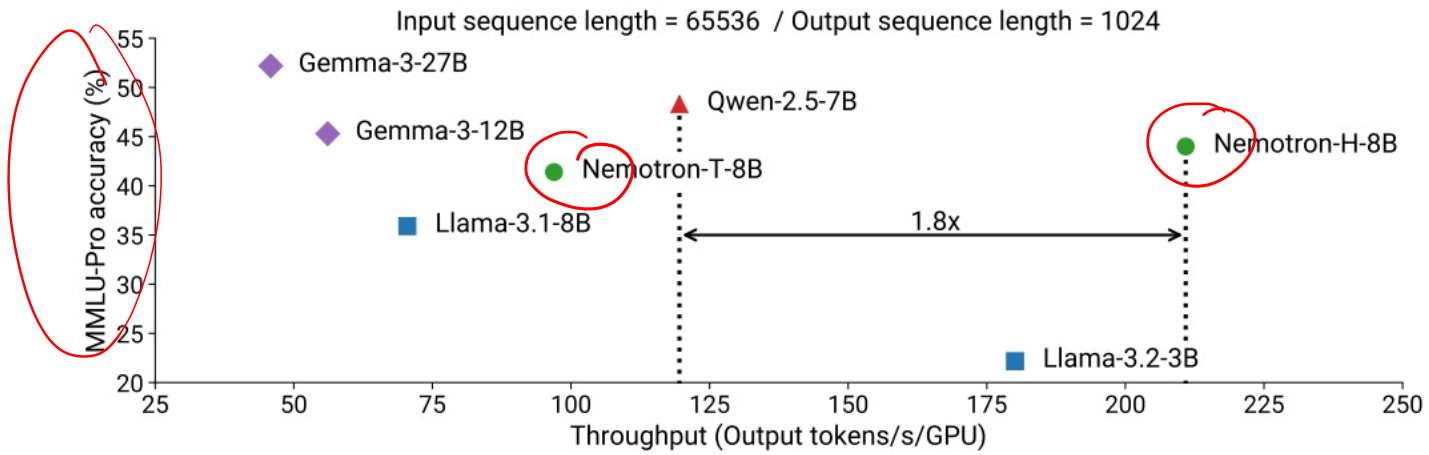
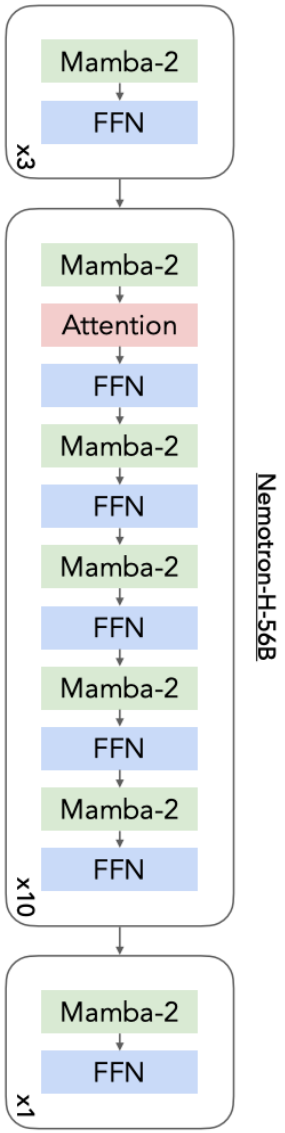


Figure 7 | MMLU-Pro accuracy versus inference throughput (normalized by number of GPUs used) for Nemotron-H-8B-Base compared to existing similarly-sized Transformer models.

Figure from https://arxiv.org/pdf/2504.03624

Nemotron-H



- Nemotron-H demonstrated that performance of a standard Transformer (Nemotron-T) could be retained while gaining dramatic throughput improvements
- Also introduced a vision-language model (VLM) variant

Task	Nemotron-H 56B-VLM	VLM w/ Qwen2.5 72B-Instruct	NVLM-D-1.0 72B (2024-09-17)
MMMU (val)	63.6	**65.1**	62.6†
MathVista	**70.7**	70.5	66.7†
ChartQA	**89.4**	88.9	86.0
AI2D	94.7	**94.9**	94.2
OCRBench	862	**869**	853
TextVQA	81.1	**83.5**	82.1
RealWorldQA	68.4	**71.4**	69.7
DocVQA	**93.2**	92.0	92.6

Figure from https://arxiv.org/pdf/2504.03624

Qwen-3-Next

- Qwen-3-Next uses Gated Delta Net layers instead of standard linear attention layers

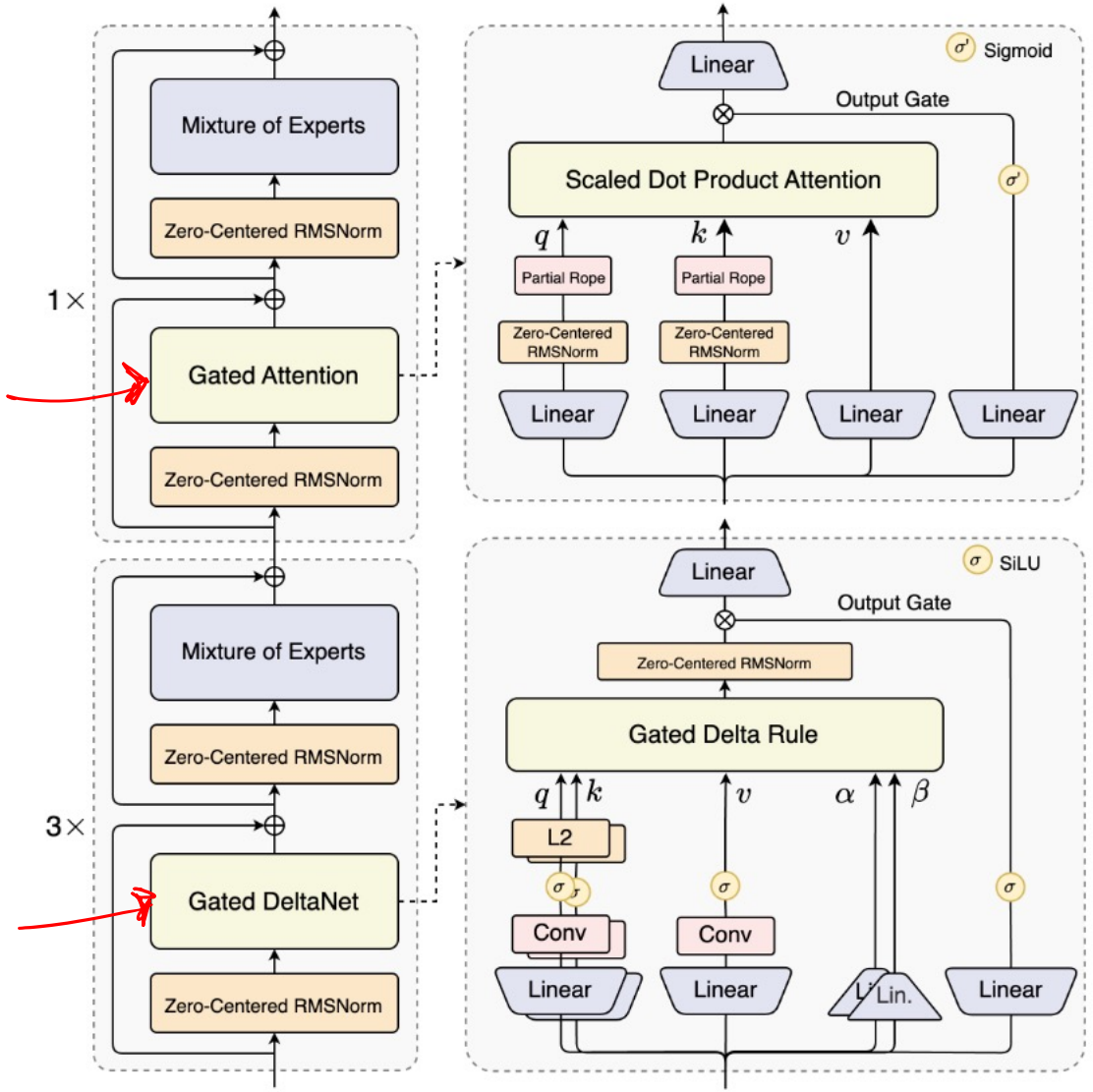


Figure from https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd

Qwen-3-Next

- Qwen-3-Next uses Gated Delta Net layers instead of standard linear attention layers
- Both training time and generation are dramatically reduced, compared to dense models of comparable size

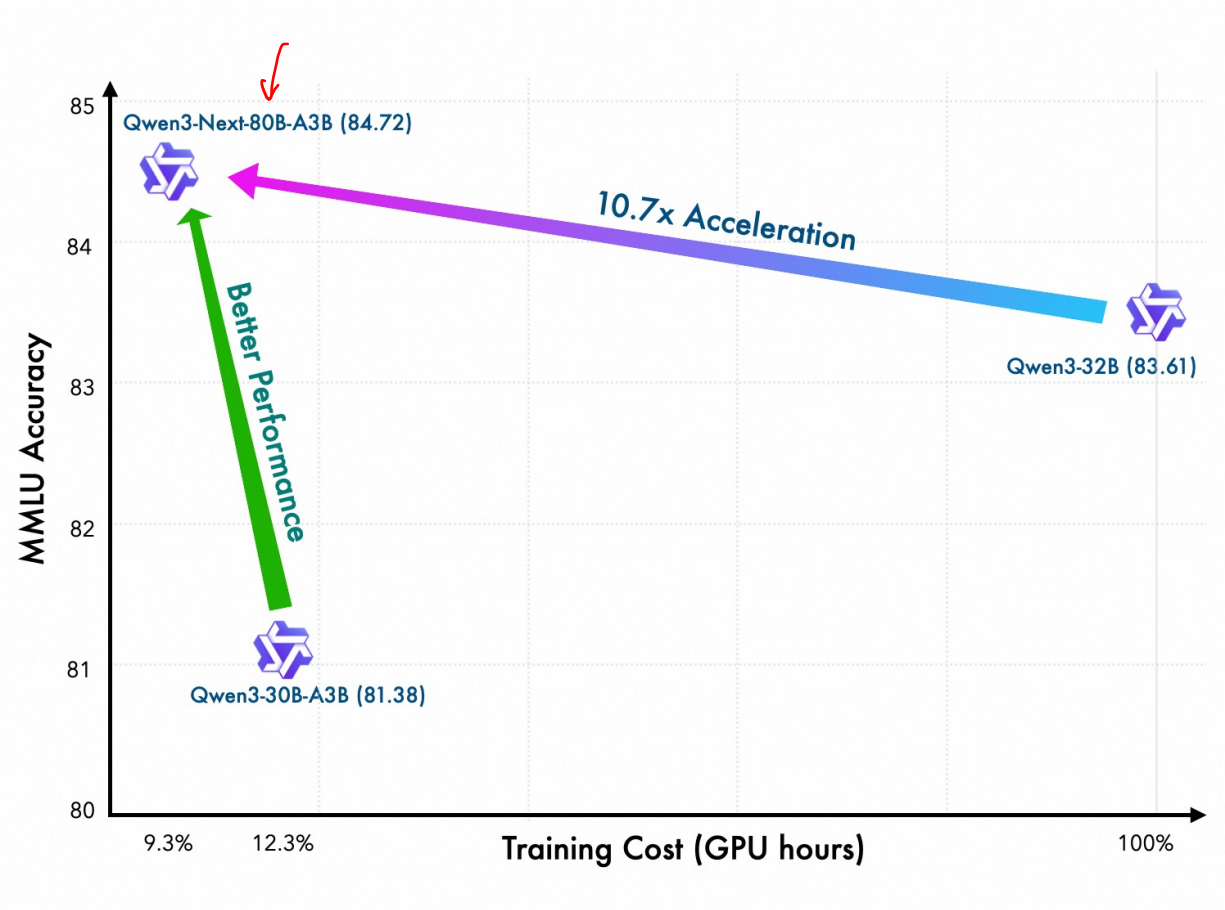


Figure from https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd

Qwen-3-Next

- Qwen-3-Next uses Gated Delta Net layers instead of standard linear attention layers
- Both training time and generation are dramatically reduced, compared to dense models of comparable size
- Impressive performance across standard benchmarks

	Qwen3-30B-A3B Base	Qwen3-32B Base	Qwen3-Next-80B-A3B Base	Qwen3-235B-A22B Base
Architecture	MoE	Dense	MoE	MoE
# Total Params	30B	32B	80B	235B
# Activated Params	3B	32B	3B	22B
General Tasks				
MMLU	81.38	83.61	84.72	**87.81**
MMLU-Redux	81.17	83.41	83.80	**87.40**
MMLU-Pro	61.49	65.54	66.05	**68.18**
SuperGPQA	35.72	39.78	41.52	**44.06**
BBH	81.54	87.38	87.13	**88.87**
Math, STEM & Coding Tasks				
GPQA	43.94	**49.49**	43.43	47.47
GSM8K	91.81	93.40	90.30	**94.39**
MATH	59.04	61.62	62.36	**71.84**
EvalPlus	71.45	72.05	72.89	**77.60**
CRUX-O	67.20	72.50	74.25	**79.00**
Multilingual Tasks				
MGSM	79.11	83.06	81.28	**83.53**
MMMLU	81.46	83.83	84.43	**86.70**
INCLUDE	67.00	67.87	69.79	**73.46**

Figure from https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd

Motivation

- https://www.isattentionallyouneed.com/

Is Attention All You Need?

Current Status: Yes

Time Remaining: 631d 22h 55m 11s

NO!!

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.