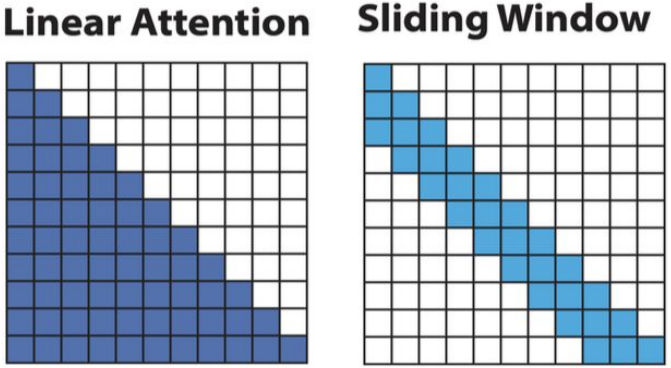
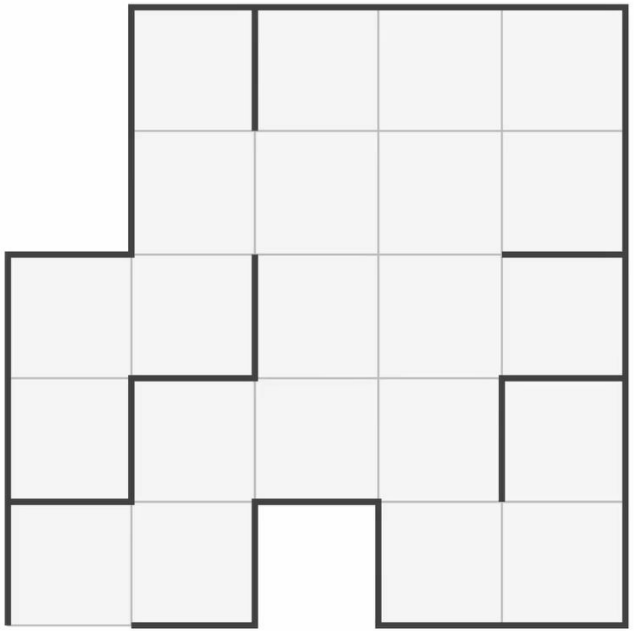
State Space Models

Quadratic Attention

- Vanila attention has quadratic complexity
- It makes Transformers hard to apply to long contexts
- Especially in the era on reasoning
- Sliding window attention (and others) is a solution, but we can do better

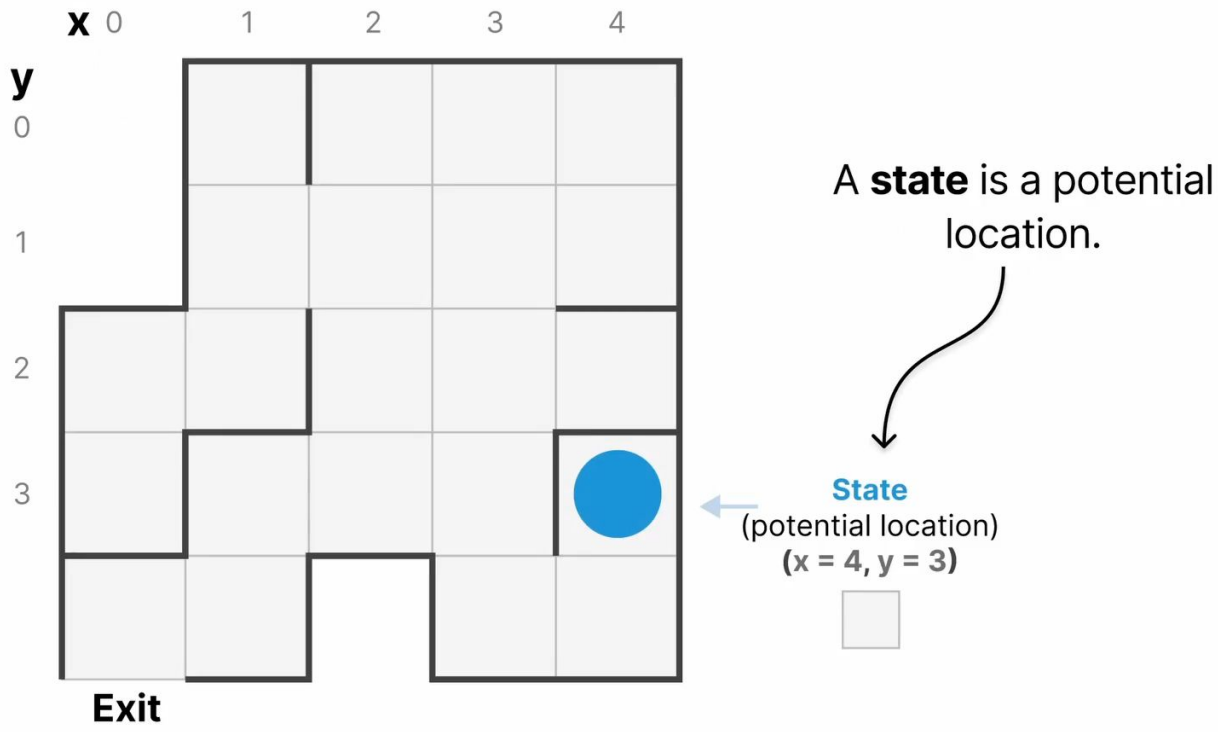


What is a State Space?

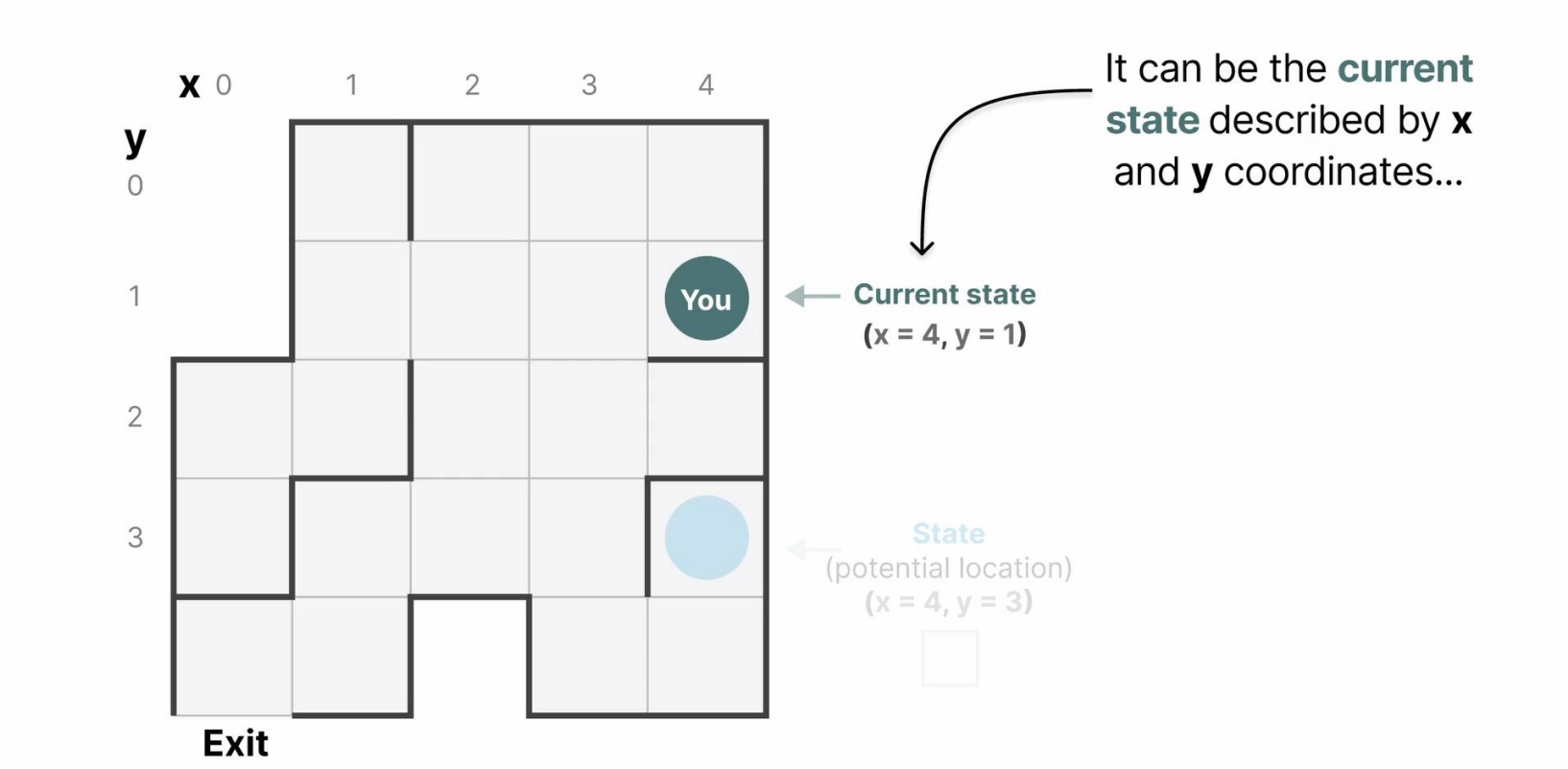


Imagine we are navigating through a maze. The "**state space**" is the map of all possible locations.

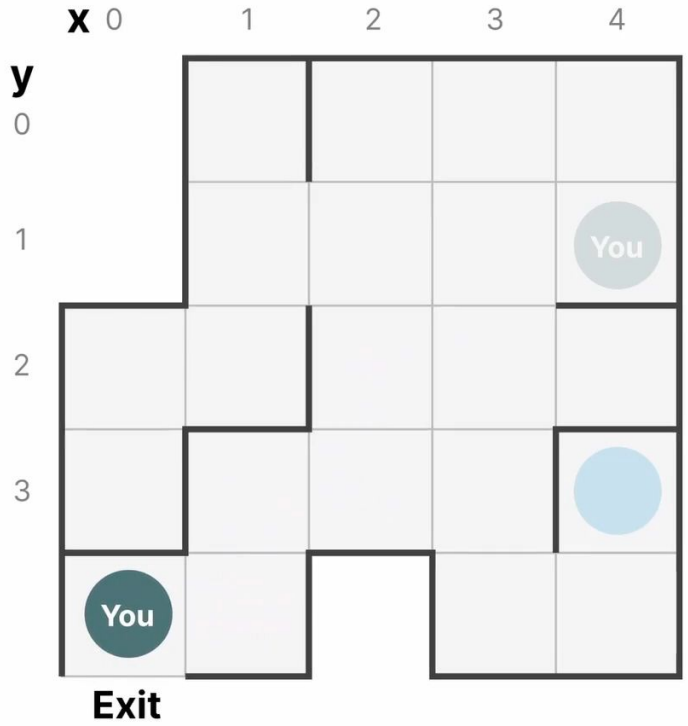
What is a State Space?



What is a State Space?



What is a State Space?

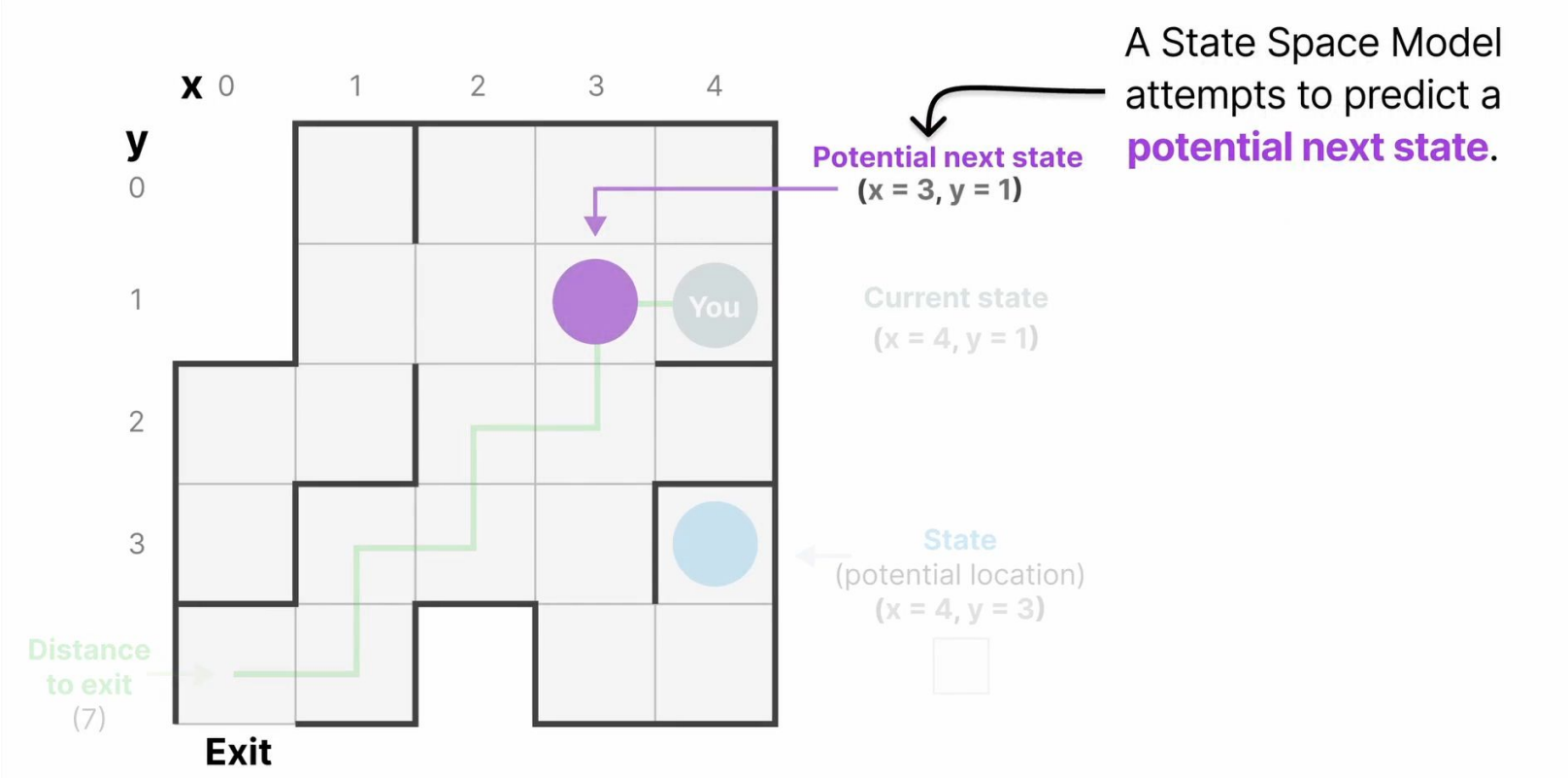


... but also contain additional information, like the **distance to the exit**.

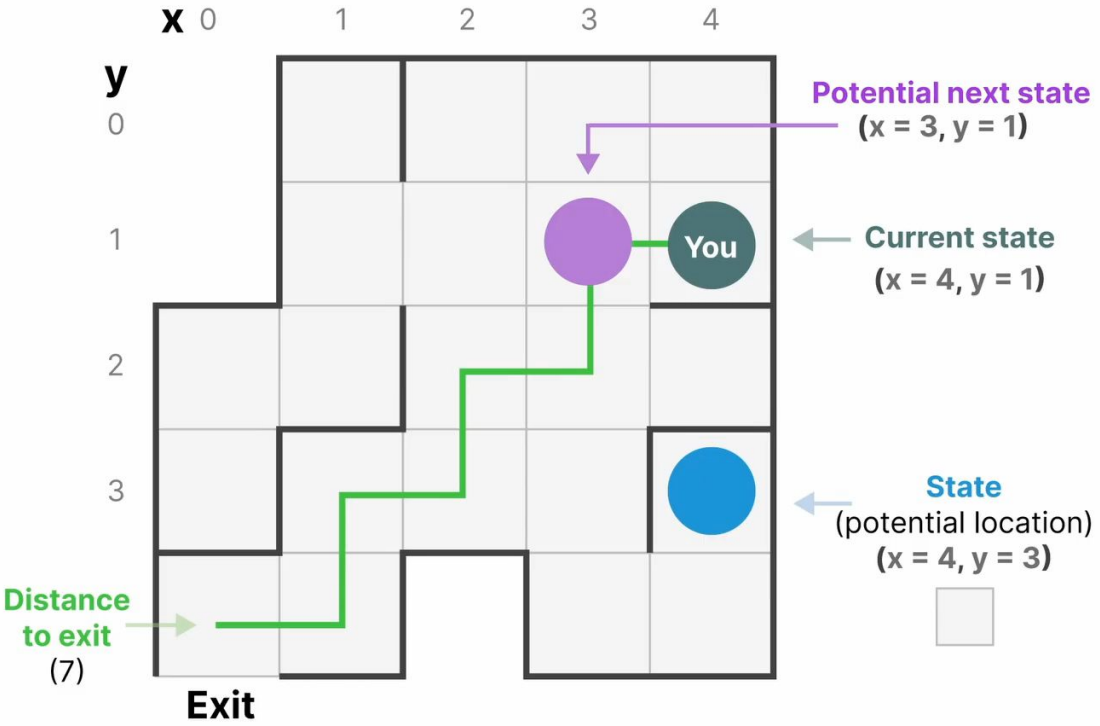
Current state
(x = 4, y = 1)

State
(potential location)
(x = 4, y = 3)

What is a State Space?

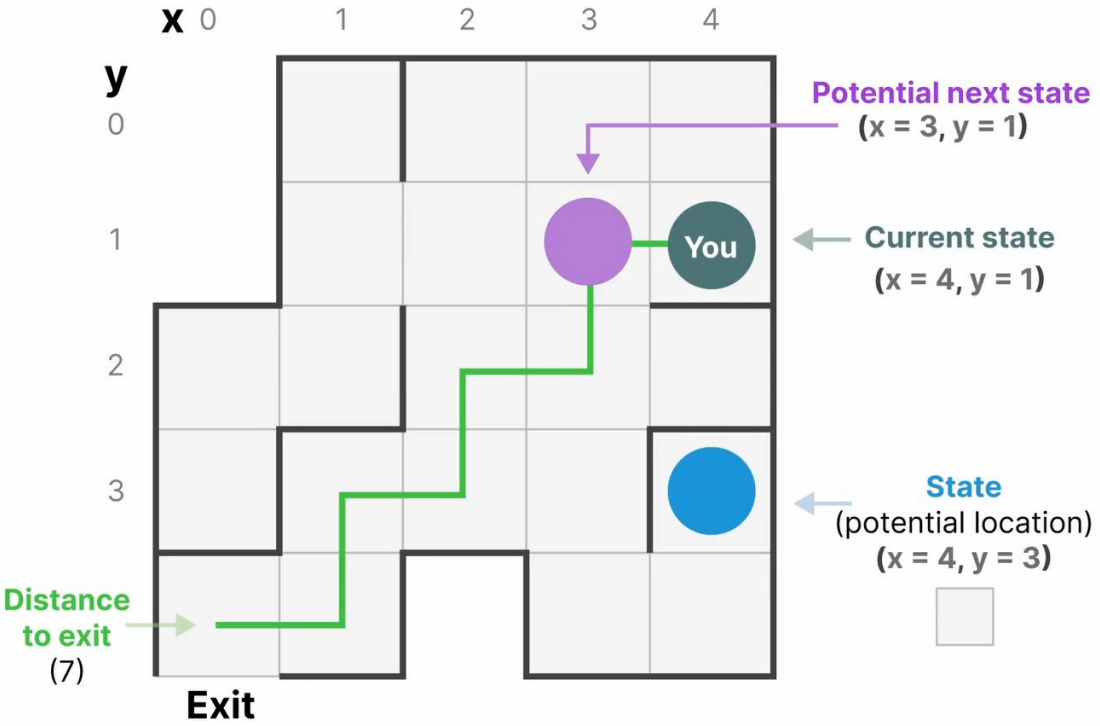


What is a State Space?



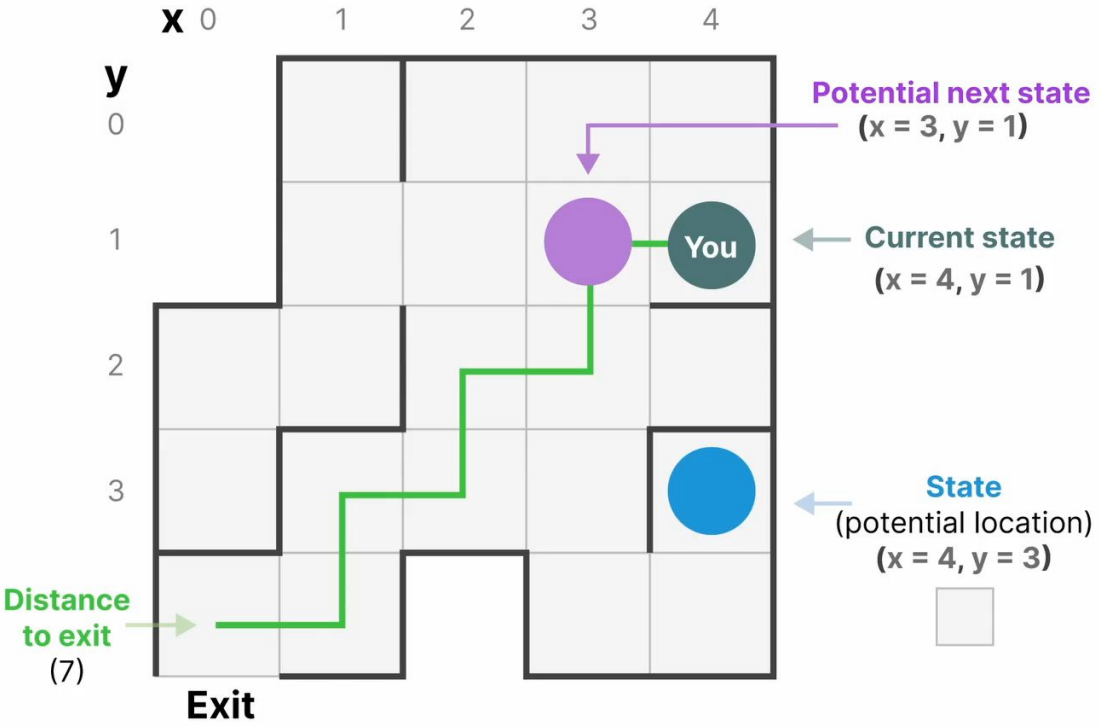
...where you can go next (possible future states)...

What is a State Space?



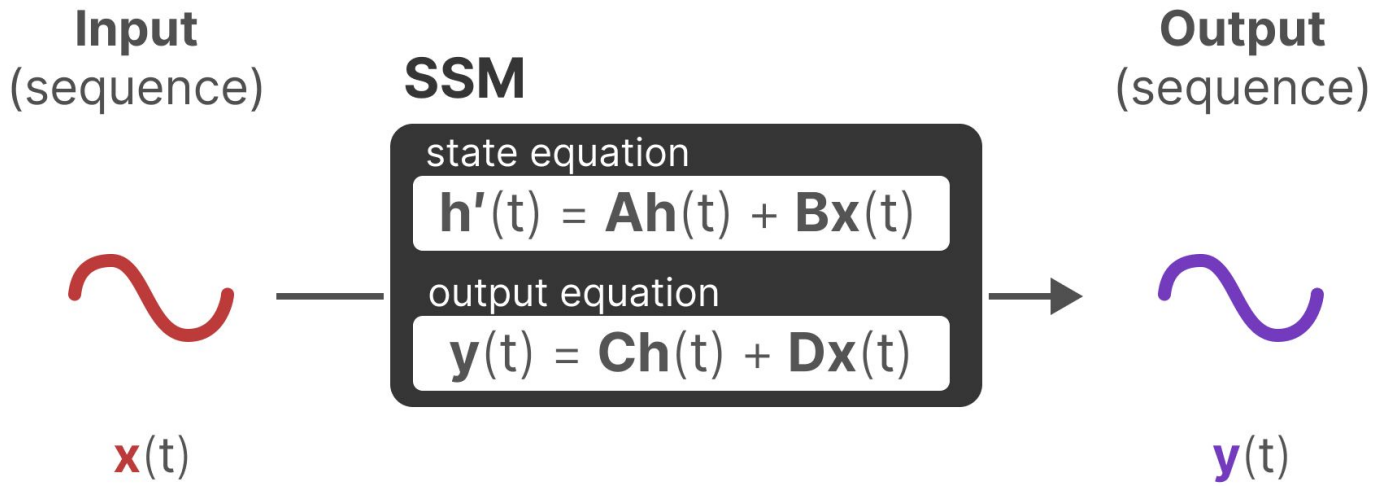
...where you can go next (possible future states)...

What is a State Space?



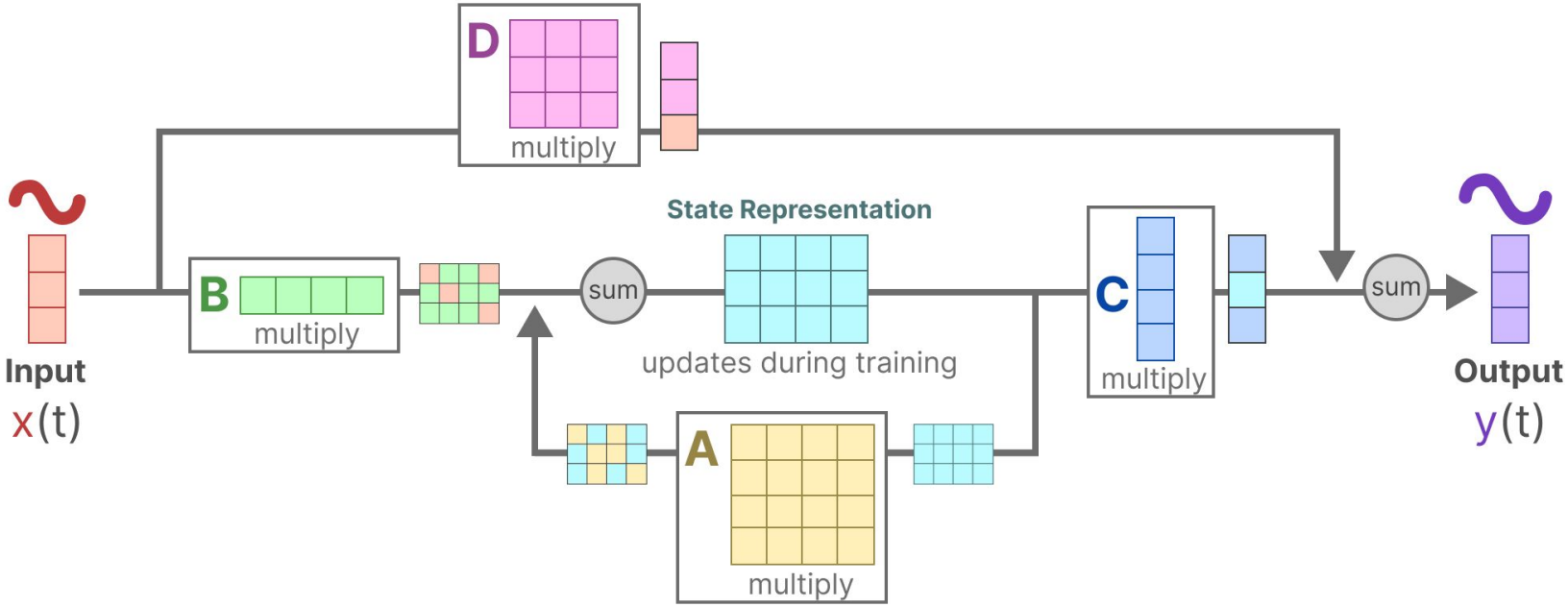
... and what changes take you to the next state (going right or left).

What is State Space Model?

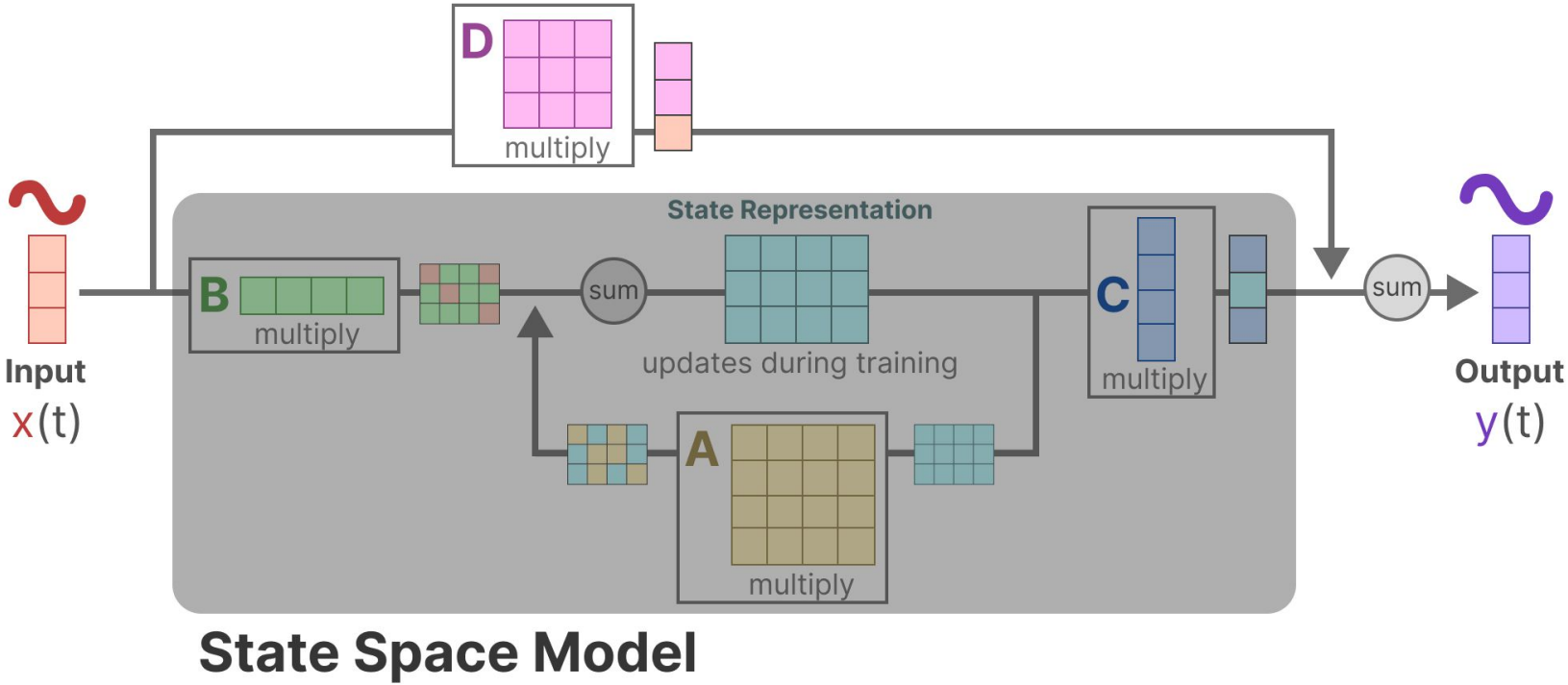


SSMs are models used to describe state representations and make predictions of what their next state could be depending on some input. However, instead of using discrete sequences it takes as input a continuous sequence and predicts the output sequence.

State Space Model

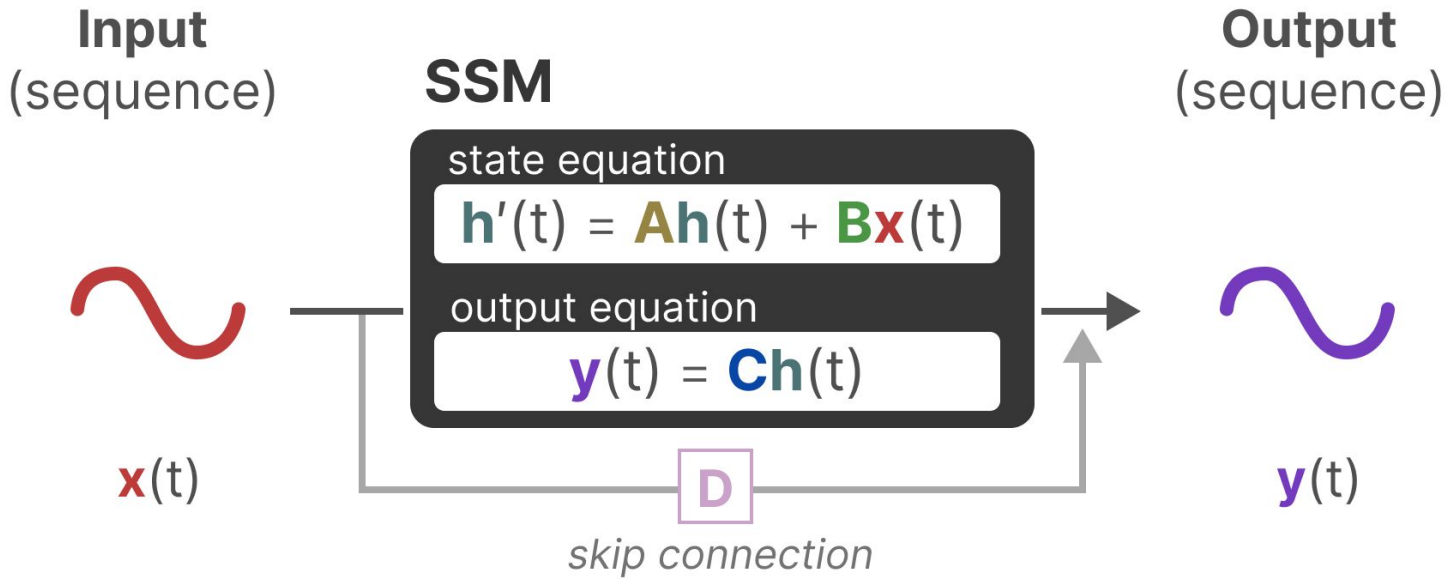


State Space Model



Since *D* is similar to a skip-connection, the SSM is often regarded as this without the skip-connection

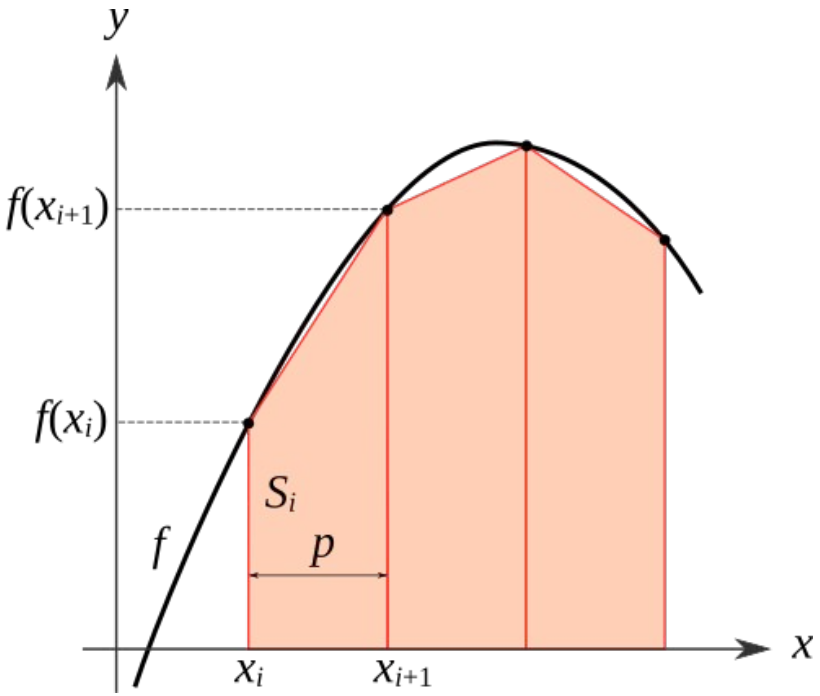
Discretization



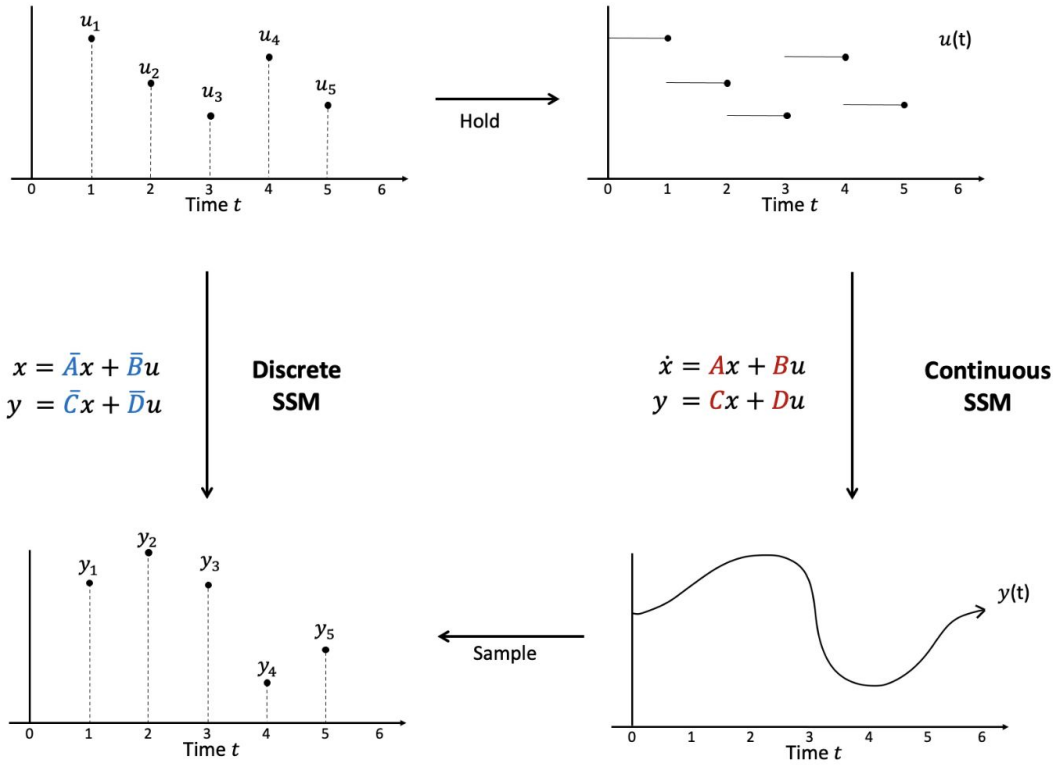
Since we generally work with discrete inputs, we want to discretize the model

Discretization

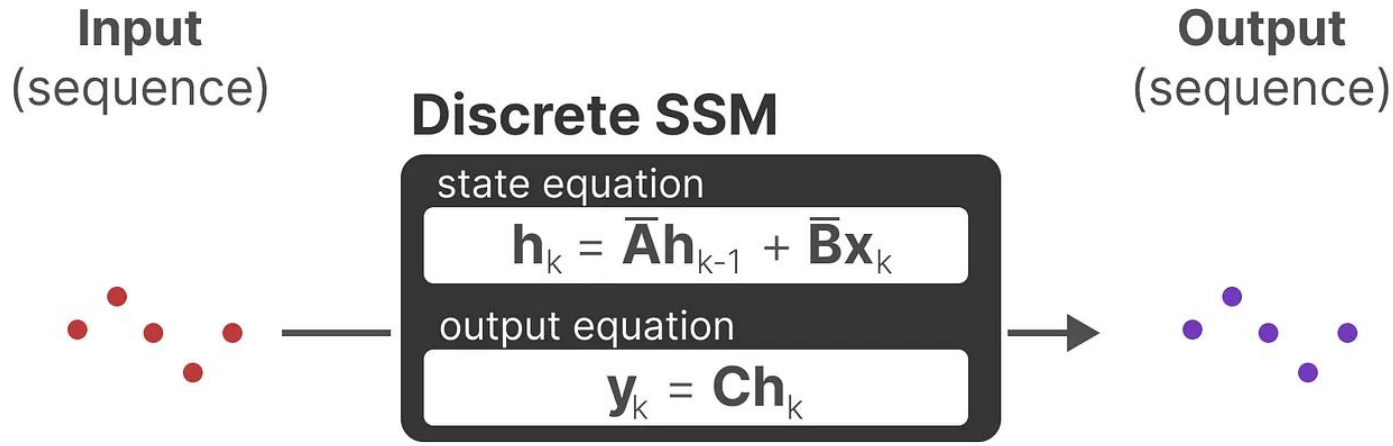
We will use the trapezoidal method to convert a continuous-time system to a discrete one, similar to how we approximate the area under a curve in integration.



Discretization



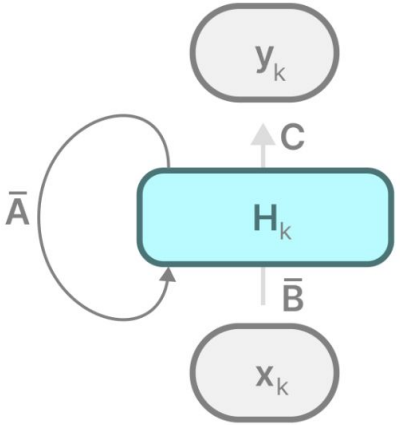
Discretization



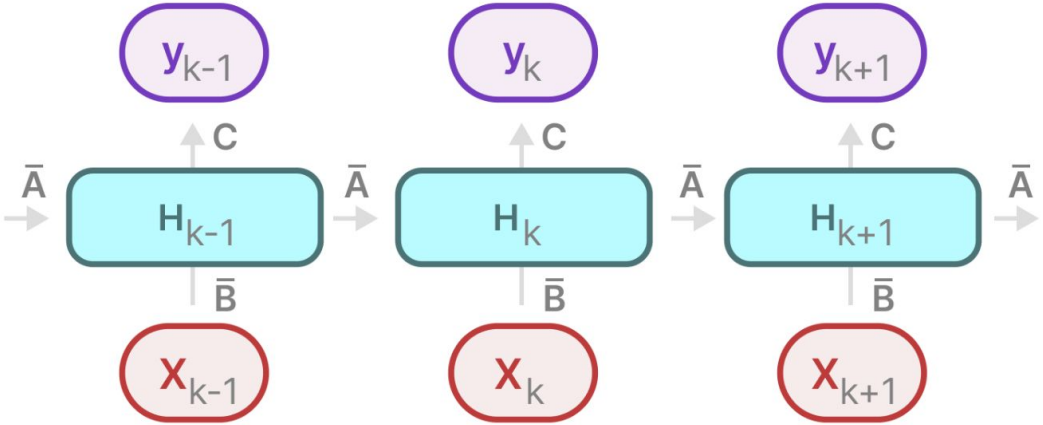
$$\overline{\boldsymbol{A}} = (\boldsymbol{I} - \Delta/2 \cdot \boldsymbol{A})^{-1}(\boldsymbol{I} + \Delta/2 \cdot \boldsymbol{A})$$

$$\overline{\boldsymbol{B}} = (\boldsymbol{I} - \Delta/2 \cdot \boldsymbol{A})^{-1}\Delta \boldsymbol{B} \qquad \overline{\boldsymbol{C}} = \boldsymbol{C}$$

Recurrent view



SSM
(Recurrent)



SSM
(Recurrent + Unfolded)

Looks familiar?

Recurrent view

SSM

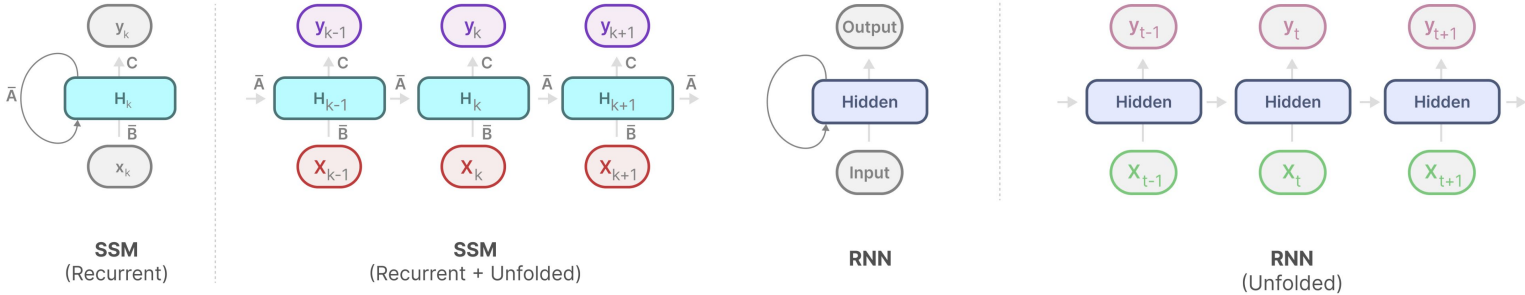
$$h_k = \overline{A} h_{k-1} + \overline{B} x_k$$

$$y_k = \overline{C} h_k$$

RNN

$$h_k = \tanh(W_{hh} h_{k-1} + W_{xh} x_k)$$

$$y_k = W_{hy} h_k$$



SSM (Recurrent)

SSM (Recurrent + Unfolded)

RNN

RNN (Unfolded)

SSM does not have non-linearity

Convolutive view

1. $h_0 = \overline{B}x_0$

   $y_0 = \overline{C}h_0 = \overline{C}\overline{B}x_0$

2. $h_1 = \overline{A}h_0 + \overline{B}x_1 = \overline{A}\overline{B}x_0 + \overline{B}x_1$

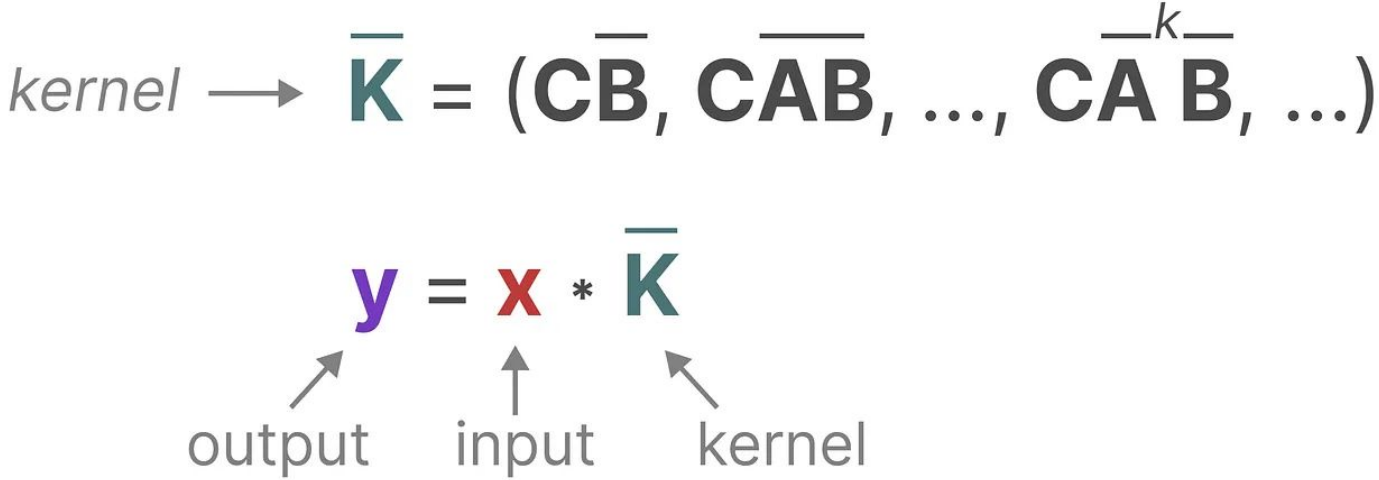
   $y_1 = \overline{C}h_1 = \overline{C}\left(\overline{A}\overline{B}x_0 + \overline{B}x_1\right) = \overline{C}\overline{A}\overline{B}x_0 + \overline{C}\overline{B}x_1$

3. $h_2 = \overline{A}h_1 + \overline{B}x_2 = \overline{A}\left(\overline{A}\overline{B}x_0 + \overline{B}x_1\right) + \overline{B}x_2 = \overline{A}^2\overline{B}x_0 + \overline{A}\overline{B}x_1 + \overline{B}x_2$

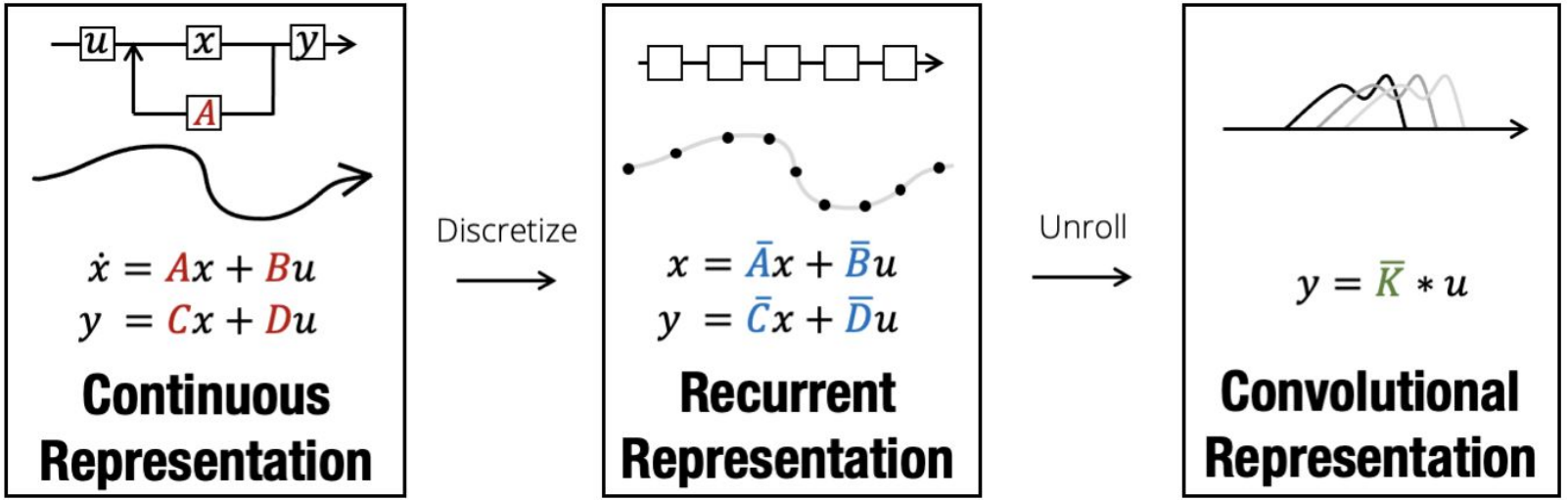
   $y_2 = \overline{C}h_2 = \overline{C}\left(\overline{A}^2\overline{B}x_0 + \overline{A}\overline{B}x_1 + \overline{B}x_2\right) = \overline{C}\overline{A}^2\overline{B}x_0 + \overline{C}\overline{A}\overline{B}x_1 + \overline{C}\overline{B}x_2$

h_k is a linear function of $(x_0, x_1, ..., x_k)$

Convolutive view



Three views of SSM



When to use each view?

Continuous view:

- ✅ Handles continuous data
- ❌ Extremely slow for both training and inference

When to use each view?

Continuous view:

- ✅ Handles continuous data
- ❌ Extremely slow for both training and inference

Recurrent view:

- ✅ Natural inductive bias for sequential data, and principle unbounded context
- ✅ Efficient inference
- ❌ Slow learning (lack of parallelism)
- ❌ Gradient vanish or explosion for long sequences

When to use each view?

Continuous view:

- ✅ Handles continuous data
- ❌ Extremely slow for both training and inference

Recurrent view:

- ✅ Natural inductive bias for sequential data, and principle unbounded context
- ✅ Efficient inference
- ❌ Slow learning (lack of parallelism)
- ❌ Gradient vanish or explosion for long sequences

Convolutive view:

- ✅ Efficient parallelizable training
- ❌ Slow in autoregressive contexts (must recalculate entire input for each new data point)
- ❌ Fixed context size

Importance of matrix A

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

- We need to compute $\bar{A}^k$ fast
- Easiest way is to make $\bar{A}^k$ diagonal (class of normal matrices)

Linear algebra recap:

- $A \in \mathbb{C}^{n\times n} - \text{normal} \Leftrightarrow A^*A = AA^*$
- $U \in \mathbb{C}^{n\times n} - \text{unitary} \Leftrightarrow U^*U = UU^* = I$
- Spectral theorem:

$$A \in \mathbb{C}^{n\times n} - \text{normal} \Leftrightarrow \exists \Lambda \in \mathbb{C}^{n\times n} - \text{diagonal}, U \in \mathbb{C}^{n\times n} - \text{unitary} : A = U\Lambda U^*$$

$$A^k = U\Lambda U^* \times U\Lambda U^* \times \ldots \times U\Lambda U^* = U\Lambda^k U^*, \text{where } \Lambda^k = \begin{pmatrix} \lambda_1^k & 0 & \ldots & 0 \\ 0 & \lambda_2^k & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_n^k \end{pmatrix}$$

Importance of matrix A

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

- We need to compute $\bar{A}^k$ fast
- Easiest way is to make $\bar{A}^k$ diagonal (class of normal matrices)

Linear algebra recap:

- $A \in \mathbb{C}^{n\times n} - \text{normal} \Leftrightarrow A^*A = AA^*$
- $U \in \mathbb{C}^{n\times n} - \text{unitary} \Leftrightarrow U^*U = UU^* = I$
- Spectral theorem:

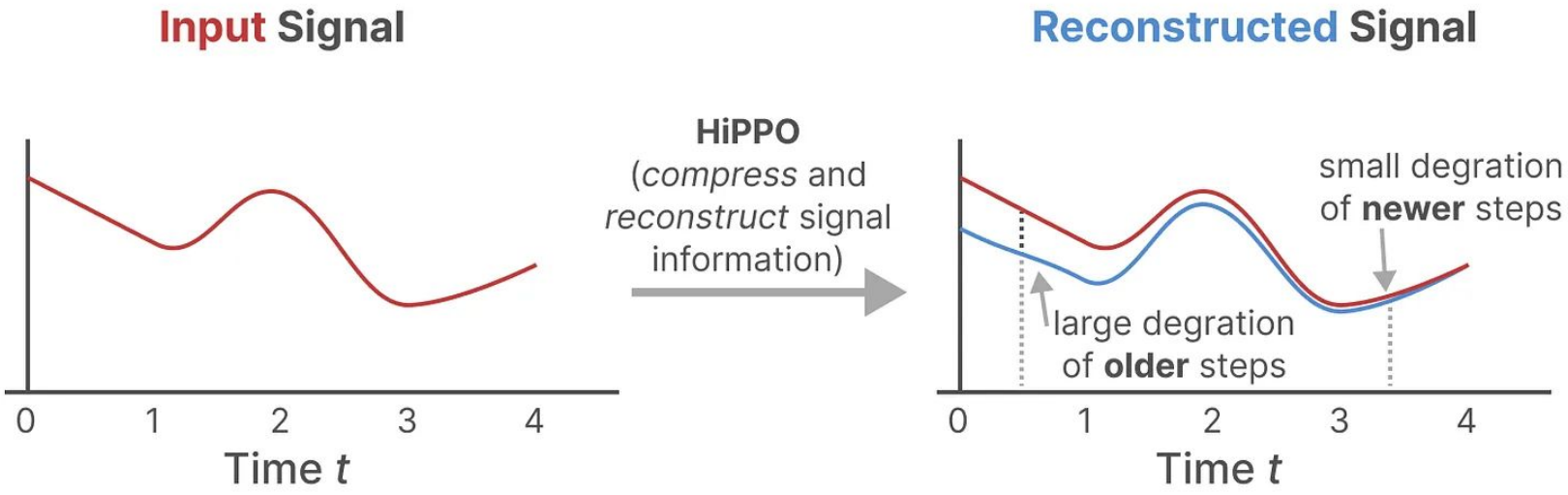
$$A \in \mathbb{C}^{n\times n} - \text{normal} \Leftrightarrow \exists \Lambda \in \mathbb{C}^{n\times n} - \text{diagonal}, U \in \mathbb{C}^{n\times n} - \text{unitary} : A = U\Lambda U^*$$

$$A^k = U\Lambda U^* \times U\Lambda U^* \times \ldots \times U\Lambda U^* = U\Lambda^k U^*, \text{where } \Lambda^k = \begin{pmatrix} \lambda_1^k & 0 & \ldots & 0 \\ 0 & \lambda_2^k & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_n^k \end{pmatrix}$$

- Matrix A produces the hidden state, so we want it to preserve a large context
- So we can’t just use an arbitrary normal matrix, initialization matters a lot here

HiPPO

High-order Polynomial Projection Operator



It uses matrix A to build a state representation that captures recent tokens well and decays older tokens

HiPPO

High-order Polynomial Projection Operator

HiPPO Matrix A_{nk}

$$\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{everything below the diagonal} \\ n+1 & \text{the diagonal} \\ 0 & \text{everything above the diagonal} \end{cases}$$

Note: there exist many modifications of the HiPPO matrix, this is the classical version

HiPPO Matrix

1	0	0	0
1	2	0	0
1	3	3	0
1	3	5	4

k (rows) × n (columns)

HiPPO

High-order Polynomial Projection Operator

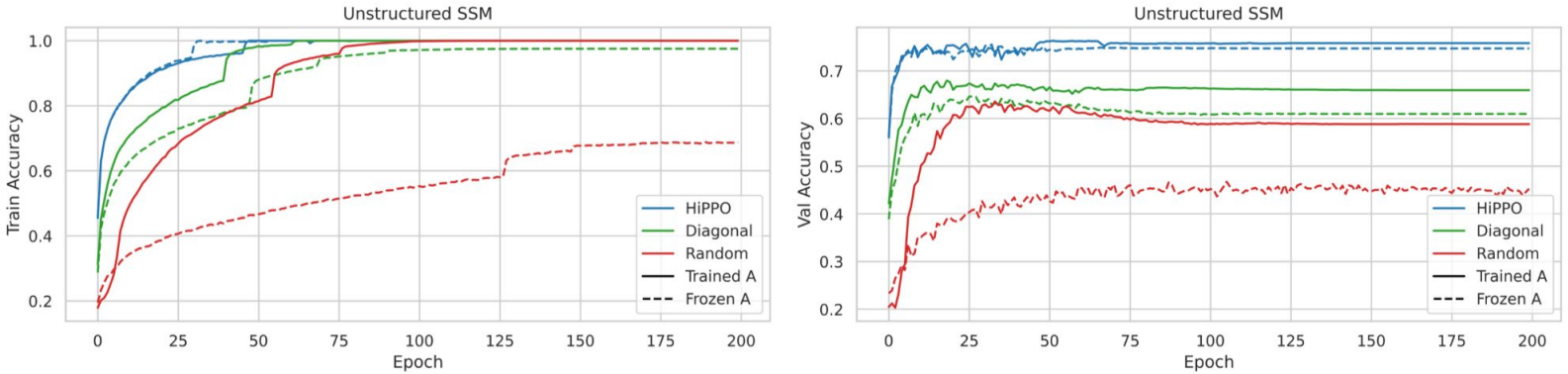


Figure 3: CIFAR-10 classification with unconstrained, real-valued SSMs with various initializations. (*Left*) Train accuracy. (*Right*) Validation accuracy.

HiPPO

High-order Polynomial Projection Operator

- An attentive listener may notice that, although the HiPPO matrix suits our needs, it is not a normal matrix

HiPPO

High-order Polynomial Projection Operator

- An attentive listener may notice that, although the HiPPO matrix suits our needs, it is not a normal matrix
- So we will use NPLR (Normal Plus Low Rank)

Theorem 1. *All HiPPO matrices from [16] have a NPLR representation*

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^* - \boldsymbol{P}\boldsymbol{Q}^\top = \boldsymbol{V}\left(\boldsymbol{\Lambda} - \left(\boldsymbol{V}^*\boldsymbol{P}\right)\left(\boldsymbol{V}^*\boldsymbol{Q}\right)^*\right)\boldsymbol{V}^* \tag{6}$$

for unitary $\boldsymbol{V} \in \mathbb{C}^{N\times N}$*, diagonal* $\boldsymbol{\Lambda}$*, and low-rank factorization* $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{R}^{N\times r}$*. These matrices HiPPO- LegS, LegT, LagT all satisfy* $r = 1$ *or* $r = 2$*. In particular, equation* (2) *is NPLR with* $r = 1$*.*

HiPPO

High-order Polynomial Projection Operator

- An attentive listener may notice that, although the HiPPO matrix suits our needs, it is not a normal matrix
- So we will use NPLR (Normal Plus Low Rank)

Theorem 1. *All HiPPO matrices from [16] have a NPLR representation*

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^* - \boldsymbol{P}\boldsymbol{Q}^\top = \boldsymbol{V}\left(\boldsymbol{\Lambda} - \left(\boldsymbol{V}^*\boldsymbol{P}\right)\left(\boldsymbol{V}^*\boldsymbol{Q}\right)^*\right)\boldsymbol{V}^* \tag{6}$$

for unitary $\boldsymbol{V} \in \mathbb{C}^{N\times N}$*, diagonal* $\boldsymbol{\Lambda}$*, and low-rank factorization* $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{R}^{N\times r}$*. These matrices HiPPO- LegS, LegT, LagT all satisfy* $r = 1$ *or* $r = 2$*. In particular, equation* (2) *is NPLR with* $r = 1$*.*

- But now we have a sum of two matrices: one is normal and the other is low-rank, so what do we do? This is where the following algorithm comes in, which we will not discuss in detail, but we'll take a look at it on the next slide.

S4

Algorithm 1 S4 Convolution Kernel (Sketch)

Input: S4 parameters $\boldsymbol{\Lambda}, \boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{C}^N$ and step size Δ
Output: SSM convolution kernel $\overline{\boldsymbol{K}} = \mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})$ for $\boldsymbol{A} = \boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*$ (equation (5))

1: $\widetilde{\boldsymbol{C}} \leftarrow \left(\boldsymbol{I} - \overline{\boldsymbol{A}}^L\right)^* \overline{\boldsymbol{C}}$ ▷ Truncate SSM generating function (SSMGF) to length L

2: $\begin{bmatrix} k_{00}(\omega) & k_{01}(\omega) \\ k_{10}(\omega) & k_{11}(\omega) \end{bmatrix} \leftarrow \left[\widetilde{\boldsymbol{C}}\ \boldsymbol{Q}\right]^* \left(\frac{2}{\Delta}\frac{1-\omega}{1+\omega} - \boldsymbol{\Lambda}\right)^{-1} [\boldsymbol{B}\ \boldsymbol{P}]$ ▷ Black-box Cauchy kernel

3: $\hat{\boldsymbol{K}}(\omega) \leftarrow \frac{2}{1+\omega}\left[k_{00}(\omega) - k_{01}(\omega)(1 + k_{11}(\omega))^{-1}k_{10}(\omega)\right]$ ▷ Woodbury Identity

4: $\hat{\boldsymbol{K}} = \{\hat{\boldsymbol{K}}(\omega) : \omega = \exp(2\pi i \frac{k}{L})\}$ ▷ Evaluate SSMGF at all roots of unity $\omega \in \Omega_L$

5: $\overline{\boldsymbol{K}} \leftarrow \mathsf{iFFT}(\hat{\boldsymbol{K}})$ ▷ Inverse Fourier Transform

- Instead of computing $\overline{\boldsymbol{K}}$ directly, we compute its spectrum by evaluating its **truncated generating function** $\sum_{j=0}^{L-1} \overline{\boldsymbol{K}}_j \zeta^j$ at the roots of unity ζ. $\overline{\boldsymbol{K}}$ can then be found by applying an inverse FFT.
- This generating function is closely related to the matrix resolvent, and now involves a matrix *inverse* instead of *power*. The low-rank term can now be corrected by applying the **Woodbury identity** which reduces $(\boldsymbol{A} + \boldsymbol{P}\boldsymbol{Q}^*)^{-1}$ in terms of $\boldsymbol{A}^{-1}$, truly reducing to the diagonal case.
- Finally, we show that the diagonal matrix case is equivalent to the computation of a **Cauchy kernel** $\frac{1}{\omega_j - \zeta_k}$, a well-studied problem with stable near-linear algorithms [30, 31].

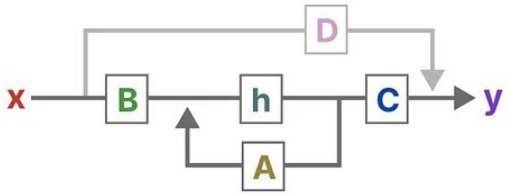
Our techniques apply to any matrix that can be decomposed as ***Normal Plus Low-Rank (NPLR)***.

S4

Structured State Spaces for Sequences (S4)

=

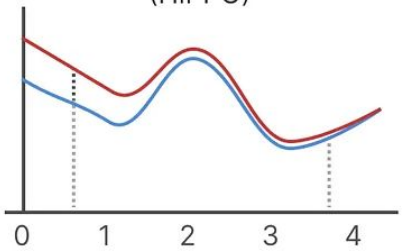
Continuous State Space



+

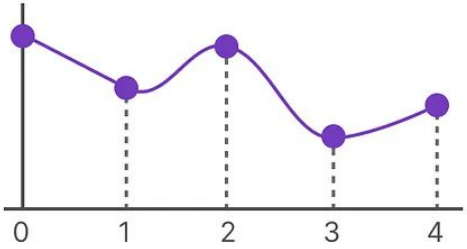
Long-Range Dependencies

(HiPPO)



+

Discrete Representations



Training mode (convolutional)

Inference mode (recurrence)

HiPPO

High-order Polynomial Projection Operator

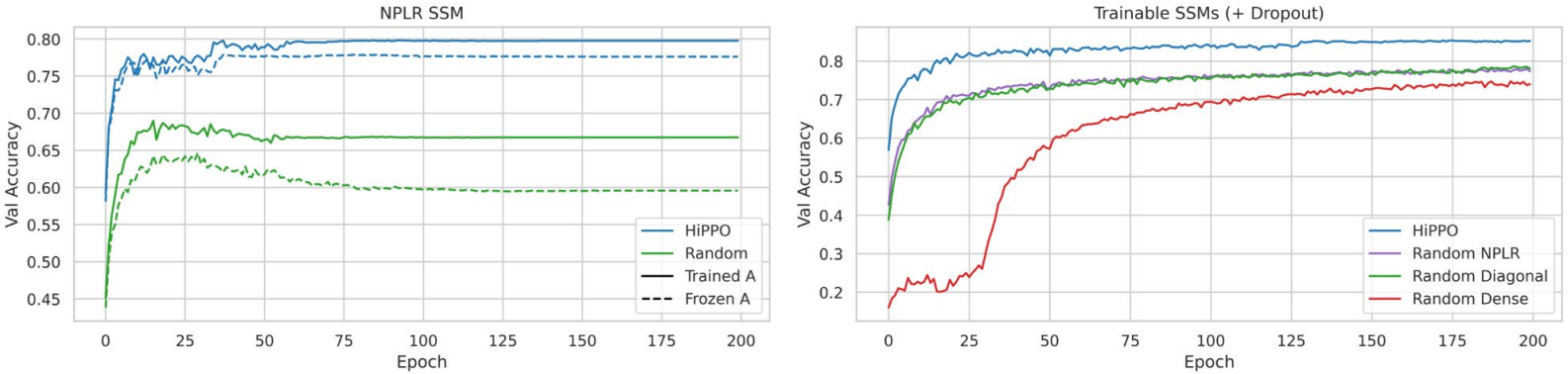


Figure 4: CIFAR-10 validation accuracy of SSMs with different initializations and parameterizations. (*Left*) NPLR parameterization with random versus HiPPO initialization. (*Right*) All methods considered in this section, including minor Dropout regularization. S4 achieves SotA accuracy on sequential CIFAR-10 with just 100K parameters.

S4

Table 1: Complexity of various sequence models in terms of sequence length ($\boldsymbol{L}$), batch size ($\boldsymbol{B}$), and hidden dimension ($\boldsymbol{H}$); tildes denote log factors. Metrics are parameter count, training computation, training space requirement, training parallelizability, and inference computation (for 1 sample and time-step). For simplicity, the state size N of S4 is tied to H. Bold denotes model is theoretically best for that metric. Convolutions are efficient for training while recurrence is efficient for inference, while SSMs combine the strengths of both.

	Convolution[3]	Recurrence	Attention	S4
Parameters	LH	$\boldsymbol{H^2}$	$\boldsymbol{H^2}$	$\boldsymbol{H^2}$
Training	$\boldsymbol{\tilde{L}H(B+H)}$	BLH^2	$B(L^2H+LH^2)$	$\boldsymbol{BH(\tilde{H}+\tilde{L})+B\tilde{L}H}$
Space	$\boldsymbol{BLH}$	$\boldsymbol{BLH}$	$B(L^2+HL)$	$\boldsymbol{BLH}$
Parallel	**Yes**	No	**Yes**	**Yes**
Inference	LH^2	$\boldsymbol{H^2}$	L^2H+H^2L	$\boldsymbol{H^2}$

S4

Table 4: (**Long Range Arena**) (*Top*) Original Transformer variants in LRA. Full results in Appendix D.2. (*Bottom*) Other models reported in the literature. *Please read Appendix D.5 before citing this table.*

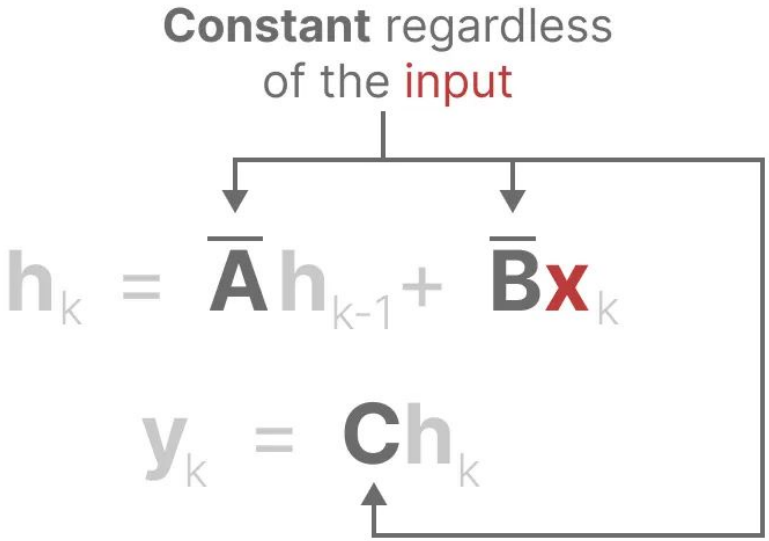
Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Reformer	37.27	56.10	53.40	38.07	68.50	✗	50.56
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	65.90	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	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	**59.60**	**86.82**	**90.90**	**88.65**	**94.20**	**96.35**	**86.09**

S4

Table 12: (**Pixel-level image classification.**) Citations refer to the original model; additional citation indicates work from which this baseline is reported.

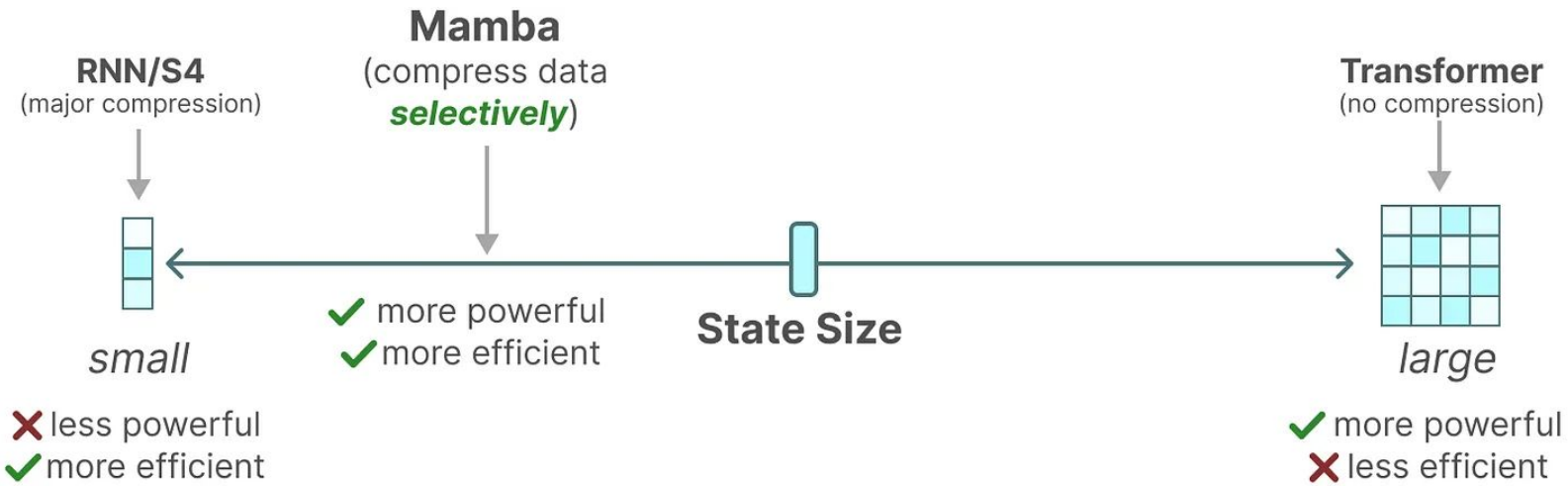
Model	sMNIST	pMNIST	sCIFAR
Transformer [42, 44]	98.9	97.9	62.2
CKConv [35]	99.32	98.54	63.74
TrellisNet [4]	99.20	98.13	73.42
TCN [3]	99.0	97.2	-
LSTM [17, 21]	98.9	95.11	63.01
r-LSTM [42]	98.4	95.2	72.2
Dilated GRU [5]	99.0	94.6	-
Dilated RNN [5]	98.0	96.1	-
IndRNN [25]	99.0	96.0	-
expRNN [24]	98.7	96.6	-
UR-LSTM	99.28	96.96	71.00
UR-GRU [17]	99.27	96.51	74.4
LMU [45]	-	97.15	-
HiPPO-RNN [16]	98.9	98.3	61.1
UNIcoRNN [38]	-	98.4	-
LMUFFT [7]	-	98.49	-
LipschitzRNN [13]	99.4	96.3	64.2
S4	**99.63**	**98.70**	**91.13**

Selective SSM

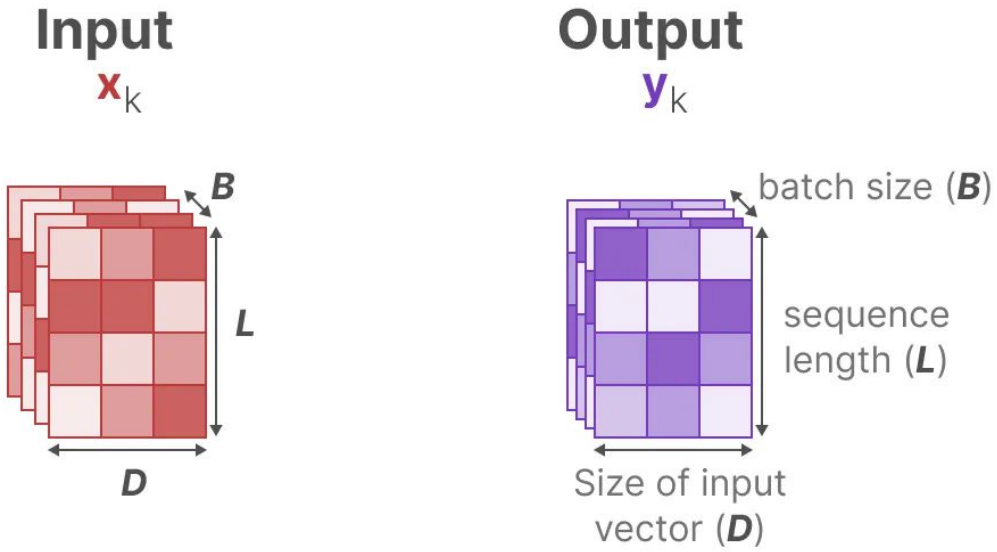


SSMs struggle with content-aware reasoning because they treat every token equally, using fixed A, B, and C matrices

Selective SSM



Selective SSM

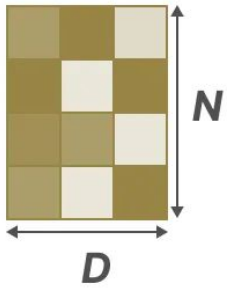


Selective SSM

Structured State Space Model (S4)

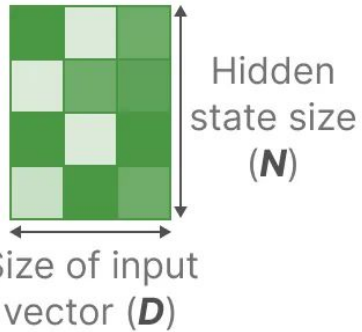
Matrix A

How the **current state** evolves over time



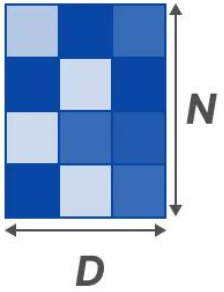
Matrix B

How the **input** influences the state



Matrix C

How the **current state** translates to the **output**



Selective SSM

Step size (Δ)

Resolution of the **input**
(discretization parameter)

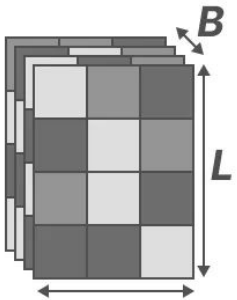
Matrix B

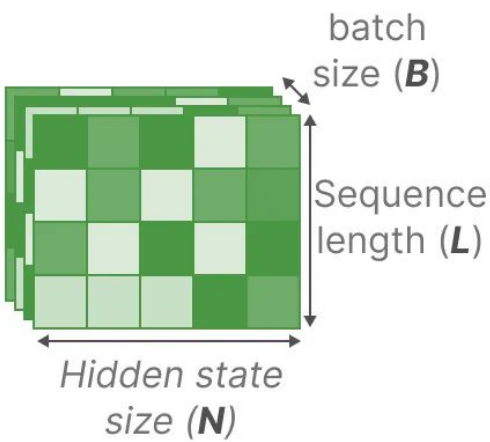
How the **input**
influences the state

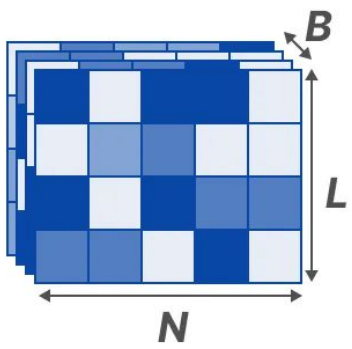
Matrix C

How the **current state**
translates to the **output**

SSM + Selection







Selective SSM

Step size (Δ)

Resolution of the **input**
(discretization parameter)

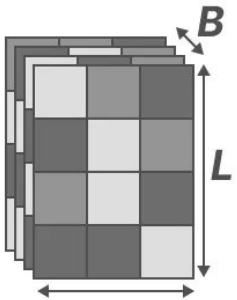
Matrix B

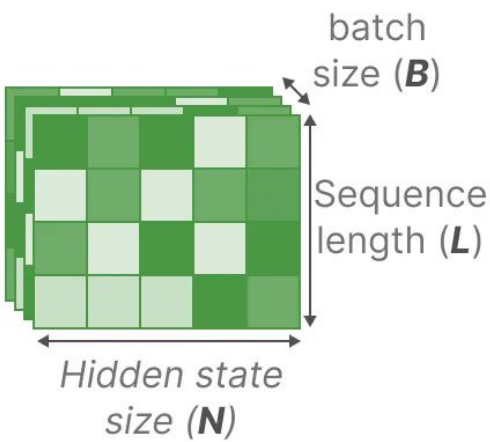
How the **input**
influences the state

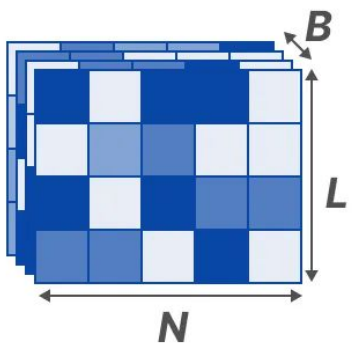
Matrix C

How the **current state**
translates to the **output**

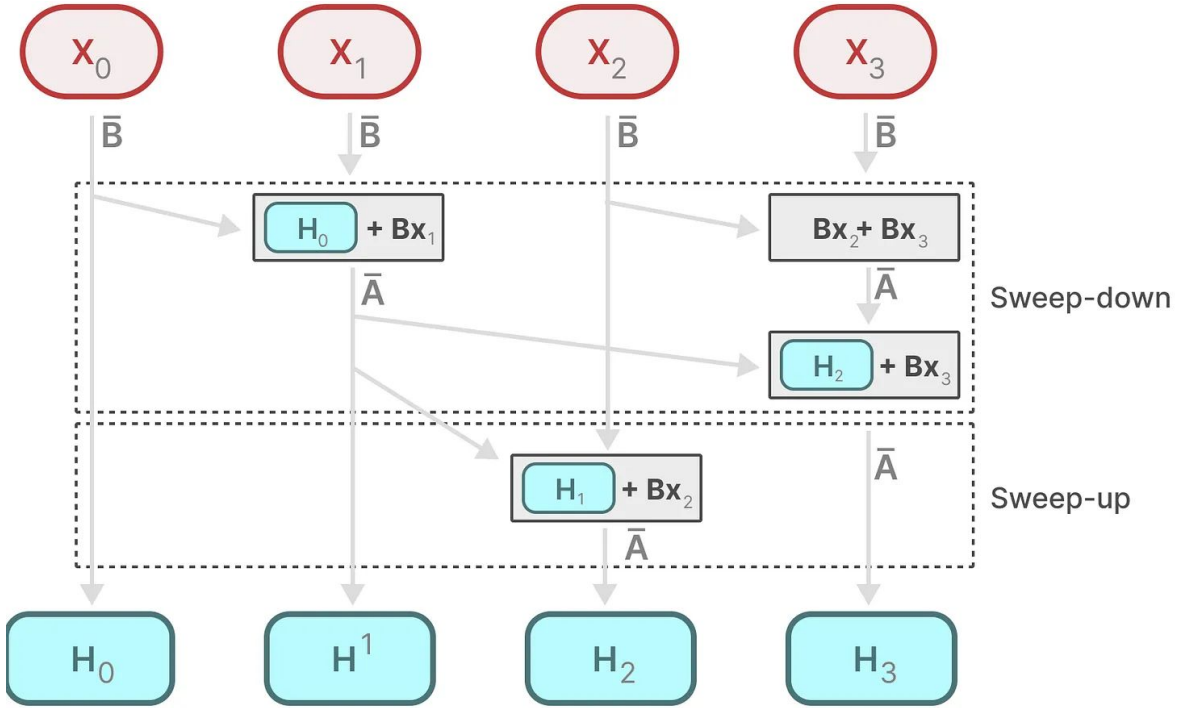
SSM + Selection







Parallel prefix sum (scan)



Parallel computation **O(n/t)**

Selective SSM

Algorithm 1 SSM (S4)

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

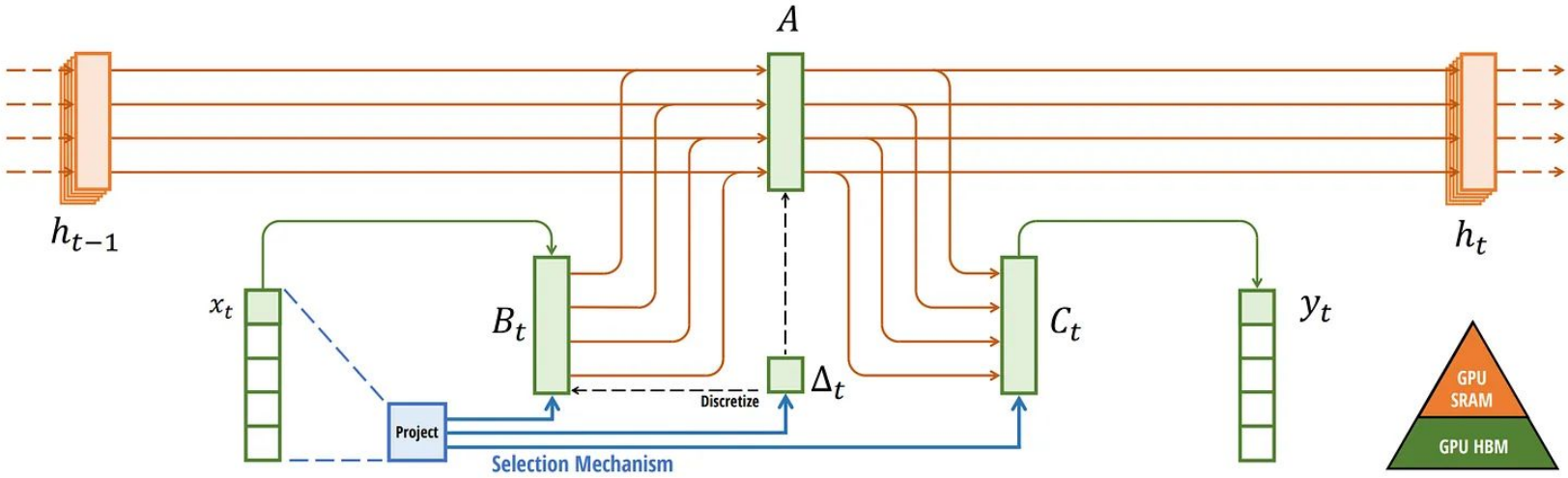
$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
$\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
$\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter})$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

Algorithm 2 SSM + Selection (S6)

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

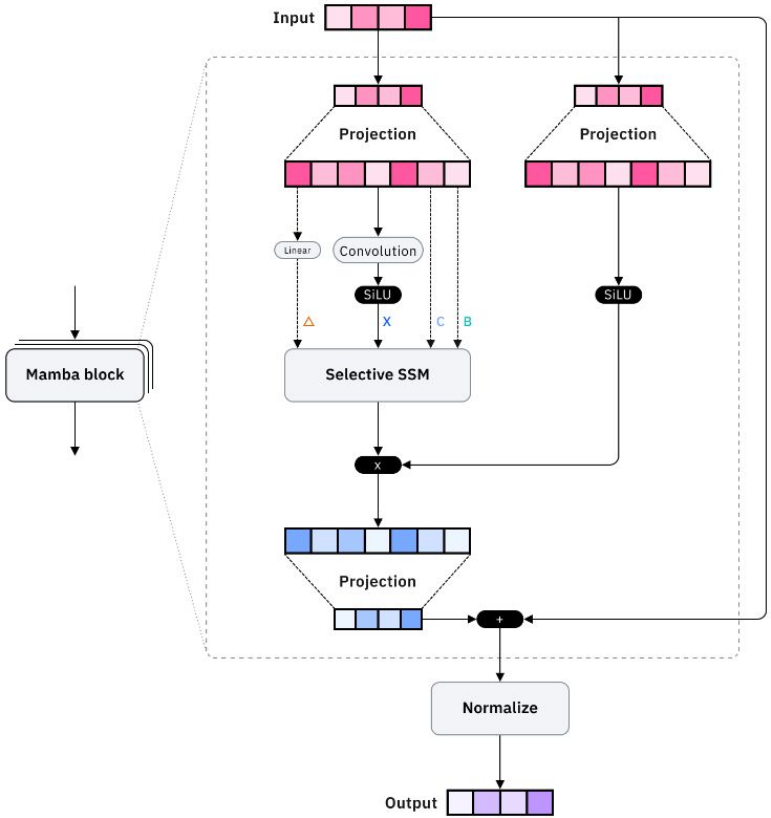
$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
$\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D})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

Selective SSM



Mamba

Selective State Space model



Mamba

Selective State Space model

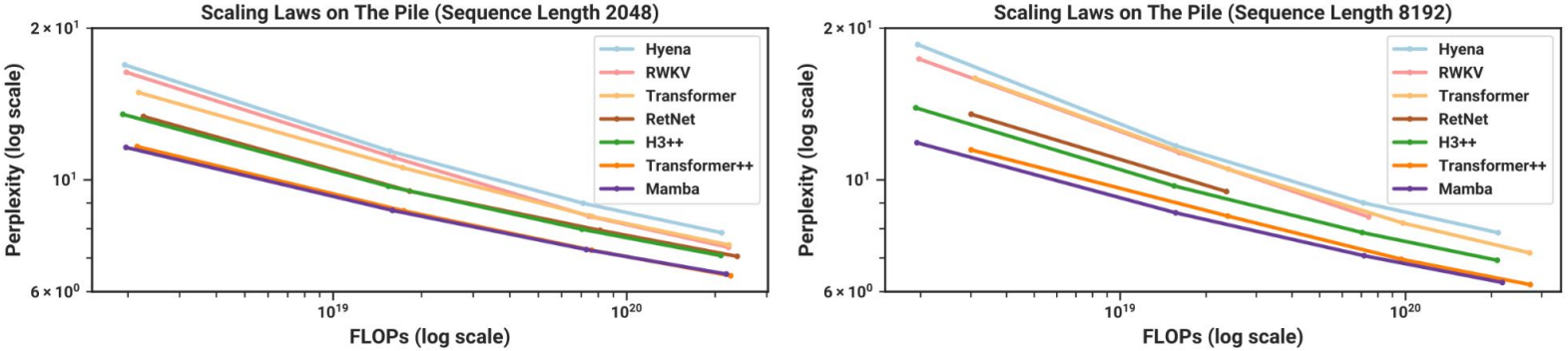


Figure 4: (**Scaling Laws.**) Models of size $\approx 125M$ to $\approx 1.3B$ parameters, trained on the Pile. Mamba scales better than all other attention-free models and is the first to match the performance of a very strong "Transformer++" recipe that has now become standard, particularly as the sequence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

Mamba

Selective State Space model

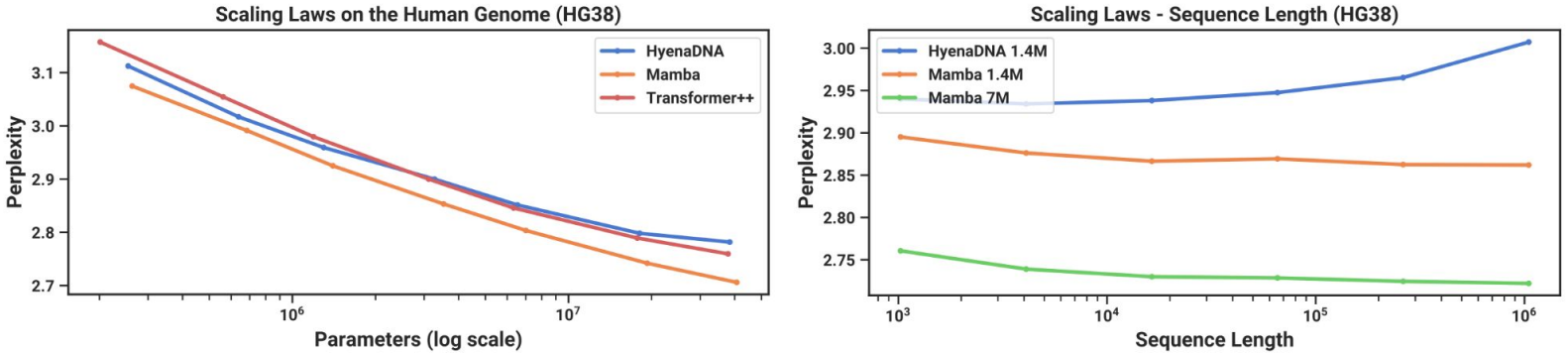


Figure 5: (**DNA Scaling Laws.**) Pretraining on the HG38 (human genome) dataset. (*Left*) Fixing short context length $2^{10} = 1024$ and increasing size from $\approx 200K$ to $\approx 40M$ parameters, Mamba scales better than baselines. (*Right*) Fixing model size and increasing sequence lengths while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

Mamba

Selective State Space model

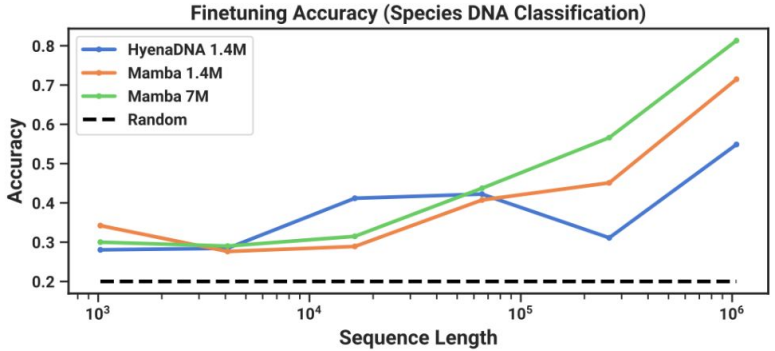


Figure 6: (**Great Apes DNA Classification.**) Accuracy after fine-tuning on sequences of length $2^{10} = 1024$ up to $2^{20} = 1048576$ using pretrained models of the same context length. Numerical results in Table 13.

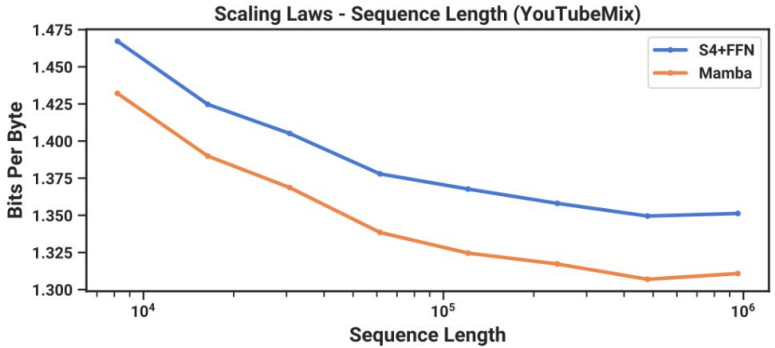


Figure 7: (**Audio Pretraining.**) Mamba improves performance over prior state-of-the-art (Sashimi) in autoregressive audio modeling, while improving up to minute-long context or million-length sequences (controlling for computation).

References and materials

- HiPPO: Recurrent Memory with Optimal Polynomial Projections (2020)
- Efficiently Modeling Long Sequences with Structured State Spaces (2021)
- Hungry Hungry Hippos: Towards Language Modeling with State Space Models (2022)
- On the Parameterization and Initialization of Diagonal State Space Models (2022)
- Mamba: Linear-Time Sequence Modeling with Selective State Spaces (2023)
- Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality (2024)
- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state
- https://srush.github.io/annotated-s4/
- https://huggingface.co/blog/lbourdois/get-on-the-ssm-train
- https://www.ibm.com/think/topics/mamba-model
- https://youtu.be/luCBXCErkCs?si=oS9g_dWR7GolS-w-
- https://youtu.be/yceNl9C6Ir0?si=MfTPDB3NkrsBn3eD