Back to RNN?

transformer - linear transformer - SSM - linear RNN - Mamba - Griffin & Hawk

2017.7 – 2020.7 – 2021.10 –2023.3 – 2023.12 – 2024.3

Complexity of Transformer

notation: sequence length N; hidden dimension d

- Self-Attention

1. $x \to Q, K, V$
   xW_Q:$[N, d] \times [d, d] \to [N, d]$
   computation $= 3 * Nd^2$
2. QK^T
   : $[N, d] \times [d, N] \to [N, N]$
   computation $= N^2 d$
3. $score \cdot V$
   : $[N, N] \times [N, d] \to [N, d]$
   computation $= N^2 d$
4. $softmax\left(\frac{QK^T}{\sqrt{d}}\right) \cdot V \cdot W_o$
   : $[N, d] \times [d, d] \to [N, d]$
   computation $= Nd^2$

total: $4Nd^2 + 2N^2d$

- FFN

computation of $A \in \mathbb{R}^{a \times b} \times B \in \mathbb{R}^{b \times c}$ is abc

1. $x_{out}W_1$
   : $[N, d] \times [d, 4d] \to [N, 4d]$
   computation $= 4Nd^2$
2. $\sigma(x_{out}W_1)W_2$
   : $[N, 4d] \times [4d, d] \to [N, d]$
   computation $= 4Nd^2$

total: $8Nd^2$

SA + FFN: $12Nd^2 + 2N^2d$

when $\boldsymbol{N > 6d}$ (in GPT2-small, d=768), N^2 term makes more contribution.

Intuition: only when N increases to a certain amount will N^2 complexity truly matter.

Linear Transformer

notation: sequence length N; hidden dimension d

- Self-Attention (original)

1. $x \to Q, K, V$
   xW_Q:$[N, d] \times [d, d] \to [N, d]$
2. QK^T
   : $[N, d] \times [d, N] \to [N, N]$
   computation $= N^2 d$
3. $score \cdot V$
   : $[N, N] \times [N, d] \to [N, d]$
   computation $= N^2 d$

…

step2&3: $2N^2 d$

- Self-Attention (linear)

1. $x \to Q, K, V$
   xW_Q:$[N, d] \times [d, d] \to [N, d]$
2. $K^T V$
   : $[d, N] \times [N, d] \to [d, d]$
   computation $= d^2 N$
3. $Q(K^T V)$
   : $[N, d] \times [d, d] \to [N, d]$
   computation $= d^2 N$

…

step2&3: $2Nd^2$

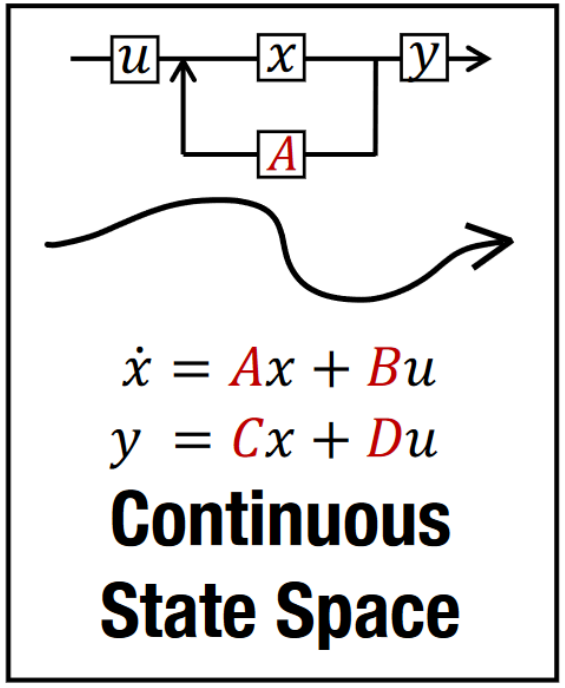
- Due to low rank bottleneck, normally $d_{linear} \sim 4 d_{original}$.
- original:
  $2N^2 d_{original}$
- linear:
  $2N\left(4d_{original}\right)^2 \sim 32 N d_{original}$
- When $\boldsymbol{N > 16 d_{original}}$, linear attention may work.

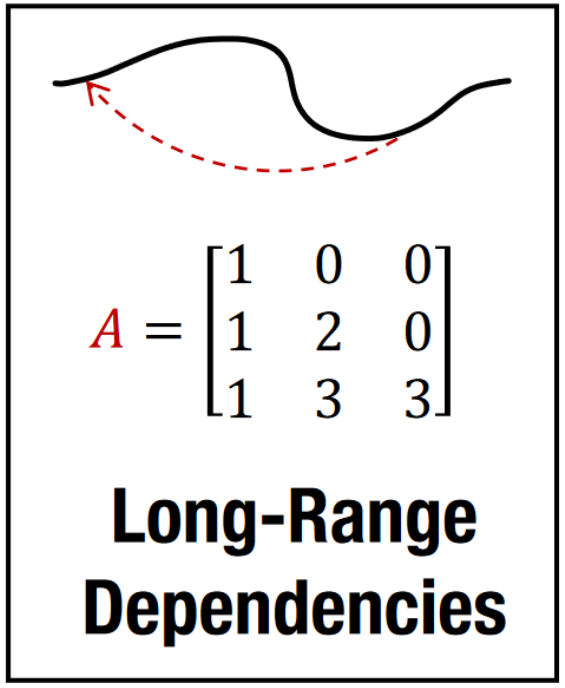
Efficiently Modeling Long Sequences with Structured State Spaces

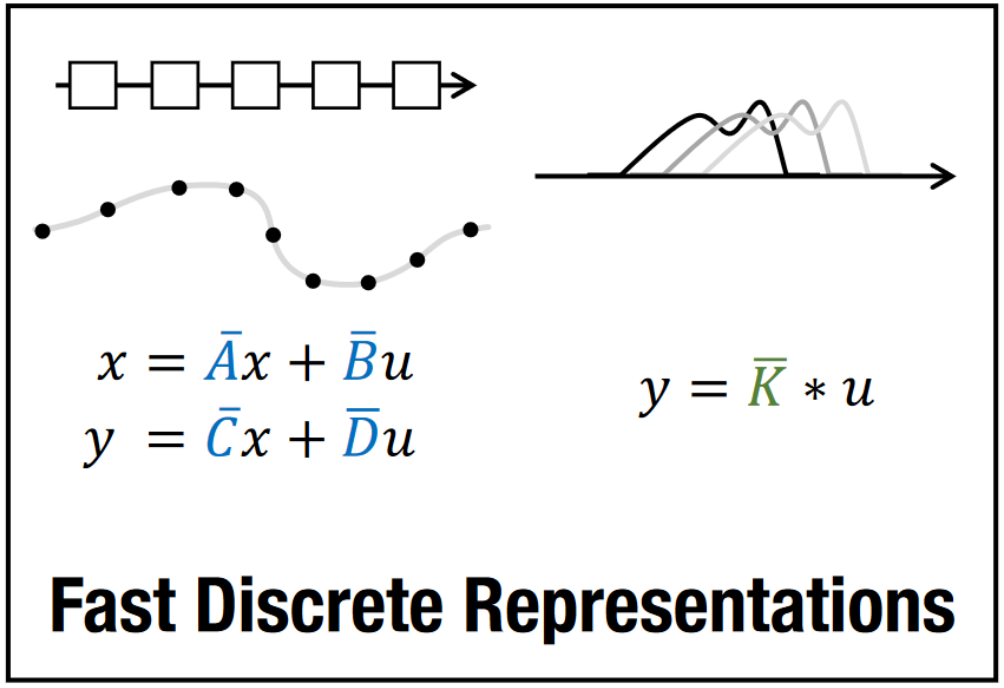
Albert Gu, Karan Goel, and Christopher Re, 2022.8.5

Motivation

- What features are required for good sequence modeling?







Motivation

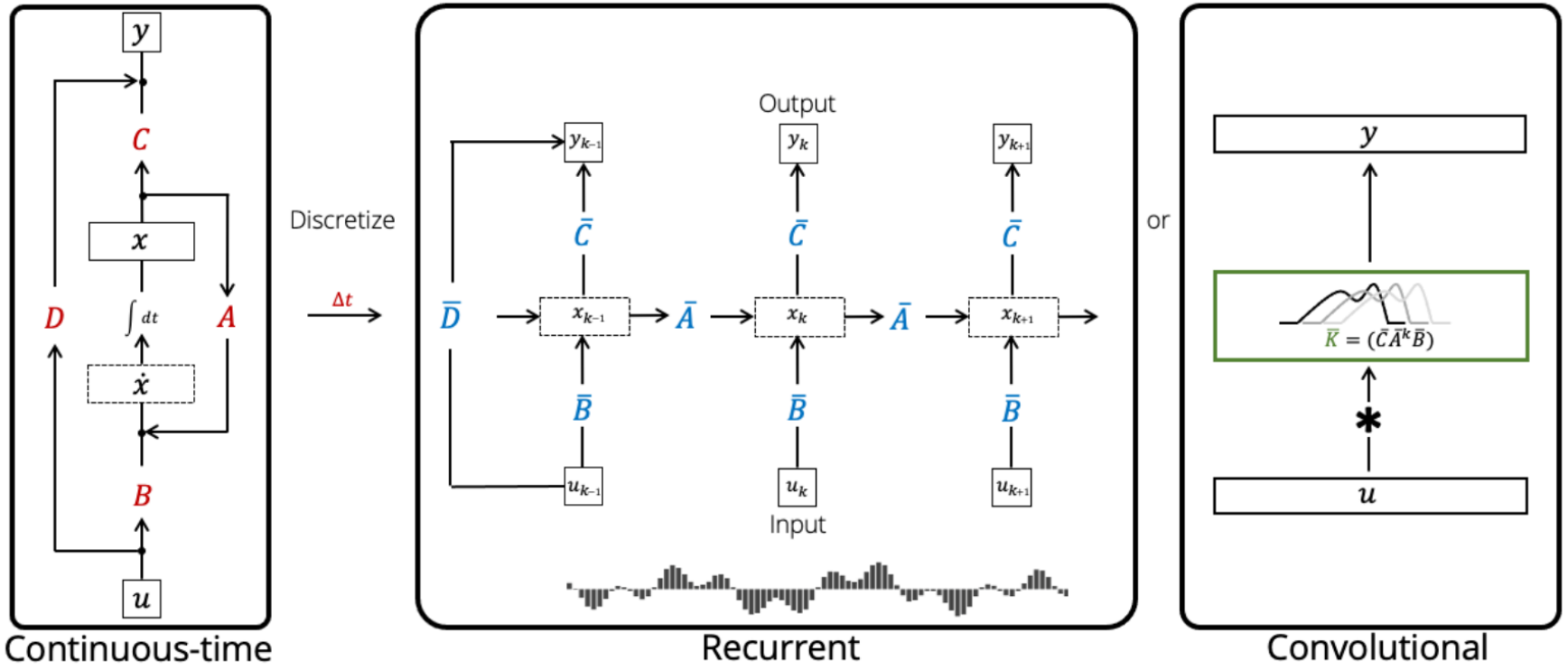
- **continuous state space:** originate from continuous state space model
- **long-range dependencies:** special initialization and parameterization
- **computation efficiency:** leverage RNN and CNN forms

- existing 3 paradigms for time series:

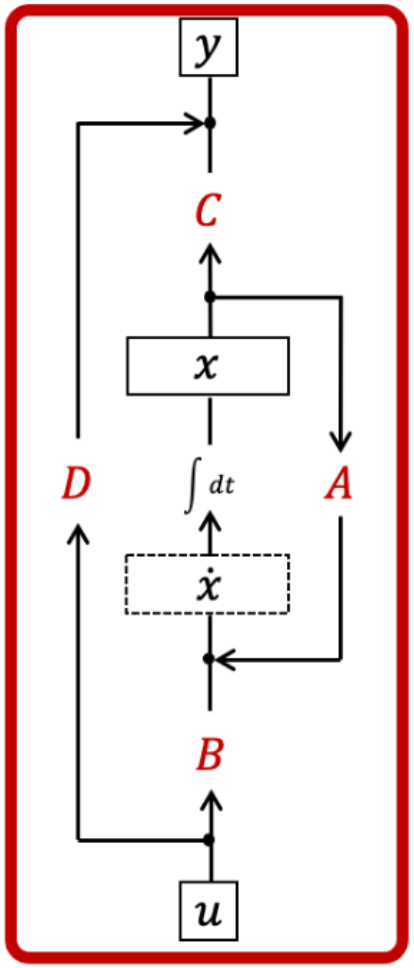
	pros	cons
recurrence	in principle have unbounded context; efficient inference (constant-time state updates)	slow to train (lack of parallelizability); vanishing/exploding gradient problem
convolution	local, interpretable features; efficient training (parallelizable)	slow in autoregressive settings (has to recompute over entire input for every new datapoint); fixed context size
continuous-time (e.g. Neural ODE)	automatically handles continuous data	extremely slow at both training and inference

Overview

- A key feature of the SSM is that it has **three different representations**, which allows it to be viewed (and computed) as a continuous-time, recurrent, or convolutional model.



The Continuous-Time Representation



- main representation: learns how to map an input $u(t)$ to an output $y(t)$ through an intermediate state $x(t)$

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

- Note that this representation is mostly theoretically; we won't be receiving data in the form of closed-form functions so we can't directly apply this representation. However, this is the base form from which the others will be computed, and offers the most intuition into the model's behavior.

Computing the SSM with Recurrence

- Real data is discrete, so what we want from a sequence model is to be able to transform an input 1-D discrete sequence (instead of a continuous input signal $u(t)$)

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

$\bar{A} \in \mathbb{R}^{N\times N}$, $x_k \in \mathbb{R}^{N\times 1}$,
$\bar{B} \in \mathbb{R}^{N\times 1}$, $u_k \in \mathbb{R}$,
$\bar{C} \in \mathbb{R}^{1\times N}$, $y_k \in \mathbb{R}$.
$\bar{D} \in \mathbb{R}$.

- $\bar{A}, \bar{B}, \bar{C}, \bar{D}$ have simple closed formulas in terms of the base parameters A, B, C, D. There are many valid formulas, here we use 'bilinear transformation':

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

Computing the SSM with Recurrence: Multiple discretization

$$x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t) \longrightarrow x_k = \overline{\boldsymbol{A}}x_{k-1} + \overline{\boldsymbol{B}}u_k$$

1. differential limitation

$$x(t+\Delta) - x(t) = \Delta \cdot x'(t)$$
$$x(t+\Delta) = \Delta \cdot x'(t) + x(t)$$

$$\begin{aligned} x(t+\Delta) &= \Delta(Ax(t) + Bu(t)) + x(t) \\ &= \Delta Ax(t) + \Delta Bu(t) + x(t) \\ &= (\Delta A + I)x(t) + \Delta Bu(t) \\ &= \overline{A}x(t) + \overline{B}u(t) \end{aligned}$$

$$\bar{A} = \Delta A + I$$
$$\bar{B} = \Delta B$$

2. bilinear transformation

$$x_{n+1} - x_n = \frac{1}{2}\Delta\left(x'\left(t_n\right) + x'\left(t_{n+1}\right)\right)$$

$$x_{n+1} = x_n + \frac{\Delta}{2}(\mathbf{A}x_n + \mathbf{B}u_n + \mathbf{A}x_{n+1} + \mathbf{B}u_{n+1})$$

$$\iff x_{n+1} - \frac{\Delta}{2}\mathbf{A}x_{n+1} = x_n + \frac{\Delta}{2}\mathbf{A}x_n + \frac{\Delta}{2}\mathbf{B}\left(u_{n+1} + u_n\right)$$

$$(*) \iff \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)x_{n+1} = \left(\mathbf{I} + \frac{\Delta}{2}\mathbf{A}\right)x_n + \Delta\mathbf{B}u_{n+1}$$

$$\iff x_{n+1} = \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1}\left(\mathbf{I} + \frac{\Delta}{2}\mathbf{A}\right)x_n + \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1}\Delta\mathbf{B}u_{n+1}$$

$$\overline{\mathbf{A}} = \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1}\left(\mathbf{I} + \frac{\Delta}{2}\mathbf{A}\right)$$
$$\overline{\mathbf{B}} = \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1}\Delta\mathbf{B}$$

3. zero-order hold (used in mamba)

$$x(t) = e^{\mathbf{A}(t-t_0)}x\left(t_0\right) + \int_{t_0}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{B}u(\tau)d\tau$$

$$\begin{aligned} x_{k+1} &= e^{\mathbf{A}\Delta}x_k + \int_{t_k}^{t_{k+1}} e^{\mathbf{A}(t_{k+1}-\tau)}\mathbf{B}u(\tau)d\tau \\ &= e^{\mathbf{A}\Delta}x_k + \int_{t_k}^{t_{k+1}} e^{\mathbf{A}(t_{k+1}-\tau)}\mathbf{B}d\tau u_k \end{aligned}$$

$$\overline{\mathbf{A}} = e^{\mathbf{A}\Delta} = exp(\mathbf{A}\Delta)$$
$$\overline{\mathbf{B}} = \int_0^{\Delta} e^{\mathbf{A}\tau}d\tau\mathbf{B} = (\Delta\boldsymbol{A})^{-1}(\exp(\Delta \cdot \boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta\boldsymbol{B}$$

Computing SSM with Convolutions

- compute with recurrence:

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

- unroll:

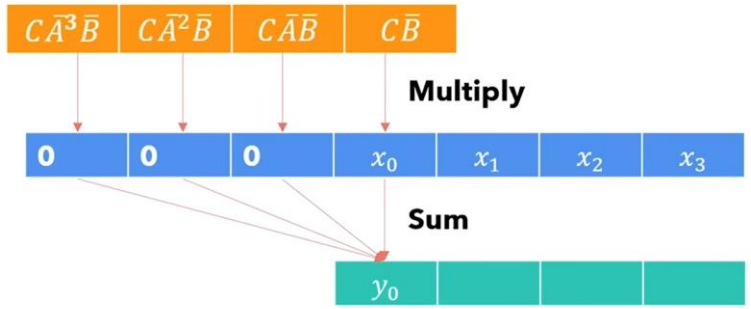
$$x_0 = \overline{\boldsymbol{B}} u_0 \quad x_1 = \overline{\boldsymbol{AB}} u_0 + \overline{\boldsymbol{B}} u_1 \quad x_2 = \overline{\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{AB}} u_1 + \overline{\boldsymbol{B}} u_2 \quad \dots$$
$$y_0 = \overline{\boldsymbol{CB}} u_0 + \overline{\boldsymbol{D}} u_0$$
$$y_1 = \overline{\boldsymbol{CAB}} u_0 + \overline{\boldsymbol{CB}} u_1 + \overline{\boldsymbol{D}} u_1$$
$$y_2 = \overline{\boldsymbol{CA}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CAB}} u_1 + \overline{\boldsymbol{CB}} u_2 + \overline{\boldsymbol{D}} u_2 \quad \dots$$
$$y_k = \overline{\boldsymbol{CA}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CA}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{CAB}} u_{k-1} + \overline{\boldsymbol{CB}} u_k$$

- By extracting these coefficients into what we call the SSM kernel $\overline{K}$

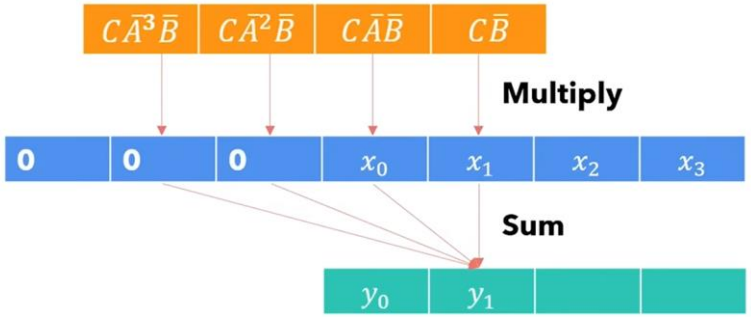
$$\overline{\boldsymbol{K}} = \left(\overline{\boldsymbol{CA}}^i \overline{\boldsymbol{B}}\right)_{i \in [L]} = (\overline{\boldsymbol{CB}}, \overline{\boldsymbol{CAB}}, \dots, \overline{\boldsymbol{CA}}^{L-1} \overline{\boldsymbol{B}}) \quad \rightarrow \quad y = u * k$$

Computing SSM with Convolutions

Notice the notation change:
here the input is x

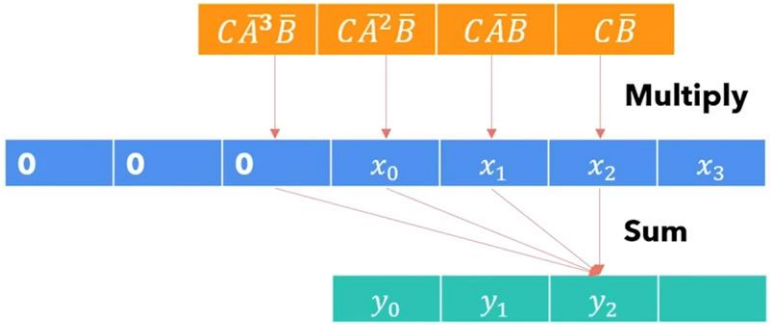


$$y_0 = C\bar{B}x_0$$

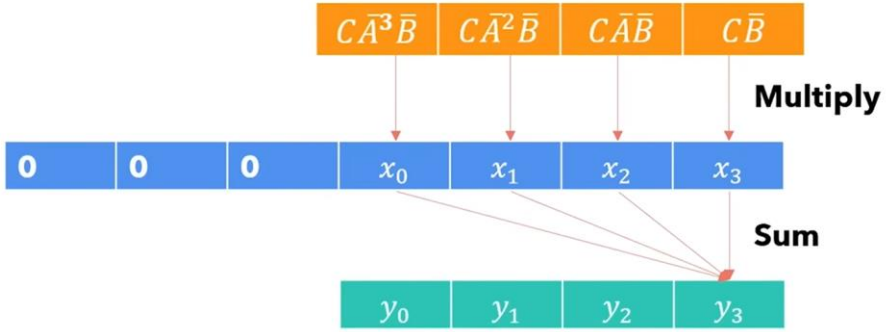


$$y_1 = C\bar{A}\bar{B}x_0 + C\bar{B}x_1$$

Kernel
Input
Output



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



$$y_3 = C\bar{A}^3\bar{B}x_0 + C\bar{A}^2\bar{B}x_1 + C\bar{A}\bar{B}x_2 + C\bar{B}x_3$$

Kernel
Input
Output

Addressing Long-Range Dependencies with HiPPO

- Now an SSM layer can be used either as an RNN or a CNN
- two problems remaining:
  - the randomly initialized SSM actually does not perform very well (mainly due to **vanishing/exploding gradients** problem)
  - computing it naively like we've done so far is really **slow and memory inefficient**
- To solve long-range dependencies, initialize the transition matrix A as HiPPO Matrix
  - intuition: this matrix aims to compress the past history into a state that has enough information to approximately reconstruct the history

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

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

$$(\textbf{HiPPO Matrix}) \qquad \boldsymbol{A}_{nk} = -\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}.$$

Previous work found that simply modifying an SSM from a random matrix A to HiPPO improved its performance on the sequential MNIST classification benchmark from 60% to 98%.

Fast Calculation of Kernel $\overline{\boldsymbol{K}} = \left(\overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{i}\overline{\boldsymbol{B}}\right)_{i\in[L]} = (\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \ldots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$

1. SSM Generating Functions

To address the problem of computing powers of $\bar{A}$, instead of computing the SSM convolution filter directly, we introduce a generating function on its coefficients and compute evaluations of it.

Definition 2 (SSM Generating Function). *We define the following quantities:*

- *The* SSM convolution function *is* $\mathcal{K}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}) = (\overline{\boldsymbol{C}}^{*}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}^{*}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \ldots)$ *and the (truncated) SSM filter of length* L

$$\mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}) = (\overline{\boldsymbol{C}}^{*}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}^{*}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \ldots, \overline{\boldsymbol{C}}^{*}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}}) \in \mathbb{R}^{L} \tag{9}$$

- *The* SSM generating function *at node* z *is*

$$\hat{\mathcal{K}}(z; \overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}) \in \mathbb{C} := \sum_{i=0}^{\infty} \overline{\boldsymbol{C}}^{*}\overline{\boldsymbol{A}}^{i}\overline{\boldsymbol{B}}z^{i} = \overline{\boldsymbol{C}}^{*}(\boldsymbol{I} - \overline{\boldsymbol{A}}z)^{-1}\overline{\boldsymbol{B}} \tag{10}$$

and the truncated SSM generating function *at node* z *is*

$$\hat{\mathcal{K}}_L(z; \overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})^{*} \in \mathbb{C} := \sum_{i=0}^{L-1} \overline{\boldsymbol{C}}^{*}\overline{\boldsymbol{A}}^{i}\overline{\boldsymbol{B}}z^{i} = \overline{\boldsymbol{C}}^{*}(\boldsymbol{I} - \overline{\boldsymbol{A}}^{L}z^{L})(\boldsymbol{I} - \overline{\boldsymbol{A}}z)^{-1}\overline{\boldsymbol{B}} \tag{11}$$

Lemma C.2. *The SSM function* $\mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})$ *can be computed from the SSM generating function* $\hat{\mathcal{K}}_L(\Omega; \overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})$ *at the roots of unity* $\Omega = \{\exp(-2\pi i\frac{k}{L} : k \in [L]\}$ *stably in* $O(L \log L)$ *operations.*

Fast Calculation of Kernel $\overline{\boldsymbol{K}} = \left(\overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{i}\overline{\boldsymbol{B}}\right)_{i\in[L]} = (\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \ldots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$

1. SSM Generating Functions: change power to inverse

Lemma C.2. *The SSM function $\mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})$ can be computed from the SSM generating function $\hat{\mathcal{K}}_L(\Omega; \overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})$ at the roots of unity $\Omega = \{\exp(-2\pi i \frac{k}{L} : k \in [L]\}$ stably in $O(L \log L)$ operations.*

$$\hat{\mathcal{K}}_L(z; \overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})^* \in \mathbb{C} := \sum_{i=0}^{L-1} \overline{\boldsymbol{C}}^* \overline{\boldsymbol{A}}^i \overline{\boldsymbol{B}} z^i = \overline{\boldsymbol{C}}^* (\boldsymbol{I} - \overline{\boldsymbol{A}}^L z^L)(\boldsymbol{I} - \overline{\boldsymbol{A}} z)^{-1} \overline{\boldsymbol{B}} \quad = \quad \boxed{\widetilde{\boldsymbol{C}}^* (\boldsymbol{I} - \overline{\boldsymbol{A}})^{-1} \overline{\boldsymbol{B}}}$$

for all $z \in \Omega_L$, $z^L = 1$

2. Diagonal Plus Low-Rank (DPLR)

Lemma C.4. *Let $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$ be the SSM matrices $\boldsymbol{A}, \boldsymbol{B}$ discretized by the bilinear discretization with step size Δ. Then*

$$\boldsymbol{C}^* \left(\boldsymbol{I} - \overline{\boldsymbol{A}}\boldsymbol{z}\right)^{-1} \overline{\boldsymbol{B}} = \frac{2\Delta}{1+z} \boldsymbol{C}^* \left[2\frac{1-z}{1+z} - \Delta \boldsymbol{A}\right]^{-1} \boldsymbol{B}$$

$$\overline{\boldsymbol{A}} = \left(\boldsymbol{I} - \tfrac{\Delta}{2}\boldsymbol{A}\right)^{-1} \left(\boldsymbol{I} + \tfrac{\Delta}{2}\boldsymbol{A}\right)$$
$$\overline{\boldsymbol{B}} = \left(\boldsymbol{I} - \tfrac{\Delta}{2}\boldsymbol{A}\right)^{-1} \Delta \boldsymbol{B}$$

Lemma C.3. *Let $\boldsymbol{A} = \boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*$ be a diagonal plus low-rank representation. Then for any root of unity $z \in \Omega$, the truncated generating function satisfies*

$$\hat{\boldsymbol{K}}(z) = \frac{2}{1+z}\left[\tilde{\boldsymbol{C}}^* \boldsymbol{R}(z)\boldsymbol{B} - \tilde{\boldsymbol{C}}^* \boldsymbol{R}(z)\boldsymbol{P}\left(1 + \boldsymbol{Q}^* \boldsymbol{R}(z)\boldsymbol{P}\right)^{-1} \boldsymbol{Q}^* \boldsymbol{R}(z)\boldsymbol{B}\right]$$

$$\tilde{\boldsymbol{C}} = (\boldsymbol{I} - \overline{\boldsymbol{A}}^L)^* \boldsymbol{C}$$

$$\boldsymbol{R}(z; \boldsymbol{\Lambda}) = \left(\frac{2}{\Delta}\frac{1-z}{1+z} - \boldsymbol{\Lambda}\right)^{-1}.$$

Parameters for S4: $\boldsymbol{\Lambda}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{P}, \boldsymbol{Q}$, $5\boldsymbol{N}$ parameters

Turning HiPPO to DPLR

1. HiPPO: Normal Plus Low-Rank (NPLR)

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

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

for unitary $\boldsymbol{V} \in \mathbb{C}^{N\times N}$*, diagonal* $\boldsymbol{\Lambda}$*, and low-rank factorization* $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{R}^{N\times r}$.

2. NPLR ~ DPLR in SSM

Lemma 3.1. *Conjugation is an equivalence relation on SSMs* $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) \sim (\boldsymbol{V}^{-1}\boldsymbol{A}\boldsymbol{V}, \boldsymbol{V}^{-1}\boldsymbol{B}, \boldsymbol{C}\boldsymbol{V})$.

Proof. Write out the two SSMs with state denoted by x and $\tilde{x}$ respectively:

$$\begin{aligned} x' &= \boldsymbol{A}x + \boldsymbol{B}u & \qquad \tilde{x}' &= \boldsymbol{V}^{-1}\boldsymbol{A}\boldsymbol{V}\tilde{x} + \boldsymbol{V}^{-1}\boldsymbol{B}u \\ y &= \boldsymbol{C}x & \qquad y &= \boldsymbol{C}\boldsymbol{V}\tilde{x} \end{aligned}$$

After multiplying the right side SSM by $\boldsymbol{V}$, the two SSMs become identical with $x = \boldsymbol{V}\tilde{x}$. Therefore these compute the exact same operator $u \mapsto y$, but with a change of basis by $\boldsymbol{V}$ in the state x. □

Theoretical Efficency

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	$\boldsymbol{H^2}$

Performance

Table 2: Deep SSMs: The S4 parameterization with Algorithm 1 is asymptotically more efficient than the LSSL.

	Training Step (ms)			Memory Alloc. (MB)		
Dim.	128	256	512	128	256	512
LSSL	9.32	20.6	140.7	222.1	1685	13140
S4	4.77	3.07	4.75	5.3	12.6	33.5
Ratio	1.9×	6.7×	**29.6**×	42.0×	133×	**392**×

Table 3: Benchmarks vs. efficient Transformers

	Length 1024		Length 4096	
	Speed	Mem.	Speed	Mem.
Transformer	1×	1×	1×	1×
Performer	1.23×	0.43×	3.79×	0.086×
Linear Trans.	**1.58**×	**0.37**×	**5.35**×	**0.067**×
S4	**1.58**×	0.43×	5.19×	0.091×

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**

Resurrecting Recurrent Neural Networks for Long Sequences

Google 2023.3.11

Linear RNN

- An classical RNN layer with N-dimensional hidden states:
  - let $(u_1, u_2, \cdots, u_L)$ be a sequence of H_{in}-dimensional inputs
  - $x_t = \sigma(Ax_{k-1} + Bu_k), y_k = Cx_k + Du_k$
  - learnable parameters: $A \in \mathbb{R}^{N \times N}, B \in \mathbb{R}^{N \times H_{in}}, C \in \mathbb{R}^{H_{out} \times N}, D \in \mathbb{R}^{H_{out} \times H_{in}}$
- If σ is the identity function, then we say the RNN layer is linear.
- Surprisingly, linear RNNs are performant with when coupled with nonlinear MLP or GLU blocks

Recurrence	sCIFAR	ListOps	Text	Retrieval
RNN-ReLU	69.7 (0.2)	37.6 (8.0)	88.0 (0.1)	88.5 (0.1)
RNN-Tanh	69.9 (0.3)	43.9 (0.1)	87.2 (0.1)	88.9 (0.2)
RNN-Lin	**72.2** (0.2)	**50.4** (0.2)	**89.1** (0.1)	**89.1** (0.1)

Diagonalization

$$x_0 = Bu_0, x_1 = ABu_0 + Bu_1, x_2 = A^2Bu_0 + ABu_1 + Bu_2, \cdots$$

$$x_k = \sum_{j=0}^{k-1} A^j B u_{k-j}$$

- $A \in \mathbb{R}^{N\times N}$ is diagonalizable in complex field, i.e. $A = P\Lambda P^{-1}$

$$\overline{x_k} = \sum_{j=0}^{k-1} \Lambda^j \bar{B} u_{k-j}$$

- where $\Lambda = diag(\lambda_1, \cdots, \lambda_N), \overline{x_k} = P^{-1}x_k, \overline{B} = P^{-1}B.$
- Then the output can be calculated as $y_k = \Re\left[\overline{C}\overline{x_k}\right] + Du_k$, where $\overline{C} = CP^{-1}$
- $\left|\lambda_j\right| < 1$ for all j to ensure stability

Parameterization

$$\overline{x_k} = \sum_{j=0}^{k-1} \Lambda^j \overline{B} u_{k-j}$$

where $\Lambda = diag(\lambda_1, \cdots, \lambda_N), \overline{x_k} = P^{-1}x_k, \overline{B} = P^{-1}B.$

1. real and imaginary parameterization: $\Lambda = real + i \cdot imag$,
2. exponential parameterization: $\Lambda = diag(\exp(-\nu + i\theta))$,
3. stable exponential parameterization: $\lambda_j = \exp(-\exp\left(\nu_j^{log}\right) + i\theta_j)$
   - optimize $\nu^{log} \in \mathbb{R}^N$
   - enforce stability, enforcing ν to be positive so that $\left|\lambda_j\right| \leq 1$

	sCIFAR	ListOps	Pathfinder
Dense A	72.2 (0.2)	50.4 (0.2)	✗
Λ Real + Im	86.5 (0.1)	58.8 (0.3)	✗
Λ Exp	85.4 (0.7)	**60.5** (0.3)	65.4 (9.0)
Λ Stable Exp	87.2 (0.4)	59.4 (0.3)	93.5 (0.5)
+ Ring Init	**88.1** (0.0)	59.4 (0.3)	**94.4** (0.3)

Initialization

Theorem 3.1 (Strong circular law). *Let μ_N be the empirical spectral measure of A_N, where A_N is a real $N \times N$ matrix with i.i.d. Gaussian entries, each with zero mean and variance $1/N$. Then, μ_N converges weakly almost surely as $N \to \infty$ to the uniform probability measure on $\{|z| \leq 1\} \subseteq \mathbb{C}$.*

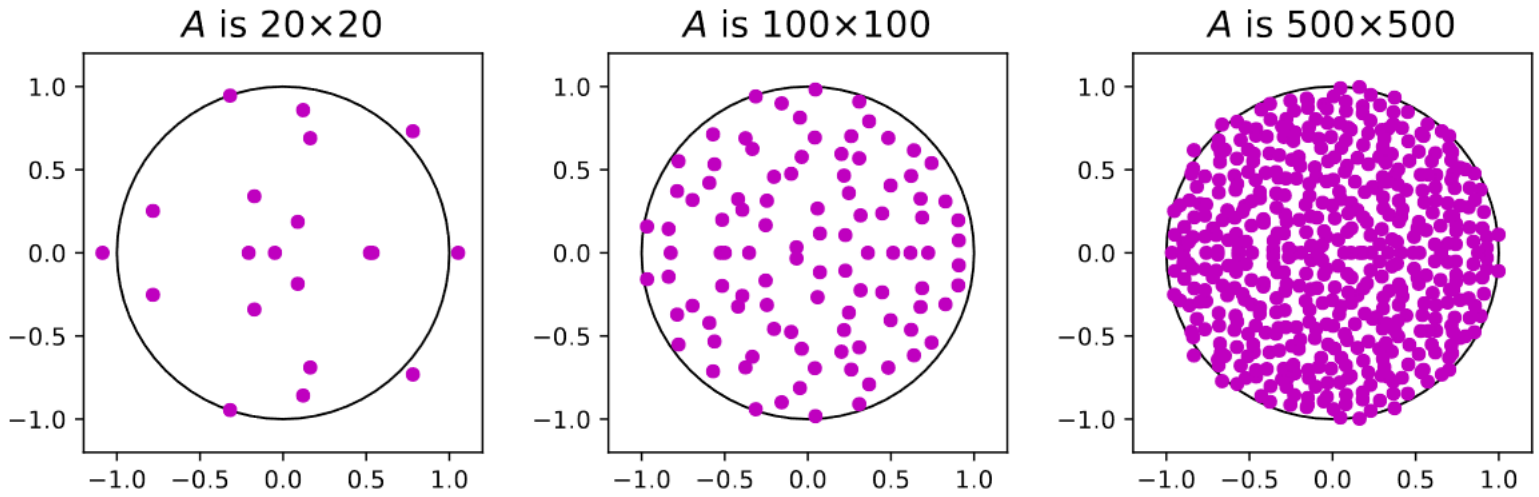


Figure 2 | *Eigenvalues of $A \in \mathbb{R}^{N \times N}$ following Glorot initialization: each entry of A is sampled independently from a Gaussian with mean 0 and variance $1/N$. The eigenvalues are complex (A is not symmetric) and are represented on the complex plane. The black circle is the unit disk $\{|z| = 1\} \subseteq \mathbb{C}$. The limit behavior (uniform initialization) is predicted by Thm. 3.1.*

Initialization

Lemma 3.2. *Let u_1, u_2 be independent uniform random variables on the interval $[0, 1]$. Let $0 \leq r_{\min} \leq r_{\max} \leq 1$. Compute $\nu = -\frac{1}{2} \log \left(u_1 (r_{\max}^2 - r_{\min}^2) + r_{\min}^2 \right)$ and $\theta = 2\pi u_2$. Then $\exp(-\nu + i\theta)$ is uniformly distributed on the ring in $\mathbb{C}$ between circles of radii $r_{\min}$ and $r_{\max}$.*

- recover the spectrum of Glorot initialization by setting $r_{\min} = 0$ and $r_{\max} = 1$

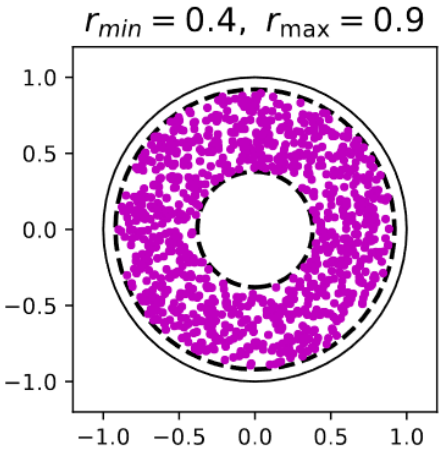


Figure 3 | *Eigenvalues of a diagonal matrix A with entries sampled using Lemma 3.2. For $r_{\min} = 0$, $r_{\max} = 1$, the distribution coincides with Glorot init. in the limit.*

	sCIFAR	ListOps	Pathfinder
Dense A	72.2 (0.2)	50.4 (0.2)	✗
Λ Real + Im	86.5 (0.1)	58.8 (0.3)	✗
Λ Exp	85.4 (0.7)	**60.5** (0.3)	65.4 (9.0)
Λ Stable Exp	87.2 (0.4)	59.4 (0.3)	93.5 (0.5)
+ Ring Init	**88.1** (0.0)	59.4 (0.3)	**94.4** (0.3)

Normalization

Proposition 3.3 (Forward-pass blow-up). *Let Λ be diagonal with eigenvalues sampled uniformly on the ring in $\mathbb{C}$ between circles of radii $r_{\min} < r_{\max} < 1$. Then, under constant or white-noise input and Glorot input projection, we have that the squared norm of the state x_k converges as $k \to \infty$ to the following quantity.*

$$\mathbb{E}[\|x_\infty\|_2^2] = \frac{1}{r_{\max}^2 - r_{\min}^2} \log\left(\frac{1 - r_{\min}^2}{1 - r_{\max}^2}\right) \mathbb{E}[\|Bu\|_2^2].$$

This result has the following intuitive form if $r_{\min} = r_{\max} = r$: if we initialize ρ-close to the unit disk, the forward pass blows up by a factor $1/\rho$ (since the contributions from previous states take longer to decay): let $\epsilon = r_{\max}^2 - r_{\min}^2$ and $\rho = 1 - r_{\max}^2$, then:

$$\lim_{\epsilon \to 0} \frac{\mathbb{E}[\|x_\infty\|_2^2]}{\mathbb{E}[\|Bu\|_2^2]} = \lim_{\epsilon \to 0} \left[\frac{1}{\epsilon} \log\left(1 + \frac{\epsilon}{\rho}\right)\right] = \lim_{\epsilon \to 0} \left[\frac{1}{\epsilon}\left(\frac{\epsilon}{\rho} + O(\epsilon^2)\right)\right] = \frac{1}{\rho} = \frac{1}{1 - r^2}. \tag{5}$$

according to $\overline{x_k} = \sum_{j=0}^{k-1} \Lambda^j \bar{B} u_{k-j}$

let $B = 1, \Lambda = \lambda$

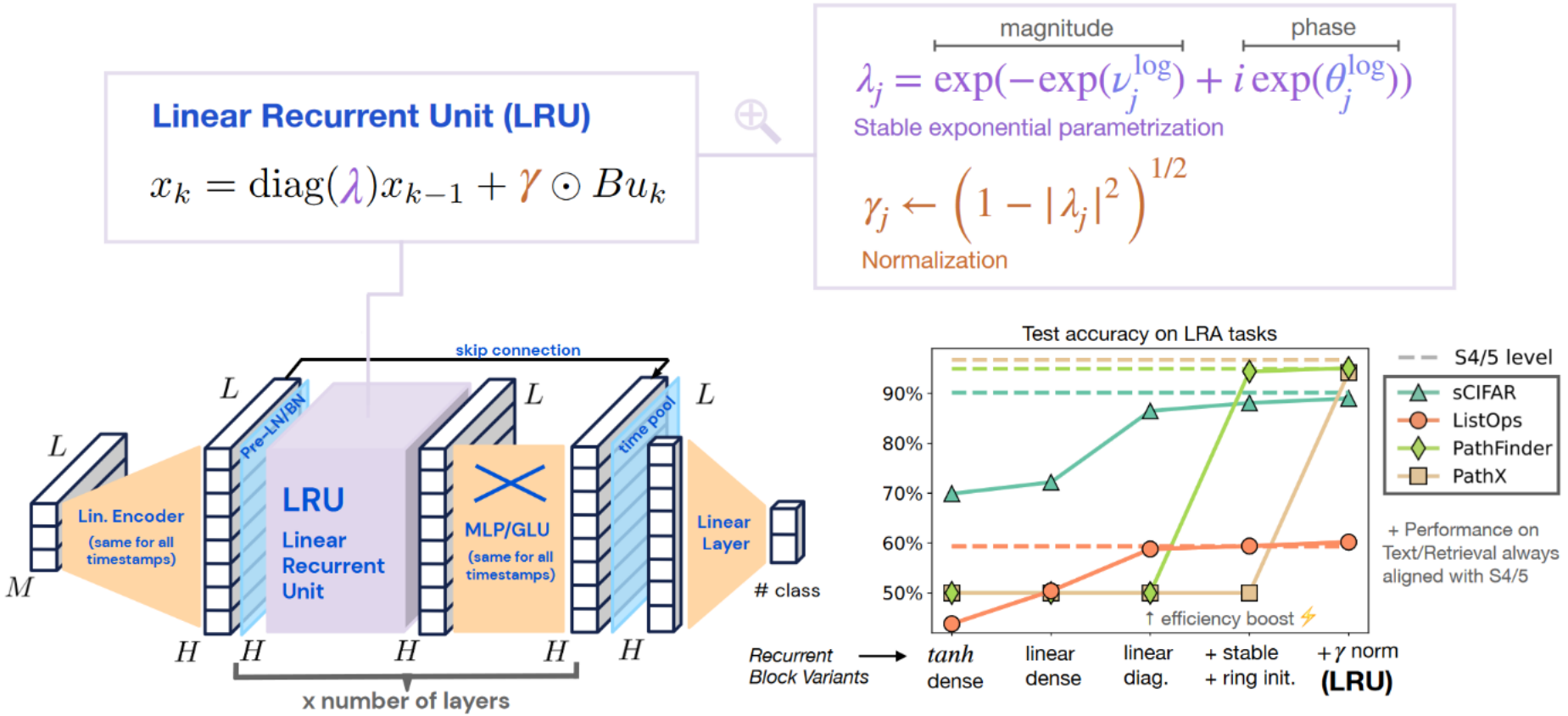
$\mathbb{E}[u_{k-i} u_{k-j}] = 0$ for $i \neq j$

$$\mathbb{E}|x_k|^2 = \left(\sum_{i=0}^{k-1} \lambda^i \mathbb{E}[u_{k-i}]\right)\left(\sum_{j=0}^{k-1} \lambda^j \mathbb{E}[u_{k-j}]\right)^* = \sum_{i,j=0}^{k-1} \lambda^i (\lambda^j)^* \mathbb{E}[u_{k-i} u_{k-j}] = \sum_{i=0}^{k-1} |\lambda|^{2i} \overset{\infty}{\to} \frac{1}{1 - |\lambda|^2}. \tag{6}$$

Additionally, note that as λ approaches 1, $1 - |\lambda|^2 \to 0$, parameterize as $\exp \gamma^{log}$

$$x_k = \Lambda x_{k-1} + \exp(\gamma^{\log}) \odot (Bu_k),$$

Architecture & Performance



- recover the efficiency and performance of SSMs (e.g. S4) using RNNs

Mamba:
Linear-Time Sequence Modeling with Selective State Spaces

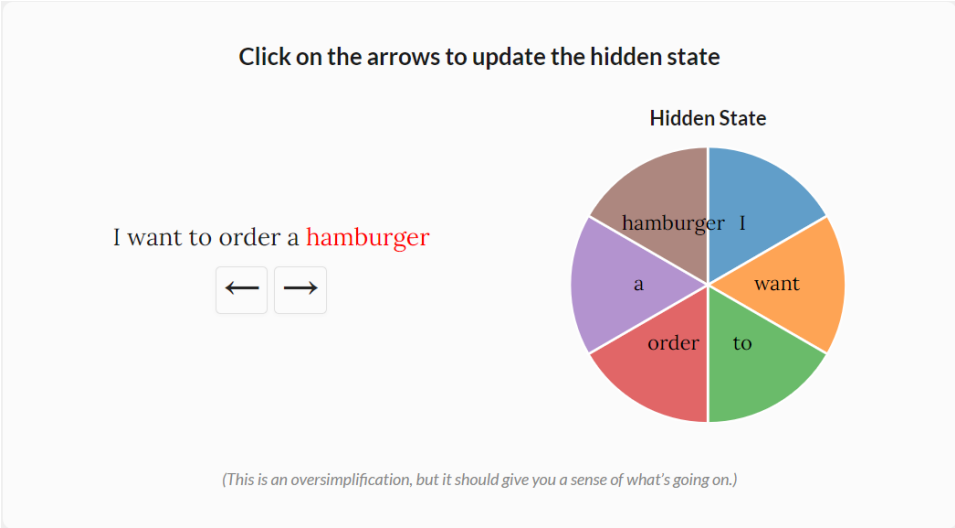
Albert Gu and Tri Dao 2023.12.1

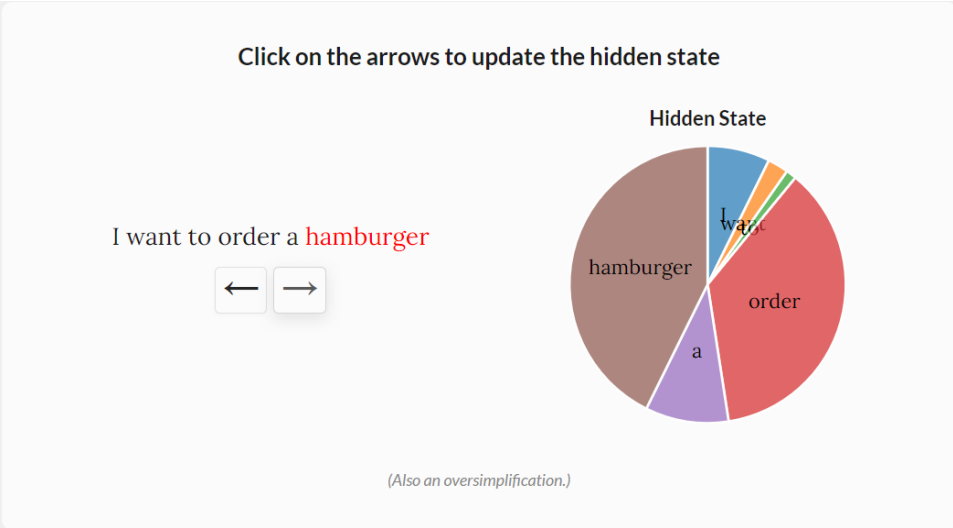
Idea 1: Selectivity

- time-invariant

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

- S4 does not have selectivity, it is forced to treat all parts of the input exactly the same.



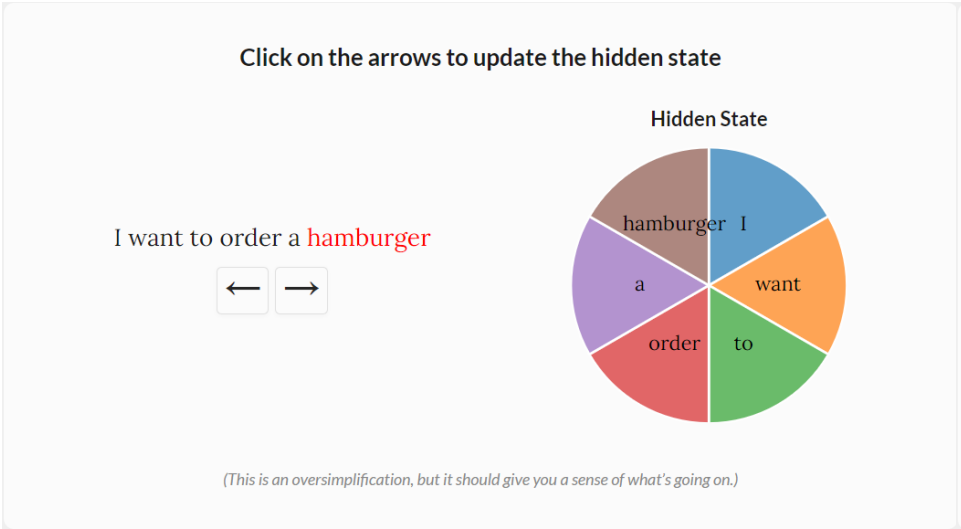


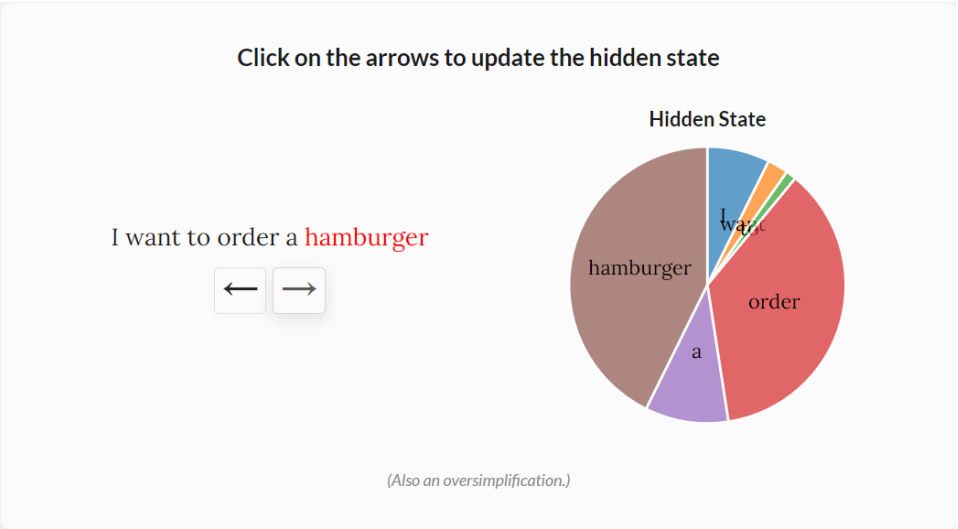
Idea 1: Selectivity

- time-invariant → time-varying

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

$$h_t = s_{\bar{\mathbf{A}}}(x_t) h_{t-1} + s_{\bar{\mathbf{B}}}(x_t) x_t$$
$$y_t = s_{\mathbf{C}}(x_t) h_t$$





Reminder: Computing SSM with Convolutions in S4

- compute with recurrence:

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

- unroll:

$$x_0 = \overline{\boldsymbol{B}} u_0$$
$$y_0 = \overline{\boldsymbol{C}}\overline{\boldsymbol{B}} u_0$$
$$y_1 = \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}} u_1$$
$$y_2 = \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^2\overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}} u_1 + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}} u_2 \quad \dots$$
$$y_k = \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^k\overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}} u_{k-1} + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}} u_k$$

not the case in mamba,
where A, B, C depends on input…

We need to give up training in conv and the parallel.

- By extracting these coefficients into what we call the SSM kernel $\overline{K}$

$$\overline{\boldsymbol{K}} = \left(\overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^i\overline{\boldsymbol{B}}\right)_{i\in[L]} = (\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}}) \quad \rightarrow \quad y = u * k$$

Idea 2: Fast Training without Convolutions

- parallel scan
  - prefix sum:

time-invariant:

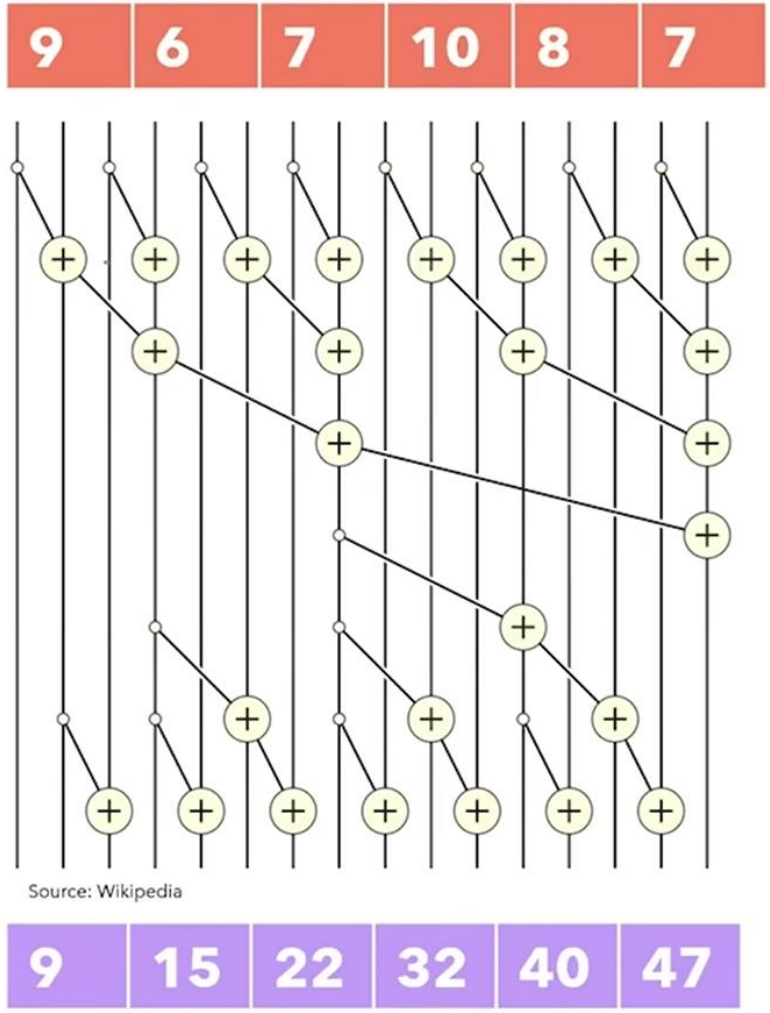
$y_0 = C\bar{B}x_0,$
$y_1 = C(\bar{A}\,\bar{B}\,x_0 + \bar{B}\,x_1),$
…
$y_k = C\bar{A}^k\bar{B}\,x_0 + C\bar{A}^{k-1}\bar{B}\,\ldots + C\bar{B}\,x_k.$

time-varying:

$y_0 = C_0\overline{B_0}x_0,$
$y_1 = C_1(\overline{A_1}\,\overline{B_0}\,x_0 + \overline{B_1}\,x_1),$
…
$y_k = C_k(\overline{A_k}\cdots\overline{A_1}\,\overline{B_0}\,x_0 + \cdots + \overline{B_k}\,x_k).$

define a new operation $\oplus$:

$$m = (\overline{A_0}, \overline{B_0}\,x_0); n = (\overline{A_1}, \overline{B_1}x_1)$$
$$m \oplus n = (\overline{A_0}\,\overline{A_1}, \overline{A_1}\overline{B_0}x_0 + \overline{B_1}x_1)$$
$$(m \oplus n) \oplus k = m \oplus (n \oplus k)$$



Algorithm

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t,$$
$$y_t = Ch_t,$$

$A \in \mathbb{R}^{N\times N}$, $h_t \in \mathbb{R}^{N\times 1}$,
$B \in \mathbb{R}^{N\times 1}$, $x_t \in \mathbb{R}$,
$C \in \mathbb{R}^{1\times N}$, $y_t \in \mathbb{R}$.

notation:
B: batch size
L: sequence length
D: input dimension

$$s_B(x) = Linear_N(x)$$
$$s_C(x) = Linear_N(x)$$
$$s_\Delta(x) = Broadcast_D(Linear_1(x))$$
$$\tau_\Delta = softplus$$

Algorithm 1 SSM (S4)

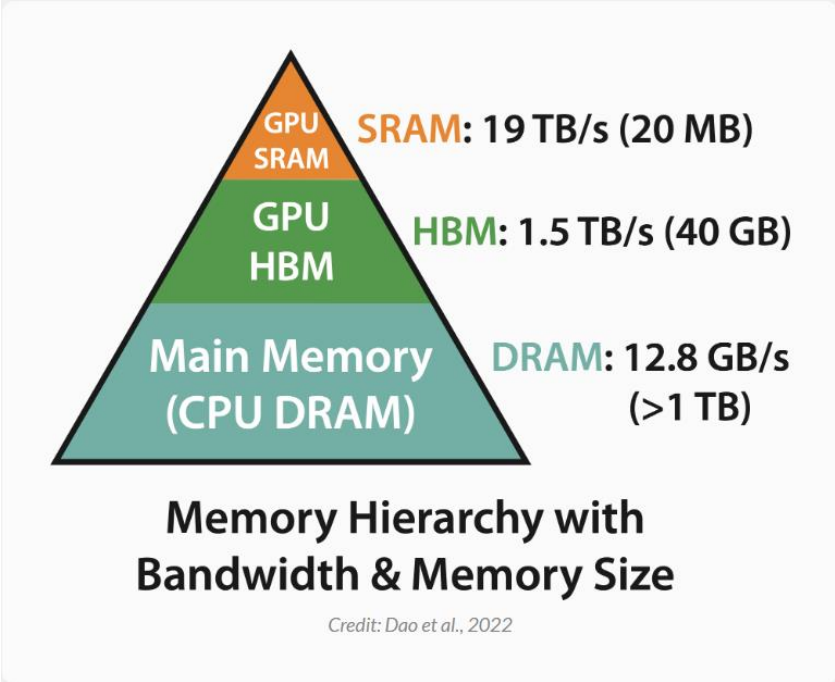
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
3: $\boldsymbol{C} : (\mathtt{D}, \mathtt{N}) \leftarrow$ 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

Idea 3: Use GPU Well

- The GPU memory hierarchy



Idea 3: Use GPU Well

- Exploiting the memory hierarchy

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: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

The main idea is to leverage properties of modern accelerators (GPUs) to materialize the state h only in more efficient levels of the memory hierarchy. In particular, most operations (except matrix multiplication) are bounded by memory bandwidth (Dao, Fu, Ermon, et al. 2022; Ivanov et al. 2021; Williams, Waterman, and Patterson 2009). This includes our scan operation, and we use kernel fusion to reduce the amount of memory IOs, leading to a significant speedup compared to a standard implementation.

Concretely, instead of preparing the scan input $(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}})$ of size $(\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N})$ in GPU HBM (high-bandwidth memory), we load the SSM parameters $(\Delta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ 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.

Idea 3: Use GPU Well

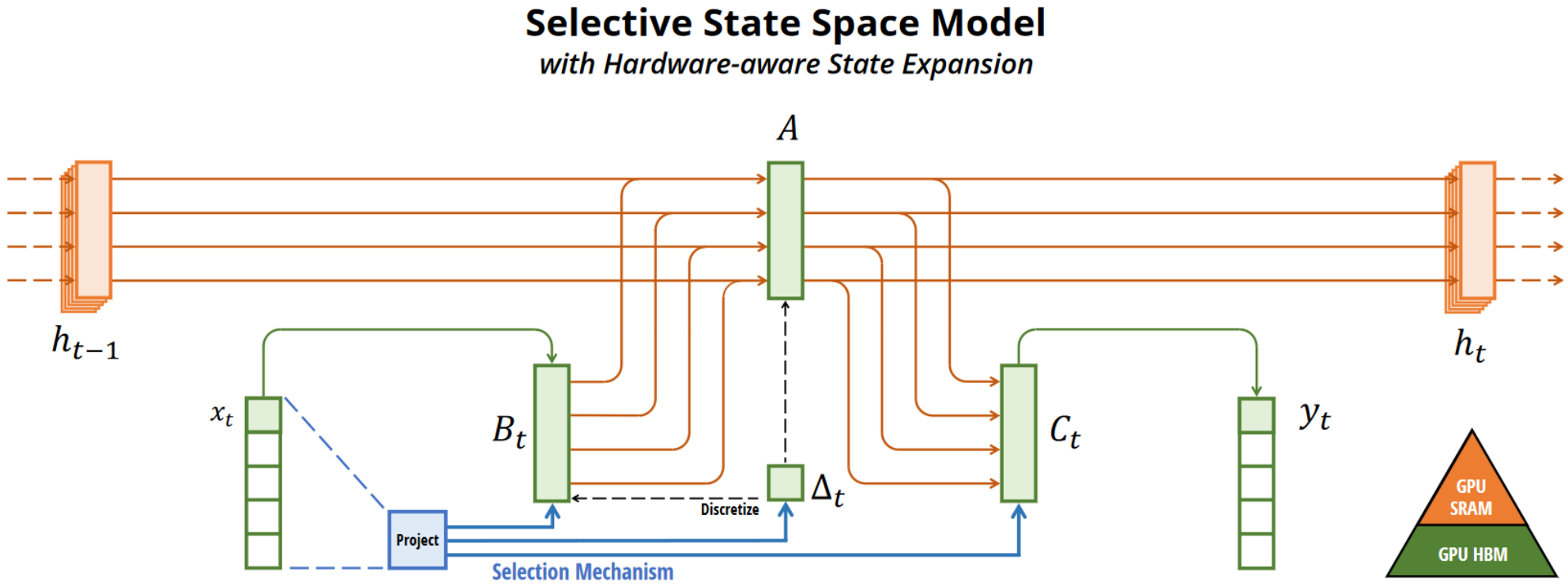


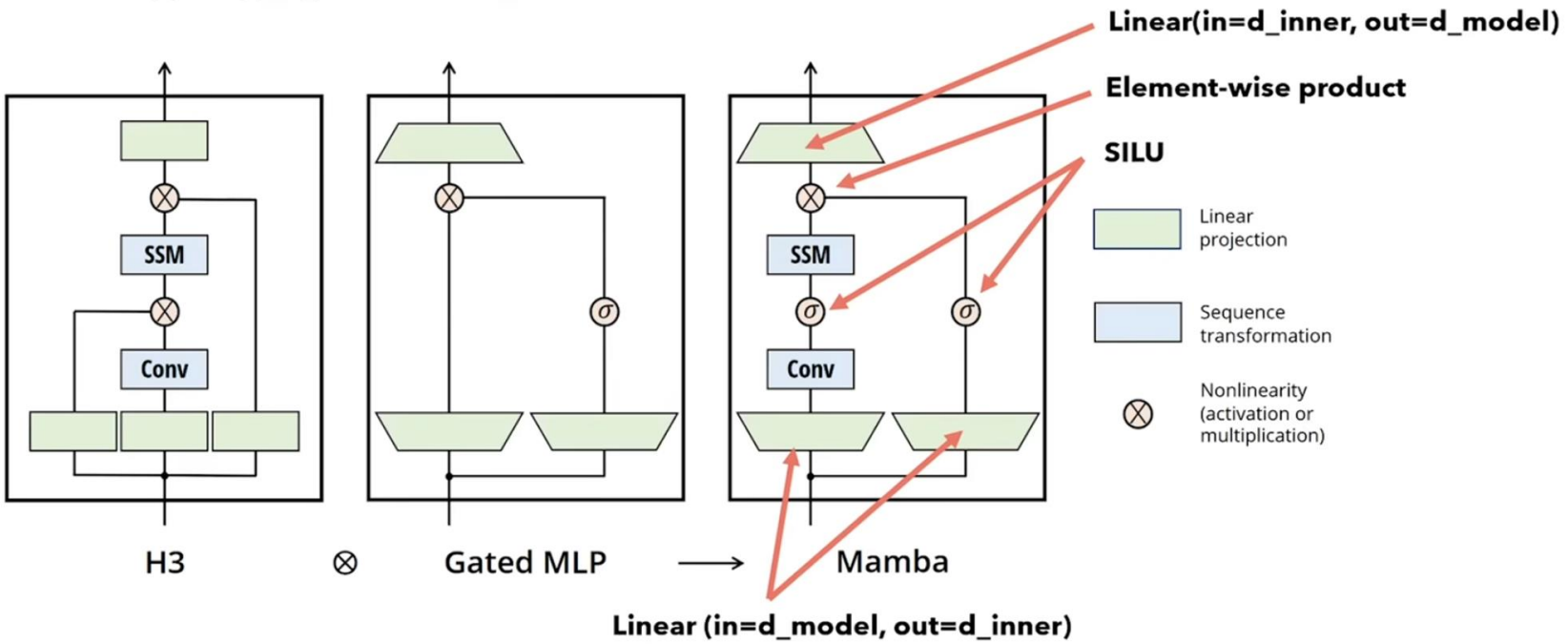
Figure 1: (**Overview.**) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate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

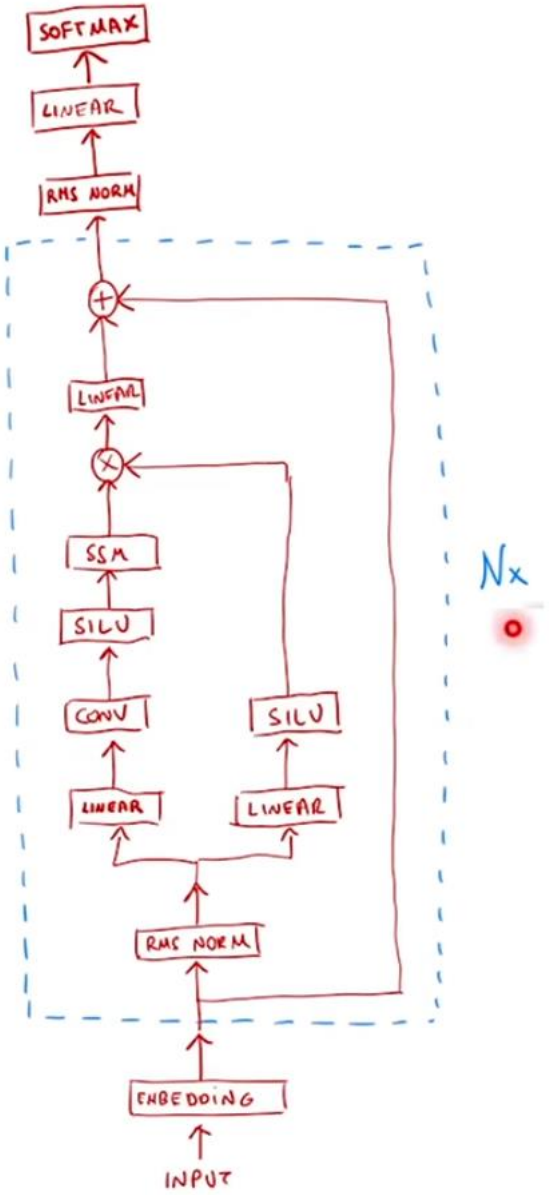
Idea 3: Use GPU Well

- Recomputation

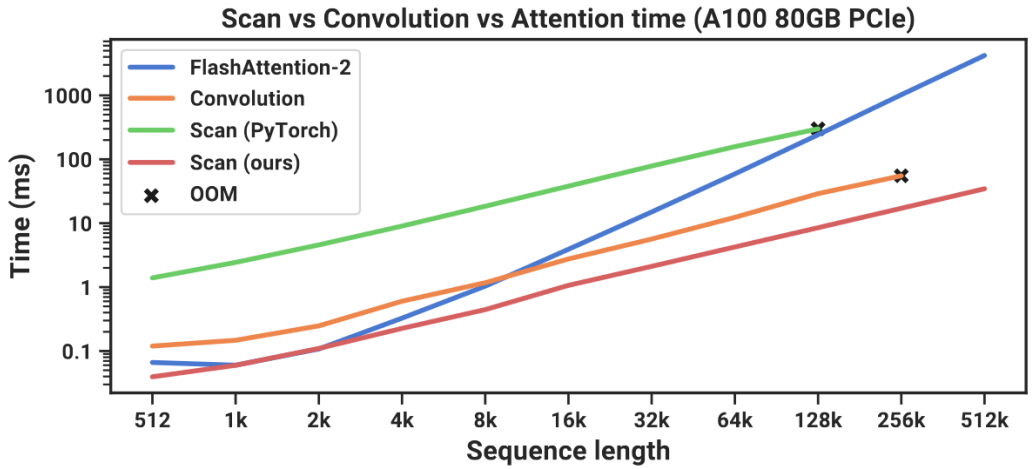
Finally, we must also avoid saving the intermediate states, which are necessary for backpropagation. We carefully apply the classic technique of recomputation to reduce the memory requirements: the intermediate states are not stored but recomputed in the backward pass when the inputs are loaded from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

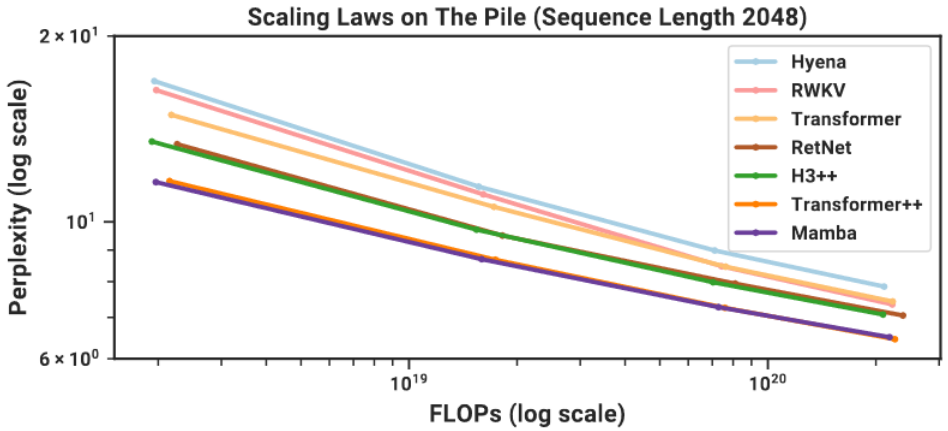
Architecture

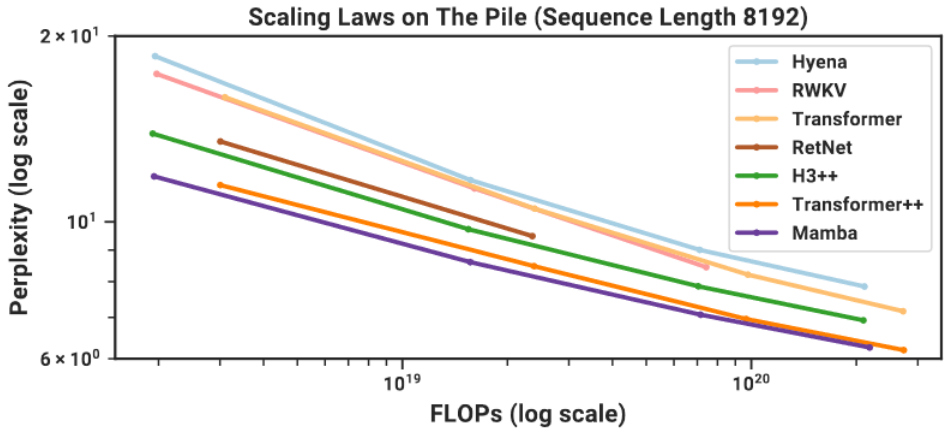




Performance







Applications of Mamba

1. Vmamba: Visual State Space Model

- Cross-Scan Module

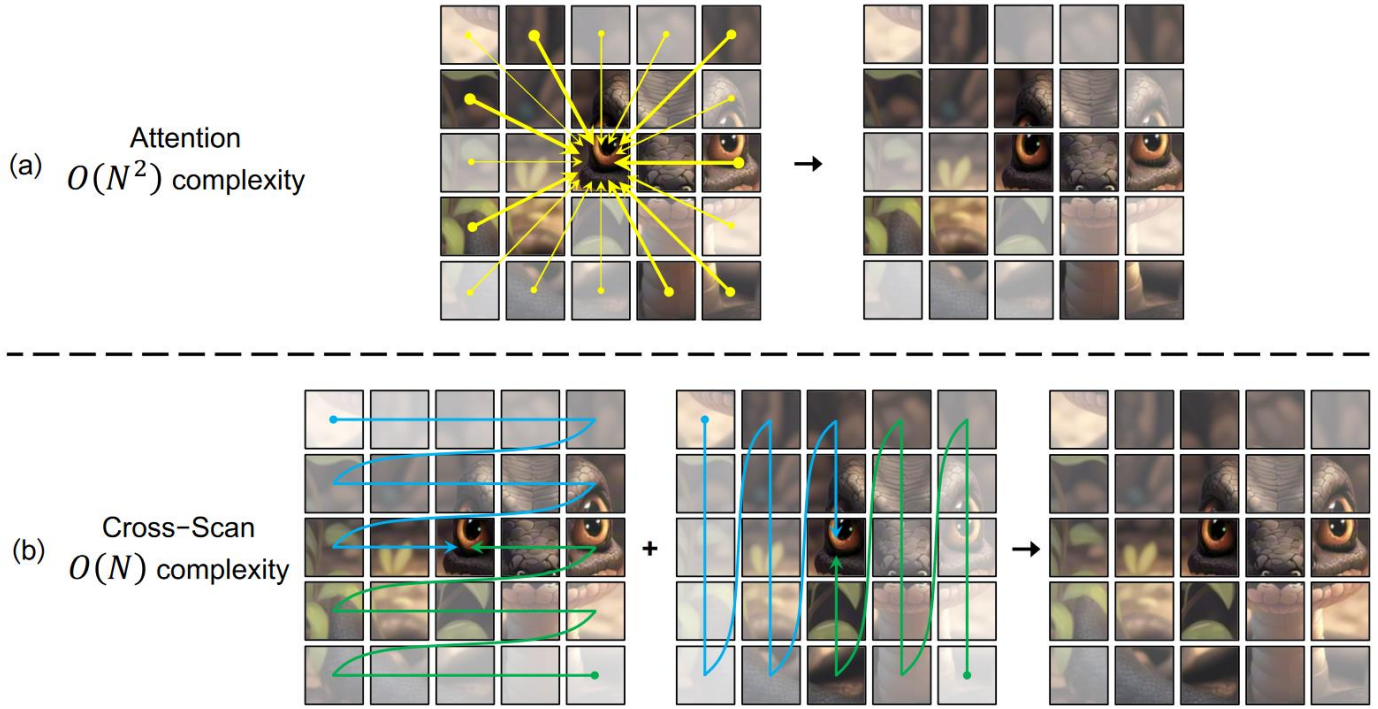
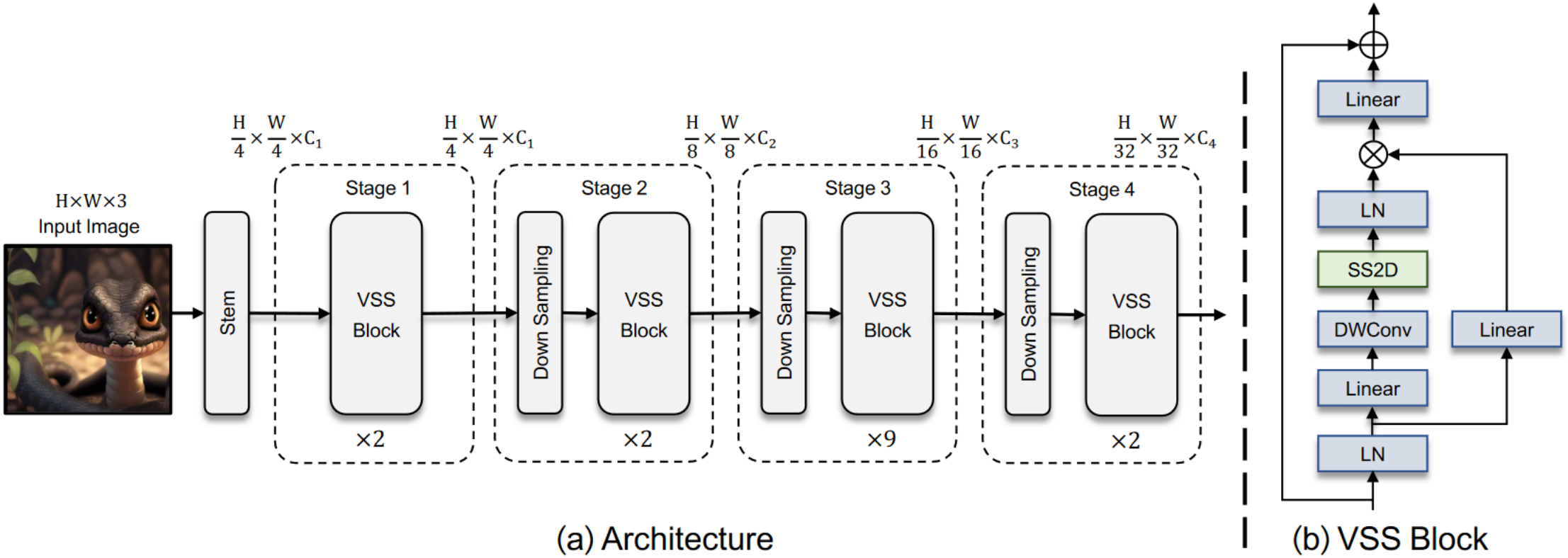


Figure 2: **Comparison of information flow: Attention vs. Cross-Scan Module (CSM)**. (a) The attention mechanism uniformly integrates all pixels for the center pixel, resulting in $\mathcal{O}(N^2)$ complexity. (b) CSM integrates pixels from top-left, bottom-right, top-right, and bottom-left with $\mathcal{O}(N)$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

1. Vmamba: Visual State Space Model



1. Vmamba: Visual State Space Model

- performance: image classification

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 [45]	224^2	22M	4.6G	79.8
DeiT-B [45]	224^2	86M	17.5G	81.8
DeiT-B [45]	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 [35]	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.

- performance: object detection

Backbone	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}	#param.	FLOPs
Mask R-CNN 1× schedule								
ResNet-50	38.2	58.8	41.4	34.7	55.7	37.2	44M	260G
Swin-T	42.7	65.2	46.8	39.3	62.2	42.2	48M	267G
ConvNeXt-T	44.2	66.6	48.3	40.1	63.3	42.8	48M	262G
PVTv2-B2	45.3	67.1	49.6	41.2	64.2	44.4	45M	309G
ViT-Adapter-S	44.7	65.8	48.3	39.9	62.5	42.8	48M	403G
VMamba-T	46.5	68.5	50.7	42.1	65.5	45.3	42M	262G
ResNet-101	38.2	58.8	41.4	34.7	55.7	37.2	63M	336G
Swin-S	44.8	66.6	48.9	40.9	63.2	44.2	69M	354G
ConvNeXt-S	45.4	67.9	50.0	41.8	65.2	45.1	70M	348G
PVTv2-B3	47.0	68.1	51.7	42.5	65.7	45.7	65M	397G
VMamba-S	48.2	69.7	52.5	43.0	66.6	46.4	64M	357G
Swin-B	46.9	-	-	42.3	-	-	107M	496G
ConvNeXt-B	47.0	69.4	51.7	42.7	66.3	46.0	108M	486G
PVTv2-B5	47.4	68.6	51.9	42.5	65.7	46.0	102M	557G
ViT-Adapter-B	47.0	68.2	51.4	41.8	65.1	44.9	102M	557G
VMamba-B	48.5	69.6	53.0	43.1	67.0	46.4	96M	482G
Mask R-CNN 3× MS schedule								
Swin-T	46.0	68.1	50.3	41.6	65.1	44.9	48M	267G
ConvNeXt-T	46.2	67.9	50.8	41.7	65.0	44.9	48M	262G
PVTv2-B2	47.8	69.7	52.6	43.1	66.8	46.7	45M	309G
ViT-Adapter-S	48.2	69.7	52.5	42.8	66.4	45.9	48M	403G
VMamba-T	48.5	69.9	52.9	43.2	66.8	46.3	42M	262G
Swin-S	48.2	69.8	52.8	43.2	67.0	46.1	69M	354G
ConvNeXt-S	47.9	70.0	52.7	42.9	66.9	46.2	70M	348G
PVTv2-B3	48.4	69.8	53.3	43.2	66.9	46.7	65M	397G
VMamba-S	49.7	70.4	54.2	44.0	67.6	47.3	64M	357G

Table 3: **Object detection and instance segmentation results on COCO dataset**. The FLOPs are calculated using inputs of size 1280 × 800. Here, AP^b and AP^m denote box AP and mask AP, respectively. "1×" indicates models fine-tuned for 12 epochs, while "3×MS" signifies the utilization of multi-scale training for 36 epochs.

2. Vision Mamba-Vim

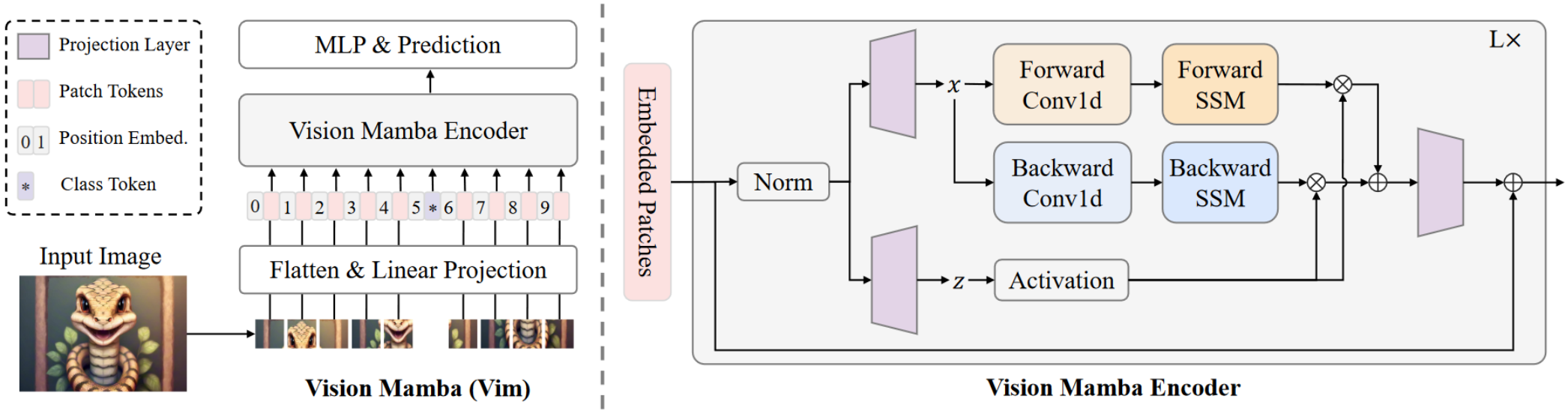


Figure 2. The overview of the proposed Vim model. We first split the input image into patches, and then project them into patch tokens. Last, we send the sequence of tokens to the proposed Vim encoder. To perform ImageNet classification, we concatenate an extra learnable classification token to the patch token sequence. Different from Mamba for text sequence modeling, Vim encoder processes the token sequence with both forward and backward directions.

2. Vision Mamba-Vim

- performance

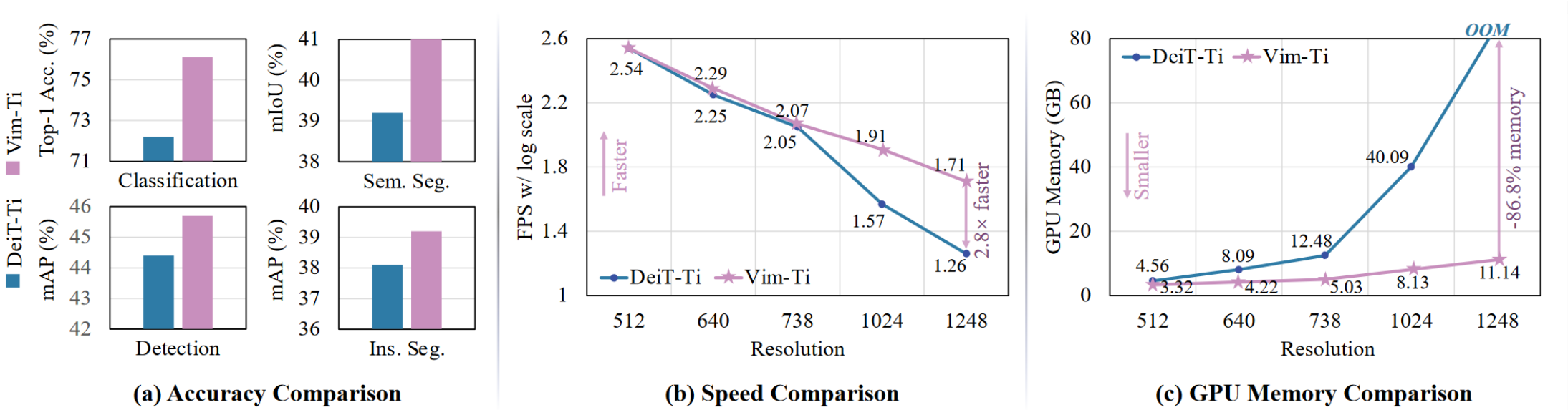


Figure 1. Performance and efficiency comparisons between DeiT [59] and our Vim model. For the accuracy comparison, we first pretrain DeiT and Vim on IN1K classification dataset [9], then we finetune the generic backbones on different downstream dense prediction tasks, *i.e.*, semantic segmentation, object detection, instance segmentation. Results show that the proposed Vim outperforms DeiT on both pretraining and finetuning tasks. Vim is also more computation and memory efficient than DeiT in dealing with high-resolution 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.

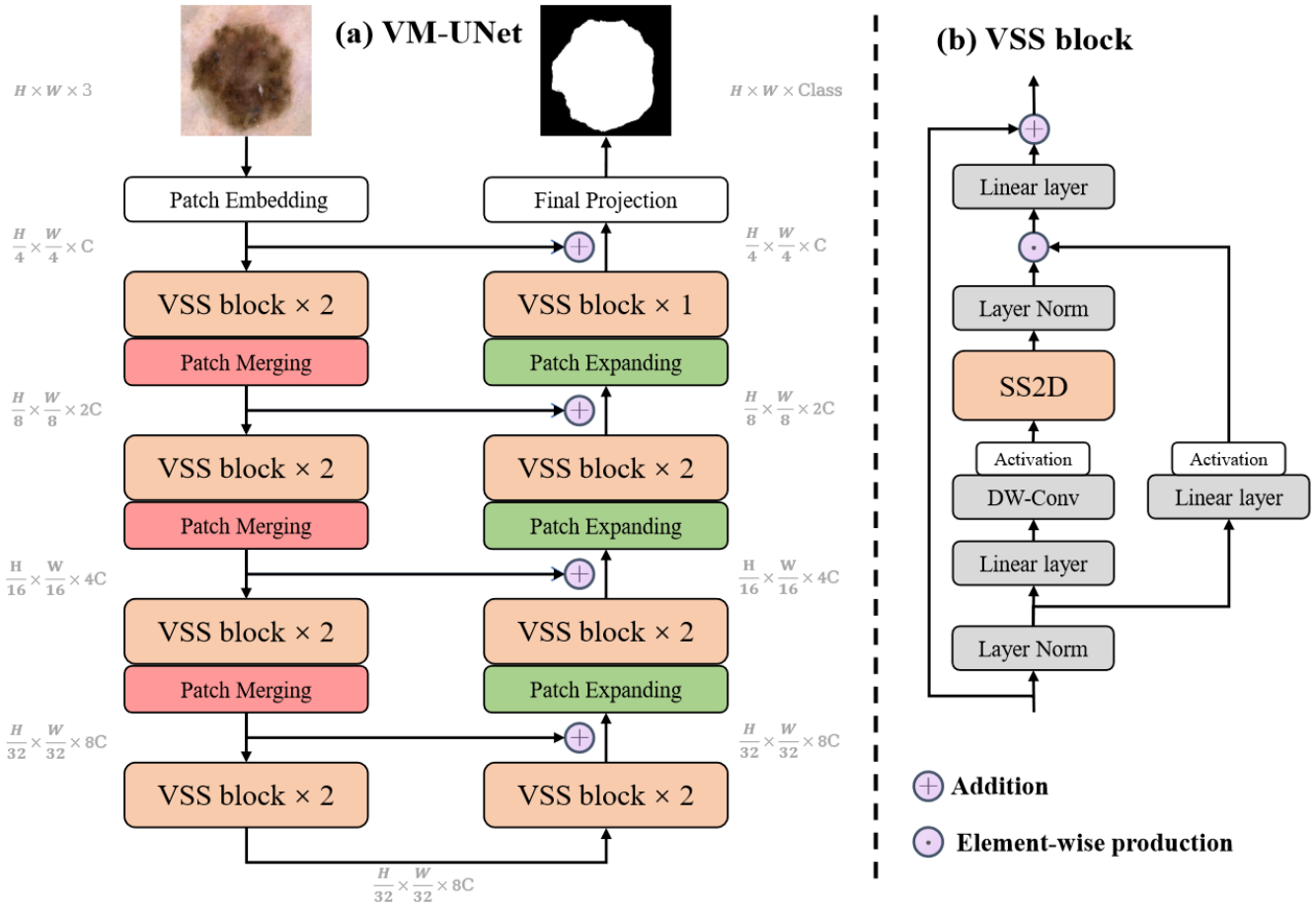
2. Vision Mamba-Vim

- performance

Method	image size	#param.	ImageNet top-1 acc.
Convnets			
ResNet-18	224^2	12M	69.8
ResNet-50	224^2	25M	76.2
ResNet-101	224^2	45M	77.4
ResNet-152	224^2	60M	78.3
ResNeXt50-32×4d	224^2	25M	77.6
RegNetY-4GF	224^2	21M	80.0
Transformers			
ViT-B/16	384^2	86M	77.9
ViT-L/16	384^2	307M	76.5
DeiT-Ti	224^2	6M	72.2
DeiT-S	224^2	22M	79.8
DeiT-B	224^2	86M	81.8
SSMs			
S4ND-ViT-B	224^2	89M	80.4
Vim-Ti	224^2	7M	76.1
Vim-Ti†	224^2	7M	78.3 +2.2
Vim-S	224^2	26M	80.5
Vim-S†	224^2	26M	81.6 +1.1

Table 1. Comparison with different backbones on ImageNet-1K validation set. † represents the model is fine-tuned with our long sequence setting.

3. VM-Unet: Vision Mamba UNet for Medical Image Segmentation



- SS2D same as in Vmamba
- Vmamba+Unet+Medical

Fig. 1. (a) The overall architecture of VM-UNet. (b) VSS block is the main construction block of VM-UNet, and SS2D is the core operation in VSS block.

3. VM-Unet: Vision Mamba UNet for Medical Image Segmentation

- performance

Table 1. Comparative experimental results on the ISIC17 and ISIC18 dataset. (**Bold** indicates the best.)

Dataset	Model	mIoU(%)↑	DSC(%)↑	Acc(%)↑	Spe(%)↑	Sen(%)↑
ISIC17	UNet [27]	76.98	86.99	95.65	97.43	86.82
	UTNetV2 [15]	77.35	87.23	95.84	98.05	84.85
	TransFuse [36]	79.21	88.40	96.17	97.98	87.14
	MALUNet [28]	78.78	88.13	96.18	**98.47**	84.78
	VM-UNet	**80.23**	**89.03**	**96.29**	97.58	**89.90**
ISIC18	UNet [27]	77.86	87.55	94.05	**96.69**	85.86
	UNet++ [37]	78.31	87.83	94.02	95.75	88.65
	Att-UNet [26]	78.43	87.91	94.13	96.23	87.60
	UTNetV2 [15]	78.97	88.25	94.32	96.48	87.60
	SANet [34]	79.52	88.59	94.39	95.97	89.46
	TransFuse [36]	80.63	89.27	94.66	95.74	**91.28**
	MALUNet [28]	80.25	89.04	94.62	96.19	89.74
	VM-UNet	**81.35**	**89.71**	**94.91**	96.13	91.12

4. PointMamba: A Simple State Space Model for Point Cloud Analysis

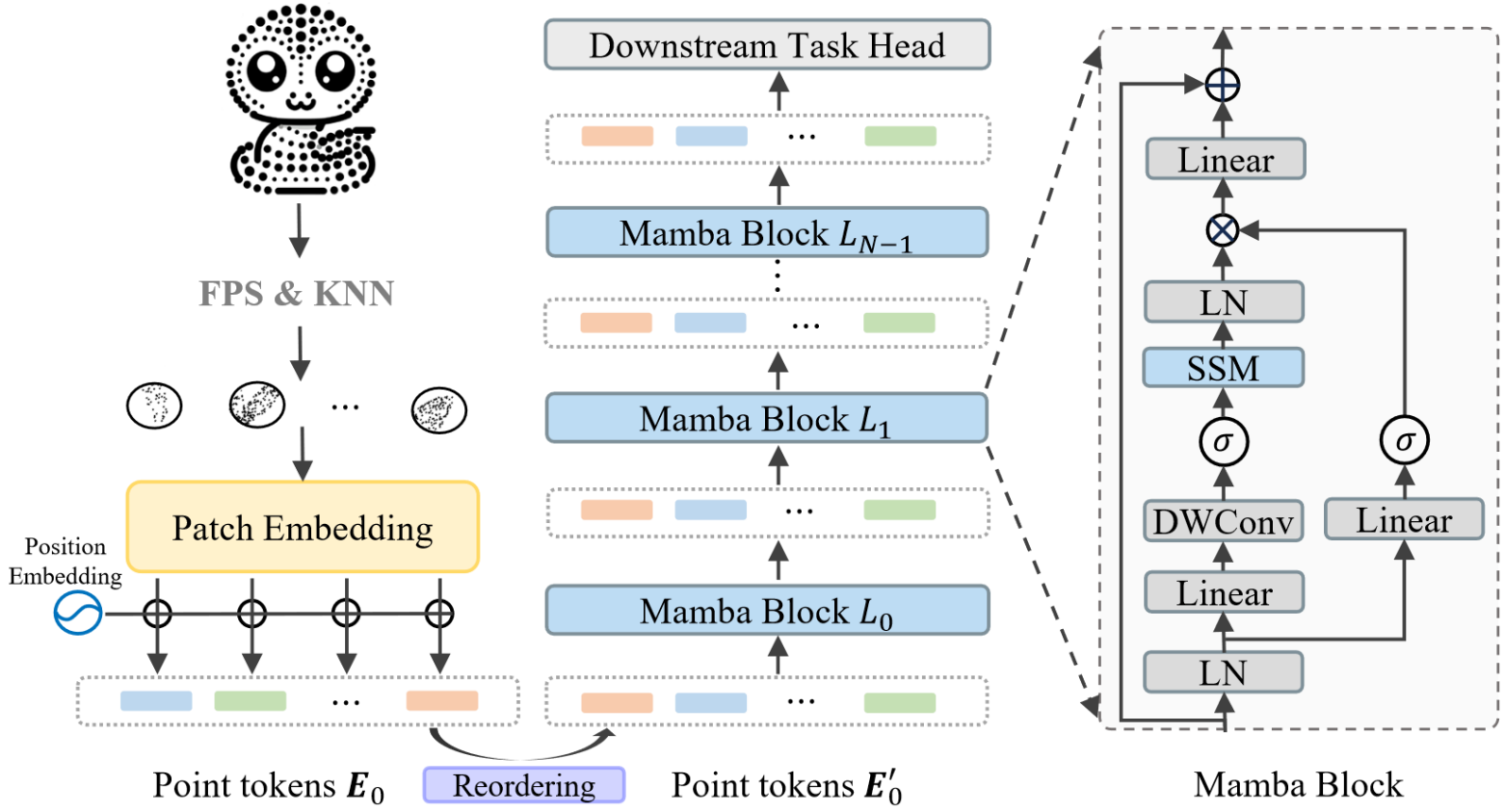


Figure 1: The pipeline of our PointMamba. We start by using FPS and KNN to obtain point patches, and then embed them into point tokens. After that, we reorder and triple the sequence. Finally, we feed the token sequence into the Mamba blocks.

4. PointMamba

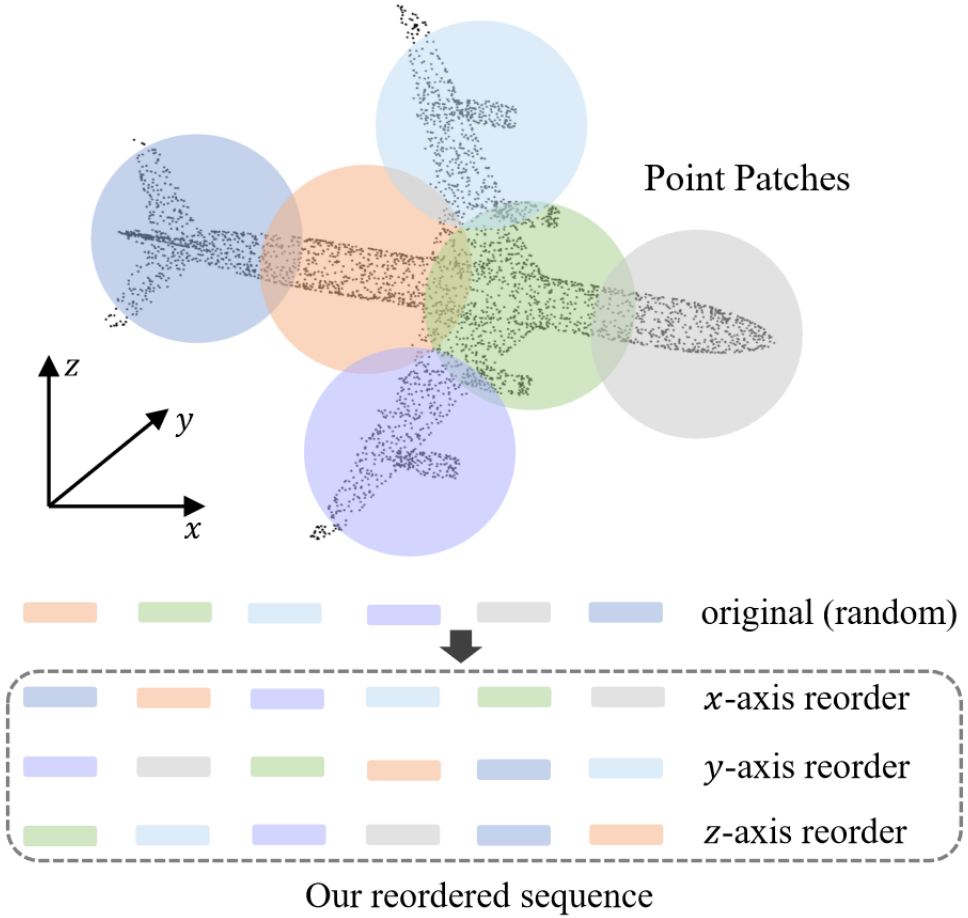


Figure 2: A simple illustration of the reordering strategy. The point patches are reordered and concatenated along the x, y, and z axes.

4. Graph-Mamba: Towards Long-Range Graph Sequence Modeling with Selective State Spaces

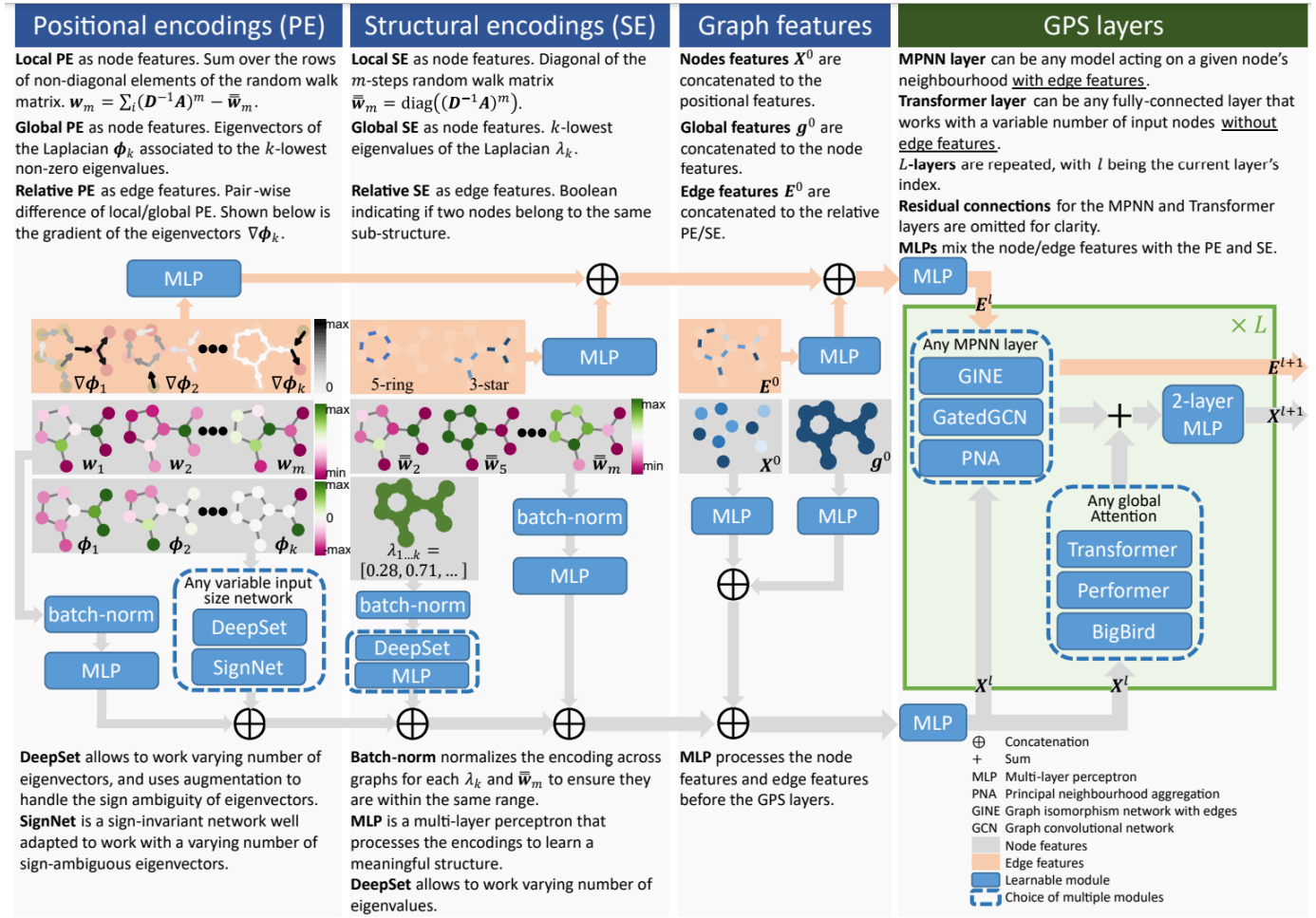
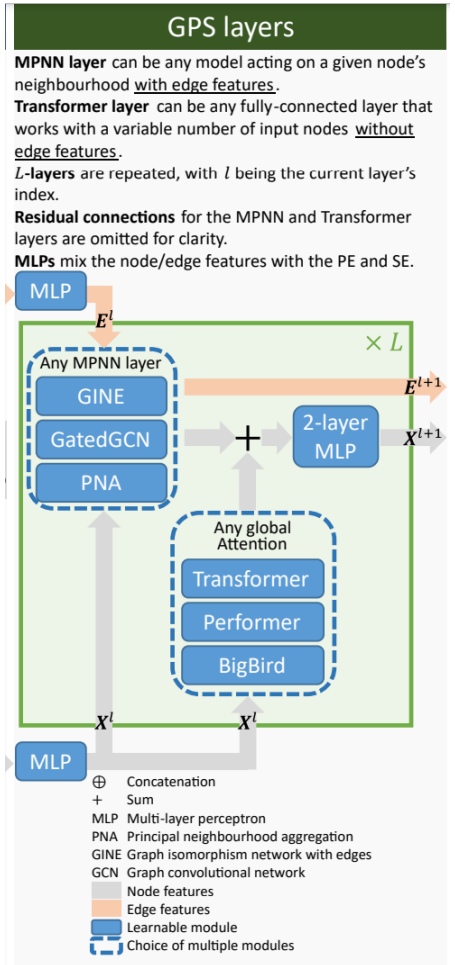


Figure 1: Modular GPS graph Transformer, with examples of PE and SE. Task specific layers for node/graph/edge-level predictions, such as pooling or output MLP, are omitted for simplicity.

- GraphGPS

4. Graph-Mamba: Towards Long-Range Graph Sequence Modeling with Selective State Spaces

- GraphGPS



- Graph-Mamba = GraphGPS (Mamba for Attention)

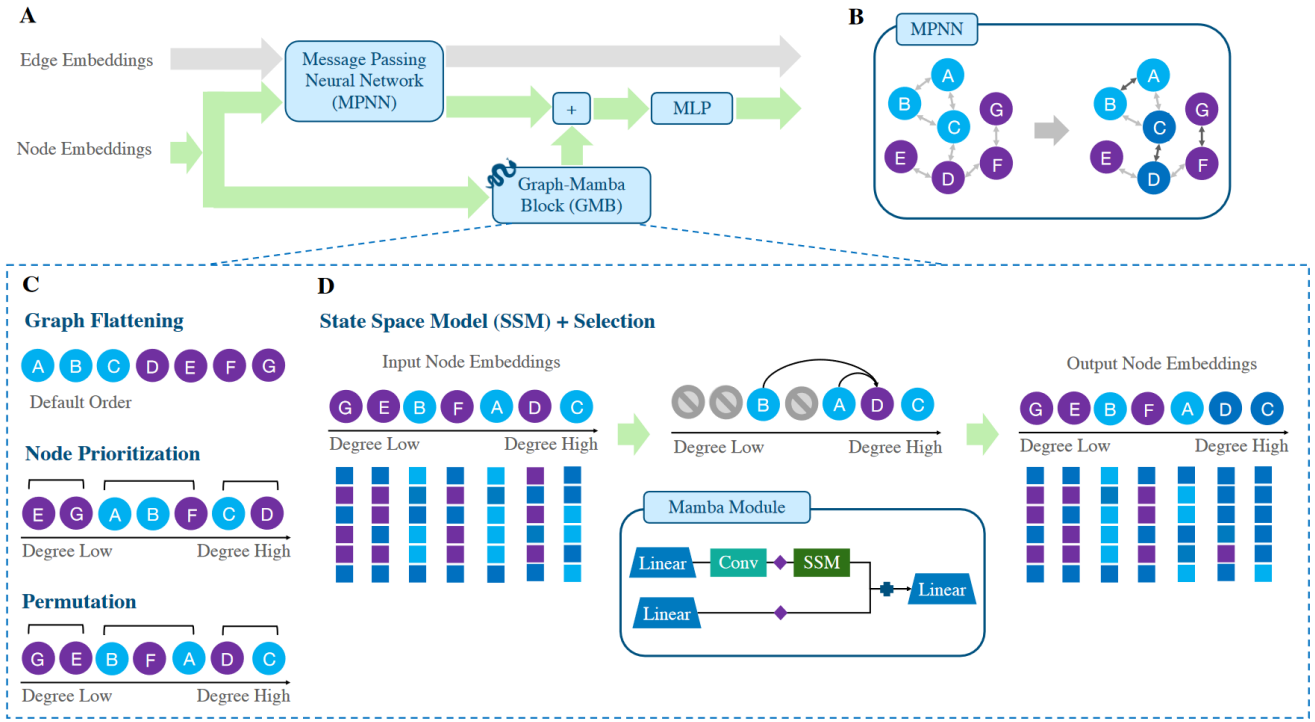


Figure 1. **Overview of Graph-Mamba architecture**, by incorporating GMB to replace the attention module in the GraphGPS framework. A) The GMB layer, an adaptation of GPS layer that combines an edge-based MPNN and a node-focused GMB to output updated node and edge embeddings. B) Graph-Mamba employs the GatedGCN model as default for MPNN. C) GMB's specialized training recipe with node prioritization and permutation techniques that perform informed graph sparsification. D) The selection mechanism with Mamba module that facilitates input-dependent context filtering.

4. Graph-Mamba: Towards Long-Range Graph Sequence Modeling with Selective State Spaces

- performance

Table 1. **Benchmark of Graph-Mamba on Long-Range Graph Datasets** with existing methods. These five datasets feature large input graphs with 150 to 1,400 nodes. Best results are colored in **first**, **second**, **third**.

Model	Peptides-Func AP ↑	Peptides-Struct MAE ↓	PascalVOC-SP F1 score ↑	COCO-SP F1 score ↑	MALNET-TINY Accuracy ↑
GCN	0.5930±0.0023	0.3496±0.0013	0.1268±0.0060	0.0841 ± 0.0010	0.8100
GIN	0.5498±0.0079	0.3547±0.0045	0.1265±0.0076	0.1339±0.0044	0.8898±0.0055
GatedGCN	0.5864±0.0077	0.3420±0.0013	0.2873±0.0219	0.2641±0.0045	0.9223±0.0065
GPS+Transformer	**0.6575±0.0049**	**0.2510±0.0015**	**0.3689±0.0131**	**0.3774±0.0150**	OOM (bs=8)
GPS+Performer	**0.6475±0.0056**	0.2558±0.0012	**0.3724±0.0131**	**0.3761±0.0101**	**0.9264±0.0078**
GPS+BigBird	0.5854±0.0079	0.2842±0.0130	0.2762±0.0069	0.2622±0.0008	0.9234±0.0034
Exphormer	0.6258±0.0092	**0.2512±0.0025**	0.3446±0.0064	0.3430±0.0108	**0.9422±0.0024**
Graph-Mamba	**0.6739±0.0087**	**0.2478±0.0016**	**0.4191±0.0126**	**0.3960±0.0175**	**0.9340± 0.0027**

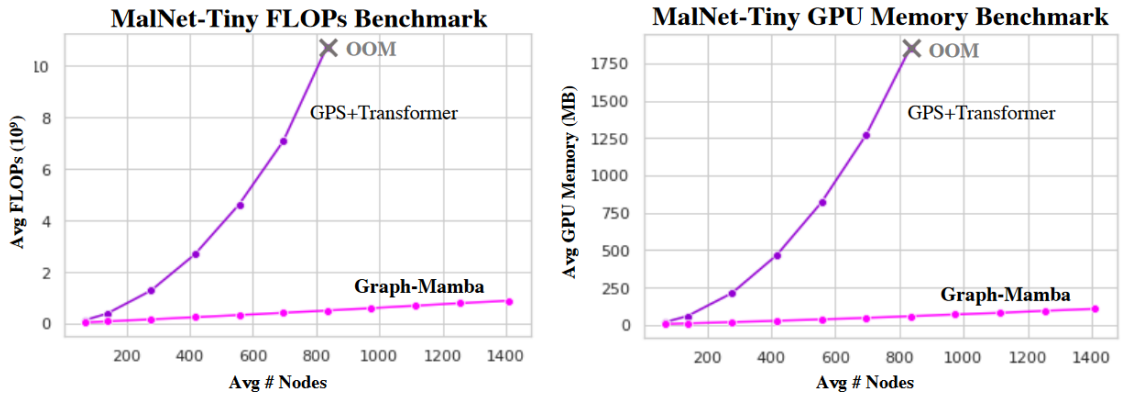


Figure 2. **FLOPs and Memory Benchmark of Graph-Mamba with GPS+Transformer** on the MalNet-Tiny dataset, subsampled at various ratios.

Griffin:
Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models

Google DeepMind 2024.3.1

Overview

model

efficiency

performance

Recurrent neural networks (RNNs) have fast inference and scale efficiently on long sequences, but they are difficult to train and hard to scale. We propose Hawk, an RNN with gated linear recurrences, and Griffin, a hybrid model that mixes gated linear recurrences with local attention. Hawk exceeds the reported performance of Mamba on downstream tasks, while Griffin matches the performance of Llama-2 despite being trained on over 6 times fewer tokens. We also show that Griffin can extrapolate on sequences significantly longer than those seen during training. Our models match the hardware efficiency of Transformers during training, and during inference they have lower latency and significantly higher throughput. We scale Griffin up to 14B parameters, and explain how to shard our models for efficient distributed training.

Architecture

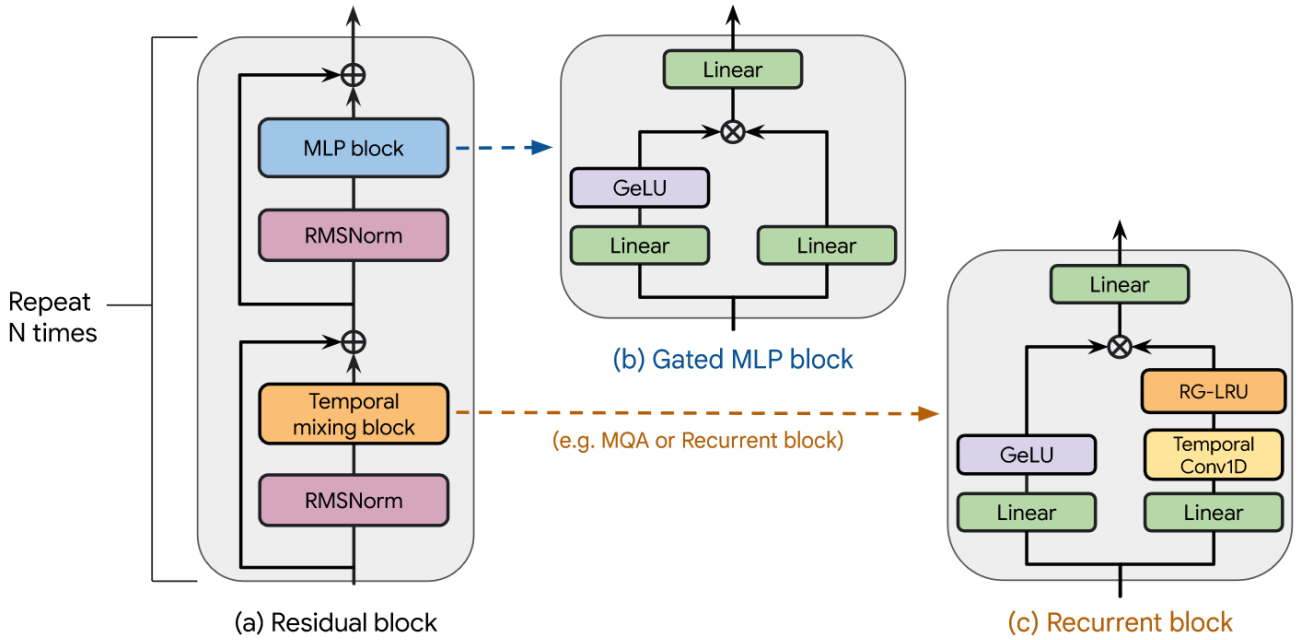


Figure 2 | a) The main backbone of our mode architecture is the residual block, which is stacked N times. b) The gated MLP block that we use. c) The recurrent block that we propose as an alternative to Multi Query Attention (MQA). It uses our proposed RG-LRU layer, defined in Section 2.4.

- Real-gated Linear Recurrent Unit (RG-LRU)

$$r_t = \sigma(W_a x_t + b_a), \quad \textit{recurrence gate} \tag{1}$$
$$i_t = \sigma(W_x x_t + b_x), \quad \textit{input gate} \tag{2}$$
$$a_t = a^{cr_t}, \tag{3}$$
$$h_t = a_t \odot h_{t-1} + \sqrt{1-a_t^2} \odot (i_t \odot x_t). \tag{4}$$

- a_t is diagonal, parameterize as $a = \sigma(\Lambda)$

$$\log a_t = \log a^{cr_t} = \log \sigma(\Lambda)^{cr_t} = -c\,\text{softplus}(\Lambda) \odot r_t.$$

Architecture

Hawk The Hawk architecture uses the same residual pattern and MLP block as our Transformer baseline, but we use the recurrent block introduced in Section 2.3 with a RG-LRU layer (see Section 2.4) as our temporal mixing block, instead of MQA. We expand the width of the recurrent block by a factor of approximately $\frac{4}{3}$ (i.e. $D_{RNN} \approx 4D/3$) in order to roughly match the parameter count of a MHA block when both use the same model dimension D.[2] See Appendix C for precise hyper-parameters.

Griffin The key advantage of recurrent blocks over global attention is that they use a fixed state size to summarize the sequence, whereas the size of MQA's KV cache grows proportional to sequence length. Since local attention (Section 2.3) has the same property, mixing recurrent blocks with local attention preserves this benefit. We have found this combination extremely effective, since local attention accurately models the recent past, while the recurrent layers can transmit information across long sequences.

Griffin uses the same residual pattern and MLP block as our Transformer baseline. However unlike both our MQA Transformer baseline and the Hawk model, Griffin uses a mixture of recurrent blocks and MQA blocks. Specifically, we employ a layered structure by alternating two residual blocks with a recurrent block followed by one residual block which uses the local (MQA) attention block described in Section 2.3. Unless otherwise stated, the local attention window size is fixed to 1024 tokens.

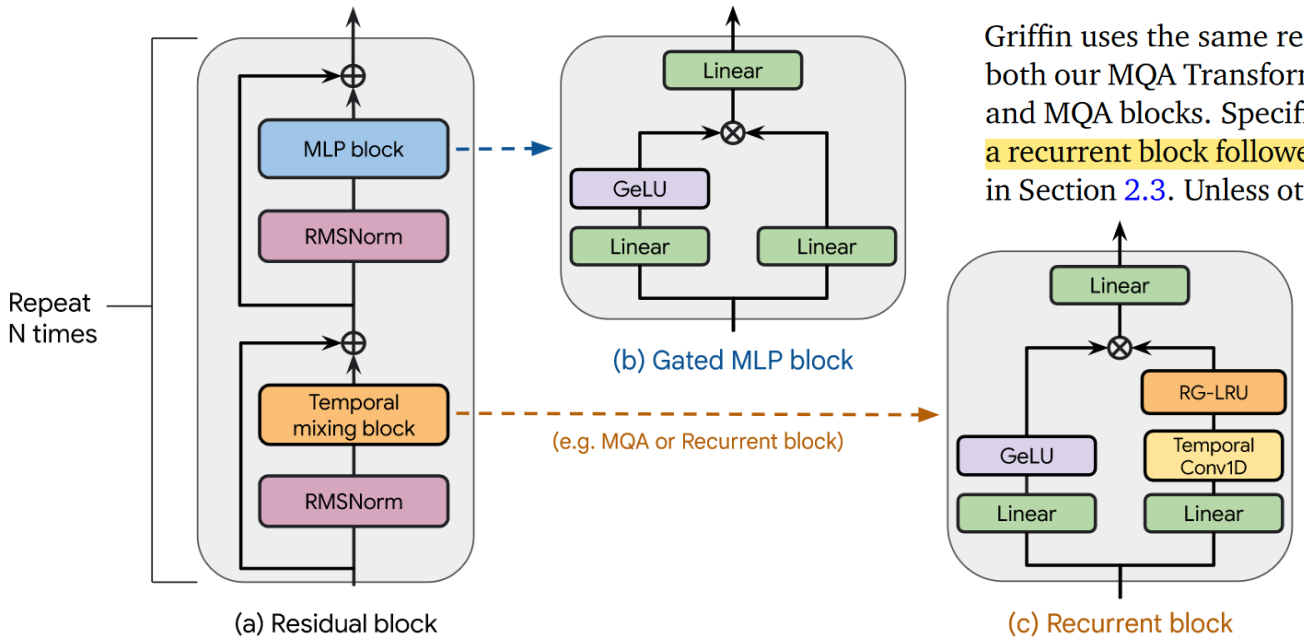


Figure 2 | a) The main backbone of our mode architecture is the residual block, which is stacked N times. b) The gated MLP block that we use. c) The recurrent block that we propose as an alternative to Multi Query Attention (MQA). It uses our proposed RG-LRU layer, defined in Section 2.4.

Performance

Table 1 | Character normalized accuracy. Hawk is competitive with our Transformer baseline, and exceeds the reported performance of Mamba despite being trained on half as many tokens. Griffin outperforms our Transformer baseline, and matches the performance of Llama-2 despite being trained on roughly 7 times fewer tokens. We report unnormalized accuracy with partial scoring for WinoGrande.

Model Type	Model Size	Training Tokens	MMLU	HellaSwag	PIQA	WinoGrande	ARC-E	ARC-C	Average
Mamba	3B	600B	26.2	71.0	78.1	65.9	68.2	41.7	58.5
Llama-2	7B	2T	45.3	77.2	78.8	69.2	75.2	45.9	65.3
	13B	2T	**54.8**	80.7	80.5	72.8	77.3	49.4	69.3
MQA Transformer (Baseline)	1B	300B	28.9	64.8	75.0	62.0	60.2	35.4	54.4
	3B	300B	31.7	71.0	77.6	66.1	68.1	39.2	59.0
	6B	300B	38.9	77.0	79.5	70.4	74.1	45.2	64.2
Hawk	1B	300B	29.7	63.3	76.1	57.2	60.6	34.6	53.6
	3B	300B	31.3	71.7	78.8	66.5	68.4	40.2	59.5
	7B	300B	35.0	77.6	80.0	69.9	74.4	45.9	63.8
Griffin	1B	300B	29.5	67.2	77.4	65.2	67.0	36.9	57.2
	3B	300B	32.6	73.5	78.1	67.2	71.5	41.4	60.7
	7B	300B	39.3	78.6	81.0	72.6	75.4	47.9	65.8
	14B	300B	49.5	**81.4**	**81.8**	**74.1**	**79.1**	**50.8**	**69.5**

Efficiency

- training duration

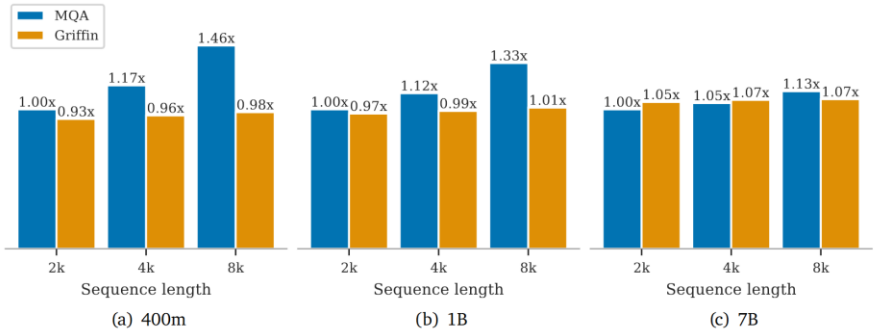


Figure 3 | Training durations per step computed relative to our MQA baseline at 2K sequence length as we vary the model size and sequence length for Griffin and MQA. Let us note that as we increase the sequence length we lower the batch size proportionally, such that the total number of tokens per batch stays fixed.

- inference latency

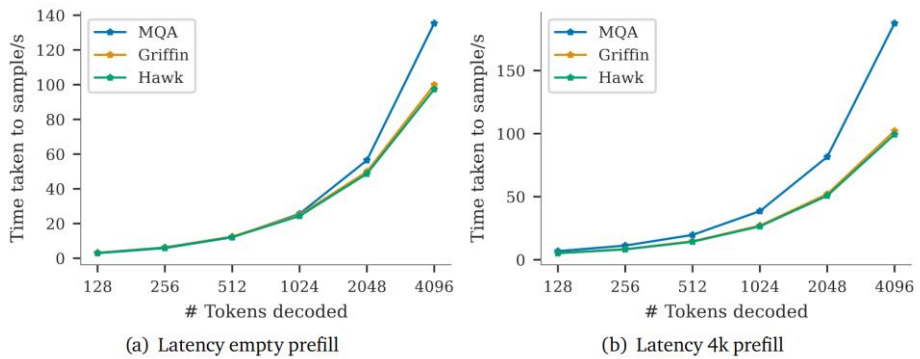


Figure 4 | Latency of different 1B parameter models for a range of sequence lengths for (a) sampling from an empty prefill and (b) sampling from a prefill of 4k tokens.