Transformers are SSMs
&
Mamba2

Papers

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu[*1] and Tri Dao[*2]

[1]Machine Learning Department, Carnegie Mellon University
[2]Department of Computer Science, Princeton University
agu@cs.cmu.edu, tri@tridao.me

Abstract

Foundation models, now powering most of the exciting applications in deep learning, are almost universally based on the Transformer architecture and its core attention module. Many subquadratic-time architectures such as linear attention, gated convolution and recurrent models, and structured state space models (SSMs) have been developed to address Transformers' computational inefficiency on long sequences, but they have not performed as well as attention on important modalities such as language. We identify that a key weakness of such models is their inability to perform content-based reasoning, and make several improvements. First, simply letting the SSM parameters be functions of the input addresses their weakness with discrete modalities, allowing the model to selectively propagate or forget information along the sequence length dimension depending on the current token. Second, even though this change prevents the use of efficient convolutions, we design a hardware-aware parallel algorithm in recurrent mode. We integrate these selective SSMs into a simplified end-to-end neural network architecture without attention or even MLP blocks (Mamba). Mamba enjoys fast inference (5× higher throughput than Transformers) and linear scaling in sequence length, and its performance improves on real data up to million-length sequences. As a general sequence model backbone, Mamba achieves state-of-the-art performance across several modalities such as language, audio, and genomics. On language modeling, our Mamba-3B model outperforms Transformers of the same size and matches Transformers twice its size, both in pretraining and downstream evaluation.

1 Introduction

Foundation models (FMs), or large models pretrained on massive data then adapted for downstream tasks, have emerged as an effective paradigm in modern machine learning. The backbone of these FMs are often sequence models, operating on arbitrary sequences of inputs from a wide variety of domains such as language, images, speech, audio, time series, and genomics (Brown et al. 2020; Dosovitskiy et al. 2020; Ismail Fawaz et al. 2020; Oord et al. 2016; Poli et al. 2023; Sutskever, Vinyals, and Quoc V Le 2014). While this concept is agnostic to a particular choice of model architecture, modern FMs are predominantly based on a single type of sequence model: the Transformer (Vaswani et al. 2017) and its core attention layer (Bahdanau, Cho, and Bengio 2015). The efficacy of self-attention is attributed to its ability to route information densely within a context window, allowing it to model complex data. However, this property brings fundamental drawbacks: an inability to model anything outside of a finite window, and quadratic scaling with respect to the window length. An enormous body of research has appeared on more efficient variants of attention to overcome these drawbacks (Tay, Dehghani, Bahri, et al. 2022), but often at the expense of the very properties that makes it effective. As of yet, none of these variants have been shown to be empirically effective at scale across domains.

Recently, structured state space sequence models (SSMs) (Gu, Goel, and Ré 2022; Gu, Johnson, Goel, et al. 2021) have emerged as a promising class of architectures for sequence modeling. These models can be interpreted as a combination of recurrent neural networks (RNNs) and convolutional neural networks (CNNs), with inspiration from classical state space models (Kalman 1960). This class of models can be computed very efficiently as either a recurrence or convolution, with linear or near-linear scaling in sequence length. Additionally, they have principled

[*]Equal contribution.

Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality

Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality

Tri Dao[*1] and Albert Gu[*2]

[1]Department of Computer Science, Princeton University
[2]Machine Learning Department, Carnegie Mellon University
tri@tridao.me, agu@cs.cmu.edu

Abstract

While Transformers have been the main architecture behind deep learning's success in language modeling, state-space models (SSMs) such as Mamba have recently been shown to match or outperform Transformers at small to medium scale. We show that these families of models are actually quite closely related, and develop a rich framework of theoretical connections between SSMs and variants of attention, connected through various decompositions of a well-studied class of structured *semiseparable matrices*. Our state space duality (SSD) framework allows us to design a new architecture (**Mamba-2**) whose core layer is an a refinement of Mamba's selective SSM that is 2-8× faster, while continuing to be competitive with Transformers on language modeling.

1 Introduction

Transformers, in particular decoder-only models (e.g. GPT (Brown et al. 2020), Llama (Touvron, Lavril, et al. 2023)) which process input sequences in a causal fashion, are one of the main drivers of modern deep learning's success. Numerous approaches attempt to approximate the core attention layer to address its efficiency issues (Tay et al. 2022), such as scaling quadratically in sequence length during training and requiring a cache of size linear in sequence length during autoregressive generation. In parallel, a class of alternative sequence models, structured state-space models (SSMs), have emerged with linear scaling in sequence length during training and constant state size during generation. They show strong performance on long-range tasks (*e.g.* S4 (Gu, Goel, and Ré 2022)) and recently matched or beat Transformers on language modeling (e.g. Mamba (Gu and Dao 2023)) at small to moderate scale. However, the development of SSMs have appeared disjoint from the community's collective effort to improve Transformers, such as understanding them theoretically as well as optimizing them on modern hardware. As a result, it is more difficult to understand and experiment with SSMs compared to Transformers, and it remains challenging to train SSMs as efficiently as Transformers from both an algorithmic and systems perspective.

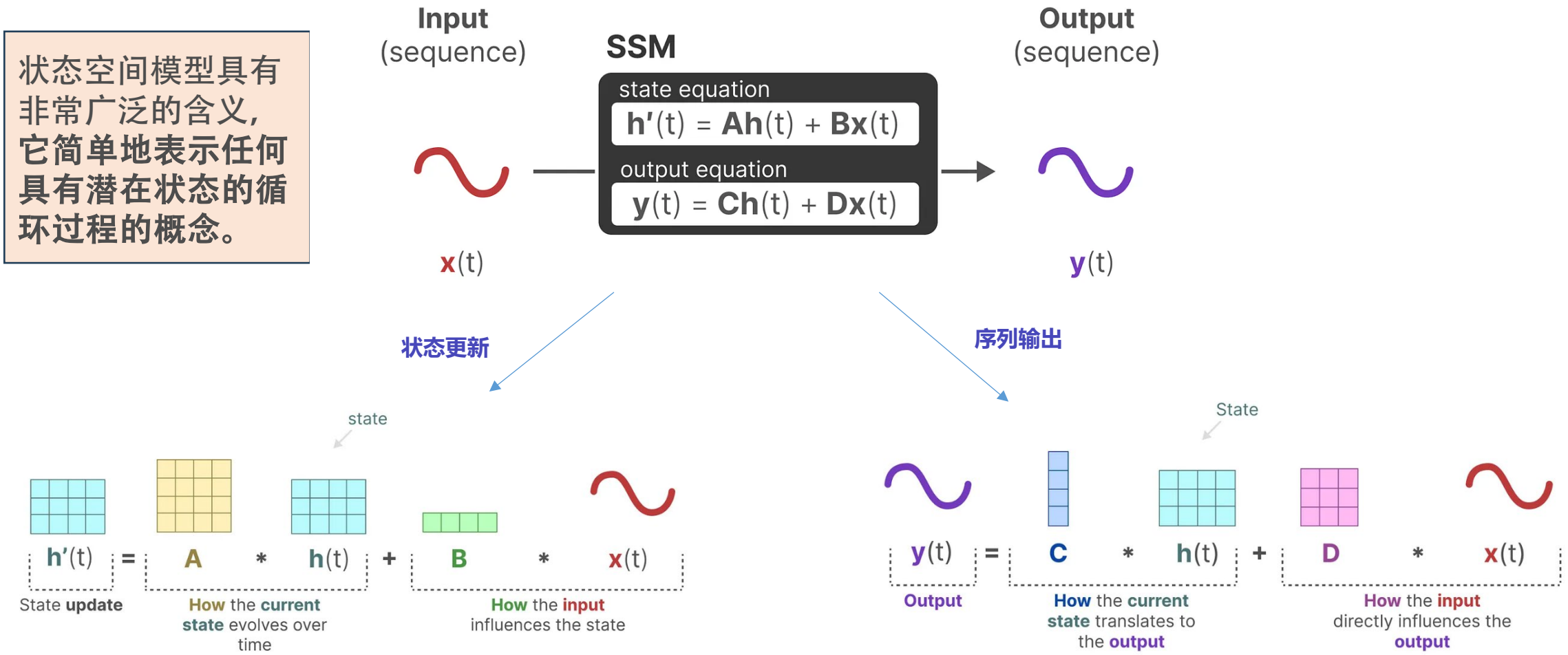
Our main goal is to develop a rich body of theoretical connections between structured SSMs and variants of attention. This will allow us to transfer algorithmic and systems optimizations originally developed for Transformers to SSMs, towards the goal of building foundation models that perform better than Transformers while scaling more efficiently in sequence length. A milestone contribution in this direction was the **Linear Attention (LA)** framework (Katharopoulos et al. 2020), which derived a connection between autoregressive attention and linear RNNs by showing the equivalence between "dual forms" of quadratic kernelized attention and a particular linear recurrence. This duality allows new capabilities such as the ability to have both efficient parallelizable training and efficient autoregressive inference. In the same spirit, this paper provides multiple viewpoints connecting linear-complexity SSMs with quadratic-complexity forms to combine the strengths of SSMs and attention.[1]

[*]Alphabetical by last name.
[1]Technically speaking, these connections only relate to certain flavors of attention; the title of this paper is an homage to Katharopoulos et al. (2020) which first showed that "Transformers are RNNs".

SSM & Mamba review

SSM Review

状态空间模型具有非常广泛的含义，**它简单地表示任何具有潜在状态的循环过程的概念。**



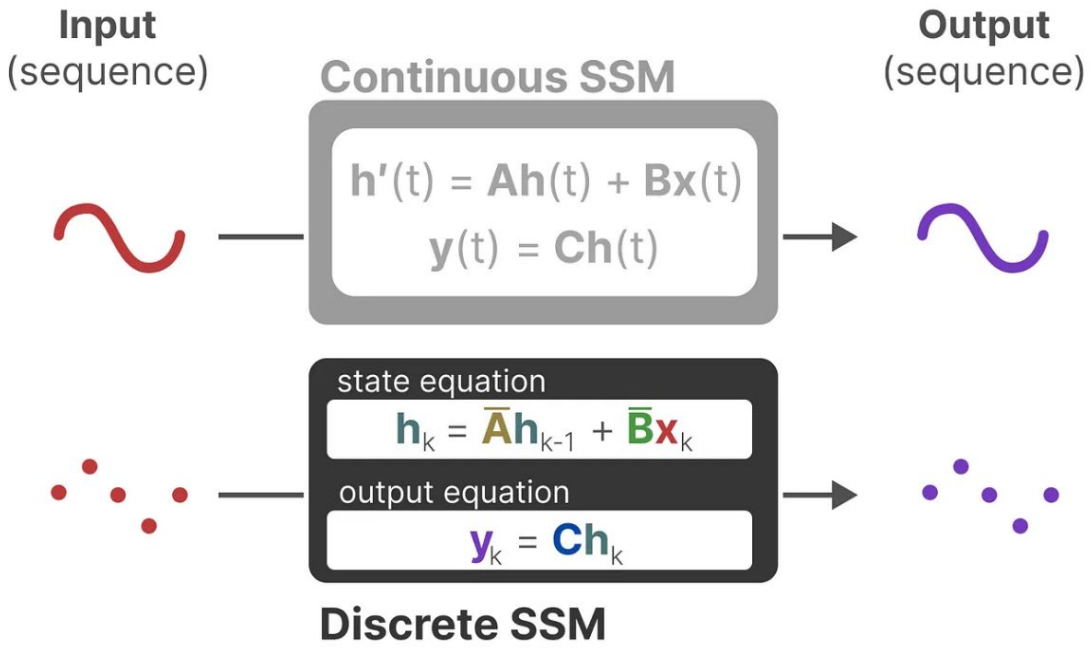
SSM Review

Discrete-time SSM: The Recurrent Representation

$$h'(t) = Ah(t) + Bx(t) \quad (1a)$$
$$y(t) = Ch(t) \quad (1b)$$

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t \quad (2a)$$
$$y_t = Ch_t \quad (2b)$$

$$\overline{K} = (C\overline{B}, C\overline{AB}, \dots, C\overline{A}^k\overline{B}, \dots) \quad (3a)$$
$$y = x * \overline{K} \quad (3b)$$



经过离散化处理后，我们将原来函数到函数的映射关系转换为序列到序列的映射关系，由此可以对离散化的数据进行训练与推理

此外，经过离散化的状态方程变为关于Xk的递归式，从而允许了离散SSM像RNN一样递归计算

SSM Review

S4 Structure : Structure State Space Model

$$h'(t) = Ah(t) + Bx(t) \quad (1a)$$
$$y(t) = Ch(t) \quad (1b)$$

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t \quad (2a)$$
$$y_t = Ch_t \quad (2b)$$

$$\overline{K} = (C\overline{B}, C\overline{AB}, \dots, C\overline{A}^k\overline{B}, \dots) \quad (3a)$$
$$y = x * \overline{K} \quad (3b)$$

S6 Structure : Selective State Space Model

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

Algorithm 1 SSM (S4)

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

Algorithm 2 SSM + Selection (S6)

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

Mamba Review

在Mamaba中，作者让 B 矩阵、C 矩阵、Δ 成为输入的函数，让模型能够根据输入内容自适应地调整其行为

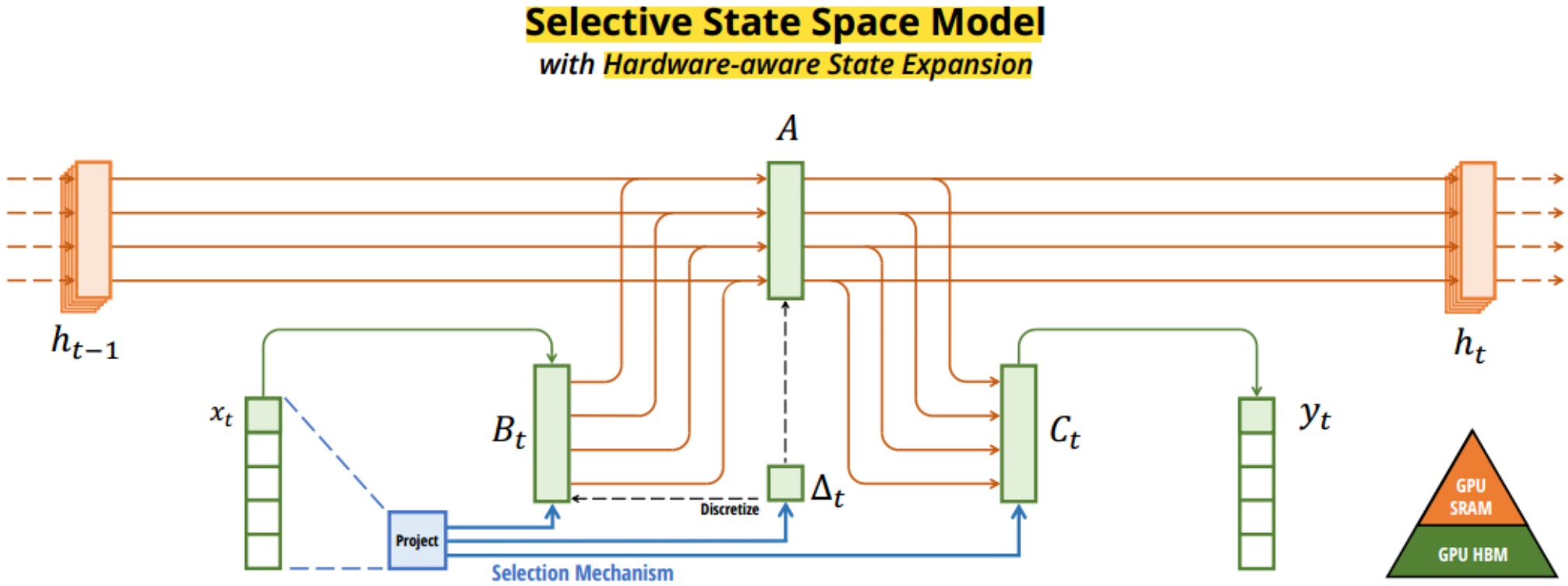


Figure 1: (**Overview.**) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate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

SSM Review

Algorithm 2 SSM + Selection (S6)

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

```
"""
u: r(B D L)
delta: r(B D L)
A: c(D N) or r(D N)
B: c(D N) or r(B N L) or r(B N 2L) or r(B G N L) or (B G N L)
C: c(D N) or r(B N L) or r(B N 2L) or r(B G N L) or (B G N L)
D: r(D)
z: r(B D L)
delta_bias: r(D), fp32

out: r(B D L)
last_state (optional): r(B D dstate) or c(B D dstate)
"""
```

$$x_k = \overline{A}x_{k-1} + \overline{B}u_k \qquad h_t = A_t h_{t-1} + B_t x_t$$
$$y_k = \overline{C}x_k + \overline{D}u_k \qquad y_t = C_t^{\mathsf{T}} h_t$$

矩阵A捕获有关先前状态的信息来构建新状态（Hippo）

HiPPO Matrix $\mathbf{A}_{nk}$

- $(2\boldsymbol{n}+1)^{1/2}(2\boldsymbol{k}+1)^{1/2}$ ← everything **below** the diagonal
- $\boldsymbol{n}+1$ ← the diagonal
- 0 ← everything **above** the diagonal

CSDN @zyw2002

Discretized matrix **A**

$$\overline{\mathbf{A}} = \exp(\Delta\mathbf{A})$$

Discretized matrix **B**

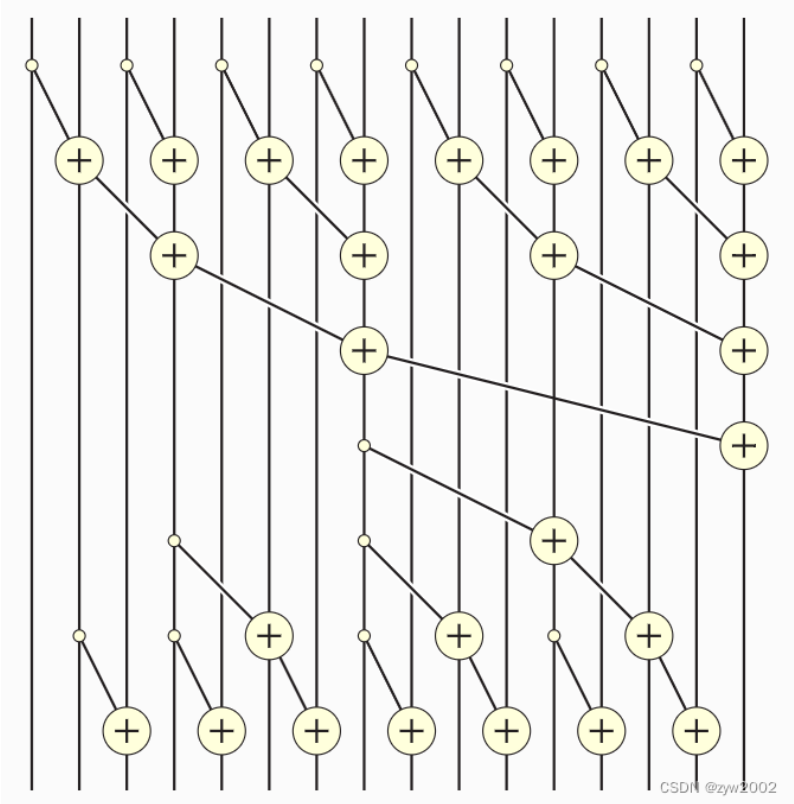
$$\overline{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(\exp(\Delta\mathbf{A}) - I)\cdot\Delta\mathbf{B}$$

CSDN @zyw2002

Mamba Review

并行扫描(parallel scan)

我们可以分段计算序列并迭代地组合它们,即动态矩阵B和C以及并行扫描算法,一起创建选择性扫描算法



E.g.

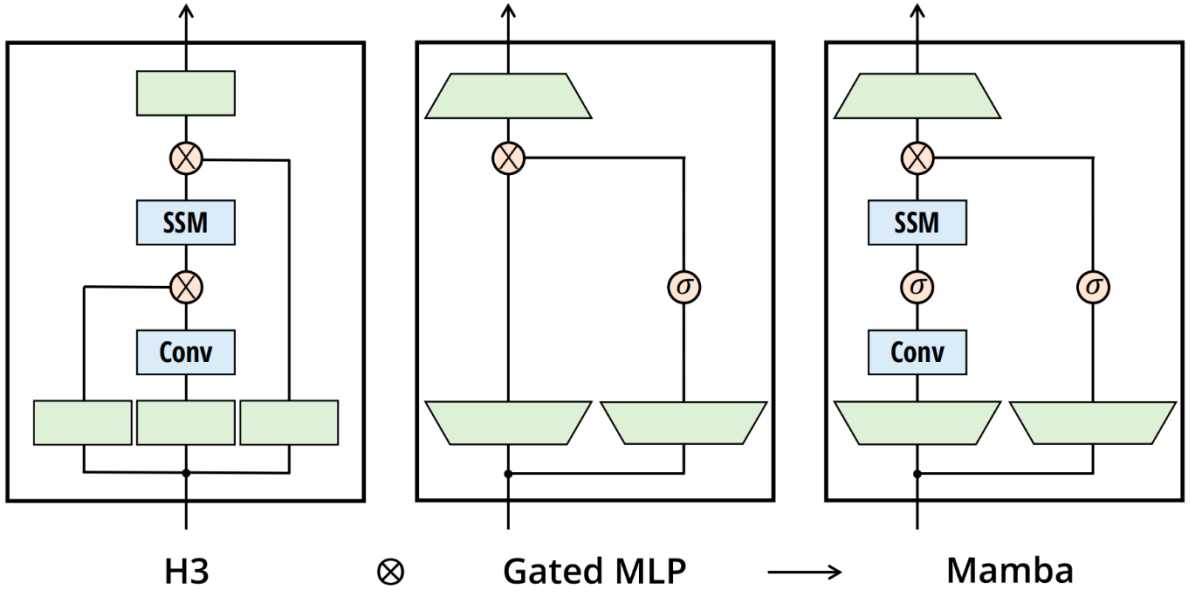
```
procedure down-sweep(A)
  a[n − 1] ← 0                                  % Set the identity
  for d from (lg n) − 1 downto 0
    in parallel for i from 0 to n − 1 by 2^(d+1)
      t ← a[i + 2^d − 1]                        % Save in temporary
      a[i + 2^d − 1] ← a[i + 2^(d+1) − 1]       % Set left child
      a[i + 2^(d+1) − 1] ← t + a[i + 2^(d+1) − 1]   % Set right child
```

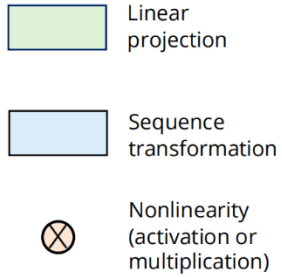
	Step	Vector in Memory						
	0	[**3**	**1**	**7**	**0**	**4**	**1**	**6**	**3**]
up	1	[3	**4**	7	**7**	4	**5**	6	**9**]
	2	[3	4	7	**11**	4	5	6	**14**]
	3	[3	4	7	11	4	5	6	**25**]
clear	4	[3	4	7	11	4	5	6	**0**]
down	5	[3	4	7	**0**	4	5	6	**11**]
	6	[3	**0**	7	**4**	4	**11**	6	**16**]
	7	[**0**	**3**	**4**	**11**	**11**	**15**	**16**	**22**]

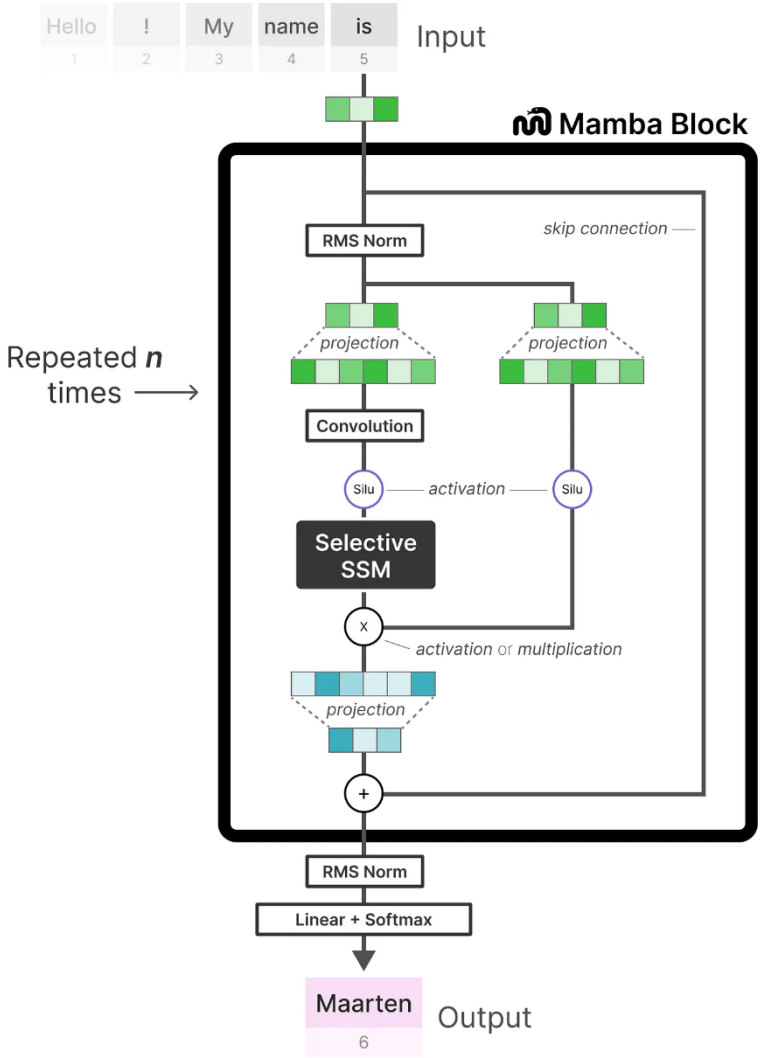
(b) Executing a +-prescan on a PRAM.

Mamba Review

深度框架







将大多数SSM架构比如H_3的基础块，与现代神经网络比如transformer中普遍存在的门控MLP相结合，组成新的Mamba块，重复这个块，与归一化和残差连接结合，便构成了Mamba架构

Transformers are SSMs
&
Mamba2

Transformers Are SSMs

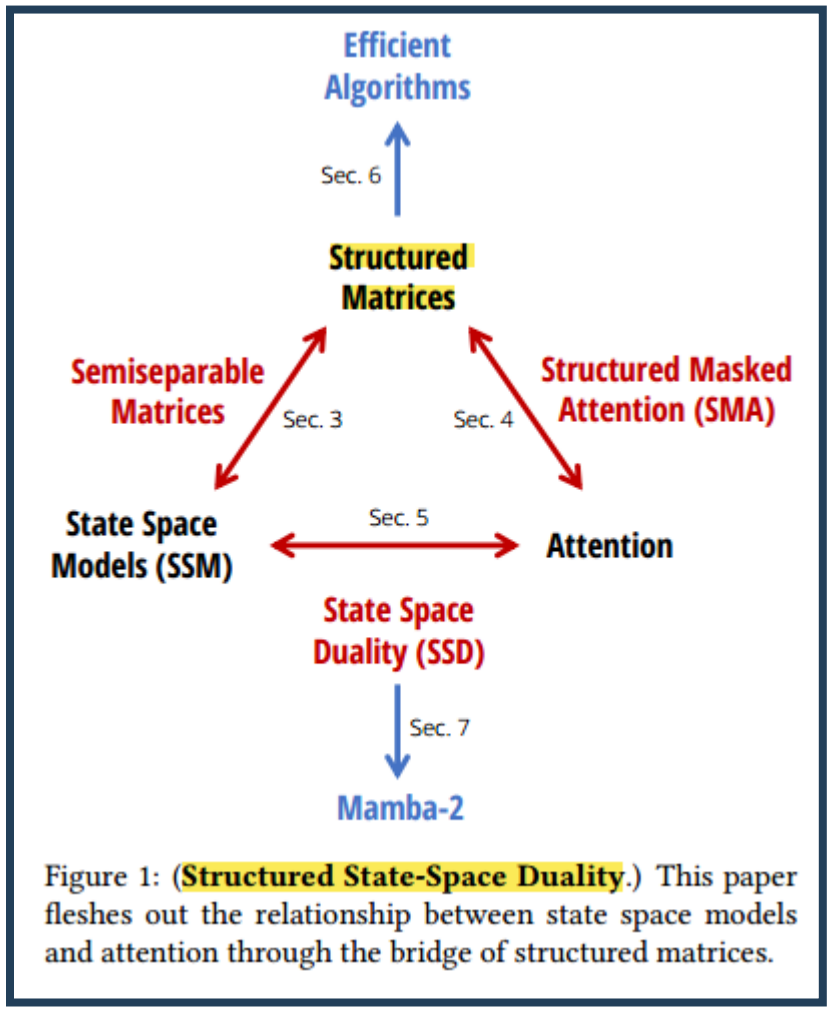


Figure 1: (**Structured State-Space Duality**.) This paper fleshes out the relationship between state space models and attention through the bridge of structured matrices.

Work List

- 状态空间模型的矩阵式 SSS
- 对注意力的重构 SMA
- 状态空间对偶理论 SSD
- Mamba-2 version

状态空间模型的矩阵式 SSS

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

(Selective state space model (SSM))

$$h_t = A_t \dots A_1 B_0 x_0 + A_t \dots A_2 B_1 x_1 + \cdots + A_t A_{t-1} B_{t-2} x_{t-2} + A_t B_{t-1} x_{t-1} + B_t x_t$$
$$= \sum_{s=0}^{t} A_{t:s}^{\times} B_s x_s.$$

$$y_t = \sum_{s=0}^{t} C_t^\top A_{t:s}^{\times} B_s x_s$$

$$y = \mathrm{SSM}(A, B, C)(x) = Mx$$

$$M_{ji} := C_j^\top A_j \cdots A_{i+1} B_i$$

$M_{ji} := C_j^\top A_j \cdots A_{i+1} B_i$ ⟶ 半可分矩阵 semiseparate matrix

Definition 3.2. *A lower triangular matrix* $M \in \mathbb{R}^{(\mathsf{T},\mathsf{T})}$ *has a* N-***sequentially semiseparable (SSS)*** *representation if it can be written in the form*

$$M_{ji} = C_j^\top A_j \cdots A_{i+1} B_i \tag{4}$$

for vectors $B_0, \dots, B_{\mathsf{T}-1}, C_0, \dots, C_{\mathsf{T}-1} \in \mathbb{R}^{\mathsf{N}}$ *and matrices* $A_0, \dots, A_{\mathsf{T}-1} \in \mathbb{R}^{(\mathsf{N},\mathsf{N})}$.

We define the operator SSS *so that* $M = \mathrm{SSS}(A_{0:\mathsf{T}}, B_{0:\mathsf{T}}, C_{0:\mathsf{T}})$.

状态空间模型的矩阵式 SSS

矩阵分解 **Proposition 3.4.** *Every* N*-semiseparable matrix has a* N*-SSS representation.*

Lemma 3.3. *An* N*-SSS matrix* M *with representation* (4) *is* N*-semiseparable.*

Proof. Consider any off-diagonal block $M_{j:j',i':i}$ where $j' > j \geq i > i'$. This has an explicit rank-N factorization as

$$\begin{bmatrix} C_j^\top A^\times_{j:i'} B_{i'} & \cdots & C_j^\top A^\times_{j:i-1} B_{i-1} \\ \vdots & & \vdots \\ C_{j'-1}^\top A^\times_{j'-1:i'} B_{i'} & \cdots & C_{j'-1}^\top A^\times_{j'-1:i-1} B_{i-1} \end{bmatrix} = \begin{bmatrix} C_j^\top A^\times_{j:j} \\ \vdots \\ C_{j'-1}^\top A^\times_{j'-1:j} \end{bmatrix} A^\times_{j:i-1} \begin{bmatrix} A^\times_{i-1:i'} B_{i'} & \cdots & A^\times_{i-1:i-1} B_{i-1} \end{bmatrix}. \tag{5}$$

$$M_{ji} = C_j^\top A_j \cdots A_{i+1} B_i \qquad M = \mathrm{SSS}(A_{0:\mathrm{T}}, B_{0:\mathrm{T}}, C_{0:\mathrm{T}}).$$

Naïve Form?

□

1-Semiseparable Matrices: the Scalar SSM Recurrence 标量SSM递归式

$$\mathrm{SSS}(a,b,c) = \mathrm{diag}(c) \cdot M \cdot \mathrm{diag}(b) \qquad \text{where} \qquad M_{ji} = a^\times_{j:i}.$$

退化的SSM

$$\begin{aligned} y_t &= a_{t:0} x_0 + \cdots + a_{t:t} x_t \\ &= a_t \left(a_{t-1:0} x_0 + \cdots + a_{t-1:t-1} x_{t-1} \right) + a_{t:t} x_t \\ &= a_t y_{t-1} + x_t. \end{aligned}$$

$$M = 1\mathrm{SS}(a_{0:T}) := \begin{bmatrix} 1 & & & & \\ a_1 & 1 & & & \\ a_2 a_1 & a_2 & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ a_{T-1} \ldots a_1 & a_{T-1} \ldots a_2 & \cdots & a_{T-1} & 1 \end{bmatrix}.$$

状态空间模型的矩阵式 SSS

State Space Models are Semiseparable Matrices　状态空间模型本质是半可分矩阵

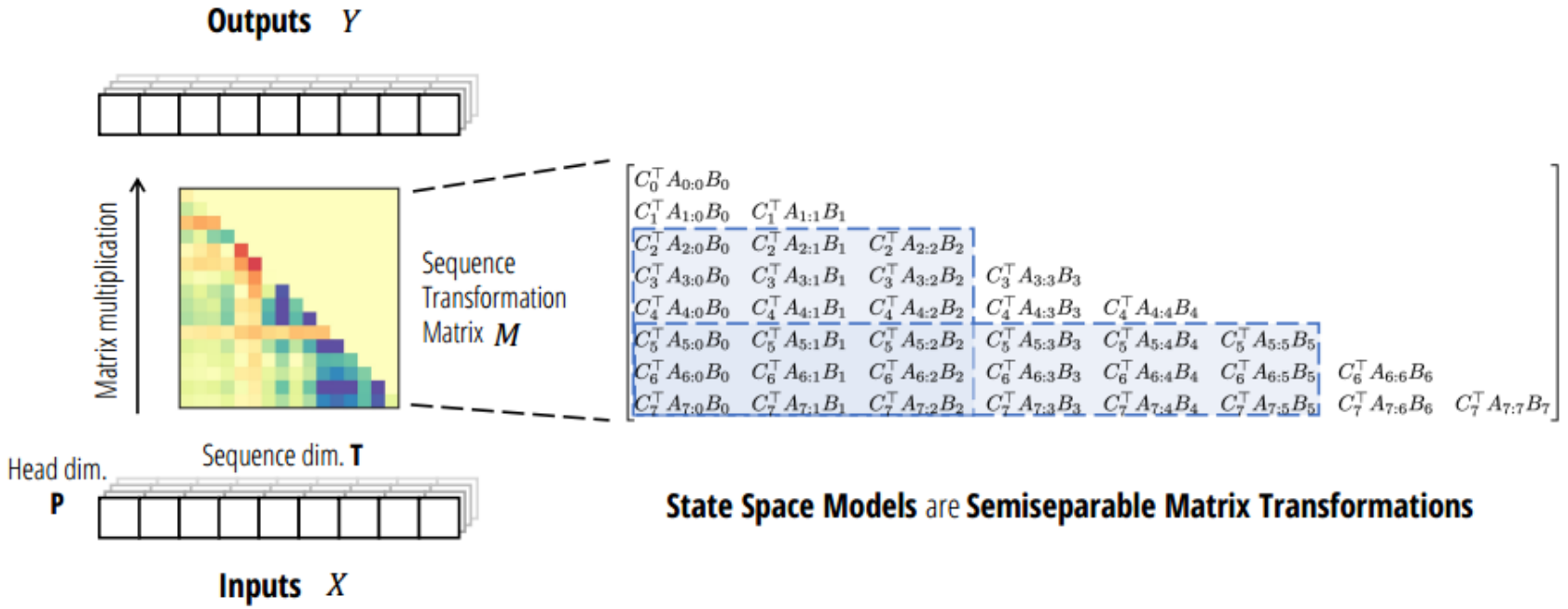


Figure 2: (**State Space Models are Semiseparable Matrices**.) As sequence transformations, state space models can be represented as a matrix transformation $M \in \mathbb{R}^{(\mathsf{T},\mathsf{T})}$ acting on the sequence dimension T, sharing the same matrix for each channel in a head (*Left*). This matrix is a semiseparable matrix (*Right*), which is a rank-structured matrix where every submatrix contained on-and-below the diagonal (*Blue*) has rank at most N, equal to the SSM's state dimension.

Structured SSMs as Sequence Transformations.

Definition 2.1. *We use the term* ***sequence transformation*** *to refer to a parameterized map on sequences* $Y = f_\theta(X)$ *where* $X, Y \in \mathbb{R}^{(\mathsf{T},\mathsf{P})}$ *and* θ *is an arbitrary collection of parameters.* T *represents the sequence or* time *axis; subscripts index into the first dimension, e.g.* $X_t, Y_t \in \mathbb{R}^{\mathsf{P}}$.

对注意力的重构 SMA

The basic form of (single-head) attention is a map on three sequences of vectors $(Q, K, V) \mapsto Y$.

$$\begin{aligned} Q &= \text{input} && (\mathsf{T}, \mathsf{N}) \\ K &= \text{input} && (\mathsf{S}, \mathsf{N}) \\ V &= \text{input} && (\mathsf{S}, \mathsf{P}) \\ G &= QK^{\mathsf{T}} && (\mathsf{T}, \mathsf{S}) \\ M &= f(G) && (\mathsf{T}, \mathsf{S}) \\ Y &= GV && (\mathsf{T}, \mathsf{P}) \end{aligned} \tag{9}$$

Masked attention is usually written in matrix notation as

$$y = (L \circ (QK^{\mathsf{T}})) \cdot V. \tag{10}$$

More precisely, with shape annotations and breaking this down into the precise sequence of computations:

$$\begin{aligned} G &= QK^{\mathsf{T}} && (\mathsf{T}, \mathsf{S}) \\ M &= G \circ L && (\mathsf{T}, \mathsf{S}) \\ Y &= MV && (\mathsf{T}, \mathsf{P}) \end{aligned} \tag{11}$$

对注意力的重构 SMA

旁引：清华大学黄高老师团队——Mamba 一种线性注意力机制视角

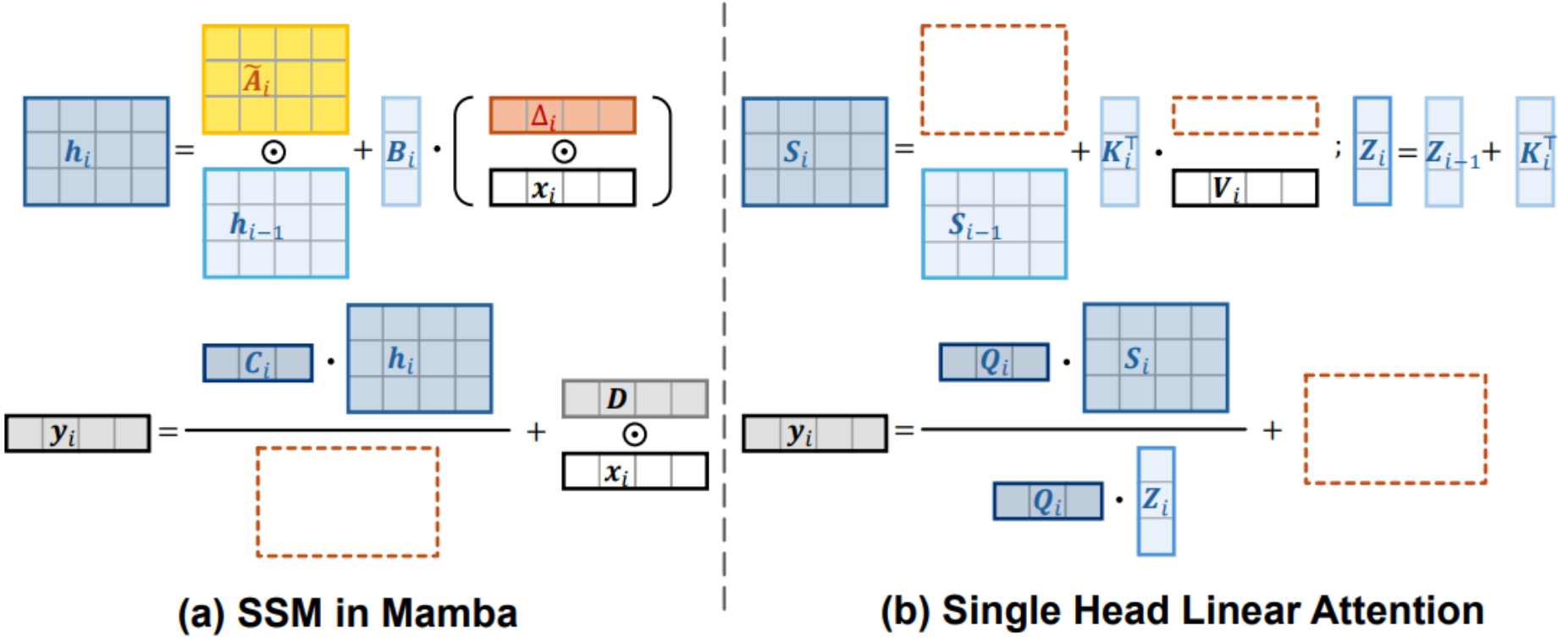
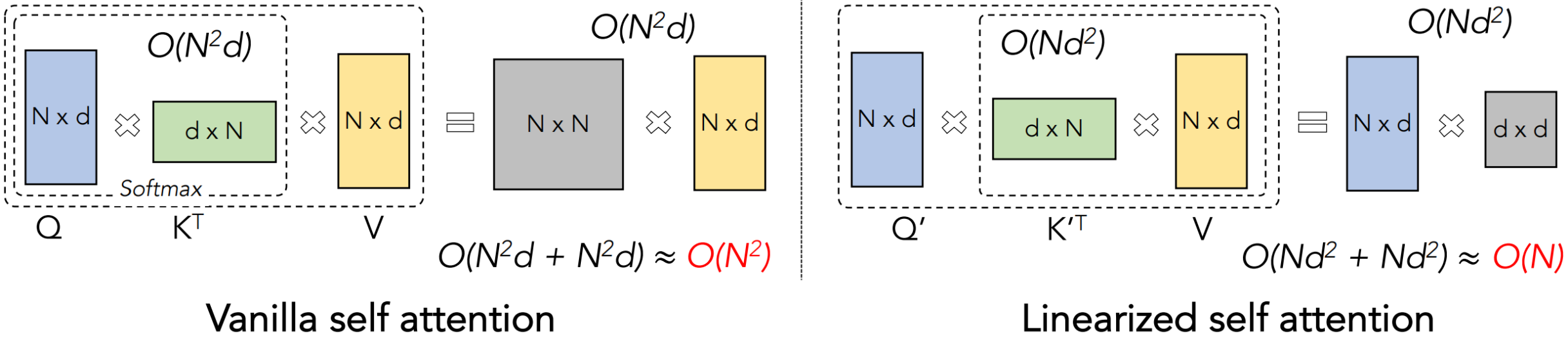


Figure 1: Illustration of selective SSM in Mamba (eq. (11)) and single head linear attention (eq. (12)). It can be seen that selective SSM resembles single-head linear attention with additional input gate $\boldsymbol{\Delta}_i$, forget gate $\widetilde{\boldsymbol{A}}_i$ and shortcut $\boldsymbol{D} \odot \boldsymbol{x}_i$, while omitting normalization $\boldsymbol{Q}_i \boldsymbol{Z}_i$.

对注意力的重构 SMA



- 两篇论文
  - Transformers are RNNs: Fast Autoregressive Transformers with **Linear Attention**
  - Transformers are SSMs: Generalized Models and Efficient Algorithms Through **Structured State Space Duality**

对注意力的重构 SMA

Let $\boldsymbol{x} \in \mathbb{R}^{N \times C}$ denote a sequence of N features with dimension C. Single head **Softmax attention** [34], also known as dot-product attention, can be written as:

$$\boldsymbol{Q}=\boldsymbol{x}\mathbf{W}_Q, \boldsymbol{K}=\boldsymbol{x}\mathbf{W}_K, \boldsymbol{V}=\boldsymbol{x}\mathbf{W}_V, \quad \boldsymbol{y}_i=\sum_{j=1}^{N} \frac{\exp\left(\boldsymbol{Q}_i \boldsymbol{K}_j^{\top}/\sqrt{d}\right)}{\sum_{j=1}^{N} \exp\left(\boldsymbol{Q}_i \boldsymbol{K}_j^{\top}/\sqrt{d}\right)} \boldsymbol{V}_j, \tag{1}$$

where $\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{C \times d}, \mathbf{W}_V \in \mathbb{R}^{C \times C}$ denote projection matrices, $\boldsymbol{Q}, \boldsymbol{K} \in \mathbb{R}^{N \times d}, \boldsymbol{V} \in \mathbb{R}^{N \times C}$ represent query/key/value matrices, and $\boldsymbol{Q}_i, \boldsymbol{K}_i \in \mathbb{R}^{1 \times d}, \boldsymbol{V}_i \in \mathbb{R}^{1 \times C}$ are individual query/key/value tokens. Softmax attention computes the similarities between each query-key pair, leading to $\mathcal{O}(N^2)$ complexity. Therefore, it incurs unbearable computational cost in long-sequence modeling scenarios.

Linear attention [20], another attention paradigm, is proposed to effectively address this problem by reducing the computation complexity to $\mathcal{O}(N)$. Specifically, linear attention replaces the non-linear Softmax function with linear normalization, and adopts an additional kernel function ϕ in Q and K:

$$\boldsymbol{Q}=\phi(\boldsymbol{x}\mathbf{W}_Q), \boldsymbol{K}=\phi(\boldsymbol{x}\mathbf{W}_K), \boldsymbol{V}=\boldsymbol{x}\mathbf{W}_V, \quad \boldsymbol{y}_i=\sum_{j=1}^{N} \frac{\boldsymbol{Q}_i \boldsymbol{K}_j^{\top}}{\sum_{j=1}^{N} \boldsymbol{Q}_i \boldsymbol{K}_j^{\top}} \boldsymbol{V}_j = \frac{\boldsymbol{Q}_i \left(\sum_{j=1}^{N} \boldsymbol{K}_j^{\top} \boldsymbol{V}_j\right)}{\boldsymbol{Q}_i \left(\sum_{j=1}^{N} \boldsymbol{K}_j^{\top}\right)}. \tag{2}$$

This enables the rearrangement of the computation order from $(\boldsymbol{Q}\boldsymbol{K}^{\top})\boldsymbol{V}$ to $\boldsymbol{Q}(\boldsymbol{K}^{\top}\boldsymbol{V})$ based on the associative property of matrix multiplication, thus reducing computation complexity to $\mathcal{O}(N)$.

线性注意
用线性归一化代替非线性的Softmax函数，并在Q和K中增加一个核函数φ

对注意力的重构 SMA

重新设计掩模注意力 Structured masked attention ----- 从**张量收缩**的视角

Definition 4.2. ***Structured masked attention (SMA)*** *(or* ***structured attention*** *for short) is defined as a* function *on queries/keys/values* Q, K, V *as well as any* structured matrix L *(i.e. has sub-quadratic matrix multiplication), through the 4-way tensor contraction*

$$Y = \text{contract}(\text{TN}, \text{SN}, \text{SP}, \text{TS} \to \text{TP})(Q, K, V, L).$$

Naïve mask attention 二次形式是实现完整矩阵的朴素矩阵乘法算法

$$G = \text{contract}(\text{TN}, \text{SN} \to \text{TS})(Q, K) \qquad (\text{T}, \text{S})$$

$$M = \text{contract}(\text{TS}, \text{TS} \to \text{TS})(G, L) \qquad (\text{T}, \text{S})$$

$$Y = \text{contract}(\text{TS}, \text{SP} \to \text{TP})(M, V) \qquad (\text{T}, \text{P})$$

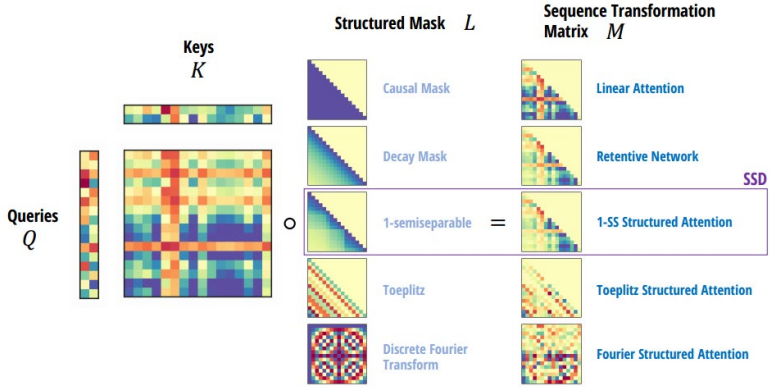
Linear attention 线性形式是一种结构化的矩阵乘法算法

$$Z = \text{contract}(\text{SP}, \text{SN} \to \text{SPN})(V, K) \qquad (\text{S}, \text{P}, \text{N})$$

$$H = \text{contract}(\text{TS}, \text{SPN} \to \text{TPN})(L, Z) \qquad (\text{T}, \text{P}, \text{N})$$

$$Y = \text{contract}(\text{TN}, \text{TPN} \to \text{TP})(Q, H) \qquad (\text{T}, \text{P})$$

Notation	Description
T	**Time** axis or **target** sequence axis
S	**Source** sequence axis (in attention)
D	Model **dimension** or d_model
N	State/feature dimension or d_state
P	Head dimension or d_head
H	Number of **heads** or n_head



对注意力的重构 SMA

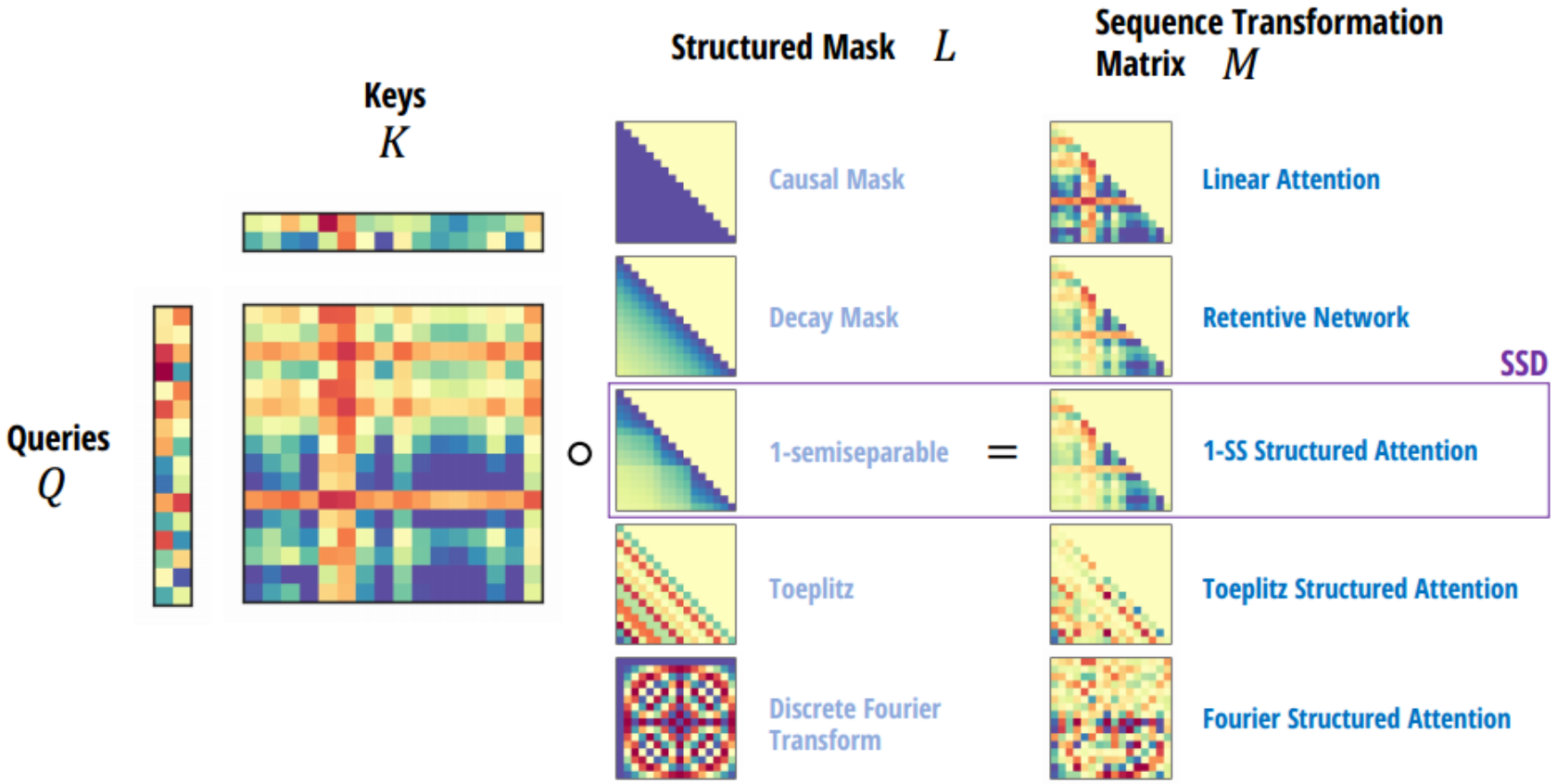


Figure 3: (**Structured Masked Attention**.) SMA constructs a masked attention matrix $M = QK^\top \circ L$ for any structured matrix L, which defines a matrix sequence transformation $Y = MV$. All instances of SMA have a dual subquadratic form induced by a different contraction ordering, combined with the efficient structured matrix multiplication by L. Previous examples include Linear Attention (Katharopoulos et al. 2020) and RetNet (Y. Sun et al. 2023). Beyond SSD (1-semiseparable SMA), the focus of this paper, many other potential instantiations of structured attention are possible.

状态空间对偶理论 SSD

由上文知，二次注意力和线性注意力是掩膜注意力的一对对偶关系

4.3.1 Summary: The Dual Forms of Masked Attention

Standard (masked kernel) attention is often conflated between a function and an algorithm. Separating this distinction presents a clear way to understand different variants of attention.

- We view **masked attention** as a particular *function* (12).
- The standard **quadratic attention** computation (13) can be viewed as an *algorithm* to compute the function.
- **Linear attention** (15) is an alternate algorithm to compute the same function.

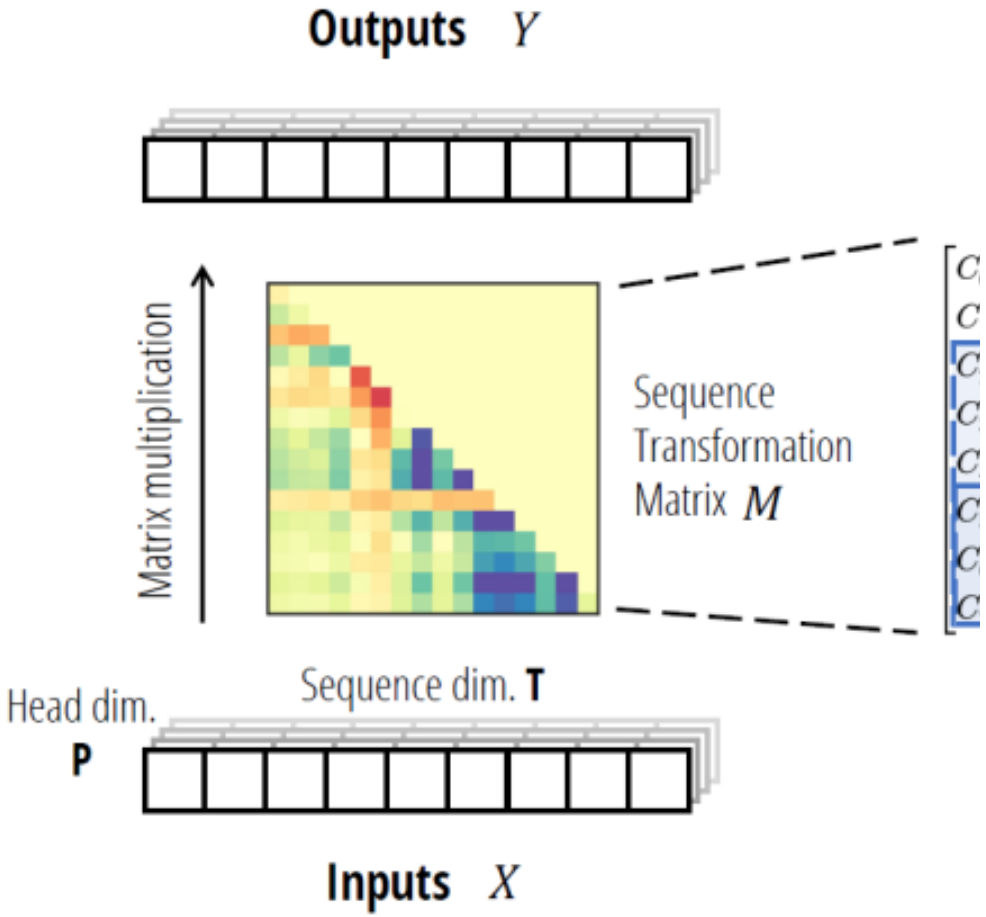
Moreover, in this case

跳转页面位置:14

- The masked attention function is simply a particular *contraction on four terms*.
- The quadratic and linear attention algorithms are simply *two different orders to perform the contractions*.

- 结构化状态空间模型的特定情况与结构化注意力的特定情况是一致的，并且线性时间SSM算法和二次时间核注意力算法是彼此的对偶形式。
- SSD模型的对偶性可以看作是半可分离矩阵上的两种不同的矩阵乘法算法。这时其实SSM就是一个半可分矩阵的转换。

状态空间对偶理论 SSD



Lemma 3.3. *An* N*-SSS matrix* M *with representation* (4) *is* N*-semiseparable.*

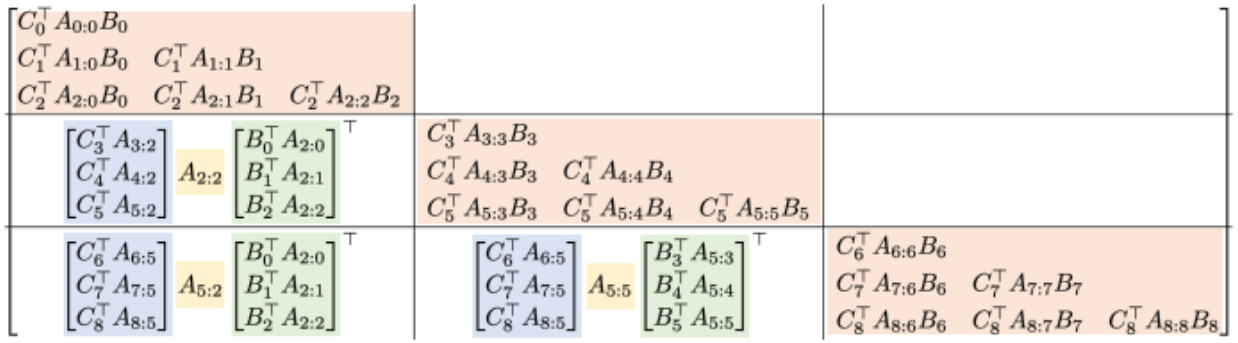
Proof. Consider any off-diagonal block $M_{j:j',i':i}$ where $j' > j \geq i > i'$. This has an explicit rank-N factorization as

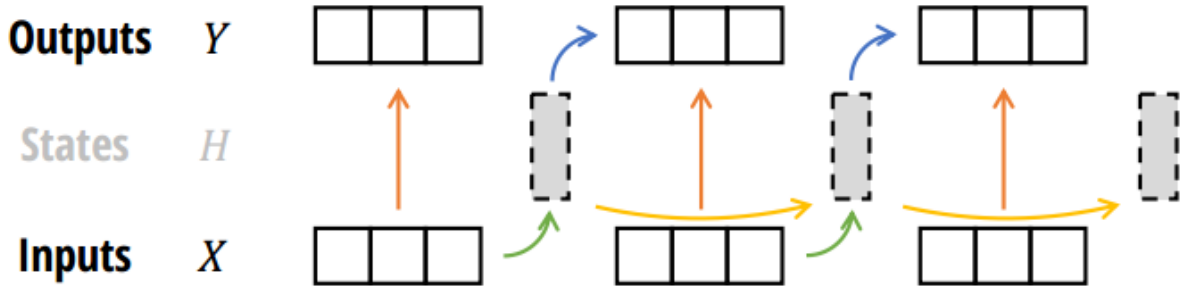
$$\begin{bmatrix} C_j^\top A^\times_{j:i'} B_{i'} & \cdots & C_j^\top A^\times_{j:i-1} B_{i-1} \\ \vdots & & \vdots \\ C_{j'-1}^\top A^\times_{j'-1:i'} B_{i'} & \cdots & C_{j'-1}^\top A^\times_{j'-1:i-1} B_{i-1} \end{bmatrix} = \begin{bmatrix} C_j^\top A^\times_{j:j} \\ \vdots \\ C_{j'-1}^\top A^\times_{j'-1:j} \end{bmatrix} A^\times_{j:i-1} \begin{bmatrix} A^\times_{i-1:i'} B_{i'} & \cdots & A^\times_{i-1:i-1} B_{i-1} \end{bmatrix}.$$

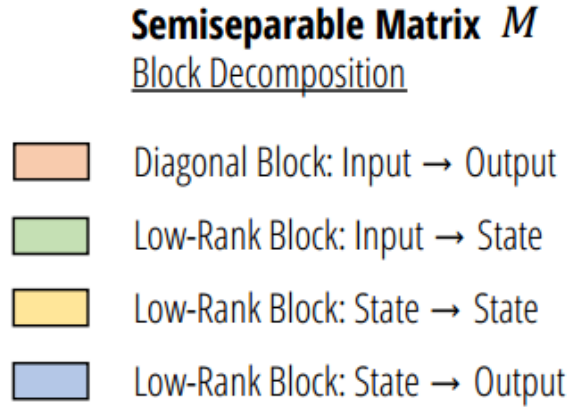
$$M = \left[\begin{array}{ccc|ccc|cc|c} C_0^\top A_{0:0} B_0 & & & & & & & & \\ C_1^\top A_{1:0} B_0 & C_1^\top A_{1:1} B_1 & & & & & & & \\ C_2^\top A_{2:0} B_0 & C_2^\top A_{2:1} B_1 & C_2^\top A_{2:2} B_2 & & & & & & \\ \hline C_3^\top A_{3:0} B_0 & C_3^\top A_{3:1} B_1 & C_3^\top A_{3:2} B_2 & C_3^\top A_{3:3} B_3 & & & & & \\ C_4^\top A_{4:0} B_0 & C_4^\top A_{4:1} B_1 & C_4^\top A_{4:2} B_2 & C_4^\top A_{4:3} B_3 & C_4^\top A_{4:4} B_4 & & & & \\ C_5^\top A_{5:0} B_0 & C_5^\top A_{5:1} B_1 & C_5^\top A_{5:2} B_2 & C_5^\top A_{5:3} B_3 & C_5^\top A_{5:4} B_4 & C_5^\top A_{5:5} B_5 & & & \\ \hline C_6^\top A_{6:0} B_0 & C_6^\top A_{6:1} B_1 & C_6^\top A_{6:2} B_2 & C_6^\top A_{6:3} B_3 & C_6^\top A_{6:4} B_4 & C_6^\top A_{6:5} B_5 & C_6^\top A_{6:6} B_6 & & \\ C_7^\top A_{7:0} B_0 & C_7^\top A_{7:1} B_1 & C_7^\top A_{7:2} B_2 & C_7^\top A_{7:3} B_3 & C_7^\top A_{7:4} B_4 & C_7^\top A_{7:5} B_5 & C_7^\top A_{7:6} B_6 & C_7^\top A_{7:7} B_7 & \\ C_8^\top A_{8:0} B_0 & C_8^\top A_{8:1} B_1 & C_8^\top A_{8:2} B_2 & C_8^\top A_{8:3} B_3 & C_8^\top A_{8:4} B_4 & C_8^\top A_{8:5} B_5 & C_8^\top A_{8:6} B_6 & C_8^\top A_{8:7} B_7 & C_8^\top A_{8:8} B_8 \end{array}\right]$$

$$= \left[\begin{array}{ccc|ccc|cc|c} C_0^\top A_{0:0} B_0 & & & & & & & & \\ C_1^\top A_{1:0} B_0 & C_1^\top A_{1:1} B_1 & & & & & & & \\ C_2^\top A_{2:0} B_0 & C_2^\top A_{2:1} B_1 & C_2^\top A_{2:2} B_2 & & & & & & \\ \hline \multicolumn{3}{c|}{\begin{bmatrix} C_3^\top A_{3:2} \\ C_4^\top A_{4:2} \\ C_5^\top A_{5:2} \end{bmatrix} A_{2:2} \begin{bmatrix} B_0^\top A_{2:0} \\ B_1^\top A_{2:1} \\ B_2^\top A_{2:2} \end{bmatrix}^\top} & C_3^\top A_{3:3} B_3 & & & & \\ & & & C_4^\top A_{4:3} B_3 & C_4^\top A_{4:4} B_4 & & & & \\ & & & C_5^\top A_{5:3} B_3 & C_5^\top A_{5:4} B_4 & C_5^\top A_{5:5} B_5 & & & \\ \hline \multicolumn{3}{c|}{\begin{bmatrix} C_6^\top A_{6:5} \\ C_7^\top A_{7:5} \\ C_8^\top A_{8:5} \end{bmatrix} A_{5:2} \begin{bmatrix} B_0^\top A_{2:0} \\ B_1^\top A_{2:1} \\ B_2^\top A_{2:2} \end{bmatrix}^\top} & \multicolumn{3}{c|}{\begin{bmatrix} C_6^\top A_{6:5} \\ C_7^\top A_{7:5} \\ C_8^\top A_{8:5} \end{bmatrix} A_{5:5} \begin{bmatrix} B_3^\top A_{5:3} \\ B_4^\top A_{5:4} \\ B_5^\top A_{5:5} \end{bmatrix}^\top} & C_6^\top A_{6:6} B_6 & & \\ & & & & & & C_7^\top A_{7:6} B_6 & C_7^\top A_{7:7} B_7 & \\ & & & & & & C_8^\top A_{8:6} B_6 & C_8^\top A_{8:7} B_7 & C_8^\top A_{8:8} B_8 \end{array}\right]$$

状态空间对偶理论 SSD







- 橙色：每个对角线块都是一个较小的半可分离矩阵，可以随意计算
- 绿色：只有T/Q 完全不同的绿色块，中间有很多是共享，可以使用批处matmul计算
- 黄色：黄色项本身形成一个1-半可分离矩阵，可以使用SSM扫描
- 蓝色：与绿色类似，这些可以使用批处理 matmul 进行计算

Figure 5: (**SSD Algorithm**.) By using the matrix transformation viewpoint of state space models to write them as semiseparable matrices (Section 3), we develop a more hardware-efficient computation of the SSD model through a block-decomposition matrix multiplication algorithm. The matrix multiplication also has an interpretation as a state space model, where blocks represent chunking the input and output sequence. Diagonal blocks represent intra-chunk computations and the off-diagonal blocks represent inter-chunk computations, factored through the SSM's hidden state.

Listing 1 Full PyTorch example of the state space dual (SSD) model.

```python
def segsum(x):
    """Naive segment sum calculation. exp(segsum(A)) produces a 1-SS matrix,
       which is equivalent to a scalar SSM."""
    T = x.size(-1)
    x_cumsum = torch.cumsum(x, dim=-1)
    x_segsum = x_cumsum[..., :, None] - x_cumsum[..., None, :]
    mask = torch.tril(torch.ones(T, T, device=x.device, dtype=bool), diagonal=0)
    x_segsum = x_segsum.masked_fill(~mask, -torch.inf)
    return x_segsum

def ssd(X, A, B, C, block_len=64, initial_states=None):
    """
    Arguments:
        X: (batch, length, n_heads, d_head)
        A: (batch, length, n_heads)
        B: (batch, length, n_heads, d_state)
        C: (batch, length, n_heads, d_state)
    Return:
        Y: (batch, length, n_heads, d_head)
    """
    assert X.dtype == A.dtype == B.dtype == C.dtype
    assert X.shape[1] % block_len == 0

    # Rearrange into blocks/chunks
    X, A, B, C = [rearrange(x, "b (c l) ... -> b c l ...", l=block_len) for x in (X, A, B, C)]

    A = rearrange(A, "b c l h -> b h c l")
    A_cumsum = torch.cumsum(A, dim=-1)

    # 1. Compute the output for each intra-chunk (diagonal blocks)
    L = torch.exp(segsum(A))
    Y_diag  = torch.einsum("bclhn,bcshn,bhcls,bcshp->bclhp", C, B, L, X)

    # 2. Compute the state for each intra-chunk
    # (right term of low-rank factorization of off-diagonal blocks; B terms)
    decay_states = torch.exp((A_cumsum[:, :, :, -1:] - A_cumsum))
    states = torch.einsum("bclhn,bhcl,bclhp->bchpn", B, decay_states, X)

    # 3. Compute the inter-chunk SSM recurrence; produces correct SSM states at chunk boundaries
    # (middle term of factorization of off-diag blocks; A terms)
    if initial_states is None:
        initial_states = torch.zeros_like(states[:, :1])
    states = torch.cat([initial_states, states], dim=1)
    decay_chunk = torch.exp(segsum(F.pad(A_cumsum[:, :, :, -1], (1, 0))))
    new_states = torch.einsum("bhzc,bchpn->bzhpn", decay_chunk, states)
    states, final_state = new_states[:, :-1], new_states[:, -1]

    # 4. Compute state -> output conversion per chunk
    # (left term of low-rank factorization of off-diagonal blocks; C terms)
    state_decay_out = torch.exp(A_cumsum)
    Y_off = torch.einsum('bclhn,bchpn,bhcl->bclhp', C, states, state_decay_out)

    # Add output of intra-chunk and inter-chunk terms (diagonal and off-diagonal blocks)
    Y = rearrange(Y_diag+Y_off, "b c l h p -> b (c l) h p")
    return Y, final_state
```

$$
\begin{aligned}
x &= uW^{(x)\top} \in \mathbb{R}^{L\times ed} \\
z &= uW^{(z)\top} \in \mathbb{R}^{L\times ed} \\
\Delta, B, C &= \text{projection}(u) \quad \text{(one or more groups of } \Delta, B, C \text{ per GPU)} \\
x_c &= \text{conv1d}(x) \in \mathbb{R}^{L\times ed} \quad \text{(depthwise, independent along } d\text{)} \\
y &= SSM_{A,B,C,\Delta}(x_c) \in \mathbb{R}^{L\times ed} \quad \text{(independent along } d\text{)} \\
y_g &= y \cdot \phi(z) \quad \text{(gating, e.g., with } \phi \text{ being SiLU)} \\
y_n &= \text{groupnorm}(y_g) \quad \text{(number of groups divisible by degree of tensor parallel)} \\
\text{out} &= y_g W^{(o)\top} \in \mathbb{R}^{L\times d}.
\end{aligned}
$$

- *Computation cost*: total of $O(\mathtt{BMNK})$ FLOPs.
- *Memory cost:* total of $O(\mathtt{B}(\mathtt{MK}+\mathtt{KN}+\mathtt{MN}))$ space.

Comparison to Pure SSM and Attention Models. Quadratic attention is also very hardware efficient by only leveraging matrix multiplications, but has $\mathtt{T}^2N$ total FLOPs. Its slower computation speed at both training and inference can directly be seen as a consequence of having a larger state size – standard attention has a state size scaling with sequence length $\mathtt{T}$ because it caches its history and does not compress its state.

Linear SSMs have $\mathtt{TNP} = \mathtt{TN}^2$ total FLOPs, which is the same as SSD. However, a naive implementation requires a state expansion (15a) that materializes extra memory, and a scalar operation (15b) that does not leverage matrix multiplications.

	Attention	SSM	**SSD**
State size	$\mathtt{T}$	$\mathtt{N}$	$\mathtt{N}$
Training FLOPs	$\mathtt{T}^2\mathtt{N}$	$\mathtt{TN}^2$	$\mathtt{TN}^2$
Inference FLOPs	$\mathtt{TN}$	$\mathtt{N}^2$	$\mathtt{N}^2$
(Naive) memory	$\mathtt{T}^2$	$\mathtt{TN}^2$	$\mathtt{TN}$
Matrix multiplication	✓		✓

SSD Summary

Structured State Space Model		Structured Masked Attention	
C	(contraction matrix)	Q	(queries)
B	(expansion matrix)	K	(keys)
X	(input sequence)	V	(values)
$A_{j:i}$	(state matrix)	L_{ji}	(mask)
N	(state expansion dim.)	N	(kernel feature dim.)
H	(hidden states (8b))	SMA linear dual (15)	
$= L \cdot XB$	(linear mode)		
SSM quadratic dual (16)		G	(Gram matrix (13a))
		$= Q \cdot K^\top$	(quadratic mode)

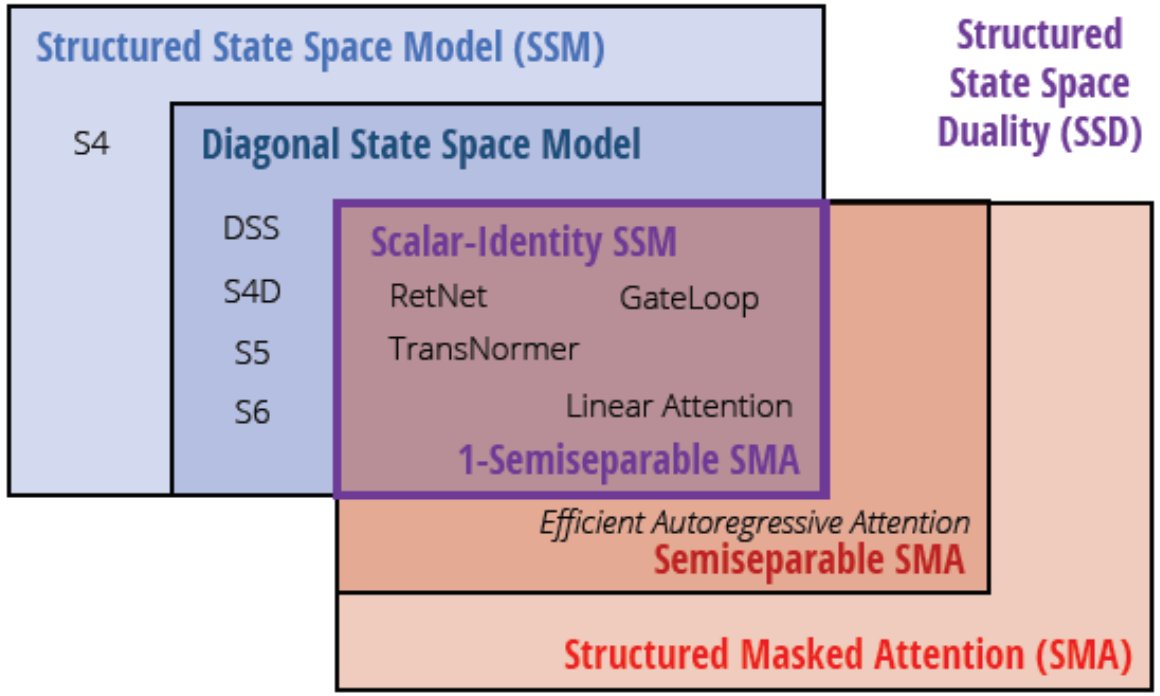


Figure 4: (**Structured State Space Duality**.) State space duality describes the close relationship between state space models and masked attention. (*Left*) General SSMs and SMA both possess linear and quadratic forms, with direct analogs in notation. (*Right*) SSMs and SMA intersect at a large class of *state space dual models* (SSD) which capture many sequence models as special cases.

Mamba-2 version

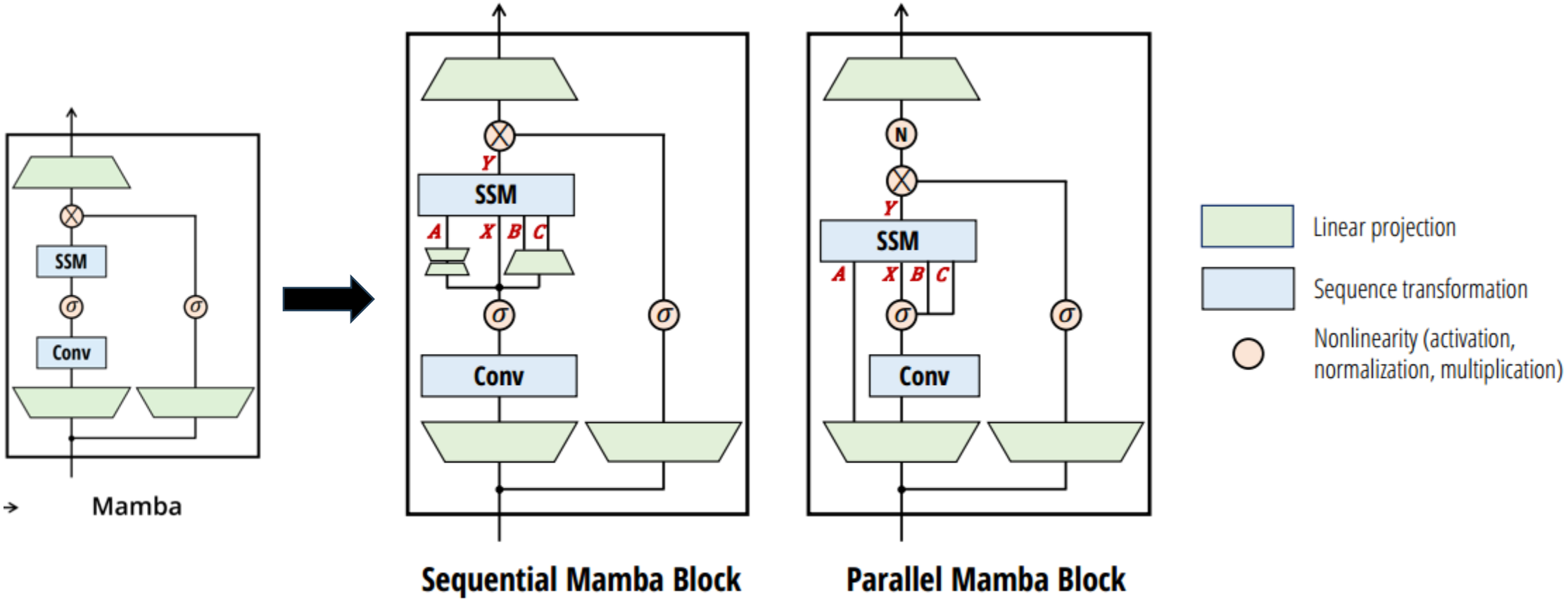


Figure 6: (**Mamba-2 Architecture**.) The Mamba-2 block simplifies the Mamba block by removing sequential linear projections; the SSM parameters A, B, C are produced at the beginning of the block instead of as a function of the SSM input X. An additional normalization layer is added as in NormFormer (Shleifer, Weston, and Ott 2021), improving stability. The B and C projections only have a single head shared across the X heads, analogous to multi-value attention (MVA).

Mamba-2 version

基于SSD思想的全新算法，Mamba-2 支持更大的状态维度，从16扩展到了256，从而能学习到更强的表示能力

新方法采用了基于块分解的矩阵乘法，利用了GPU的存储层次结构，从而提升了训练速度。

Multi-head SSM
(Multi-head Attn.)

$$\begin{aligned} X &\quad (T, H, P) \\ A &\quad (T, H) \\ B &\quad (T, H, N) \\ C &\quad (T, H, N) \end{aligned} \tag{17}$$

Multi-contract SSM
(Multi-query Attn.)

$$\begin{aligned} X &\quad (T, 1, P) \\ A &\quad (T, H) \\ B &\quad (T, 1, N) \\ C &\quad (T, H, N) \end{aligned} \tag{18}$$

Multi-expand SSM
(Multi-key Attn.)

$$\begin{aligned} X &\quad (T, 1, P) \\ A &\quad (T, H) \\ B &\quad (T, H, N) \\ C &\quad (T, 1, N) \end{aligned} \tag{19}$$

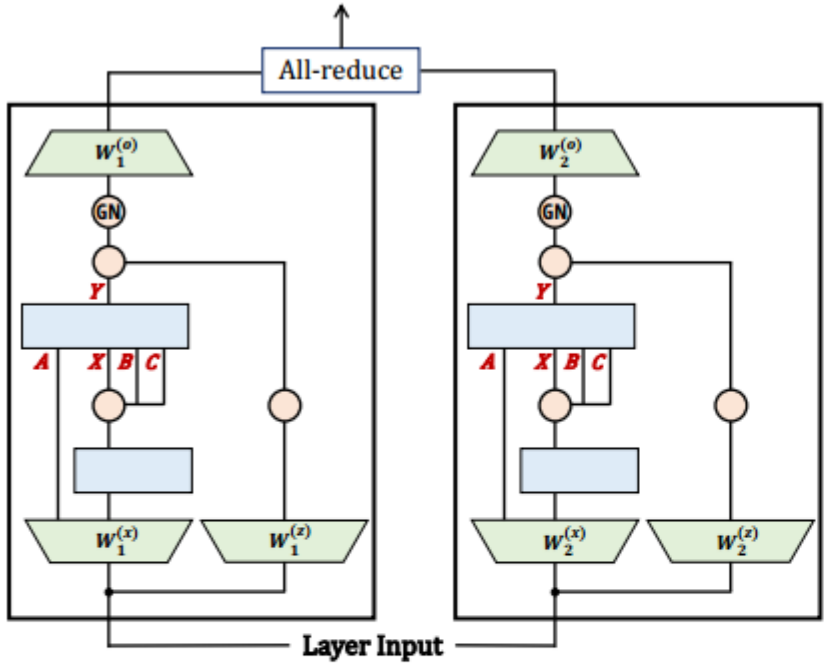
Multi-input SSM
(Multi-value Attn.)

$$\begin{aligned} X &\quad (T, H, P) \\ A &\quad (T, H) \\ B &\quad (T, 1, N) \\ C &\quad (T, 1, N) \end{aligned} \tag{20}$$

借鉴多头注意力机制，创建了多输入SSM。
通过与注意力机制之间的联系，**SSD能够轻松将Transformer架构多年来积累的优化方法引入SSM。**

Mamba-2 version

- 借鉴多头注意力机制，创建了多输入SSM。
- 通过与注意力机制之间的联系，**SSD能够轻松将Transformer架构多年来积累的优化方法引入SSM。**



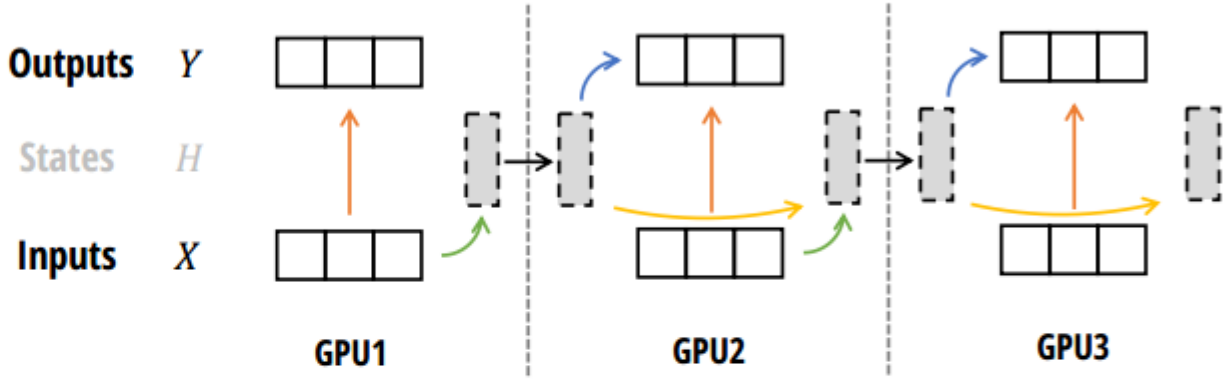


Figure 7: (**Parallelism with the Mamba-2 Block**.) (*Left*: **Tensor Parallelism**) We split the input projection matrices $W^{(x)}, W^{(z)}$ and the output projection matrix $W^{(o)}$. Each SSM head $(A, B, C, X) \mapsto Y$ lives on a single device. Choosing GroupNorm for the final normalization layer avoids extra communication. We need one all-reduce per layer, just like the MLP or attention blocks in a Transformer. (*Right*: **Sequence/Context Parallelism**) Analogous to the SSD algorithm, with multiple devices, we can split along the sequence dimension. Each device computes the state of its sequence, then pass that state to the next GPU.

Mamba-2 version

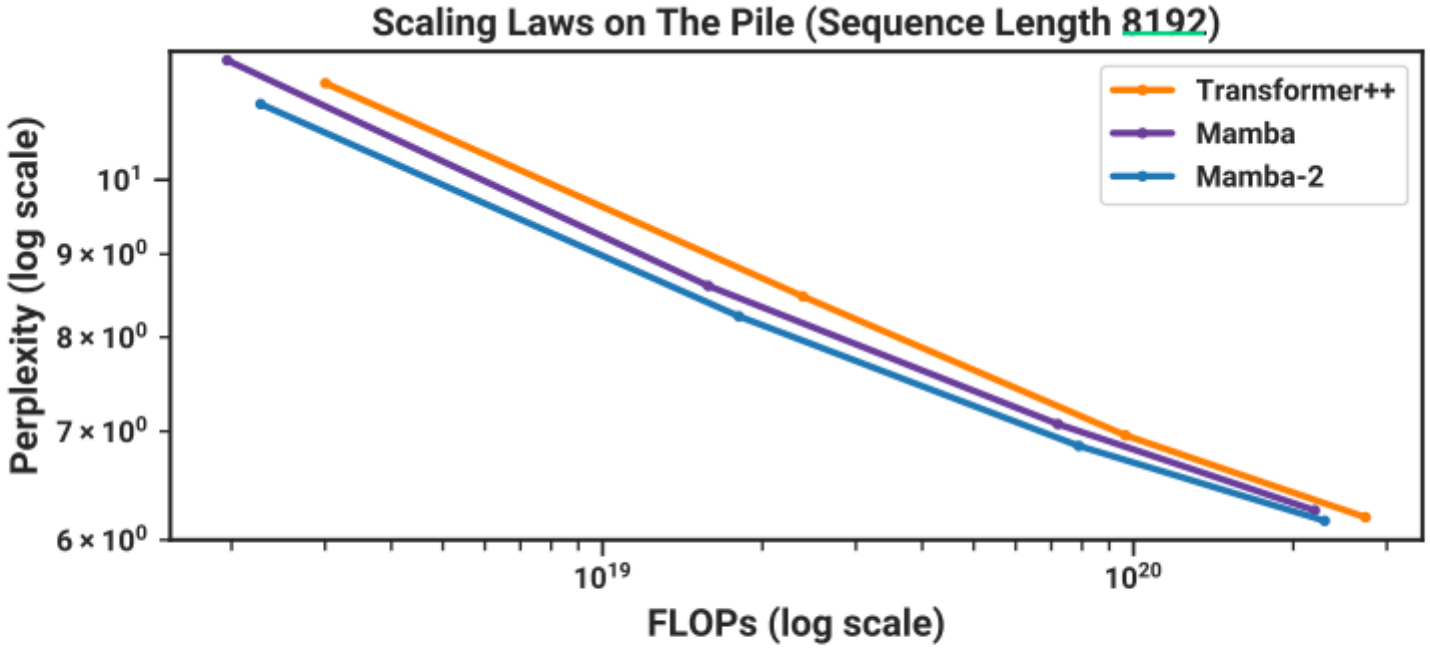


Figure 9: **(Scaling Laws.)** Models of size $\approx 125M$ to $\approx 1.3B$ parameters, trained on the Pile. Mamba-2 matches or exceeds the performance of Mamba as well as a strong "Transformer++" recipe. Compared to our Transformer baseline, Mamba-2 is Pareto dominant on performance (perplexity), theoretical FLOPs, and actual wall-clock time.

Mamba-2 version

Table 10: (**Zero-shot Evaluations**.) Best results for each size in bold, second best unlined. We compare against open source LMs with various tokenizers, trained for up to 300B tokens. Pile refers to the validation split, comparing only against models trained on the same dataset and tokenizer (GPT-NeoX-20B). For each model size, Mamba-2 outperforms Mamba, and generally matches Pythia at twice the model size.

Model	Token.	Pile ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	Arc-E acc ↑	Arc-C acc ↑	WinoGrande acc ↑	OpenbookQA acc ↑	Average acc ↑
Hybrid H3-130M	GPT2	—	89.48	25.8	31.7	64.2	44.4	**24.2**	50.6	27.0	38.2
Pythia-160M	NeoX	29.64	38.10	33.0	30.2	61.4	43.2	24.1	51.9	29.2	39.0
Mamba-130M	NeoX	10.56	**16.07**	**44.3**	35.2	64.5	**48.0**	**24.2**	51.9	28.8	42.4
Mamba-2-130M	NeoX	**10.48**	16.86	43.9	**35.3**	**64.9**	47.4	**24.2**	**52.1**	**30.6**	**42.6**
Hybrid H3-360M	GPT2	—	12.58	48.0	41.5	68.1	51.4	24.7	54.1	31.6	45.6
Pythia-410M	NeoX	9.95	10.84	51.4	40.6	66.9	52.1	24.6	53.8	30.0	45.6
Mamba-370M	NeoX	8.28	8.14	55.6	46.5	69.5	**55.1**	**28.0**	55.3	30.8	48.7
Mamba-2-370M	NeoX	**8.21**	**8.02**	**55.8**	**46.9**	**70.5**	54.9	26.9	**55.7**	**32.4**	**49.0**
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	31.4	49.0
Mamba-790M	NeoX	7.33	6.02	**62.7**	**55.1**	**72.1**	**61.2**	**29.5**	56.1	34.2	53.0
Mamba-2-780M	NeoX	**7.26**	**5.86**	61.7	54.9	72.0	61.0	28.5	**60.2**	**36.2**	**53.5**
GPT-Neo 1.3B	GPT2	—	7.50	57.2	48.9	71.1	56.2	25.9	54.9	33.6	49.7
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	34.4	50.3
OPT-1.3B	OPT	—	6.64	58.0	53.7	72.4	56.7	29.6	59.5	33.2	51.9
Pythia-1.4B	NeoX	7.51	6.08	61.7	52.1	71.0	60.5	28.5	57.2	30.8	51.7
RWKV4-1.5B	NeoX	7.70	7.04	56.4	52.5	72.4	60.5	29.4	54.6	34.0	51.4
Mamba-1.4B	NeoX	6.80	5.04	65.0	59.1	**74.2**	**65.5**	32.8	**61.5**	36.4	**56.4**
Mamba-2-1.3B	NeoX	**6.66**	**5.02**	**65.7**	**59.9**	73.2	64.3	**33.3**	60.9	**37.8**	**56.4**
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8	72.1	61.1	30.2	57.6	33.2	53.2
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	33.6	54.5
OPT-2.7B	OPT	—	5.12	63.6	60.6	74.8	60.8	31.3	61.0	35.2	55.3
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3	74.0	64.1	32.9	59.7	35.2	55.7
RWKV4-3B	NeoX	7.00	5.24	63.9	59.6	73.7	67.8	33.1	59.6	37.0	56.4
Mamba-2.8B	NeoX	6.22	4.23	69.2	66.1	75.2	**69.7**	36.3	63.5	**39.6**	59.9
Mamba-2-2.7B	NeoX	**6.09**	**4.10**	**69.7**	**66.6**	**76.4**	69.6	**36.4**	**64.0**	38.8	**60.2**
GPT-J-6B	GPT2	–	4.10	68.3	66.3	75.4	67.0	36.6	64.1	38.2	59.4
OPT-6.7B	OPT	–	4.25	67.7	67.2	76.3	65.6	34.9	65.5	37.4	59.2
Pythia-6.9B	NeoX	6.51	4.45	67.1	64.0	75.2	67.3	35.5	61.3	38.0	58.3
RWKV4-7.4B	NeoX	6.31	4.38	67.2	65.5	76.1	67.8	37.5	61.0	40.2	59.3

Q&A
Thanks!