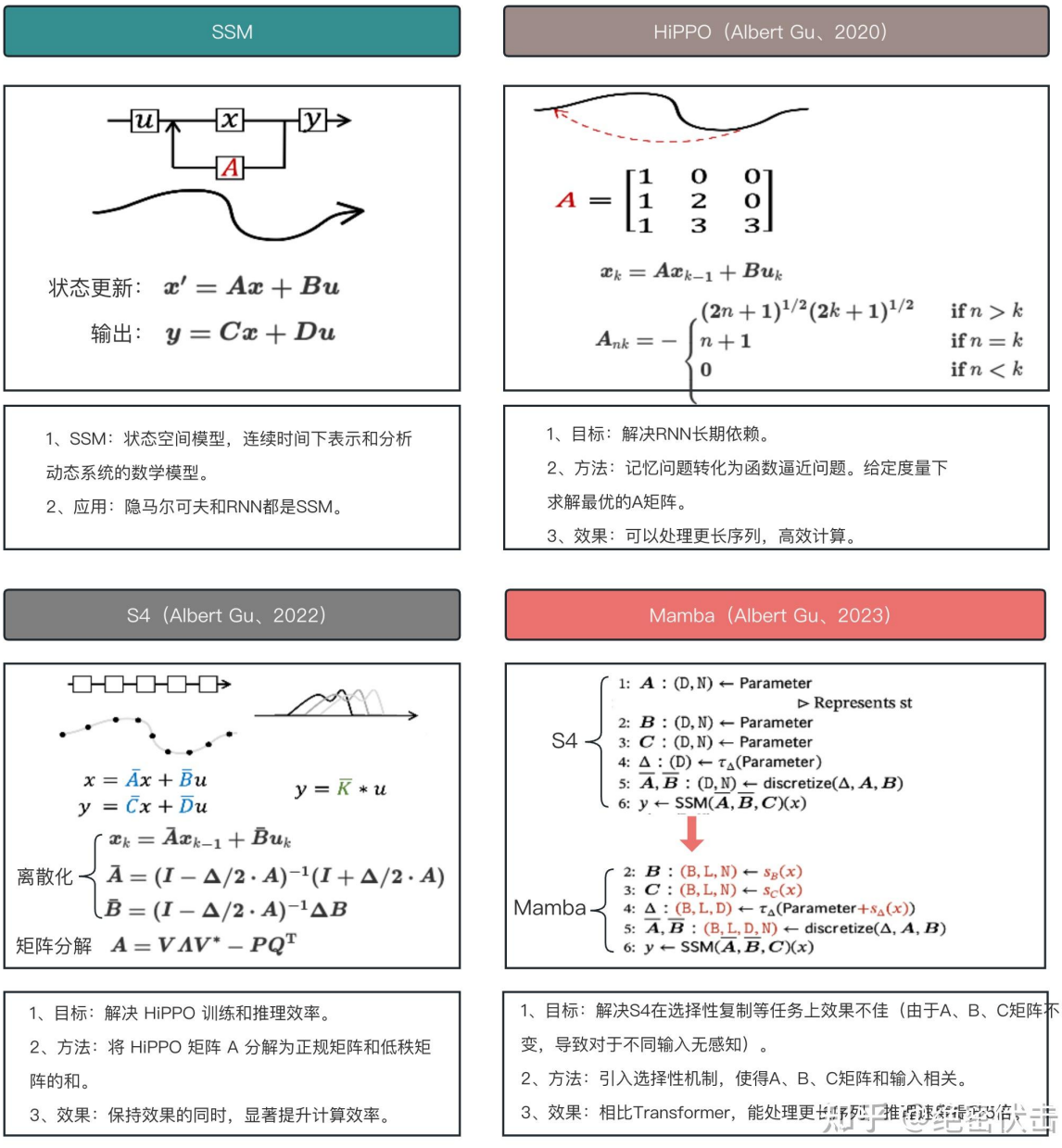
Mamba论文与代码详解

本文详细的解读 Mamba 架构，由于 Mamba 是基于 SSM->HiPPO->S4->Mamba 演化过来的，而 HiPPO、S4、Mamba 的一作者都是卡内基梅隆大学机器学习系助理教授 **Albert Gu**。因此，本文将从标准 SSM 开始，逐步介绍 HiPPO、S4、Mamba。

下图总结了SSM、HiPPO、S4、Mamba的主要区别，以及各个模型的主要内容。本文内容也将按图中内容展开。



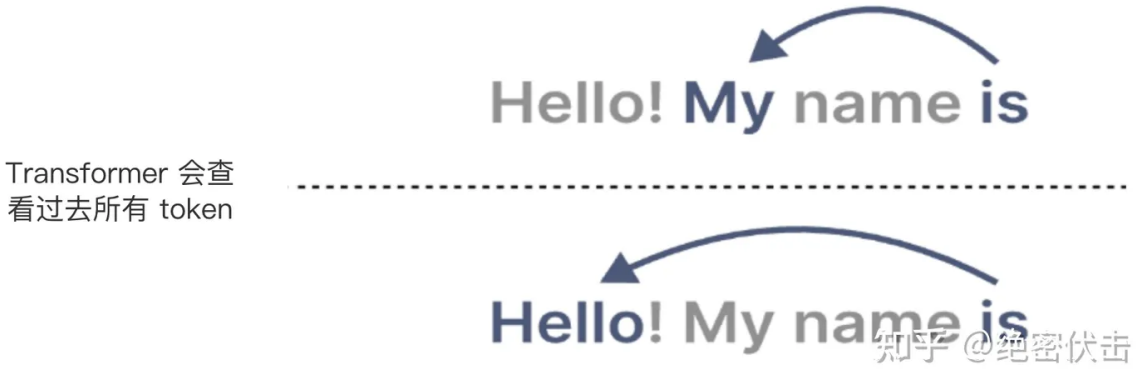
一、现有架构问题

序列建模的核心问题是：同时解决**有效**和**高效**。有效是指能够选择性记忆历史信息，解决**长距离依赖**（Long-Range Dependencies，LRDs）问题；高效是指计算高效。

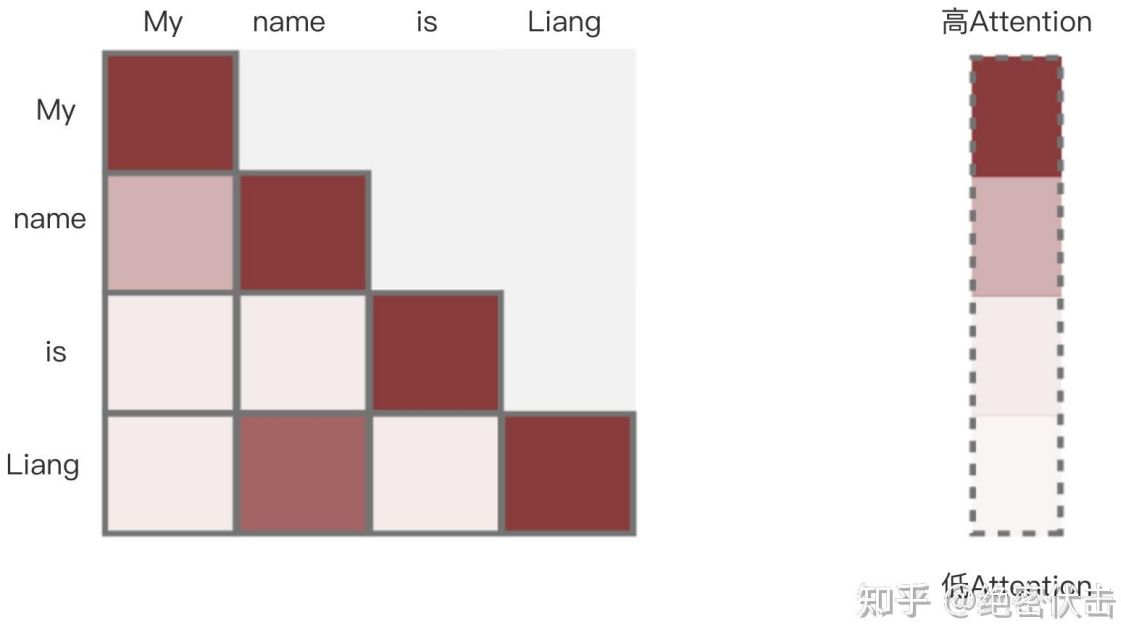
尽管传统的模型如循环神经网络（RNNs）、卷积神经网络（CNNs）和 Transformers 在处理长距离依赖方面有专门的变体，但它们在**处理超过 10000 步的极长序列时仍然面临挑战。**

1.1 Transformer 问题

Transformer 的一个主要优点是，无论它接收到多长的输入，它都使用序列中的所有 token 信息（无论序列有多长）来对输入数据进行处理。



但是为了获得全局信息，注意力机制在长序列上非常耗费显存。注意力创建一个矩阵，将每个 token 与之前的每个 token 进行比较。矩阵中的权重由 token 对之间的相关性决定。

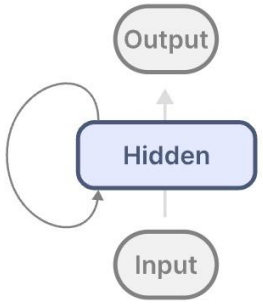


在训练过程中，Attention 计算可以并行化，所以可以极大地加快训练速度。但是在推理过程中，当生成下一个 token 时，我们需要重新计算整个序列的注意力。

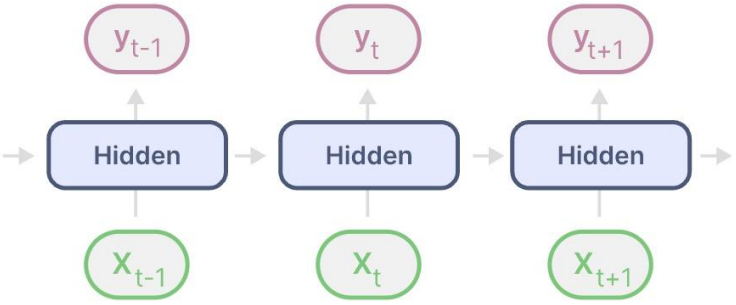
长度为 L 的序列生成 token 大约需要 L^2 的计算量，如果序列长度增加，计算量会平方级增长。因此，**需要重新计算整个序列是 Transformer 体系结构的主要瓶颈。**

	Training	Inference
Transformers	**Fast!** (parallelizable)	**Slow...** (scales **quadratically** with sequence length)

1.2 RNN 的问题



RNN

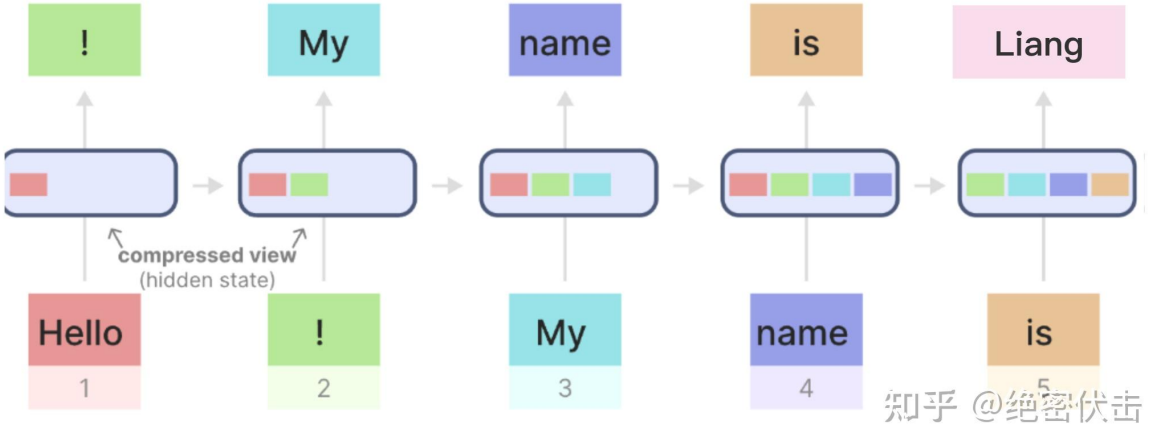


RNN (Unfolded)

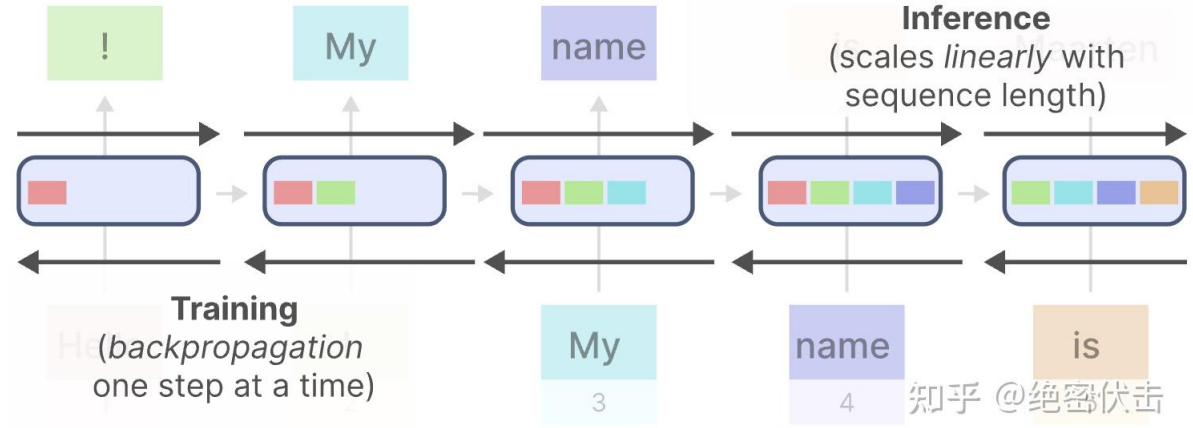
在生成输出时，RNN 只需要考虑之前的隐藏状态和当前的输入。这样不会重新计算以前的隐藏状态，这正Transformer 不具备的。

这种结构可以让 RNN 进行**快速推理**，并且理论上可以无限扩展上下文长度，因为每次推理只取一个隐藏状态和当前输入，内存占用非常稳定。

RNN 的每个隐藏状态都是之前所有隐藏状态的聚合。但是这里会有一个问题，在生成 token "Liang" 时，最后一个隐藏状态不再包含关于 token "Hello" 的信息。这会导致随着时间的推移，RNN 会忘记更久的信息，因为它只考虑前一个状态。



并且 RNN 的这种顺序性产生了另一个问题。**训练不能并行进行**，因为它需要按顺序完成每一步。



RNN的统一定义为：

$$x_t = f(\mathbf{u}_t, \mathbf{x}_{t-1}; \theta) \tag{1}$$

其中$\mathbf{x}_t$是每一步的输出，它由当前输入$\mathbf{u}_t$和前一时刻输出 $\mathbf{x}_{t-1}$ 共同决定，而 θ 则是可训练参数。那么参数 θ 的梯度可以表示为：

$$\nabla_\theta \mathbf{x}_t = \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_{t-1}} \nabla_\theta \mathbf{x}_{t-1} + \frac{\partial \mathbf{x}_t}{\theta} \tag{2}$$

可以看到，当前梯度依赖上个 token 的梯度。

与 Transformer 相比，RNN 的问题完全相反！它的**推理速度非常快，但不能并行化导致训练很慢。**

人们一直在寻找一种既能像 Transformer 那样并行化训练，能够记住先前的信息，又能在推理时时间是随序列长度线性增长的模型，Mamba 就是这样应运而生的。解下来我们从 SSM 开始，逐步介绍 Mamaba。

二、状态空间模型 SSM

2.1 什么是 SSM

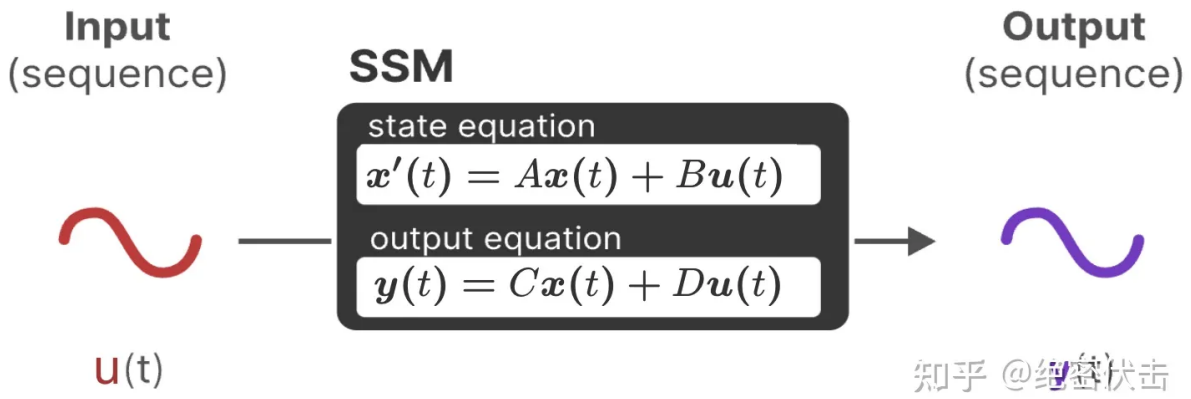
状态空间模型（State Space Models，SSM）由简单的方程(3)定义。它将一维输入信号$u(t) \in \mathbb{R}$映射到 N 维潜在状态 $x(t) \in \mathbb{R}^N$，然后再投影到一维输出信号 $y(t)$。

$$\begin{aligned} \frac{d\mathbf{x}_t}{dt} &= \mathbf{A}x(t) + \mathbf{B}u(t) \in \mathbb{R}^N \\ y(t) &= \mathbf{C}x(t) + Du(t) \in \mathbb{R} \end{aligned} \tag{3}$$

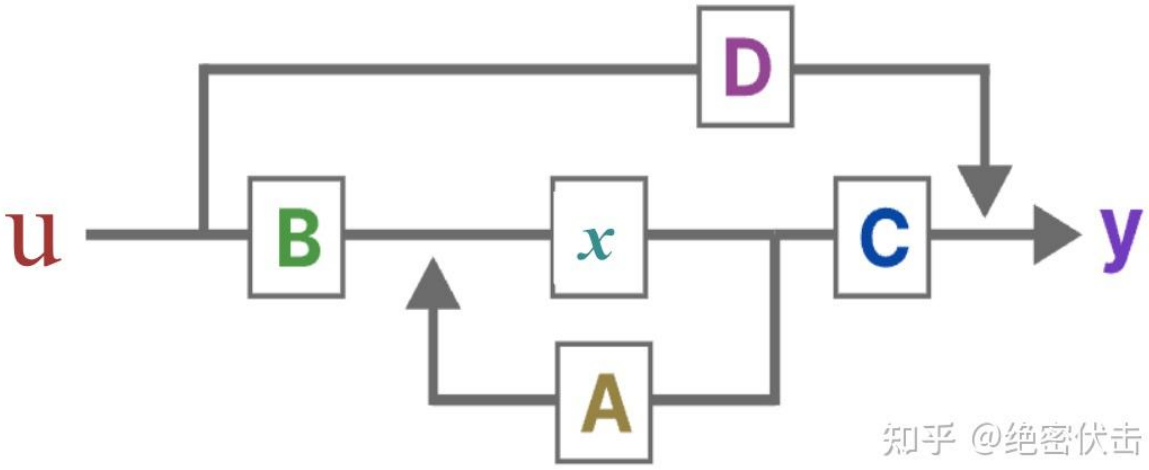
其中，$\mathbf{A} \in \mathbb{R}^{N \times N}$ 是状态转移矩阵，$\mathbf{B} \in \mathbb{R}^{N \times 1}$是输入到状态的矩阵，$\mathbf{C} \in \mathbb{R}^{N \times 1}$是状态到输出的矩阵，D是直接从输入到输出的参数（很多时候取 D = 0）。

2.2 SSM 架构

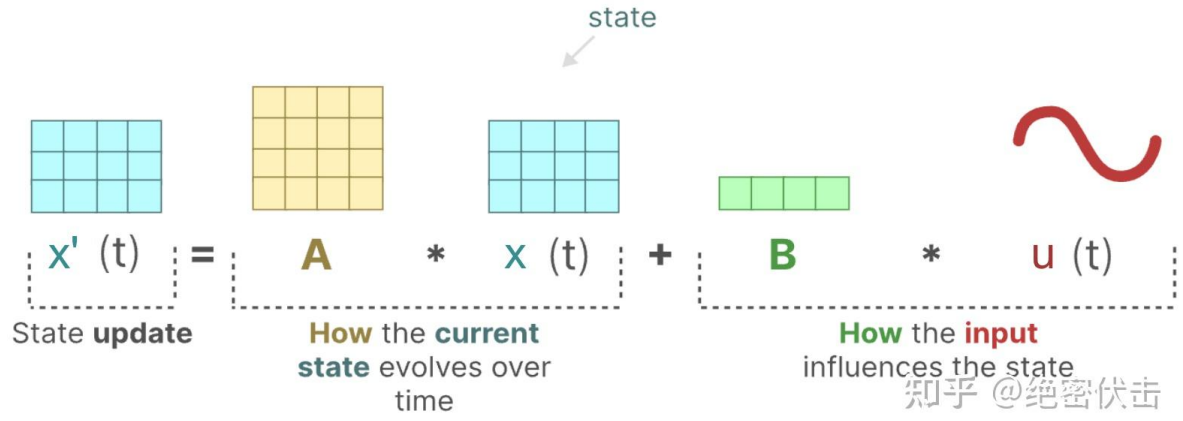
下图是 SSM 的架构，主要包含两个部分：状态更新方程和输出方程。



SSM 可以简化为以下结构：



下面我们看一下更详细的结构，首先是状态更新，如下所示：

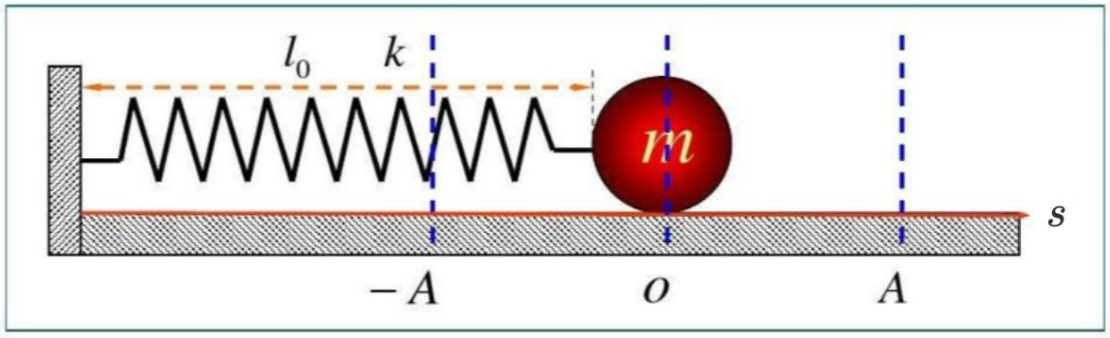


备注：图中的输入$\mathbf{u}(t) \in \mathbb{R}^D$，表示输入的信号是 D 维的。 SSM 也可以用于处理多维信号输入。

然后是输出方程，详细机构如下所示：

2.3 SSM 例子：弹簧振子

下面举一个描述弹簧振子系统的 SSM 例子。



$$s = 0, F = 0$$

考虑一个质量为 m 的物体，它连接在一个劲度系数为 k 的弹簧上，并且受到阻尼系数为 c 的阻尼力作用。当物体从平衡位置偏离时，它会在弹簧力的作用下进行振动。我们可以用状态空间模型来描述这个系统的动态。

状态变量可以选择为物体的位移 $s(t)$ 和速度 $v(t)$ 。输入 $u(t)$ 在这个例子中可以为零，因为我们没有外部力作用在物体上。输出 $y(t)$ 可以是我们感兴趣的位移 $s(t)$。

状态向量定义为：

$$\mathbf{x}(t) = \begin{bmatrix} s(t) \\ v(t) \end{bmatrix} \tag{4}$$

输入向量为：

$$\mathbf{u}(t) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5}$$

输出位移$s(t)$。弹簧振子的状态空间方程可以表示为：

$$\begin{aligned} \frac{d}{dt}\begin{bmatrix} s(t) \\ v(t) \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix}\begin{bmatrix} s(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\mathbf{u}(t) \\ y(t) &= \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} s(t) \\ v(t) \end{bmatrix} \end{aligned} \tag{6}$$

在了解 SSM 基本概念之后，接下来我们介绍基于 SSM 的 HiPPO 架构。

三、HiPPO（High-order Polynomial Projection Operators）

HiPPO 是 Albert Gu 于2020年在论文 HiPPO: Recurrent Memory with Optimal Polynomial Projections 中提出的新架构。HiPPO 主要为了解决**如何在有限的存储空间中有效地解决序列建模的长距离依赖问题。**

HiPPO 通过函数逼近产生状态矩阵 A 的最优解，有效的解决了长距离依赖问题。

问题背景：在处理序列数据时，一个核心问题是如何在增量方式下表示累积的历史信息。这涉及到如何在有限的存储空间中有效地更新和维护历史数据的表示。

HiPPO框架：作者介绍了一个名为 HiPPO（High-order Polynomial Projection Operators）的通用框架，它通过将连续信号和离散时间序列投影到多项式基上，实现了在线数据压缩。

重要性度量：HiPPO 框架考虑了一个度量，用于指定过去每个时间步的重要性。这个度量帮助HiPPO产生在线函数逼近问题的最优解。

理论贡献：HiPPO 框架不仅提供了对现有记忆单元的简短推导，还推广了循环神经网络（如GRUs）中普遍存在的门控机制。

新的记忆更新机制：作者提出了一个新的记忆更新机制（HiPPO-LegS），它能够随时间扩展以记住所有历史信息，避免了对时间尺度的先验假设。

理论优势：HiPPO-LegS 具有时间尺度鲁棒性、快速更新和有界梯度的理论优势。

实验结果：在基准测试中，HiPPO-LegS 在打乱的 MNIST 数据集上达到了98.3%的新最佳准确率。在一个新的轨迹分类任务中，HiPPO-LegS 在处理分布外时间尺度和缺失数据方面，比其他 RNN 和神经 ODE（一阶常微分方程）基线模型的性能提高了25-40%的准确率。

下面介绍 HiPPO 实现的具体细节。

3.1 HiPPO 架构：高阶多项式投影

3.1.1 HiPPO问题设置

问题定义

给定一个在时间 t 上的输入函数$u(t) \in \mathbb{R}$，需要在每个时间点操作累计历史$u^{\leq t} := u(x)|x \leq t$，以便理解到目前为止看到的输入并对未来进行预测。

由于函数空间的庞大，无法完美记住整个历史，因此需要将其进行压缩，HiPPO 提出了将历史投影到有界维数的子空间的一半方法。

函数逼近与度量

为了评估逼近的质量，需要在函数空间中定义一个距离。任何在 $[0, \infty)$ 上的概率度量 μ 都可以为平方可积函数空间提供内积 $< u, g >_\mu = \int_0^\infty u(x)g(x)d\mu(x)$ ，从而诱导出一个希尔伯特空间 H_μ 和相应的范数$||u||_{L^2(\mu)} =< u, u >_\mu^{1/2}$。

为了选择合适的子空间，需要一个度量来量化历史的重要性。这个度量$\mu(t)$随时间变化，支持在$[-\infty, t)$上，因为 $u^{\leq t}$ 只在时间t之前定义。

多项式基展开

任何 N 维的函数子空间 G 都是逼近的合适候选。参数 N 对应于逼近的阶数，或者说压缩的大小；投影的历史可以通过G的任何基的N个系数来表示。

论文中使用多项式作为自然基，因此G是小于N阶的多项式的集合。

在线逼近

由于我们关心在每个时间 t 对 $u^{\leq t}$ 的逼近，我们也让度量 $\mu(t)$ 随时间变化。总体上，我们寻找一个 $g(t) \in G$ ，使得 $||u^{\leq t} - g(t)||_{L^2(\mu(t))}$ 最小。直观上，度量 μ 控制输入域各部分的重要性。

挑战

挑战在于如何在给定度量 $\mu(t)$ 的情况下以封闭形式解决优化问题，以及在 $t \to \infty$ 时如何线性地维护这些系数。

3.1.2 HiPPO 通用架构

通过连续动态系统计算投影

这部分是 HiPPO 的关键步骤，它涉及到将输入函数 $u(t)$ 在时间 t 投影到一个多项式空间上，以便在线更新记忆表示。

投影的表示：投影可以通过输入函数 $u(t)$ 在时间 t 的限制 $u^{\leq t}$ 的 N 个系数来表示。这些系数是通过在多项式空间的基上展开$u \leq t$得到的。

正交多项式基：为了选择合适的基，作者利用了正交多项式的性质。正交多项式为$u(t)$提供了一个自然的基，使得 $u \leq t$ 的投影可以表示为这些基的线性组合。

系数的计算：投影的系数 $\mathbf{x}(t)$ 是通过内积 $< u^{\leq t}, g_n >_{\mu(t)}$ 计算得到的，其中g_n是正交多项式基的元素。

连续动态系统：为了在线更新这些系数，作者提出了一个连续动态系统，这个系统描述了系数$\mathbf{x}(t)$是如何随时间 t 变化的。这种动态系统可以表示为 $\frac{d}{dt}\mathbf{x}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)u(t)$，其中 $\mathbf{A}(t)$、$\mathbf{B}(t)$ 是依赖于时间的矩阵。

投影操作符：作者定义了一个投影操作符 proj_t，它将 $u^{\leq t}$ 映射到 G（多项式空间）中的 $g(t)$，使得 $||u^{\leq t} - g(t)||_{L^2(\mu(t))}$ 最小化。这个操作符是 HiPPO 框架的核心。

系数提取操作符：除了投影操作符，作者还定义了一个系数提取操作符 coef_t，它将多项式 $g(t)$ 映射到其对应的系数 $\mathbf{x}(t)$。

在线更新：通过这个连续动态系统，HiPPO 框架能够在线更新记忆表示，即随着新数据的到来，系统能实时地调整系数 $\mathbf{x}(t)$。

在线函数逼近

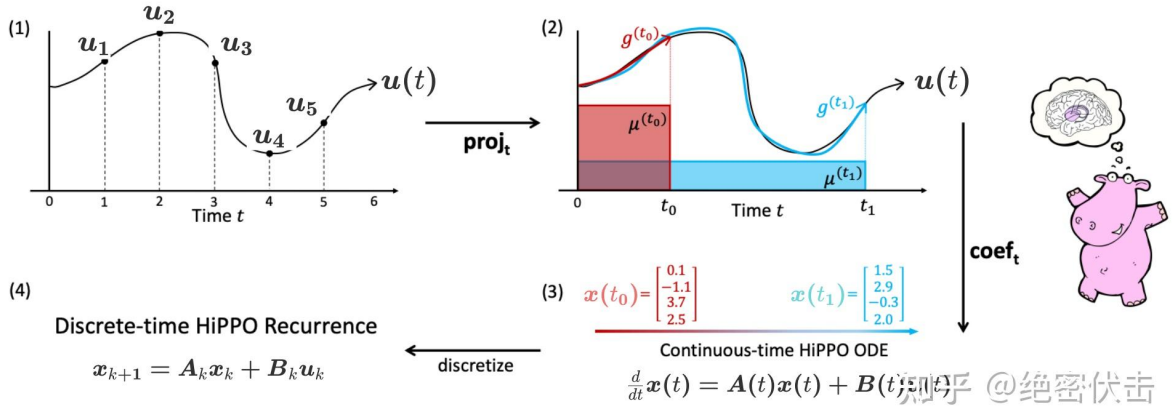


图2-6展示了 HiPPO 框架，首先需要找到投影 proj_t，将输入 $u(t)$ 投影到多项式空间；然后将投影通过一组系数$\mathbf{x}(t)$ 来表示，这些系数捕捉了函数 $\mathbf{u}(t)$ 的历史信息；使用连续时间下的一阶常微分方程来表示系数 $\mathbf{x}(t)$ 如何随时间 t 动态变化；最后，将连续时间的动态变换转化为离散时间的递归关系（比如双线性变换），这允许 HiPPO 在每个时间步 k 更新系数 $\mathbf{x}_k$。

3.1.3 高阶投影：度量方法以及 HiPPO 动态系统

作者定义了两种度量方法，分别是 LegT 和 LagT。LegT 度量为最近的历史信息分配均匀的权重，表示如下：

$$\mathbf{LegT}: \mu^{(t)}(x) = \frac{1}{\theta}\mathbb{I}_{[t-\theta,t]}(x) \tag{7}$$

LagT 度量使用指数衰减的方式来衡量历史信息的重要性，表示如下：

$$\mathbf{LagT}: \mu^{(t)}(x) = e^{-(t-x)}\mathbb{I}_{[-\infty,t]}(x) = \begin{cases} e^{x-t} & \text{if } x \le t \\ 0 & \text{if } x > t \end{cases} \tag{8}$$

对于 LegT 和 LagT，系数 $\mathbf{x}(t)$ 可以使用 ODE（一阶常微分方程）来表示：

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t) \tag{9}$$

其中 A 和 B 是与度量 $\mu(t)$ 相关的矩阵。这个 ODE 描述了系数$\mathbf{x}(t)$如何随时间 t 和输入函数 $\mathbf{u}(t)$ 变化。

备注：公式(9)是 HiPPO 框架的关键部分，**具体推导可以参看论文中的附录 D。**

对于 LegT 度量，矩阵 A 和 矩阵 B 可以表示如下：

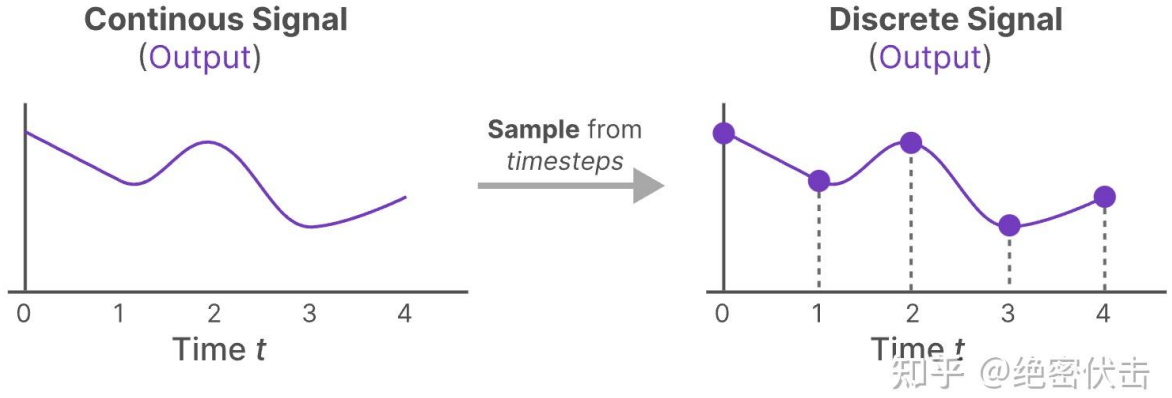
$$\mathbf{LegT}: A_{nk} = -\frac{1}{\theta}\begin{cases} (-1)^{(n-k)}(2n+1) & \text{if } n \ge k \\ 2n+1 & \text{if } n > k \end{cases}, B_n = \frac{1}{\theta}(2n+1)(-1)^n \tag{10}$$

对于 LagT 度量，可以表示如下：

$$\mathbf{LagT}: A_{nk} = -\frac{1}{\theta}\begin{cases} 1 & \text{if } n \ge k \\ 0 & \text{if } n > k \end{cases}, B_n = 1 \tag{11}$$

3.1.4 HiPPO 框架中的连续时间动态转换为离散时间递归关系

由于我们处理的输入往往是离散的，因此我们需要将公式（9）的 ODE 离散化。ODE 离散化是一种常用的数据技术，它将**连续时间的常微分方程转换为离散时间的差分方程。**这通常涉及到选择一个合适的时间步长（或步长Δt），并使用数值方法（如欧拉方法、双线性）来近似连续微分。



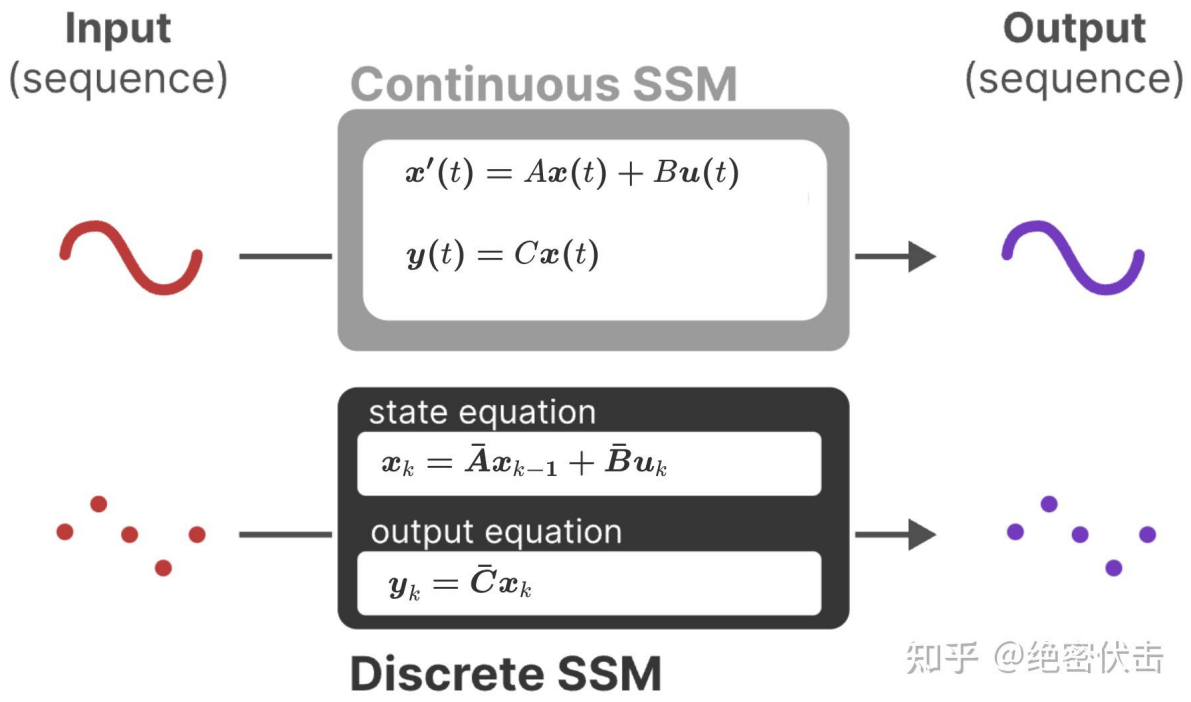
使用双线性离散化，如下所示：

$$\begin{aligned} x(t+\Delta t) - \frac{\Delta t}{2}Ax(t+\Delta t) &= (I + \Delta t/2A)x(t) + \Delta t Bu(t) \\ x(t+\Delta t) &= (I - \Delta t/2A)^{-1}(I + \Delta t/2A)x(t) + \Delta t/2A)^{-1}Bu(t) \end{aligned} \tag{12}$$

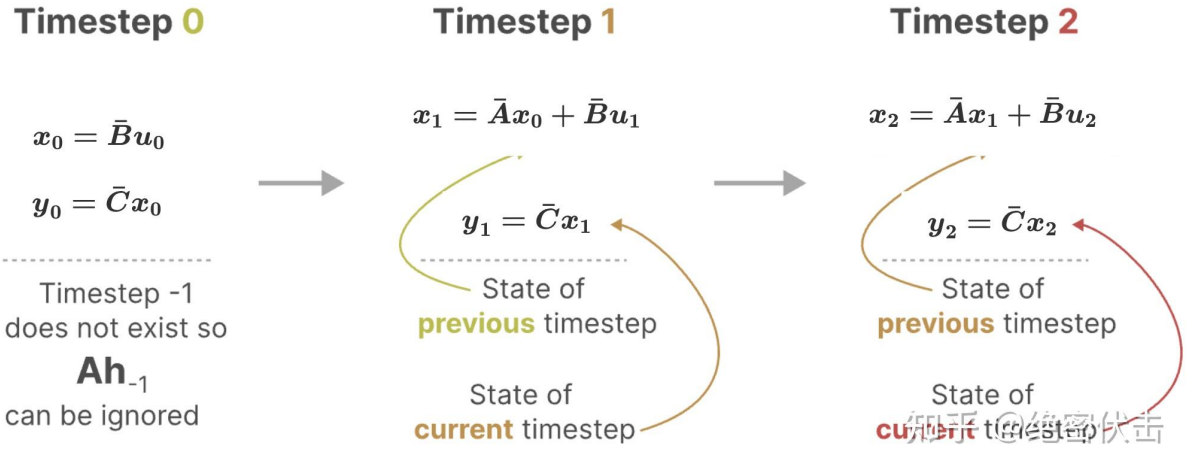
结合公式(9)和公式(12)，我们可以得到离散化的状态更新公式，表示如下：

$$\begin{aligned} \mathbf{x}_k &= \bar{\mathbf{A}}\mathbf{x}_{k-1} + \bar{\mathbf{B}}\mathbf{u}_k, \quad \bar{\mathbf{A}} = (I - \Delta/2 \cdot \mathbf{A}) \\ y_k &= \bar{\mathbf{C}}\mathbf{x}_k \quad \bar{\mathbf{B}} = (I - \Delta/2 \cdot \mathbf{A})^{-1}\mathbf{B} \quad \bar{\mathbf{C}} = \mathbf{C} \end{aligned} \tag{13}$$

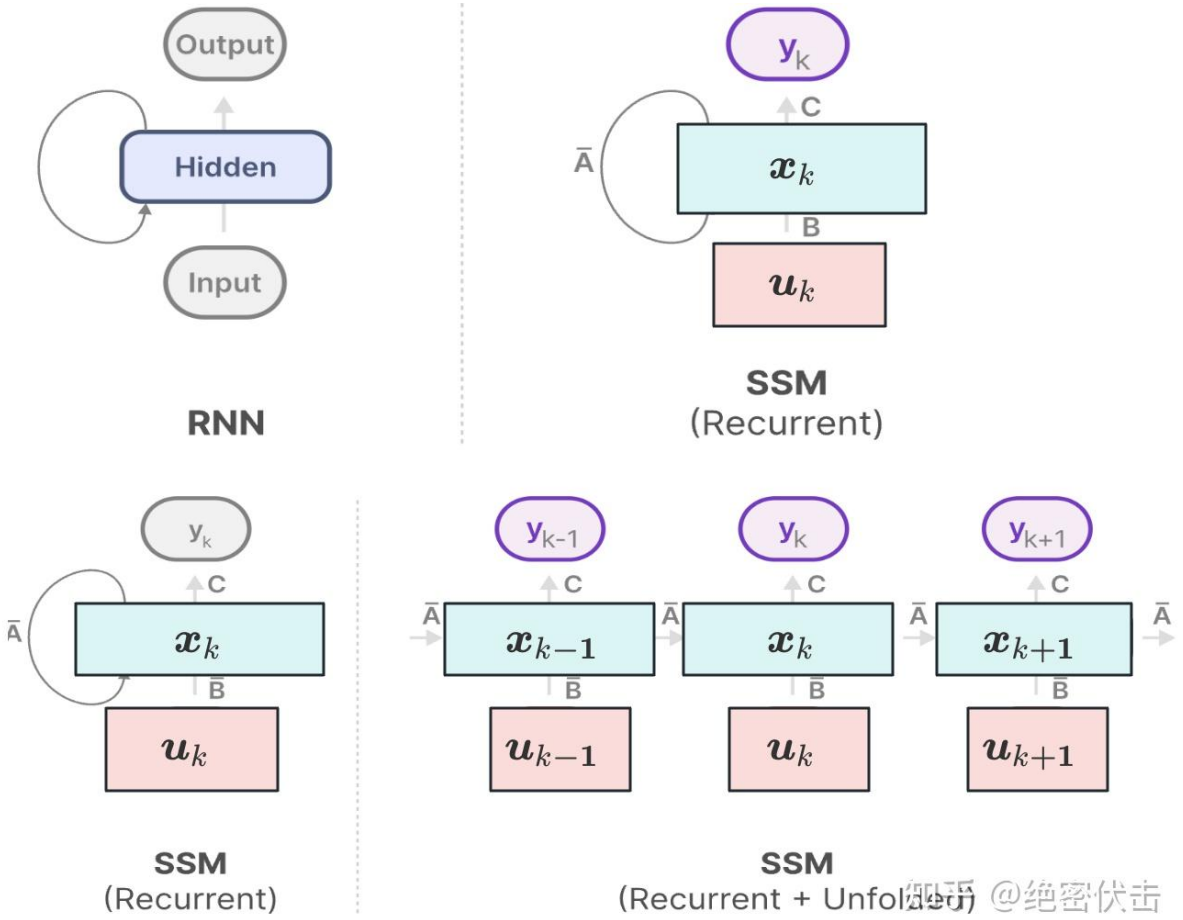
离散化之后的 SSM 结构可以表示如下：



在每个时间步长，我们计算当前输入$\bar{\mathbf{B}}\mathbf{u}_k$如何影响前一个状态$\bar{\mathbf{A}}\mathbf{x}_{t-1}$，然后计算预测输出$\bar{\mathbf{C}}\mathbf{x}_k$。



这种表示看起来是不是有点熟悉？其实他的处理方法和RNN一样。



3.2 HiPPO-LegS

HiPPO-LegS 是作者基于新的度量提出的全新架构，具有时间鲁棒性、有界梯度、有界近似误差、长时间记忆等效果。

...

四、S4 (Structured State Space Model)

S4 是 HiPPO 的后续工作，论文名称为：Efficiently Modeling Long Sequences with Structured State Spaces。

S4 的主要工作是将 HiPPO 中的矩阵 A（称为 **HiPPO 矩阵**）转换为正规矩阵（正规矩阵可以分解为对角矩阵）和低秩矩阵的和，以此提高计算效率。

S4 通过这种分解，将计算复杂度降低到了$O(N+L)$，其中 N 是 HiPPO 矩阵的维度，L 是序列长度。

在处理长度为 16000 的序列的语音分类任务中，S4 模型将专门设计的语音卷积神经网络（Speech CNNs）的测试错误率降低了一半，达到了1.7%。相比之下，所有的循环神经网络（RNN）和 Transformer 基线模型都无法学习，错误率均在70%以上。

下面我们就来介绍一下这篇工作。

4.1 HiPPO 解决了长期依赖

作者讨论了如何处理长距离依赖（Long-Range Dependencies，LRDs）的问题，LRDs 是序列建模中的一个关键挑战，因为它们涉及到在序列中跨越大量时间步的依赖关系。

作者指出，基本的 SSM 在实际应用中表现不佳，特别是在处理 LRDs 时。这是因为线性一阶常微分方程（ODEs）的解通常是指数函数，这可能导致梯度在序列长度上呈指数级增长，从而引发梯度消失或爆炸的问题。

为了解决这个问题，作者利用了 HiPPO 理论。**HiPPO 理论指定了一类特殊的矩阵 A，当这些矩阵被纳入 SSM 的方程中时，可以使状态 x(t) 能够记住输入 u(t) 的历史信息。这些特殊矩阵被称为 HiPPO 矩阵，它们具有特定的数学形式，可以有效地捕捉长期依赖关系。**

HiPPO 矩阵的一个关键特性是它们允许 SSM 在数学和实证上捕捉 LRDs。例如，通过将随机矩阵 A 替换为 HiPPO 矩阵，可以在序列 MNIST 基准测试上显著提高 SSM 的性能。

HiPPO 矩阵表示如下：

$$\textbf{HiPPO矩阵} \quad A_{nk} = -\begin{cases} (2n+1)^{1/2}(2kn+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases} \tag{16}$$

4.2 在线推理：使用递归形式

S4 在推理时，使用公式(13)的递归形式，每次只需要和上一个状态进行计算，具有和 RNN 相似的推理效率。

4.3 训练 S4：卷积表示

由于离散时间 SSM 的递归性质，它在硬件上进行训练时存在效率问题。因此，作者将离散时间 SSM 的**递归方程转换为离散卷积的形式。**通过展开递归方程，可以得到一个卷积核，这个卷积核可以用来在序列数据上应用卷积操作。这种转换允许 SSM 利用快速傅里叶变换（FFT）等高效的卷积计算方法，从而在训练过程中提高计算效率。

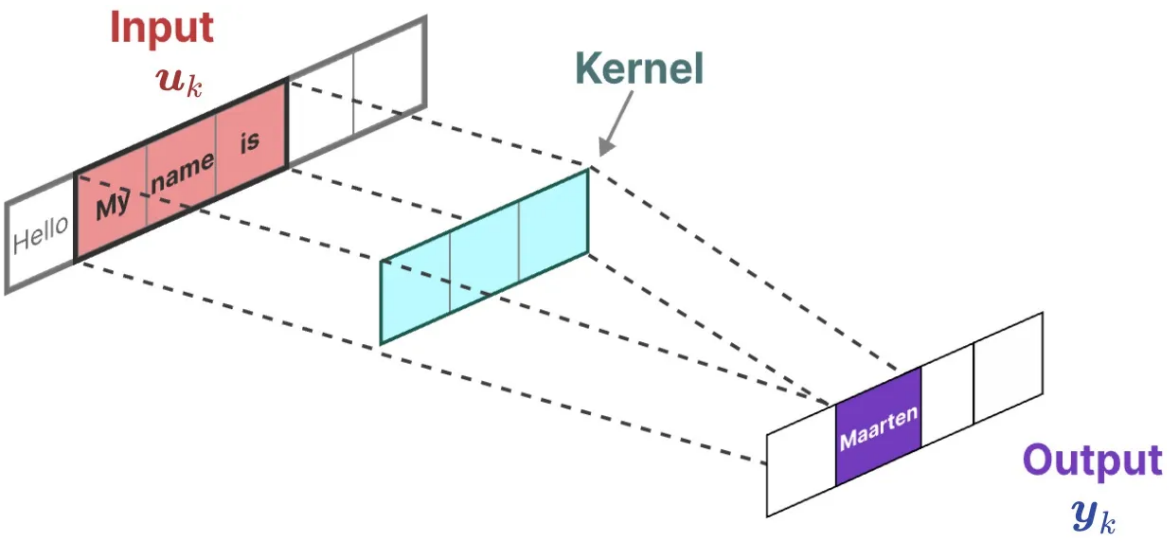
$$\begin{aligned} &\mathbf{x}_0 = \bar{\mathbf{B}}u_0 \quad \mathbf{x}_1 = \bar{\mathbf{A}}\bar{\mathbf{B}}u_0 + \bar{\mathbf{B}}u_1 \quad \mathbf{x}_2 = \bar{\mathbf{A}}^2\bar{\mathbf{B}}u_0 + \bar{\mathbf{A}}\bar{\mathbf{B}}u_1 + \bar{\mathbf{B}}u_2 \quad \dots \\ &\mathbf{y}_0 = \bar{\mathbf{C}}\bar{\mathbf{B}}u_0 \quad \mathbf{y}_1 = \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}u_0 + \bar{\mathbf{C}}\bar{\mathbf{B}}u_1 \quad \mathbf{y}_2 = \bar{\mathbf{C}}\bar{\mathbf{A}}^2\bar{\mathbf{B}}u_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}u_1 + \bar{\mathbf{C}}\bar{\mathbf{B}}u_2 \quad \dots \end{aligned} \tag{17}$$

上面式子可以转化为卷积的形式：

$$\begin{aligned} &\mathbf{y}_k = \bar{\mathbf{C}}\bar{\mathbf{A}}^k\bar{\mathbf{B}}u_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}^{k-1}\bar{\mathbf{B}}u_1 + \ldots + \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}u_{k-1} + \bar{\mathbf{C}}\bar{\mathbf{B}}u_k \\ &\mathbf{y} = \bar{\mathbf{K}} * u \\ &\bar{\mathbf{K}} \in \mathbb{R}^L := \mathcal{K}_L(\bar{\mathbf{A}}, \bar{\mathbf{B}}, \bar{\mathbf{C}}) = (\bar{\mathbf{C}}, \bar{\mathbf{A}}^{\mathbf{i}}, \bar{\mathbf{B}})_{i\in[L]} = (\bar{\mathbf{C}}\bar{\mathbf{B}}, \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}, \ldots, \bar{\mathbf{C}}\bar{\mathbf{A}}^{\mathbf{L-1}}\bar{\mathbf{B}}) \end{aligned} \tag{18}$$

其中，$\bar{\mathbf{K}}$ 是一个与 SSM 的参数（A, B, C）相关的卷积核，可以通过离散傅里叶变换（DFT）和逆变换（IDFT）来计算。这种卷积表示不仅在理论上是可行的，而且在实践中也是非常有效的，因为它允许在保持模型性能的同时，显著减少训练过程中的计算和内存需求。

作者在这一节中还讨论了如何计算 SSMn卷积核，这是他们技术贡献的关键部分。通过这种卷积表示，SSM 可以被有效地训练，同时保持其在处理长距离依赖（LRDs）方面的能力。这种表示形式为 SSM 在各种序列建模任务中的应用提供了灵活性，包括图像处理、语音识别和时间序列分析等。

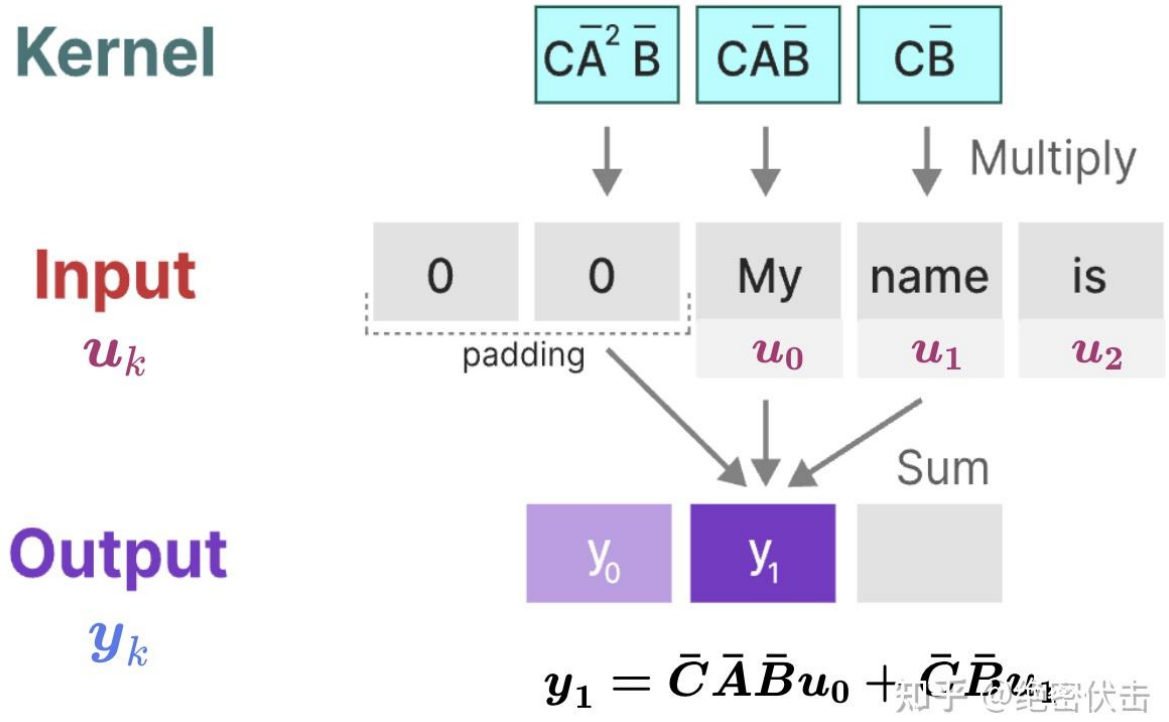


$$\text{kernel} \longrightarrow \overline{\mathbf{K}} = (\overline{\mathbf{CB}}, \overline{\mathbf{CAB}}, \ldots, \overline{\mathbf{CA}}^{\overline{k}}\overline{\mathbf{B}}, \ldots)$$

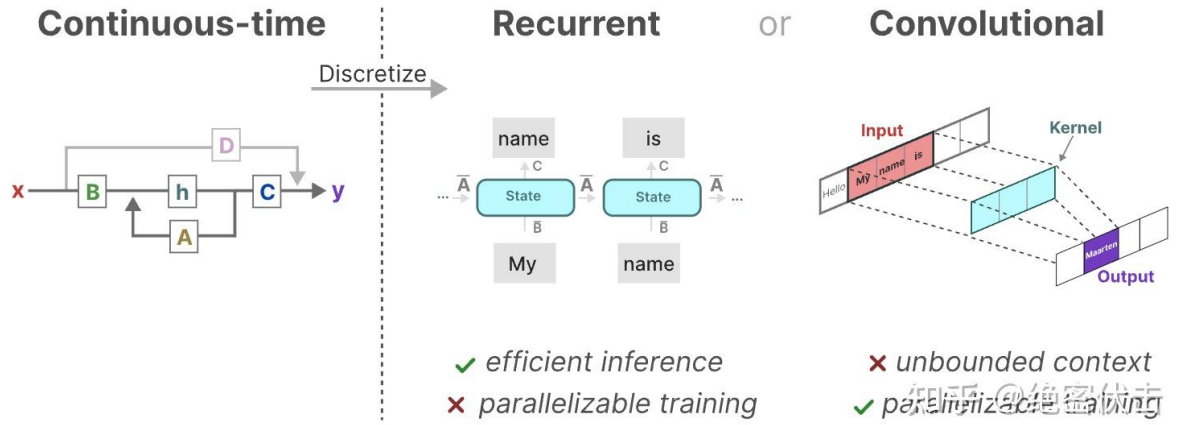
$$\mathbf{y} = \mathbf{u} * \overline{\mathbf{K}}$$

output input kernel

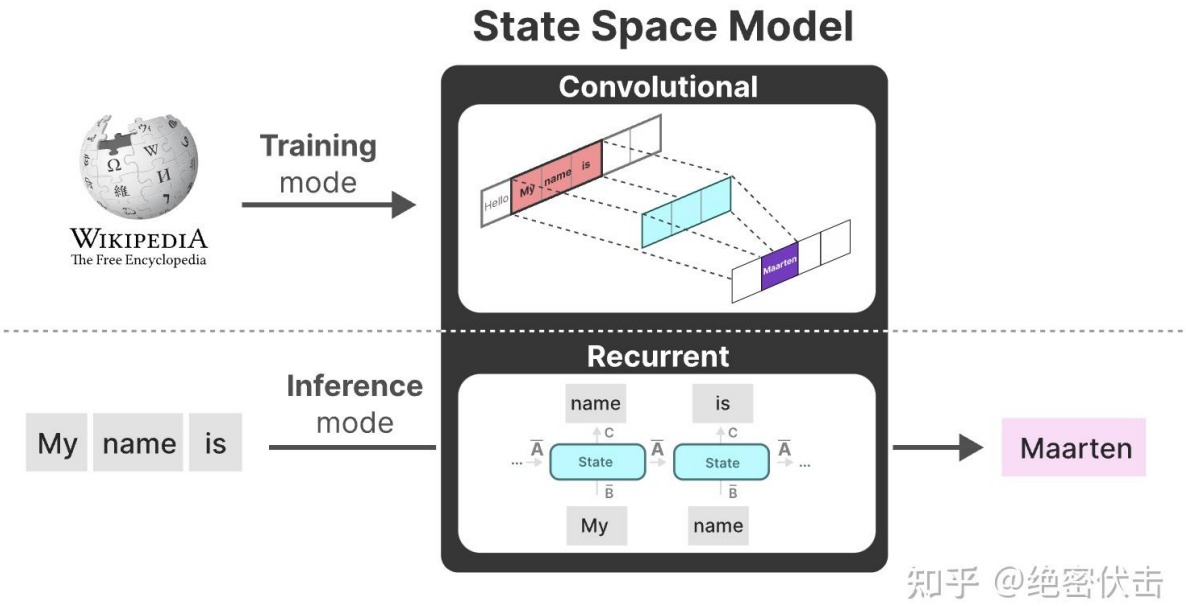
下面是一个具体的例子，如何使用卷积核生成输出。



卷积的一个主要好处是它可以并行训练。但是由于核大小是固定，它们的推理不如 RNN 快速并且对序列长度有限制。



这里可以使用一个简单的技巧，即根据任务选择表示。在训练过程中使用可以并行化的卷积表示，在推理过程中，我们使用高效的循环表示。



4.4 为什么对角化可以减少 SSM 计算复杂度

为了进一步提升计算效率，作者讨论了对角化在计算离散时间状态空间模型（SSM）中的应用，以及为什么直接应用对角化方法在实践中并不可行。

对角化是一种线性代数技术，它可以将一个矩阵转换为对角形式，从而简化矩阵的乘法和其他运算。在 SSM 的上下文中，对角化可以显著减少计算复杂度，因为对角矩阵的幂运算（如在递归方程中出现的）可以通过简单的元素指数运算来完成。

下面我们解释下，为什么对角化可以减少 SSM 计算复杂度。

首先，我们引入论文中的定理 3.1

...

五、Mamba

我们终于介绍完了理解 Mamba 所需要的基础知识。状态空间模型可用于建模文本序列，但仍有一系列我们想要避免的缺点。

在本节中，我们将介绍 Mamba 的两大主要贡献：

1. 一种选择性扫描算法，该算法允许模型过滤（不）相关信息；

2. 一种硬件感知算法，该算法允许通过并行扫描、内核融合和重新计算来高效存储（中间）结果。

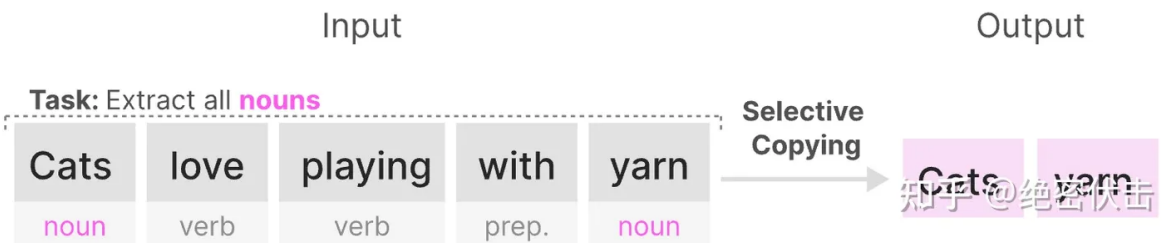
它们共同创建了选择性 SSM 或 S6 模型，这些模型可以像自注意力一样用于创建 Mamba 块。

在探讨这两大主要贡献之前，让我们首先探讨一下为什么它们是必要的。

状态空间模型，甚至是S4（结构化状态空间模型），在某些对语言建模和生成至关重要的任务上表现不佳，即关注或忽略特定输入的能力。

我们可以通过两个合成任务来说明这一点，即选择性复制和归纳头。

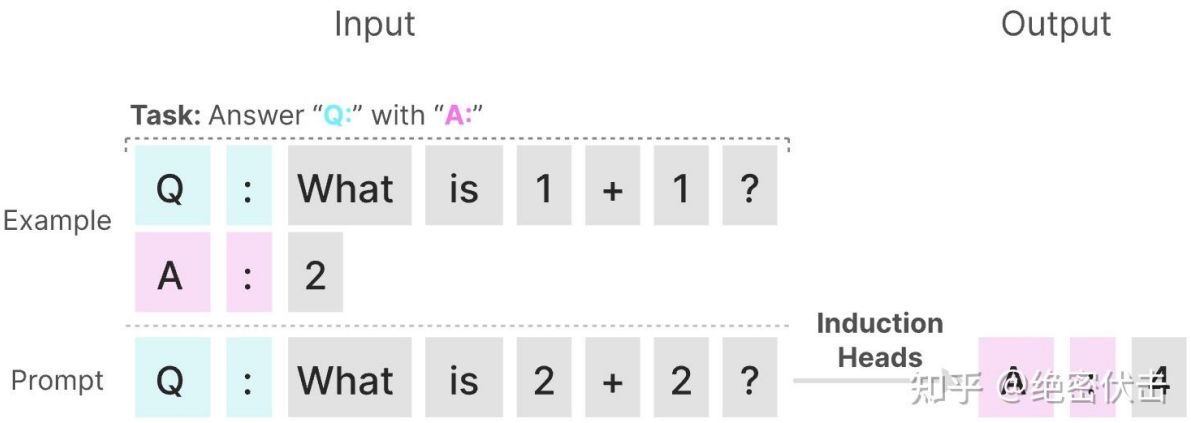
在选择性复制任务中，SSM 的目标是复制输入的部分内容并按顺序输出它们：



然而，由于（循环/卷积）SSM 是线性时间不变的，因此在这项任务中表现不佳。正如我们之前看到的，对于 SSM生成的每个 token，矩阵 A、B 和 C 都是相同的。

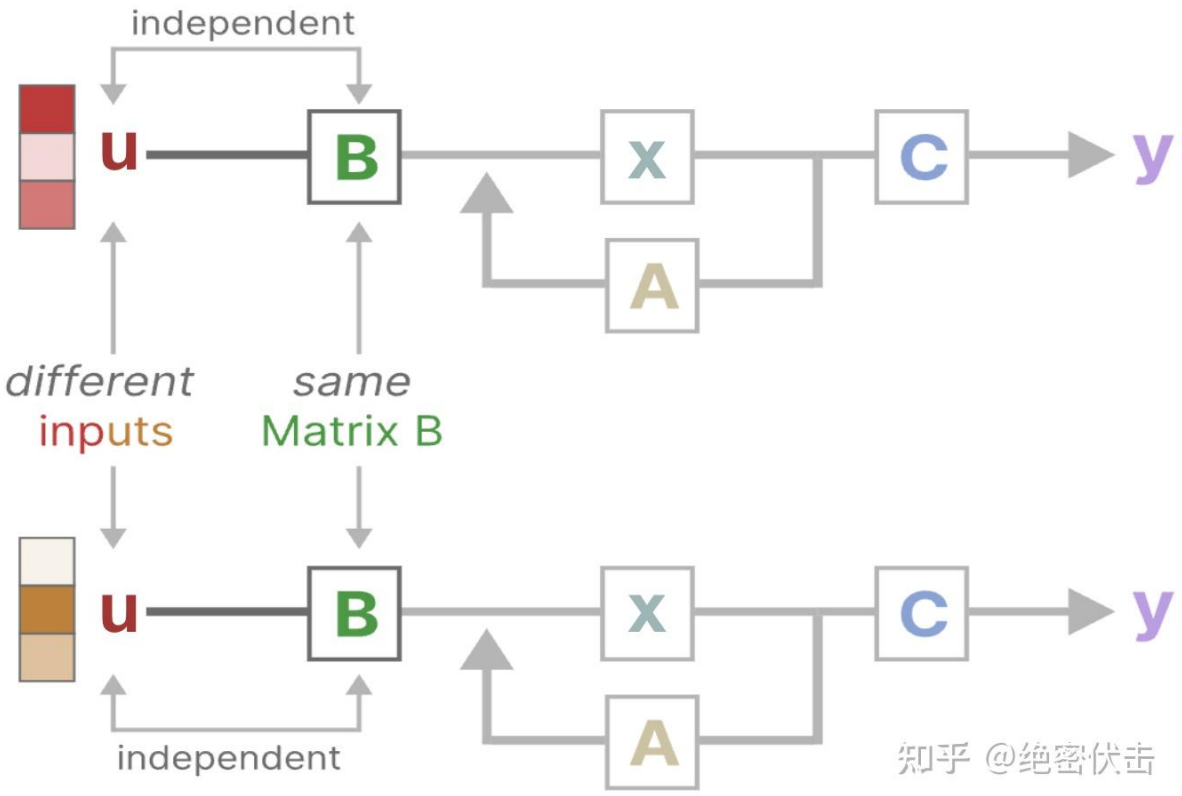
因此，由于固定的 A、B 和 C 矩阵，SSM 无法执行内容感知推理，因为它对每个 token 都一视同仁。这是一个问题，因为我们希望 SSM 能对输入（提示）进行推理。

SSM 表现不佳的第二项任务是归纳头，其目标是重现输入中发现的模式：

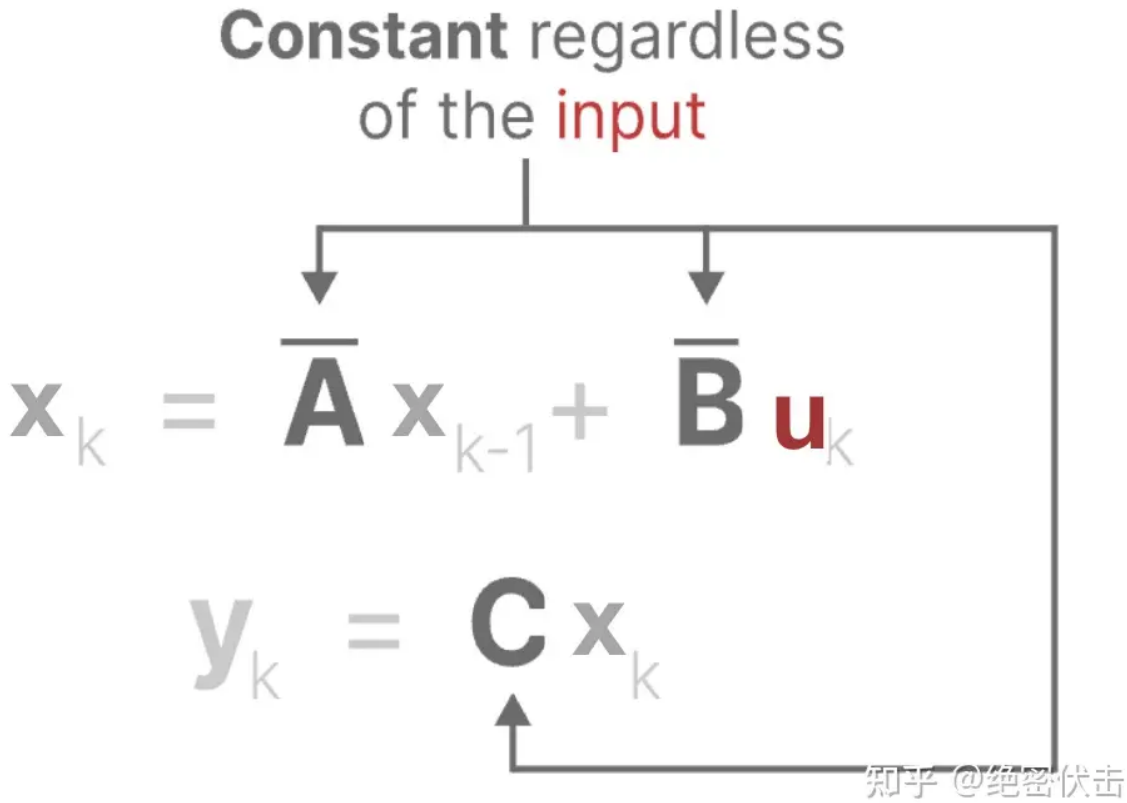


在上面的例子中，我们本质上是在执行一次提示，我们试图"教"模型在每个"Q:"之后提供一个"A:"的回应。然而，由于 SSM 是时间不变的，它无法选择从历史中回忆哪些之前的 token。

让我们通过关注矩阵 B 来说明这一点。无论输入 u 是什么，矩阵 B 都保持不变，因此与 u 无关：



同理，无论输入是什么，A 和 C 也不变，这就是我们上面说的静态。



相比之下，这些任务对于 Transformer 来说相对容易，因为它们会根据输入序列动态地改变自己的注意力。它们可以选择性地“查看”或“关注”序列的不同部分。

SSM 在这些任务上的糟糕表现说明了时间不变 SSM 的潜在问题，即矩阵 A、B 和 C 的静态性质导致内容感知方面的问题。

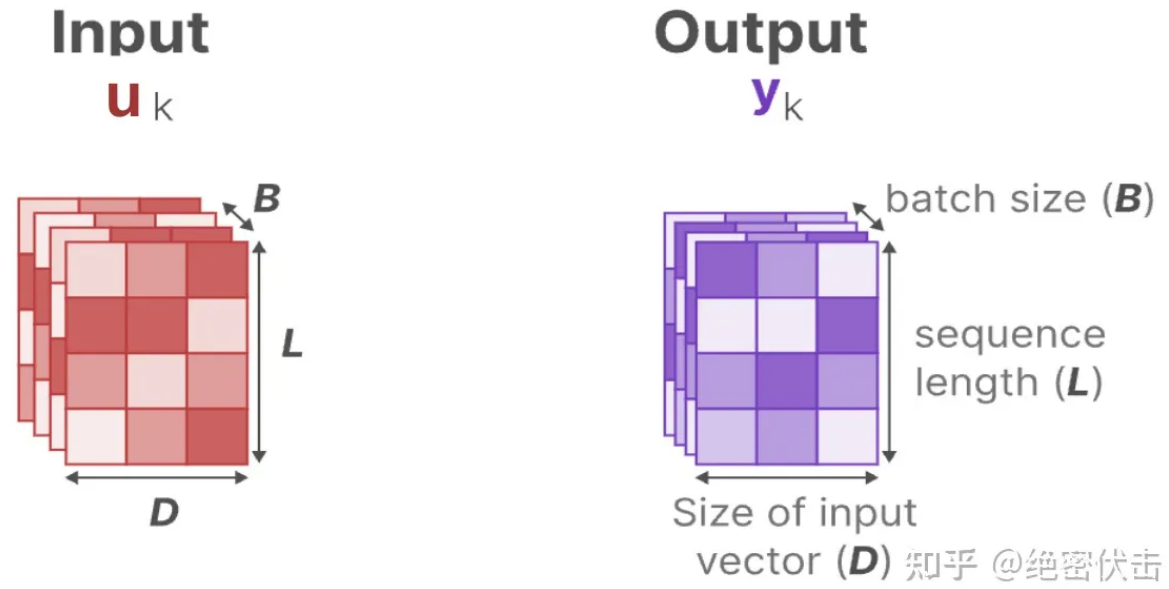
5.1 通过选择机制改进 SSM

为了解决上面的问题，作者提出了一种新的选择性 SSM（Selective State Space Models，简称 S6 或 Mamba）。这种模型通过让 SSM 的矩阵 A、B、C **依赖于输入数据**，从而实现了选择性。这意味着模型可以根据当前的输入动态地调整其状态，选择性地传播或忽略信息。

Mamba 集成了 S4 和 Transformer 的精华，一个更加高效（S4），一个更加强大（Transformer）。

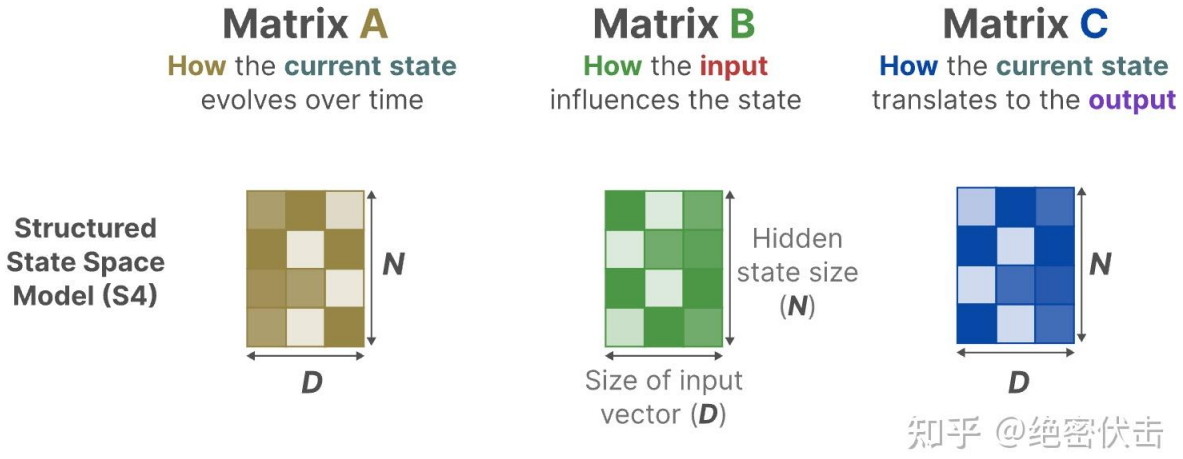
正如上面所提到的，它是通过有选择地将数据压缩到状态中来实现的。当你有一个输入句子时，通常会有一些信息，比如停用词，没有太多意义。

为了有选择地压缩信息，我们需要让参数依赖于输入。为此，我们首先来探讨一下 SSM 在训练过程中输入和输出的维度。



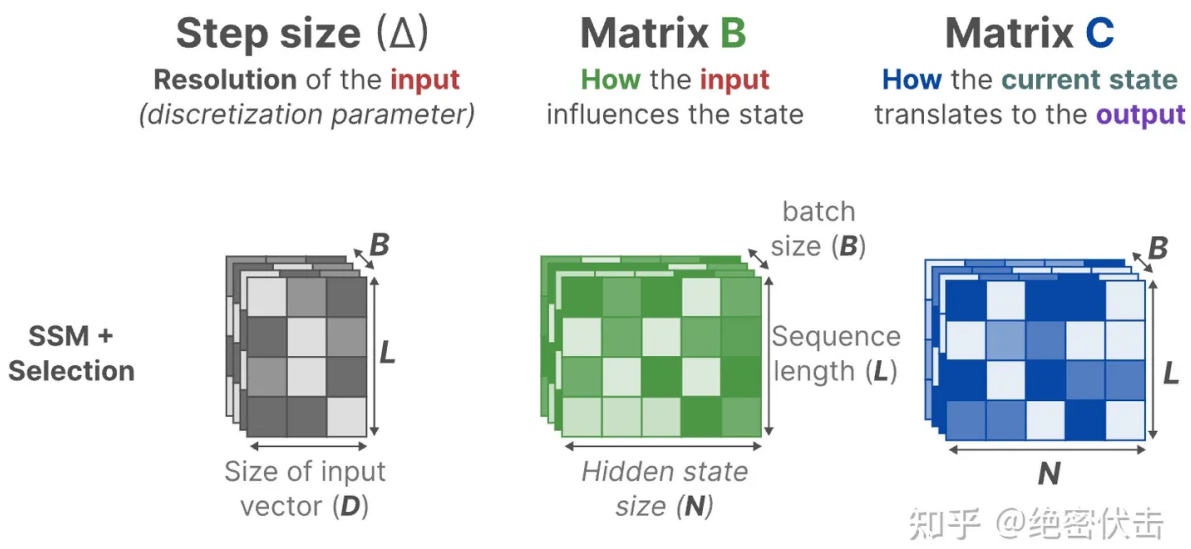
备注：在前面的 HiPPO 和 S4 中，我们假设的输入信号 $u(t)$ 是 1 维的，而实际应用中大多数都是多维的，后面我们默认是多维输入（默认维度为 D）。**而且需要强调的是 S4 用的是 Single-input-single-output (SISO)，即对应于每一个输入的维度，都有一套独立的 SSM 参数 （传统的 RNN 是 MIMO, multiple-input-multiple-output, 很容易混淆)。**

在 S4 中，矩阵 A、B 和 C 与输入无关，因为它们的维度 N 和 D 是静态的，不会改变。



备注：实际上矩阵 $\mathbf{A} \in \mathbb{R}^{N \times N}$，但是通过对角化技术，可以转化为 N 维，因此图5-7中，每个输入维度对应的矩阵 A 维度为 N，输入一共是 D 维，因此矩阵 A 可以表示为 $N \times D$ 的矩阵。

相反，Mamba 通过将输入序列的长度和批次大小结合起来，使矩阵 B 和 C，甚至步长 Δ 都依赖于输入：

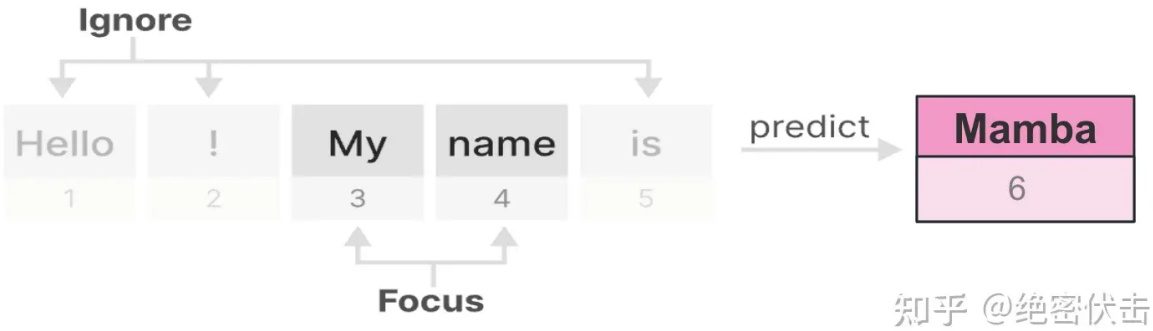


这意味着对于每个输入 token，我们现在有不同的 B 和 C 矩阵。

备注：这里矩阵 A 保持不变，因为我们希望状态本身保持静态，但影响它的方式 (通过 B 和 C) 是动态的。

它们一起选择性地决定在隐藏状态中保留什么和忽略什么，因为它们现在依赖于输入。

在 SSM 中，通过调整 Δ，模型可以控制对当前输入的关注度，从而实现类似于 RNN 门控的效果。例如，**当 Δ 较大时，模型倾向于关注当前输入并忽略之前的信息**；而当Δ较小时，模型则倾向于保留更多的历史信息：



下面我们看一下选择性 SSM 的完整过程，如下所示：

Algorithm 1 SSM (S4)

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

图 5-10 中的 Represents structured $N \times N$ matrix 说的就是原始 $N \times N$ 的矩阵 A 经过对角化之后变成维度 N

算法 2 展示了作者所使用的主要选择机制。这一套的思路由来已久，Transformers 里面的 QKV、LSTM 里面的、Gating 都是类似的思想。

S4 和 选择性 SSM 的核心区别在于，它们将几个关键参数(Δ, B, C)设定为输入的函数，并且伴随着整个 tensor 形状的相关变化。特别是，这些参数现在具有一个长度维度 L，这意味着模型已经从时间不变（time-invariant）转变为时间变化（time-varying）。其中：

$$\begin{aligned} S_B(x) &= \text{Linear}_N(x) \\ S_C(x) &= \text{Linear}_N(x) \\ S_\Delta(x) &= \text{Broadcast}_D(\text{Linear}_1(x)) \\ \tau_\Delta &= \text{softplus} \end{aligned} \tag{36}$$

Linear_d表示将输入映射到 d 维。S_Δ, τ_Δ 类似于 RNN 中的门控机制。

最后作者选择把A设成了与输入无关，作者给出的解释是离散化之后 $\bar{\mathbf{A}} = \exp(\Delta\mathbf{A})$，$\Delta$的数据依赖能够让整体的 $\bar{\mathbf{A}}$与输入相关。

5.2 选择性 SSM 和门控之间的关系

时间步Δ

时间步 Δ 和 RNN 的门控有很强的关联，**依赖输入的Δ跟 RNN 的遗忘门的功能类似。**

当 $N = 1, A = -1, B = 1, S_\Delta = \text{Linear}(x), \tau_\Delta = \text{softplus}$，那么算法 2 中的选择性 SSM 可以表示如下：

$$\begin{aligned} g_t &= \sigma(\text{Linear}(\mathrm{x_t})) \\ h_t &= (1 - g_t)h_{t-1} + g_t x_t \end{aligned} \tag{37}$$

可以看到这就是一个带门控的 RNN。

矩阵 B 和 C

在 SSM 中，修改 B 和 C 以使其具有选择性，允许模型更精细地控制是否让输入进入状态 h 或状态进入输出 y，所以 B 和 C 类似于 RNN 中的输入门和输出门。

矩阵 A

A 有点类似于起到多尺度/细粒度门控的作用。虽然Δ已经有点遗忘门的作用，但注意到对于每个输入维度来说，Δ只是一个标量，而$\mathbf{A} \in \mathbb{R}^{\mathbf{N}\times\mathbf{1}}$，也就是说对应这个维度的 SSM 来说，$\mathbf{A}$在每个 hidden state 维度上的作用可以不相同，起到细粒度门控的作用，这也是 LSTM 网络里面用 element-wise product 的原因（LSTM 中遗忘门是跟隐藏层维度相同的一个向量，而不仅仅是一个标量）。

5.3 Mamba 高效实现

因为现在的参数 $\mathbf{A}, \mathbf{B}, \mathbf{C}$ 都是输入相关了，所以不再是线性时间不变系统，也就失去了卷积的性质，不能用 FFT来进行高效训练了。

Mamba 作者采用了一种称为硬件感知的算法，实际上就是用三种经典技术来解决这个问题：**内核融合（kernel fusion）**、**并行扫描（parallel scan）**和**重计算（recomputation）**。

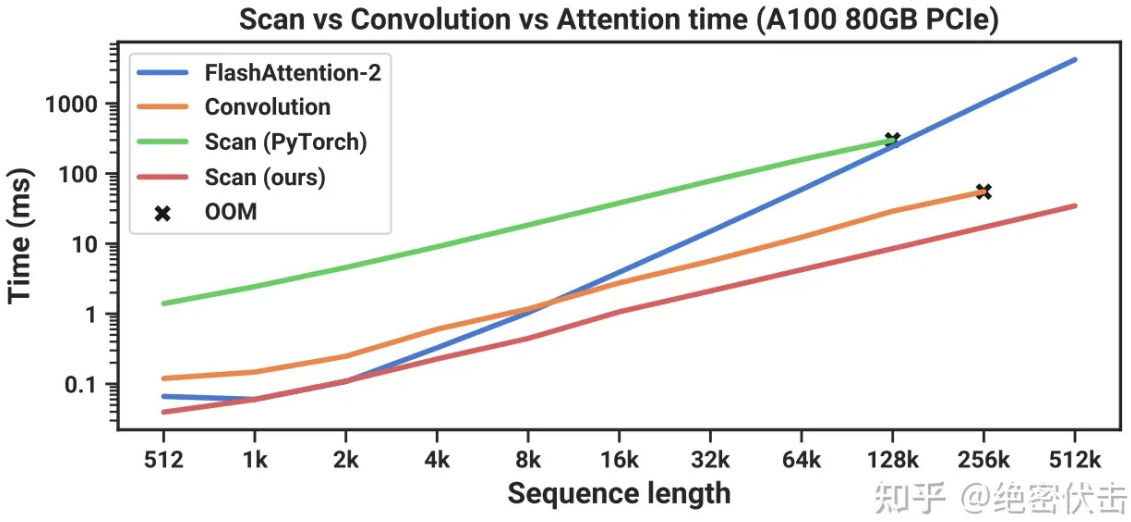
一般的实现会提前先把大小为 (B, L, D, N) 的 $\bar{\mathbf{A}}, \bar{\mathbf{B}}$先算出来，然后把它们从 HBM (high-bandwidth memory 或 GPU memory) 读到SRAM，然后调用 scan 算子算出 (B, L, D, N) 的 output，写到 HBM 里面。再开一个kernel 把 (B, L, D, N) 的 output 以及(B, L, N) 的 C 读进来，multiply and sum with C 得到最后的 (B, L, D) output 。整个过程的读写是 $O(BLDN)$。

而 Mambda 作者的方法是：

- 把 (Δ, A, B, C) 读到 SRAM 里面，总共大小是 $O(BLN + DN)$
- 在 SRAM 里面做离散化，得到(B, L, D, N)的$\bar{\mathbf{A}}, \bar{\mathbf{B}}$
- 在 SRAM 里面做 scan，得到(B, L, D, N)的 output

- multiply and sum with C，得到最后的(B, L, D)output 写入HBM

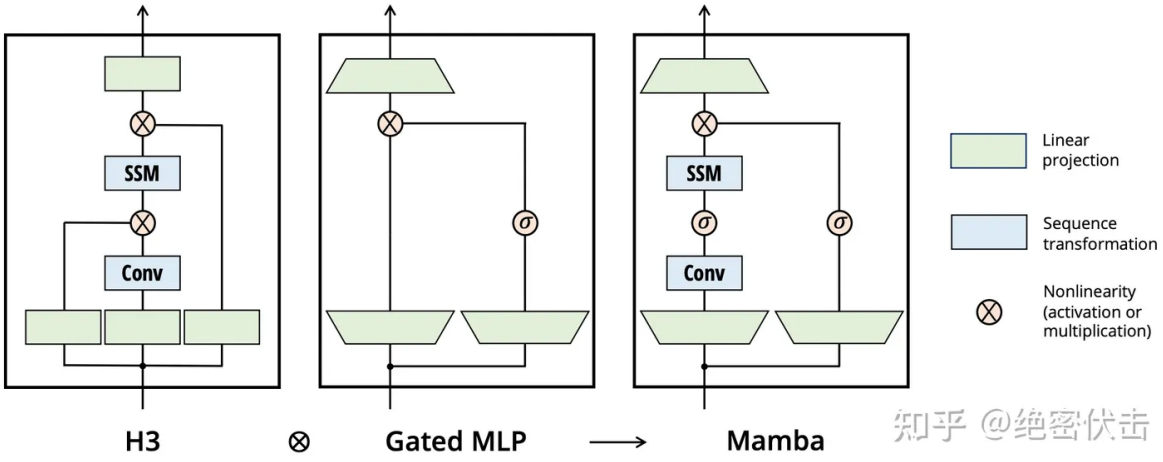
整个过程的总读写量是 $O(BLN)$，比之前省了 $O(N)$ 倍。 backward 的时候就把 $\bar{\mathbf{A}}, \bar{\mathbf{B}}$ 重算一遍，类似于flashattn 重算 attention 分数矩阵的思想。只要重算的时间比读 $O(BLND)$ 快就算有效。



Mamba 的实现比其它方法实现快很多倍，scan 在输入长度 2k 的时候就开始比 FlashAttention 快了，之后越长越快。同时 scan 也比 Convolution 快。

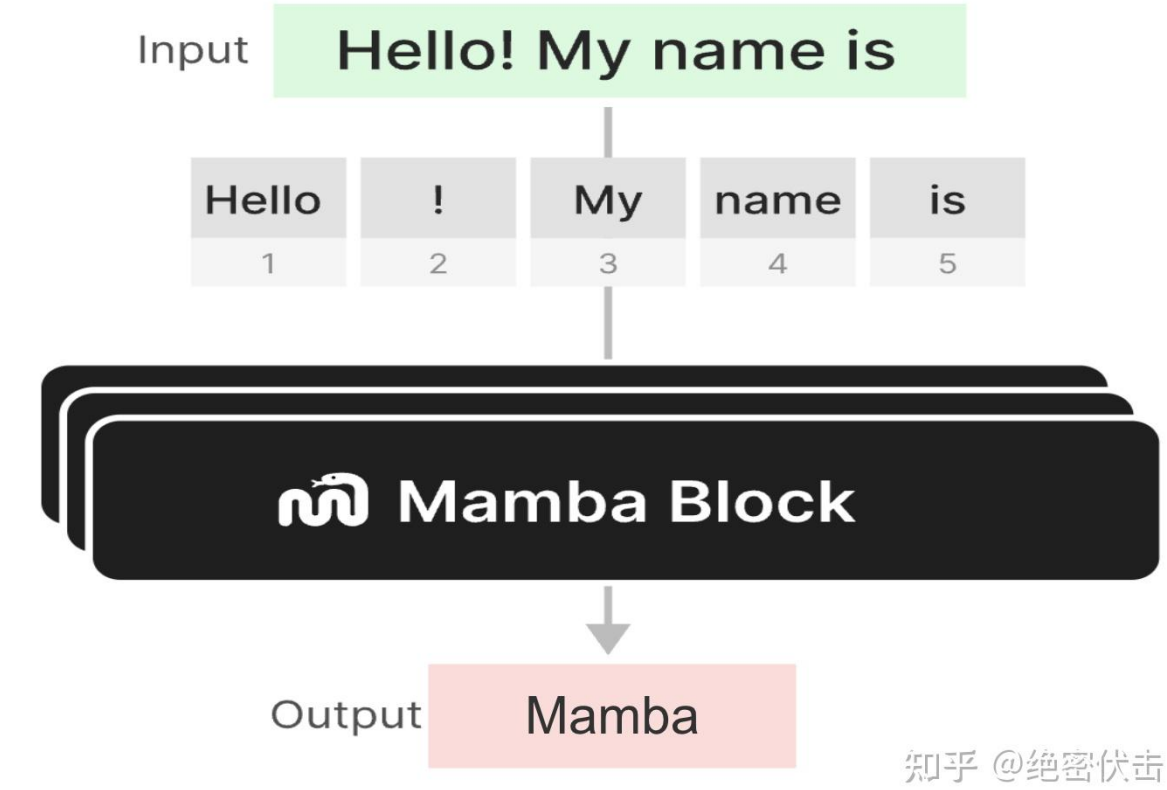
5.4 Mamba 架构

下图是 Mamba 的模型结构：

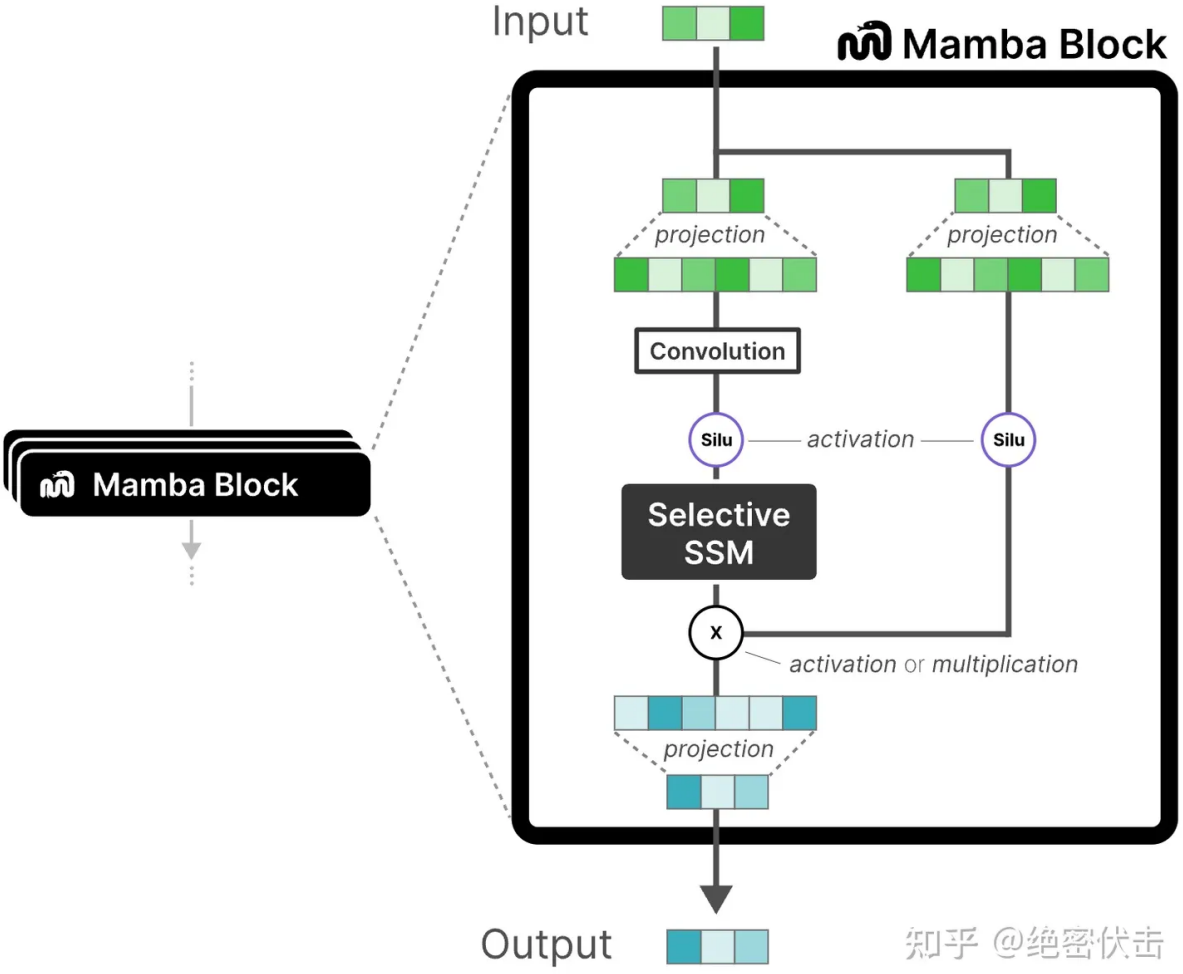


之前的 SSM 模型要 work，都会加上 output gating，之后再过个线性层 channel mixing，如上图的最左边所示。这两个部分跟 Gated MLP（上图中间）右边的支路和最上面的 channel mixing 是一样的。所以 SSM 层如果跟Gated MLP 合并的话，难免会感觉有点冗余，所以作者干脆把两个合二为一，把 token mixing 层和 channel mixing。

图 5-11 的 Mamba 可以作为一个块来实现，就像我们可以在解码器块中表示自注意力一样。



与解码器一样，我们可以堆叠多个 Mamba 块，并使用它们的输出作为下一个 Mamba 块的输入：

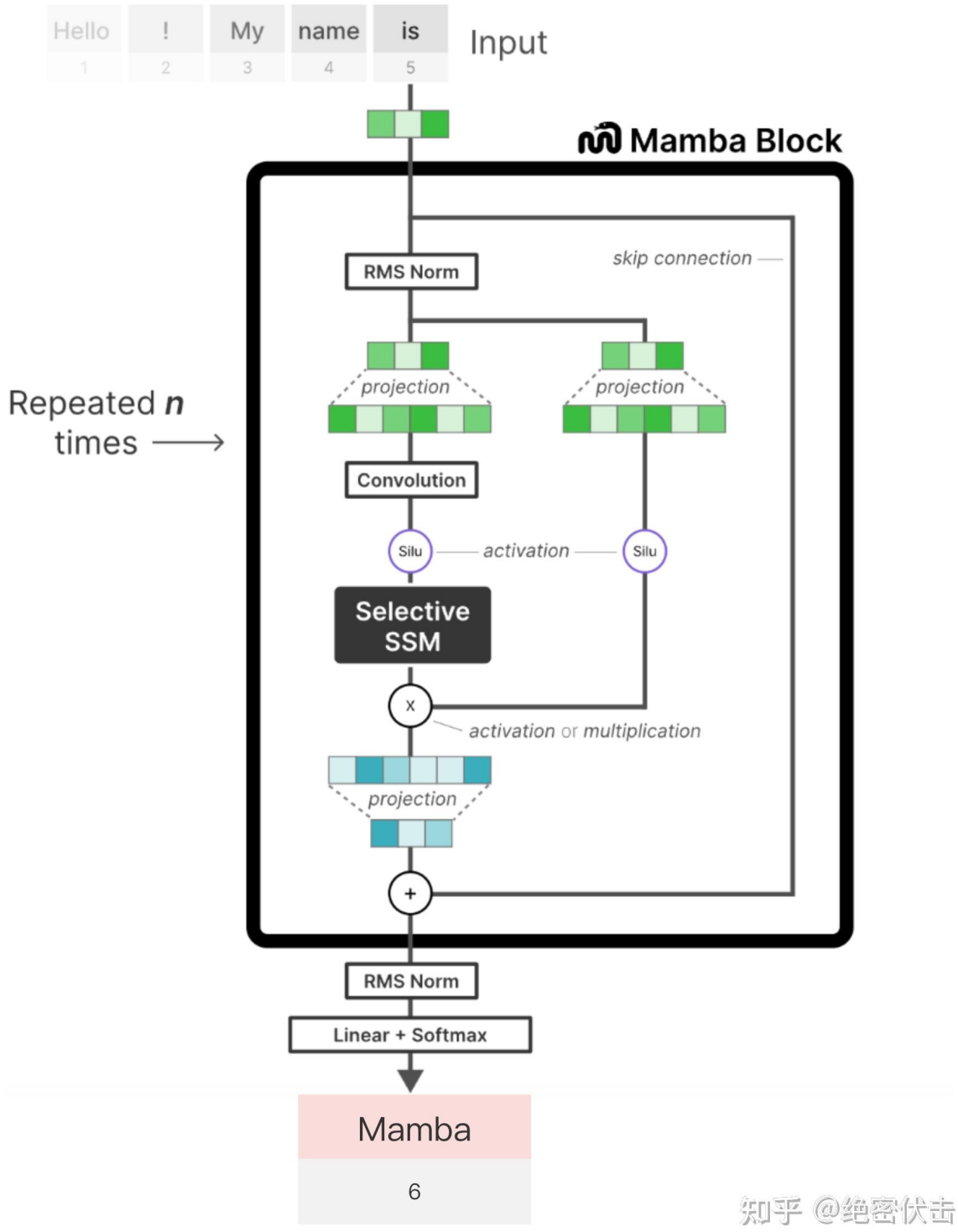


它首先进行线性投影以扩展输入 embedding。然后，在应用选择性 SSM 之前进行卷积。选择性 SSM 具有以下属性：

- 通过离散化创建递归 SSM；

- 对矩阵 A 进行 HiPPO 初始化，以捕获远程依赖关系；
- 选择性扫描算法，有选择地压缩信息；
- 硬件感知算法，加速计算；

下面是一个端到端（输入到输出）的例子：



下面我们看一下 Mamba 和 Transformer 以及 RNN 的对比：

	Training	Inference
Transformers	**Fast!** (parallelizable)	**Slow...** (scales **quadratically** with sequence length)
RNNs	**Slow...** (not parallelizable)	**Fast!** (scales **linearly** with sequence length)
Mamba	**Fast!** (parallelizable)	**Fast!** (scales **linearly** with sequence length + **unbounded** context)