Post LLM, **Online Function Approximation problem**은 어떻게 해결할 수 있는가? **RNN**에서 **Mamba2**까지

박택현

데이터사이언스전문대학원

pthpark1@pusan.ac.kr

Contents

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Albert Gu & Tri Dao

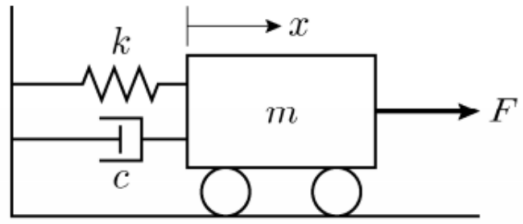
- **Christopher Ré**
  - **HazyResearch**
  - 피인용수: 50917 (04.29 기준)
  - 스텐포드 교수
  - 코넬대 컴공 97학번 (석박 워싱턴대)

- **Albert Gu**
  - 피인용수: 11725 (04.29 기준)
  - 카네기멜론 조교수 (23.08 ~)
  - 카네기멜론 컴공 11학번 (석박 스텐포드)
  - SSM

- **Tri Dao**
  - 피인용수: 13137 (04.29 기준)
  - 프린스턴 조교수 (24.06 ~)
  - 스텐포드 컴공 12학번 (학석박 스텐포드)
  - SSM, FlashAttention

What is State Space Model(SSM)?

- State Space Model 은 칼만 교수가 제안한 모델링 방식으로, 로봇, 자동차, 미사일과 같은 동적시스템을 행렬로 모델링하는 것. $\dot{x} = Ax + Bu$, $y = Cx + Du$ 형태로 시스템을 표현한다.
- 최근 이러한 SSM 기반 딥러닝 모델(RWKV, RetNet, Mamba,Hyena)들이 각광받기 시작했음.



전기기사 2019년 기출

$$m\ddot{x} + c\dot{x} + kx = F$$

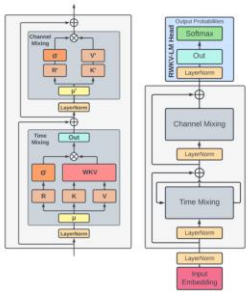
$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F$$

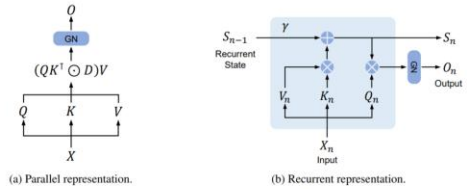
상태 방정식
(State Equation)

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

출력 방정식
(Output Equation)

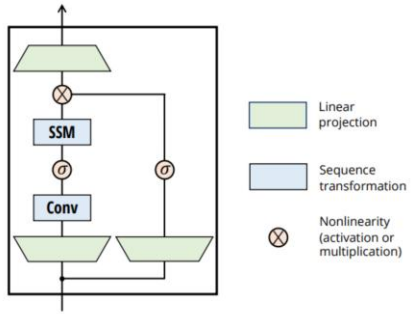


RWKV(EMNLP)



(a) Parallel representation. (b) Recurrent representation.

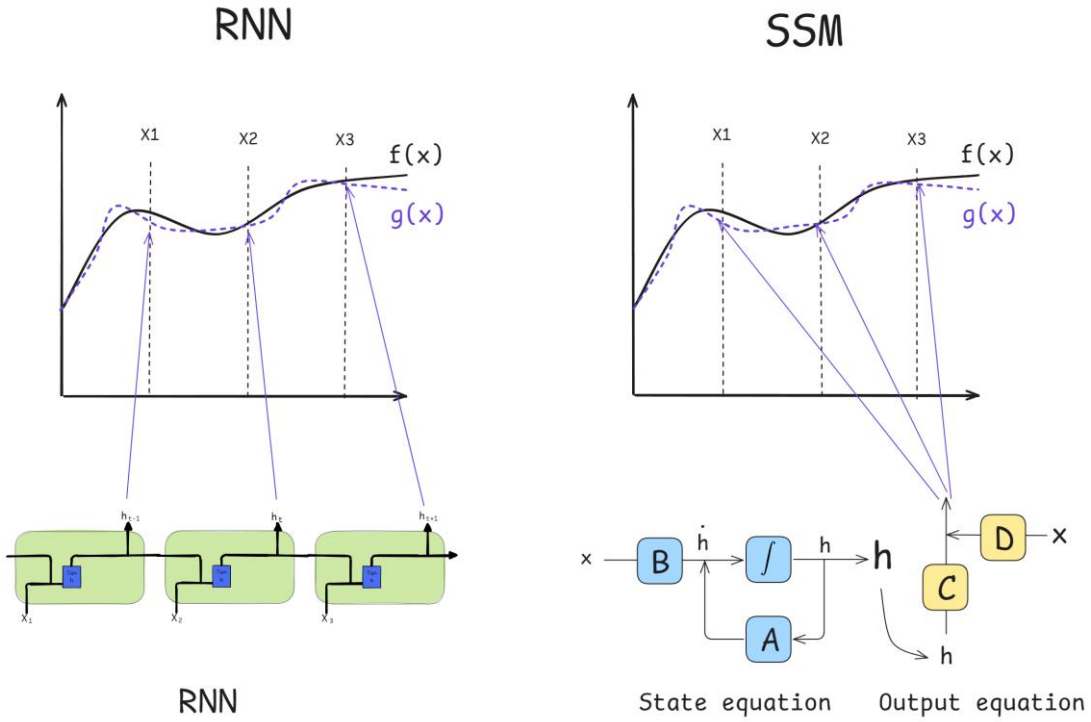
RetNet(ICLR)



Mamba(ICML)

What is State Space Model(SSM)?

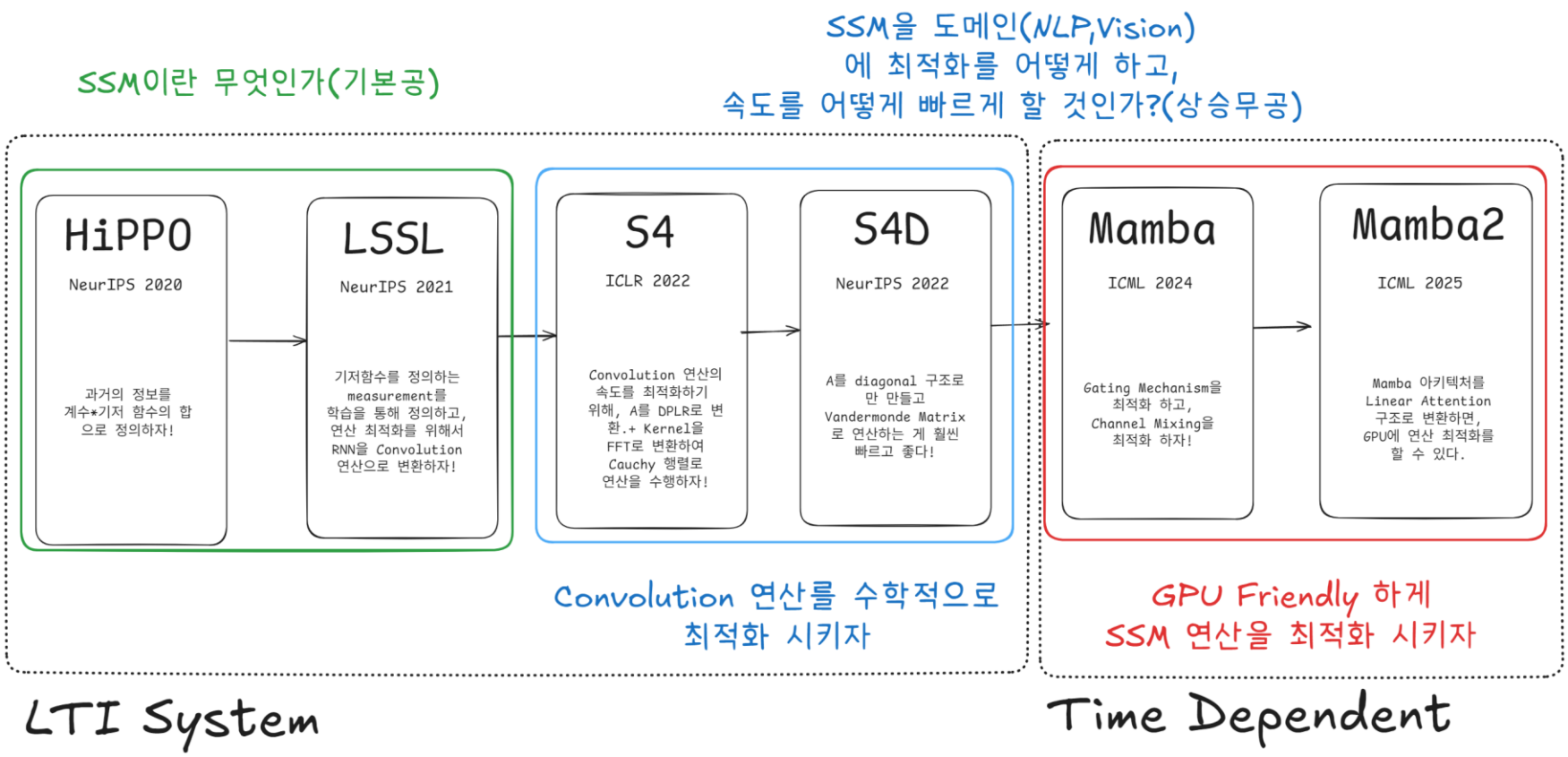
- RNN 계열은 기존의 현실세계 함수 f를 잘 근사화 하는 함수 g를 찾는 게 목적.
- SSM 계열은 과거 $f_{\leq t}$을 압축하는 **Online Approximation Vector**를 잘 만드는 State Equation을 만들고, 이를 기반으로 값을 잘 예상하는 출력 함수(Output Equation)을 통한 예측이 목적.



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Evolution of State Space Model

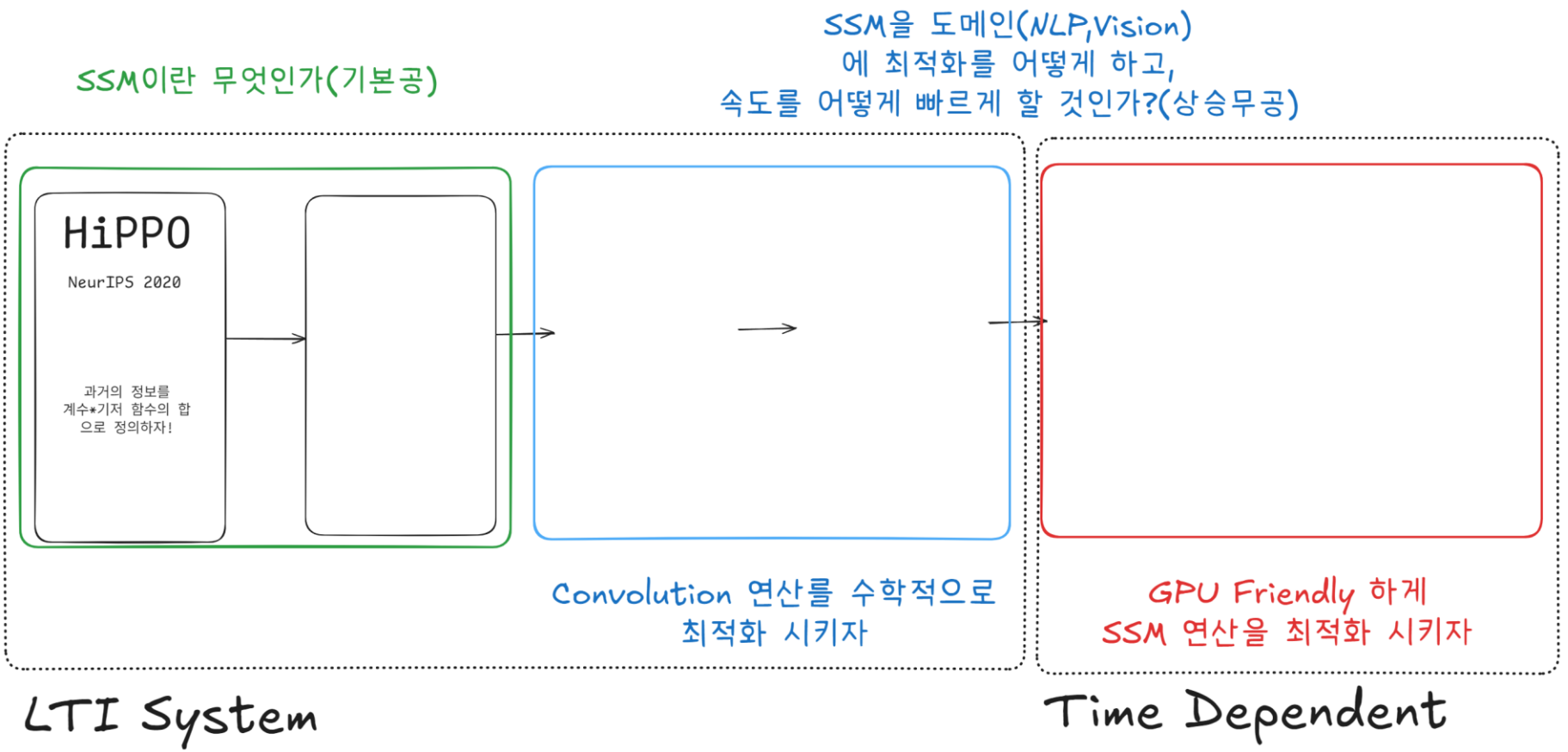
- SSM 모델의 창시자인 Albert Gu 와 Tri Dao 의 논문을 중심으로 연구 흐름을 보면 다음과 같음.



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Hippo

- f(t)의 과거의 정보를 **Orthogonal Polynomial Basis**의 합들로 정의하자!

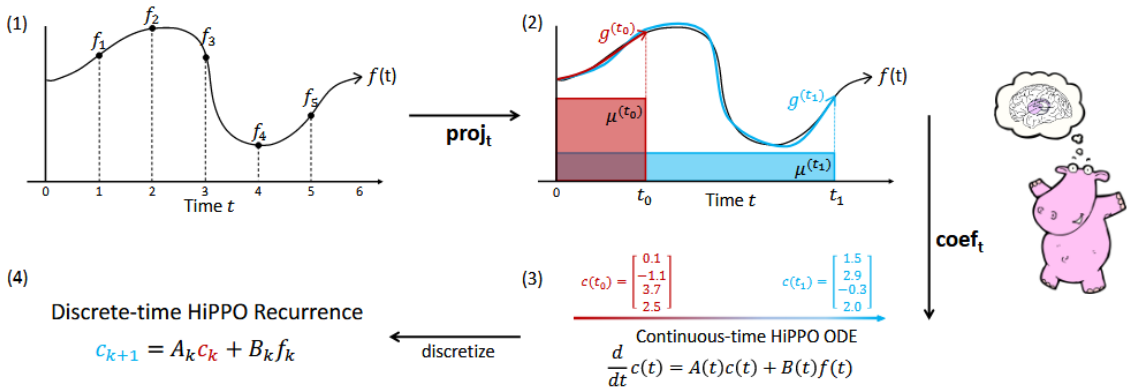


Hippo

- **Problem Setup**
  - Sequence 모델이 잘 만들어 졌다고 함은 Temproal Dependency를 잘 모델링 하고 학습을 하는 것. 즉, 과거의 데이터를 잘 요약하고 있음을 의미함. (**Online memory approximation** 을 잘하고 있다.)
  - **Online memory approximation:** $f_{\leq t} \coloneqq f(x)\,|_{x \leq t}$라고 할 때, $f_{\leq t}$에 대한 정보를 가장 잘 담을 수 있도록 하는 $g^{(t)}$ 를 찾는 것.
    - $(a\,rgmin\,|f_{\leq t} - g^{(t)}|_{\,L_2(\omega^{(t)})}, g^{(t)} \in G\ )$
  - **Universal approximation**: 모든 $f(x)$에 대해서 근사하는 함수 $g(x)$를 찾는 것.
    - $(argmin|f - g|_{L^P} < \mathcal{E}, \exists g\ \in \mathcal{F})$

Hippo

- Orthogonal Polynomial Basis의 합으로 f(t)를 표현하자!
  - **Orthogonal Polynomial Basis**로 $(f_{\leq t})$를 projection하여 근사화$(g^{(t)} := \sum c_i(t)\phi_i(t))$ 벡터화 후, 계수$(c(t))$의 변화를 통해서 $f_{\leq t}$ 변화를 추적하자. (ex) Fourier 시리즈,PCA로 함수를 표현)
  - $c(t)$가 $\frac{d}{dt}c(t) = A(t)c(t) + B(t)f(t)$ 형태로 변화한다고 정의한다면, 풀어야 하는 문제는
  - $argmin\,|f_{\leq t} - g^{(t)}|_{L_2(\omega^{(t)})}, g^{(t)} \in G$ 를 t에 따라서 푸는 문제로 바뀌게 됨. 이 문제를 풀기 위해서 조건은 다음과 같음
    - 1. G 는 Hilbert space이어야 함.
      - ✓ $f_{\leq t}$ 와 $g^{(t)}$에 대해서 **Norm**이 정의 되어야 하기 때문.
      - ✓ $\frac{d}{dt}c(t) = A(t)c(t) + B(t)f(t)$ 가 항상 수렴이 보장되어야 하기 때문(**이산화를 목표로 하기 때문**).
    - 2. Inner Product $< f, g > = \int f(x)g(x)\omega(x)dx$ 가 정의 되어야 하므로 measurement $\omega\,(x)$가 정의 되어야 함.
  - **Orthogonal Polynomial Basis**: 르장드르 다항식, 라게르 다항식, Chebyshev



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Hippo

- **Orthogonal Polynomial Basis**의 합으로 f(t)를 표현하자!
  - $f_{\leq t}$를 Closed-formed이 보장되는 OP로 Projection을 한 후($g^{(t)}$), $g^{(t)}$ 의 Coef 가 t에 따라서 변화하는 것을 $\frac{d}{dt}c(t) = A(t)c(t) + B(t)f(t)$ 으로 정리하고, 이 ODE 를 이산화 하면, $c_t = A_t c_{t-1} + B_t f_t$형태로 변환할 수 있음. (증명 수식은 생략...)

  - 본 논문에서는 르장드르 다항식을 기반으로 만들어진 Hippo-LegT 와 라게르 다항식으로 만든 Hippo-LagT를 소개함.

$$\textbf{LegT}: \mu^{(t)}(x) = \frac{1}{\theta}\mathbb{I}_{[t-\theta,t]}(x) \qquad \textbf{LagT}: \mu^{(t)}(x) = e^{-(t-x)}\mathbb{I}_{(-\infty,t]}(x) = \begin{cases} e^{x-t} & \text{if } x \leq t \\ 0 & \text{if } x > t \end{cases}$$

LegT:

$$A_{nk} = \frac{1}{\theta}\begin{cases} (-1)^{n-k}(2n+1) & if\ n \geq k \\ 2n+1 & if\ n \leq k \end{cases}, \quad B_n = \frac{1}{\theta}(2n+1)(-1)^n \quad (1)$$

LagT:

$$A_{nk} = \begin{cases} 1 & if\ n \geq k \\ 0 & if\ n < k \end{cases}, \quad B_n = 1 \quad (2)$$

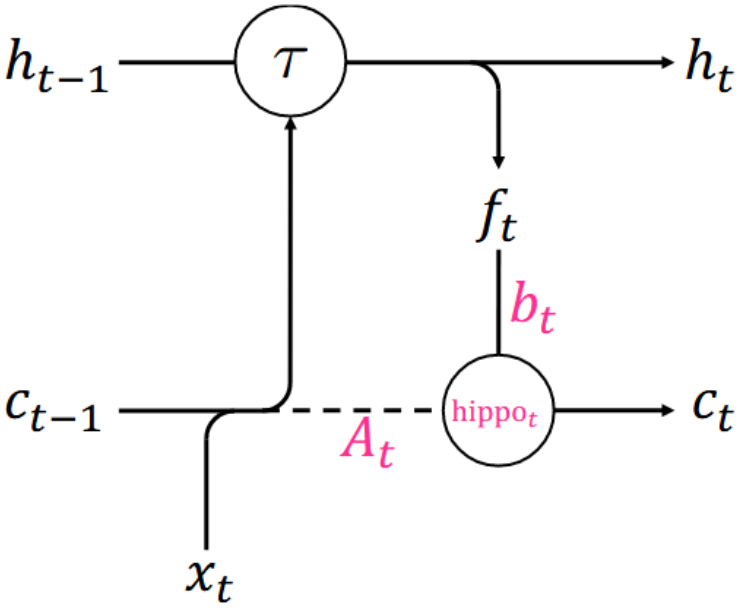
$$\boldsymbol{A} = \begin{bmatrix} 1 & & & & & & & \\ -1 & 2 & & & & & & \\ 1 & -3 & 3 & & & & & \\ -1 & 3 & -5 & 4 & & & & \\ 1 & -3 & 5 & -7 & 5 & & & \\ -1 & 3 & -5 & 7 & -9 & 6 & & \\ 1 & -3 & 5 & -7 & 9 & -11 & 7 & \\ -1 & 3 & -5 & 7 & -9 & 11 & -13 & 8 \\ \vdots & & & & & & & \ddots \end{bmatrix}$$

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Hippo

- RNN + Hippo
  - 기존의 RNN과 Hippo를 결합하여 사용.

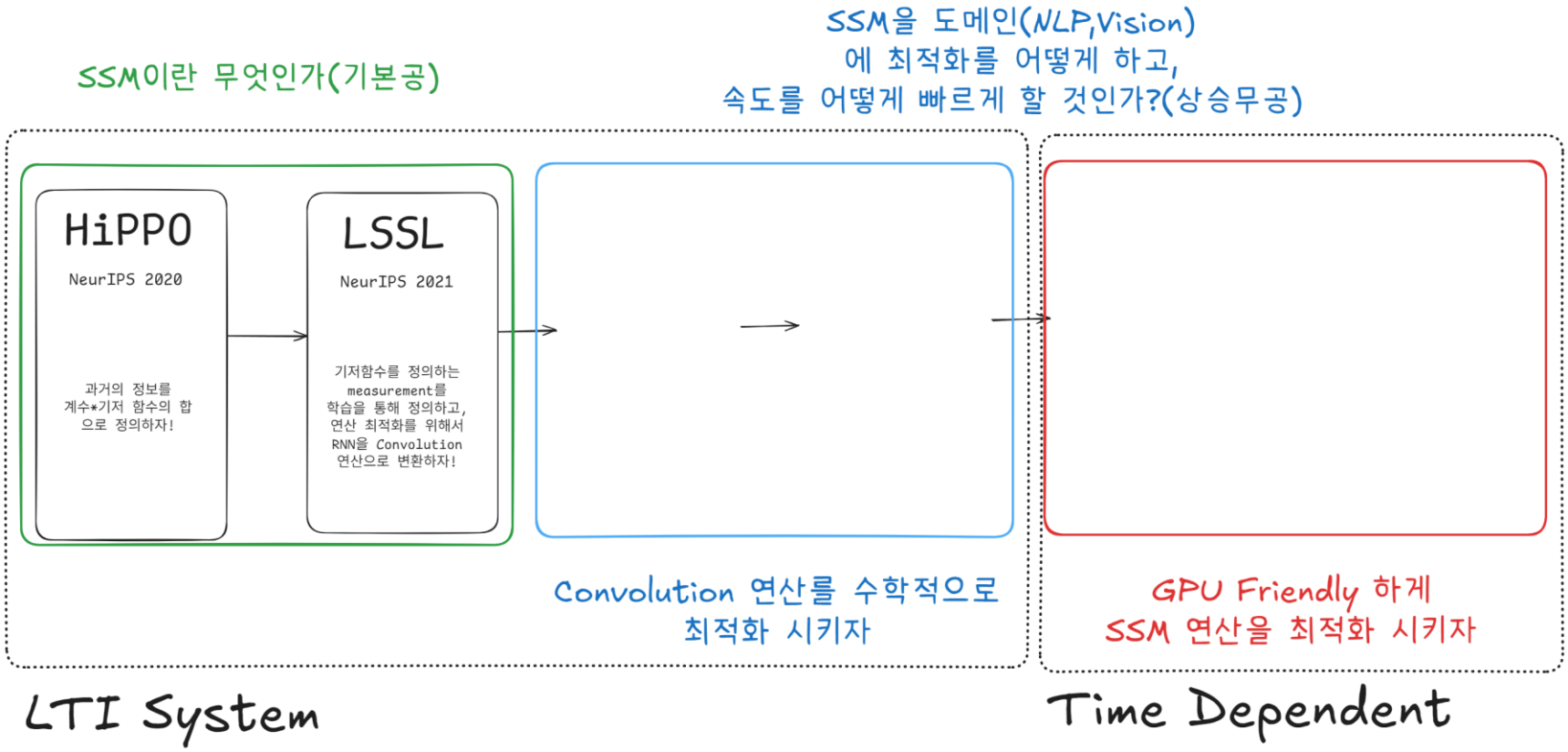
$$\tau(h, x) = (1 - g(h, x)) \circ h + g(h, x) \circ \tanh(\mathcal{L}_\tau(h, x)), \qquad g(h, x) = \sigma(\mathcal{L}_g(h, x)).$$



$$\begin{aligned} h_t \in \mathbb{R}^d &= \tau(h_{t-1}, [c_{t-1}, x_t]) \\ f_t \in \mathbb{R}^1 &= \mathcal{L}_f(h_t) \\ c_t \in \mathbb{R}^N &= \text{hippo}_t(f) \\ &= A_t c_{t-1} + B_t f_t \end{aligned} \tag{31}$$

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

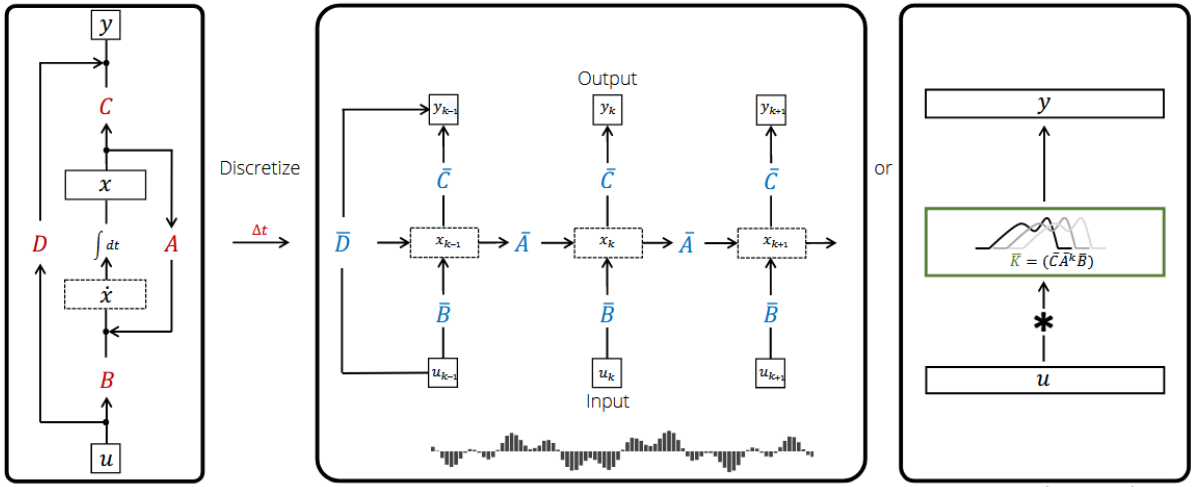
LSSL



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

LSSL

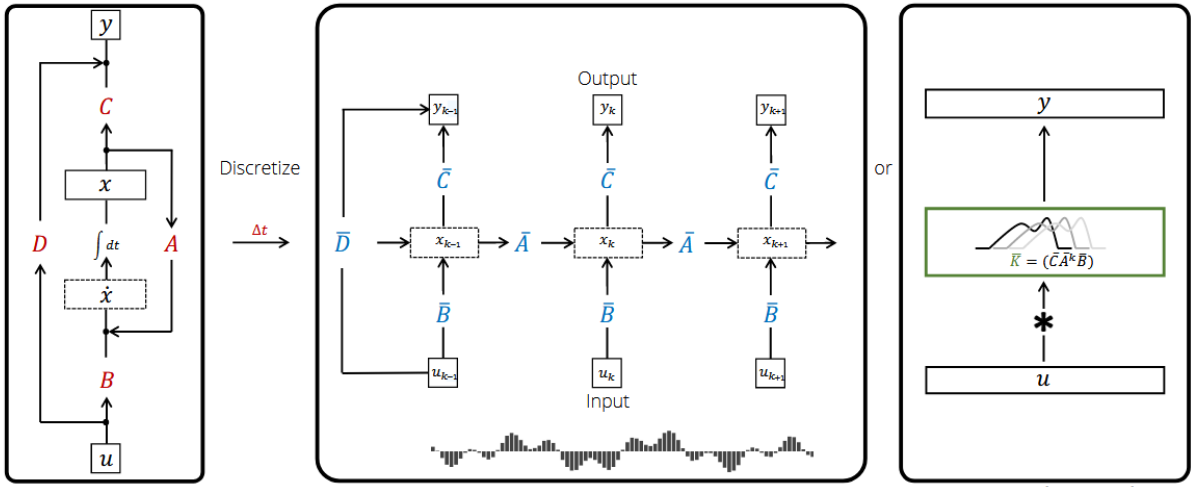
- **Hippo**: Orthogonal Polynomial Basis의 합(수식화 되어 고정)과 RNN 을 결합하여서 모델링.
- **LSSL**: Orthogonal Polynomial Basis(A matrix,B matrix)와 이산화시 사용하는 Δt 를 학습. 그리고 출력함수 $y_t = Cx_t + Du_t$ 를 통해 RNN과 결합하지 않고 사용. (이 때 이후로 SSM 모델이라고 부르기 시작함.)



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

LSSL

- 기존 RNN 계열(LSTM, GRU, RNN)이 Backward Euler Method로 미분 방정식을 이산화한 것과 달리, Picard Method를 이산화한 형태로 볼 수 있어 더 나은 성능을 보임.
- 순환 구조의 한계였던 병렬 처리 문제를 LSSL은 Recurrent 연산을 Convolution 연산으로 치환함으로써, 학습과 추론을 병렬적으로 수행할 수 있음.



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

LSSL

- Structured Learnable Matrices
  - 기존의 Hippo 에서는 특정한 measure ω 에 따라서 최적화된 **Orthogonal Polynomial basis** 를 정의하고, 이를 기반으로 A matrix를 생성.
  - LSSL에서는 임의의 ω에는 항상 A matrix(LRW(Sa et al(Gu가 포함) ACM 2018)인)가 존재하고, 메모리 시스템이 선형 SSM ($\dot{x} = Ax + Bu$)으로 표현 가능하다는 것을 증명. (Thoerem 1)
    - 이러한 증명을 통해서 A Matrix를 Trainable 하게 사용할 수 있고, LRW Matrix임을 이용해서 A^k와 같은 연산을 Linear한 속도로 할 수 있음을 알 수 있음.
    - LRW는 간단하게 $a_i(X) = \sum_{j=1}^{t} g_{i,j}(X) \cdot a_{i-j}(X), (i \geq t)$로 표현되는 재귀기반 다항식으로 구성된 matrix.
  - Hippo: $\sum c_i(t)\phi_i(t)$ 형태로 함수를 표현.
  - LSSL: $y_t = Cx_t + Du_t$ 형태로 함수를 표현.
  - 이 부분에 대해서는 Appendix A, B 를 통해서 OP basis를 고정하지 않은 상태에서도 A,B,C의 학습만을 통해서 표현력이 보장됨을 보여주고 있음.

Thoerem 1. For an arbitrary measure w, the optimal memorization operator hippo(ω) has the For $\dot{x}(t) = Ax(t) + Bu(t)$ for a low recurrence-width(LRW) state matrix A

LSSL

- State Space Model(SSM)의 Convolution 연산으로의 변환
  - LSSL 에서 수행하는 연산을 간단하게 보면 다음과 같음.
    - $x_t = \overline{A}x_{t-1} + \overline{B}u_t$
    - $y_t = Cx_t + Du_t$
  - 이 식을 풀어 쓰면, $y_k = C\left(\overline{A}^k\right)\overline{B}u_0 + C\left(\overline{A}^{k-1}\right)\overline{B}u_1 + \cdots C\overline{AB}u_{k-1} + \overline{B}u_k + \mathrm{D}u_k$ 가 되는데, 이 연산을 non-circular convolution 으로 볼 수 있음.
  - $y = \mathcal{K}_L\left(\overline{A}, \overline{B}, C\right) * u + Du$
  - $\mathcal{K}_L(A, B, C) = \left(CA^iB\right)_{i\in[L]} \in \mathbb{R}^L = (CB, CAB, \ldots, CA^{L-1}B)$
  - 또한, $\mathcal{K}_L(A, B, C)$ 는 Krylov function 이기 때문에 빠르게 연산을 할 수 있음. (커널 제작 연산)
  - 이 때, $\overline{A}$은 Quasi-separable Matrix (LRW)이므로 $\overline{A}^k$연산을 $O(N)$에 할 수 있고, Convolution 연산 같은 경우 FFT로 연산을 수행할 수 있기 때문에 $O(NlogN)$으로 학습이 가능함. (컨볼루션 연산)

For simplicity let the initial state be $x_{-1} = 0$. Then unrolling (3) explicitly yields

$$x_0 = \overline{B}u_0 \qquad x_1 = \overline{AB}u_0 + \overline{B}u_1 \qquad x_2 = \overline{A}^2\overline{B}u_0 + \overline{AB}u_1 + \overline{B}u_2 \qquad \ldots$$
$$y_0 = \overline{CB}u_0 \qquad y_1 = \overline{CAB}u_0 + \overline{CB}u_1 \qquad y_2 = \overline{CA}^2\overline{B}u_0 + \overline{CAB}u_1 + \overline{CB}u_2 \qquad \ldots$$

This can be vectorized into a convolution (4) with an explicit formula for the convolution kernel (5).

$$\begin{aligned} y_k &= \overline{C}\overline{A}^k\overline{B}u_0 + \overline{C}\overline{A}^{k-1}\overline{B}u_1 + \cdots + \overline{CAB}u_{k-1} + \overline{CB}u_k \\ y &= \overline{K} * u. \end{aligned} \tag{4}$$

$$\overline{K} \in \mathbb{R}^L := \mathcal{K}_L(\overline{A}, \overline{B}, \overline{C}) := \left(\overline{C}\overline{A}^i\overline{B}\right)_{i\in[L]} = (\overline{CB}, \overline{CAB}, \ldots, \overline{C}\overline{A}^{L-1}\overline{B}). \tag{5}$$

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

LSSL

- RNNs are LSSLs
  - $e^{z_k} = \Delta t_k$ 라 할 때, 동역학 모델을 Backward Euler method로 이산화 하게 되면 RNN 의 꼴과 동일하게 됨.
  - 즉, RNN 계열 모델들은 동역학 모델을 이산화 시킨 모델이라고 볼 수 있음.

Backward Euler method. The backward Euler method approximates (11) by holding the right endpoint constant throughout the integral (i.e., the "rectangle rule" with right endpoint), $f(s, x(s)) \approx f(t_k, x(t_k))$. The discrete-time update becomes

$$\begin{aligned} x_k - x_{k-1} &= (t_k - t_{k-1}) f(t_k, x(t_k)) \\ &= \Delta t_k f(t_k, x_k). \end{aligned} \tag{13}$$

$$\dot{x}(t) = -x + f(t, x(t)) \tag{14}$$

Lemma C.1. *Consider an RNN of the form*

$$x_k = (1 - \sigma(z_k)) x_{k-1} + \sigma(z_k) \overline{f}(k, x_{k-1}), \tag{15}$$

where $\overline{f}(k, x)$ is an arbitrary function that is Lipschitz in its second argument (e.g., it may depend on an external input u_k).

Then equation (15) is a discretization of the dynamics (14) with step sizes $\Delta t_k = \exp(z_k)$, i.e. $x_k \approx x(t_k)$ where $t_k = \sum_{i=1}^{k} \Delta t_i$.

Proof. Apply the backwards Euler discretization (13) to equation (14) to get

$$\begin{aligned} x_k - x_{k-1} &= \Delta t_k \left[-x_k + f(t_k, x_k) \right] \\ (1 + \Delta t_k) x_k &= x_{k-1} + \Delta t_k f(t_k, x_k) \\ x_k &= \frac{1}{1 + \Delta t_k} x_{k-1} + \frac{\Delta t_k}{1 + \Delta t_k} f(t_k, x_k). \end{aligned}$$

Note that $\frac{\Delta t_k}{1+\Delta t_k} = \frac{e^{z_k}}{1+e^{z_k}} = \frac{1}{1+e^{-z_k}}$ and $\frac{1}{1+\Delta t_k} = 1 - \frac{\Delta t_k}{1+\Delta t_k}$, thus

$$x_k = (1 - \sigma(z_k)) x_{k-1} + \sigma(z_k) \overline{f}(k, x_{k-1}).$$

Here we are denoting $\overline{f}(k, x) = f(t_k, x)$ to be a discrete-time version of f evaluatable at the given timesteps t_k. □

Note that a potential external input function $u(t)$ or sequence u_k is captured through the abstraction $f(t, x)$. For example, a basic RNN could define $\overline{f}(k, x) = f(t_k, x) = \tanh(Wx + Uu_k)$.

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

LSSL

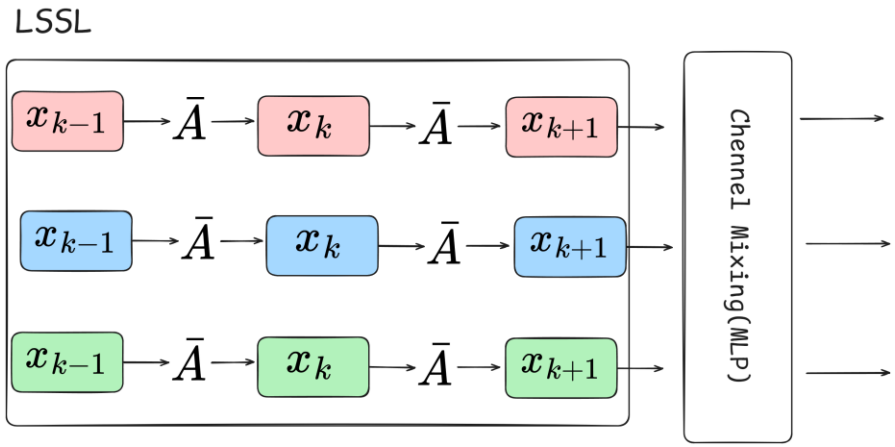
- RNNs are LSSLs
  - RNN :동역학 모델을 BackWard Euler로 1차 이산화.
  - LSSL : 동역학 모델을 Picard iteration 을 통해 여러 번 근사화.(Using Stacking) (ex)테일러 급수)

Table 6: A summary of the characteristics of popular RNN methods and their approximation mechanisms for capturing the dynamics $\dot{x}(t) = -x(t) + f(t, x(t))$ (equation (14)). The LSSL entries are for the very specific case with order $N = 1$ and $A = -1, B = 1, C = 1, D = 0$; LSSLs are more general.

Method	RNN	RNN	LSSL	LSSL
Variant	Gated	Gated, linear	Discrete (4)+(5)	Continuous (1)+(2)
Special cases	LSTM [28], GRU [14]	QRNN [5], SRU [33]		
Deep?	Single-layer	Deep	Deep	Deep
Continuous?	Discrete-time	Discrete-time	Discrete-time	Continuous-time
Linear?	Non-linear	Linear	Linear	Linear
Approximation				
Depth-wise	-	Picard iteration	Picard iteration	Picard iteration
Time-wise	Backwards Euler	GBT($\alpha = 1$)	GBT($\alpha = \frac{1}{2}$) (i.e. Bilinear)	-

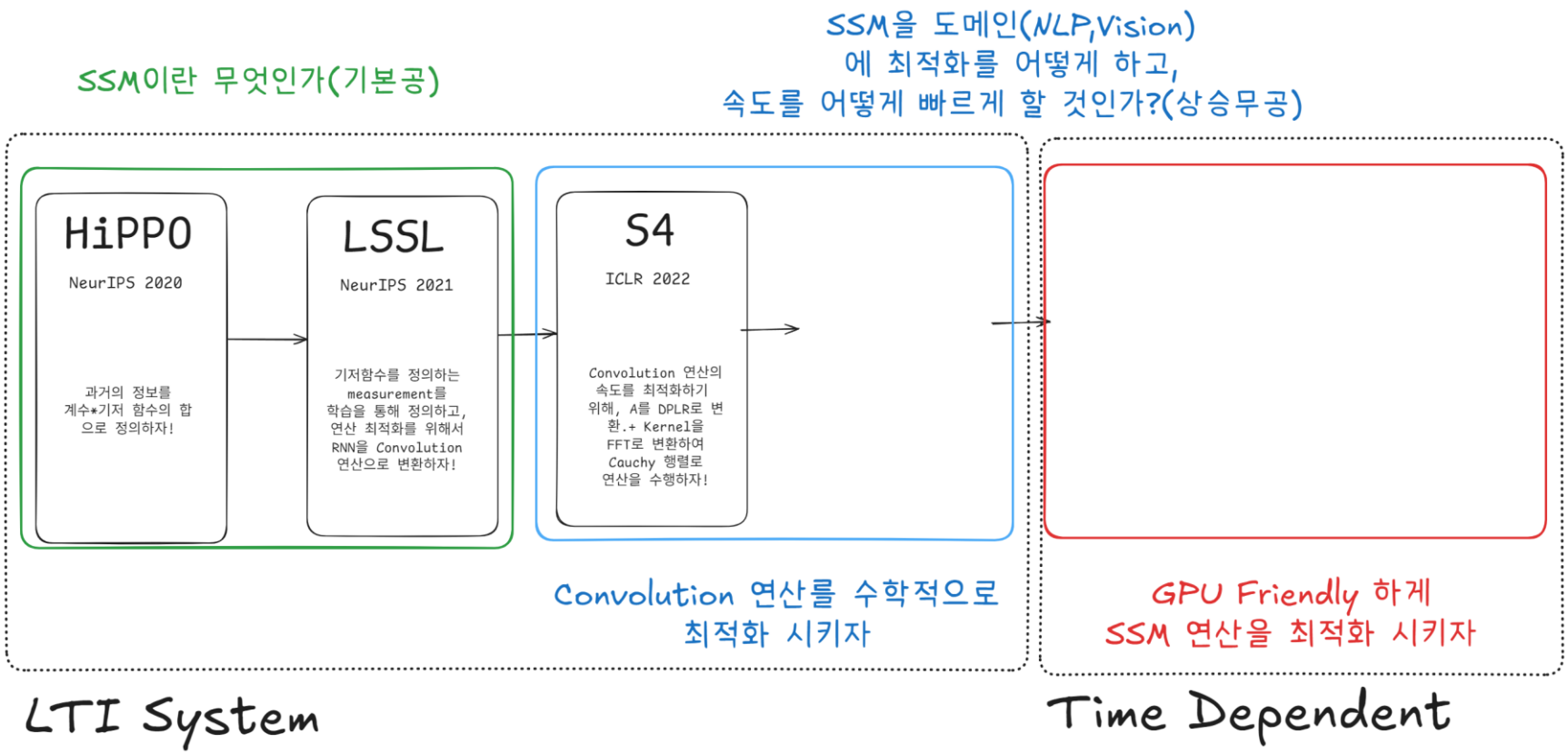
LSSL

- Deep LSSLs Architecture
  - Basic LSSL:1D sequene에 대해서 정의가 되어 있음.
  - Deep LSSL: linear를 통해서 hidden dimension, 새로운 차원을 정의하고, 각 차원 H에 대해서 독립적인 LSSL을 통해서 연산을 수행.+ FeedForward Layer 추가하여, H feature 에 대한 분석을 수행. Transformer 구조와 비슷하게 stacking을 수행하기 위해 , Residual, Layer Normalization 추가.



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

S4



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

S4

- LSSL는 이론적으로 좋은 성능을 보였으나, 컨볼루션 연산시 $O(N^2L)$의 시간 복잡도와, 학습 때, HiPPO 행렬의 대각화가 불가능하거나 수치적으로 불안정한 문제가 존재 했었음.
- S4에서는 이러한 수치적 불안전성을 A Matrix를 Diagonal + Low-Rank (DPLR) 구조로 분해하고, Kernel($\bar{K}$)를 생성하는 알고리즘을 FFT를 기반으로 Cauchy Kernel 알고리즘을 만들어 안정적으로 계산할 수 있도록 했음.

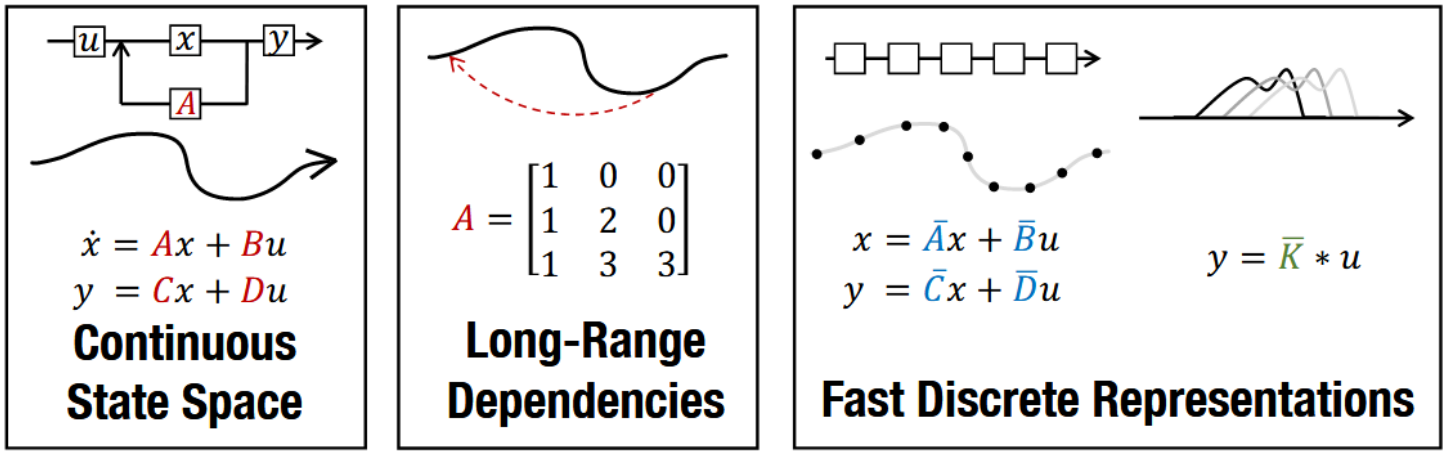


Figure 1: (**Left**) State Space Models (SSM) parameterized by matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ map an input signal $u(t)$ to output $y(t)$ through a latent state $x(t)$. (**Center**) Recent theory on continuous-time memorization derives special $\boldsymbol{A}$ matrices that allow SSMs to capture LRDs mathematically and empirically. (**Right**) SSMs can be computed either as a recurrence (left) or convolution (right). However, materializing these conceptual views requires utilizing different representations of its parameters (red, blue, green) which are very expensive to compute. S4 introduces a novel parameterization that efficiently swaps between these representations, allowing it to handle a wide range of tasks, be efficient at both training and inference, and excel at long sequences.

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

S4

- 어떤 행렬 V로 Conjugation을 수행해도 시스템의 동작이 동일함.(Lemma 3.1)
- A를 NPLR 구조로 바꿔도 모델이 표현하는 시퀀스가 동일하다는 것 뜻.

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:

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

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. □

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

S4

- "이론적으로 SSM에서 다루는 모든 Matrix A들은 LRW Matrix"(LSSL)
- 이를 통해서 $A = V\Lambda V^* - PQ^*$ 형태로 바꿀 수 있음을 알 수 있음. (이렇게 변환이 되면 역행렬 구하기가 너무 쉬움(Woodbury Identity & Hadamard Product)
- 변환된 A Matrix를 통해서 커널 $\mathrm{K}_L(\overline{A}, \overline{B}, C)$를 FFT 로 빠르게 변환할 수 있음.
  - $\mathrm{K}_L(\overline{A}, \overline{B}, C) \rightarrow \hat{K}(\omega_j) = \sum K_n \omega_j^n$ 형태로 나타내지고, FFT(Cauchy 연산으로 바뀌어서 매우 빨라짐)로 기존 Convolution 연산을 곱셈으로 빠르게 변환 .
- S4 를 통해서 LSSL가 수학적으로는 동일 하지만, 훨씬 안정적으로 학습이 가능하고 속도가 매우 빠른 SSM 연산이 가능한 모델이 만듦.

Algorithm 1 S4 Convolution Kernel (Sketch)

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

1: $\widetilde{\boldsymbol{C}} \leftarrow \left(\boldsymbol{I} - \overline{\boldsymbol{A}}^L\right)^* \overline{\boldsymbol{C}}$ ▷ Truncate SSM generating function (SSMGF) to length L
2: $\begin{bmatrix} k_{00}(\omega) & k_{01}(\omega) \\ k_{10}(\omega) & k_{11}(\omega) \end{bmatrix} \leftarrow \left[\widetilde{\boldsymbol{C}}\ \boldsymbol{Q}\right]^* \left(\frac{2}{\Delta}\frac{1-\omega}{1+\omega} - \boldsymbol{\Lambda}\right)^{-1} [\boldsymbol{B}\ \boldsymbol{P}]$ ▷ Black-box Cauchy kernel
3: $\hat{\boldsymbol{K}}(\omega) \leftarrow \frac{2}{1+\omega}\left[k_{00}(\omega) - k_{01}(\omega)(1 + k_{11}(\omega))^{-1}k_{10}(\omega)\right]$ ▷ Woodbury Identity
4: $\hat{\boldsymbol{K}} = \{\hat{\boldsymbol{K}}(\omega) : \omega = \exp(2\pi i \frac{k}{L})\}$ ▷ Evaluate SSMGF at all roots of unity $\omega \in \Omega_L$
5: $\overline{\boldsymbol{K}} \leftarrow \mathsf{iFFT}(\hat{\boldsymbol{K}})$ ▷ Inverse Fourier Transform

	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	$\boldsymbol{H^2}$	$L^2H + H^2L$	$\boldsymbol{H^2}$

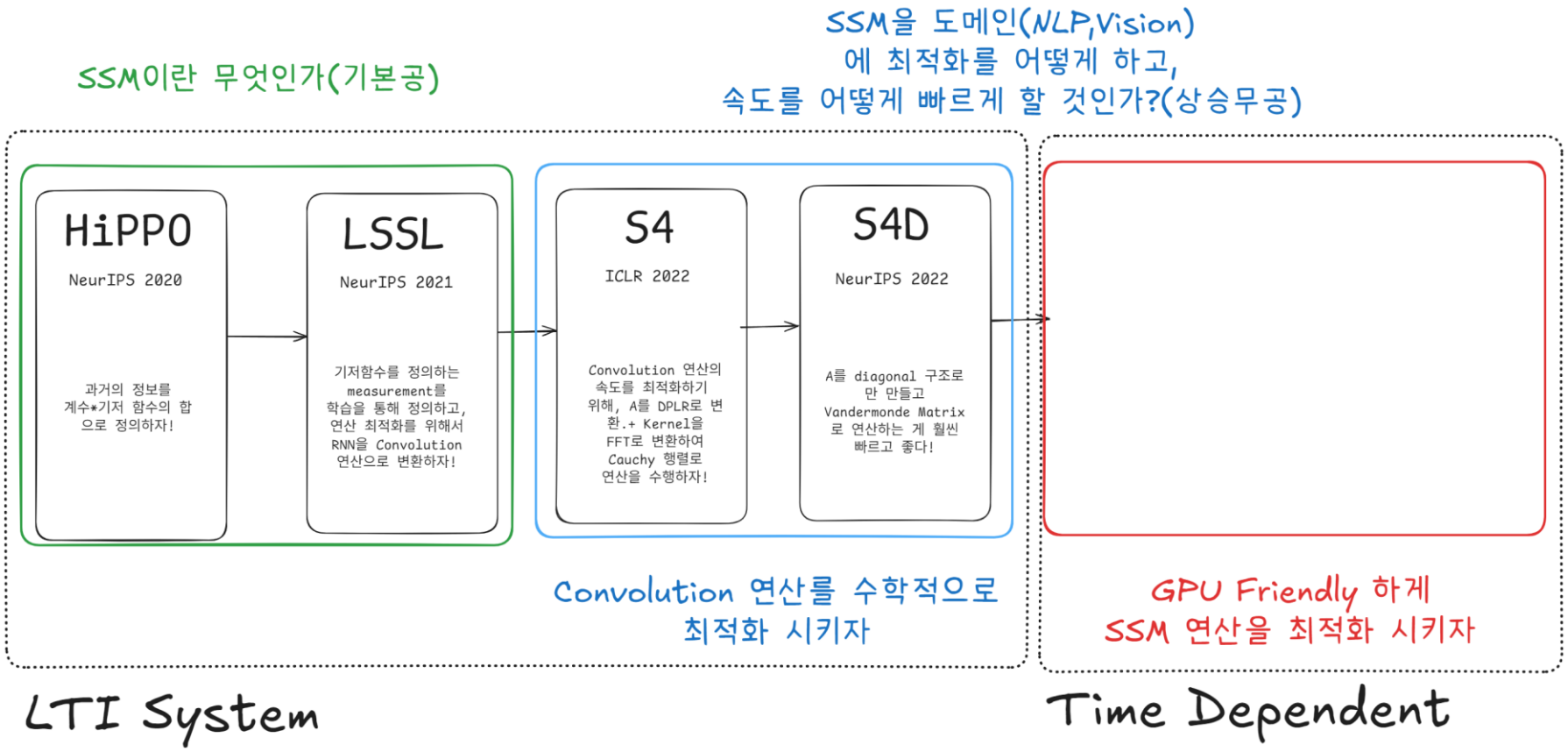
Prof. Hyerim Bae (hrbae@pusan.ac.kr)

S4

- Deep Learning S4
  - A= $\Lambda - PQ^*$, $\Lambda \in \mathbb{C}^{N\times N}$(diagonal),
  - $P, Q \in \mathbb{C}^{N\times 1}$
  - $B \in \mathbb{C}^N$, $C \in \mathbb{C}^N$
- Conjugate basis V 같은 경우에는 이미 V 위에서 모든 연산이 수행된다는 가정을 기반으로 해서 따로 연산을 수행하지는 않음.
- 나머지는 전부 기존 LSSL 과 동일하게 각 Channel 별로 이산식 하나씩 세워서 수행하고, MLP 를 통해서 Channel Mixing 을 수행함.

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

S4D



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

S4D

- S4는 Diagonal + Low-Rank (DPLR) 으로 A matrix를 변환해서 빠른 속도로 연산하는 것을 보장했다면, S4D는 순수한 Diagonal Matrix로 A Matrix를 변환한 상태에서 성능을 보장하는 것을 목표로 하는 논문.
- SSM 을 연속시간에서 정의를 한다면, (1) 와 (2) 형태로 표현할 수 있고, (3)의 형태로 기저함수의 선형조합으로 재해석할 수 있음.

$$\begin{aligned} x'(t) &= \boldsymbol{A}x(t) + \boldsymbol{B}u(t) \\ y(t) &= \boldsymbol{C}x(t) \end{aligned} \tag{1}$$

$$\begin{aligned} K(t) &= \boldsymbol{C}e^{t\boldsymbol{A}}\boldsymbol{B} \\ y(t) &= (K * u)(t) \end{aligned} \tag{2}$$

$$K(t) = \sum_{n=0}^{N-1} \boldsymbol{C}_n K_n(t) \qquad K_n(t) := \boldsymbol{e}_n^\top e^{t\boldsymbol{A}}\boldsymbol{B} \tag{3}$$

$$\textbf{(Bilinear)}\ \begin{aligned} \overline{\boldsymbol{A}} &= (\boldsymbol{I} - \Delta/2\boldsymbol{A})^{-1}(\boldsymbol{I} + \Delta/2\boldsymbol{A}) \\ \overline{\boldsymbol{B}} &= (\boldsymbol{I} - \Delta/2\boldsymbol{A})^{-1} \cdot \Delta\boldsymbol{B} \end{aligned} \qquad \textbf{(ZOH)}\ \begin{aligned} \overline{\boldsymbol{A}} &= \exp(\Delta\boldsymbol{A}) \\ \overline{\boldsymbol{B}} &= (\Delta\boldsymbol{A})^{-1}(\exp(\Delta \cdot \boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta\boldsymbol{B}. \end{aligned}$$

$$y = u * \overline{\boldsymbol{K}} \qquad \text{where } \overline{\boldsymbol{K}} = (\boldsymbol{C}\overline{\boldsymbol{B}}, \boldsymbol{C}\overline{\boldsymbol{A}\boldsymbol{B}}, \ldots, \boldsymbol{C}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}}). \tag{6}$$

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

S4D

- Convolution 을 수행하는 Kernel은 S4에서는 FFT(Cauchy)를 썼지만, S4D에서는 Vandermonde 를 통해서 연산을 수행. (Vandermonde 가 DPLR 에서는 e^A연산 비용이 높아서 사용을 안했지만, A matrix가 Diagonal 하기 때문에 상관이 없음.)

$$\boldsymbol{A}_{nk} = -\begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & n > k \\ n+1 & n = k \\ 0 & n < k \end{cases} \tag{4}$$

$$\boldsymbol{B}_n = (2n+1)^{\frac{1}{2}} \quad \boldsymbol{P}_n = (n+1/2)^{\frac{1}{2}}$$

(HiPPO-LegS matrix used in S4)

$$\boldsymbol{A}^{(N)}_{nk} = -\begin{cases} (n+\frac{1}{2})^{1/2}(k+\frac{1}{2})^{1/2} & n > k \\ \frac{1}{2} & n = k \\ (n+\frac{1}{2})^{1/2}(k+\frac{1}{2})^{1/2} & n < k \end{cases} \tag{5}$$

$$\boldsymbol{A} = \boldsymbol{A}^{(N)} - \boldsymbol{P}\boldsymbol{P}^\top, \qquad \boldsymbol{A}^{(D)} := \mathrm{eig}(\boldsymbol{A}^{(N)})$$

(Normal / diagonal plus low-rank form)

$$\overline{\boldsymbol{K}}_\ell = \sum_{n=0}^{N-1} \boldsymbol{C}_n \overline{\boldsymbol{A}}_n^\ell \overline{\boldsymbol{B}}_n \implies \overline{\boldsymbol{K}} = (\overline{\boldsymbol{B}}^\top \circ \boldsymbol{C}) \cdot \mathcal{V}_L(\overline{\boldsymbol{A}}) \qquad \text{where } \mathcal{V}_L(\overline{\boldsymbol{A}})_{n,\ell} = \overline{\boldsymbol{A}}_n^\ell \tag{7}$$

$$\overline{\boldsymbol{K}} = \begin{bmatrix} \overline{\boldsymbol{B}}_0\boldsymbol{C}_0 & \dots & \overline{\boldsymbol{B}}_{N-1}\boldsymbol{C}_{N-1} \end{bmatrix} \begin{bmatrix} 1 & \overline{\boldsymbol{A}}_0 & \overline{\boldsymbol{A}}_0^2 & \dots & \overline{\boldsymbol{A}}_0^{L-1} \\ 1 & \overline{\boldsymbol{A}}_1 & \overline{\boldsymbol{A}}_1^2 & \dots & \overline{\boldsymbol{A}}_1^{L-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \overline{\boldsymbol{A}}_{N-1} & \overline{\boldsymbol{A}}_{N-1}^2 & \dots & \overline{\boldsymbol{A}}_{N-1}^{L-1} \end{bmatrix}$$

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

S4D

- Diagonal A Matrix 와 Vandermonde 커널만으로 연산이 잘되는것을 보여줌.

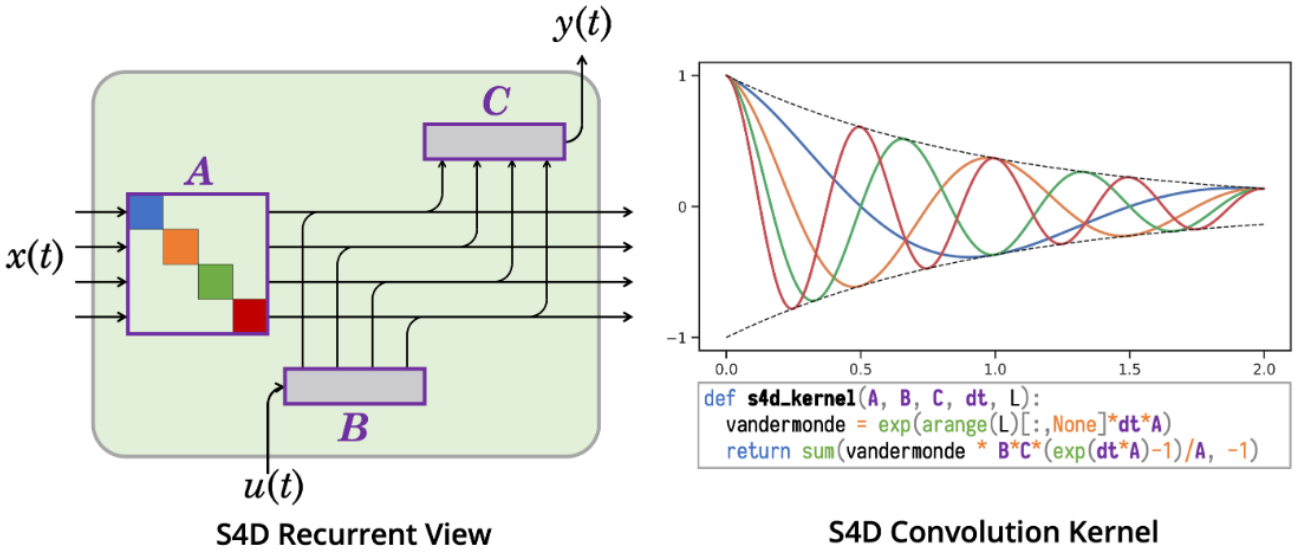


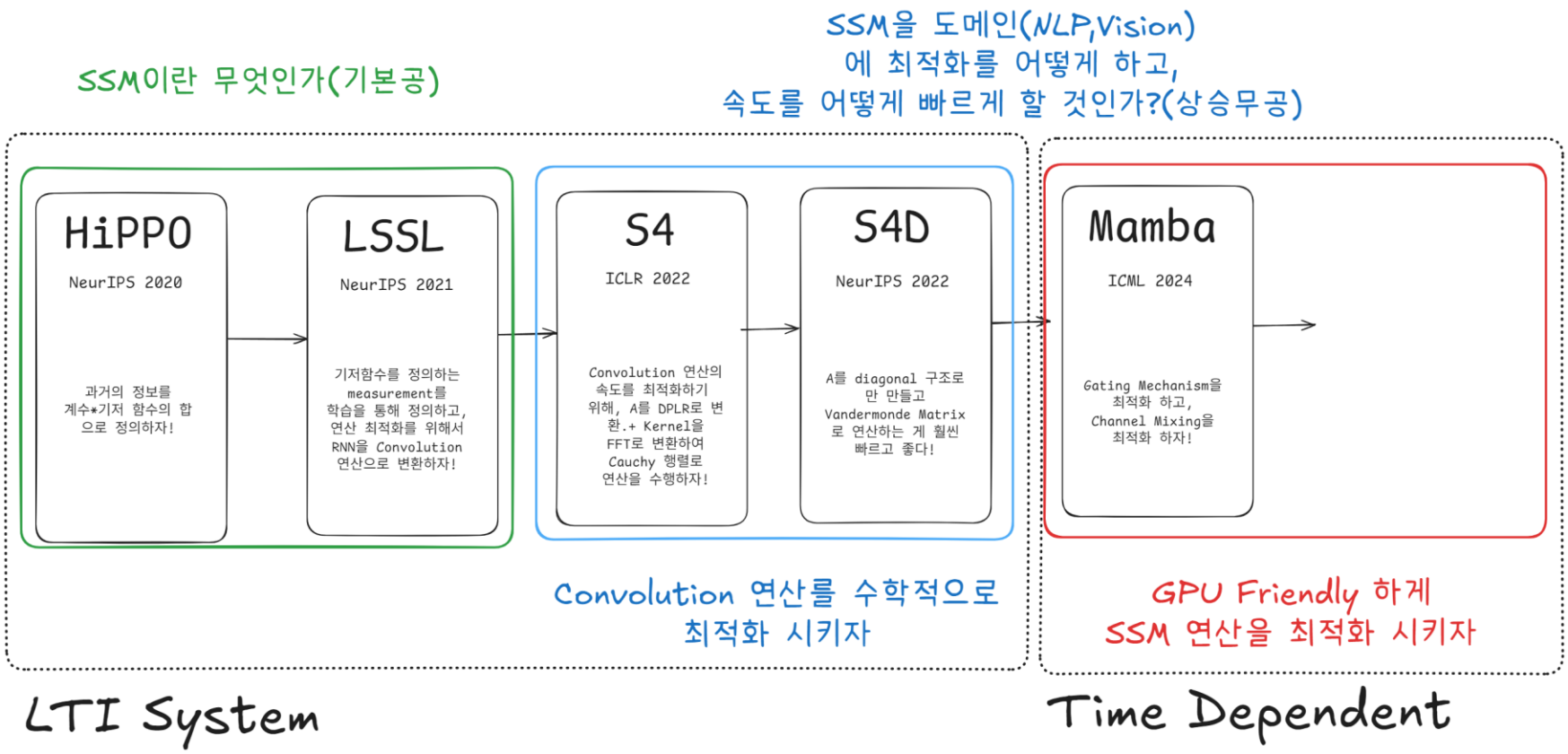
Figure 1: S4D is a diagonal SSM which inherits the strengths of S4 while being much simpler. (*Left*) The diagonal structure allows it to be viewed as a collection of 1-dimensional SSMs. (*Right*) As a convolutional model, S4D has a simple interpretable convolution kernel which can be implemented in two lines of code. Colors denote independent 1-D SSMs; purple denotes trainable parameters.

Table 1: (**Parameterization choices for Structured SSMs.**) Aside from the core structure of $\boldsymbol{A}$ and the computation of its convolution kernel, SSMs have several design choices which are consolidated in S4D.

Method	Structure	Kernel Computation	Discretization	Constraint $\Re(\boldsymbol{A})$	Trainable $\boldsymbol{B}$	Initialization of $\boldsymbol{A}$
S4	DPLR	Cauchy	Bilinear	exp	Yes	HiPPO
DSS	diagonal	softmax	ZOH	id (none)	No	HiPPO-D
S4D	diagonal	Vandermonde	either	exp / ReLU	optional	various

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba

- S4의 경우, LTI 시스템으로서 병렬성과 연산최적화를 해냈지만, 현재 상황을 분석하는 것에 약점을 보이는 문제점이 있음. 이 때문에 **Long-Range Dependency** 처리가 약했음.
- 그래서 Mamba 논문에서는Non LTI System인 **Selective SSM**을 제안하여, LTI 시스템이었던 기존의 S4의 한계를 넘고, LTI 시스템이기에 얻었던 장점인 병렬성을 GPU 최적화를 통해서 성능을 향상시킴.

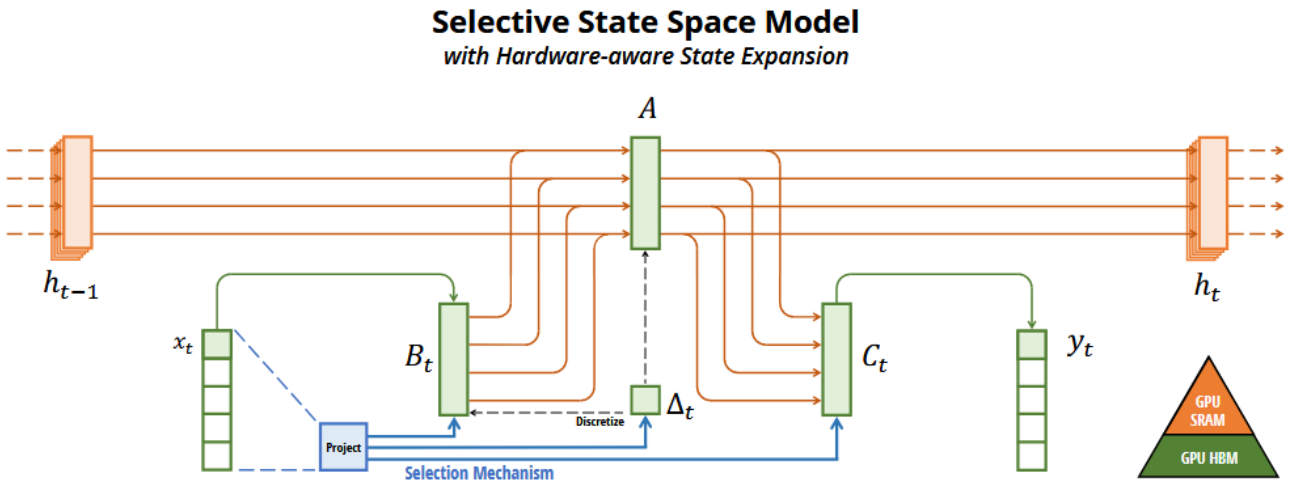


Figure 1: **(Overview.)** Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate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

Mamba

- Selective State Space Model

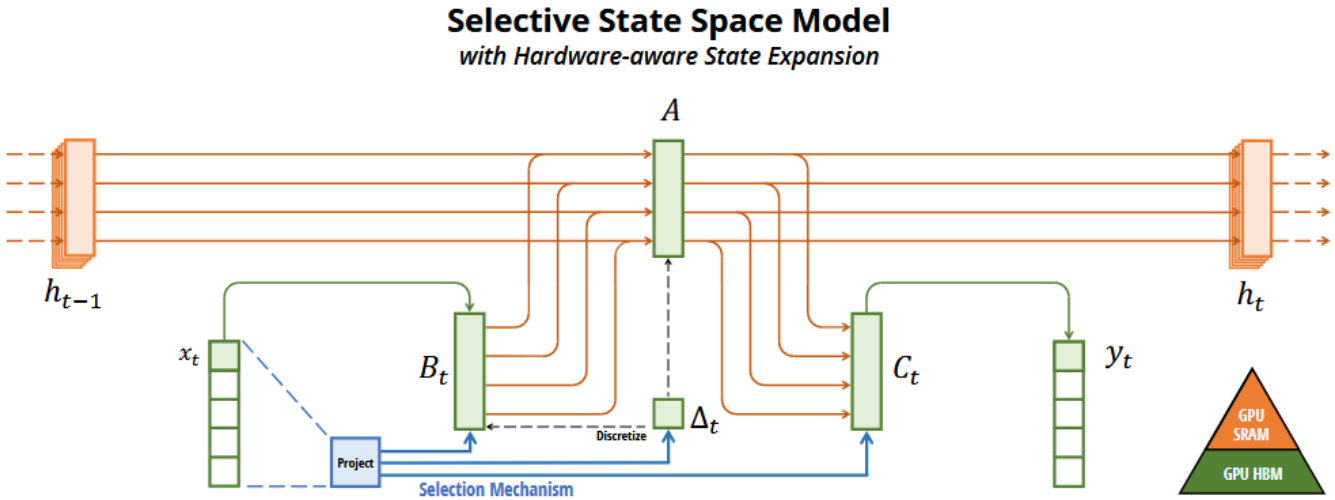


Figure 1: **(Overview.)** Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate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

$$h'(t) = Ah(t) + Bx(t) \quad (1a) \qquad h_t = \overline{A}h_{t-1} + \overline{B}x_t \quad (2a) \qquad \overline{K} = (C\overline{B}, C\overline{AB}, \ldots, C\overline{A}^k\overline{B}, \ldots) \quad (3a)$$

$$y(t) = Ch(t) \quad (1b) \qquad y_t = Ch_t \quad (2b) \qquad y = x * \overline{K} \quad (3b)$$

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

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba

- Selective State Space Model

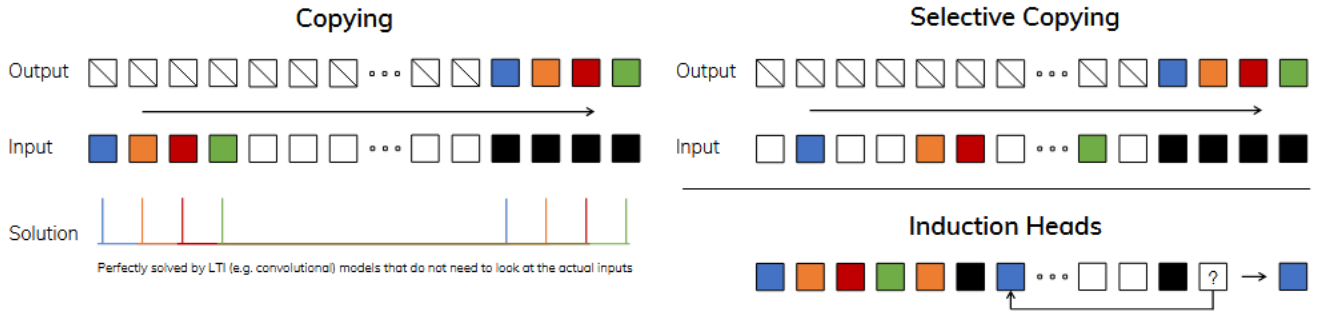


Figure 2: (*Left*) The standard version of the Copying task involves constant spacing between input and output elements and is easily solved by time-invariant models such as linear recurrences and global convolutions. (*Right Top*) The Selective Copying task has random spacing in between inputs and requires time-varying models that can *selectively* remember or ignore inputs depending on their content. (*Right Bottom*) The Induction Heads task is an example of associative recall that requires retrieving an answer based on context, a key ability for LLMs.

Algorithm 1 SSM (S4)

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

Algorithm 2 SSM + Selection (S6)

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

Mamba

- 또한 기존의 Transformer 와 유사한 아키텍처에서 MLP 아키텍처를 모방한 형태로 변형했음. (H3, Hyena, RWKV 와 같은 논문에서 차용)

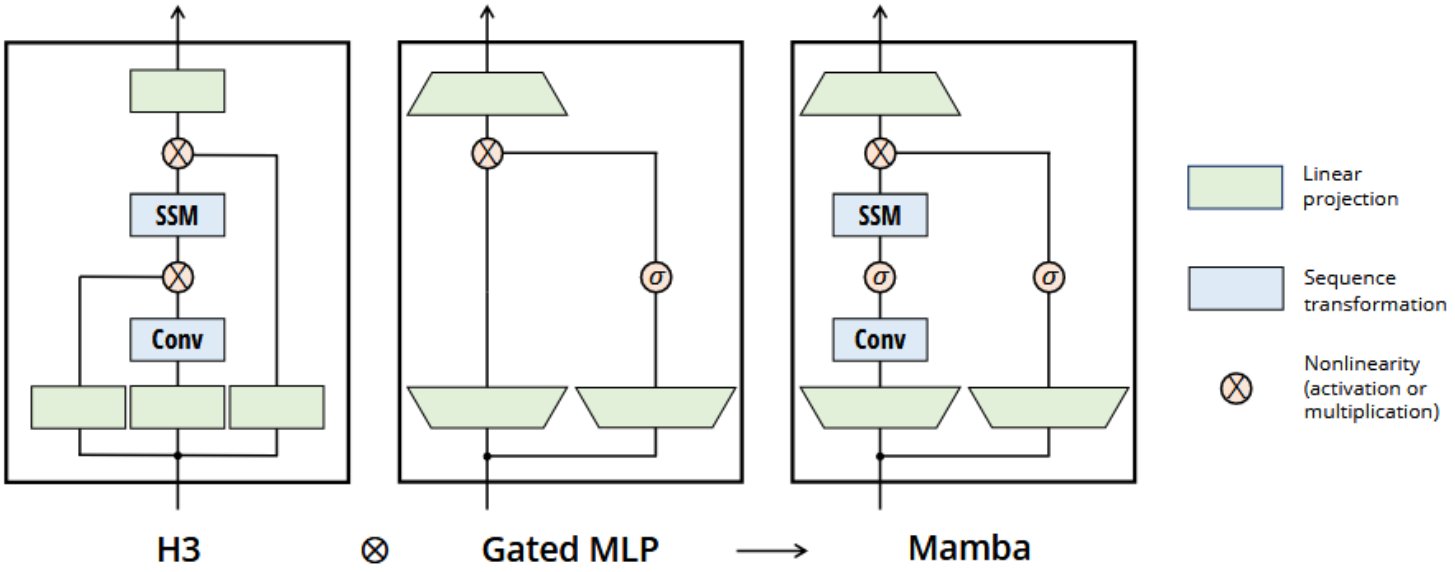
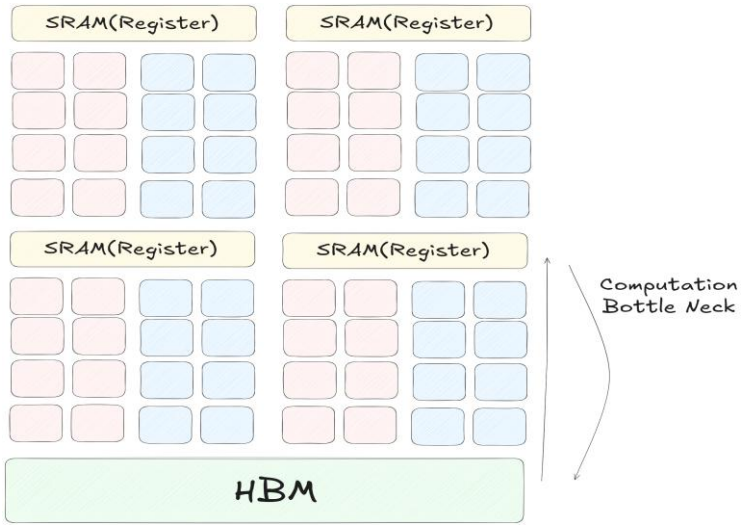
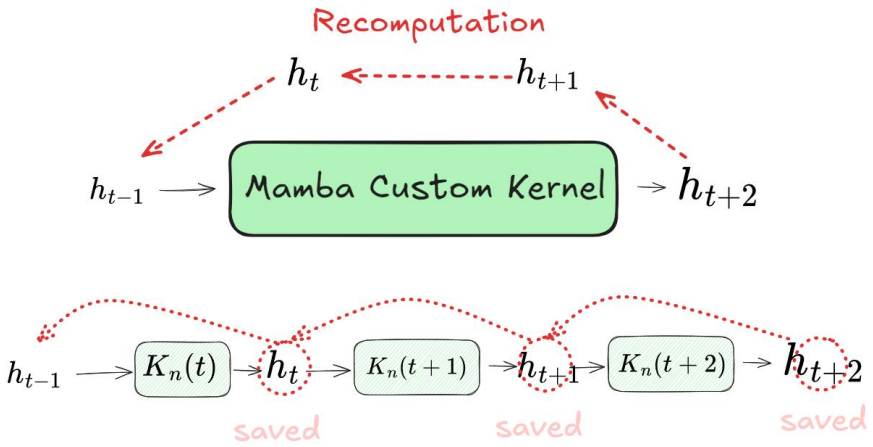


Figure 3: (**Architecture.**) Our simplified block design combines the H3 block, which is the basis of most SSM architectures, with the ubiquitous MLP block of modern neural networks. Instead of interleaving these two blocks, we simply repeat the Mamba block homogenously. Compared to the H3 block, Mamba replaces the first multiplicative gate with an activation function. Compared to the MLP block, Mamba adds an SSM to the main branch. For σ we use the SiLU / Swish activation (Hendrycks and Gimpel 2016; Ramachandran, Zoph, and Quoc V Le 2017).

Mamba

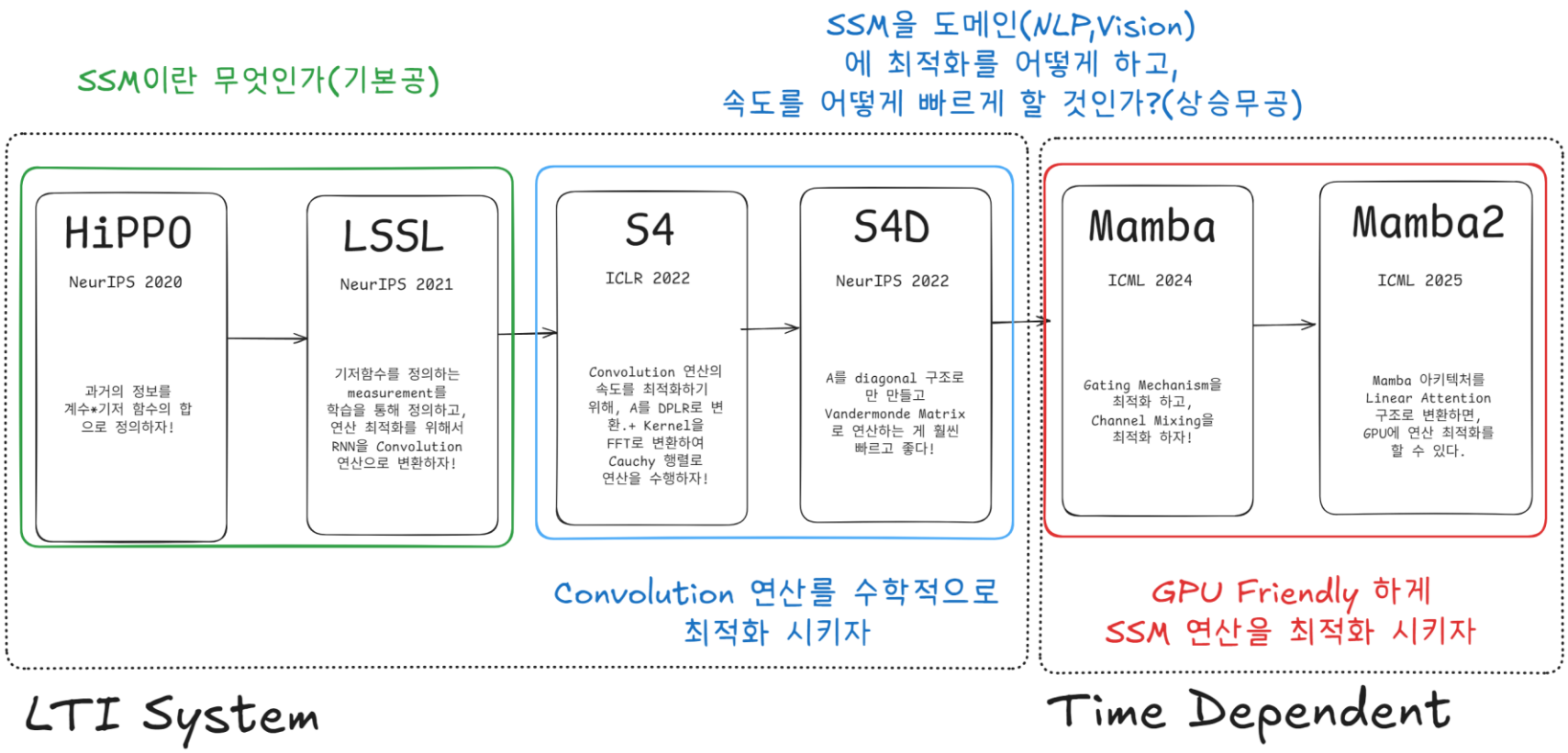
- Selective SSM의 경우 A, B, C, Δ 가 time variant 하기 때문에 먼저 계산하기 기법(Conv)가 불가능하고, 계속해서 저장되는 hidden state로 인해서 메모리가 매우매우 많이 필요함. 이러한 문제점을 해결하기 위해서 GPU friendly 하게 recurrence 연산을 최적화하여 속도를 가속화하고 Memory 효과적으로 만듦.
  - **Kernel Fusion**
    - 기존의 Pytorch로 연산을 구현하게 되면, 여러 커널(Tensor, nn.Module)을 통해서 수행하므로, 연산에 사용되는 A, B, C, Δ 가 계속해서 GPU의 HBM과 SRAM을 왔다 갔다 하면서 병목이 생김. 이러한 문제점을 해결하기 위해서 Selective SSM을 하나의 kernel 연산으로 묶어서 GPU 내부 SRAM에서만 처리 (이를 통해 속도가 40배 이상 향상됨) +hidden state를 저장하지 않고 연산함으로서 메모리를 절약함.
  - **Recomputation**
    - Recurrent 모델이 작동될 때, Pytorch의 Autograd는 연산의 중간 결과를 계속 h_t형태로 저장을 하고 있음. 이러한 저장은 메모리를 과도하게 잡기 때문에 Mamba에서는 forward에서 저장하지 않고 backward 때 다시 계산을 해서 사용하게 됨. (like FlashAttention)





Prof. Hyerim Bae (hrbae@pusan.ac.kr)

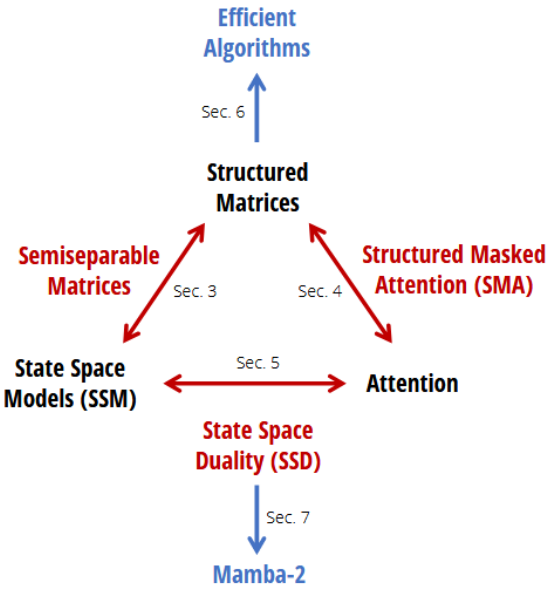
Mamba 2



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba 2

- 기존의 Mamba 와 같은 경우, Recurrence Neural Network 형태에서 시작해서 연산을 최적화 하는 형태로 진행되었음.
- Mamba 2에서는 Linear Attention, Mamba, Attention 을 모두 하나의 Generalized 된 Masked Attention(**Structured Matrics**)의 형태로 보면서, Mamba의 연산을 Attention 연산과 유사한 Matrix 연산으로 최적화(Stated Space Duality(SSD))하고, GPU에 최적화 된 형태로 연산을 변환함. 이를 통해 기존의 Mamba 에 비해서 2~8 배 정도 빠르게 연산이 가능하게 만들어줌.



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba 2

- Mamba is Semiseparable Matrix
  - Mamba를 모든 x 시퀀스에 대해서 정리를 하게 되면, Nth Semiseparable Matrix와 동치인 matrix로 추상화 할 수 있음.
  - 이러한 구조화된 Structed Matrix 형태로 된다는 점을 주목

$$h_t = A_t \dots A_1 B_0 x_0 + A_t \dots A_2 B_1 x_1 + \cdots + A_t A_{t-1} B_{t-2} x_{t-2} + A_t B_{t-1} x_{t-1} + B_t x_t$$

$$= \sum_{s=0}^{t} A_{t:s}^{\times} B_s x_s.$$

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

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

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

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

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

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

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba 2

- Attention is also Structed Matrix
  - Softmax Attention($softmax(QK^T)V$)와 Linear Attention($\varphi(Q)\varphi(K)V$) 연산 같은 경우도 SSM 과 같이 구조화 된 Structed Matrix로 구분할 수 있고, 일반적인 Masked Attention 같은 경우, Softmax Attention을 Linear Attention 화 한다고 하면 RNN 형태로 변환이 가능할 수 있는 형태처럼 변환이 가능함.($\varphi(Q)\varphi(K)\mathrm{y_t} = y_{t-1} + \varphi(Q)\varphi(K)$)

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

$$
\exp(QK^\top) = \varphi(Q)\varphi(K)^\top.
$$

$$
y = (L \circ (QK^\top)) \cdot V.
$$

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

$$
y = \begin{bmatrix} 1 & & \\ \vdots & \ddots & \\ 1 & \dots & 1 \end{bmatrix} x \quad \Longleftrightarrow \quad \begin{aligned} y_0 &= x_0 \\ y_t &= y_{t-1} + x_t \end{aligned}.
$$

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba 2

- Structured State Space Duality
  - 결과적으로 SSM 과 Attention 은 L(마스크)를 구성하는 방법이 다를 뿐인 같은 형태라고 볼 수 있음.

Method	Online Learning Objective $L_t(S)$	Online Update
LA	$\|\mathbf{S}_t - \mathbf{S}_{t-1}\|_F^2 - 2\langle \mathbf{S}_t \boldsymbol{k}_t, \boldsymbol{v}_t \rangle$	$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^T$
Mamba2	$\|\mathbf{S}_t - \alpha_t \mathbf{S}_{t-1}\|_F^2 - 2\langle \mathbf{S}_t \boldsymbol{k}_t, \boldsymbol{v}_t \rangle$	$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^T$

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)

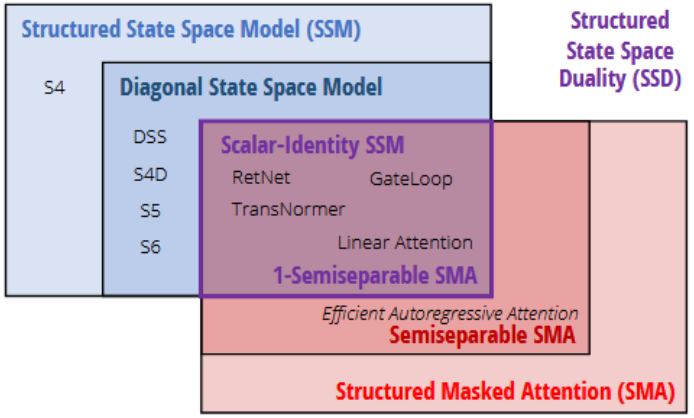


Figure 4: (**Structured State Space Duality.**) State space duality describes the close relationship between state space models and masked attention. (*Left*) General SSMs and SMA both possess linear and quadratic forms, with direct analogs in notation. (*Right*) SSMs and SMA intersect at a large class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

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba 2

- Structured State Space Duality
  - 결과적으로 SSM 과 Attention, RetNet, Toeplitz Fourier 전부 이러한 Structed Mask Attention으로 간단하게 표현 할 수 있음.

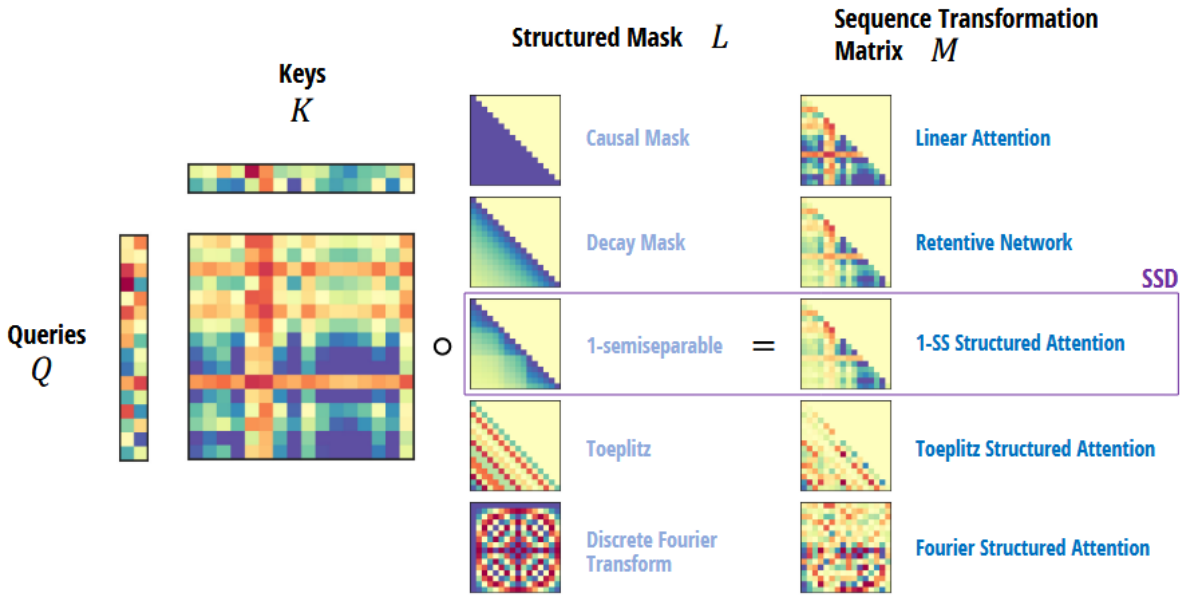


Figure 3: (**Structured Masked Attention.**) SMA constructs a masked attention matrix $M = QK^\top \circ L$ for any structured matrix L, which defines a matrix sequence transformation $Y = MV$. All instances of SMA have a dual subquadratic form induced by a different contraction ordering, combined with 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.

Mamba 2

- Semiseparable Matrix Multiplication
  - Semiseparable Matrix structure 의 경우 이런 식으로 쪼개서 계산할 수 있음. 이렇게 쪼개면, 각 연산에 대해서 병렬적으로 연산이 가능한 Matrix가 생성되는데 이 부분을 병렬적으로 연산할 수 있도록 만들어 속도의 최적화를 만듦.

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

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

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba 2

- Semiseparable Matrix Multiplication
  - Semiseparable Matrix structure 의 경우 이런 식으로 쪼개서 계산할 수 있음. 이렇게 쪼개면, 각 연산에 대해서 병렬적으로 연산이 가능한 Matrix가 생성되는데 이 부분을 병렬적으로 연산할 수 있도록 만들어 속도의 최적화를 만듦.

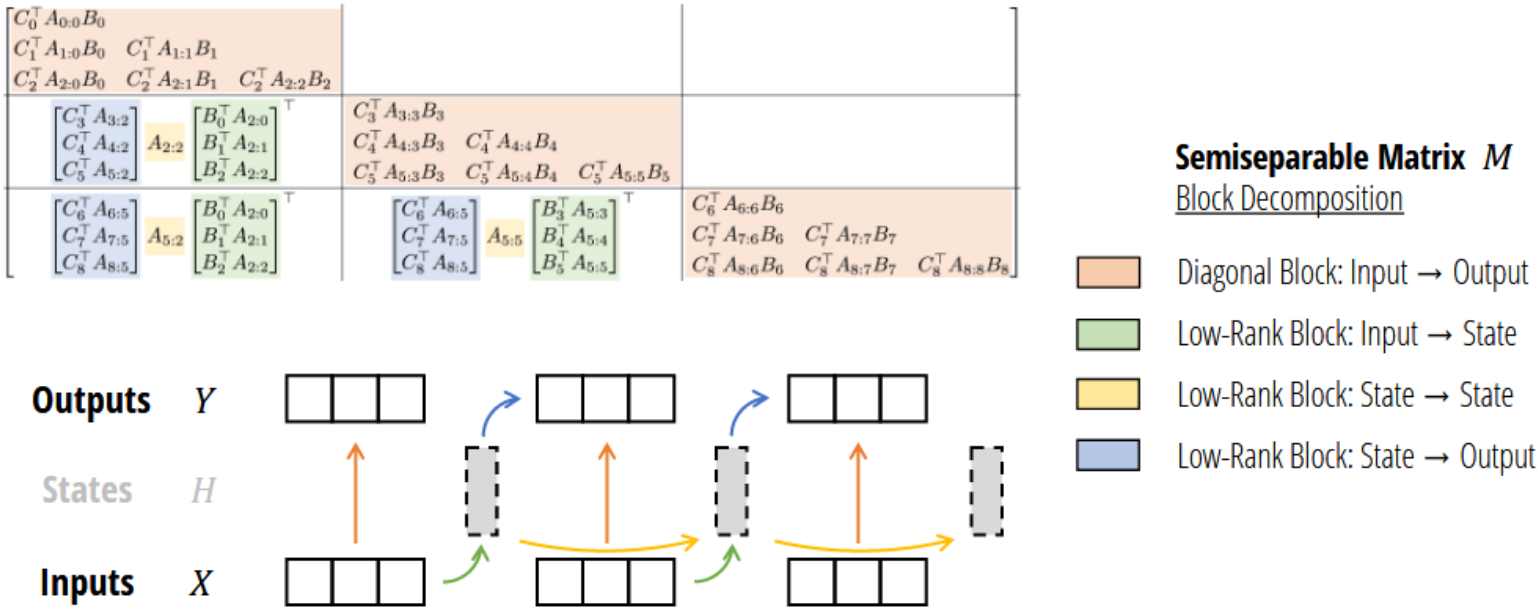


Figure 5: **(SSD Algorithm.)** By using the matrix transformation viewpoint of state space models to write them as semiseparable matrices (Section 3), we develop a more hardware-efficient computation of the SSD model through a block-decomposition matrix multiplication algorithm. The matrix multiplication 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.

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba 2

- SSD Model

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
```

Mamba 2

- Mamba-2 Architecture
  - 기존의 S4 같은 경우 Multi-Input SSM형태(여러 개의 SSM 을 붙이는 구조)임. 이를 Grouped-Value Attention (GVA) 형태와 유사하게 변환함.

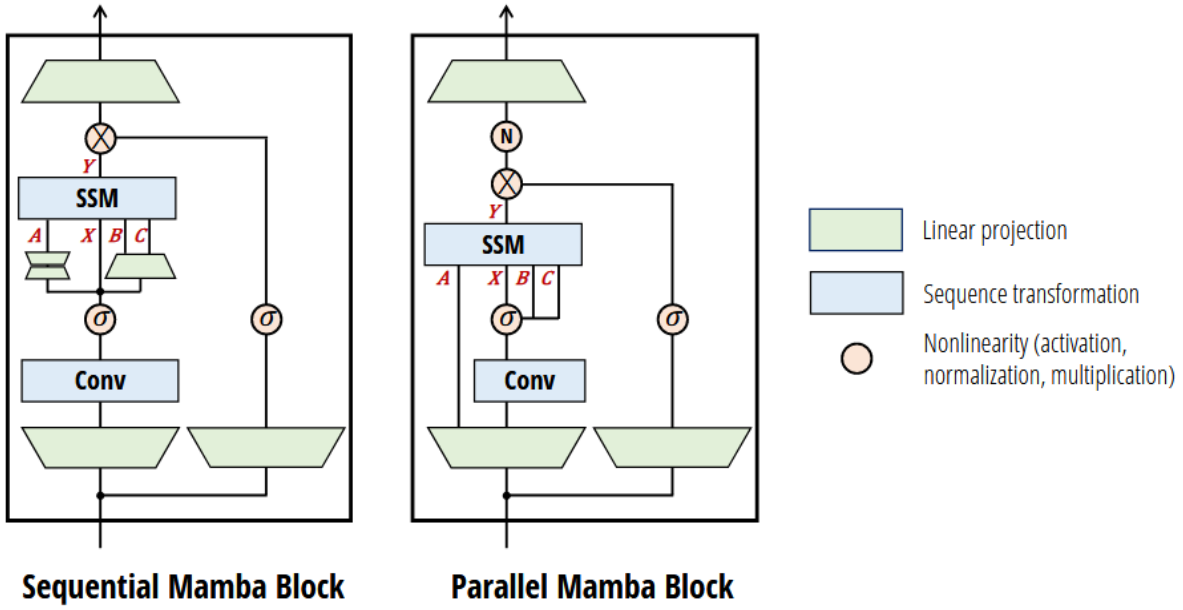


Figure 6: (**Mamba-2 Architecture.**) The Mamba-2 block simplifies the Mamba block by removing sequential linear projections; the SSM parameters A, B, C are produced at the beginning of the block instead of as a function of the SSM input X. An additional normalization layer is added as in NormFormer (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).

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba 2

- Tensor Parallel
  - Linear 연산을 한번에 모아서 연산하는 형태로 만들어서 최적화함. (이건 QKV 연산할 때도 꿀팁)

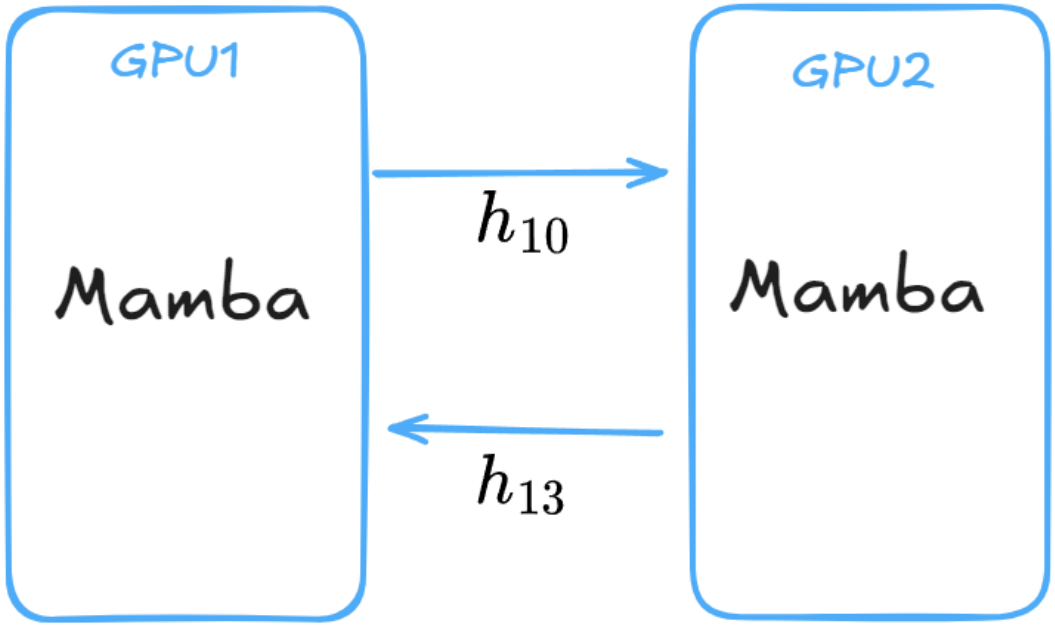
and $W^{(o)} = \begin{bmatrix} W_1^{(o)} \\ W_2^{(o)} \end{bmatrix}$ with $W_i^{(o)} \in \mathbb{R}^{ed/2 \times d}$. For $i = 1, 2$, the TP Mamba-2 layer can be written as:

$$
\begin{aligned}
x^{(i)} &= u W_i^{(x)\top} \in \mathbb{R}^{L \times ed/2} \\
z^{(i)} &= u W_i^{(z)\top} \in \mathbb{R}^{L \times ed/2} \\
\Delta^{(i)}, B^{(i)}, C^{(i)} &= \text{projection}(u) \quad \text{(one or more groups of } \Delta, B, C \text{ per GPU)} \\
x_c^{(i)} &= \text{conv1d}(x^{(i)}) \in \mathbb{R}^{L \times ed/2} \\
y^{(i)} &= SSM_{A,B,C,\Delta}(x_c^{(i)}) \in \mathbb{R}^{L \times ed/2} \\
y_g^{(i)} &= y^{(i)} \cdot \phi(z^{(i)}) \\
y_n^{(i)} &= \text{groupnorm}(y_g^{(i)}) \quad \text{(number of groups divisible by degree of tensor parallel)} \\
\text{out}^{(i)} &= y_g^{(i)} W_i^{(o)\top} \in \mathbb{R}^{L \times d/2} \\
\text{out} &= \sum_i \text{out}^{(i)}. \quad \text{(summing outputs from all GPUs with an all-reduce)}
\end{aligned}
$$

Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba 2

- Sequence Parallelism
  - 입력 sequenc가 길이가 매우 길면, 하나의 GPU에 전체 sequenc를 올릴 수 없는 문제가 있는데, 이를 Block 을 시간축으로 나눠서 여러 GPU가 나눠서 계산할 수 있도록 함.



Prof. Hyerim Bae (hrbae@pusan.ac.kr)

Mamba 2

- Scaling Laws(사이즈가 커지면 성능이 좋아진다) 법칙을 Mamba -2 도 지키는 것을 확인 할 수 있음.

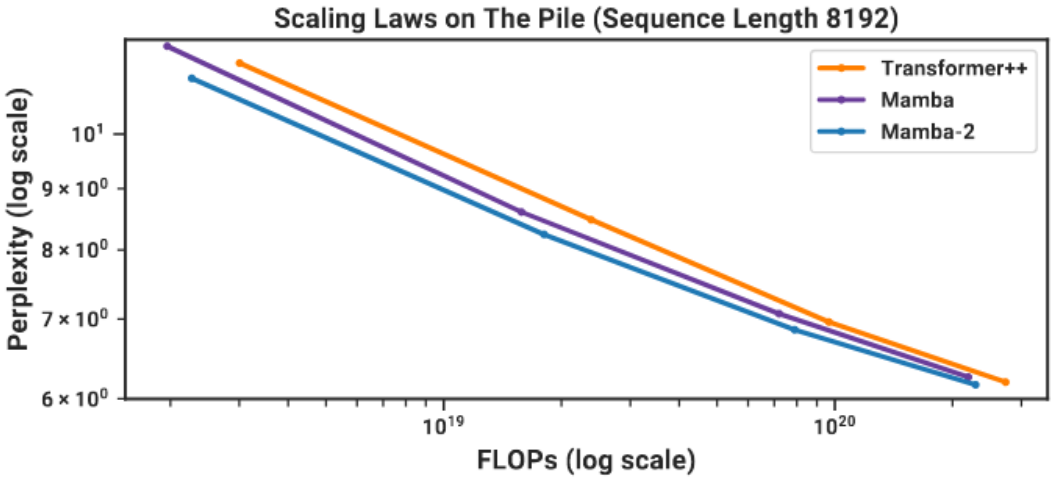


Figure 9: **(Scaling Laws.)** Models of size $\approx 125M$ to $\approx 1.3B$ parameters, trained on the Pile. Mamba-2 matches or exceeds the performance of Mamba as well as a strong "Transformer++" recipe. Compared to our Transformer baseline, Mamba-2 is Pareto dominant on performance (perplexity),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

Mamba 2

- Attention 에 비해서 연산 속도가 매우 빠른 것을 확인할 수 있다.

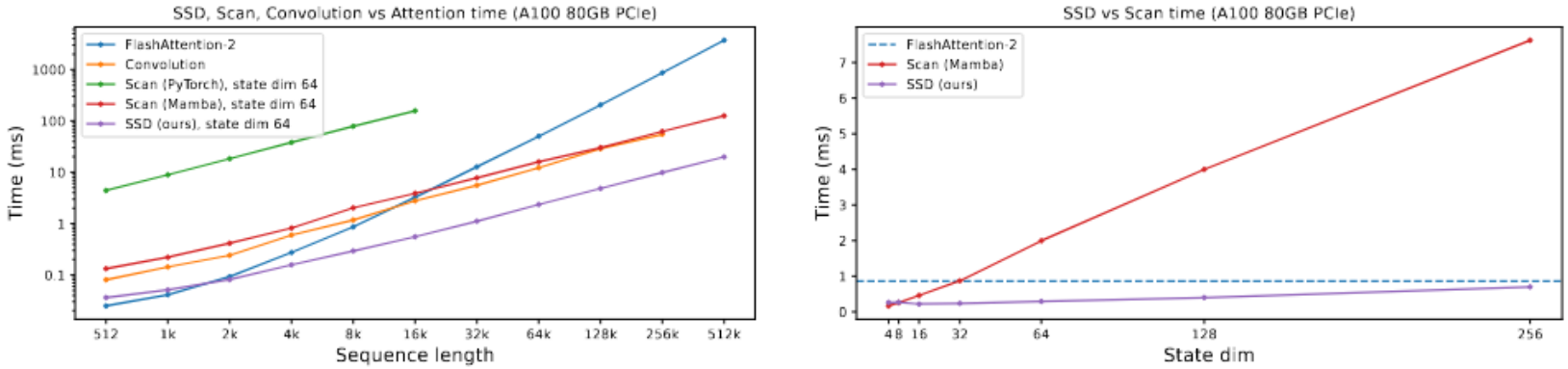


Figure 10: (**Efficiency Benchmarks.**) (*Left*) Our SSD is $2 - 8\times$ faster than a Mamba fused scan for large state expansion ($N = 64$) and faster than FlashAttention-2 for sequence length 2k and above. (*Right*) Sequence length 4K: Increasing state expansion slows down the Mamba optimized scan implementation linearly. SSD can handle much larger state expansion factors without much slowdown.

Mamba 2

- Language 성능을 비교해봤을 때, Transformer에 비해서 좋은 성능을 보이는 것을 알 수 있다.

Table 1: (**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. Full results in Table 10.

Model	Token.	Pile ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	Arc-E acc ↑	Arc-C acc ↑	WinoGrande acc ↑	OpenBookQA acc ↑	Average acc ↑
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**
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	34.4	50.3
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**
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	33.6	54.5
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**

Prof. Hyerim Bae (hrbae@pusan.ac.kr)