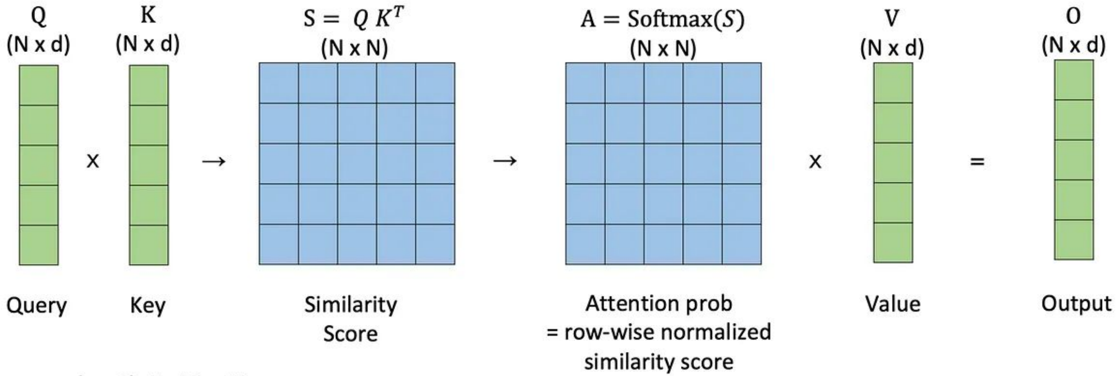
Sequence Modeling with State Space Models

Transformer에 대항하는 새로운 아키텍처의 등장

Transformer의 Attention Mechanism



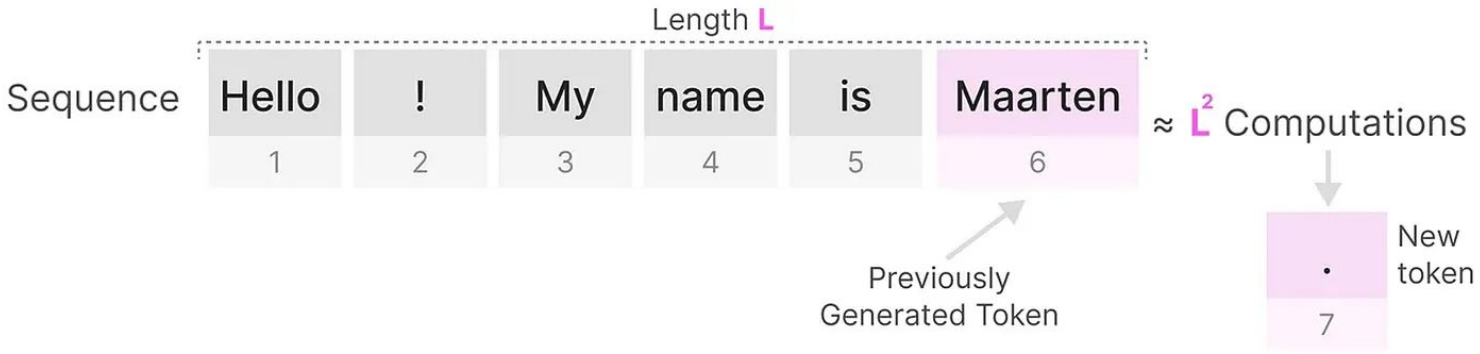
Typical sequence length N: 1K – 8K
Head dimension d: 64 – 128

$$\mathrm{Softmax}([s_1, \cdots, s_N]) = \left[\frac{e^{s_1}}{\sum_i e^{s_i}}, \cdots, \frac{e^{s_N}}{\sum_i e^{s_i}}\right]$$

$$O = \mathrm{Softmax}(QK^T)V$$

Attention scales quadratically in sequence length N

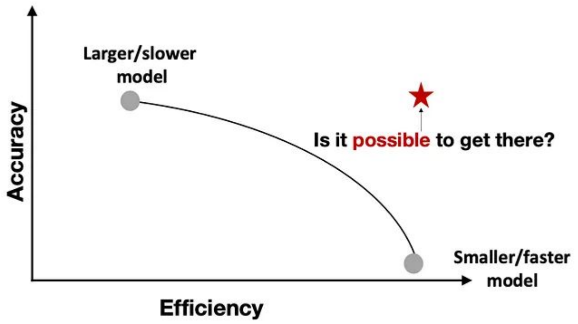
Transformer의 Attention Mechanism

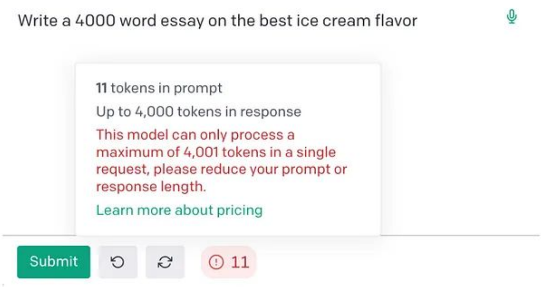


다음 토큰을 생성할 때, 이미 일부 토큰을 생성했더라도 전체 시퀀스에 대한 attention score를 다시 계산해야 합니다. -> 연산량이 너무 많다.. O(n^2)

Transformer의 Attention Mechanism

Core Challenge with Scale: Efficiency



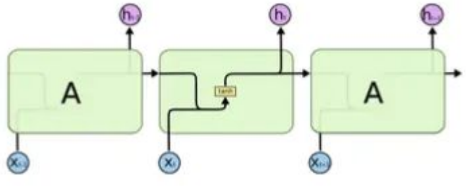


Efficiency eases training, deployment, and facilitates research

Efficiency unlocks new capabilities (e.g., long context)

Transformer와 RNN

Recurrent Neural Networks (RNN)

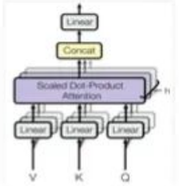


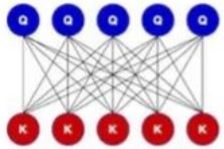
Sequential

✓ Natural autoregressive (causal) model

✗ Slow training on accelerators and poor optimization (vanishing gradients)

Attention (Transformers)





Dense interactions

✓ Strong performance, parallelizable

✗ Quadratic-time training, linear-time inference (in the length of the sequence)

RNN의 단점과 장점

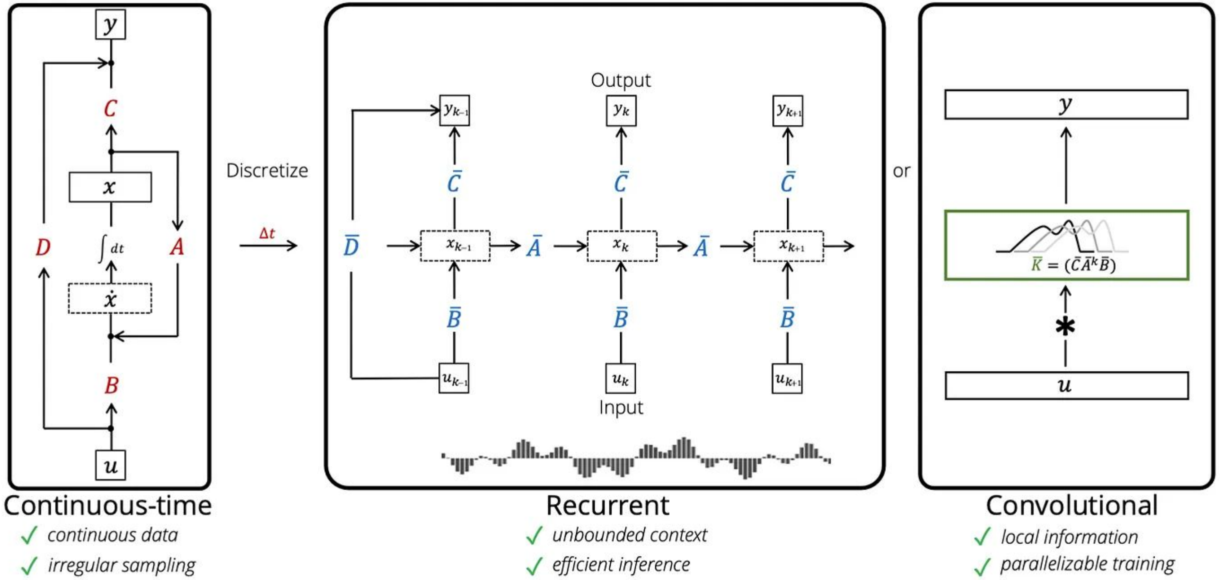
장점:

- 이전 메모리(hidden state)만 기억하고 있으면 되기 때문에 연산량이 적다.
- 추론 속도가 빠르다.

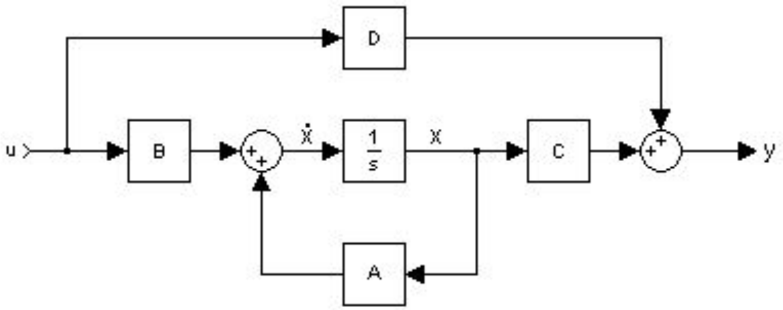
단점:

- 재귀적 특성때문에 학습이 느리다. (=수렴 속도가 느리다.)
- gradient vanishing/exploding 문제가 있다.
- 장기 의존성이 좋지 않다.

장점만 가지고 있을 수는 없을까?



State space representation



어떤 선형 시스템의 가장 일반적인 상태공간 표현식은 p개의 입력과 q개의 출력, n개의 상태 변수를 갖는 경우로, 아래와 같은 형태로 적을 수 있다:

$$\dot{\mathbf{x}}(t) = A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t)$$
$$\mathbf{y}(t) = C(t)\mathbf{x}(t) + D(t)\mathbf{u}(t)$$

Linear Time-Invariant, LTI SSM

선형성(Linear)

시스템이 선형적이라는 것은 입력의 선형 결합이 출력의 선형 결합으로 이어진다는 것을 의미합니다. **즉, 두 입력 신호의 합이 두 출력 신호의 합으로 나타납니다.**

시간 불변성(Time-Invariant)

시스템이 시간 불변적이라는 것은 시스템의 특성이 시간에 따라 변하지 않는다는 것을 의미합니다. **입력 신호가 시간적으로 이동하면, 출력 신호도 동일한 시간 이동을 겪습니다.**

State space Model

State equation $\mathbf{h}'(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}\mathbf{x}(t)$

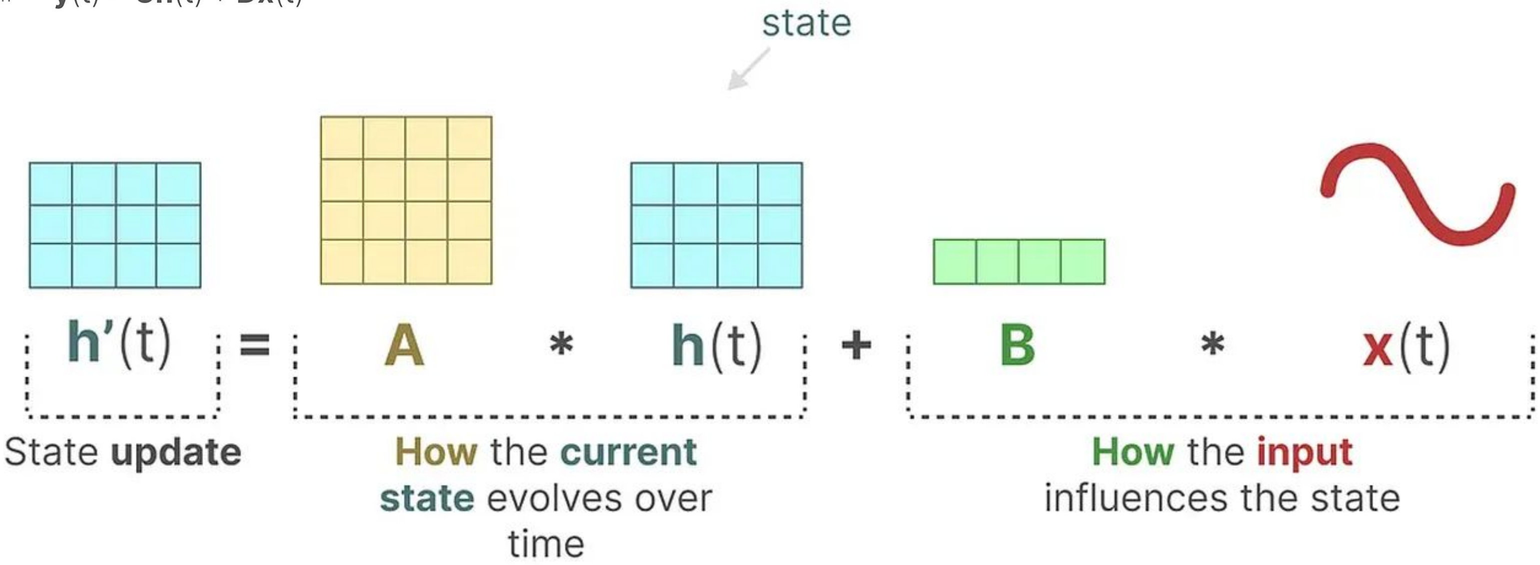
Output equation $\mathbf{y}(t) = \mathbf{C}\mathbf{h}(t) + \mathbf{D}\mathbf{x}(t)$

-> “상태 공간”은 모든 가능한 위치(상태)의 지도

State equation $\boxed{h'(t) = Ah(t) + Bx(t)}$

Output equation $y(t) = Ch(t) + Dx(t)$

State space Model Detail(1)



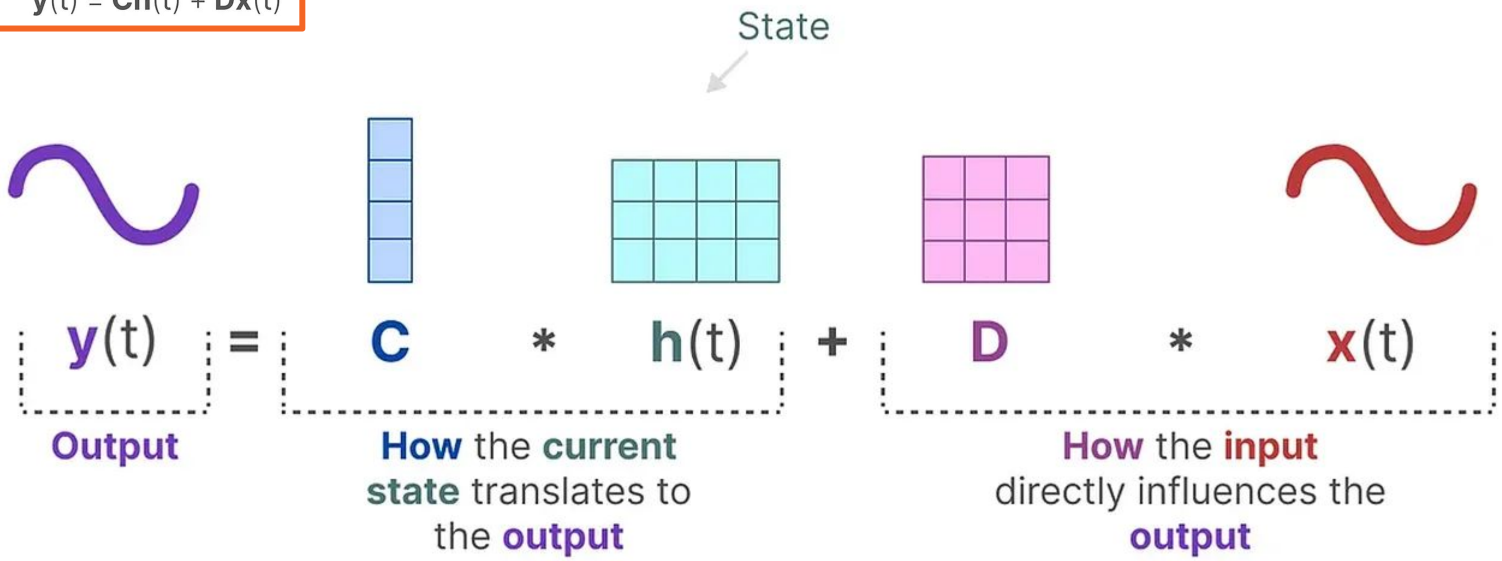
A: h(t), 상태가 어떻게 변하는지?

B: x(t), 입력이 상태에 어떻게 영향을 미치는지?

State equation $h'(t) = Ah(t) + Bx(t)$

Output equation $y(t) = Ch(t) + Dx(t)$

State space Model Detail(2)



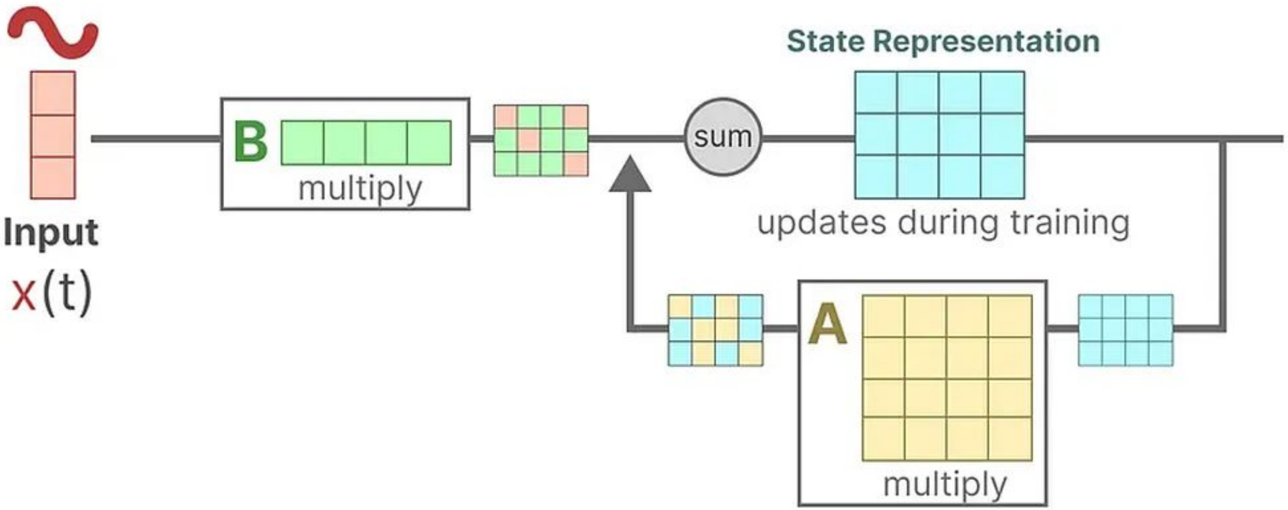
C: h(t), 상태가 어떻게 출력으로 변환되는지?

D: x(t), 입력이 출력에 어떻게 영향을 미치는지?

State equation $\boxed{h'(t) = Ah(t) + Bx(t)}$

Output equation $y(t) = Ch(t) + Dx(t)$

State equation

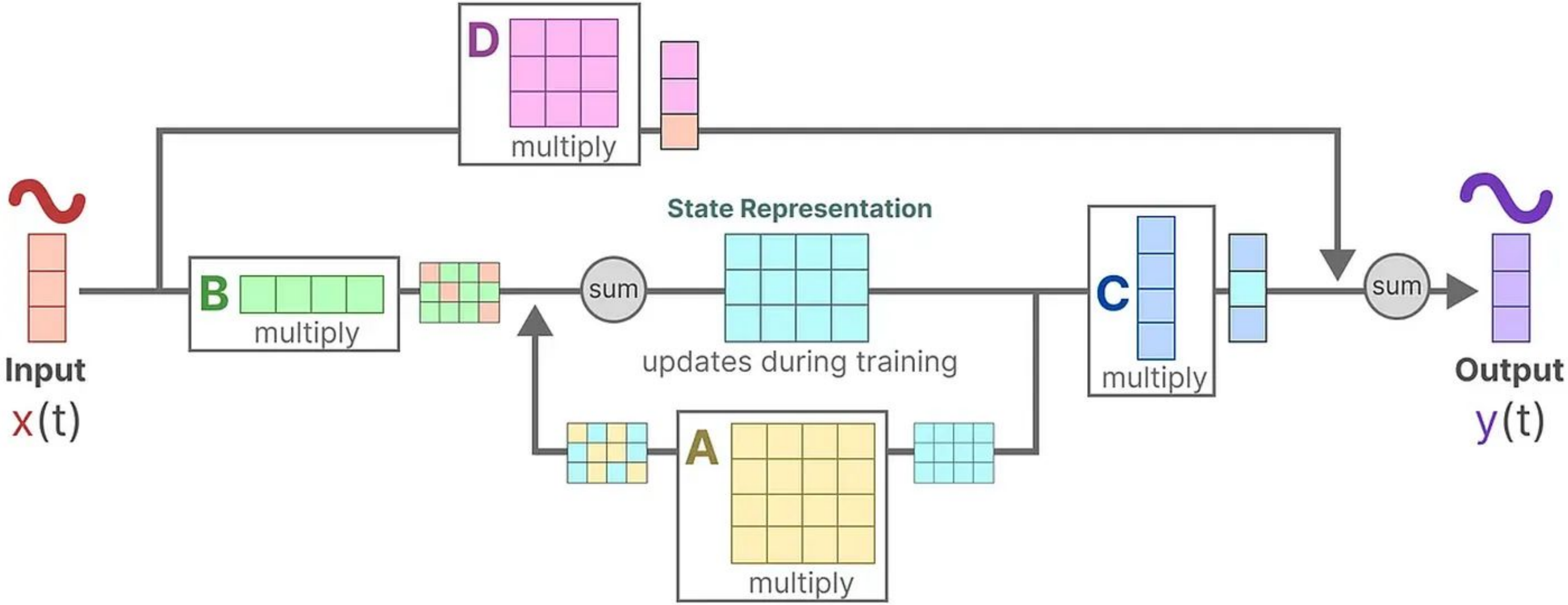


State Representation 부분이 일반적인 신경망의 hidden state와 비슷합니다(지식을 표현하는 잠재 공간)

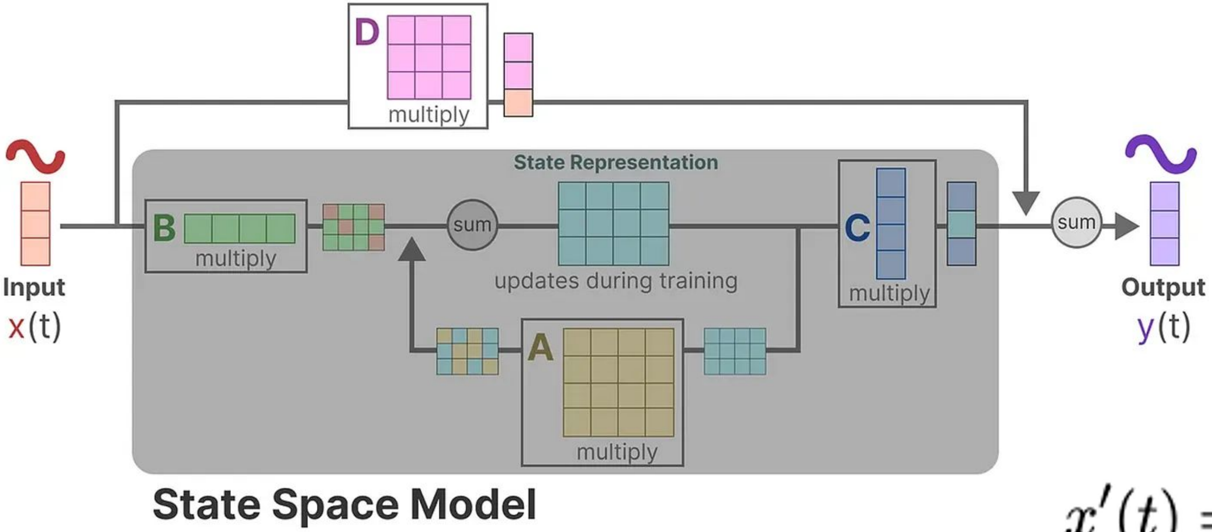
State equation $\mathbf{h'}(t) = \mathbf{Ah}(t) + \mathbf{Bx}(t)$

Output equation $\mathbf{y}(t) = \mathbf{Ch}(t) + \mathbf{Dx}(t)$

Output equation



State space Model

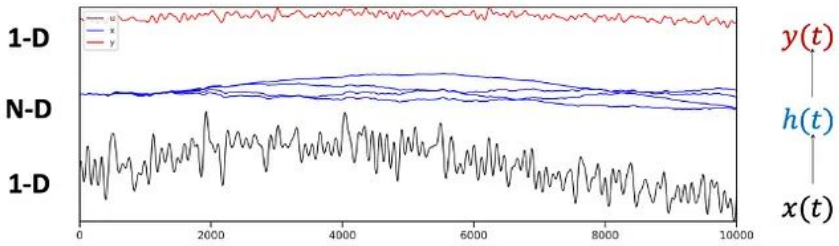


$$x'(t) = \boldsymbol{A}(t)x(t) + \boldsymbol{B}(t)u(t)$$

$$y(t) = \boldsymbol{C}(t)x(t)$$

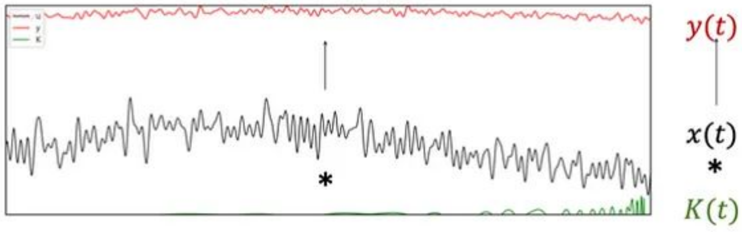
continuous convolution(LTI SSM)

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



Computing SSMs Convolutionally

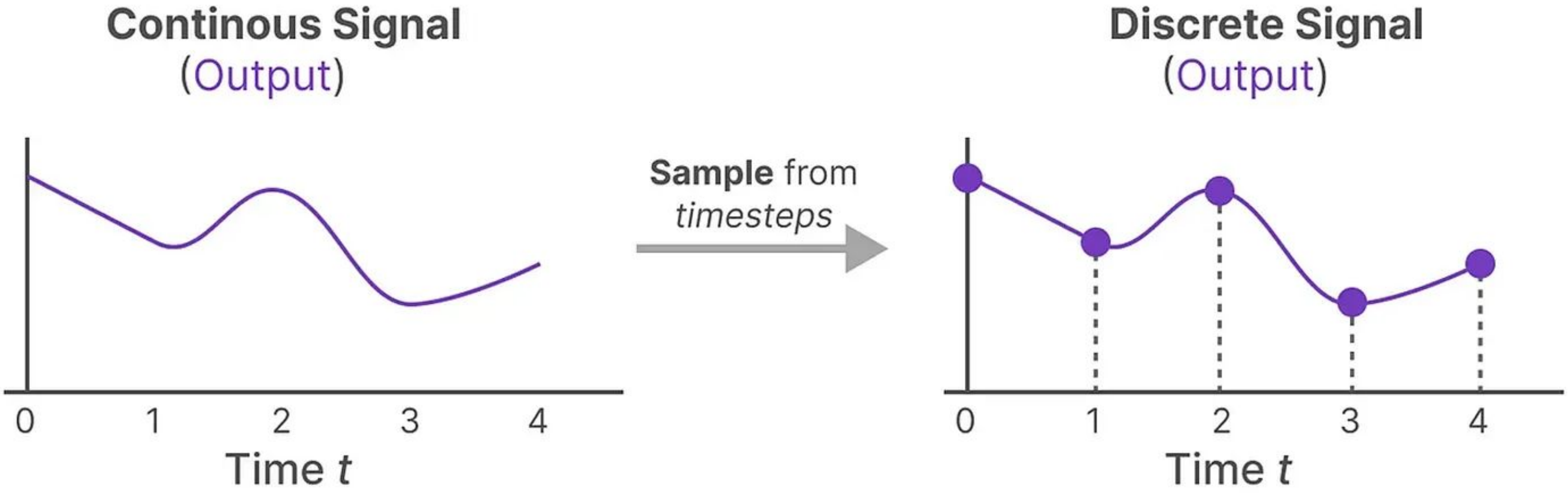
$$y(t) = x(t) * K(t)$$



Generalizes convolutional neural networks (CNN)

$$y(t) = (K * u)(t) = \int_0^\infty K(s) \cdot u(t-s)\, ds \qquad where \qquad K(t) = \boldsymbol{C}e^{t\boldsymbol{A}}\boldsymbol{B}.$$

Discretization



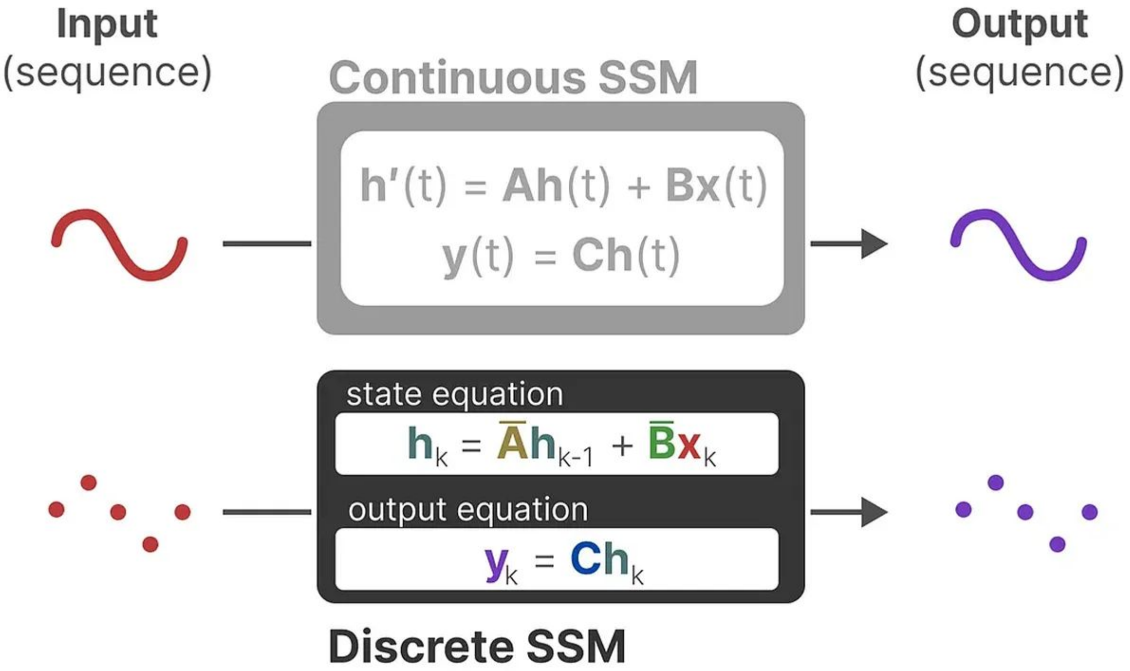
First-Order Approximation

$$
\begin{aligned}
x_k &= x_{k-1} + \Delta(\boldsymbol{A}x_{k-1} + \boldsymbol{B}u_k) \\
&= (\boldsymbol{I} + \Delta\boldsymbol{A})x_{k-1} + (\Delta\boldsymbol{B})u_k \\
&= \overline{\boldsymbol{A}}x_{k-1} + \overline{\boldsymbol{B}}u_k
\end{aligned}
$$

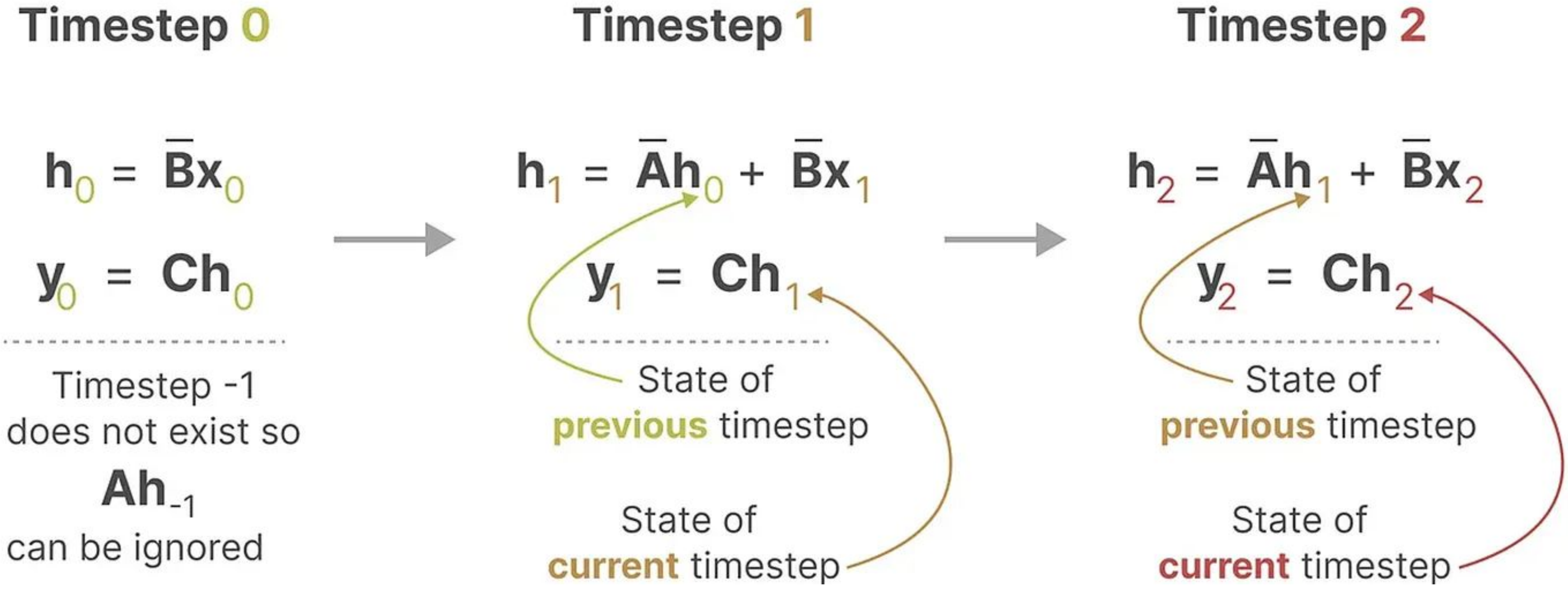
Δ로 샘플링하여 이산화를 진행한다. 실제로는 Bilinear, ZOH와 같은 방법이 사용된다.

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

discrete SSM



Recurrence (Efficient Inference)



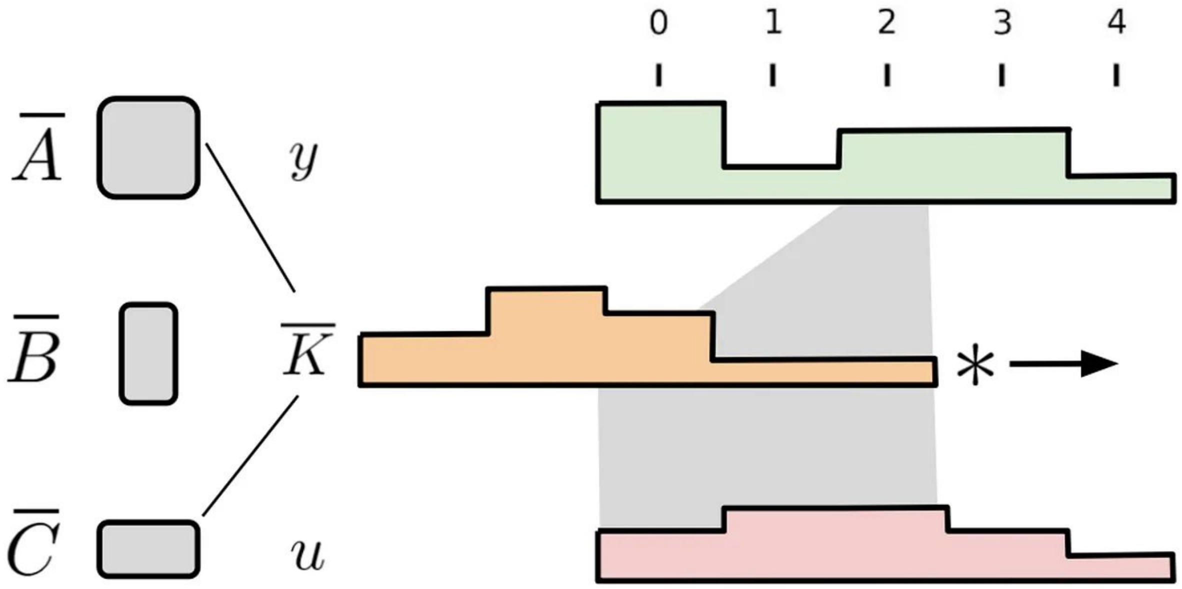
Convolutional(Efficient Training)

$$x_0 = \overline{\boldsymbol{B}} u_0 \qquad x_1 = \overline{\boldsymbol{A}\boldsymbol{B}} u_0 + \overline{\boldsymbol{B}} u_1 \qquad x_2 = \overline{\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{A}\boldsymbol{B}} u_1 + \overline{\boldsymbol{B}} u_2$$

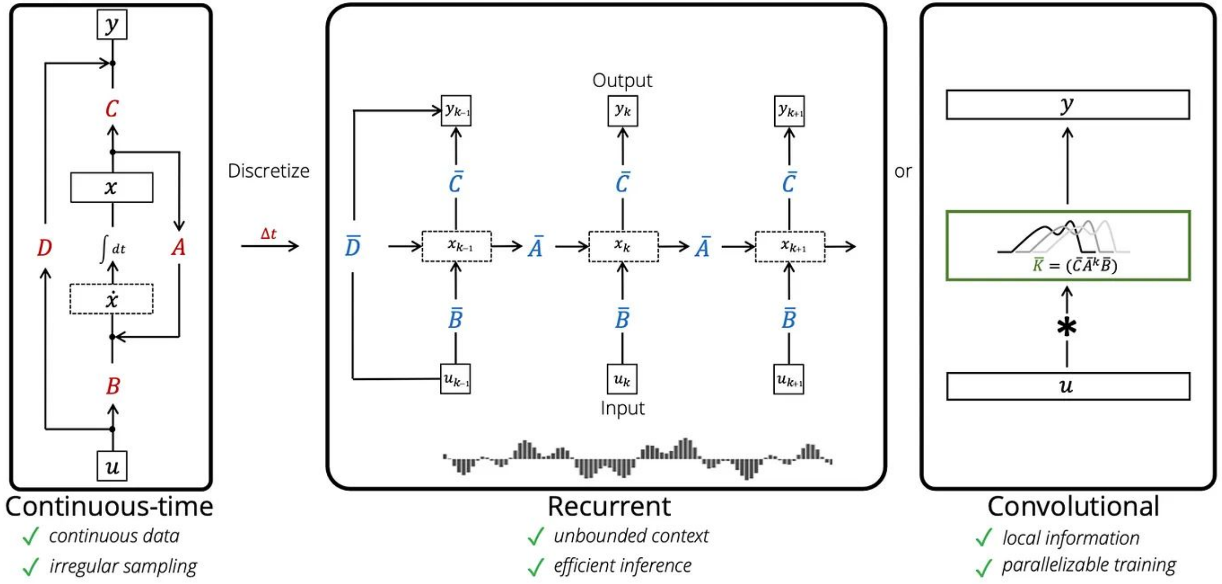
$$y_0 = \boldsymbol{C}\overline{\boldsymbol{B}} u_0 \qquad y_1 = \boldsymbol{C}\overline{\boldsymbol{A}\boldsymbol{B}} u_0 + \boldsymbol{C}\overline{\boldsymbol{B}} u_1 \qquad y_2 = \boldsymbol{C}\overline{\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \boldsymbol{C}\overline{\boldsymbol{A}\boldsymbol{B}} u_1 + \boldsymbol{C}\overline{\boldsymbol{B}} u_2$$

$$y_k = \boldsymbol{C}\overline{\boldsymbol{A}}^k \overline{\boldsymbol{B}} u_0 + \boldsymbol{C}\overline{\boldsymbol{A}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \boldsymbol{C}\overline{\boldsymbol{A}\boldsymbol{B}} u_{k-1} + \boldsymbol{C}\overline{\boldsymbol{B}} u_k.$$

Convolutional(Efficient Training)

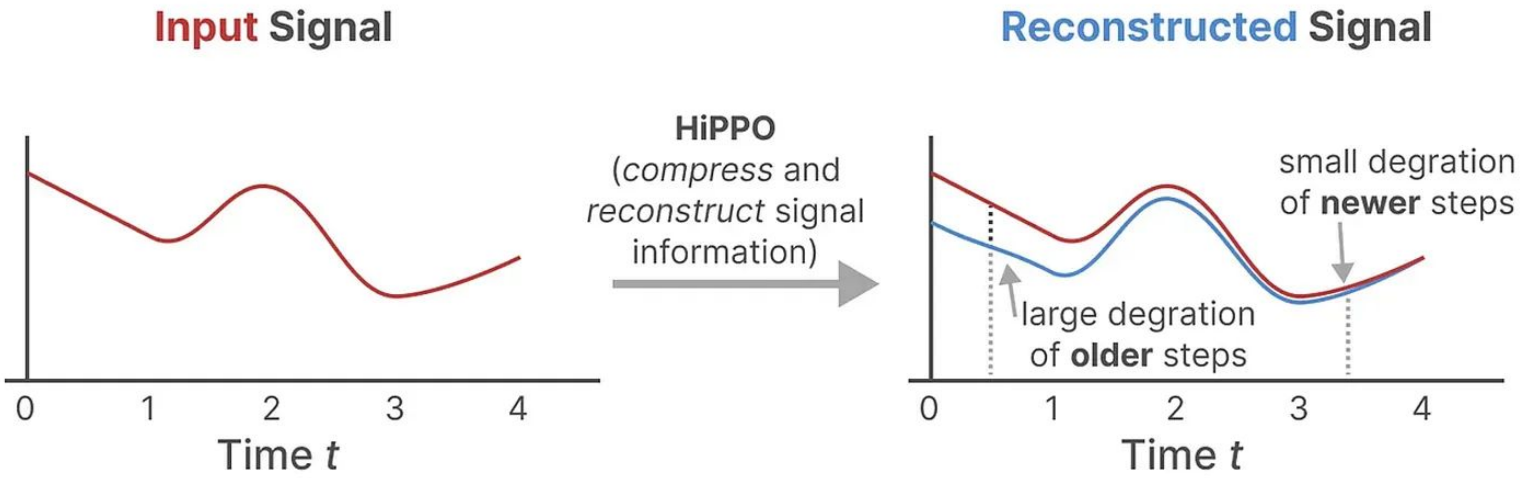


장점만 가지고 있을 수도 있겠다?



HiPPO(NIPS 2020)

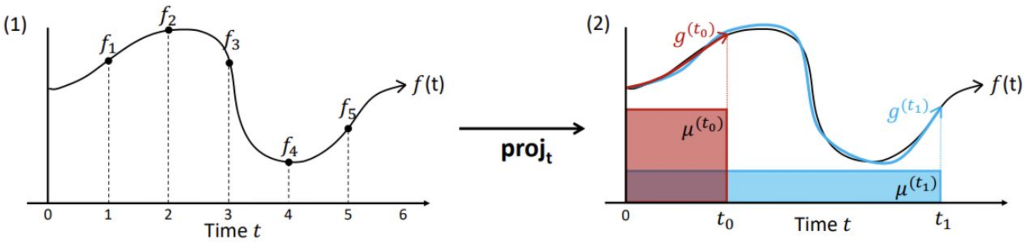
$$\text{(HiPPO Matrix)} \qquad A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

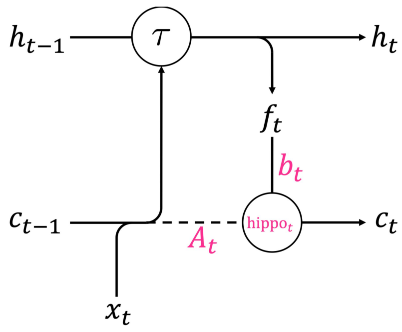


행렬 A를 사용하여 최근 토큰을 잘 포착하고 오래된 토큰을 감소시키는 상태 표현을 구축합니다. (장기 기억을 잘 유지하기 위함)

HiPPO - Detail

$$c_{t+1} = A_t c_t + B_t f_t$$





대량의 누적 데이터를 polynomial bases에 projection하는 것.

(4)

Discrete-time HiPPO Recurrence

$$c_{k+1} = A_k c_k + B_k f_k$$

discretize

(3)

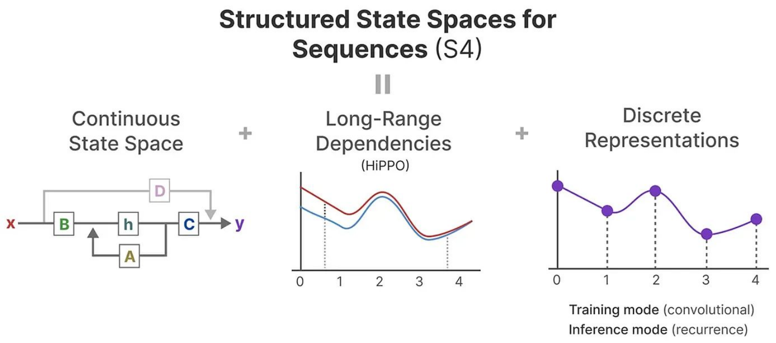
$$c(t_0) = \begin{bmatrix} 0.1 \\ -1.1 \\ 3.7 \\ 2.5 \end{bmatrix} \qquad c(t_1) = \begin{bmatrix} 1.5 \\ 2.9 \\ -0.3 \\ 2.0 \end{bmatrix}$$

Continuous-time HiPPO ODE

$$\frac{d}{dt}c(t) = A(t)c(t) + B(t)f(t)$$

모든 시간 t 에는 f를 polynomial-space로 optimal projection하는 g(t)가 있다.
(with measure μ(t) weighing the past.) 적절한 basis 선택으로 c(t) 는 history 를 나타내게 된다.

S4(ICLR 2022)

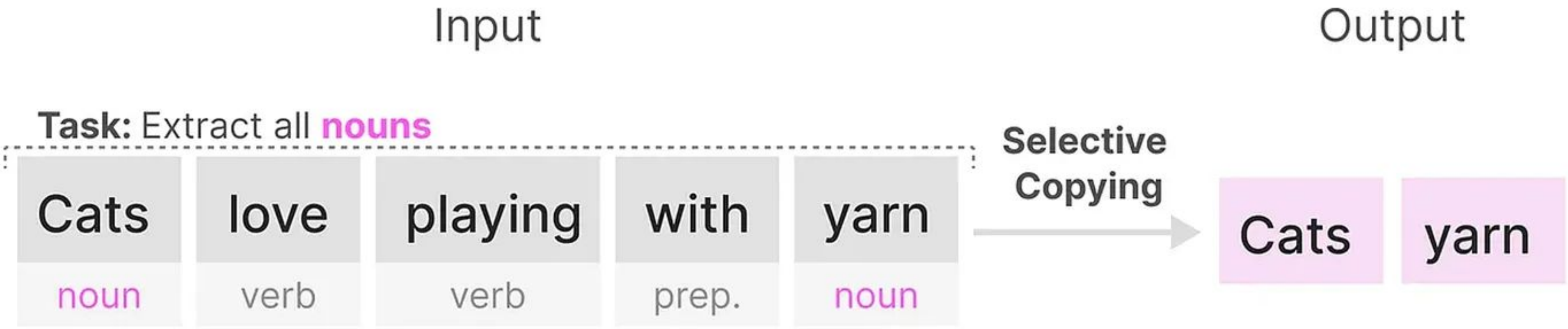


$$\overline{K} = (\overline{C}\overline{B}, \overline{C}\overline{A}\overline{B}, \dots, \overline{C}\overline{A}^{L-1}\overline{B})$$

이산 시간 SSM을 계산하는 데 있어서, 근본적인 병목 현상은 반복적인 행렬 곱셈을 포함한다는 점

$$(\boldsymbol{\Lambda} + \boldsymbol{P}\boldsymbol{Q}^*)^{-1} = \boldsymbol{\Lambda}^{-1} - \boldsymbol{\Lambda}^{-1}\boldsymbol{P}(1 + \boldsymbol{Q}^*\boldsymbol{\Lambda}^{-1}\boldsymbol{P})^{-1}\boldsymbol{Q}^*\boldsymbol{\Lambda}^{-1}$$

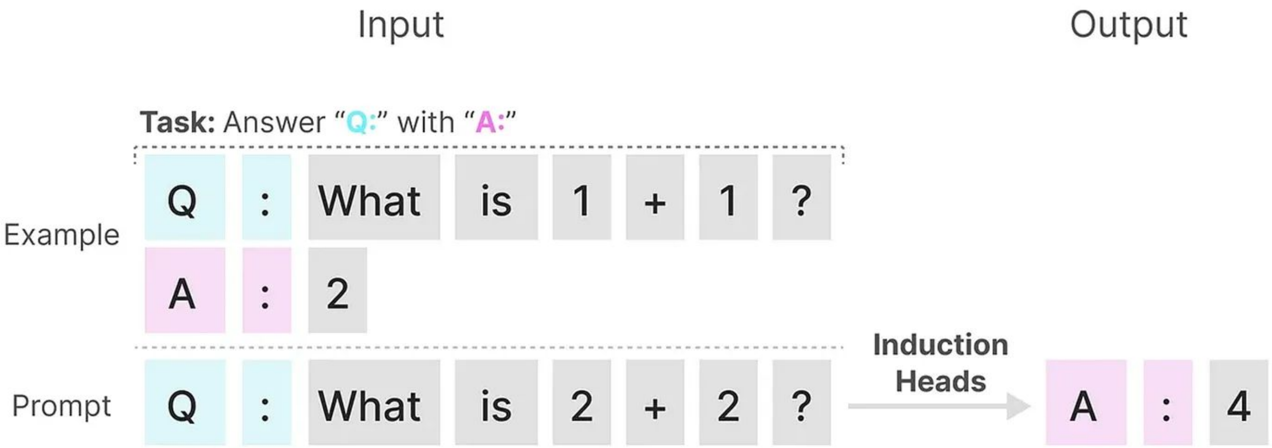
Selective Copying



입력의 일부를 복사하여 순서대로 출력하는 작업에서 SSM은 **LTI(선형 시간 불변성)**때문에 성능이 떨어집니다. **행렬 A, B, C는 SSM이 생성하는 모든 토큰에 대해 동일**하기 때문입니다.

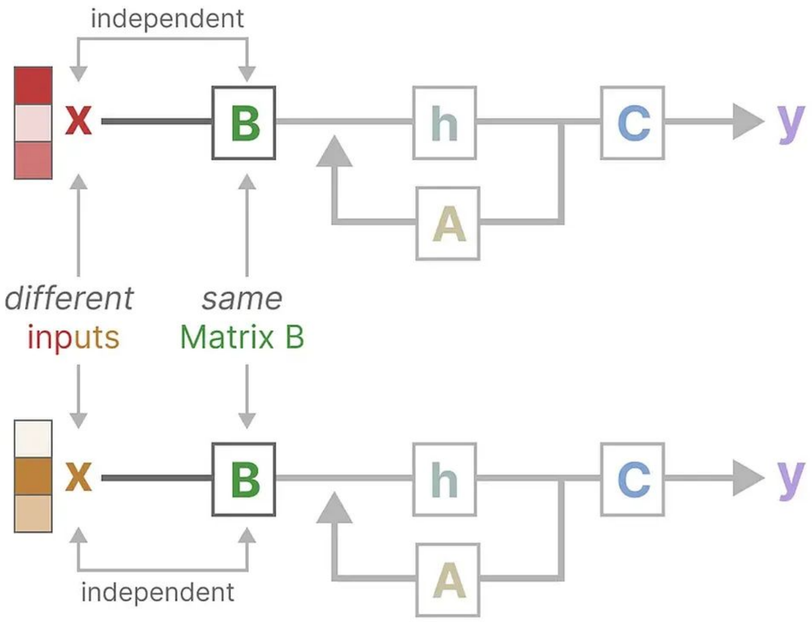
결과적으로, SSM은 내용 인식 추론(프롬프트에 대한 추론?)을 수행할 수 없습니다.

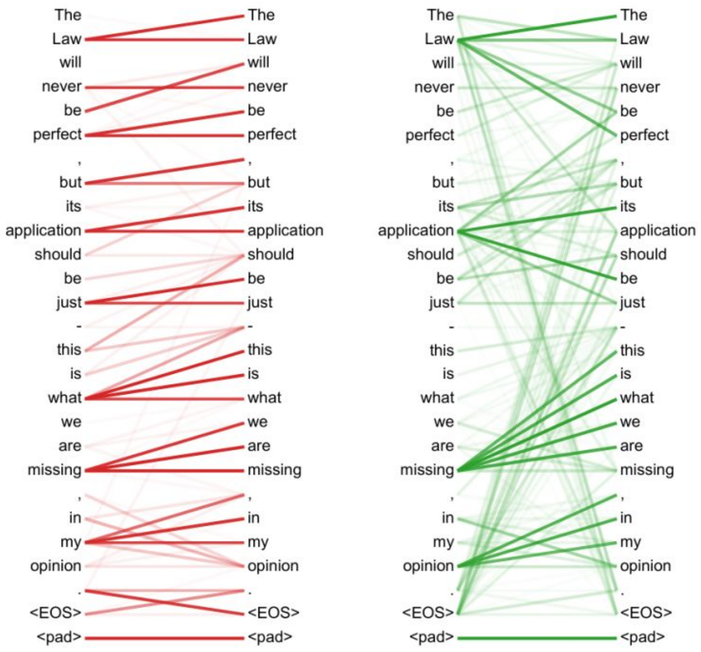
Induction heads



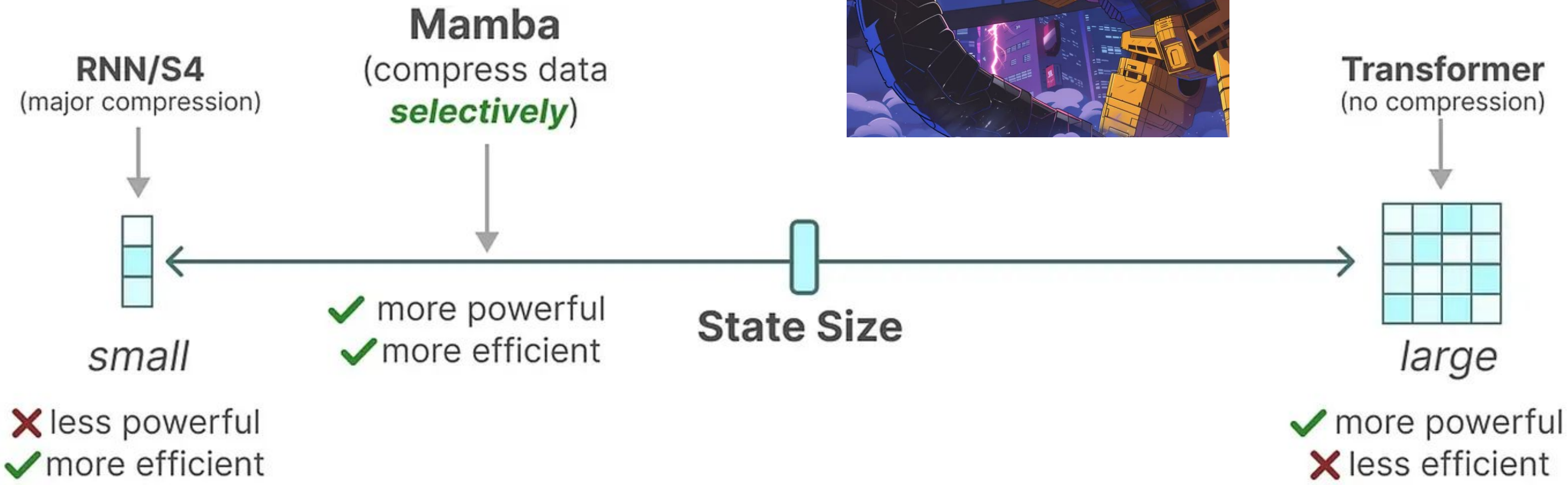
입력에서 발견된 패턴을 재현하는 작업에서 SSM이 LTI(시간 불변성)을 가지고 있기 때문에, 이전 토큰 중 어떤 것을 기억할지 선택할 수 없습니다.

LTI의 한계? Attention에선 쉬운데..

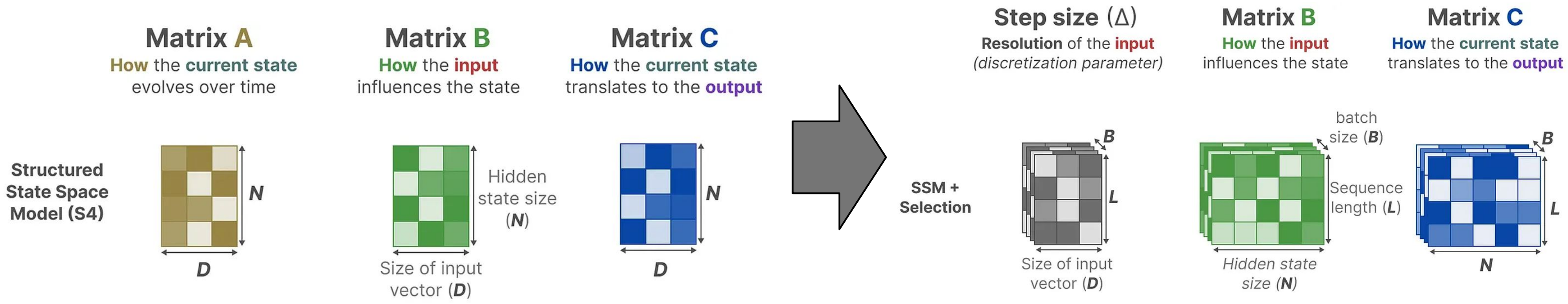




Mamba


RNN/S4
(major compression)
Mamba
(compress data *selectively*)
Transformer
(no compression)
State Size
small
large
less powerful
more efficient
more powerful
more efficient
more powerful
less efficient

New Matrix B, C

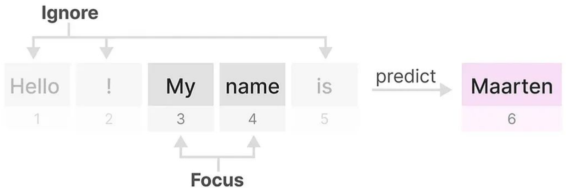


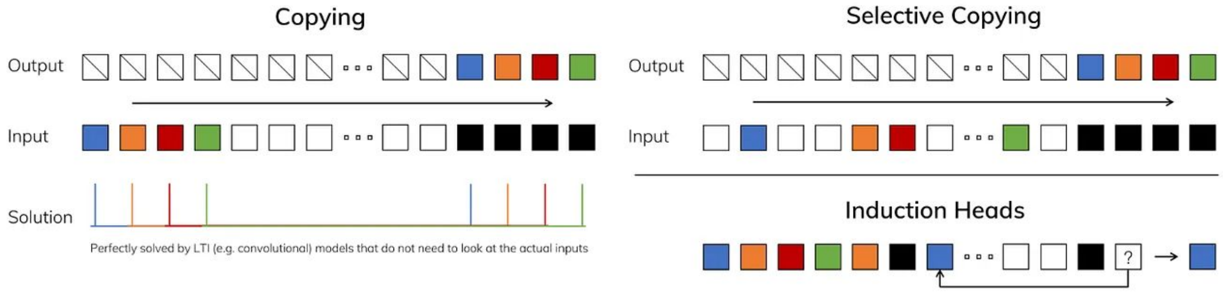
State equation $$\mathbf{h}'(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}\mathbf{x}(t)$$

Output equation $$\mathbf{y}(t) = \mathbf{C}\mathbf{h}(t)$$

위 수식과 같이 입력에 직접적으로 영향을 받는 B와 C에 대해 동적으로 영향을 받기 위함 -> A는 그대로 사용

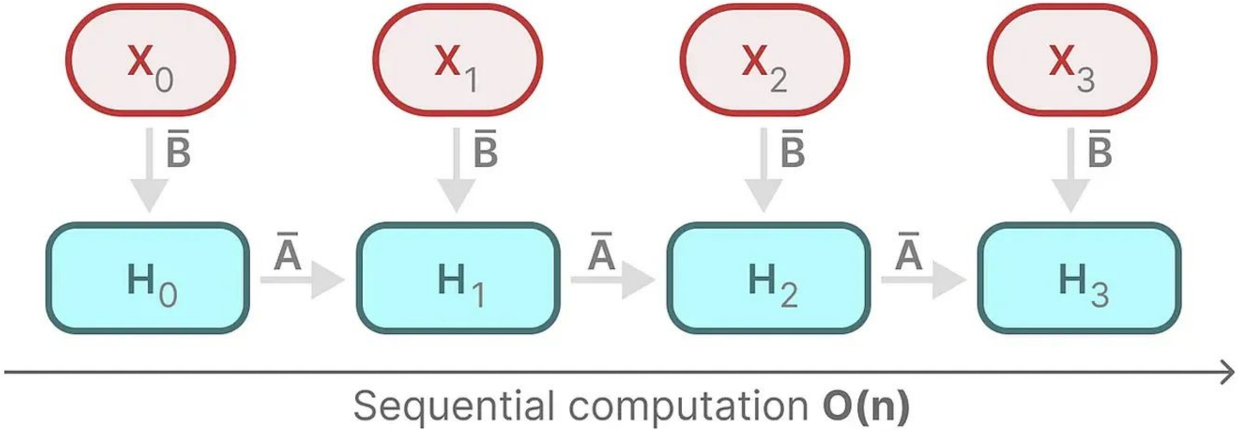
Selective Copying





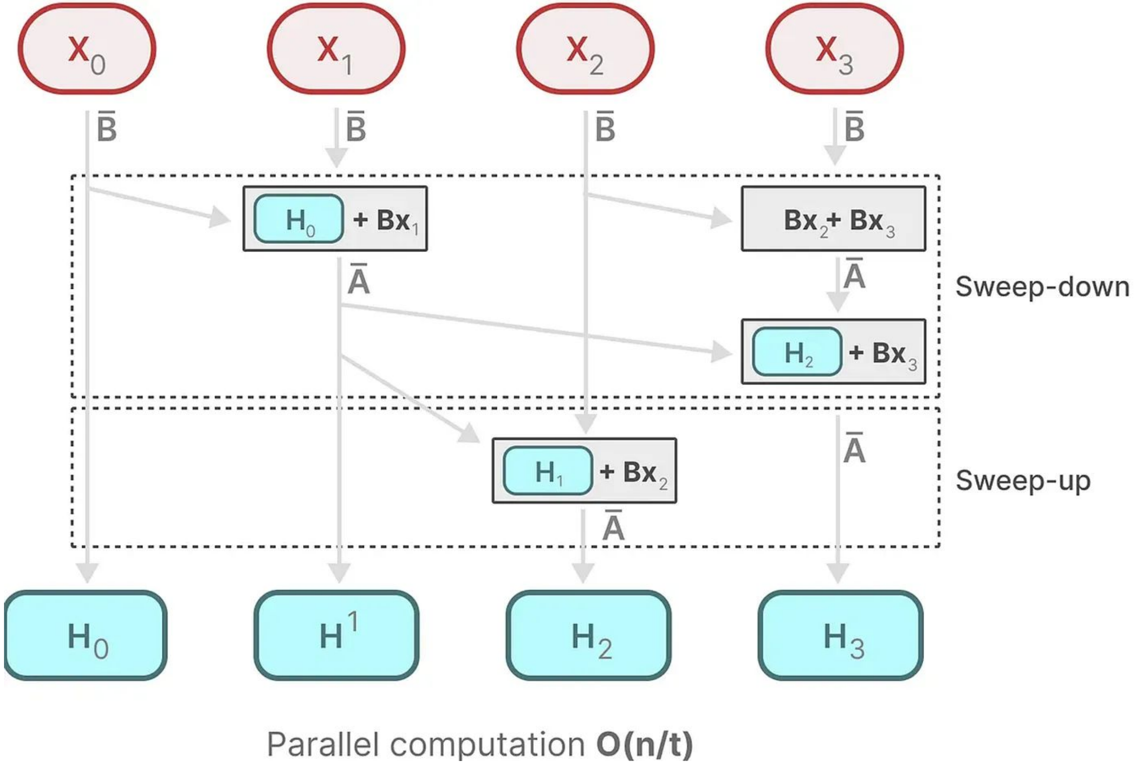
따라서 B와 C는 입력에 의존하여 hidden state에서 무엇을 유지하고 무엇을 무시할지 **선택적(selectively)으로 선택합니다.** (Δ에 따라 조절)

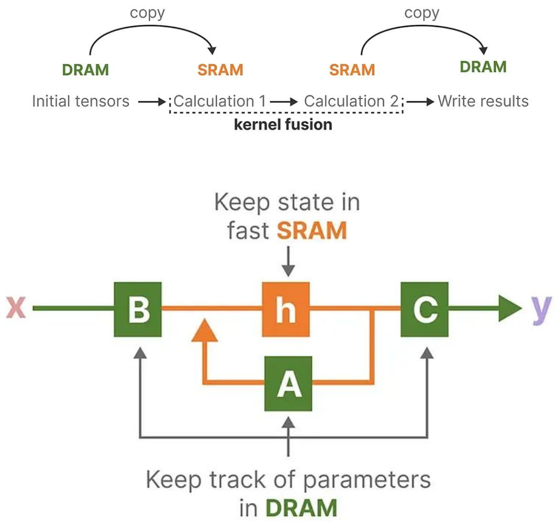
The Scan Operation



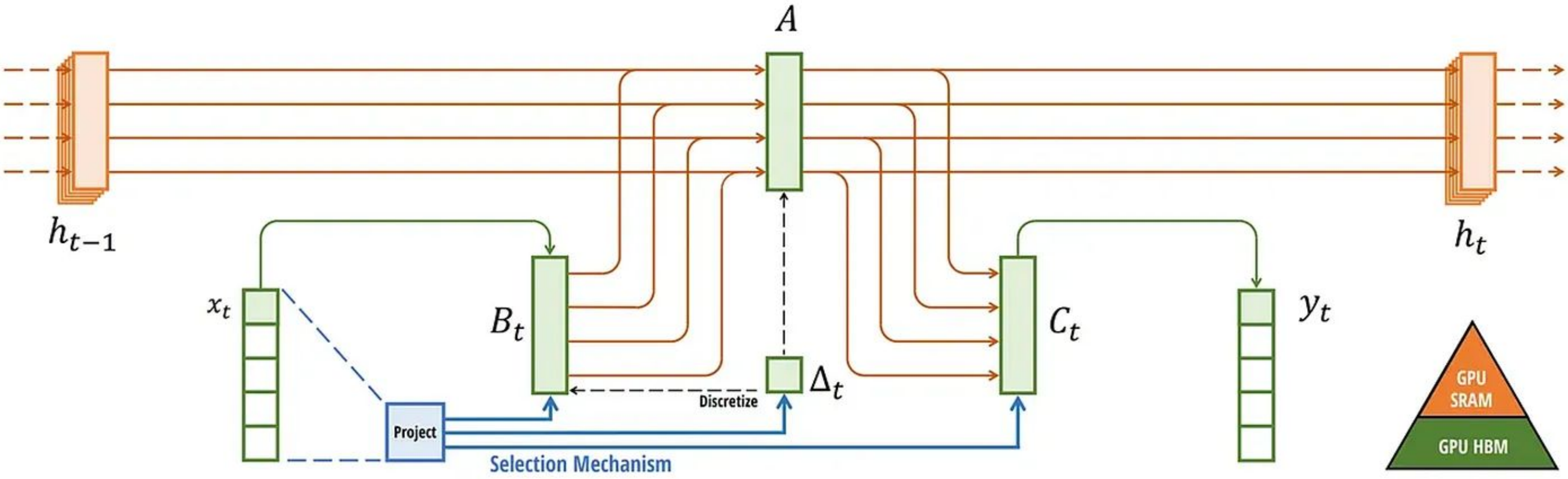
이러한 행렬이 이제 동적이기 때문에, 고정된 커널을 가정하는 컨볼루션 표현을 사용하여 계산할 수 없습니다. **따라서 컨볼루션의 병렬화를 잃고 재귀적 표현만을 사용할 수 있습니다(SSM의 장점 삭제)**

The Scan Operation



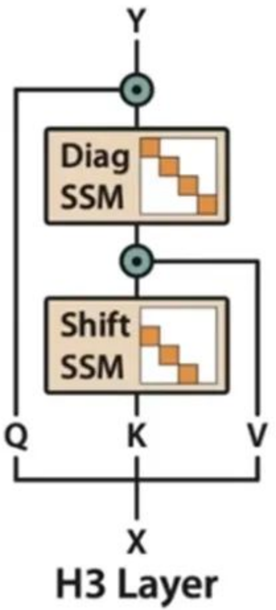


selective SSM 또는 S6 모델이라고



중간 상태들은 저장되지 않지만, 그래디언트를 계산하기 위해 역방향 패스에서 필요합니다. 대신, 저자들은 역방향 패스 동안 이러한 중간 상태들을 재계산합니다. 이것이 비효율적으로 보일 수 있지만, 상대적으로 느린 DRAM에서 모든 중간 상태를 읽는 것보다 훨씬 덜 비용이 많이 듭니다.

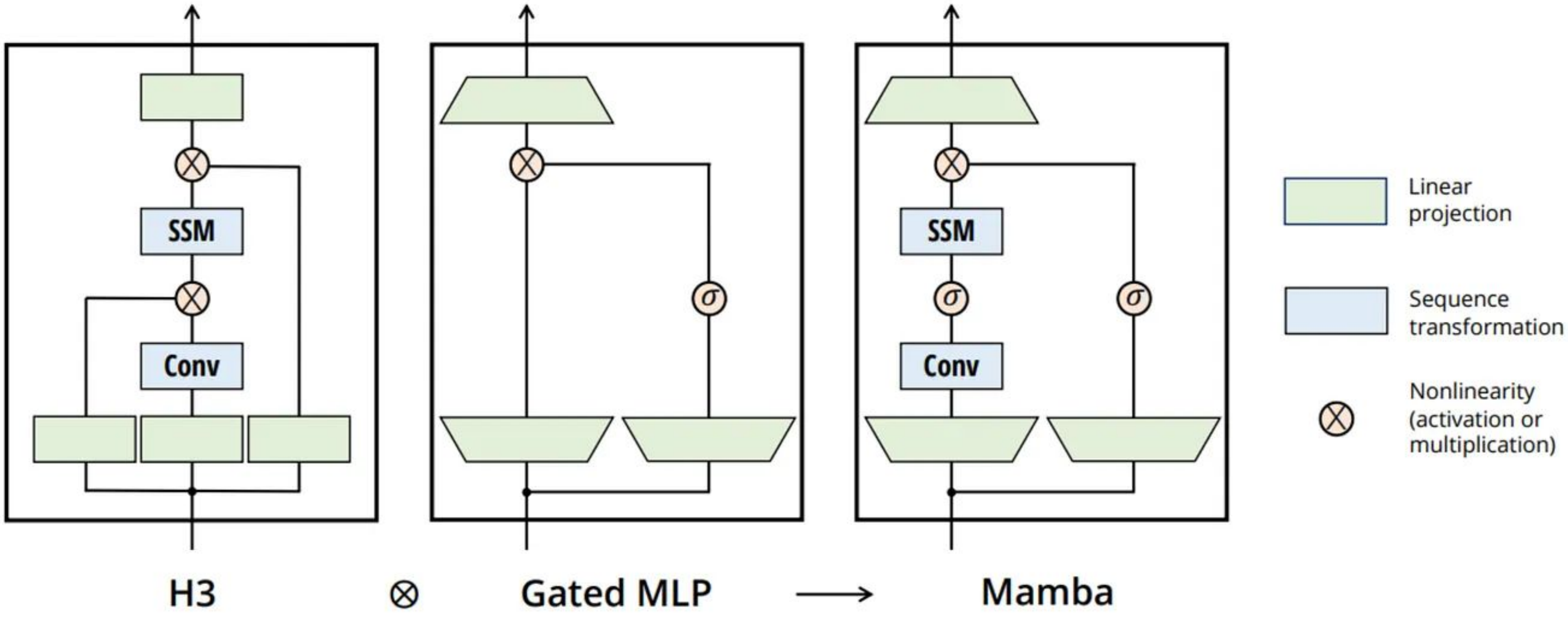
H3



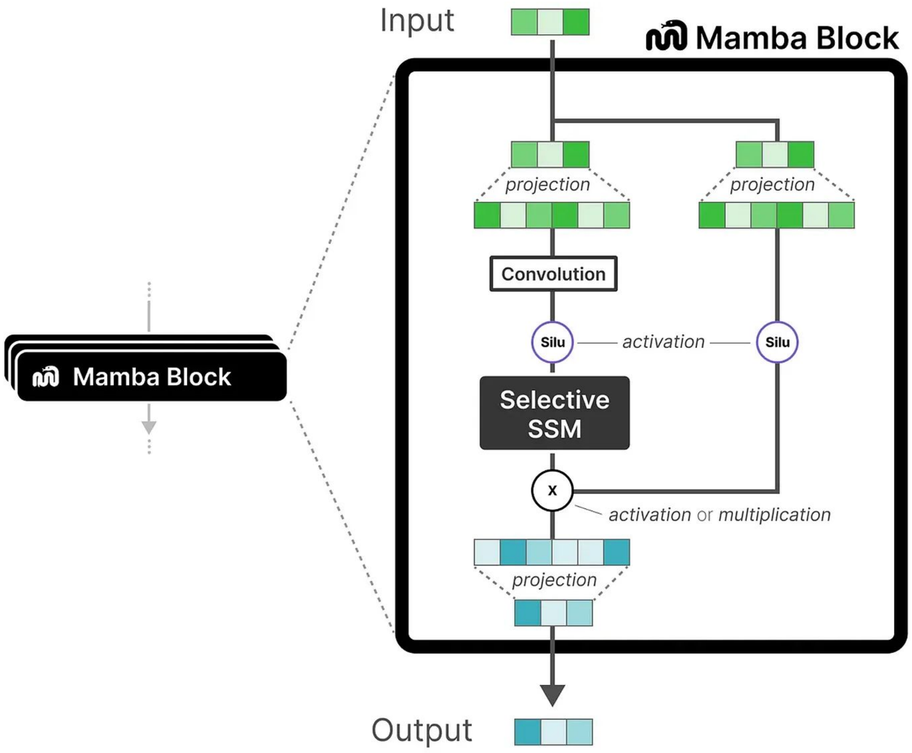
H3 Design

- Stacks **two** SSMs instead of one
- Shift SSM: local lookup across sequence
- Diag SSM: global memory

Mamba Block


SSM
Conv
σ
Linear projection
Sequence transformation
Nonlinearity (activation or multiplication)
H3
⊗
Gated MLP
Mamba

selective SSM



- 이산화를 통해 생성된 재귀적 SSM
- 장거리 의존성을 포착하기 위한 HiPPO 초기화가 있는 행렬 A
- 정보를 선택적으로 압축하기 위한 선택적 스캔 알고리즘
- 계산 속도를 높이기 위한 하드웨어 인식 알고리즘

-> 결론적으로는 빠른 추론과 훈련뿐만 아니라 무한한 맥락도 얻을 수 있습니다.

성능 + 효율

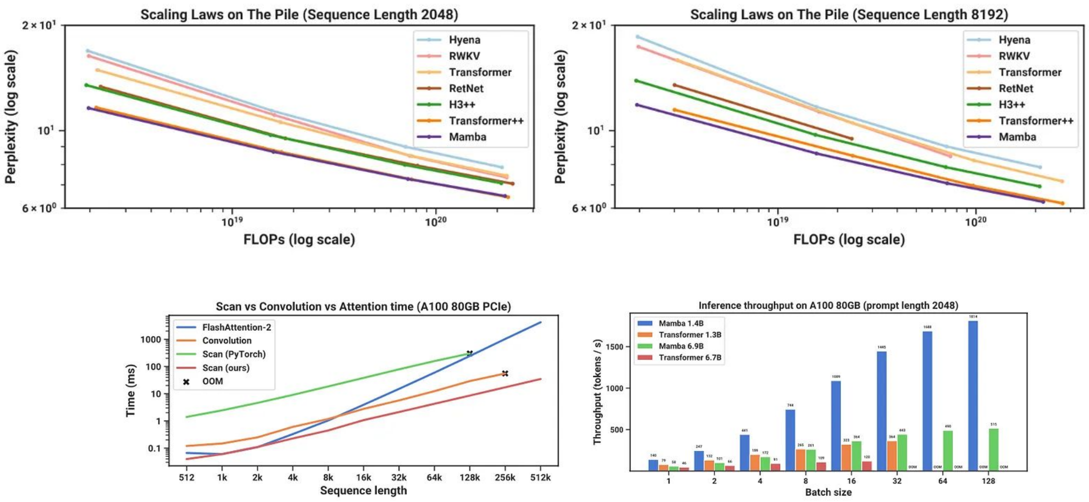


Figure 8: (**Efficiency Benchmarks**.) (*Left*) Training: our efficient scan is 40× faster than a standard implementation. (*Right*) Inference: as a recurrent model, Mamba can achieve 5× higher throughput than Transformers.

Table 3: (**Zero-shot Evaluations**.) Best results for each size in bold. 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 is best-in-class on every single evaluation result, and generally matches baselines 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 ↑	Average acc ↑
Hybrid H3-130M	GPT2	—	89.48	25.77	31.7	64.2	44.4	24.2	50.6	40.1
Pythia-160M	NeoX	29.64	38.10	33.0	30.2	61.4	43.2	24.1	**51.9**	40.6
Mamba-130M	NeoX	**10.56**	**16.07**	**44.3**	**35.3**	**64.5**	**48.0**	**24.3**	**51.9**	**44.7**
Hybrid H3-360M	GPT2	—	12.58	48.0	41.5	68.1	51.4	24.7	54.1	48.0
Pythia-410M	NeoX	9.95	10.84	51.4	40.6	66.9	52.1	24.6	53.8	48.2
Mamba-370M	NeoX	**8.28**	**8.14**	**55.6**	**46.5**	**69.5**	**55.1**	**28.0**	**55.3**	**50.0**
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	51.9
Mamba-790M	NeoX	**7.33**	**6.02**	**62.7**	**55.1**	**72.1**	**61.2**	**29.5**	**56.1**	**57.1**
GPT-Neo 1.3B	GPT2	—	7.50	57.2	48.9	71.1	56.2	25.9	54.9	52.4
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	53.0
OPT-1.3B	OPT	—	6.64	58.0	53.7	72.4	56.7	29.6	59.5	55.0
Pythia-1.4B	NeoX	7.51	6.08	61.7	52.1	71.0	60.5	28.5	57.2	55.2
RWKV-1.5B	NeoX	7.70	7.04	56.4	52.5	72.4	60.5	29.4	54.6	54.3
Mamba-1.4B	NeoX	**6.80**	**5.04**	**64.9**	**59.1**	**74.2**	**65.5**	**32.8**	**61.5**	**59.7**
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8	72.1	61.1	30.2	57.6	56.5
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	58.0
OPT-2.7B	OPT	—	5.12	63.6	60.6	74.8	60.8	31.3	61.0	58.7
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3	74.0	64.1	32.9	59.7	59.1
RWKV-3B	NeoX	7.00	5.24	63.9	59.6	73.7	67.8	33.1	59.6	59.6
Mamba-2.8B	NeoX	**6.22**	**4.23**	**69.2**	**66.1**	**75.2**	**69.7**	**36.3**	**63.5**	**63.3**
GPT-J-6B	GPT2	–	4.10	68.3	66.3	75.4	67.0	36.6	64.1	63.0
OPT-6.7B	OPT	–	4.25	67.7	67.2	76.3	65.6	34.9	65.5	62.9
Pythia-6.9B	NeoX	6.51	4.45	67.1	64.0	75.2	67.3	35.5	61.3	61.7
RWKV-7.4B	NeoX	6.31	4.38	67.2	65.5	76.1	67.8	37.5	61.0	62.5

출처

- Sequence Modeling with State Space Models
- A Visual Guide to Mamba and State Space Models
- Annotated-s4
- SeriesHiPPO
- cs224n-2024
- gu_dissertation-augmented