Mamba as a Bridge: Where Vision Foundation Models Meet Vision Language Models for Domain-Generalized Semantic Segmentation

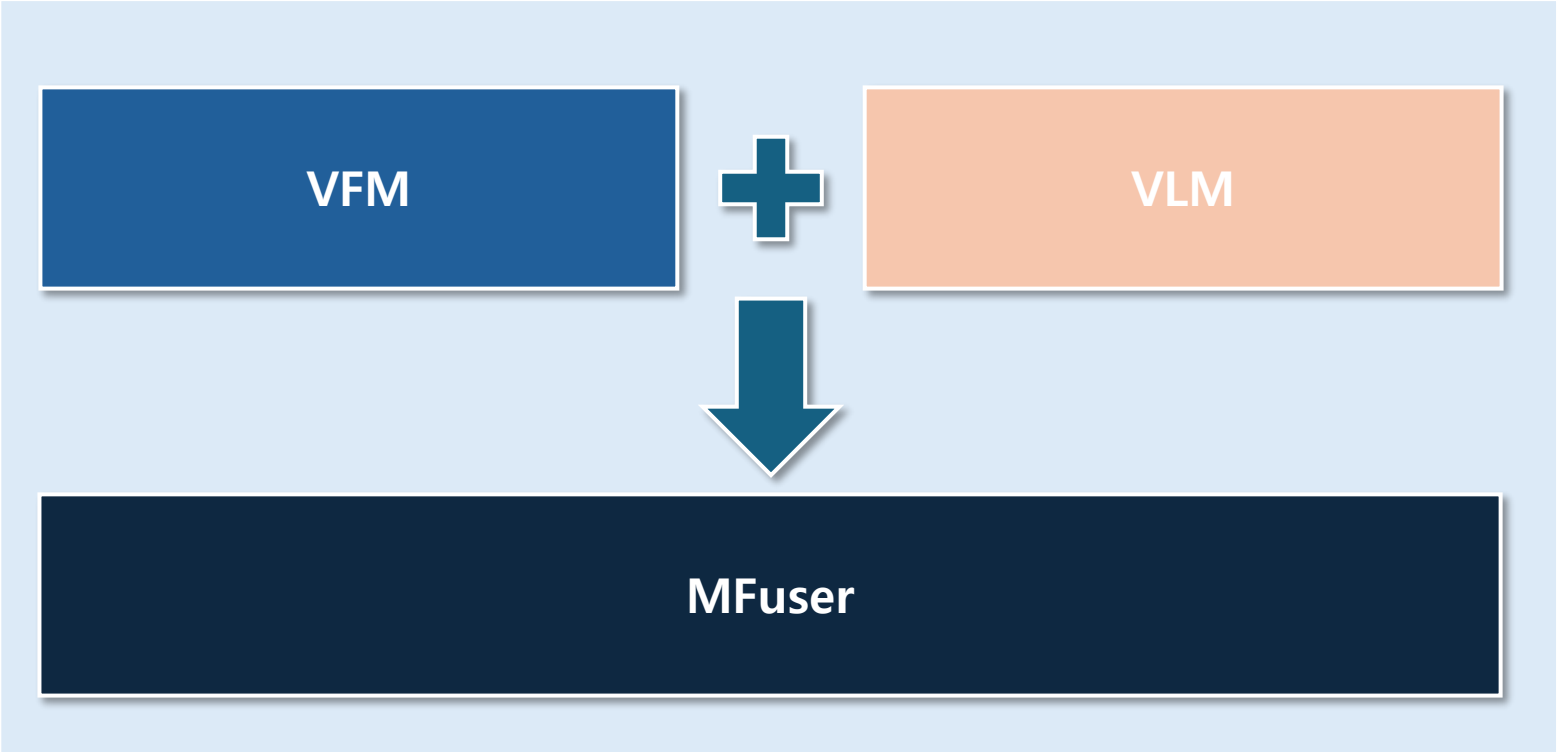
(Xin Zhang, Robby T. Tan, in CVPR 2025)

Department of Metaverse Convergence at Chung-Ang University
Myungjun Yun

Background

MFuser?

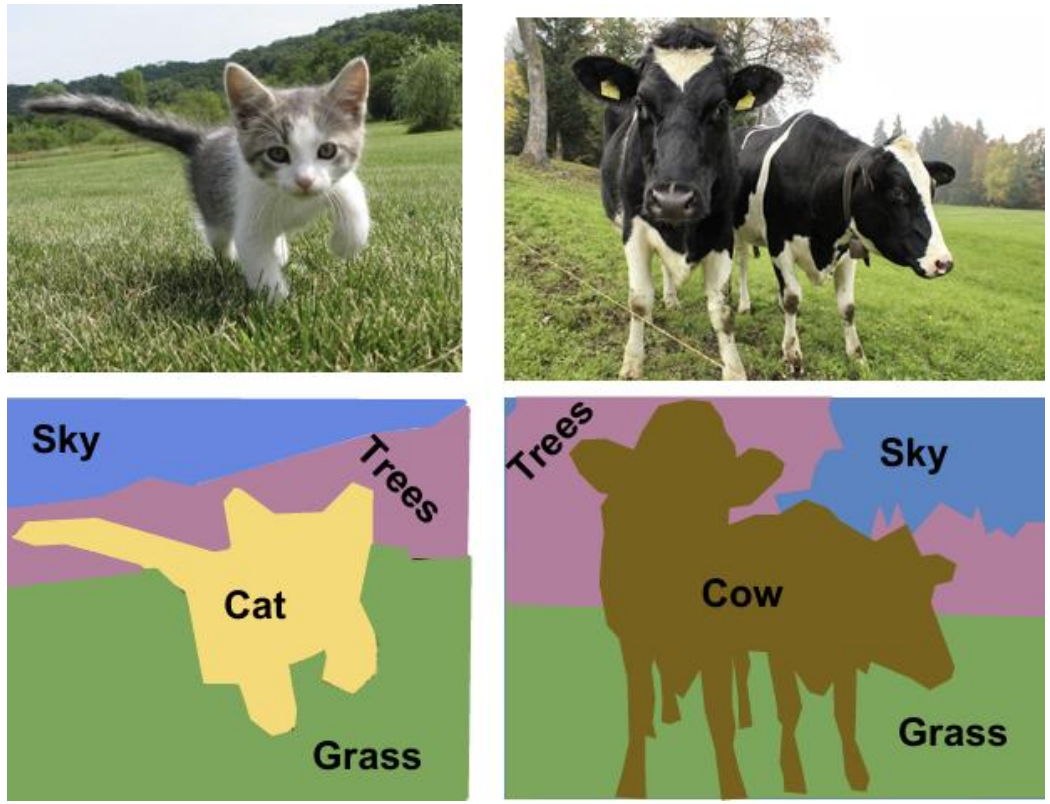
- **VFM**(Vision Foundation Model) + **VLM**(Vision Language Model)의 **장점**을 결합
- VFM과 VLM을 결합할 때 늘어나는 **patch sequence를 효율적으로 결합하는 Mamba 기반의 융합 프레임워크**



Background

Semantic Segmentation?

- Image의 모든 pixel을 특정 class로 분류해 pixel 단위로 이미지를 segmentation하는 task
- 자율주행, 의료 등 다양한 분야에서 사용



Background

DGSS(Domain-Generalized Semantic Segmentation)?

- Semantic segmentation은 train dataset에 없는 새로운 도메인과 조건에서 낮은 성능(저조도, 비)
- Train dataset에 없는 새로운 도메인과 조건에서도 강건하게 동작하는 Semantic Segmentation 모델 개발

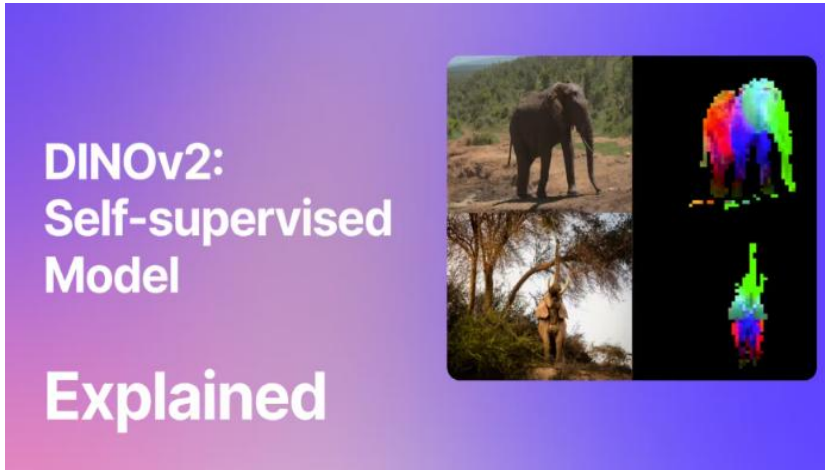
Unseen domain image

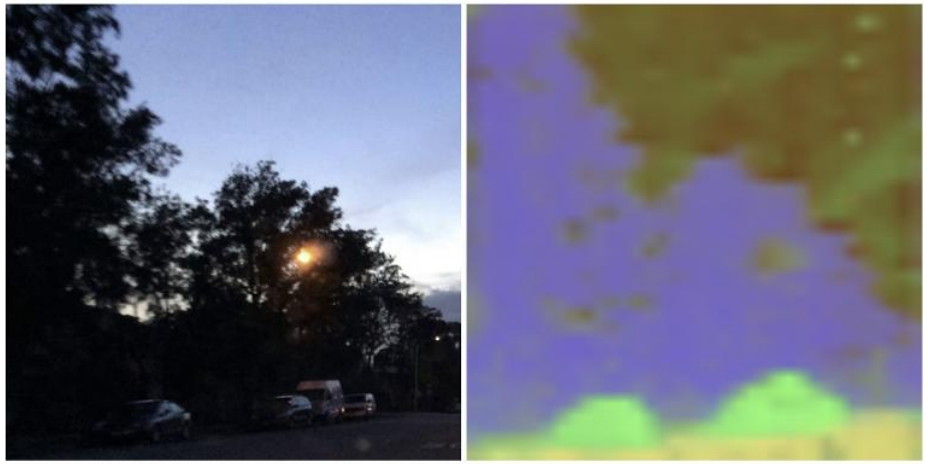
Output

Background

VFM

- 대규모 Image data로만 학습한 foundation model
- Image의 detail한 특징 파악
- Train dataset에 없는 (의료, 항공...) domain에서는 약한 성능



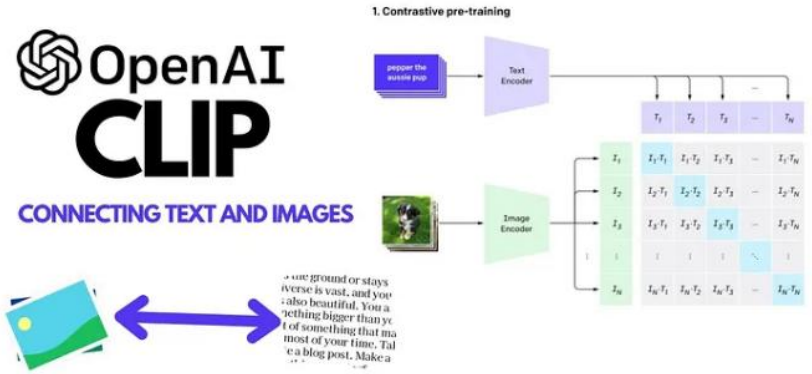


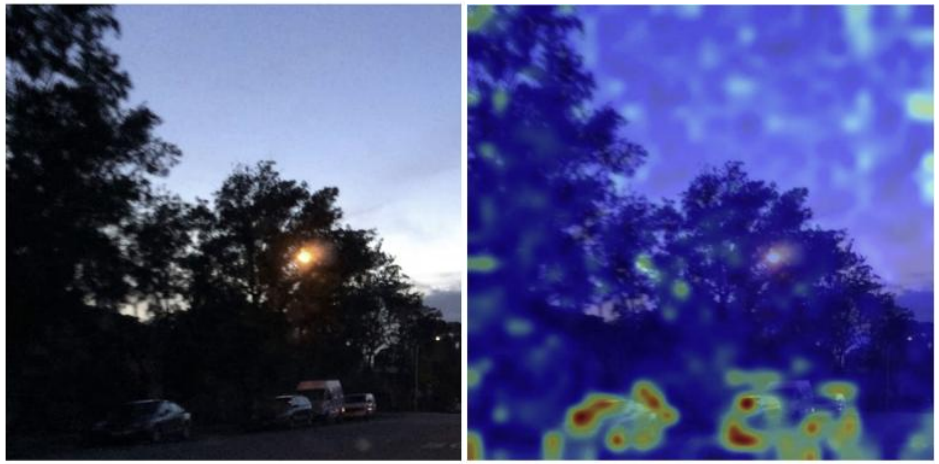
Unseen Image VFM

Background

VLM

- Image ,text data로 학습한 foundation model
- Text Embedding을 semantic anchor로 활용해 domain이 변화해도 강건한 성능 유지
- 정확한 영역 파악, Detail한 특징 파악에 어려움



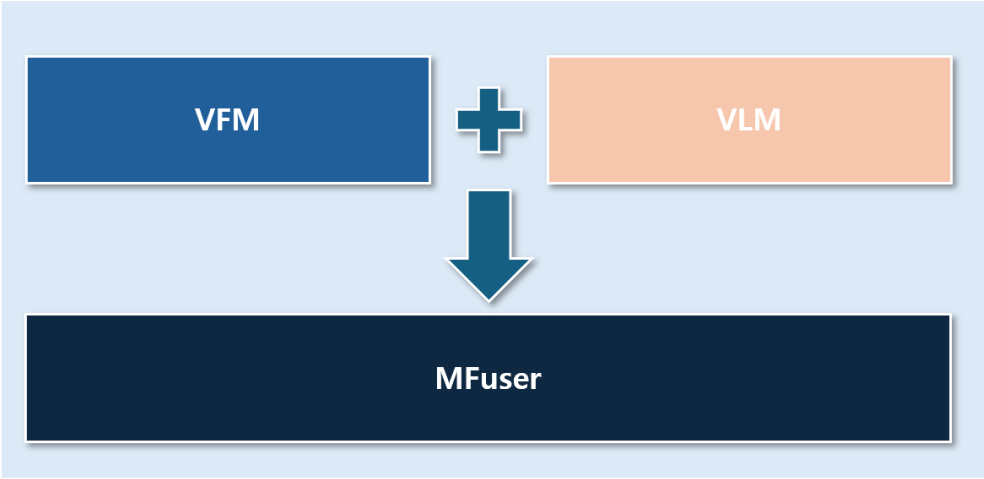


Unseen Image VLM

Background

MFuser

- VFM, VLM encoder에 많은 수의 parameters가 있어 fully fine-tuning은 비효율적임
- 원래 Encoder parameters는 고정, 추가로 trainable parameter 도입
- 하지만 VFM, VLM의 encoder를 결합하면 patch sequence가 두배가 되어 처리가 복잡해짐
- **이를 위해 VFM, VLM의 강점을 효율적으로 통합하는 Mamba 기반 MFuser 제안**

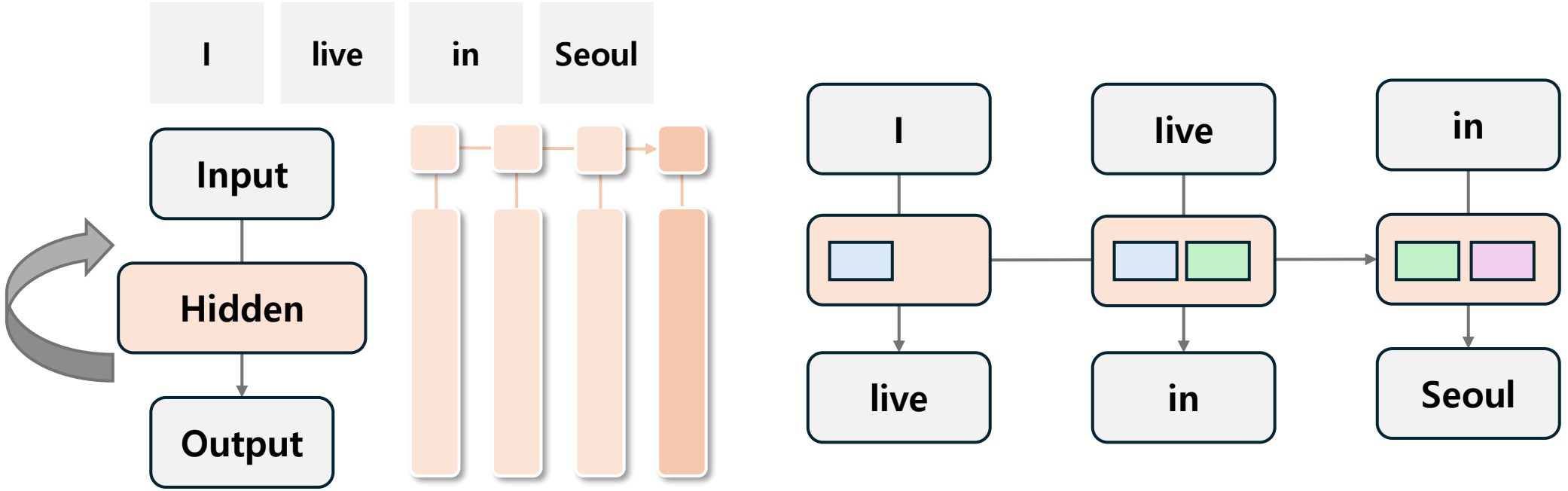


Unseen Image MFuser

Background

RNN

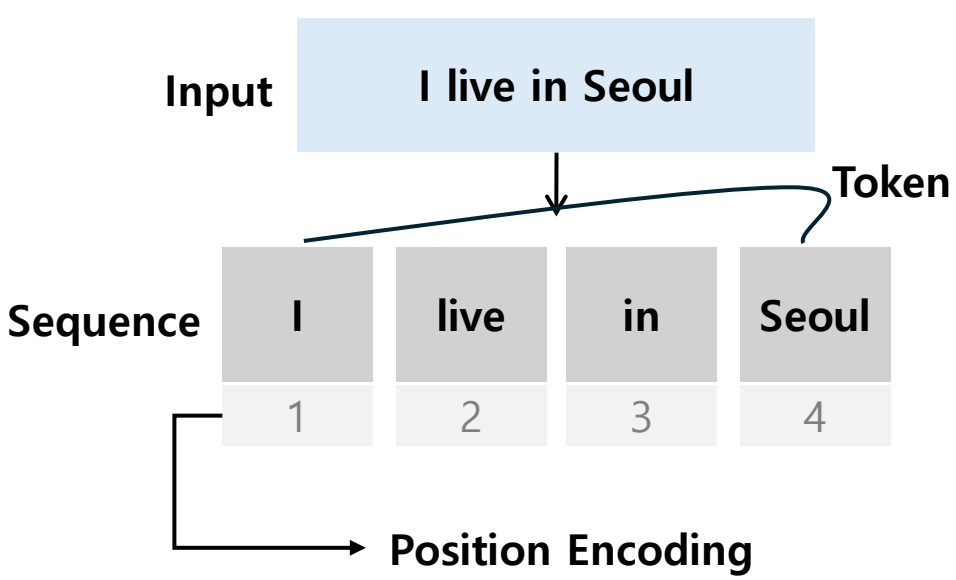
- Input을 순서대로 처리해 data에 순서를 학습 -> 문맥 파악에 효과적
- Inference시 이전 상태와 현재 상태만 고려해 출력 -> Fast inference
- Train 과정에서 이전 상태에 출력이 필요해 병렬화 불가능 -> Slow Training
- Input sequence 길이에 상관없이 고정된 크기의 hidden state 유지(압축) -> 입력 Sequence가 길어지면 장기 의존성 문제 발생

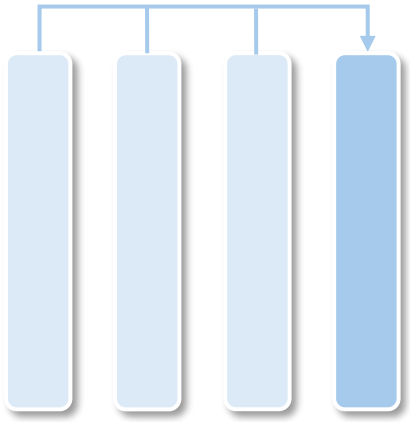


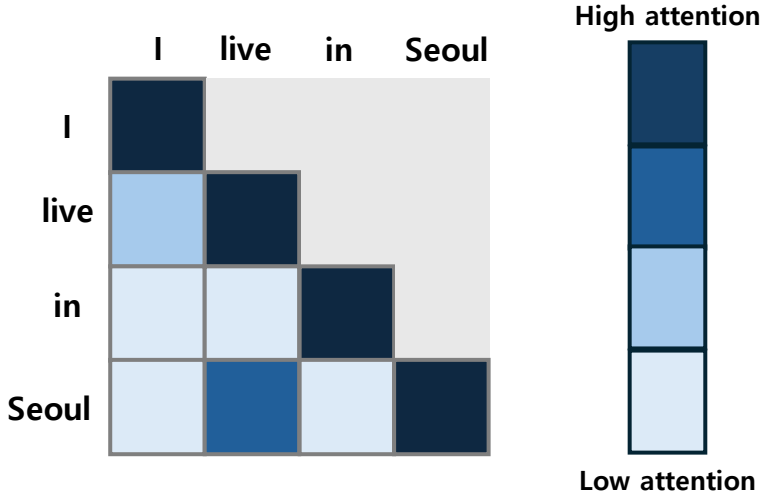
Background

Transformer

- 모든 Input을 **token**으로 구성된 **sequence**로 봄 -> sequence의 이전 token에 attend해 문맥 파악 가능
- **Position Encoding**을 통해 토큰들의 순서 정보 주입 -> sequence의 순서 정보를 통해 문장의 구조 & 문맥 이해
- 모든 **Token간에 관계**를 **병렬적**으로 계산 가능 -> Training Parallelism
- **But..** Inference시 다음 token을 생성하는데 이전 token을 순차적으로 attend -> Slow inference





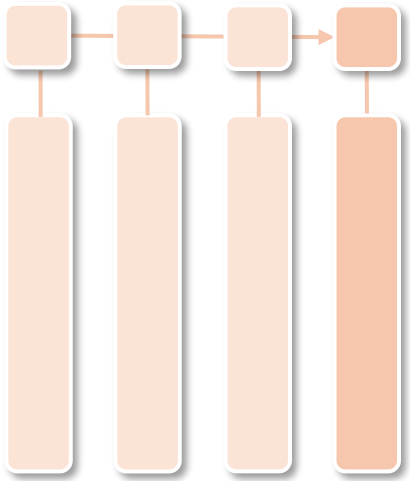


Background

Mamba

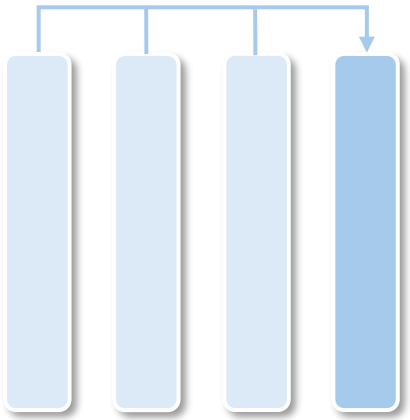
- **RNN ->** 순차적으로 정보를 처리하지만 Sequence가 길어지면 앞부분의 정보를 잊어버림 & 병렬화 문제로 느린 학습 속도
- **Transformer ->** 병렬 구조로 모든 단어의 관계를 한 번에 처리해 학습은 빠르지만, 추론 시 단어를 순차적으로 생성해야 해서 느림

모든 정보를 attend?
Memory에 context 저장
느린 학습 속도 문제 해결..
한정된 space에 잘 압축..



RNN

VS

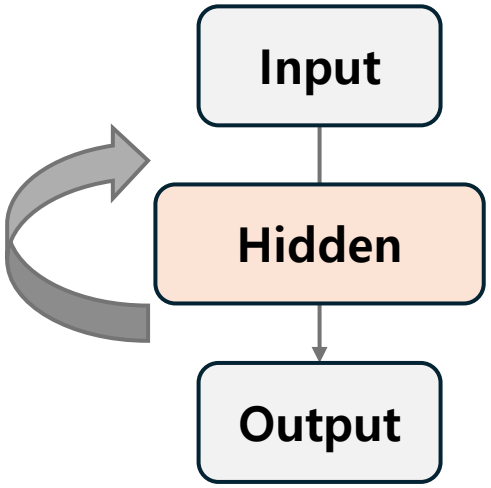


Transformer

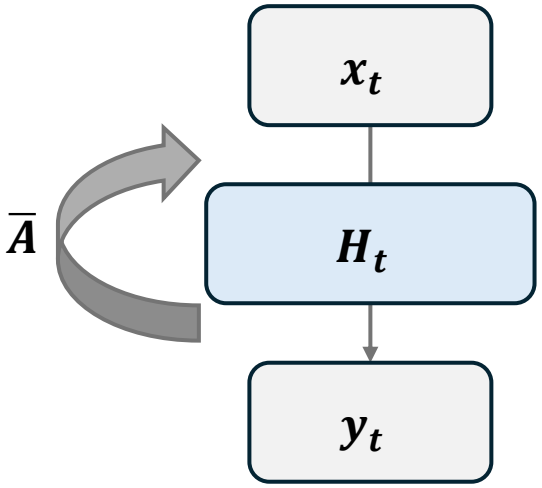
모든 정보를 왜 압축?
필요한 정보 Attend
High memory & cost..
느린 추론 속도..

Background

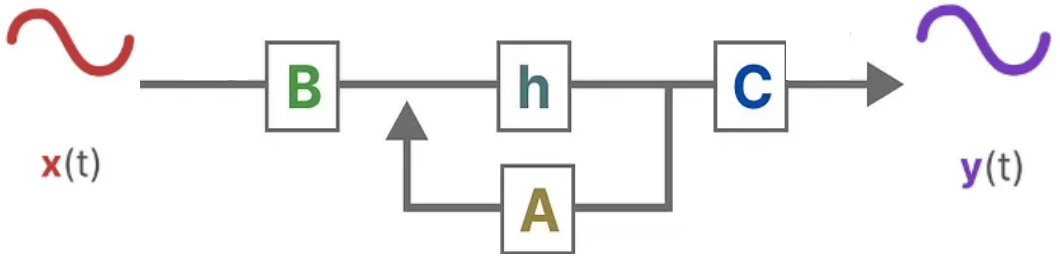
- **SSM(State Space Model)**
  - ✓ **Convolution**을 통해 Fast training
  - ✓ **Recurrent**를 통해 Fast inference



RNN



SSM



$$x_t = \bar{A}x_{t-1} + \bar{B}u_t$$

$$y_t = Cx_t$$

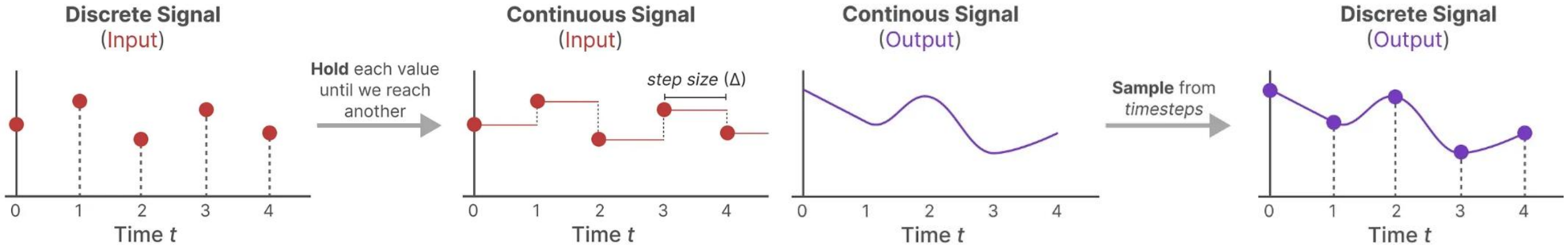
x_t: Hidden state vector, u_t: Input vector, y_t: Output vector

$\bar{A}$: Update memory matrix, $\bar{B}$: Input matrix, C: Output matrix

Background

SSM

- **Discretize**
  - ✓ SSM은 연속 신호에서 사용했던 모델임, 따라서 Discrete-to-Continuous 필요
  - ✓ 이를 위해 ZOH(Zero-Order Hold) 사용
  - ✓ 출력된 Continuous signal을 sampling을 통해 discretize

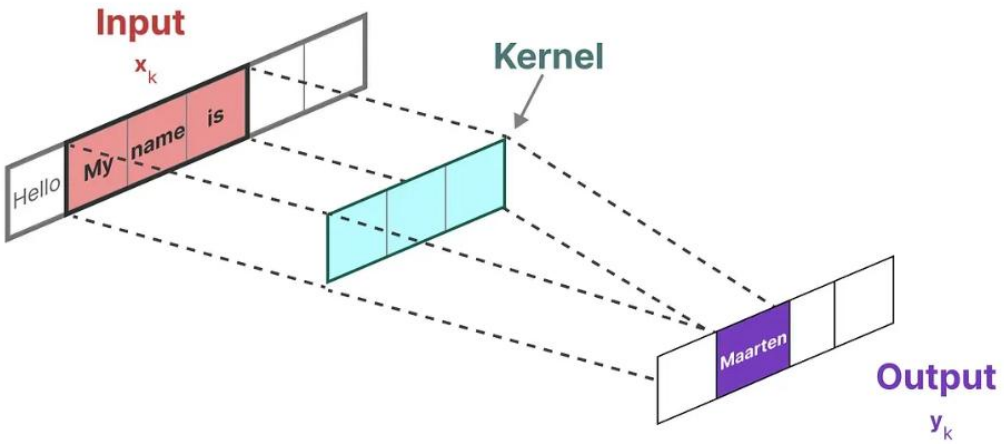


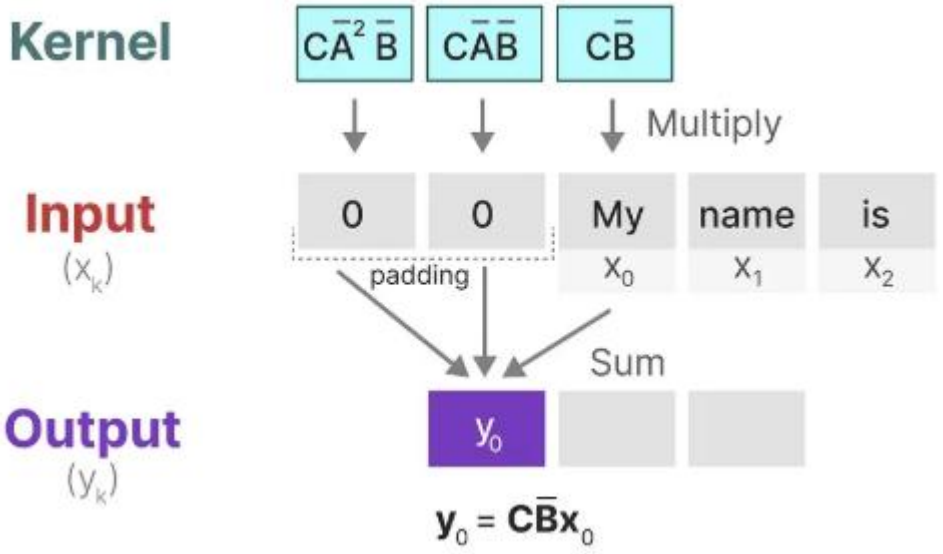
Background

SSM

- **Convolution**
  - ✓ **LTI(Linear Time-Invariant)system**

    시간이 흘러도 모델의 동작을 결정하는 행렬 $\bar{A}, \bar{B}, C$가 변하지 않아 Recurrent 구조를 Convolution 구조로 변환 가능

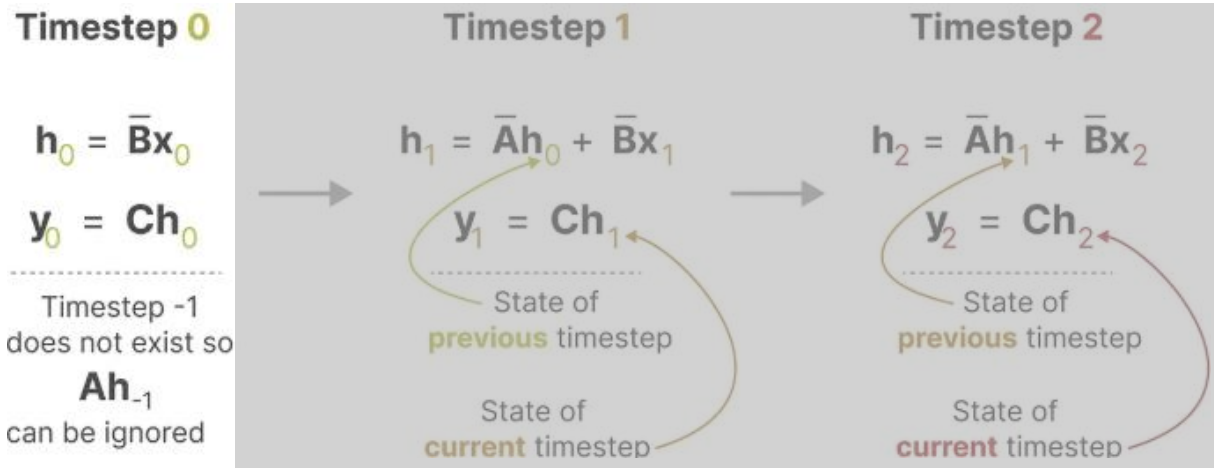


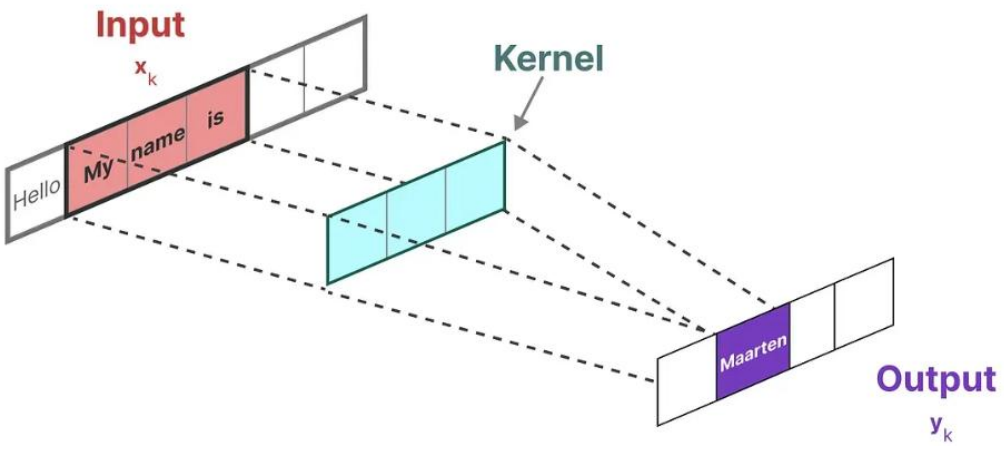


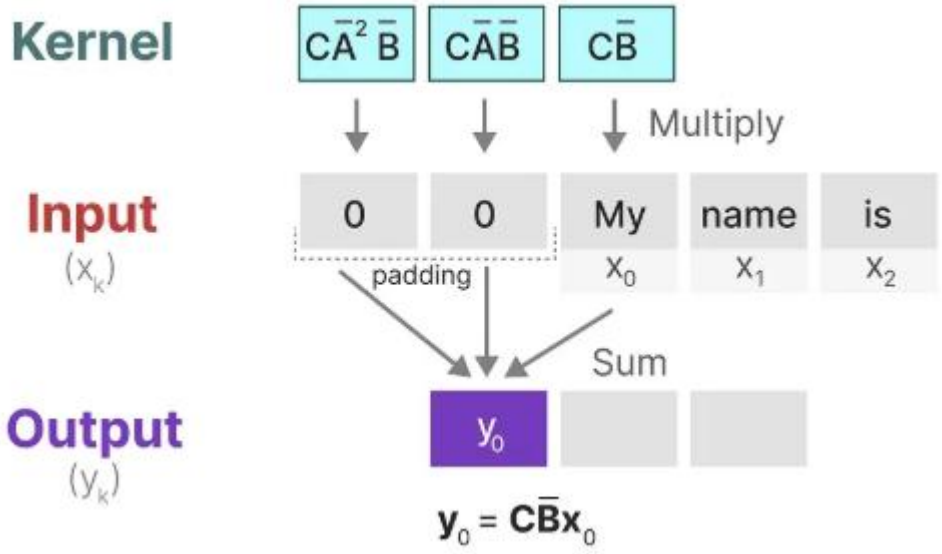
Background

SSM

- **Convolution**



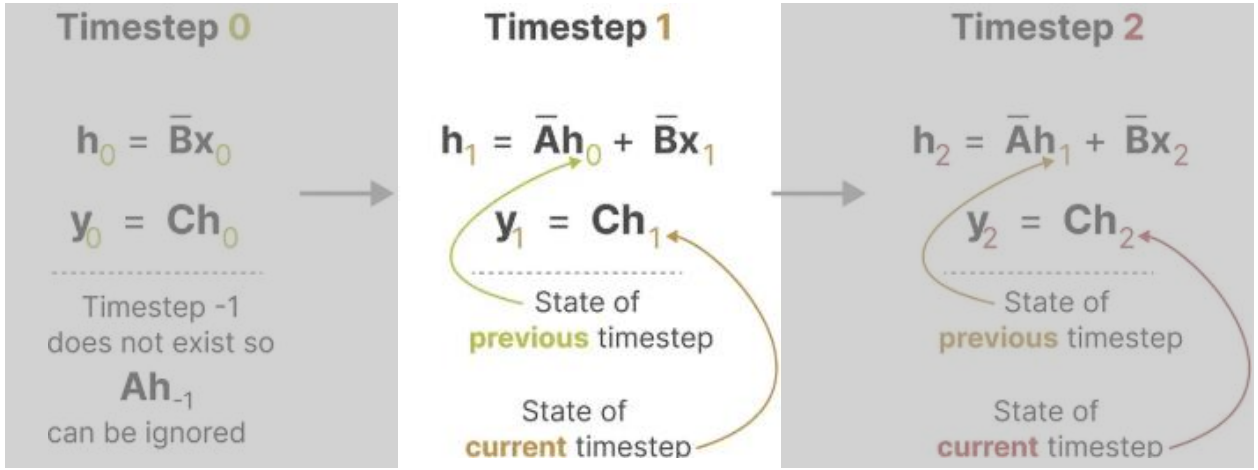


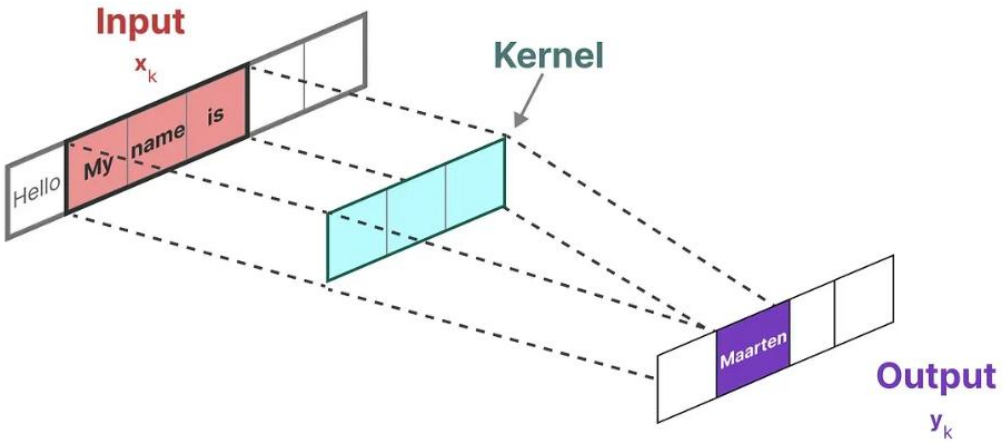


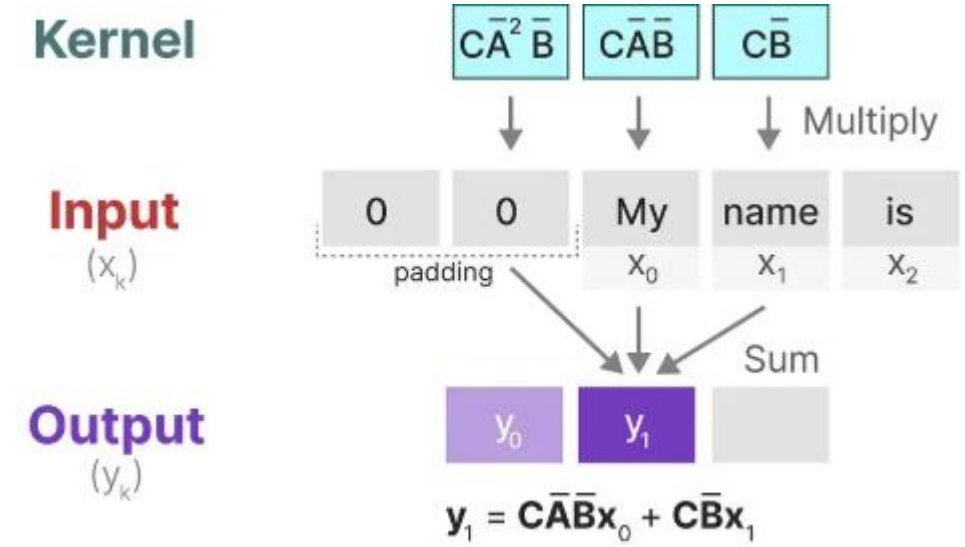
Background

SSM

- **Convolution**



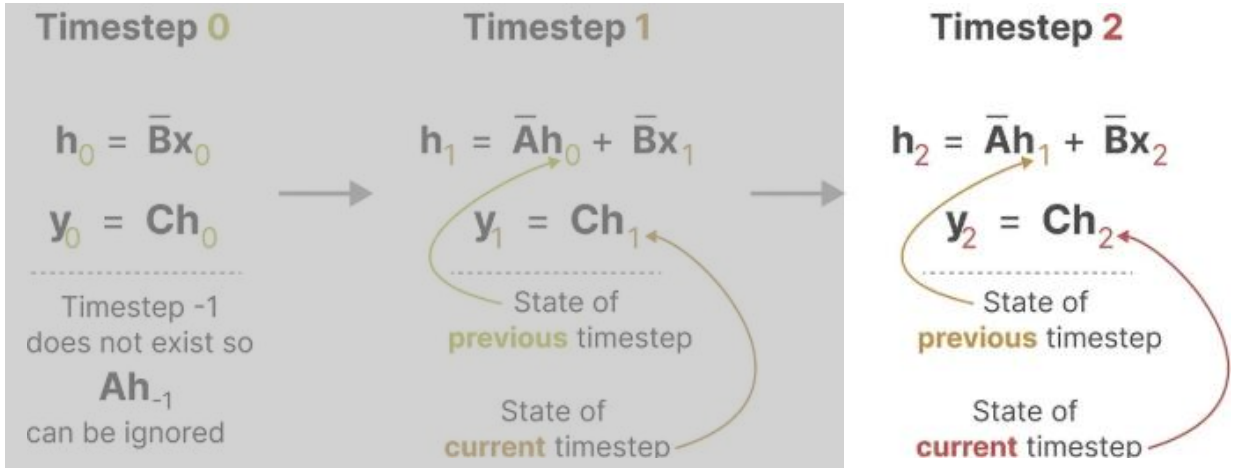


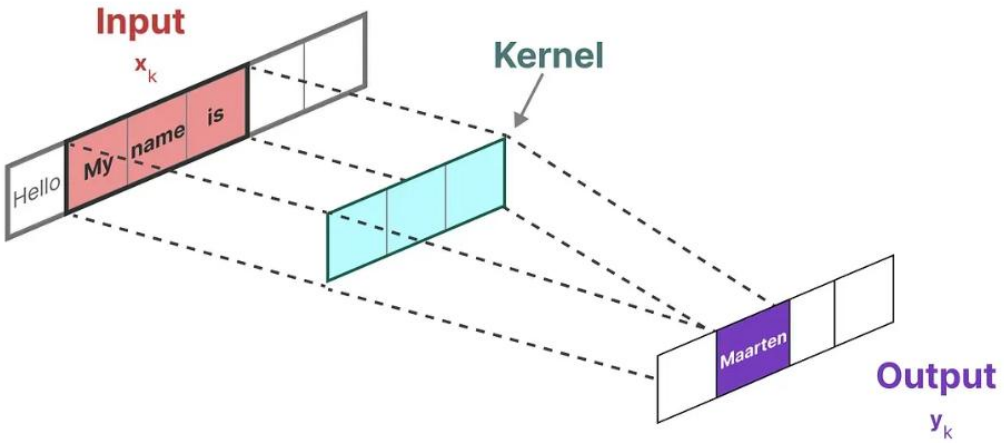


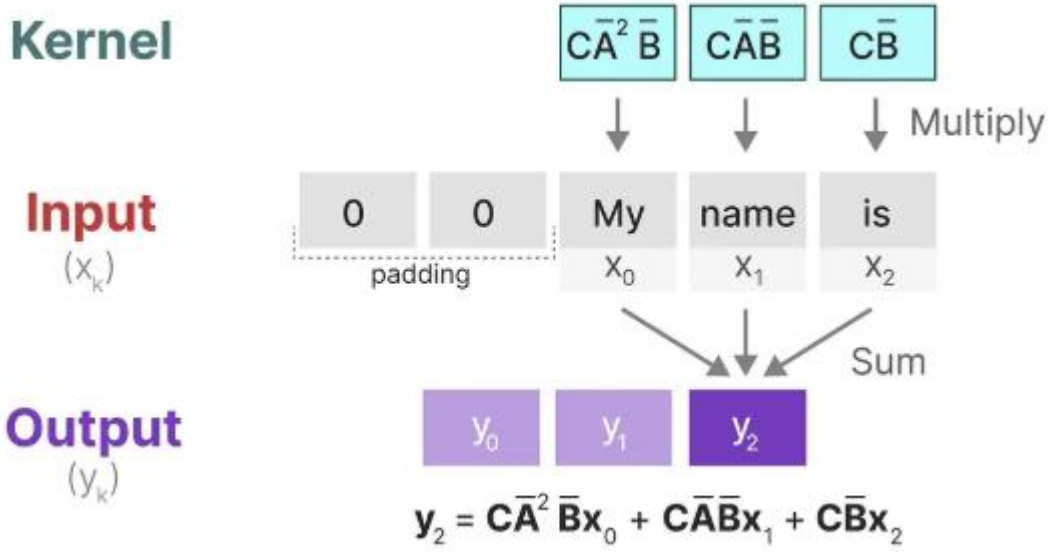
Background

SSM

- **Convolution**



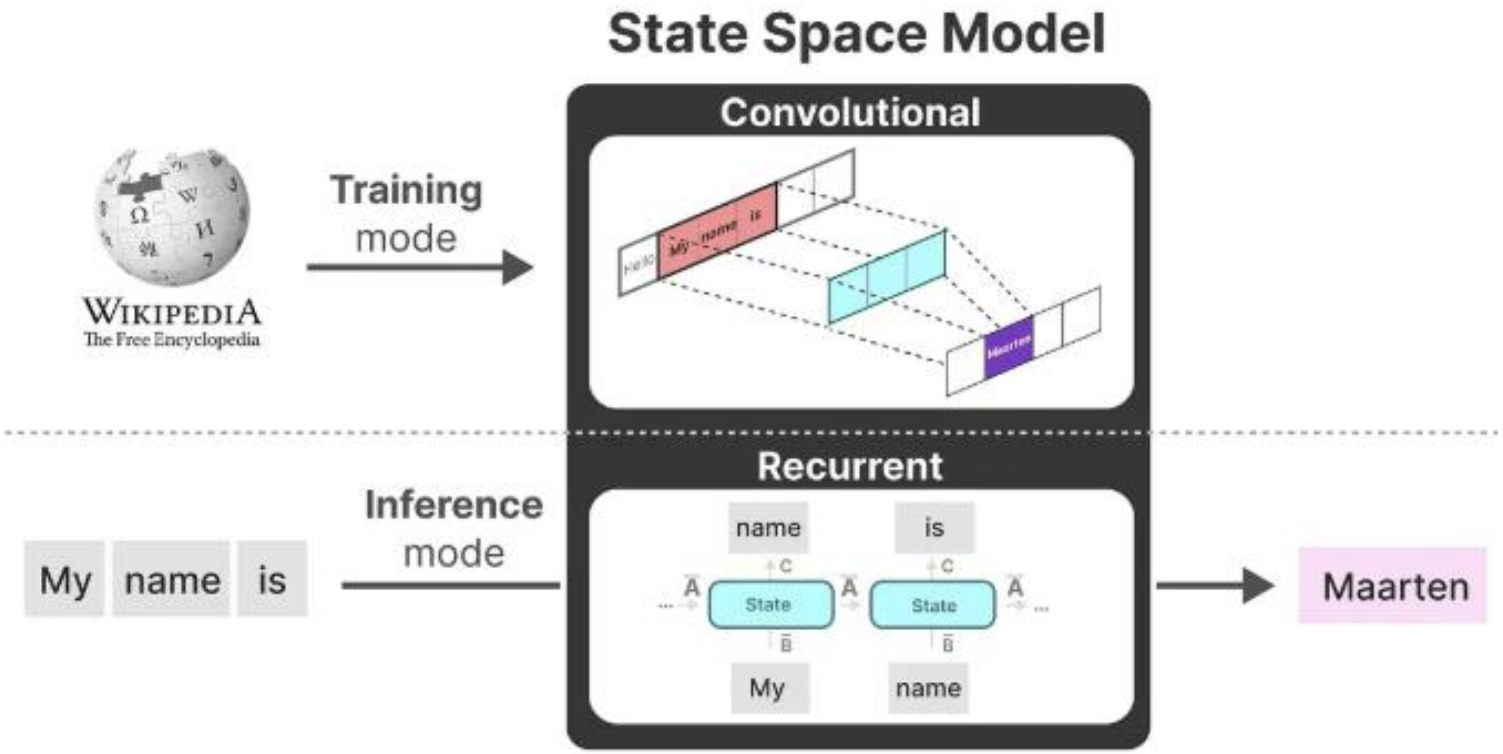




Background

SSM

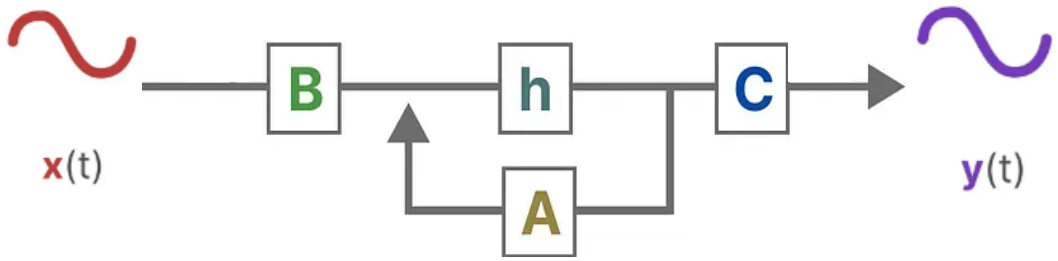
- Training mode -> Convolution 연산을 통해 parallelizable training이 가능해 빠른 학습 가능
- Inference mode -> Recurrent를 통해 효율적인 추론 가능



Background

Mamba?

- A matrix
  - ✓ Memory의 update를 담당
  - ✓ A matrix를 생성하는 것에 따라 이전 상태만 기억할지 모든 상태를 기억할지의 차이가 만들어짐



$$x_t = \bar{A}x_{t-1} + \bar{B}u_t$$

$$y_t = Cx_t$$

$\boldsymbol{x_t}$: Hidden state vector, $\boldsymbol{u_t}$: Input vector, $\boldsymbol{y_t}$: Output vector

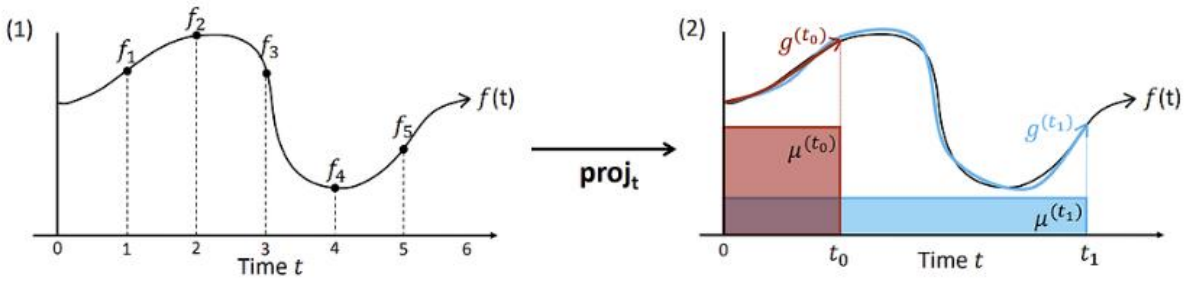
$\bar{\boldsymbol{A}}$: Update memory matrix, $\bar{\boldsymbol{B}}$: Input matrix, $\boldsymbol{C}$: Output matrix

Background

Mamba?

- **HiPPO**
  - ✓ 과거의 모든 정보를 압축하는 대신, 지금까지의 입력을 가장 잘 표현하는 하나의 연속 함수로 기억
  - ✓ 과거의 정보는 감소, 최신 정보는 민감하게 받아들임

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



HiPPO

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

HiPPO?

- A matrix 계산
- $n, k = 0,0$

$$\begin{bmatrix} 1 & N/A & N/A \\ N/A & N/A & N/A \\ N/A & N/A & N/A \end{bmatrix}$$

- $n, k = 0,1$

$$\begin{bmatrix} 1 & 0 & N/A \\ \sqrt{3} & N/A & N/A \\ N/A & N/A & N/A \end{bmatrix}$$

- $n, k = 1,2$

$$\begin{bmatrix} 1 & 0 & 0 \\ \sqrt{3} & 2 & 0 \\ N/A & N/A & N/A \end{bmatrix}$$

- $n, k = 1,0$

$$\begin{bmatrix} 1 & N/A & N/A \\ \sqrt{3} & N/A & N/A \\ N/A & N/A & N/A \end{bmatrix}$$

- $n, k = 0,2$

$$\begin{bmatrix} 1 & 0 & 0 \\ \sqrt{3} & N/A & N/A \\ N/A & N/A & N/A \end{bmatrix}$$

- $n, k = 2,1$

$$\begin{bmatrix} 1 & 0 & 0 \\ \sqrt{3} & 2 & 0 \\ \sqrt{5} & \sqrt{15} & N/A \end{bmatrix}$$

- $n, k = 2,0$

$$\begin{bmatrix} 1 & N/A & N/A \\ \sqrt{3} & N/A & N/A \\ \sqrt{5} & N/A & N/A \end{bmatrix}$$

- $n, k = 1,1$

$$\begin{bmatrix} 1 & 0 & 0 \\ \sqrt{3} & 2 & N/A \\ N/A & N/A & N/A \end{bmatrix}$$

- $n, k = 2,2$

$$\begin{bmatrix} 1 & 0 & 0 \\ \sqrt{3} & 2 & 0 \\ \sqrt{5} & \sqrt{15} & 3 \end{bmatrix}$$

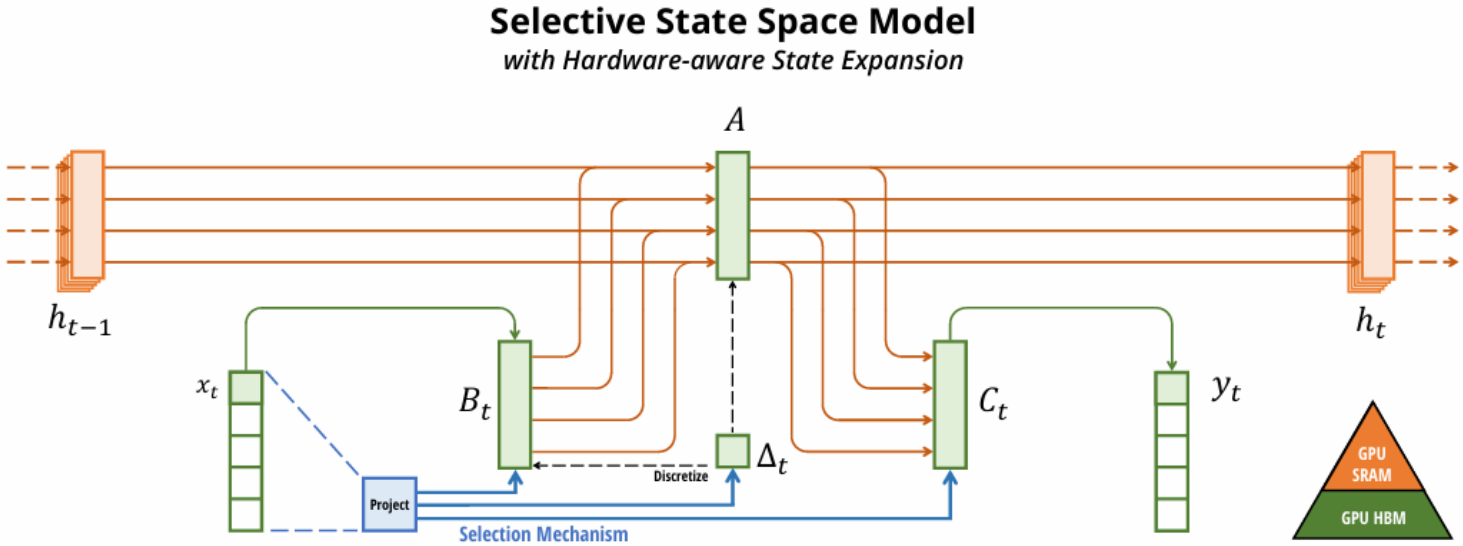
Background

Mamba

- SSM은 Audio, Vision 같이 Continuous한 분야에서는 좋은 성능을 보여줌
- 하지만 Text같이 Discrete한 분야에서는 약한 성능을 보임(특정 입력에 집중하거나 무시)

✓ **Selection Mechanism**

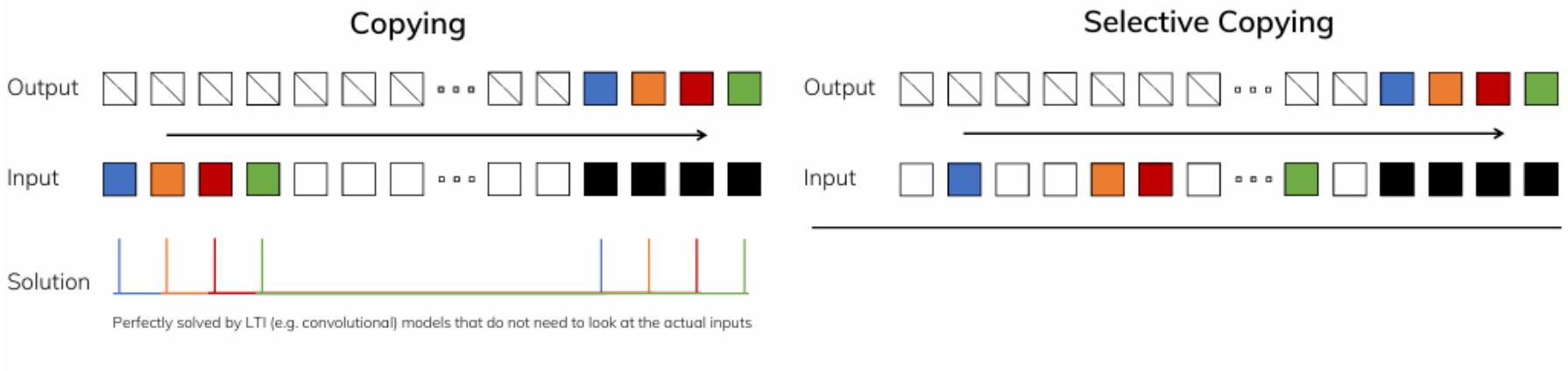
✓ **Hardware-aware Algorithm**



Background

Selection Mechanism

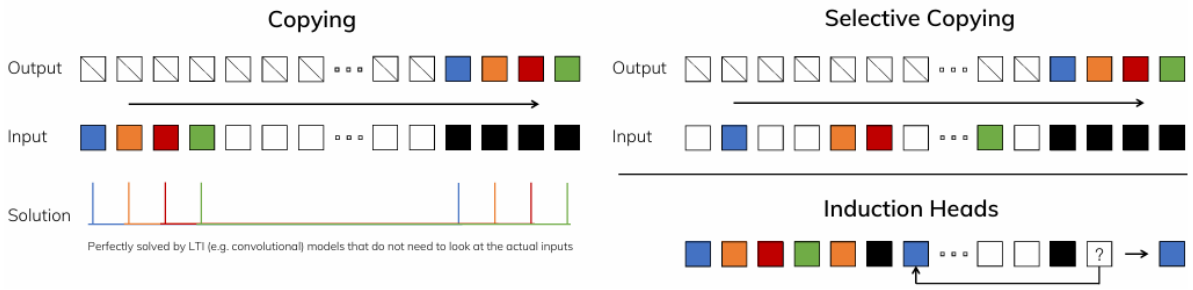
- **Selective Copying?**
  - ✓ Input의 일부를 copying -> 순서대로 출력하는 task (input의 중요도를 판단 후 기억할지, 무시할지 결정)
  - ✓ **특정 Token만** hidden state에 **선택적으로 update**할 수 있어야 함

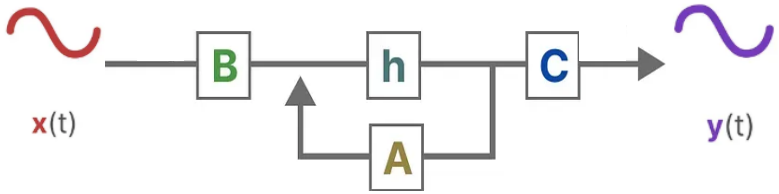


Background

Selection Mechanism

- **Selective Copying?**
  - ✓ **SSM은 Linear Time-Invariant** 때문에 이 task를 잘 수행하지 못함
  - ✓ 고정된 Convolution kernel을 사용해 어떤 input이 들어와도 동일한 연산을 통해 중요한 정보만 선별하는 능력이 떨어짐





$$x_t = \bar{A}x_{t-1} + \bar{B}u_t$$

$$y_t = Cx_t$$

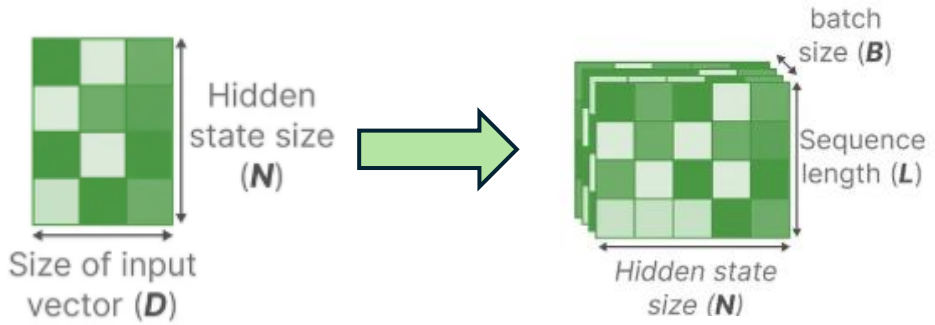
$\boldsymbol{x_t}$: Hidden state vector, $\boldsymbol{u_t}$: Input vector, $\boldsymbol{y_t}$: Output vector

$\bar{\boldsymbol{A}}$: Update memory matrix, $\bar{\boldsymbol{B}}$: Input matrix, $\boldsymbol{C}$: Output matrix

Background

Selection Mechanism

- Time-Invariant이 문제였다면 Time-variant로 Input에 따라 다른 연산이 가능하게 설계



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($\Delta, \boldsymbol{A}, \boldsymbol{B}$)
6: $y \leftarrow \mathrm{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

Algorithm 2 SSM + Selection (S6)

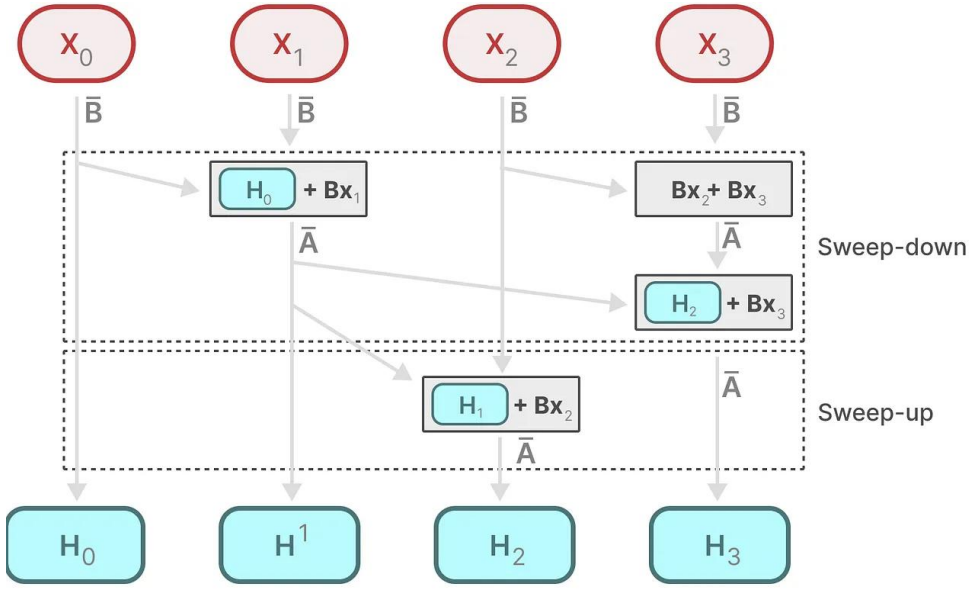
Input: x : (B, L, D)
Output: y : (B, L, D)
1: $\boldsymbol{A}$: (D, N) ← Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B}$: (B, L, N) ← $s_B(x)$
3: $\boldsymbol{C}$: (B, L, N) ← $s_C(x)$
4: Δ : (B, L, D) ← τ_Δ(Parameter+$s_\Delta(x)$)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$: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

$S_B(x)$=Linear$_N(x)$
$S_C(x)$=Linear$_N(x)$
$S_\Delta(x)$=$Brodcast_D$(Linear$_1(x)$)
τ_Δ=Softplus

Background

Parallel Scan

- Training Parallelism?
  - ✓ Selection Mechanism으로 인해 더 이상 Time-Invariant하지 않기 때문에 미리 Kernel을 만드는 것이 불가능 해짐
  - ✓ 따라서 Convolution을 통한 Training Parallelism은 불가
  - ✓ **이 문제를 해결하기 위해 Parallel Scan 사용**
  - ✓ **연산의 Associativity를 이용**



Background

Parallel Scan

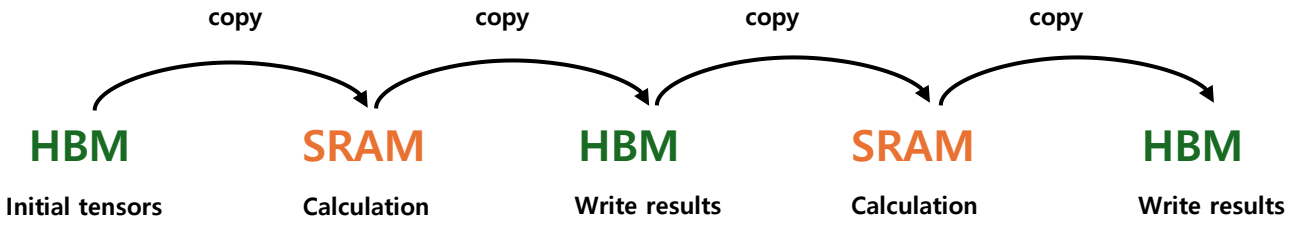
- Training Parallelism?
  - ✓ **순서와 상관없이 먼저 계산할 수 있는거 먼저 계산**

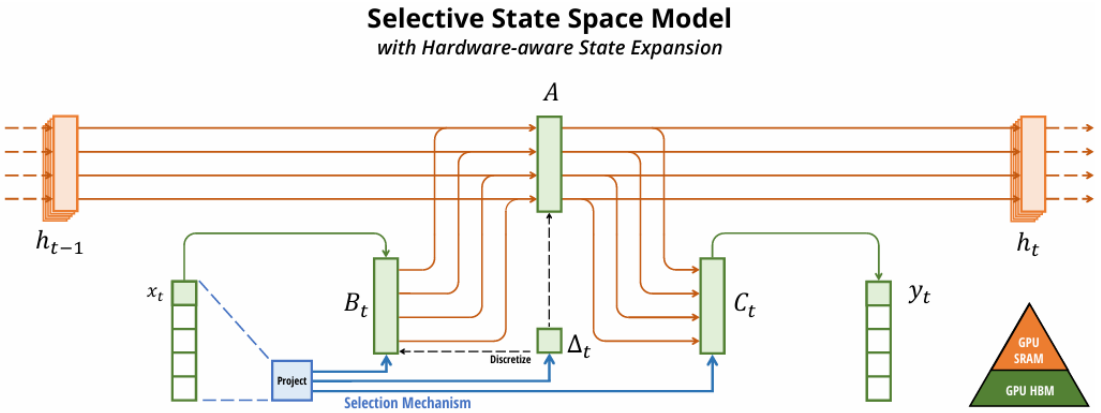
$$
\begin{aligned}
\mathbf{H}_0 &= \overline{\mathbf{B}}_0\mathbf{x}_0 \\
\mathbf{H}_1 &= \overline{\mathbf{A}}_1(\overline{\mathbf{B}}_0\mathbf{x}_0) + \overline{\mathbf{B}}_1\mathbf{x}_1 \\
&= \overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1 \\
\mathbf{H}_2 &= \overline{\mathbf{A}}_2(\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1) + \overline{\mathbf{B}}_2\mathbf{x}_2 \\
&= \overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2 \\
\mathbf{H}_3 &= \overline{\mathbf{A}}_3(\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2) + \overline{\mathbf{B}}_3\mathbf{x}_3 \\
&= \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{A}}_3\overline{\mathbf{B}}_2\mathbf{x}_2 + \overline{\mathbf{B}}_3\mathbf{x}_3
\end{aligned}
$$

Background

Hardware-aware Algorithm

- ✓ GPU에서 연산은 용량이 크지만 속도가 느린 HBM에서 용량이 작지만 속도는 빠른 SRAM으로 tensor를 Copy
- ✓ SRAM에서 연산이 진행되고 다시 HBM으로 paste
- ✓ 계산 복잡도가 높은 연산을 제외한 연산은 SRAM에서의 연산보다 Copy & Paste 과정이 더 오래 걸림



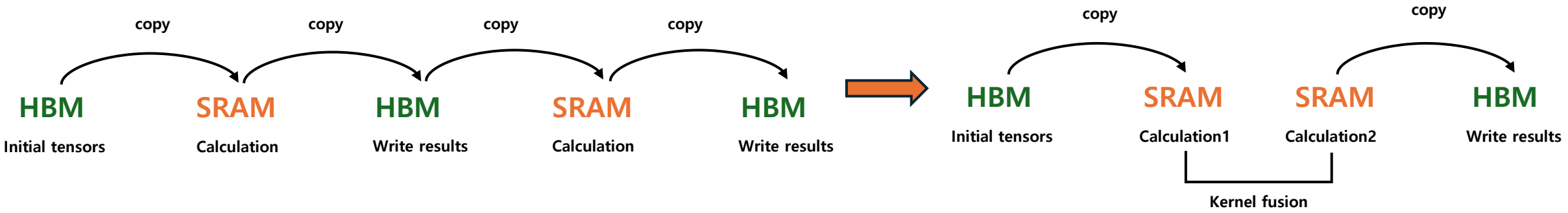


Background

Hardware-aware Algorithm

- **Kernel fusion**

✓ 여러 개의 연산을 하나로 합쳐줌



Background

Hardware-aware Algorithm

- **Kernel fusion**

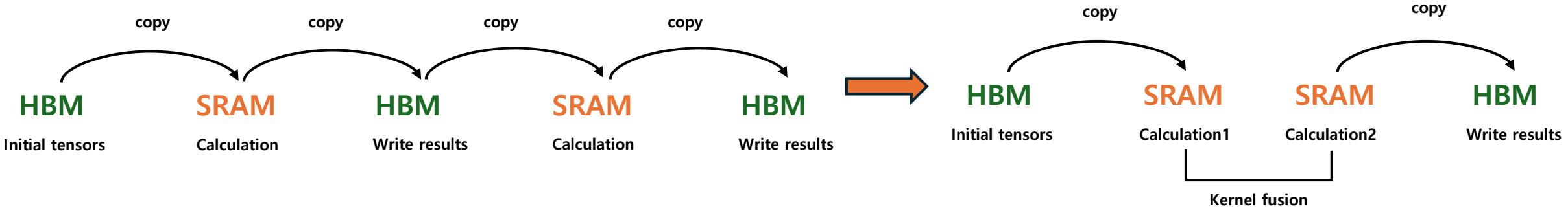
✓ A, B, C, Δ를 HBM에서 SRAM으로 Copy

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: $\boldsymbol{A}, \boldsymbol{B}$: (B, L, D, N) ← discretize($\Delta, \boldsymbol{A}, \boldsymbol{B}$)
6: $y \leftarrow \mathrm{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

$S_B(x)$=Linear$_N(x)$
$S_C(x)$=Linear$_N(x)$
$S_\Delta(x)$=$Brodcast_D$(Linear$_1(x)$)
τ_Δ=Softplus



Background

Hardware-aware Algorithm

- **Kernel fusion**

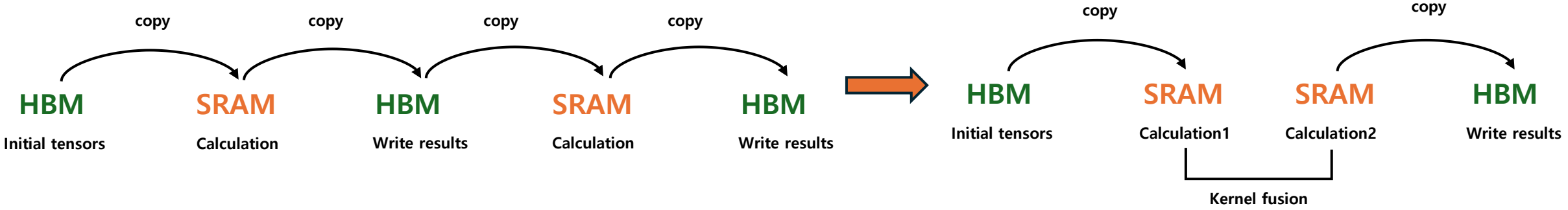
✓ A, B, C, Δ를 HBM에서 SRAM으로 Copy

✓ 3차원에서 4차원으로 늘어나는 Discretize를 SRAM에서 진행

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: $\boldsymbol{A}, \boldsymbol{B}$: (B, L, D, N) ← discretize($\Delta, \boldsymbol{A}, \boldsymbol{B}$)
6: y ← SSM($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$)($x$)
▷ Time-varying: recurrence (*scan*) only
7: **return** y

$S_B(x)$=Linear$_N(x)$
$S_C(x)$=Linear$_N(x)$
$S_\Delta(x)$=$Brodcast_D$(Linear$_1(x)$)
τ_Δ=Softplus



Background

Result

- **Kernel fusion**

✓ A, B, C, Δ를 HBM에서 SRAM으로 Copy

✓ 3차원에서 4차원으로 늘어나는 Discretize를 SRAM에서 진행

✓ 연산이 끝난 output을 SRAM에서 HBM으로 Copy

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($\Delta, \boldsymbol{A}, \boldsymbol{B}$)
6: $y \leftarrow \mathrm{SSM}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

$S_B(x)$=Linear$_N(x)$
$S_C(x)$=Linear$_N(x)$
$S_\Delta(x)$=$Brodcast_D$(Linear$_1(x)$)
τ_Δ=Softplus

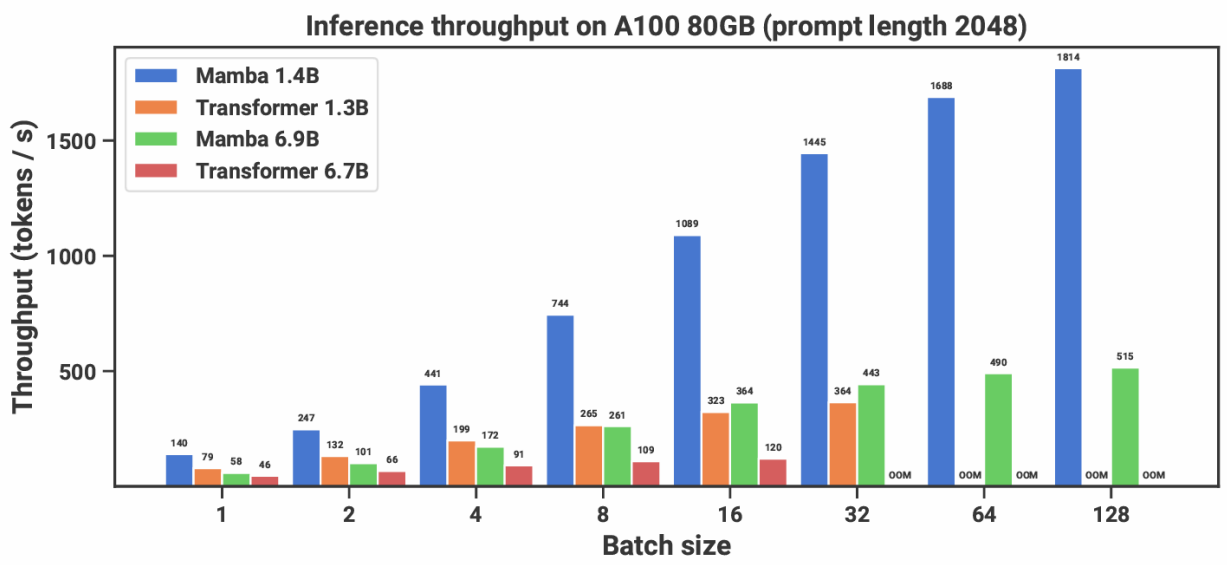
Background

Result

- ✓ Selection Mechanism으로 인해 Selective Copying task에서 정확도 향상
- ✓ Inference시 Transformer보다 월등한 속도를 보임

Model	Arch.	Layer	Acc.
S4	No gate	S4	18.3
-	No gate	S6	**97.0**
-	Mamba	S4	56.4
-	Mamba	Hyena	28.4
Mamba	Mamba	S6	**99.8**

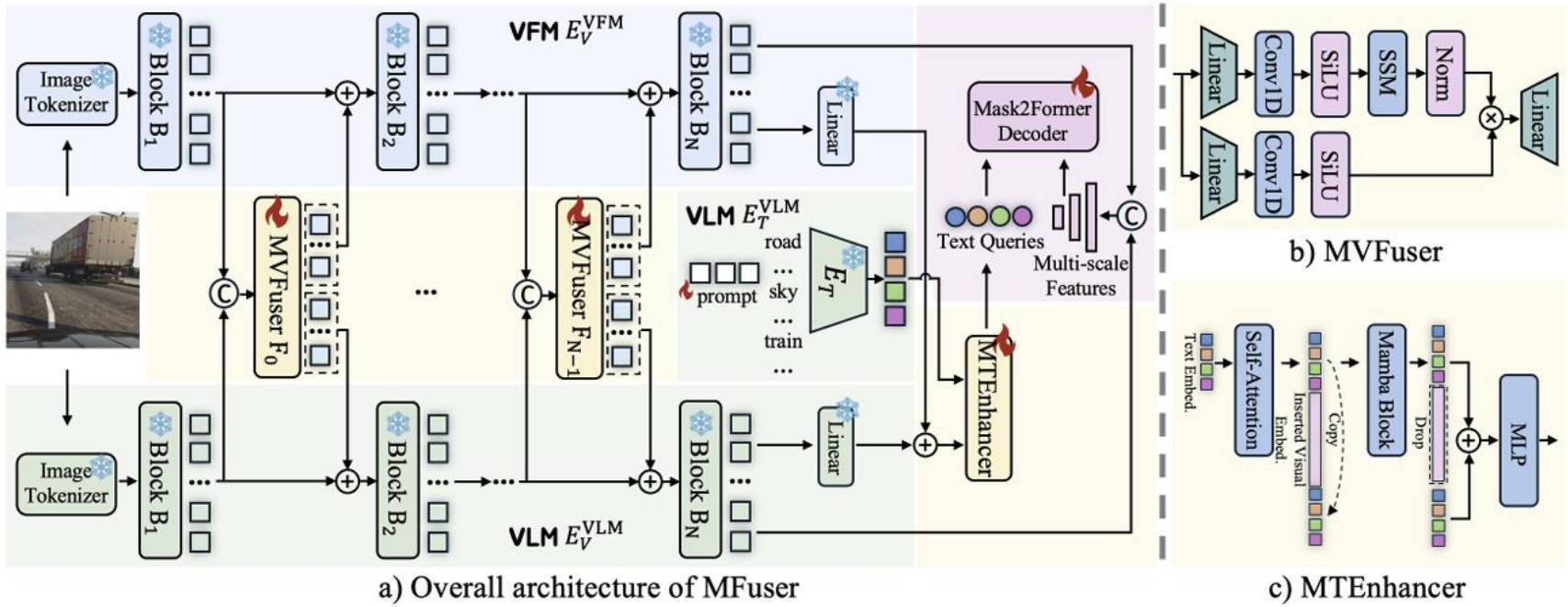
Table 1: (**Selective Copying**.) Accuracy for combinations of architectures and inner sequence layers.



MFuser

Method

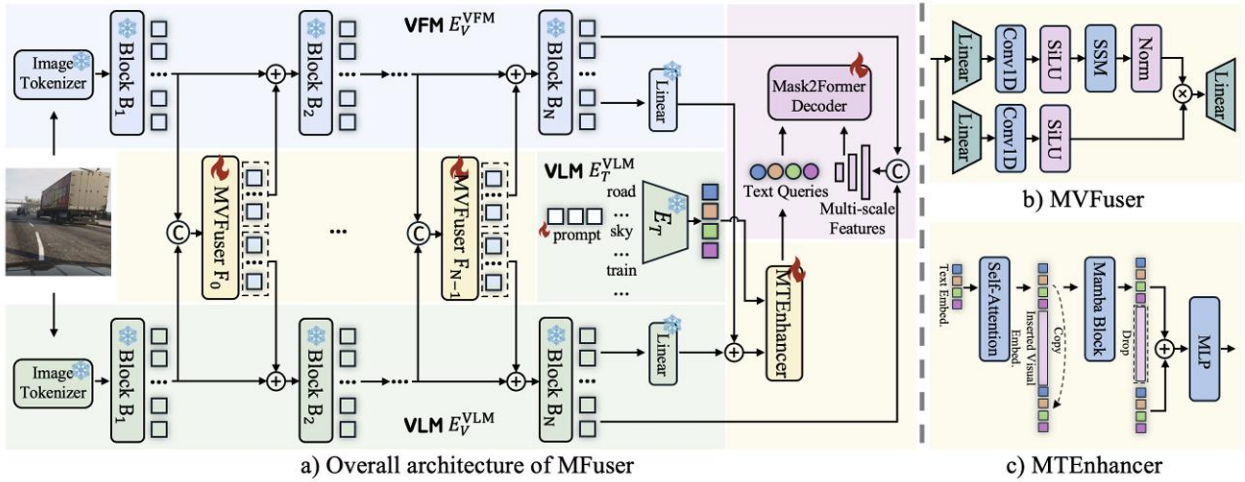
- MVFuser
- MTEnhancer

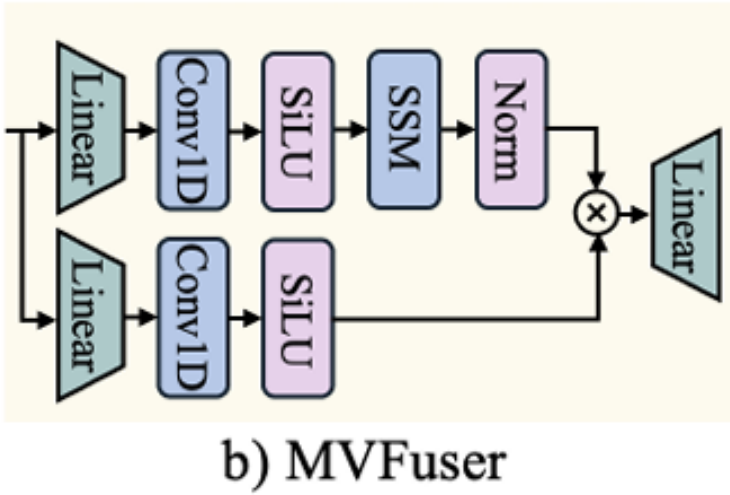


MFuser

MVFuser

- VFM, VLM encoder의 효율적인 Fine-tuning
- VFM의 detail, VLM의 robustness 결합
- Long sequence 문제 해결 -> Mamba 구조 사용

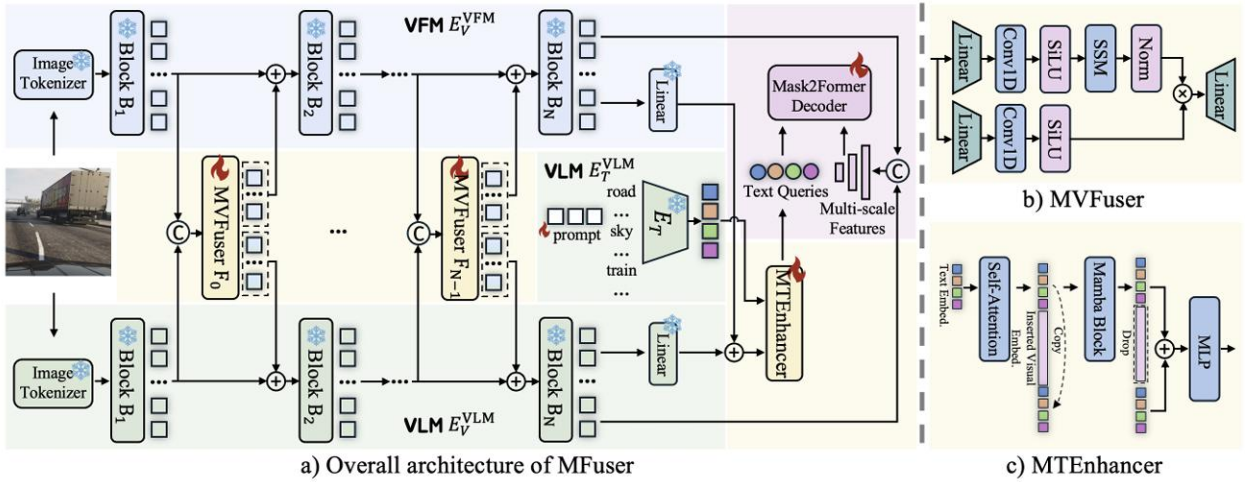


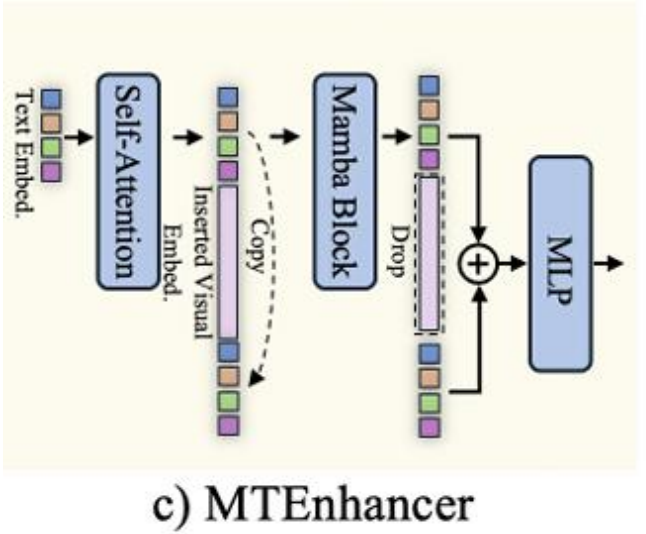


MFuser

MTEnhancer

- 기존 VLM 기반 방법론들은 시각적 특징과 Text Embedding이 미리 잘 정렬되어 있다고 가정
- 실제로는 클래스 이름만으로 얻은 Text Embedding이 다양한 이미지 유형에 충분히 적응하지 못하는 경우 발생
- MVFuser에서 융합된 시각적 특징을 Text Embedding에 통합하여 Text Embedding을 강화

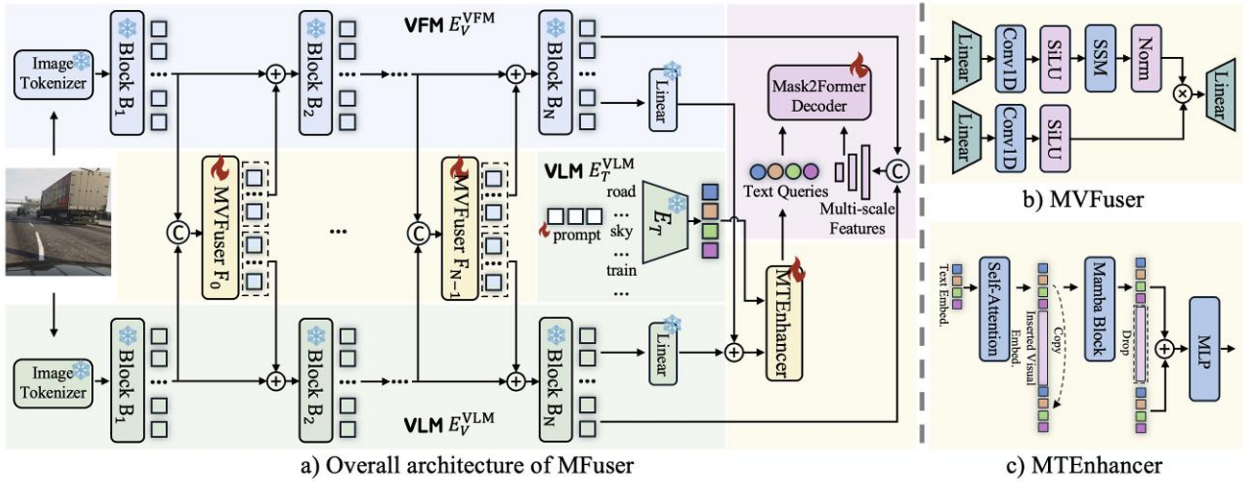


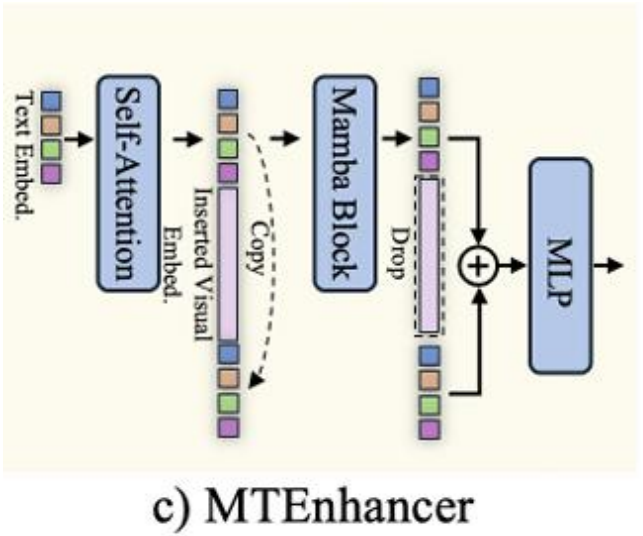


MFuser

MTEnhancer

✓ 먼저 Text embedding을 입력으로 받아 class간의 관계를 Encoding(Self-Attention)

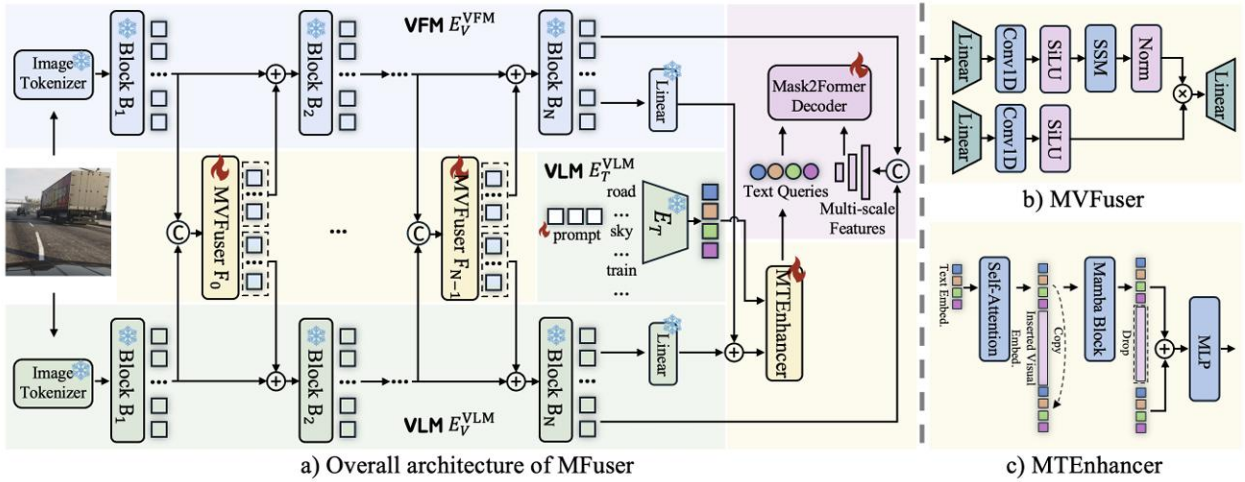


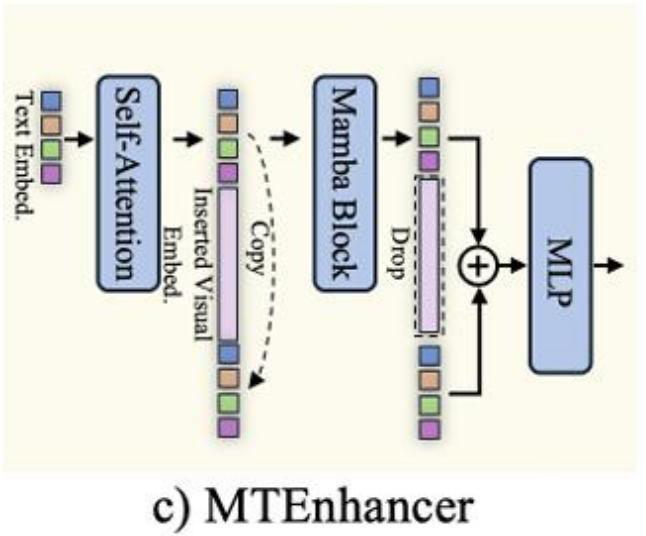


MFuser

MTEnhancer

- ✓ 먼저 Text embedding을 입력으로 받아 class간의 관계를 Encoding(Self-Attention)
- ✓ 시각적 특징 융합

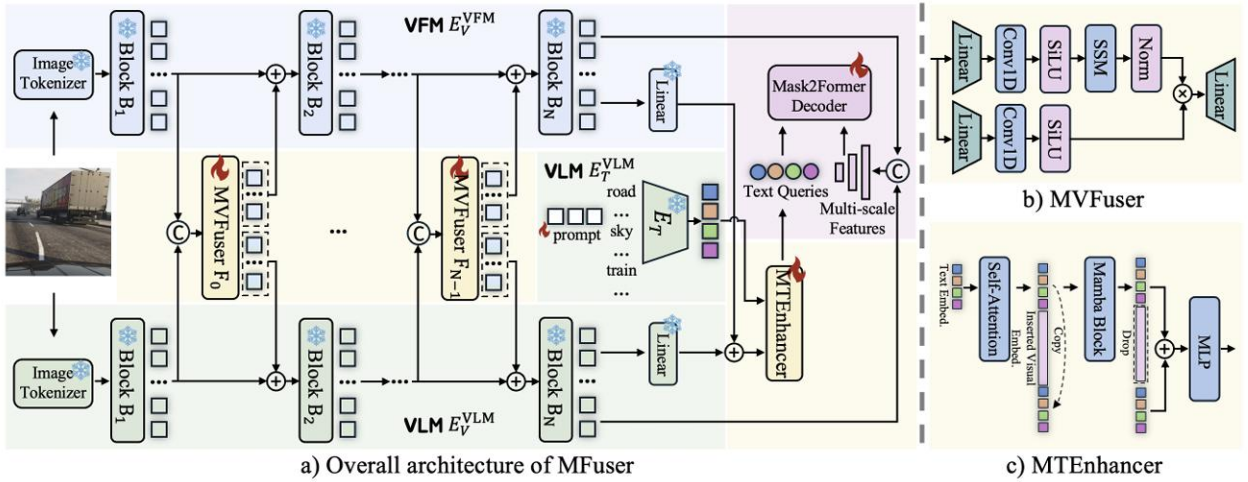


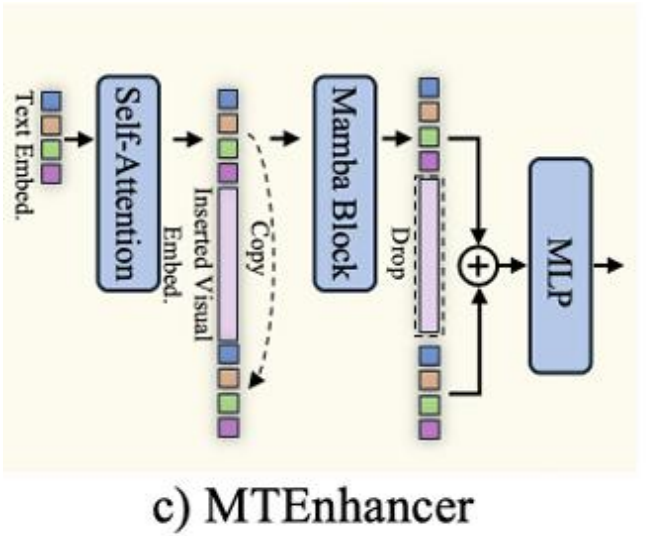


MFuser

MTEnhancer

- ✓ 먼저 Text embedding을 입력으로 받아 class간의 관계를 Encoding(Self-Attention)
- ✓ 시각적 특징 융합
- ✓ Text embedding update





MFuser

Result

Qualitative results

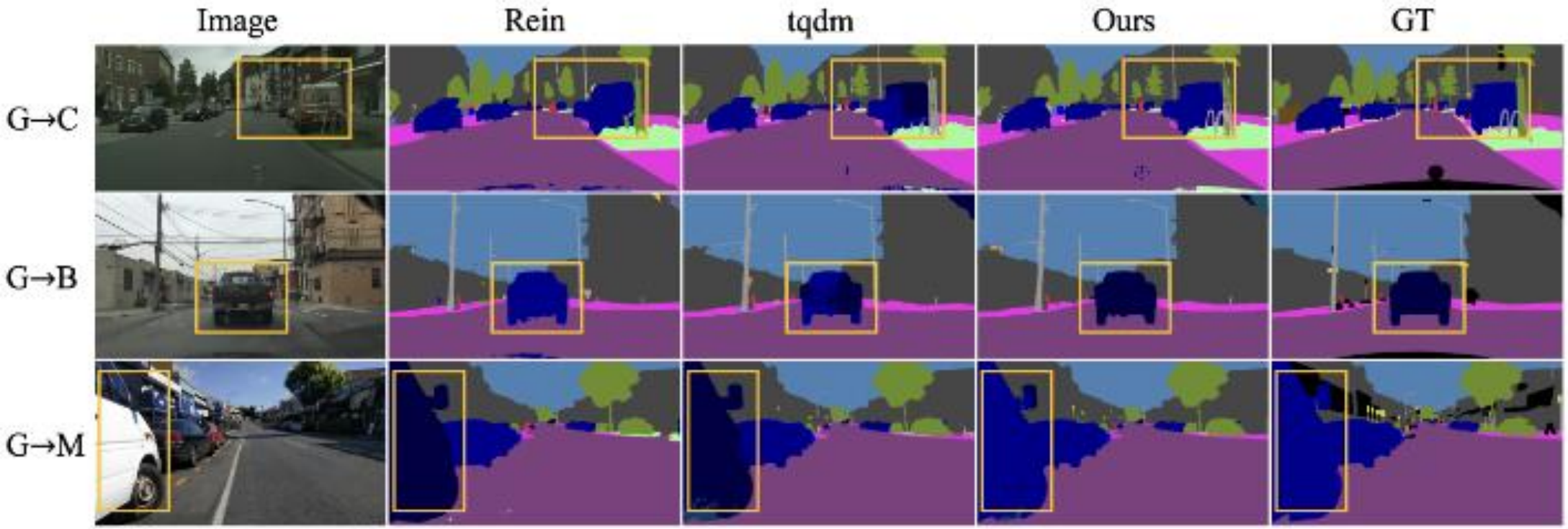
- synthetic-to-real에서 비교한 결과 SOTA 달성
- G(GTAV): GTA V dataset, C(Cityscapes): 거리 풍경, B(BDD100K): 다양한 지리적, 환경적 조건을 포함한 주행 dataset
  M(Mapillary): 다양한 장소 dataset

Method	Backbone	synthetic-to-real			
		G→C	G→B	G→M	Avg.
SAN-SAW [43]	RN101	45.33	41.18	40.77	42.43
WildNet [29]	RN101	45.79	41.73	47.08	44.87
SHADE [66]	RN101	46.66	43.66	45.50	45.27
TLDR [27]	RN101	47.58	44.88	48.80	47.09
FAMix [14]	RN101	49.47	46.40	51.97	49.28
SHADE [67]	MiT-B5	53.27	48.19	54.99	52.15
IBAFormer [53]	MiT-B5	56.34	49.76	58.26	54.79
VLTSeg [25]	CLIP-B	47.50	45.70	54.30	49.17
CLOUDS [2]	ConvNeXt-L	60.20	57.40	67.00	61.50
VLTSeg [25]	EVA02-L	65.60	58.40	66.50	63.50
Rein [55]	EVA02-L	65.30	60.50	64.90	63.60
Rein [55]	DINOv2-L	66.40	60.40	66.10	64.30
SET [63]	DINOv2-L	68.06	61.64	67.68	65.79
tqdm [40]	EVA02-L	68.88	59.18	70.10	66.05
MFuser	CLIP-L	**71.24**	61.08	71.14	67.82
MFuser	SIGLIP-L	71.10	61.19	**71.71**	68.00
MFuser	EVA02-L	70.19	**63.13**	71.28	**68.20**

	Params. (M)	FLOPs (G)	C	B	M	Avg.
self-attn (concat.)	4.20	98.64	70.24	62.31	71.11	67.89
self-attn (separate)	8.40	71.08	69.68	61.91	70.85	67.48
bi-deform-attn	3.35	34.65	69.46	61.17	70.11	66.91
MVFuser	1.67	17.21	**70.19**	**63.13**	**71.28**	**68.20**

MFuser

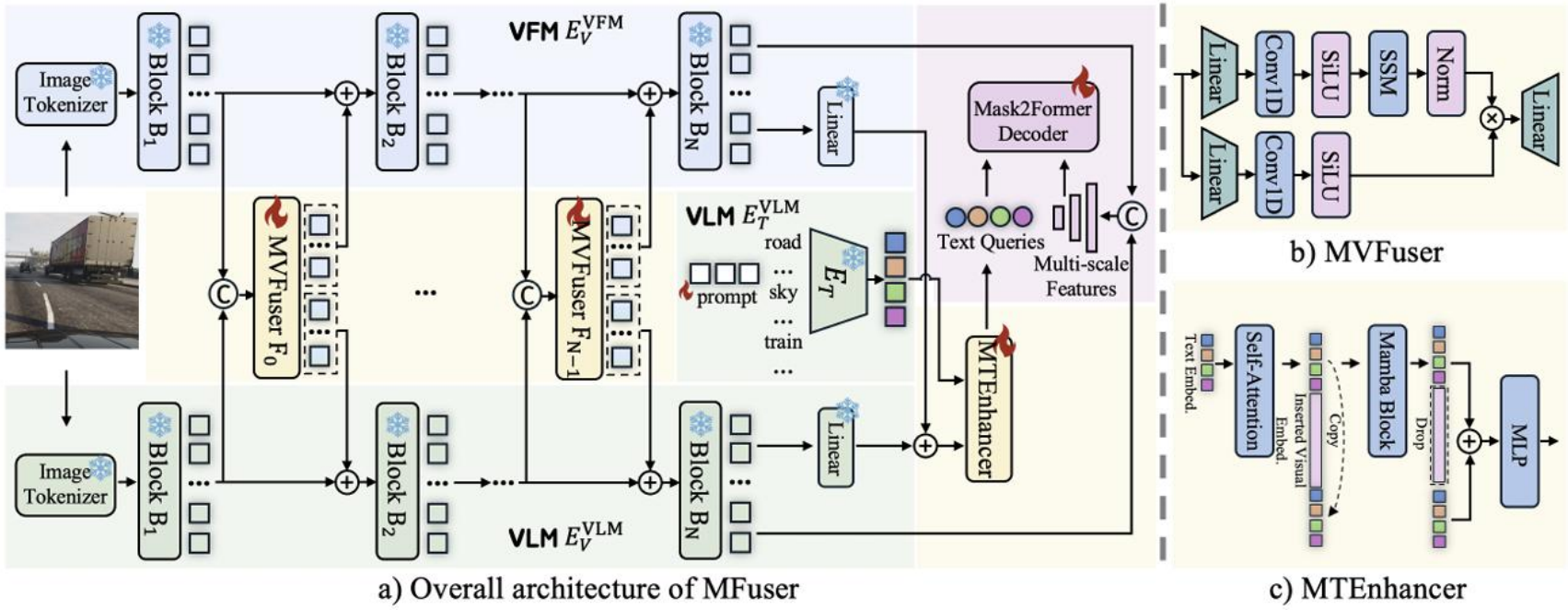
Result



MFuser

Conclusion

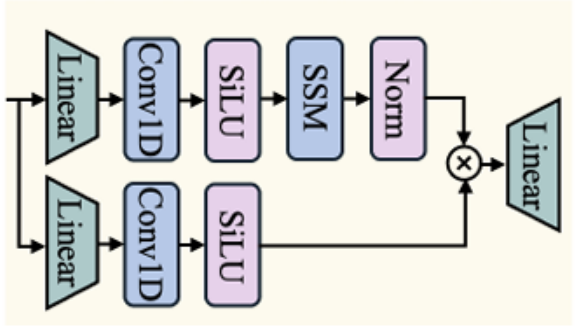
- DGSS를 위해 VFM과 VLM을 통합하도록 설계된 새로운 fusion framework인 MFuser를 제안
- MFuser는 VFM과 VLM의 상호 보완적인 강점을 활용하여 선형 복잡성을 갖춘 효율적이고 확장 가능한 fusion을 통해 증가된 patch token의 문제를 해결



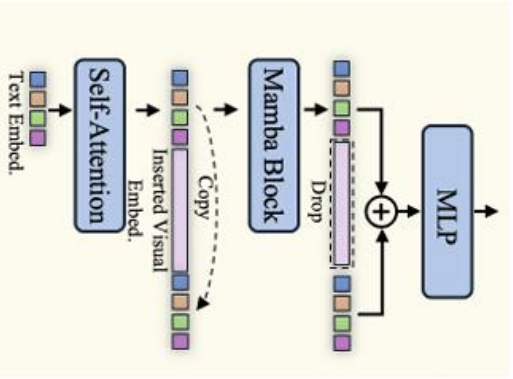
MFuser

Discussion

- MVFuser에서 gating mechanism을 사용해서 서로의 장점을 결합했다고 했는데 그 방식에 대한 설명이 좀 더 있었으면 좋겠음
- MTEnhancer에서 Text embedding을 왜 copy했는지 의문
- Parameters를 단일 모델과 비교한게 아니라 두 모델을 결합한 방법과만 비교해서 단일 모델과 비교해도 경쟁력이 있을까 의문



b) MVFuser



c) MTEnhancer

	Params. (M)	FLOPs (G)	C	B	M	Avg.
self-attn (concat.)	4.20	98.64	70.24	62.31	71.11	67.89
self-attn (separate)	8.40	71.08	69.68	61.91	70.85	67.48
bi-deform-attn	3.35	34.65	69.46	61.17	70.11	66.91
MVFuser	1.67	17.21	**70.19**	**63.13**	**71.28**	**68.20**

MFuser

Discussion

- VLM, VFM을 어떤 모델을 사용했는지에 대해 따라 성능 차이 발생
- 다른 모델에 비해 성능이 좋지만, Domain-Generalized 관점에서는 오히려 데이터셋에 따라 편차가 심함
- Mamba model은 이미지 처리가 아니라 자연어 처리 기반으로 만들어짐
- 이미지를 처리하려면 추가적인 연산이 필요해 Low Parameters로 좋은 성능을 내는 Mamba의 장점이 사라짐

Method	Backbone	synthetic-to-real			
		G→C	G→B	G→M	Avg.
SAN-SAW [43]	RN101	45.33	41.18	40.77	42.43
WildNet [29]	RN101	45.79	41.73	47.08	44.87
SHADE [66]	RN101	46.66	43.66	45.50	45.27
TLDR [27]	RN101	47.58	44.88	48.80	47.09
FAMix [14]	RN101	49.47	46.40	51.97	49.28
SHADE [67]	MiT-B5	53.27	48.19	54.99	52.15
IBAFormer [53]	MiT-B5	56.34	49.76	58.26	54.79
VLTSeg [25]	CLIP-B	47.50	45.70	54.30	49.17
CLOUDS [2]	ConvNeXt-L	60.20	57.40	67.00	61.50
VLTSeg [25]	EVA02-L	65.60	58.40	66.50	63.50
Rein [55]	EVA02-L	65.30	60.50	64.90	63.60
Rein [55]	DINOv2-L	66.40	60.40	66.10	64.30
SET [63]	DINOv2-L	68.06	61.64	67.68	65.79
tqdm [40]	EVA02-L	68.88	59.18	70.10	66.05
MFuser	CLIP-L	**71.24**	61.08	71.14	67.82
MFuser	SIGLIP-L	71.10	61.19	**71.71**	68.00
MFuser	EVA02-L	70.19	**63.13**	71.28	**68.20**

Mask R-CNN 1× schedule				
Backbone	AP^b	AP^m	Params	FLOPs
Swin-T	42.7	39.3	48M	267G
ConvNeXt-T	44.2	40.1	48M	262G
VMamba-T	47.3	42.7	50M	271G
Swin-S	44.8	40.9	69M	354G
ConvNeXt-S	45.4	41.8	70M	348G
VMamba-S	48.7	43.7	70M	349G
Swin-B	46.9	42.3	107M	496G
ConvNeXt-B	47.0	42.7	108M	486G
VMamba-B	49.2	44.1	108M	485G
Mask R-CNN 3× MS schedule				
Swin-T	46.0	41.6	48M	267G
ConvNeXt-T	46.2	41.7	48M	262G
NAT-T	47.7	42.6	48M	258G
VMamba-T	48.8	43.7	50M	271G
Swin-S	48.2	43.2	69M	354G
ConvNeXt-S	47.9	42.9	70M	348G
NAT-S	48.4	43.2	70M	330G
VMamba-S	49.9	44.2	70M	349G

ADE20K with crop size 512				
Backbone	mIOU (SS)	mIOU (MS)	Params	FLOPs
ResNet-50	42.1	42.8	67M	953G
DeiT-S + MLN	43.8	45.1	58M	1217G
Swin-T	44.5	45.8	60M	945G
ConvNeXt-T	46.0	46.7	60M	939G
NAT-T	47.1	48.4	58M	934G
Vim-S	44.9	-	46M	-
VMamba-T	47.9	48.8	62M	949G
ResNet-101	43.8	44.9	86M	1030G
DeiT-B + MLN	45.5	47.2	144M	2007G
Swin-S	47.6	49.5	81M	1039G
ConvNeXt-S	48.7	49.6	82M	1027G
NAT-S	48.0	49.5	82M	1010G
VMamba-S	50.6	51.2	82M	1028G
Swin-B	48.1	49.7	121M	1188G
ConvNeXt-B	49.1	49.9	122M	1170G
NAT-B	48.5	49.7	123M	1137G
RepLKNet-31B	49.9	50.6	112M	1170G
VMamba-B	51.0	51.6	122M	1170G

감사합니다.