Test-Time Training Done Right

Tianyuan Zhang

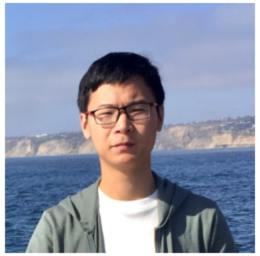
June 9th, 2025

ASAP Seminar Series:

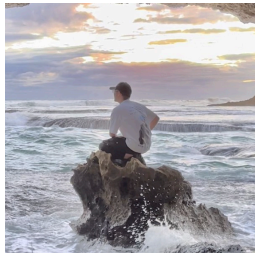
Advances in Sequence modeling from Algorithmic Perspectives

Test-Time Training Done Right

Tianyuan Zhang



Sai Bi



Yicong Hong

Kai Zhang

Fujun Luan

Songlin Yang

Kalyan Sunkavalli

Bill Freeman

Hao Tan

https://tianyuanzhang.com/projects/ttt-done-right/

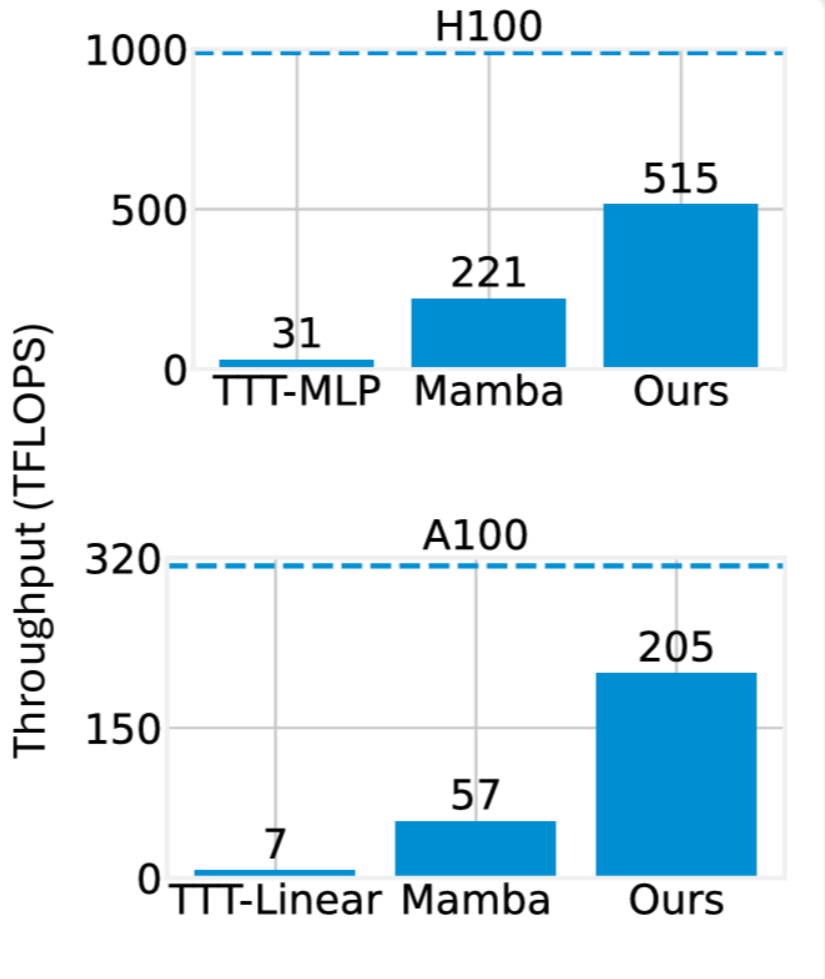
Outline

- What is Test-Time Training, and why Test-Time Training.
- What does "Test-Time Training **Done Right**" mean.
- Details and insights about "Test-Time Training Done Right".

Test-Time Training **Done Right**

- 10x GPU FLOPs utilization.
- Without cumbersome kernel code.

Large online batch size (**c**hunk-size) **t**est-time training(LaCT)



(a) GPU Throughput

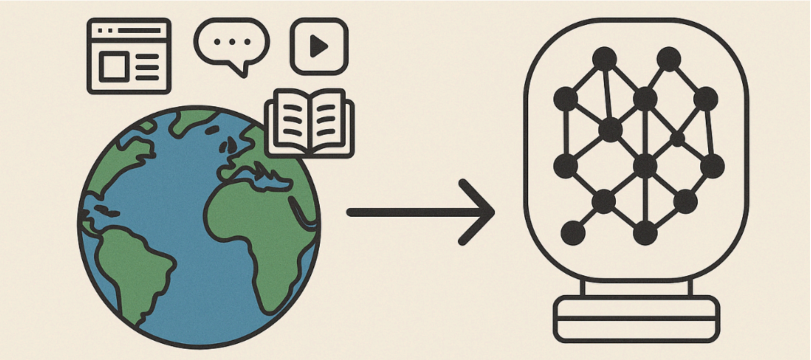
What is Test-Time Training

- General meaning:

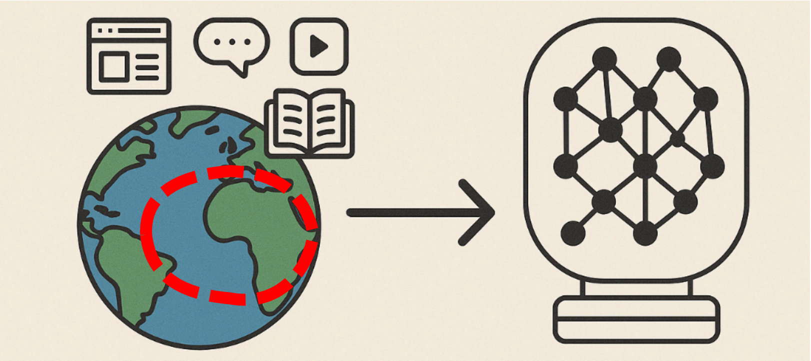
- Most current work focus on:

Current Training Paradigm

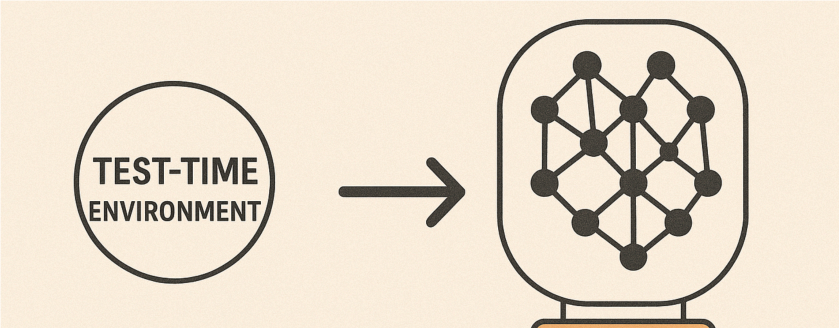
Pretraining:
Compress world knowledge



Post training:
Specialize in certain domain/behaviors



Test-time training:



Sun et al. https://yueatsprograms.github.io/ttt/home.html

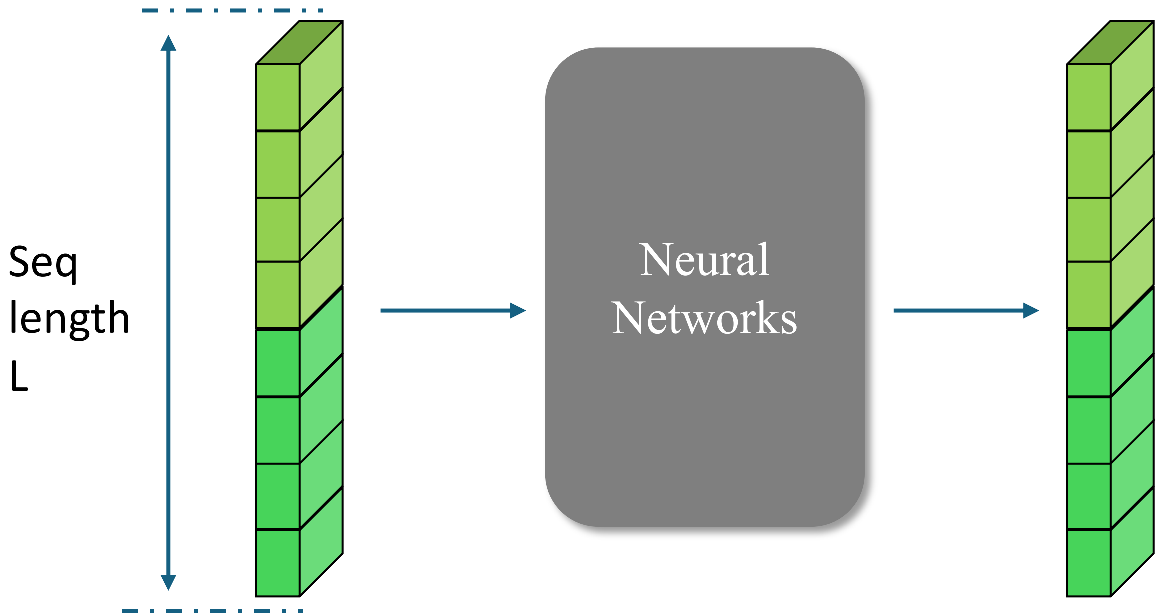
What is Test-Time Training

- General meaning:
  - One specific stage of learning.

Akyürek et al. *The Surprising Effectiveness of Test-Time Training for Few-Shot Learning. Arxiv 2024.11*
Gandelsman et al. *Test-Time Training with Masked Autoencoders. NeurIPS 2022.*

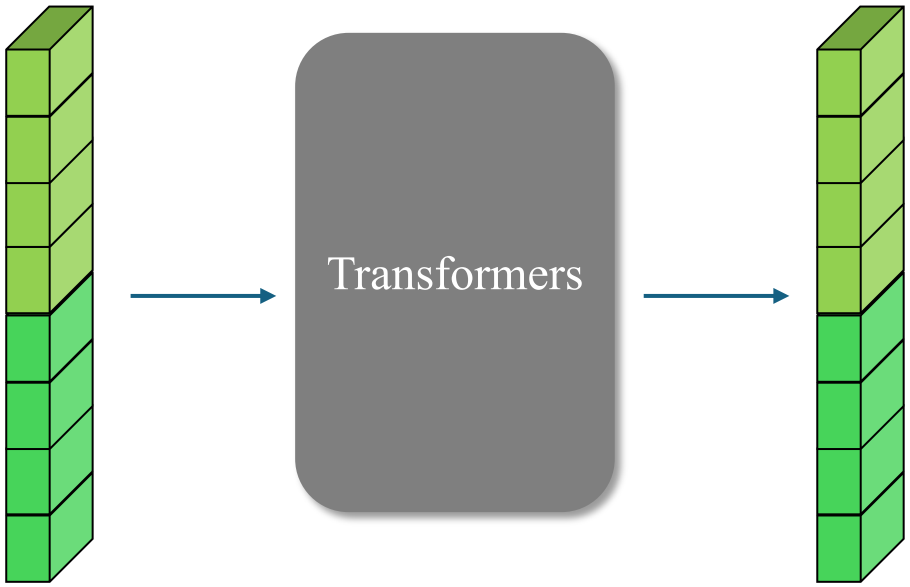
- Most current work focus on:
  - "Test-Time Training" for designing new sequence models

“Sequence” to “Sequence” models



Text, images, videos, audios, DNAs etc.

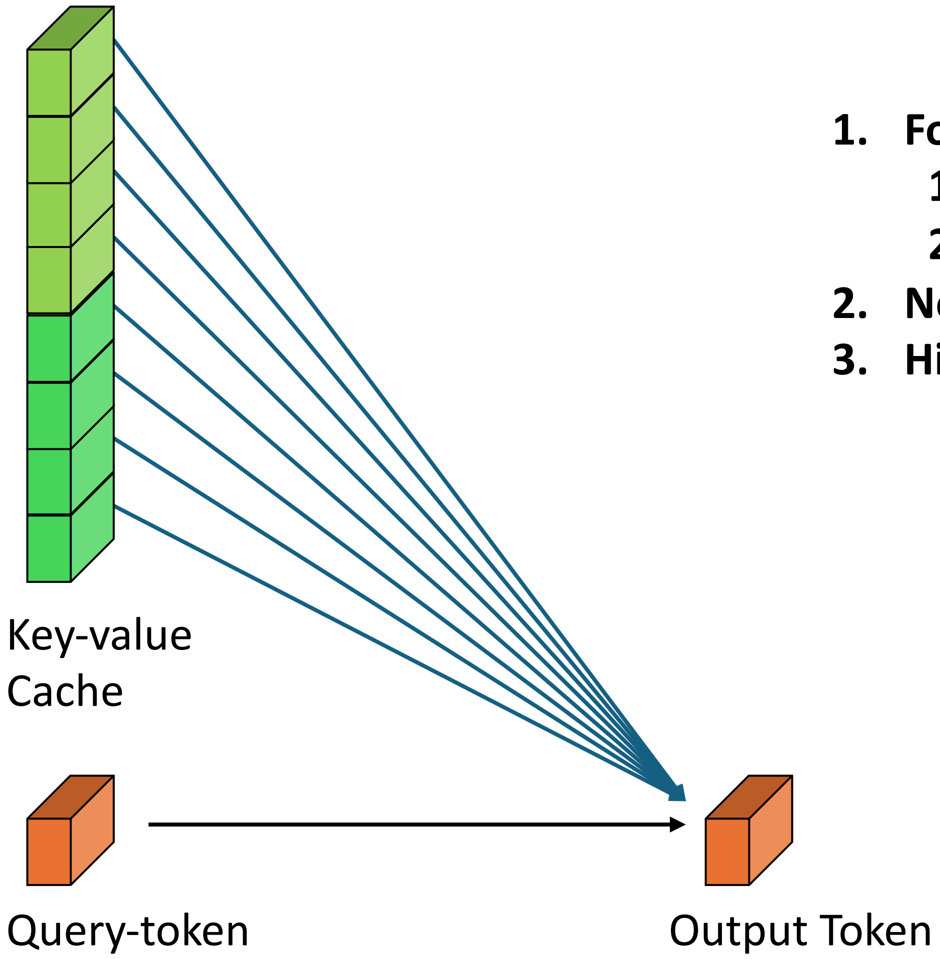
Transformer



Each token is involved in two types of computes:

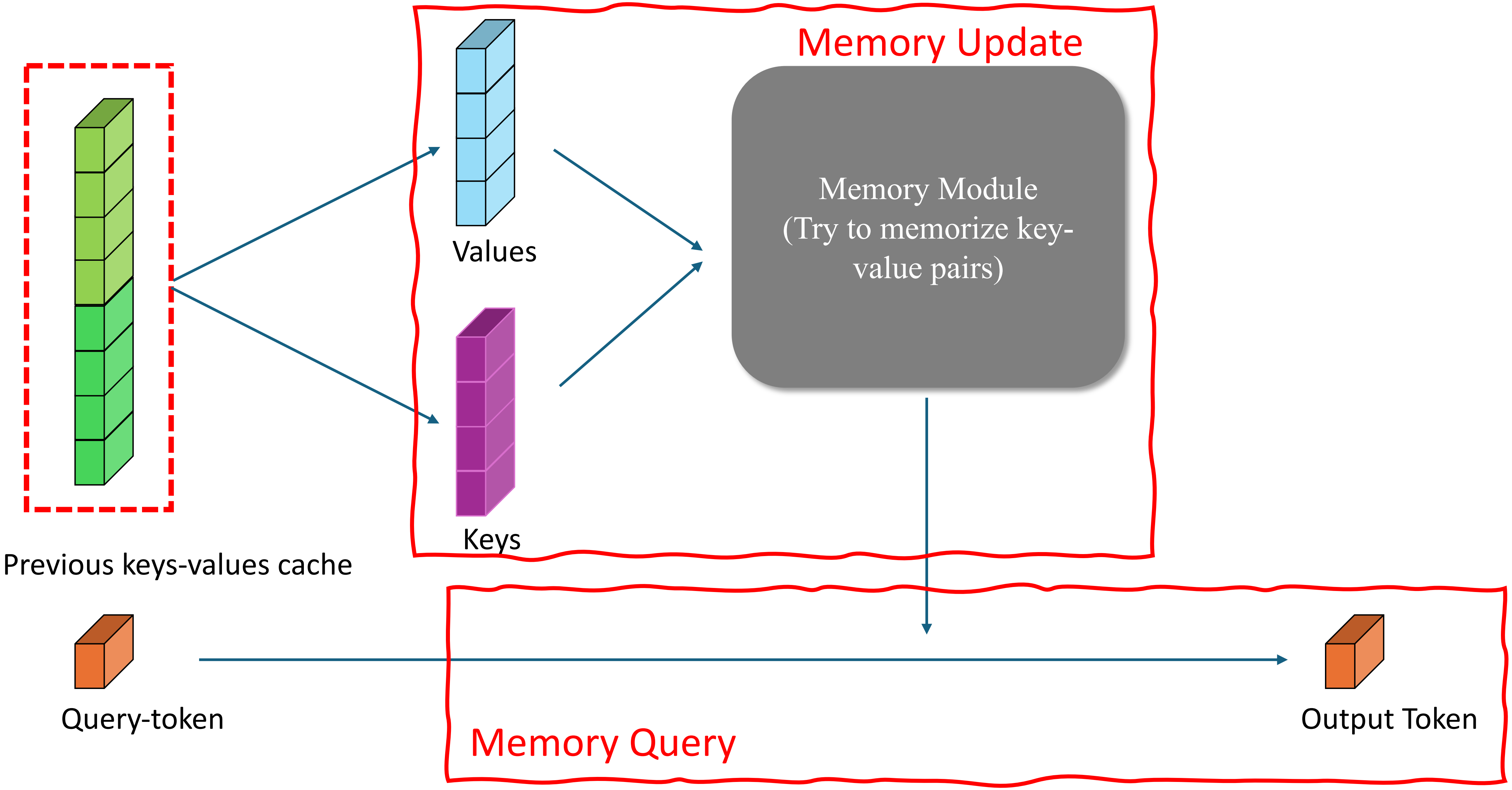
1. Per-token independently: MLP
   1. Cost: O(L)
2. Token communicate between each other: **Attention**
   1. Cost: **O(L^2)**

Attention: no in-context compression



1. **For every new token:**
   1. **O(n) memory**
   2. **O(n) compute**
2. **No in-context compression**
3. **High parallelism**

One example of memory module



Test-Time Training for new sequence models

- Input Sequence: $\boldsymbol{x} = [x_1, x_2, \dots, x_N], x_i \in \mathrm{R}^d$
  - Each token will be split into *query (q)* , *key (k)*, *value (v)*

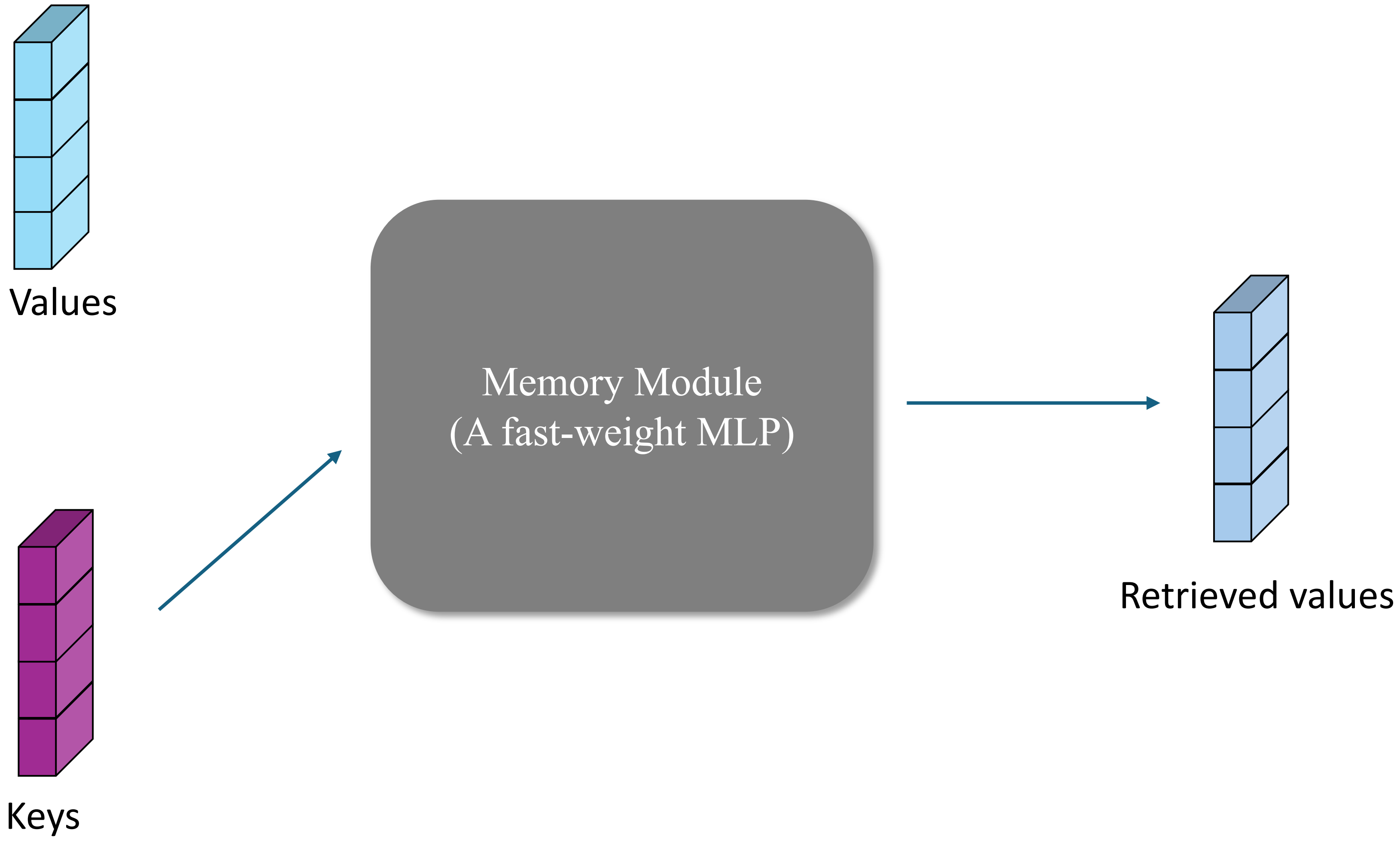
- Fast weight function: $f_W(\cdot) : \mathrm{R}^d \to \mathrm{R}^d$
  - W as the online adapted weight, which stores memory
  - f_W could be neural networks, linear, MLP, or even a transformer.

Memory Update as online gradient descent

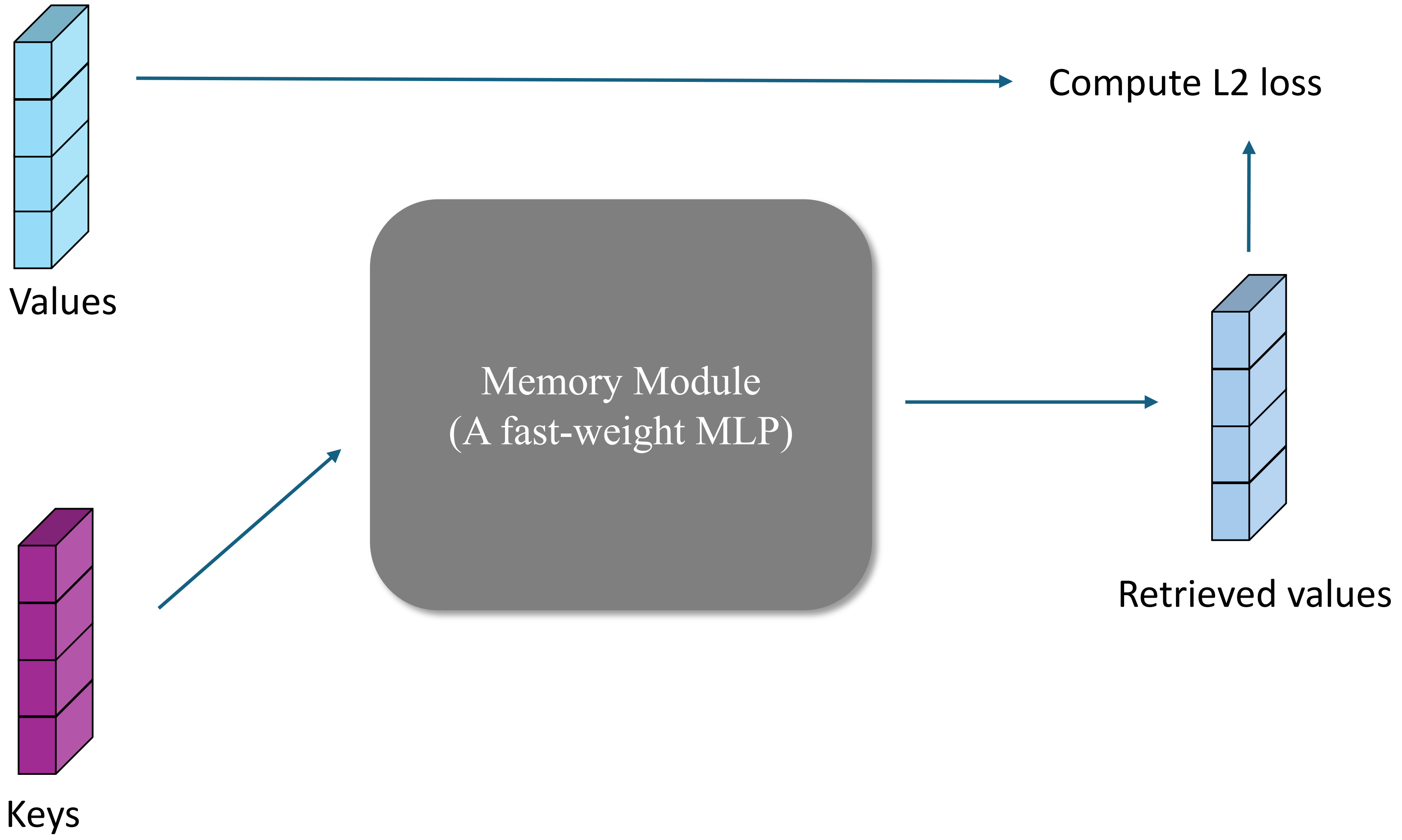
$$W = W - \nabla_W L(f_W(k), v)$$

- Common online objectives:
  - Key-Value Association:
    - $L_{\text{dot}} = -f_W(k)^T v$
    - $L_2 = |f_W(k) - v|_2^2$

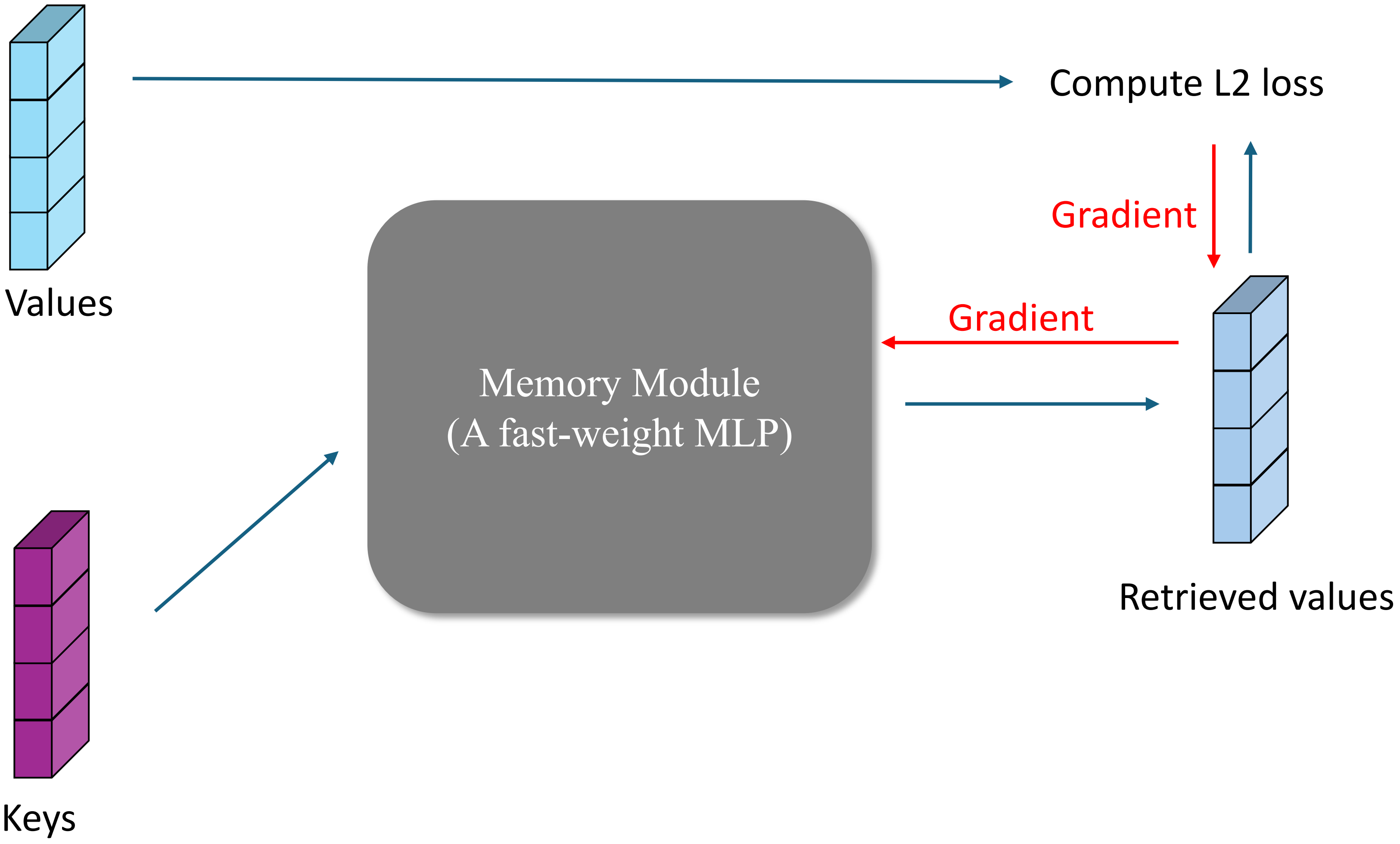
Memory Update as online gradient descent



Memory Update as online gradient descent



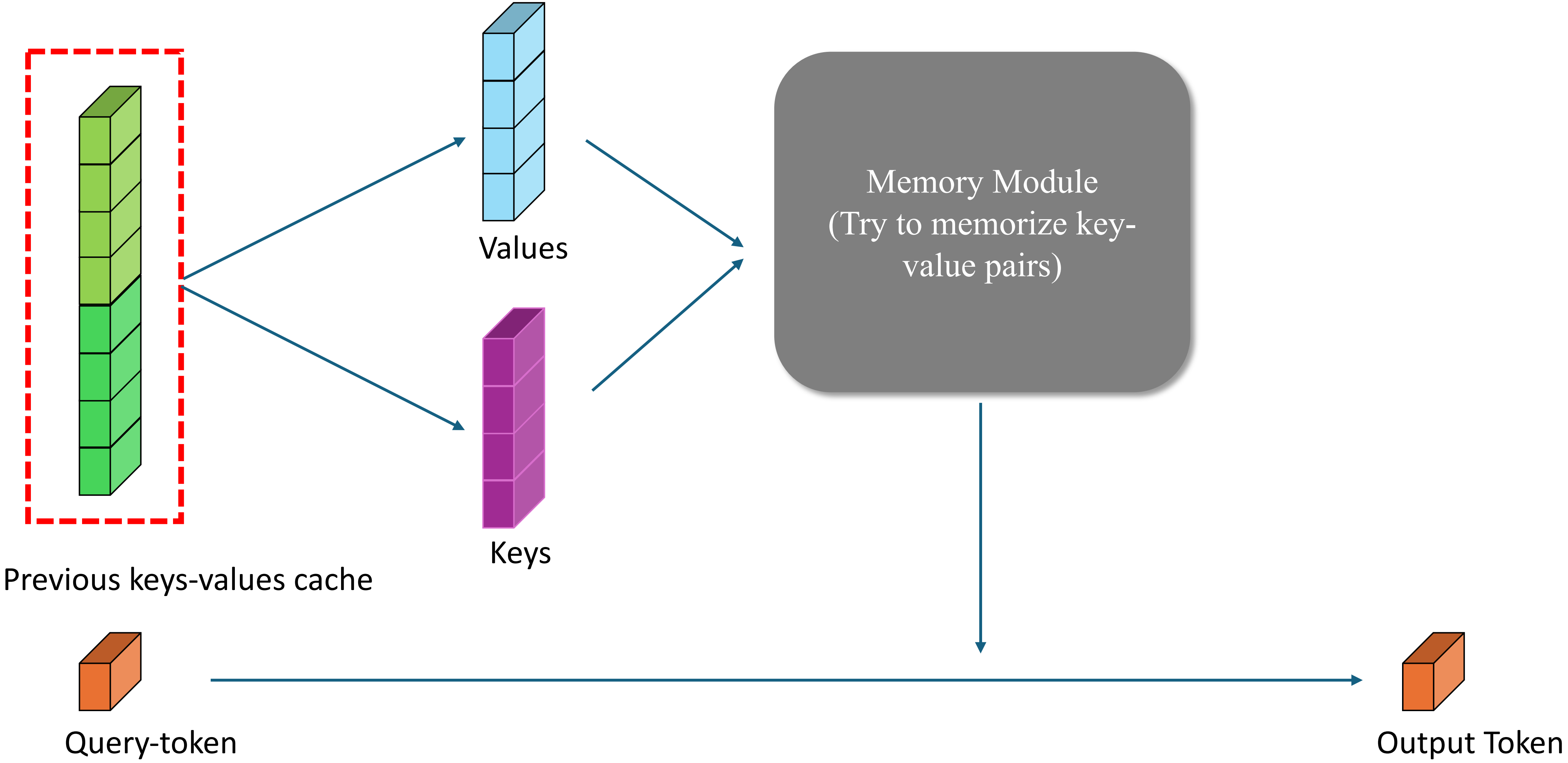
Memory Update as online gradient descent



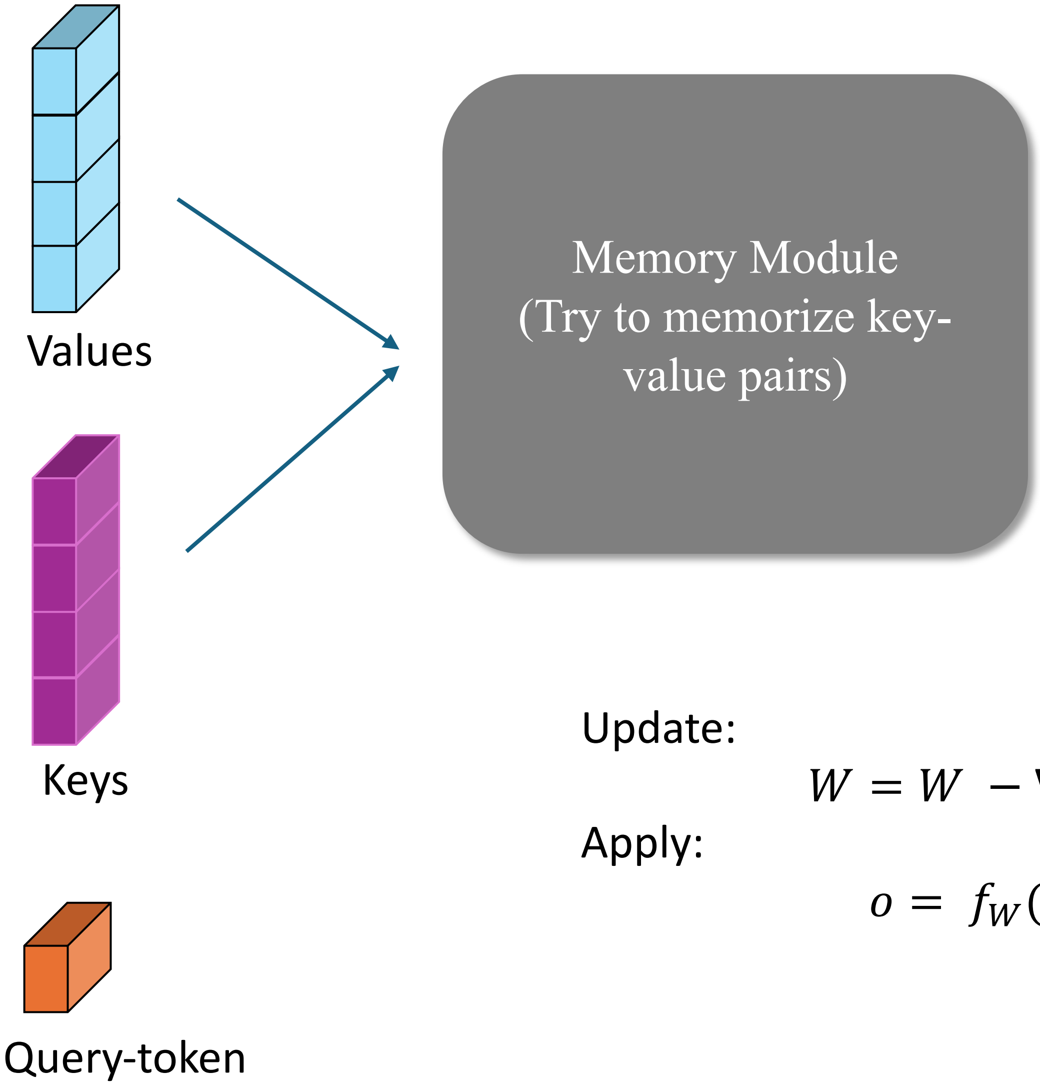
Memory query (called apply)

- $o = f_W(q)$

Fast weight MLP as memory



Fast weight MLP as memory



Update:

$$W = W - \nabla_W L(f_W(k), v)$$

Apply:

$$o = f_W(q)$$

TTT Opens a vast Design Space

- Fast weight functions.
- Test-time training objectives.
- Test-time training optimizers.

Second-order gradients?

Forward Pass

- $W = W - \nabla_W L(f_{\boxed{W}}(\boxed{k}), \boxed{v})$
- $\boxed{o} = f_{\boxed{W}}(\boxed{q})$

Gradient flows

Hardware friendly Test-Time Training

Hardware friendly: Tensor cores

Technical Specifications	
	H100 SXM
FP64	34 teraFLOPS
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS
INT8 Tensor Core*	3,958 TOPS
GPU Memory	80GB
GPU Memory Bandwidth	3.35TB/s
Decoders	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)

989 TFLOPS for dense **matmuls**

[m, k] @ [k, n] -> [m, n]. k>=16

Hardware friendly: Tensor cores

- Tensor core only do 2D matmul
  - [M, K] @ [K, N] -> [M, N].
  - For H100 with bf16, smallest K should be 16
  - $\nabla_W L(f_W(k), v)$ contains lot's of matrix-vector multiplication.

- Online minibatch size >= 16.
  - $\sum \nabla_W L(f_W(k_i), v_i)$

Sun et al. *Learning to (Learn at Test Time): RNNs with Expressive Hidden States. Arxiv 2024.07*
Behrouz et al. *Titans: Learning to Memorize at Test Time. arxiv 2025.01*

Hardware friendly: compute intensity

Technical Specifications	
	H100 SXM
FP64	34 teraFLOPS
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS
INT8 Tensor Core*	3,958 TOPS
GPU Memory	80GB
GPU Memory Bandwidth	3.35TB/s
Decoders	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)

989 TFLOP/S / 3.35TB/s = 295 FLOPs per byte

Compute/Memory: $\frac{32d^2}{2d^2+64d}$ < ttt-batch-size

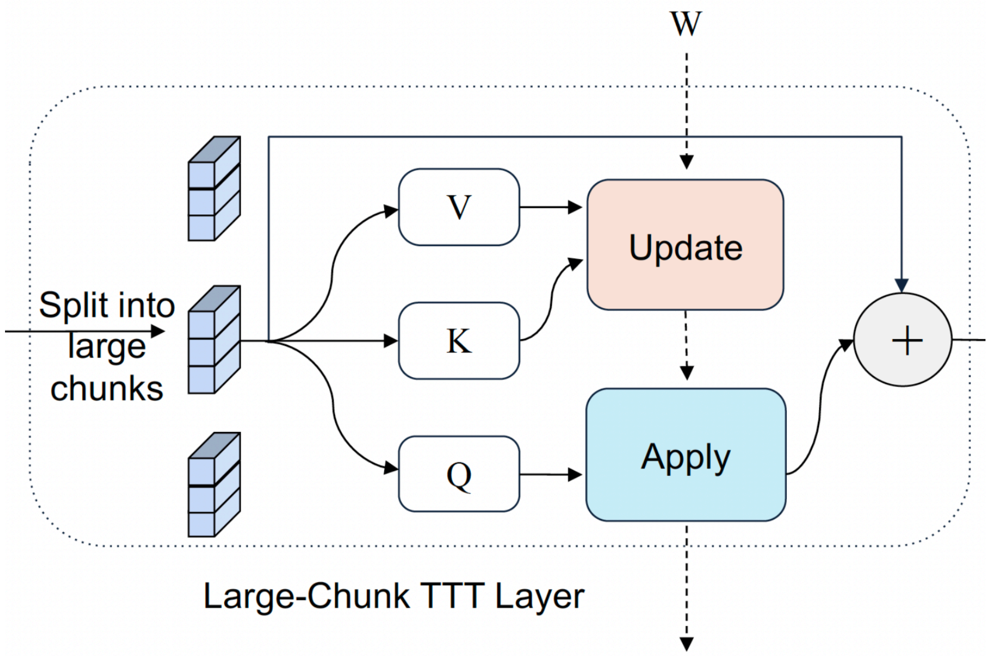
Hardware friendly: parallelism over sequence dimension

- All potential parallelism dimension:
  - Batch
  - Feature Dimension (heads)
  - Sequence Length
    - Restricted to the ttt-batch size!

All previous discussion leads to a common solution:
Use large test-time training batch size (we call it chunk-size)

Large chunk TTT is hardware friendly

- 2D matmuls
- High compute intensity
- High degree of parallelism

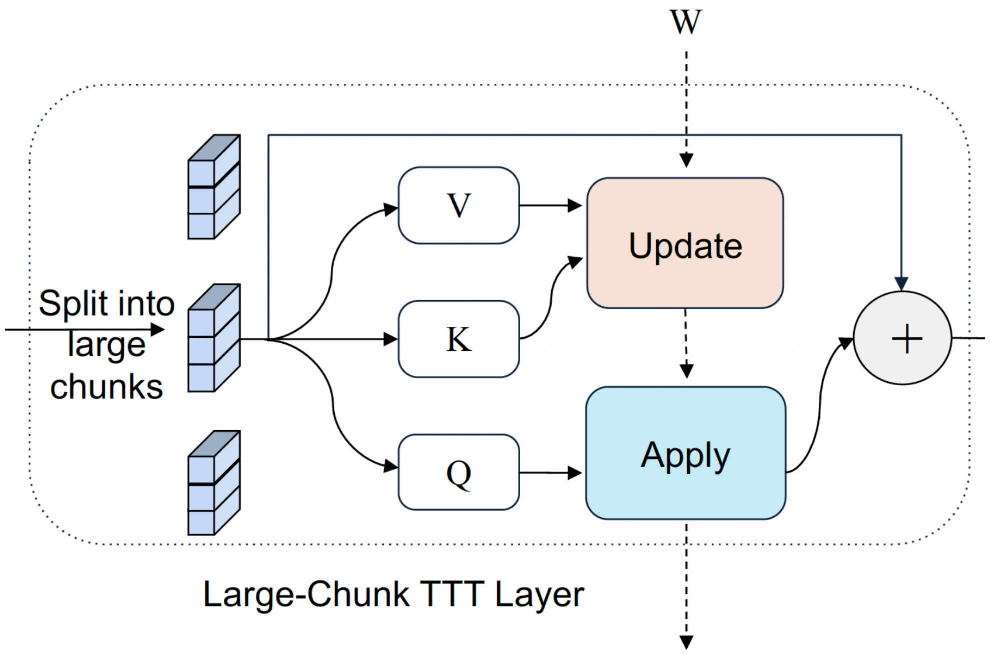


All previous discussion leads to a common solution:
Use large test-time training batch size (we call it chunk-size)

2k – 1 million tokens in our experiment

Large chunk TTT is hardware friendly

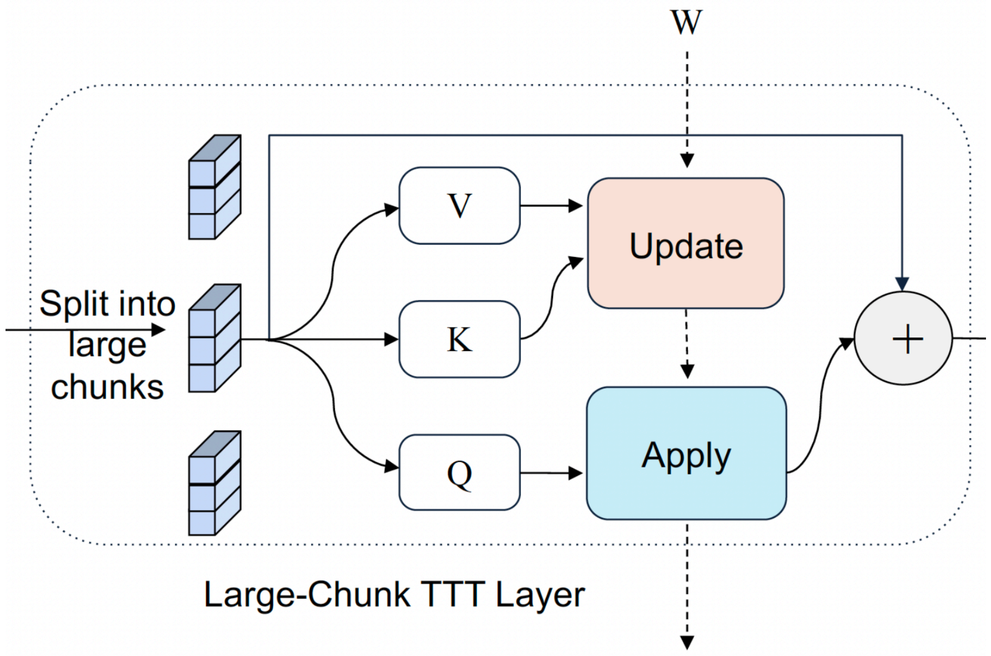
- 2D matmuls
- High compute intensity
- High degree of parallelism



More importantly, Pytorch code is enough:
No kernel codes: error prone, slower research iteration not all researcher can write kernel code

About data topology

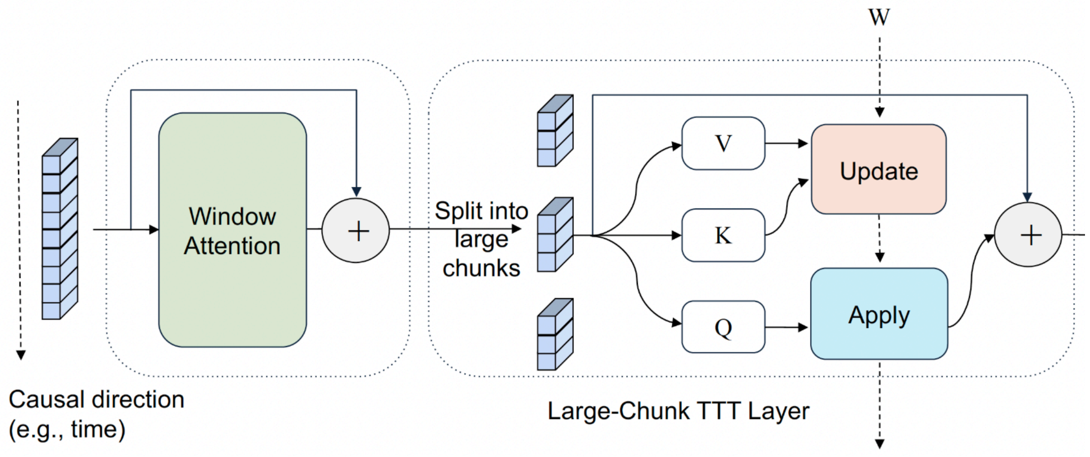
- Set within chunk
- Causal between chunks
- Natively suitable for: *sequence of set*



Positional encoding and window-attention would help

Locality handled by sliding window attention

- Attention is efficient and effective for locality in the data
- Leave the TTT's limited state size to handle long memory



Arora et al. *Simple linear attention language models balance the recall-throughput tradeoff*. 2024
Hua et al. *Transformer quality in linear time*. ICML 2022
Munkhdalai et al. *Leave no context behind: Efficient infinite context transformers with infini-attention*. 2024

Details on SwiGLU-MLP as fast weight

Fast Weight Function:

$$f_W(x) = W_2\left[\mathrm{SiLU}(W_1 x) \circ (W_3 x)\right]$$

Online training objectives:

$$\mathcal{L}\big(f_W(k_i), v_i\big) = -f_W(k_i)^\top v_i$$

GD with weight-norm:

$$\text{weight-update}(W, g) = \text{L2-Normalize}(W - g).$$

Details on SwiGLU-MLP as fast weight

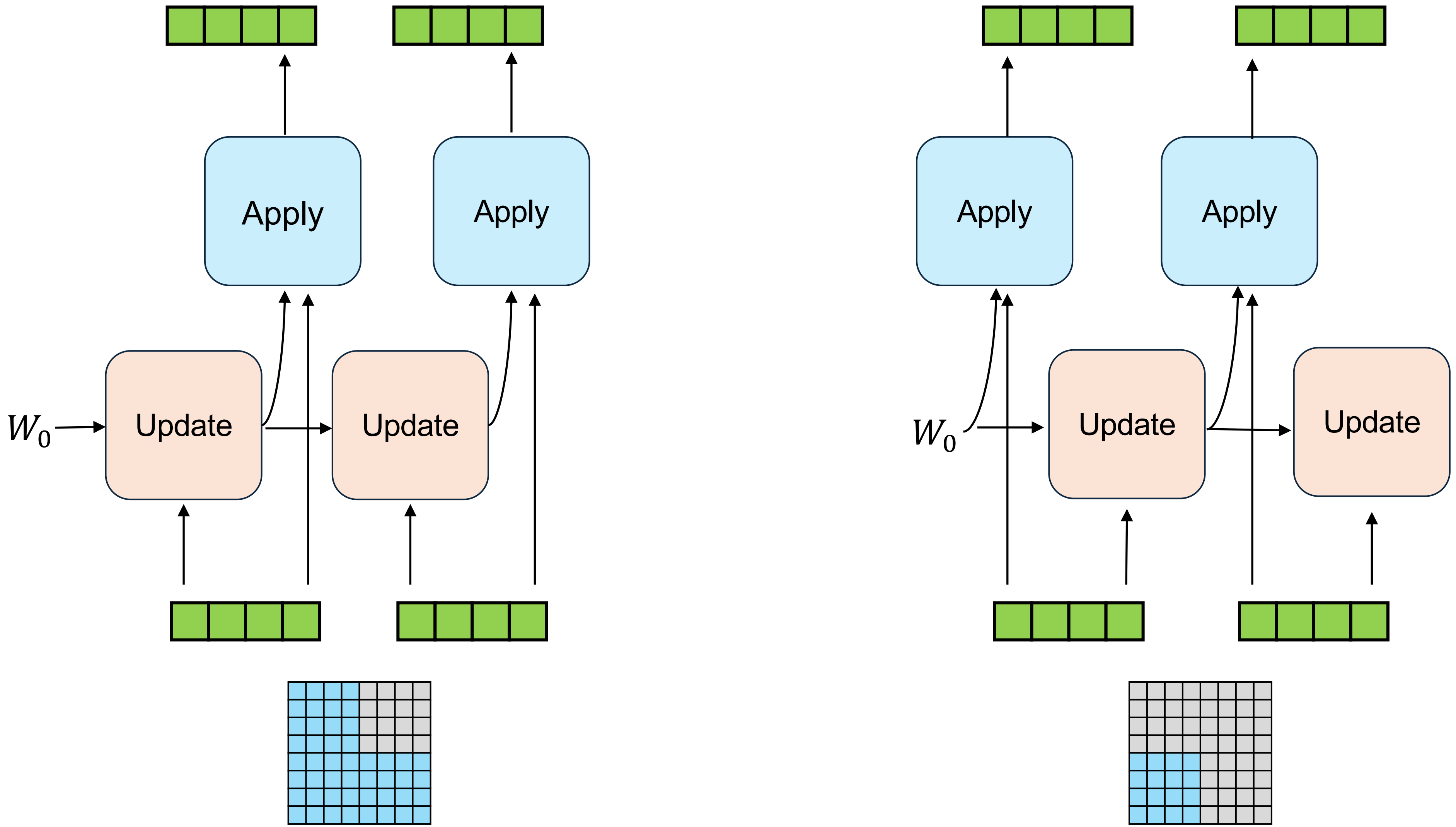
Experiments

- Novel View Synthesis
  - Set of images
- Language models
  - 1D order sequence
- Autoregressive video generation
  - Sequence of images

LaCT for language model

- Chunk-structure in language?
  - Chunk size as hyper-params:
    - 2048 or 4096
- Per-token causality
  - Handled by sliding window attention

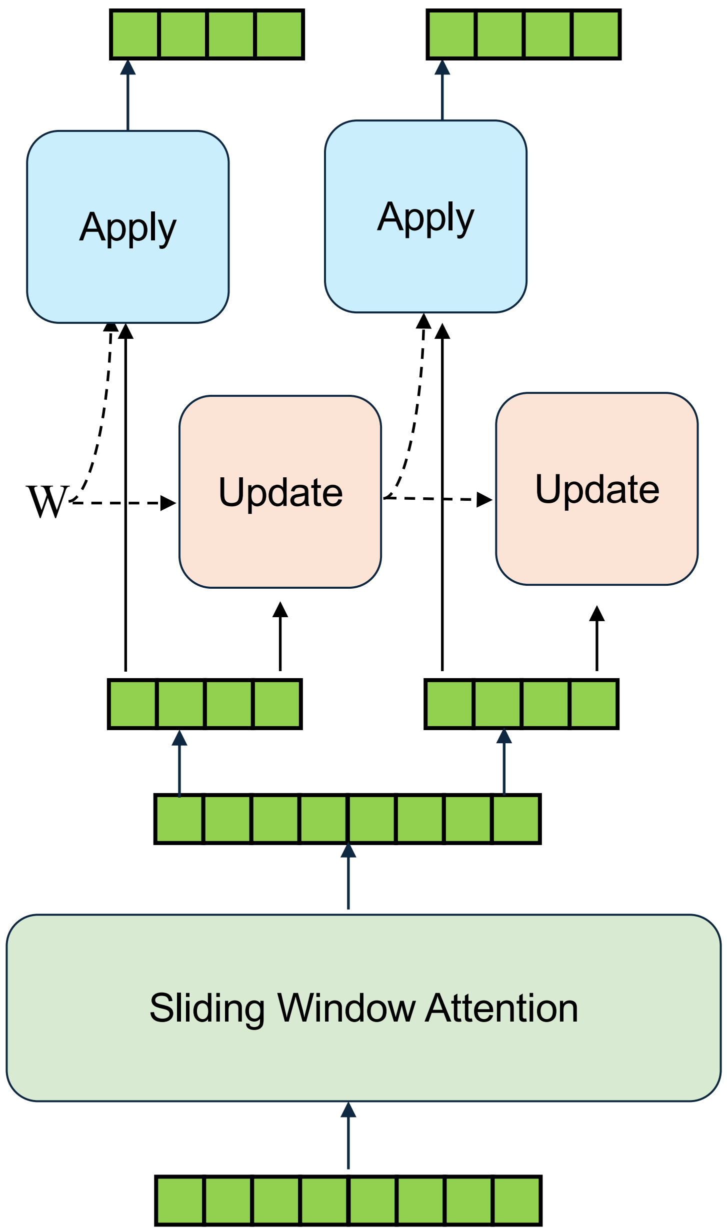
Orders between “*apply*” and “*update*”

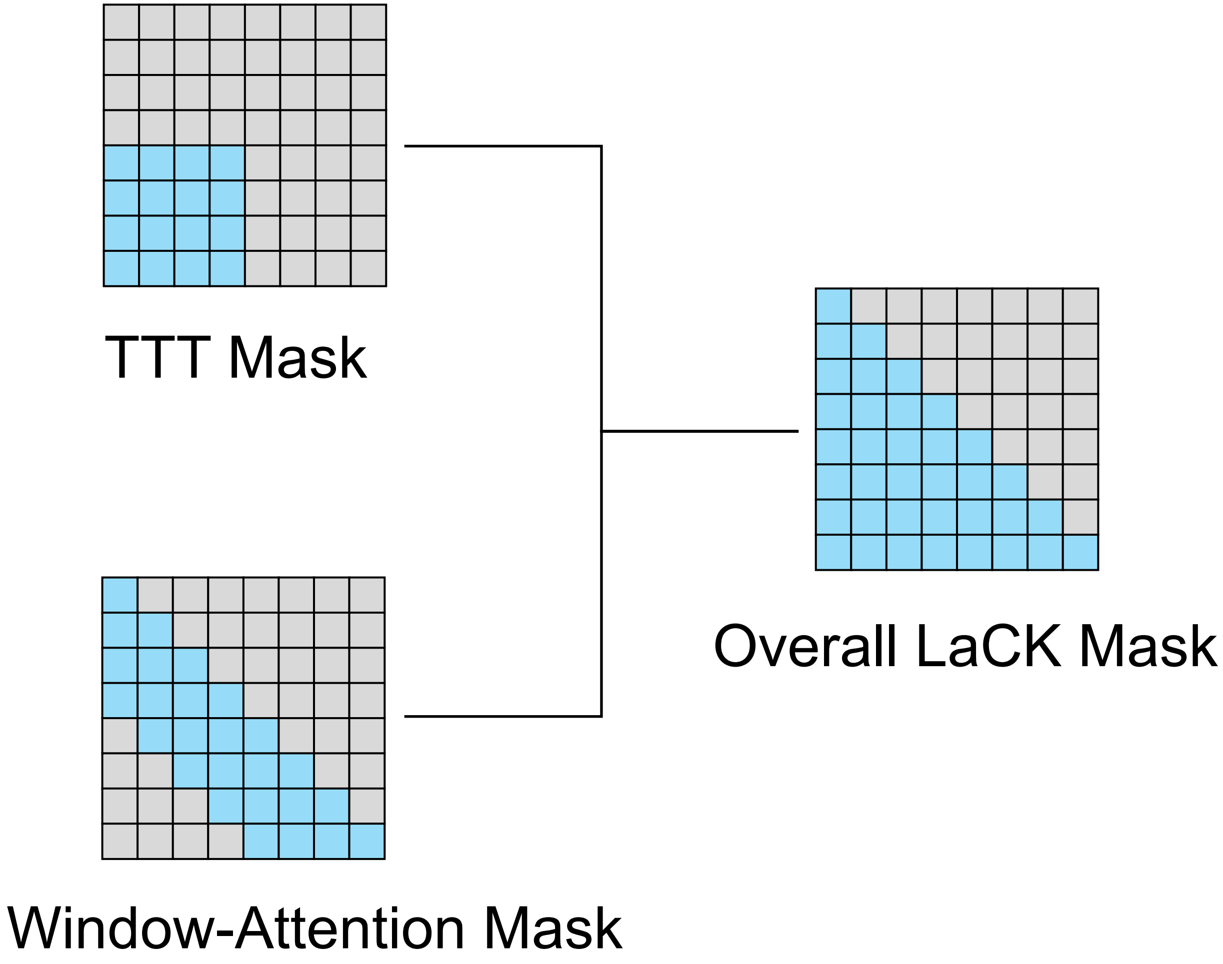


(a) Block-Wise Causal Mask

(b) Shifted Block-Wise Causal Mask

LaCT for language model





Details on sliding window attention

```
q, k, v = LinearQKV(x).split(3)

#### Local quadratic-cost window attention
attn_q = q * learnable_q_scale + learnable_q_offset # per-channel rescale and shift
attn_k = k * learnable_k_scale + learnable_k_offset # per-channel rescale and shift
attn_o = local_softmax_multihead_attn(attn_q, attn_k, v, attn_mask)
```

Hua et al. *Transformer Quality in Linear Time. ICML 2022*

Baselines

	State size	Train TPS	Update Rule	Memory read-out
Transformer	–	4.1K	–	–
Transformer SWA	–	6.4K	–	–

Details:

1. RoPE base: 1M.
2. GLA: no output gate. Value has full dimension
3. DeltaNet: no short conv.
4. Extra params: < 3%.

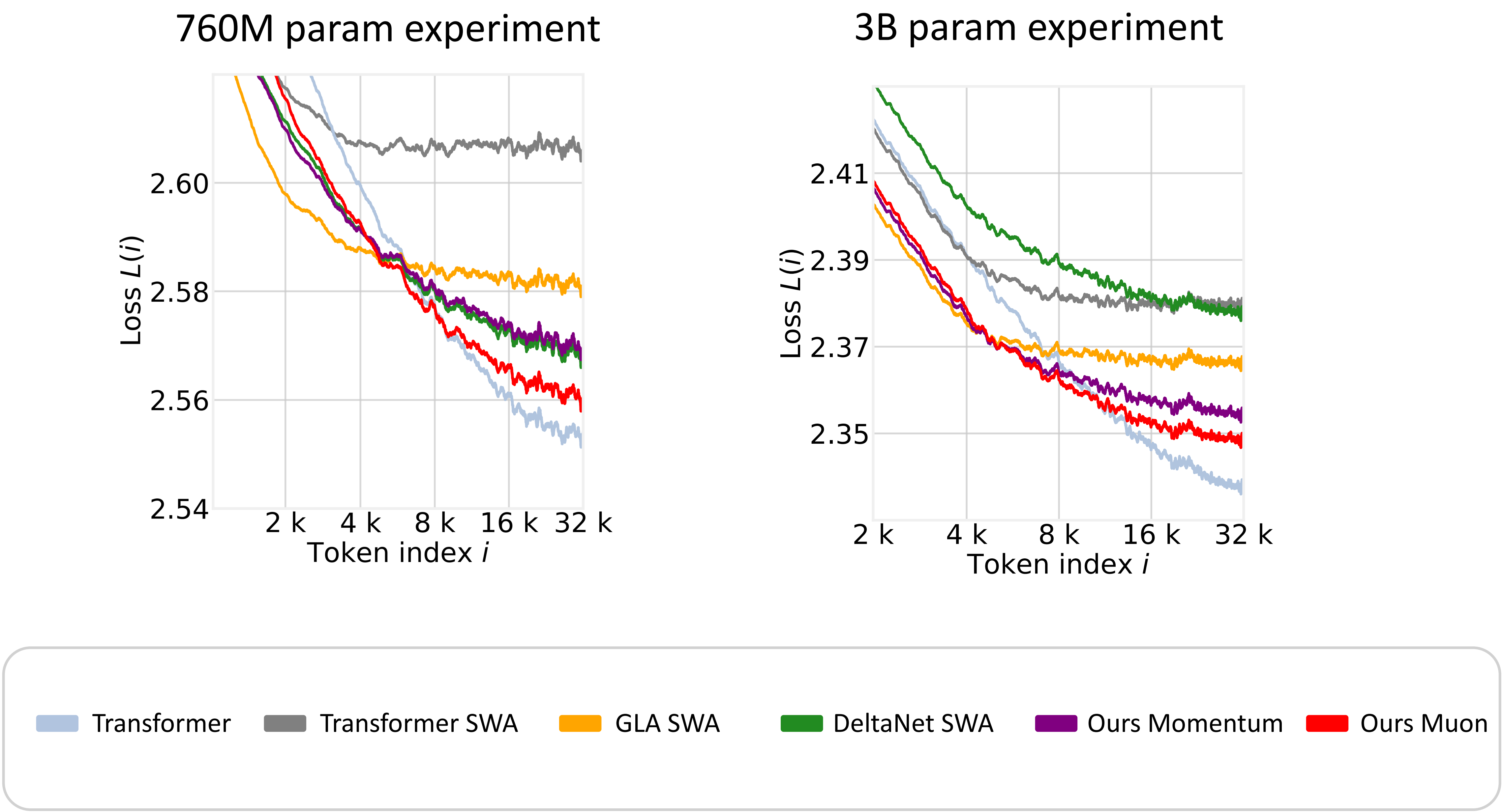
Zhang, Yu and Yang, Songlin. https://github.com/fla-org/flame

Experiment setup

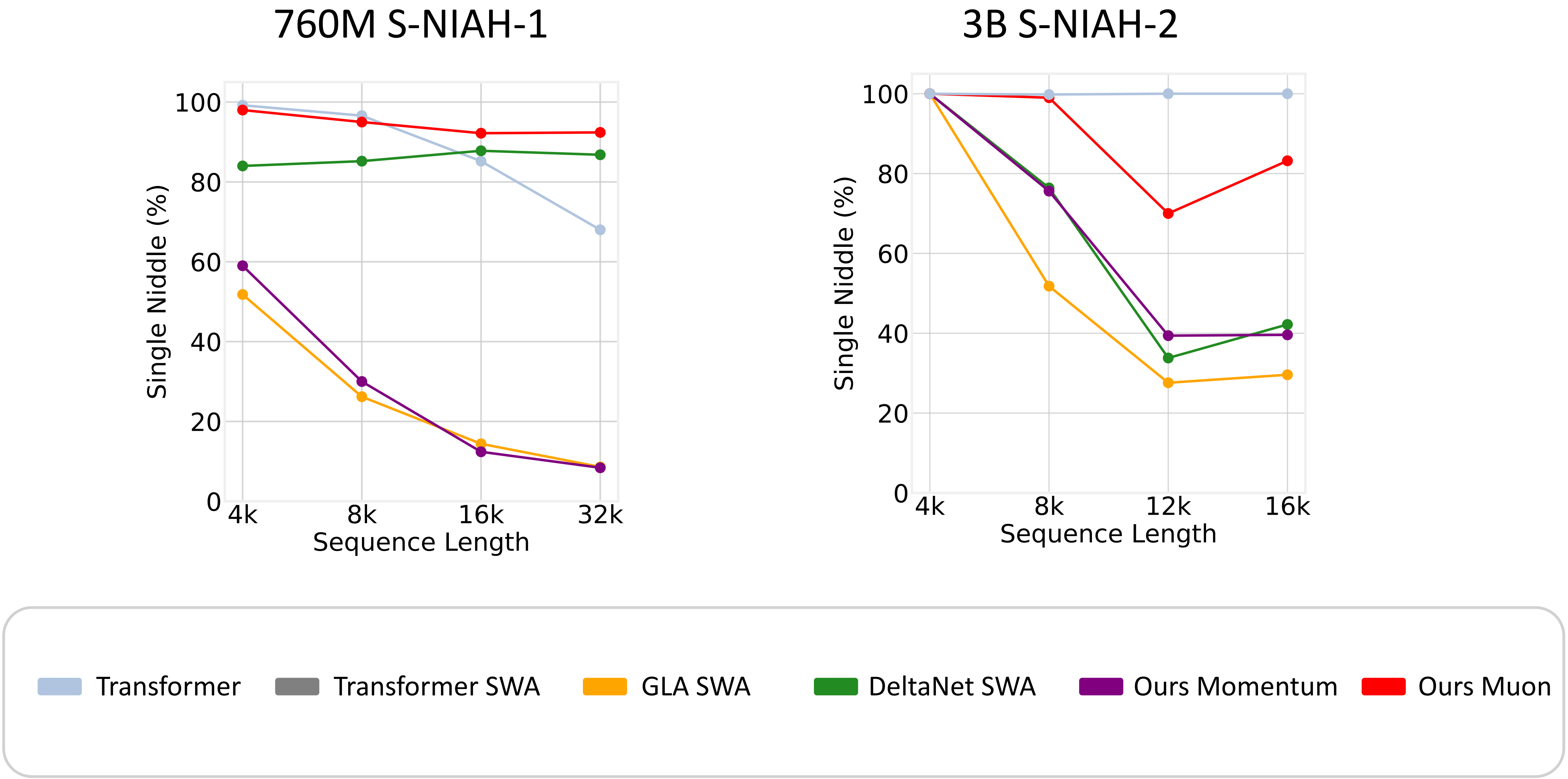
- Two scales:
  - 760M model with 40B text tokens.
    - Seq len: 32k
    - SWA size = chunk size = 2048
  - 3B model with 60B text tokens.
    - Seq len: 32k
    - SWA size = chunk size = 4096

- Evaluation:
  - Measure validation loss on different token positions
  - S-NIAH

Dataset: togethercomputer/Long-Data-Collections

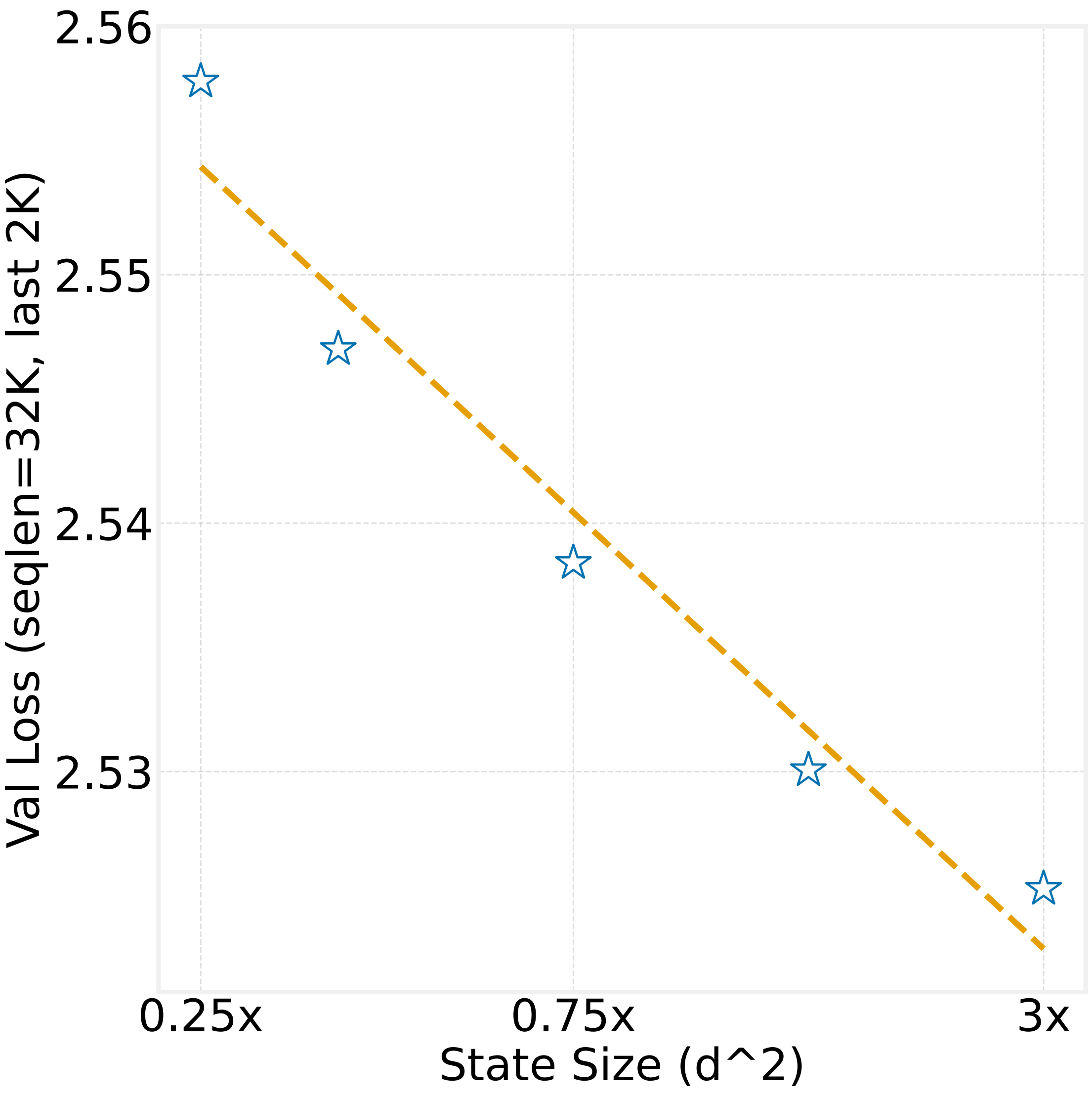


760M param experiment
3B param experiment
Loss $L(i)$
Token index i
2.60
2.58
2.56
2.54
2.41
2.39
2.37
2.35
2 k
4 k
8 k
16 k
32 k
Transformer
Transformer SWA
GLA SWA
DeltaNet SWA
Ours Momentum
Ours Muon



760M S-NIAH-1
3B S-NIAH-2
Single Niddle (%)
Sequence Length
0
20
40
60
80
100
4k
8k
16k
32k
12k
Transformer
Transformer SWA
GLA SWA
DeltaNet SWA
Ours Momentum
Ours Muon

State Size Scaling



Novel View Synthesis

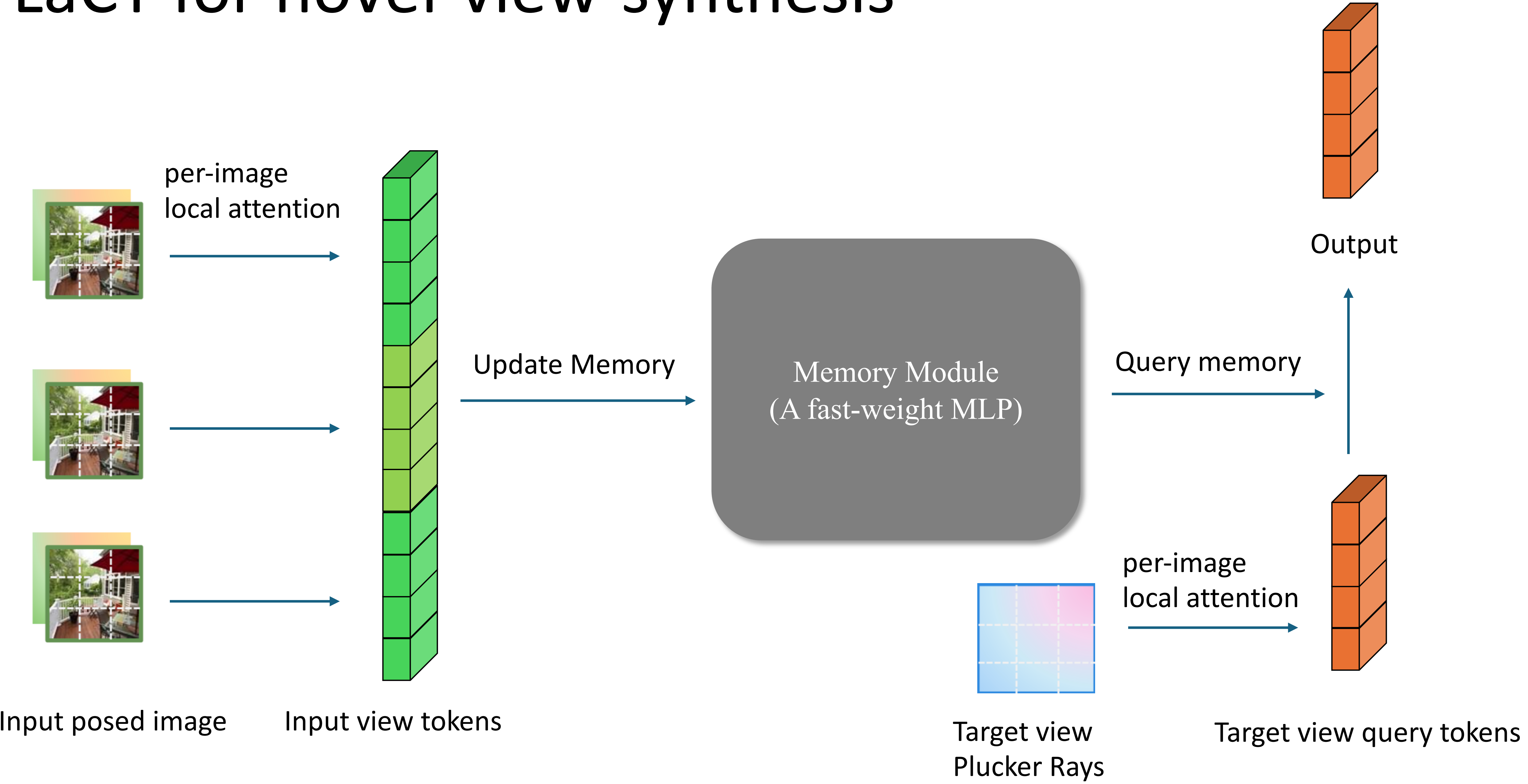
- Input:
  - Multiview posed images
  - Camera pose of novel views
- Outputs:
  - Novel views.

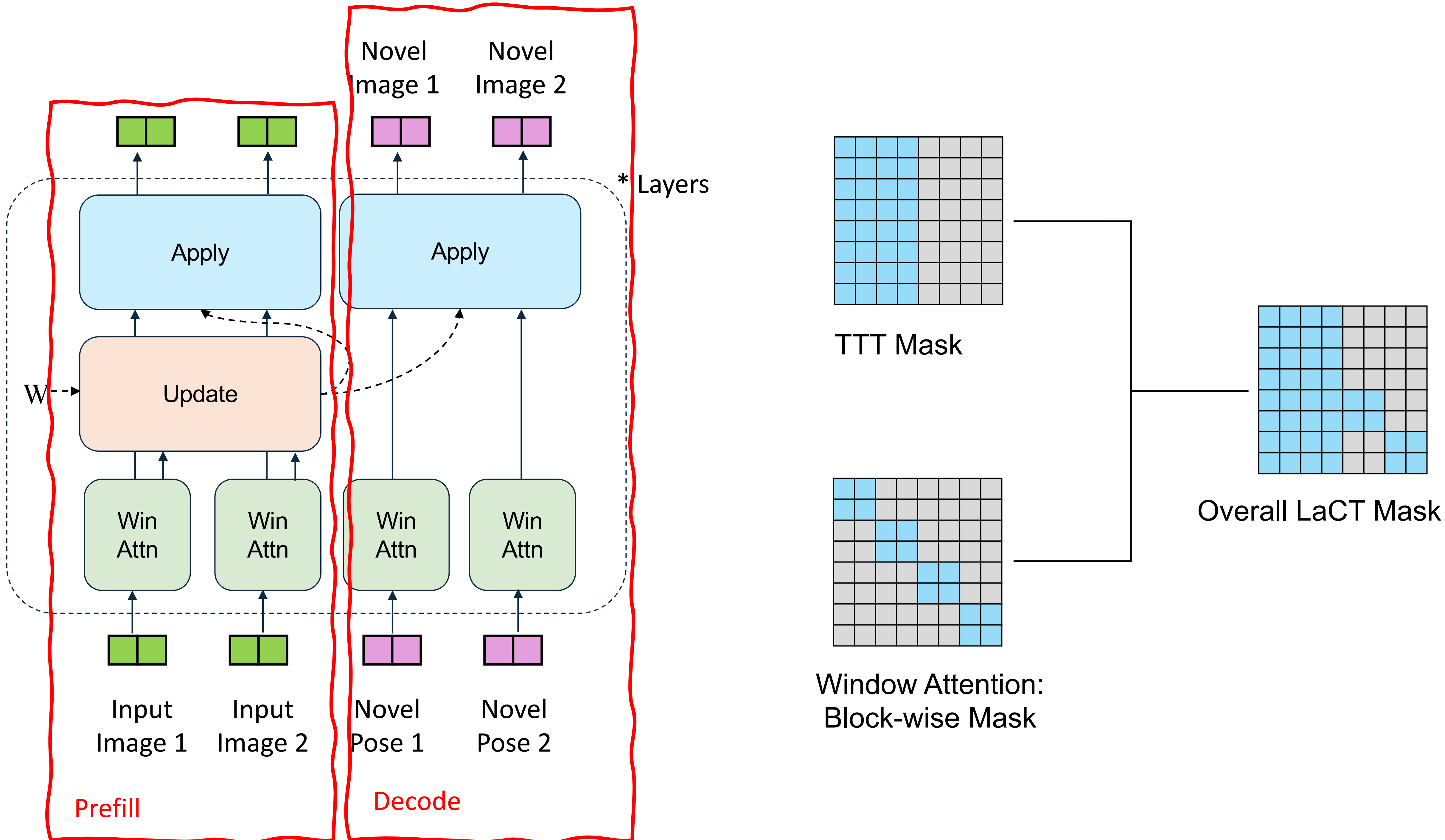
Why novel view synthesis?

- "Smart retrieval" task:
  - Retrieval: all information about novel view are provided in current sequence.
  - Smart: certain level of 3D reasoning is needed.
- *"Compression"* rather than "*Global-Random-Access*".
- Support various sequence length.

1. A small-sized model is good enough => Fast research iteration
2. Golden metrics exist => Effective research iteration

LaCT for novel-view-synthesis




Novel Image 1
Novel Image 2
* Layers
Apply
Apply
W
Update
Win Attn
Win Attn
Win Attn
Win Attn
Input Image 1
Input Image 2
Novel Pose 1
Novel Pose 2
Prefill
Decode
TTT Mask
Window Attention: Block-wise Mask
Overall LaCT Mask

Prefill and Decode

- Memory update => Prefill
- Memory readout => Decode.
  - Decoding is fixed cost.
  - 37 FPS on A100 for 512x512 images.

Baseline

- Full-Attention
  - Replace LaCT with two attentions:

Prefill: • Input tokens self-attention.

Decode: • Novel view tokens cross-attend to Input tokens.

	State Size	**Prefill Compute**	**Decoding Compute**
Full attention	$O(n)$	$O(n^2)$	$O(n)$

- Register Attention (Perceiver-style)
  - Replace LaCT with two attentions:

Prefill: • Input – register full attention

Decode: • Novel view tokens cross-attend to register tokens.

Baseline

	State Size	Prefill Compute	Decoding Compute	# Params	Prefill speed	Rendering FPS
Full attention	$O(n)$	$O(n^2)$	$O(n)$	284M	16.1 s	2.3 FPS
Perceiver Attention	$O(1)$	$O(n^2)$	$O(1)$	287M	16.8 s	34.4 FPS
Ours	$O(1)$	$O(n)$	$O(1)$	312M	1.4 s	38.7 FPS

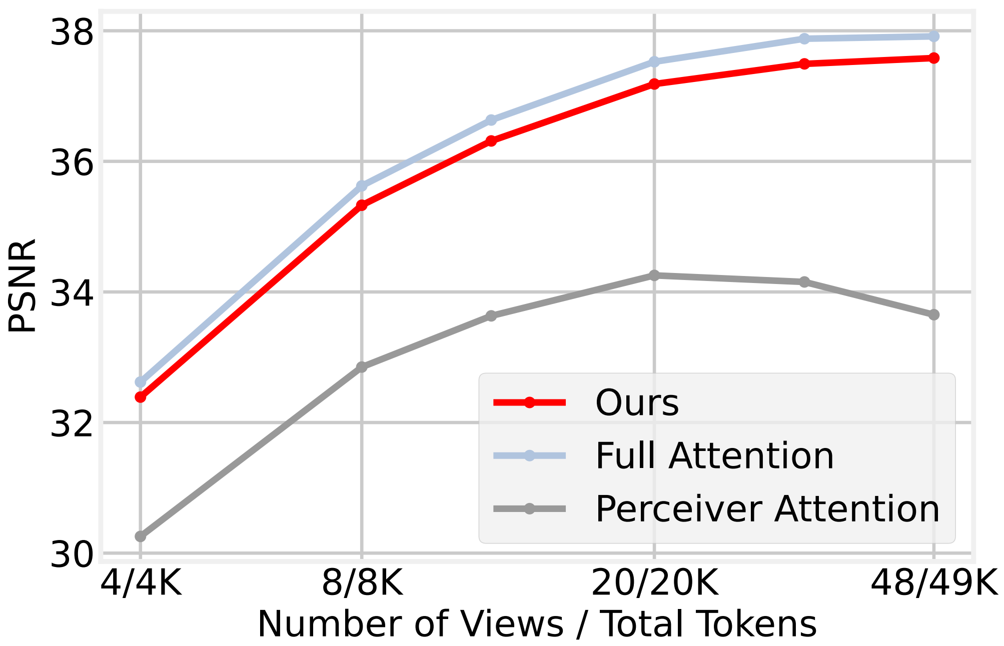
Speed tested on A100 with 48 512x512 input images => 196K image tokens

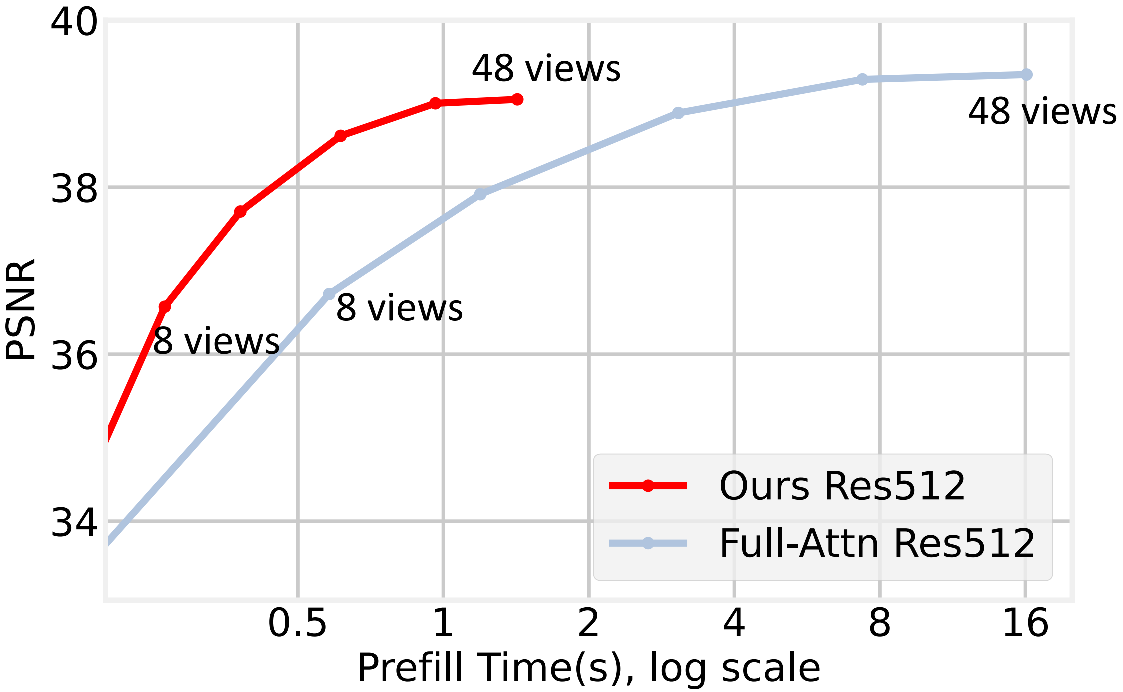
Experiment setup

Object dataset:
4-48 images
Resolution: 256x256 or 512x512

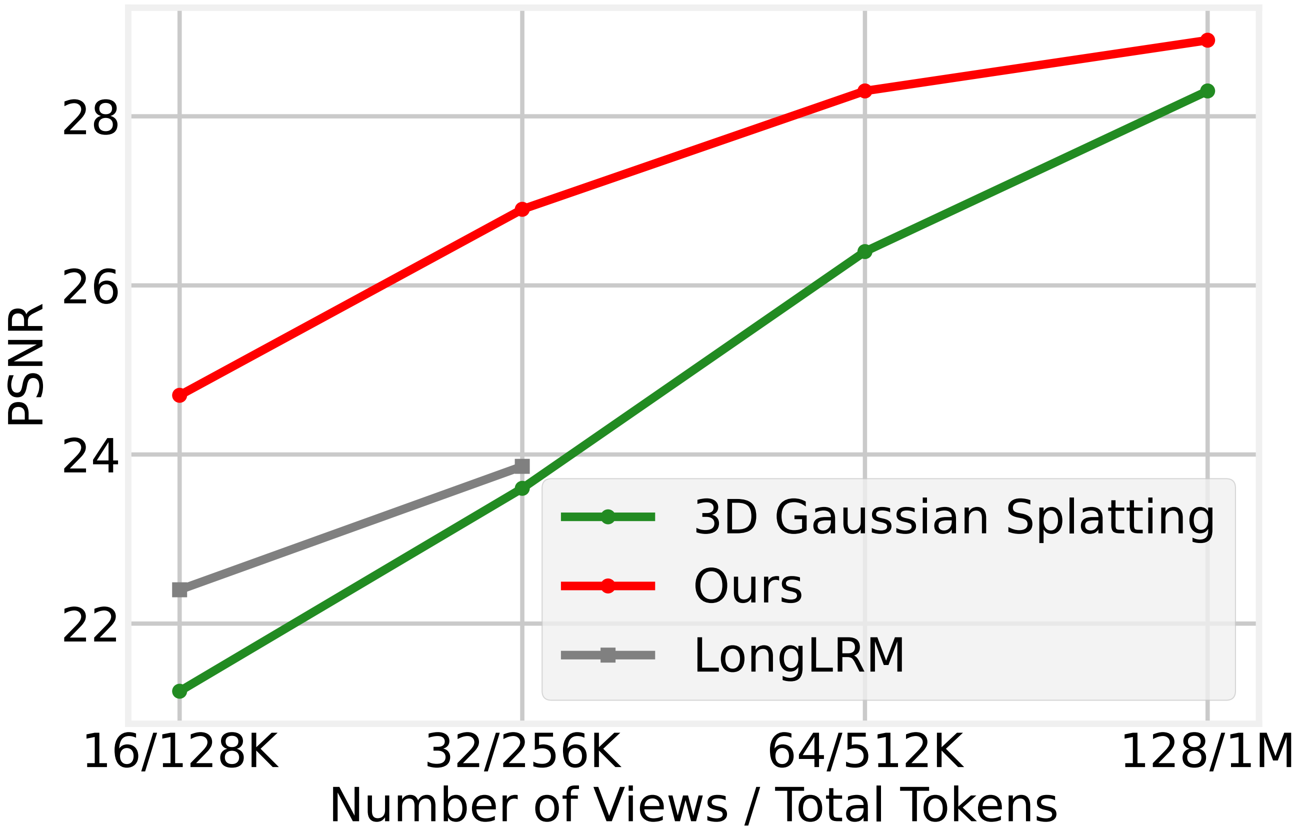
Scene dataset:
16-128 images
Resolution: 960x536

Results on object dataset





Results on scene dataset



LaCT for auto-regressive video diffusion

Teacher forcing training or AR video diffusion

Interleaved sequence

Noisy frame T=1

Clean frame T=1

Noisy frame T=2

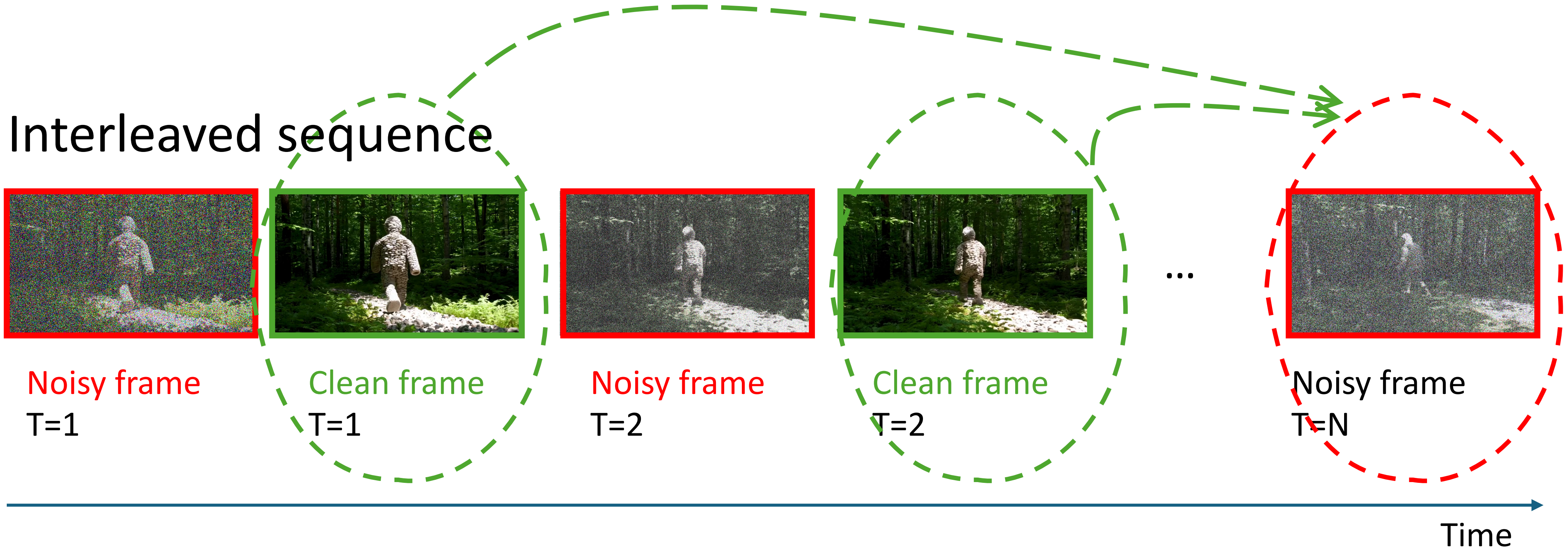
Clean frame T=2

...

Noisy frame T=N

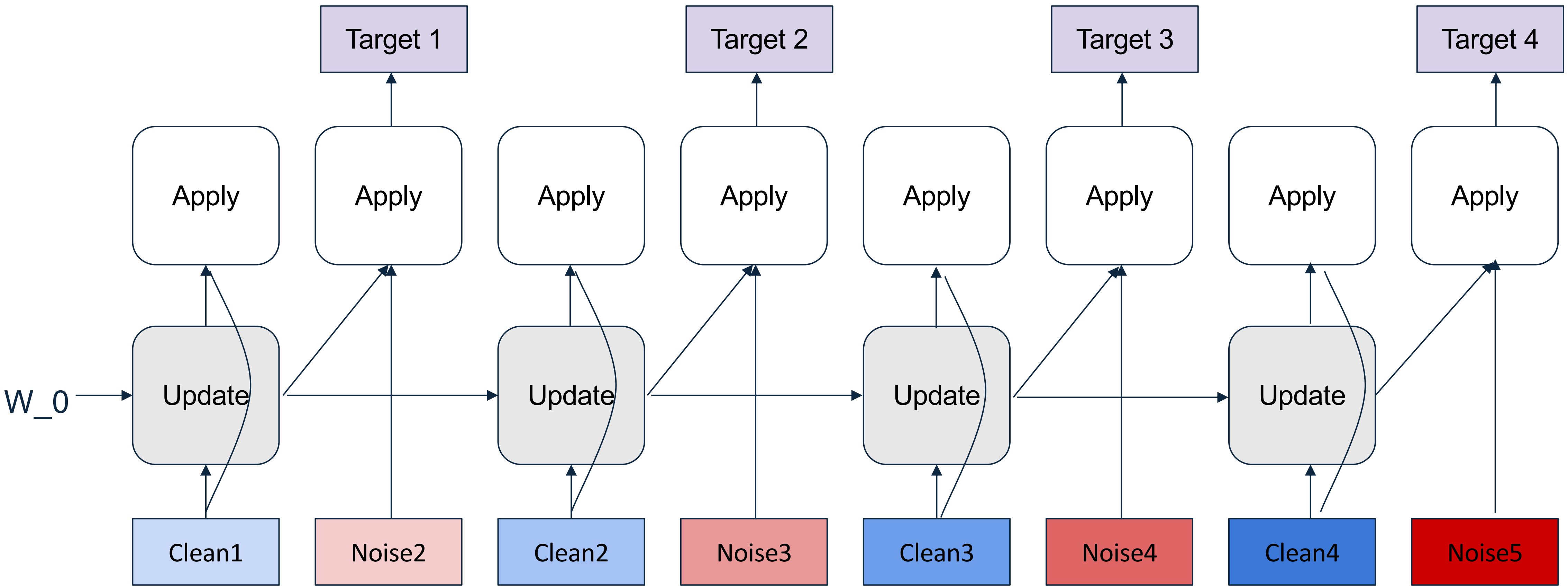
Time

Teacher forcing training or AR video diffusion



LaCT for AR video: Only update fast weight on clean frames

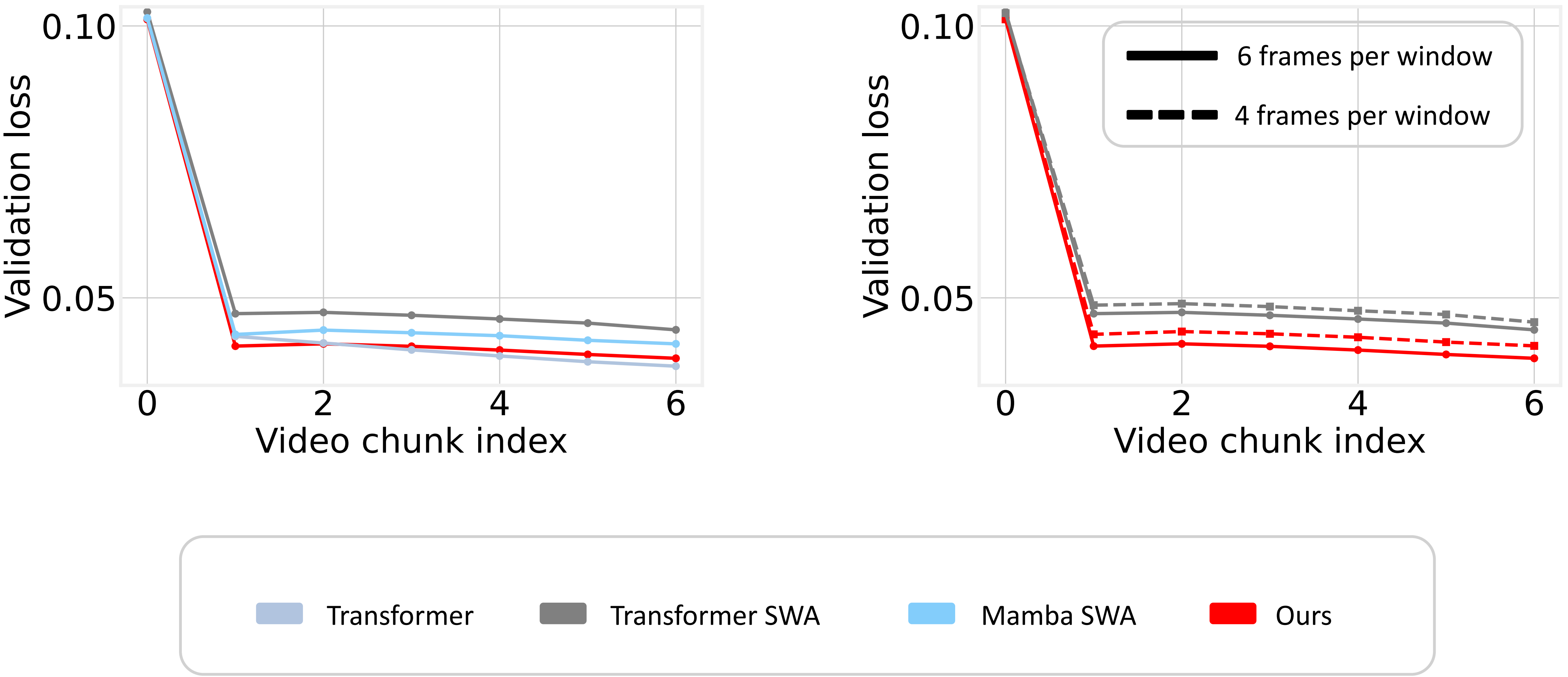
LaCT for AR video diffusion



AR video experiment setup

- Finetune a bidirectional video model to AR video model
  - Wan T2V: small: 1.3B, big:14B
- Finetune for 5k iterations on internal text-video dataset
- Measure validation loss at different frame chunks

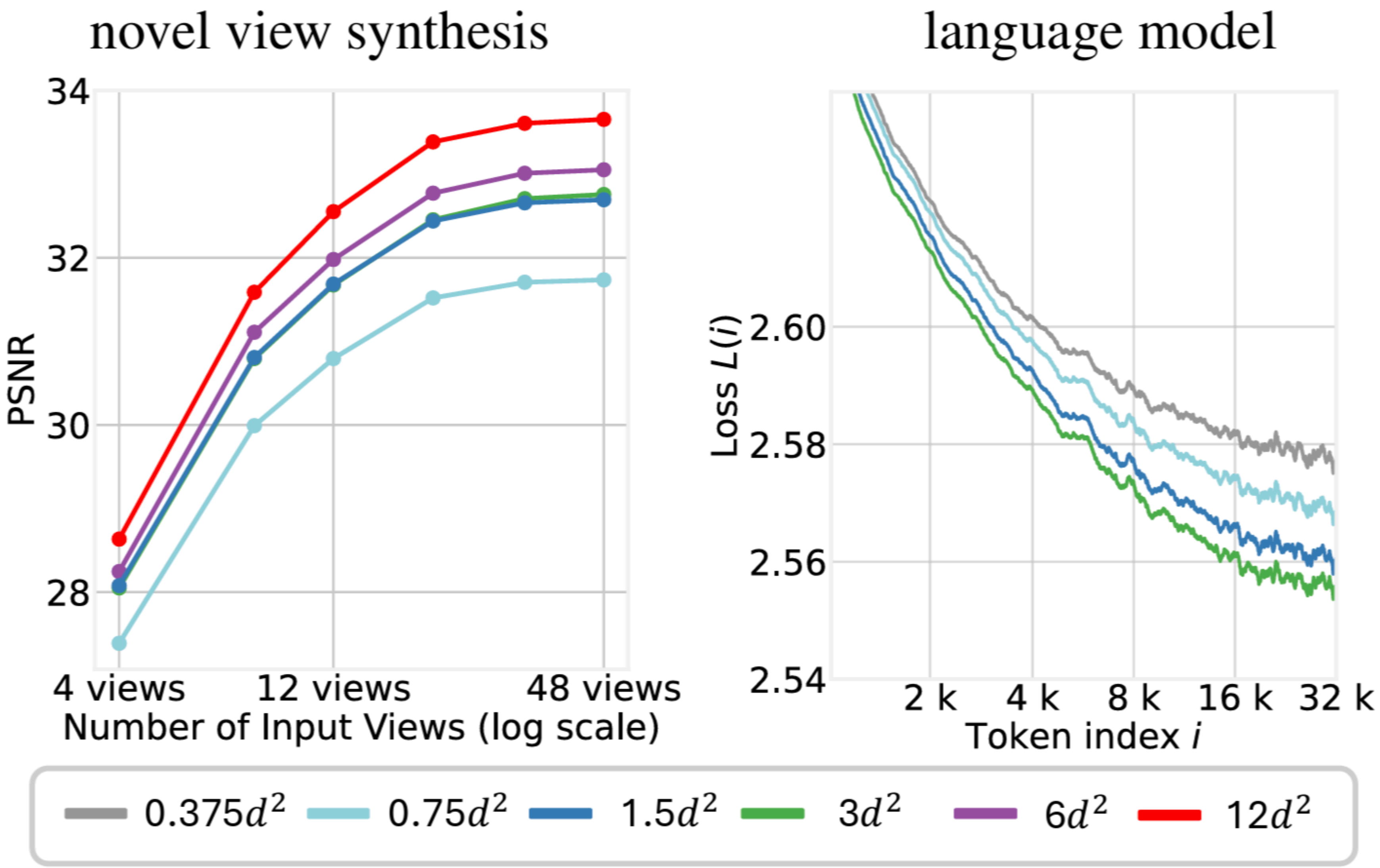
Video results



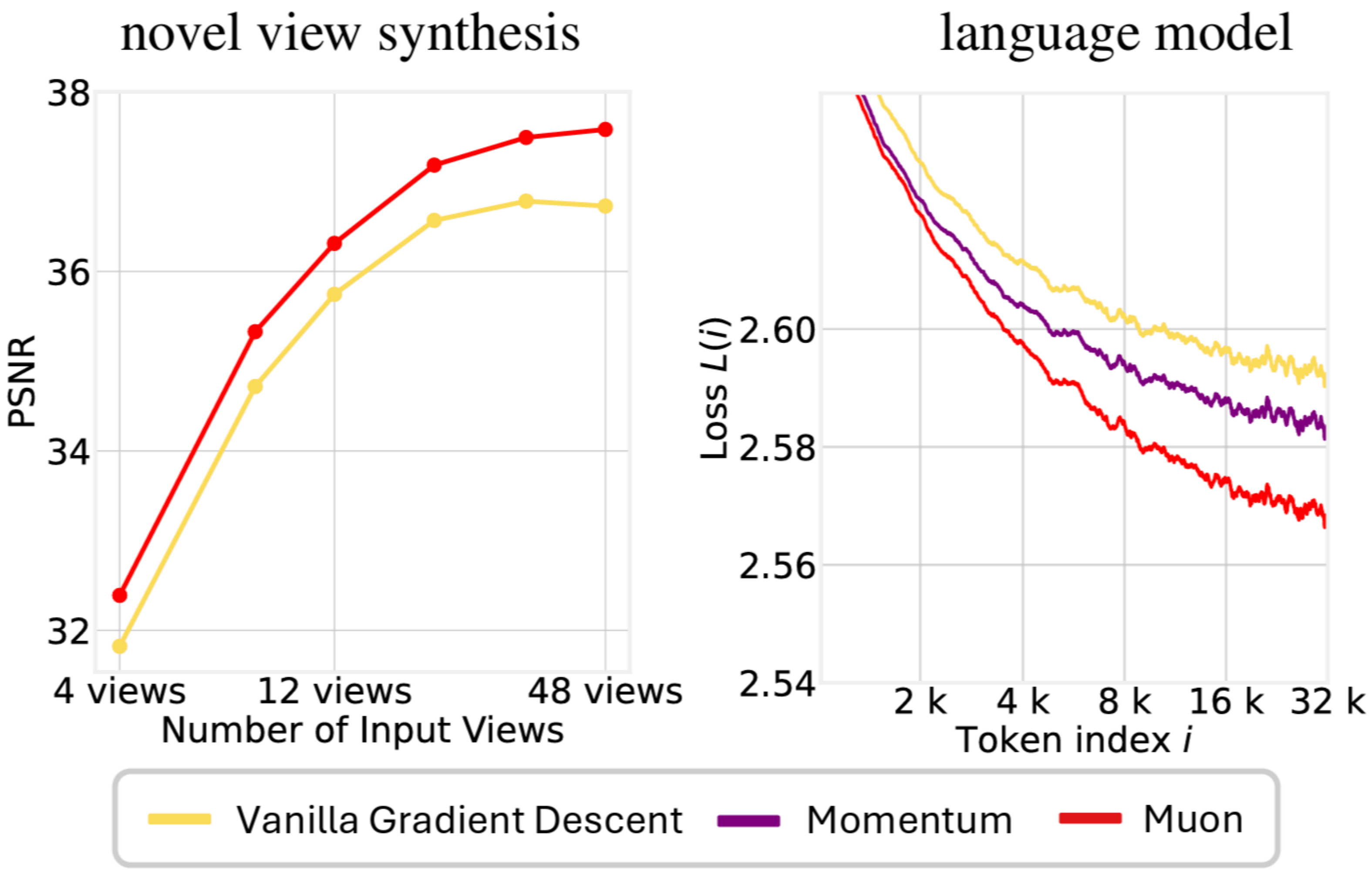
Interesting studies

- State Size Scaling
- Different optimizers
- Chunk-recurrence v.s. per-token recurrence
- Linear v.s. NonLinear fast weight function

State Size Scaling



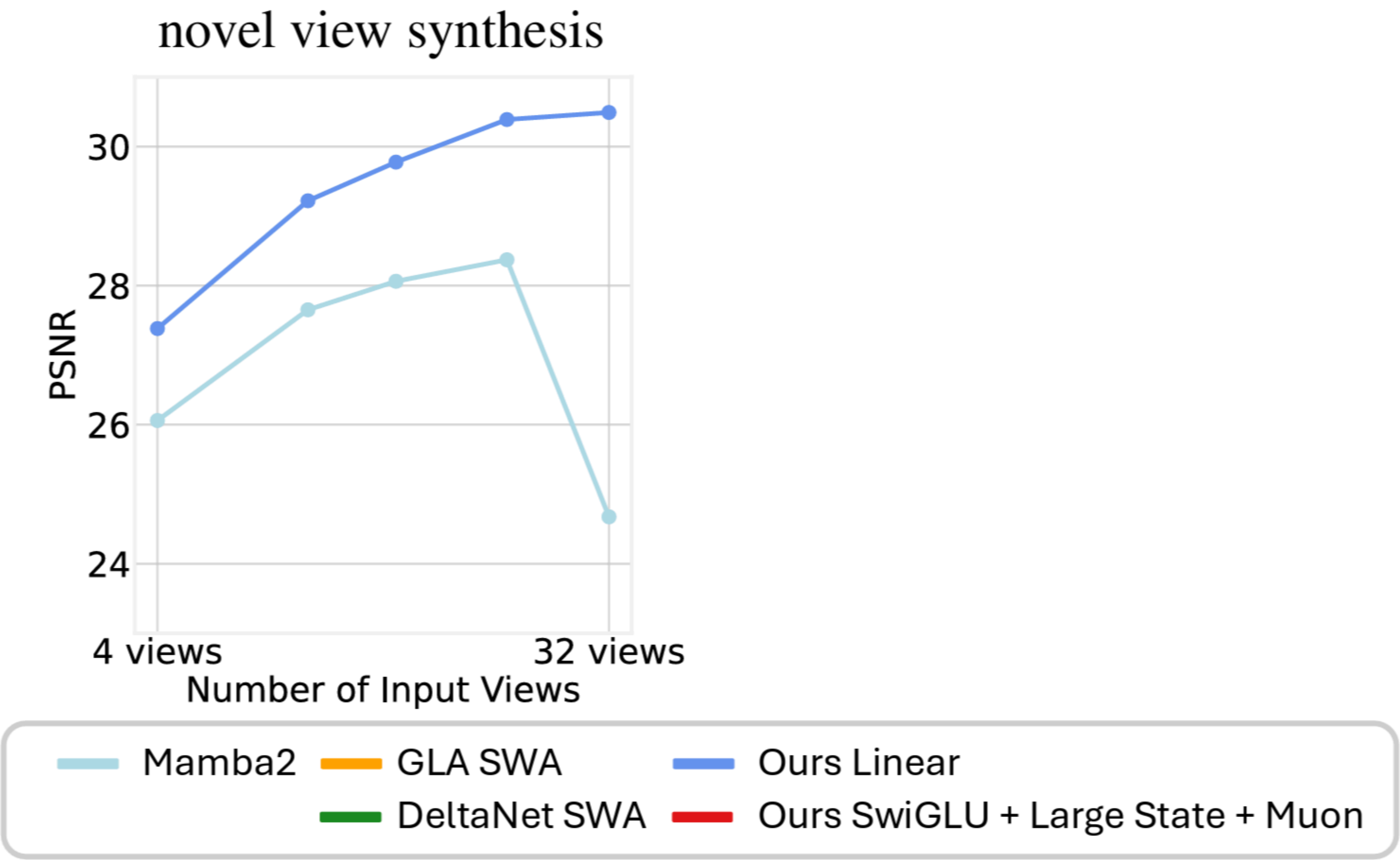
Different test-time training optimizers



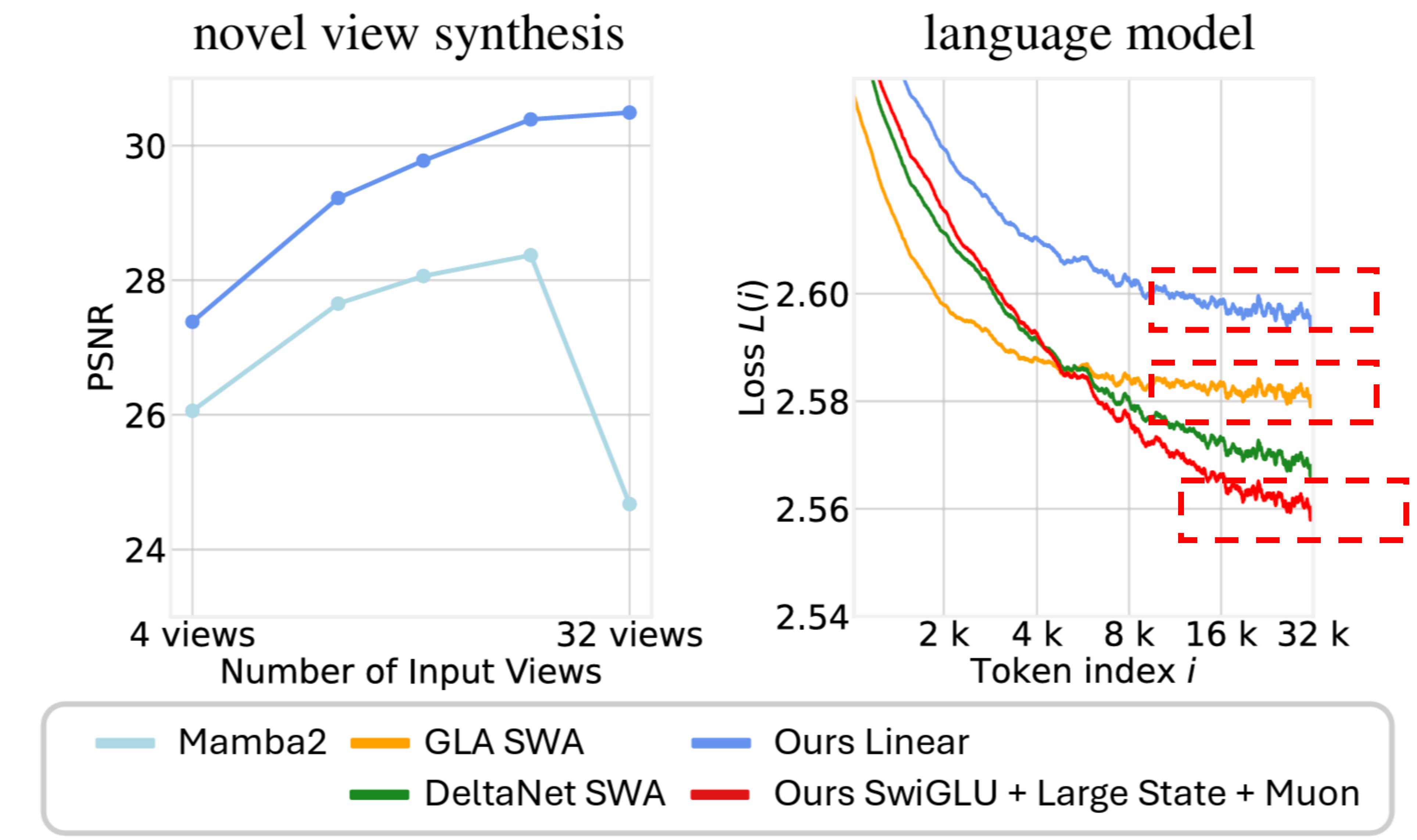
Chunk-recurrence v.s. token recurrence

- Trading depth-of-recursion for parallelism

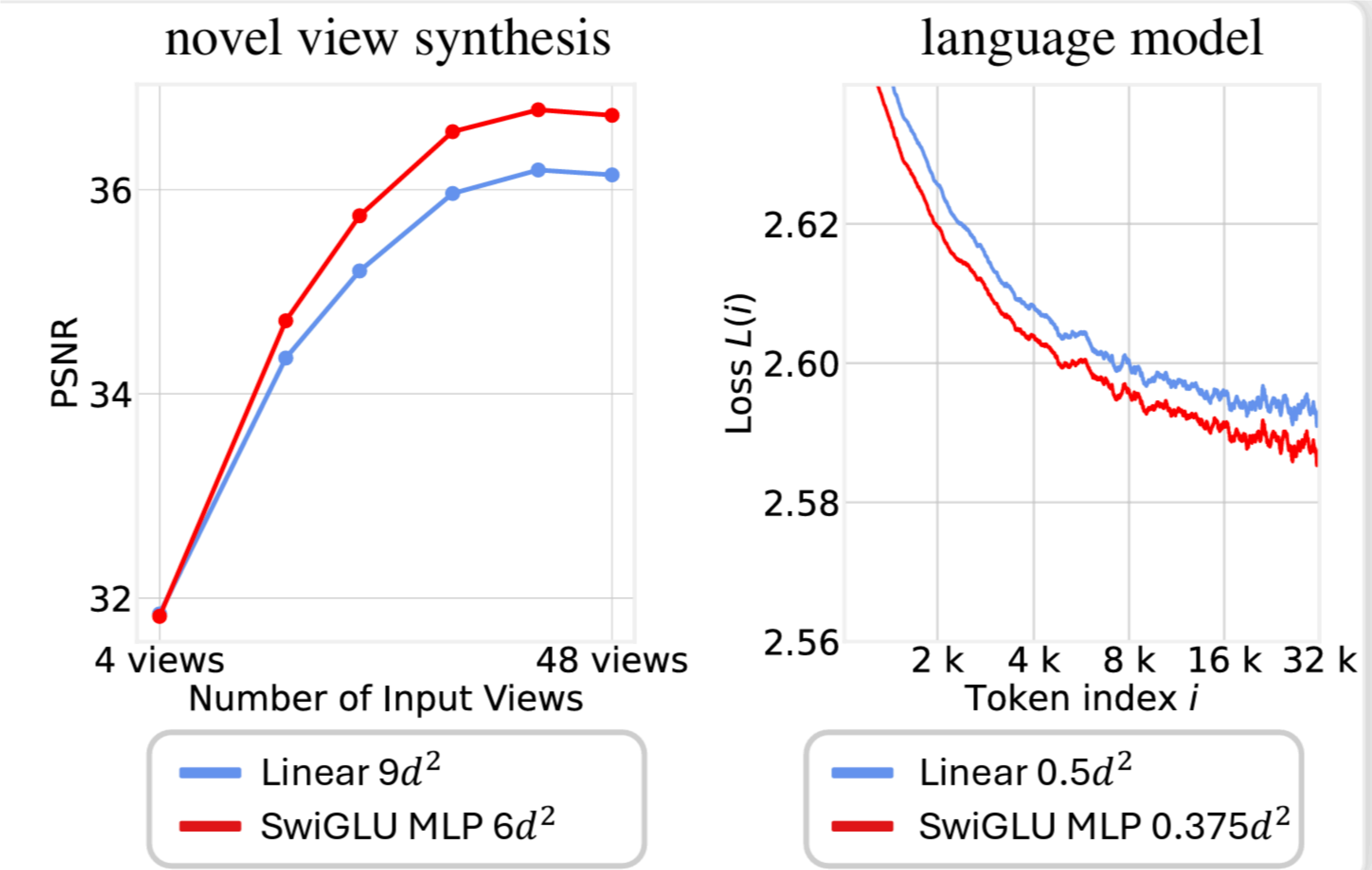
Chunk-recurrence v.s. token recurrence



Chunk-recurrence v.s. token recurrence



Linear Memory v.s. NonLinear Memory



Summary

- Large chunk-size TTT boost GPU utilization by 10x
- No kernel code => much faster research exploration
- Using TTT for long memory, using window attention for local memory