

# **E.T.: Re-Thinking Self-Attention for Transformer Models on GPUs**

2024.03.28

# E.T.

- System/Algorithm optimizations for transformer model inference

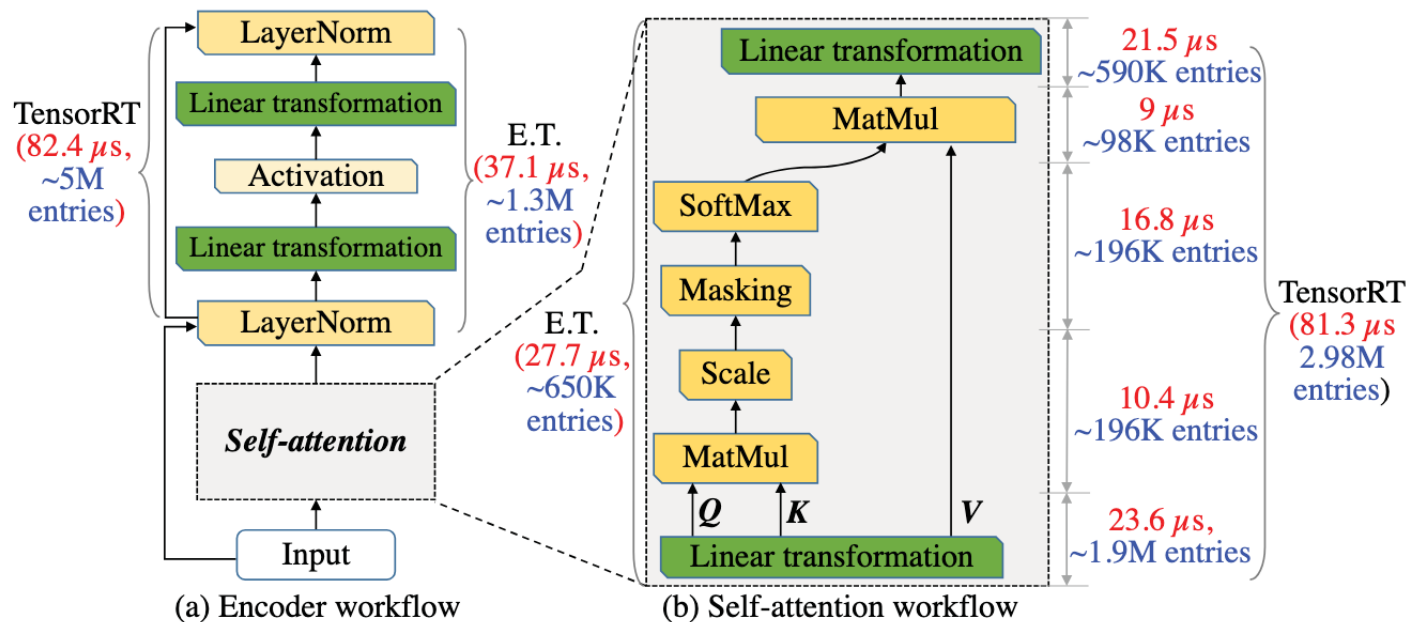
# E.T.

- System/Algorithm optimizations for transformer model inference
  - A novel **self-attention architecture** with new operators
  - With **tensor-tile pruning** algorithms and **attention-aware pruning**

# E.T.

- Kernel optimizations
  - Kernel fusion
  - Tailored attention
  - Incremental/decode/generation phase: maximize thread occupancy and minimize on-device high-bandwidth memory (HBM) access (i.e., using shared memory or registers)

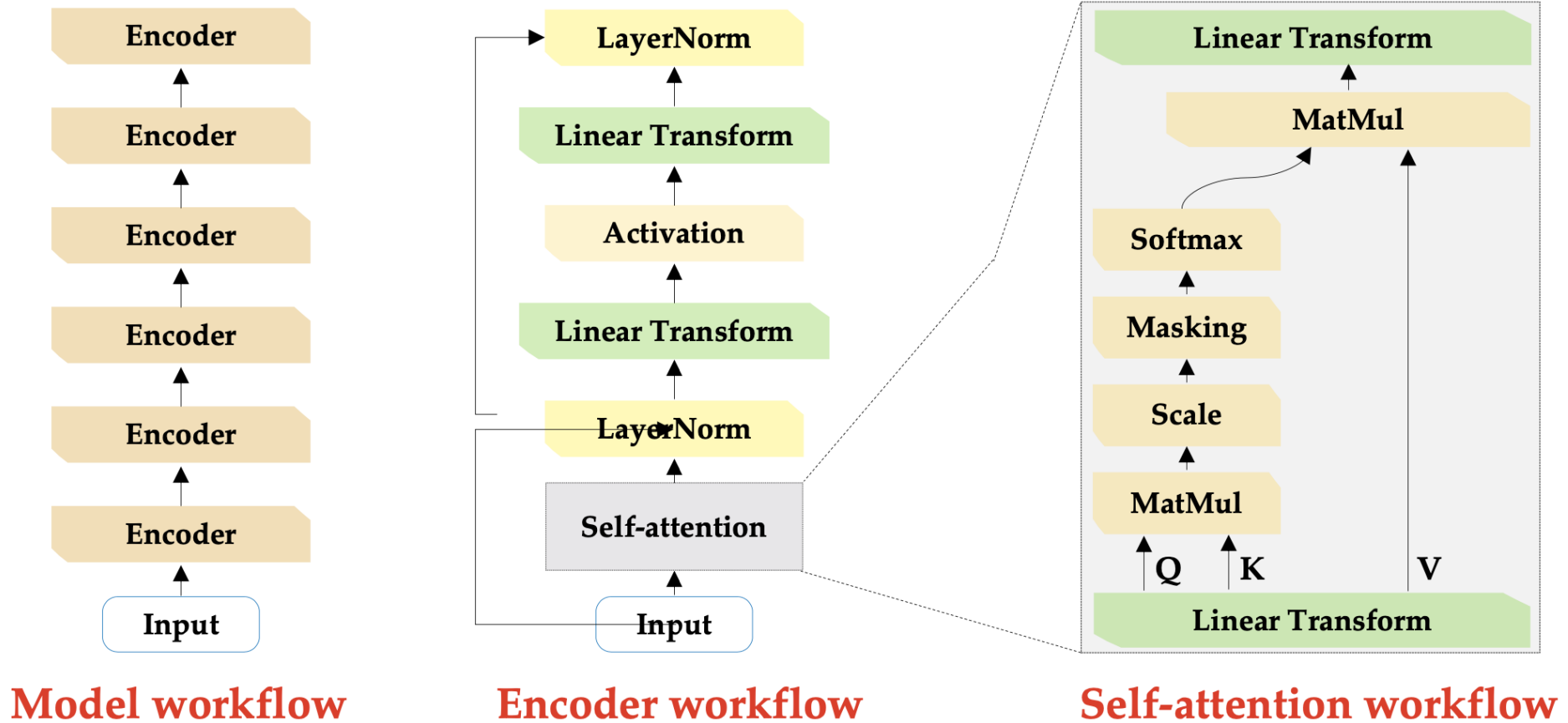
# E.T.



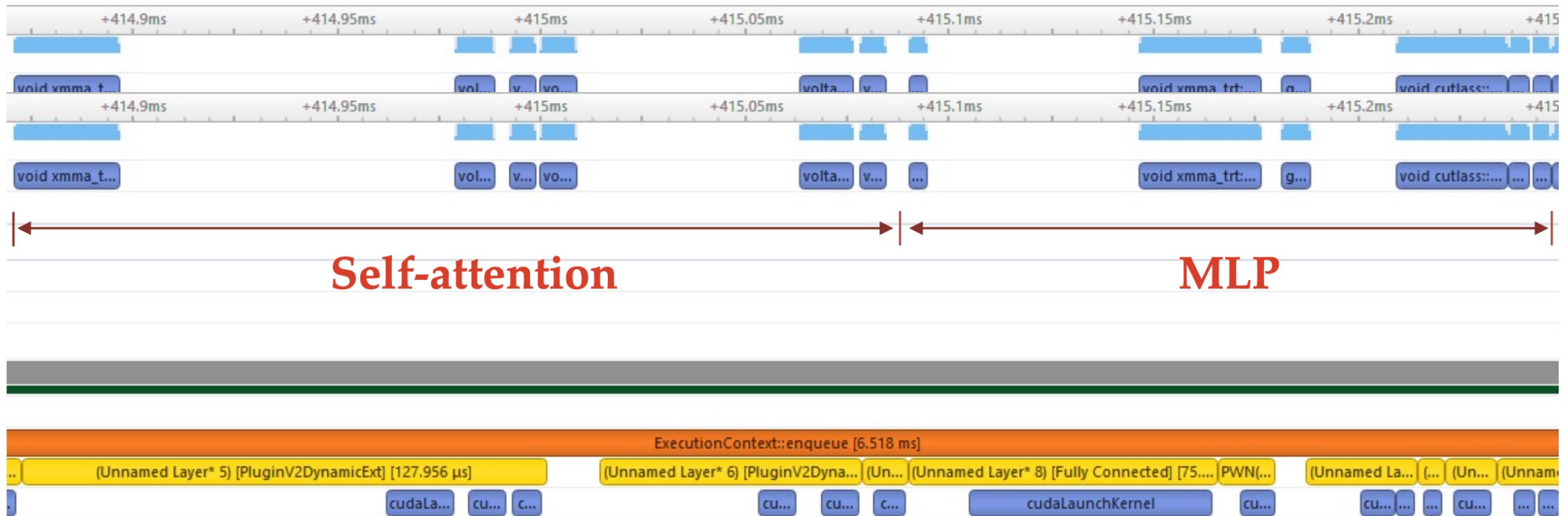
- 2.5x speedup compared with TensorRT

**Figure 1: The architecture of a four-head encoder. The time consumption is measured on WikiText-2 dataset [30], where the input sequence has 128 tokens. Our pruning ratio is 80%.**

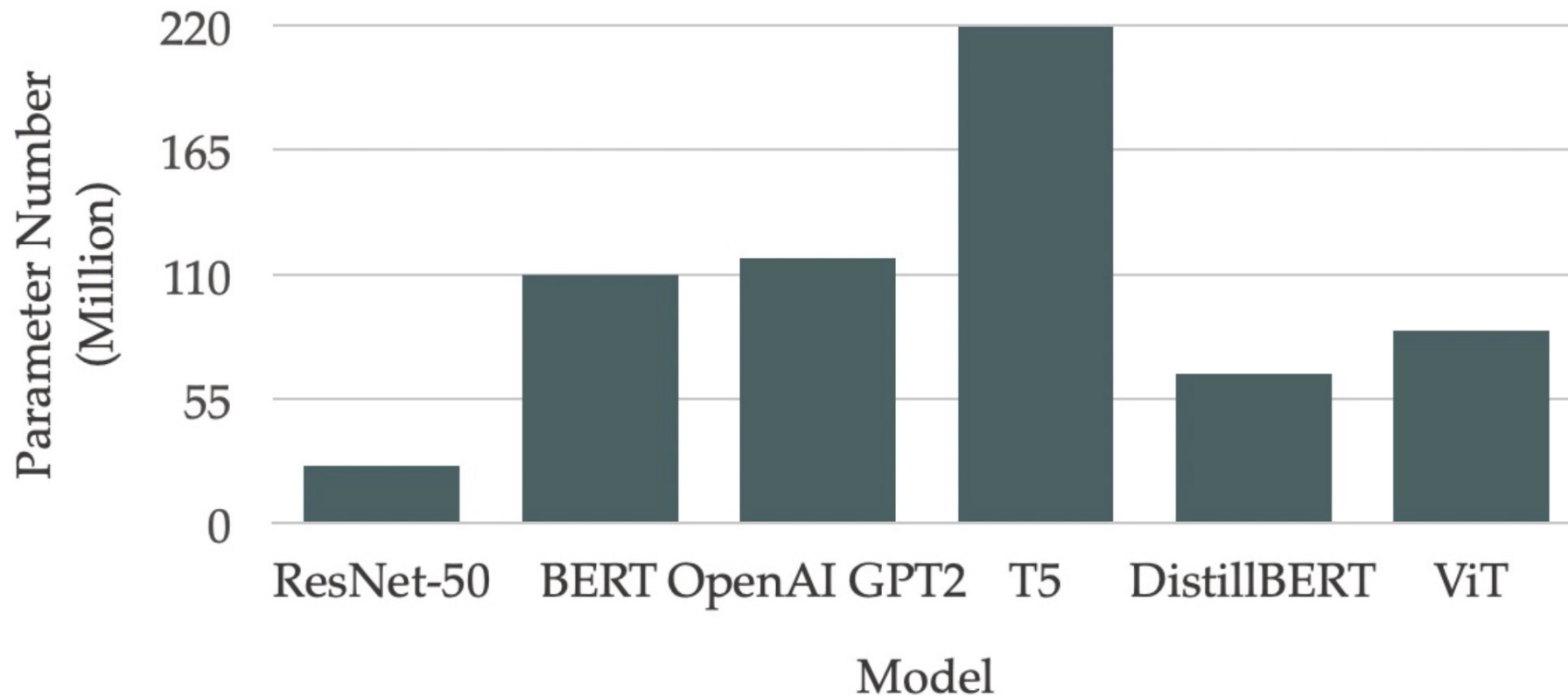
# Challenge #1: Long Turnaround Time



# Challenge #1: Long Turnaround Time

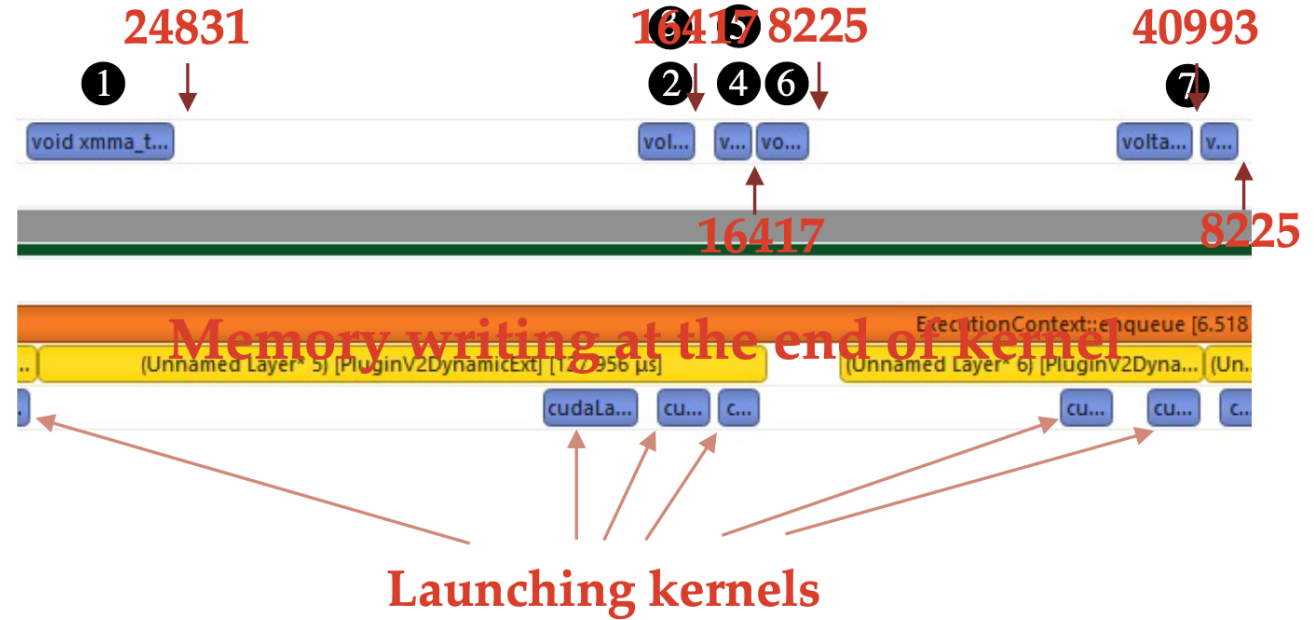
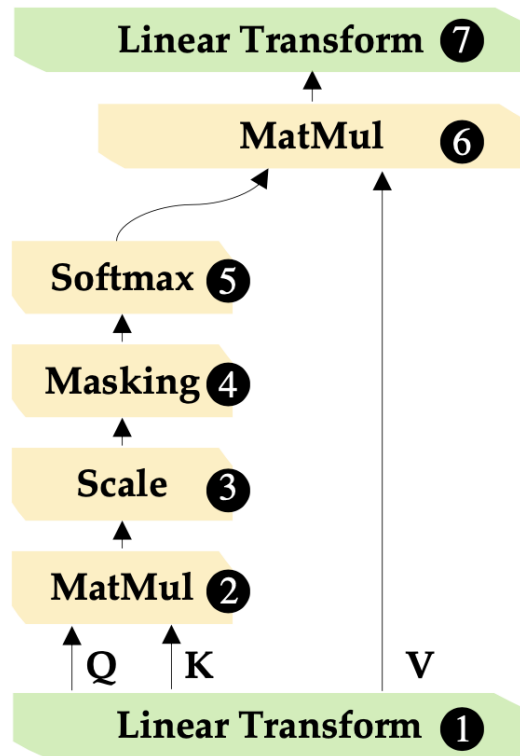


# Challenge #2: Gigantic model size

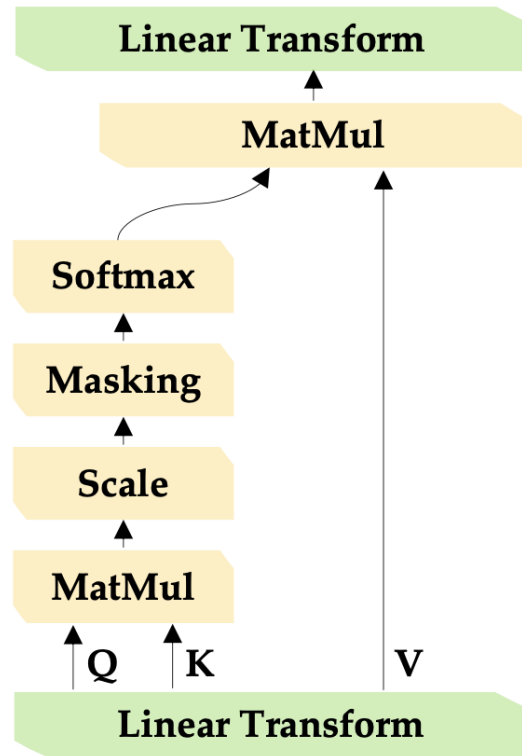




# Kernels are not free



# Think self-attention as a primitive



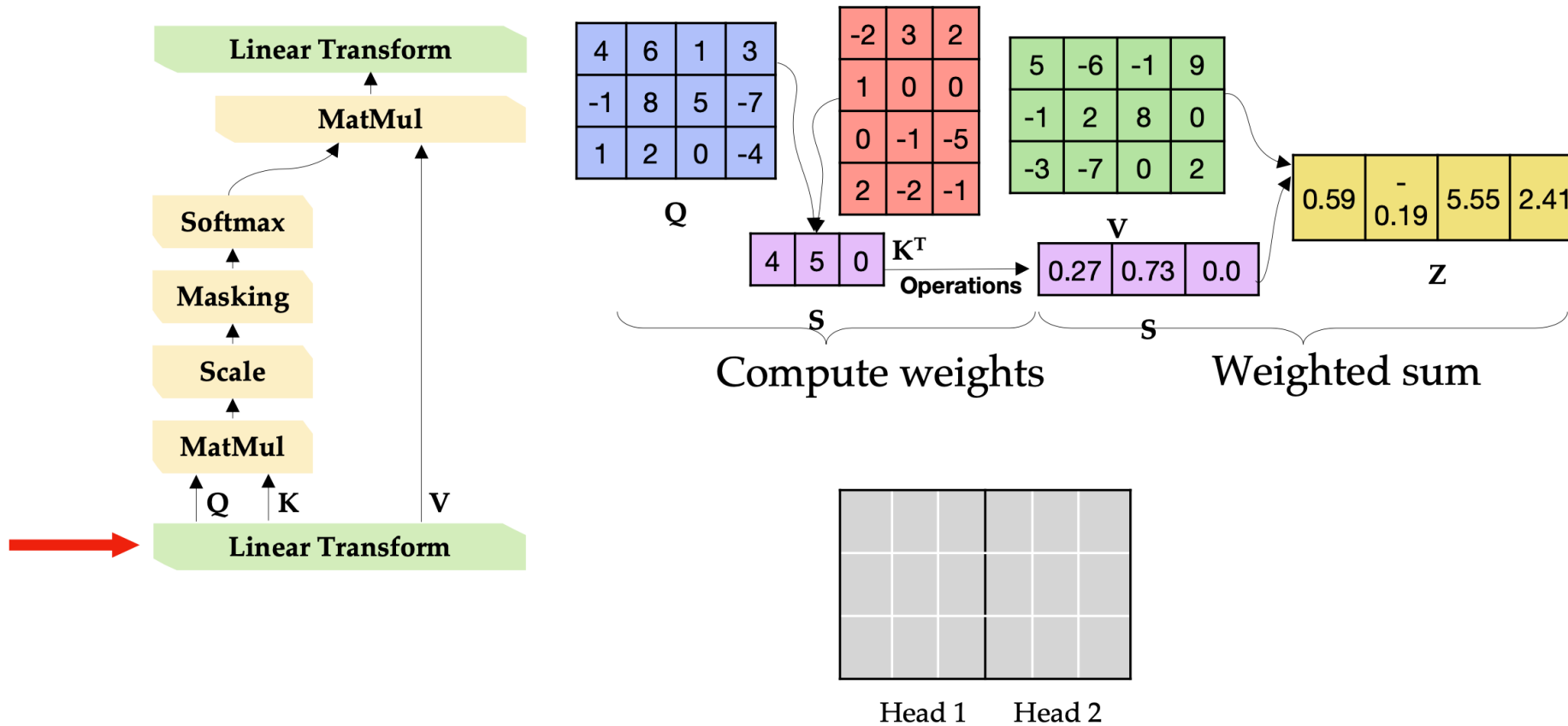
How are you



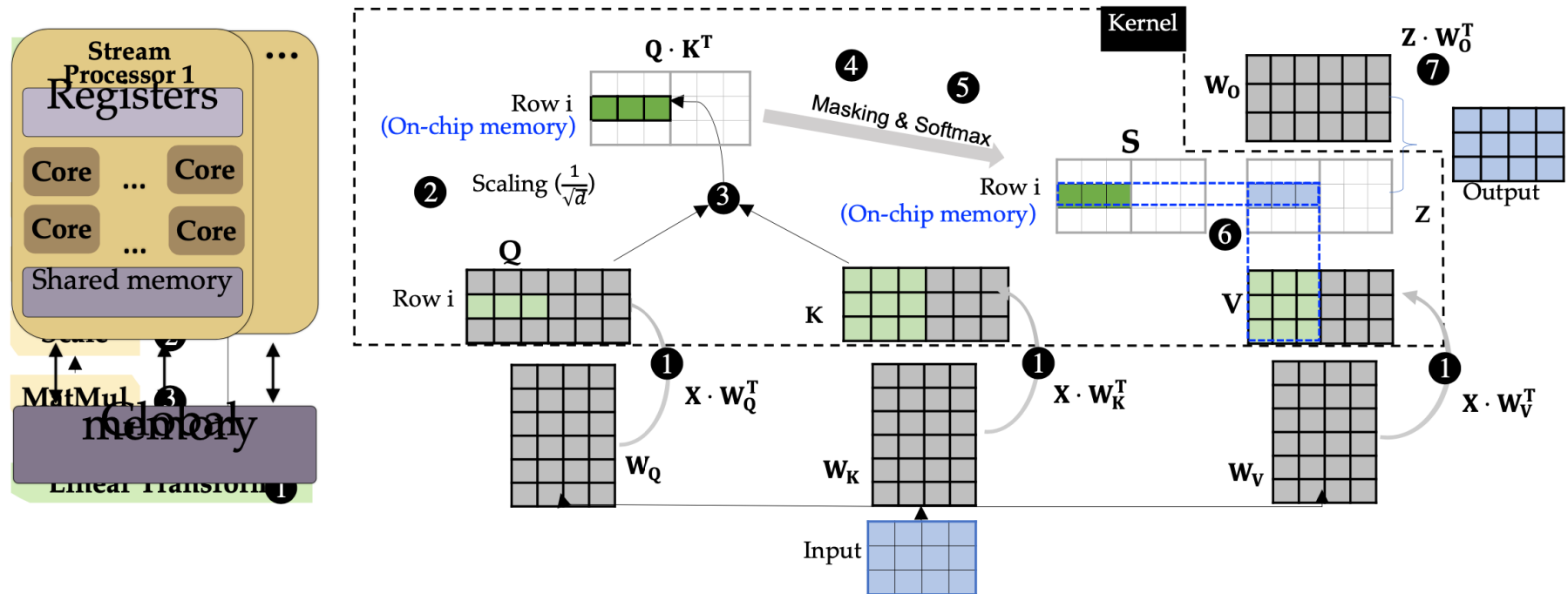
how	5	2	1	4
are	-3	-1	0	0
you	1	-2	-1	-1

**Input**

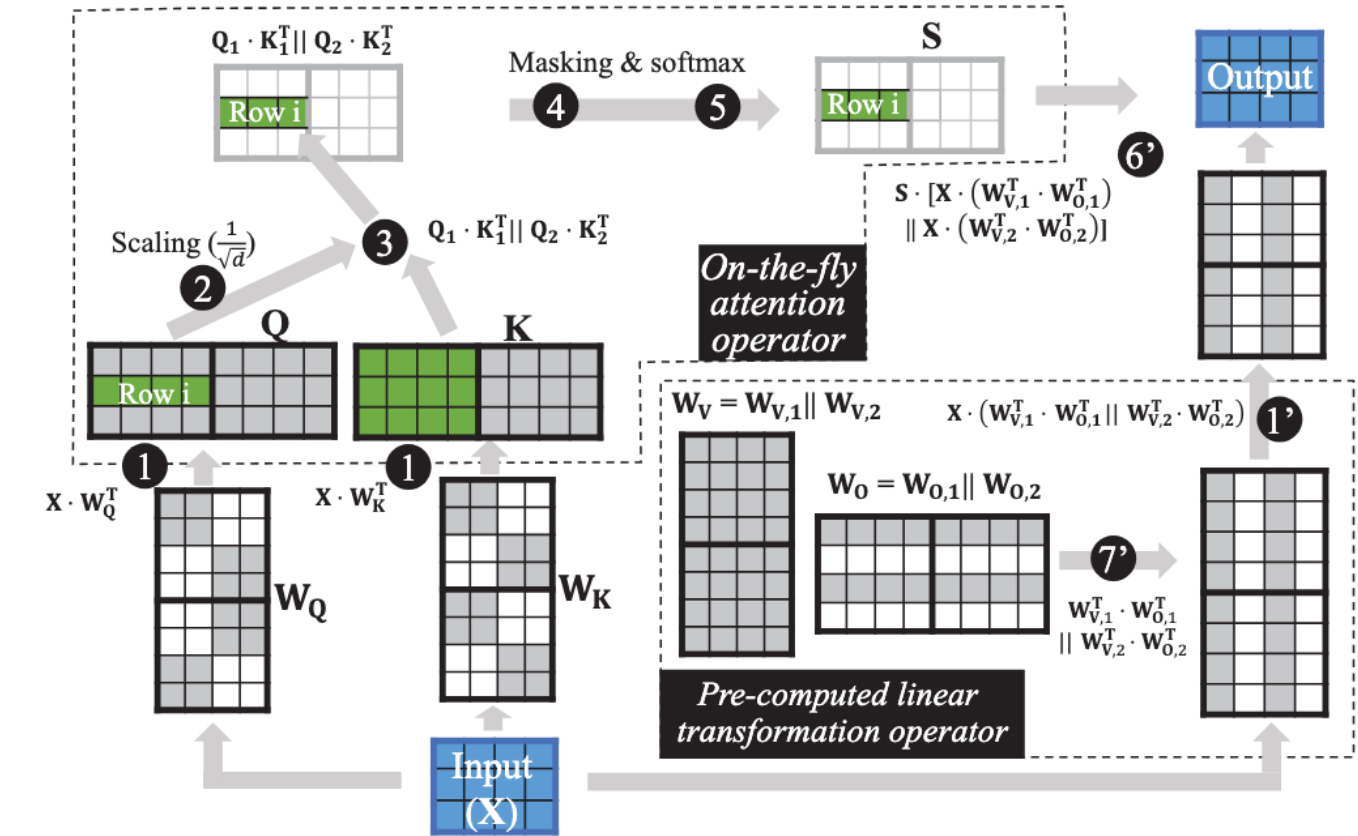
# Think self-attention as a primitive



# Compute the self-attention on-the-fly



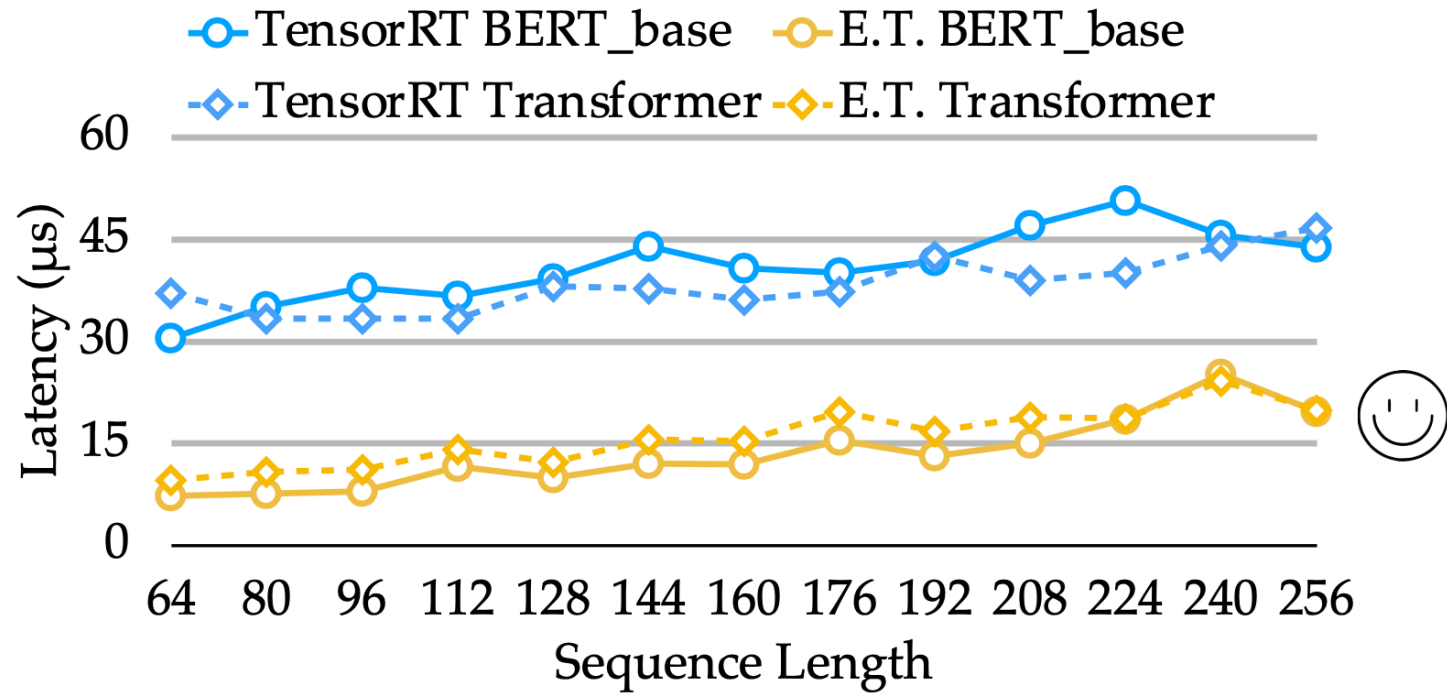
# Pre-computed linear transformer operator



$$\begin{aligned} \text{Output} &= \sum_{h=1}^H (\mathbf{S}_h \cdot \mathbf{V}_h) \cdot \mathbf{W}_{O,h}^T = \sum_{h=1}^H \mathbf{S}_h \cdot (\mathbf{V}_h \cdot \mathbf{W}_{O,h}^T) \\ &= \sum_{h=1}^H \mathbf{S}_h \cdot (\mathbf{X} \cdot \mathbf{W}_{V,h}^T \cdot \mathbf{W}_{O,h}^T) = \sum_{h=1}^H \mathbf{S}_h \cdot \boxed{\mathbf{X} \cdot (\mathbf{W}_{V,h}^T \cdot \mathbf{W}_{O,h}^T)}. \end{aligned}$$

ator. (b) On-the-fly attention operator w/ pre-computed linear transformation operator.

# Evaluate on-the-fly attention



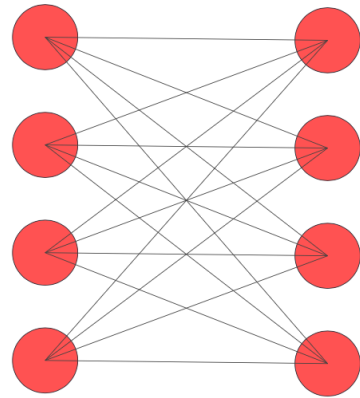
## BERT\_base:

- Model Size: 768
- Number of heads: 12

## Transformer:

- Model Size: 800
- Number of heads: 4

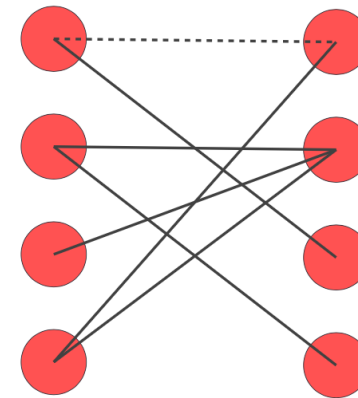
# Pruning make the model small



Dense

4	3	6	9
8	7	6	2
4	8	3	2
5	2	4	9

Pruning



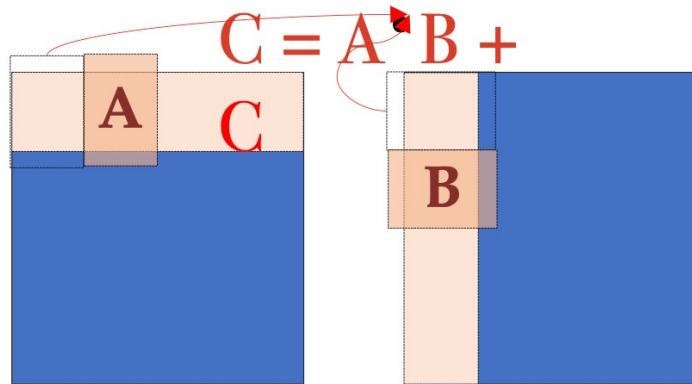
Sparse

0	0	6	0
0	7	0	2
0	8	0	0
5	2	0	0

Pruning



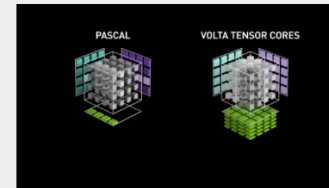
# Using emerging hardware



## All-New Matrix Core Technology for HPC and AI

Powered by the all-new Matrix Core technology, this powerful engine delivers nearly 3.5x performance boost for HPC (FP32 matrix) and nearly 7x for AI (FP16) workloads compared to the prior generation AMD data center GPU.<sup>2</sup>

- All-New FP32 and FP16 Matrix Core Technology
- BFloat16 operations for AI
- Enhanced performance



## VOLTA TENSOR CORES

### First Generation

Designed specifically for deep learning, the first-generation Tensor Cores in NVIDIA Volta™ deliver groundbreaking performance with mixed-precision matrix multiply in FP16 and FP32—up to 12X higher peak teraFLOPS (TFLOPS) for training and 6X higher peak TFLOPS for inference over NVIDIA Pascal. This key capability enables Volta to deliver 3X performance speedups in training and inference over Pascal.

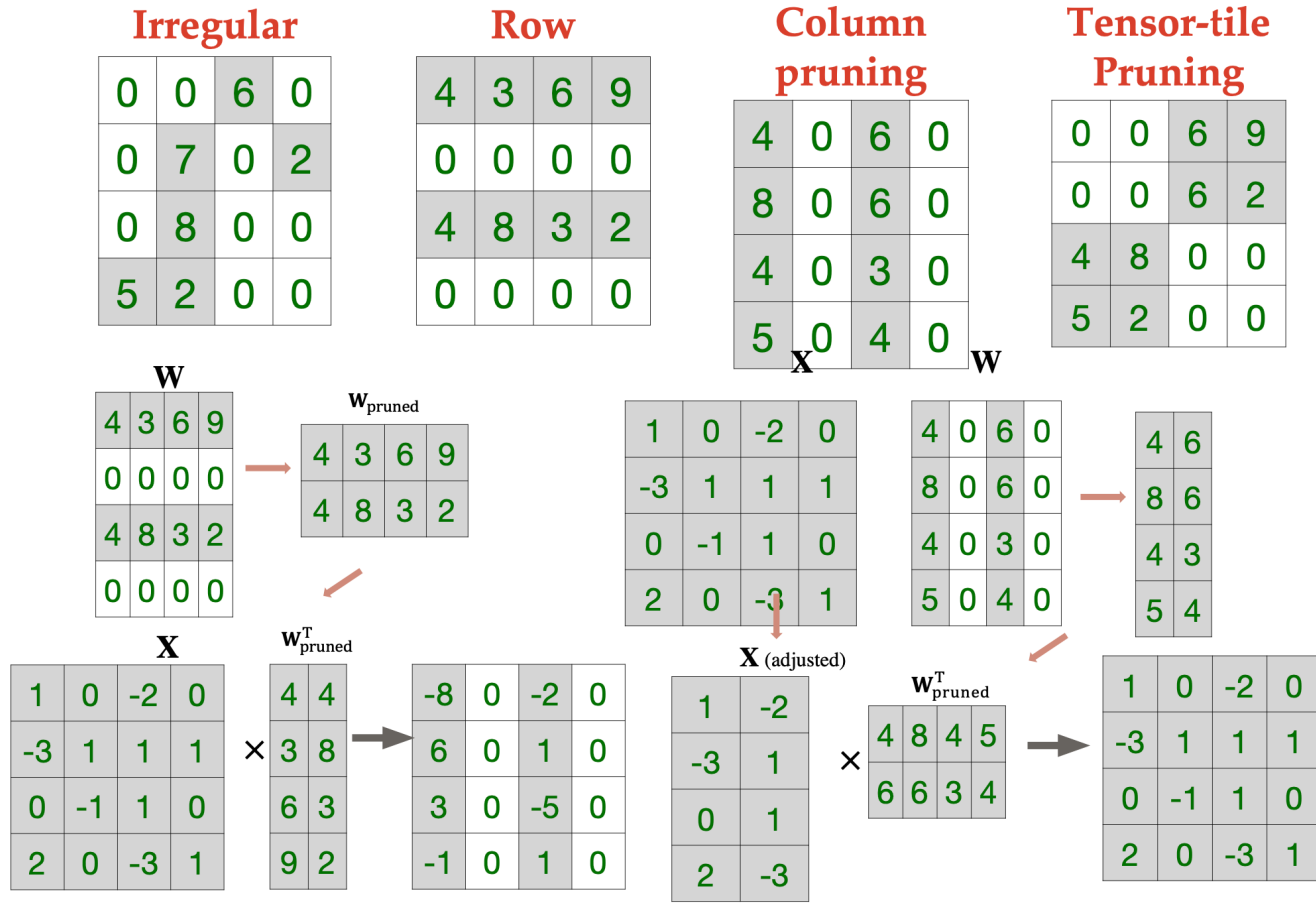
[LEARN MORE ABOUT VOLTA >](#)

[1]. <https://www.amd.com/en/technologies/cdna>

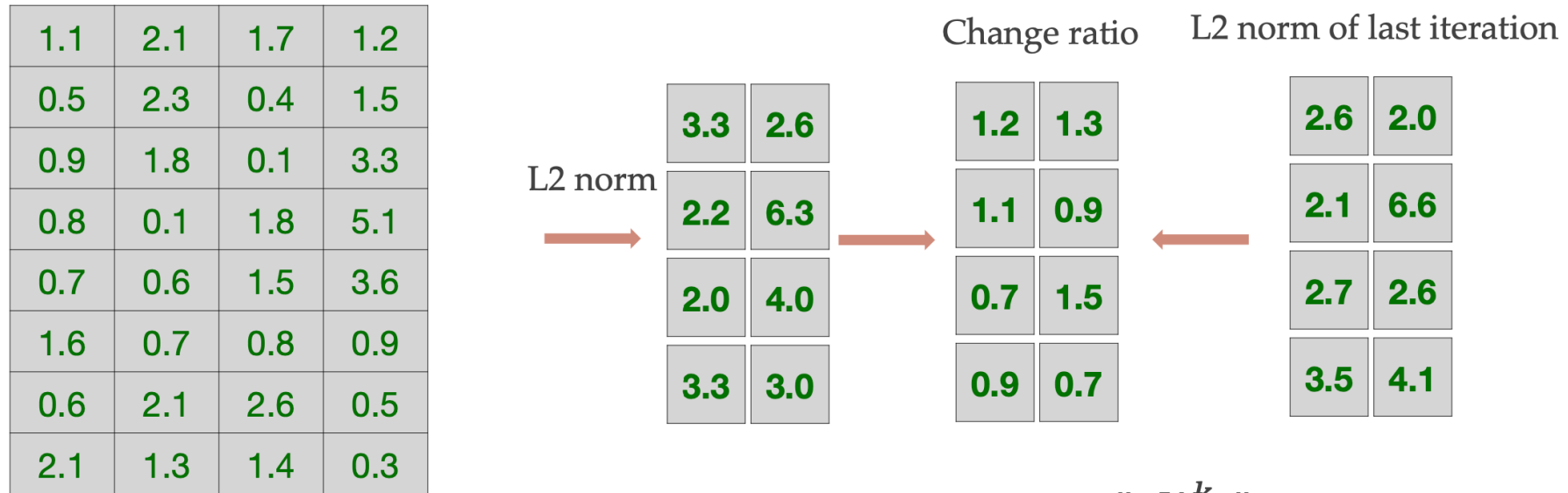
[2]. <https://www.nvidia.com/en-us/data-center/tensor-cores>



# Efficient computing on sparse models



# Prune the model as fine-tuning



$$\min_f (\underbrace{\{W^k\}_{k=1}^N, \{b^k\}_{k=1}^N}_{\text{Original loss}}) + \lambda \underbrace{\sum_{k=1}^N \sum_{i=1}^p \sum_{j=1}^q \frac{\|W_{ij}^k\|_2}{\|W_{ij}^{k-1}\|_2 + \epsilon}}_{\text{Regularizer}}$$

Original loss

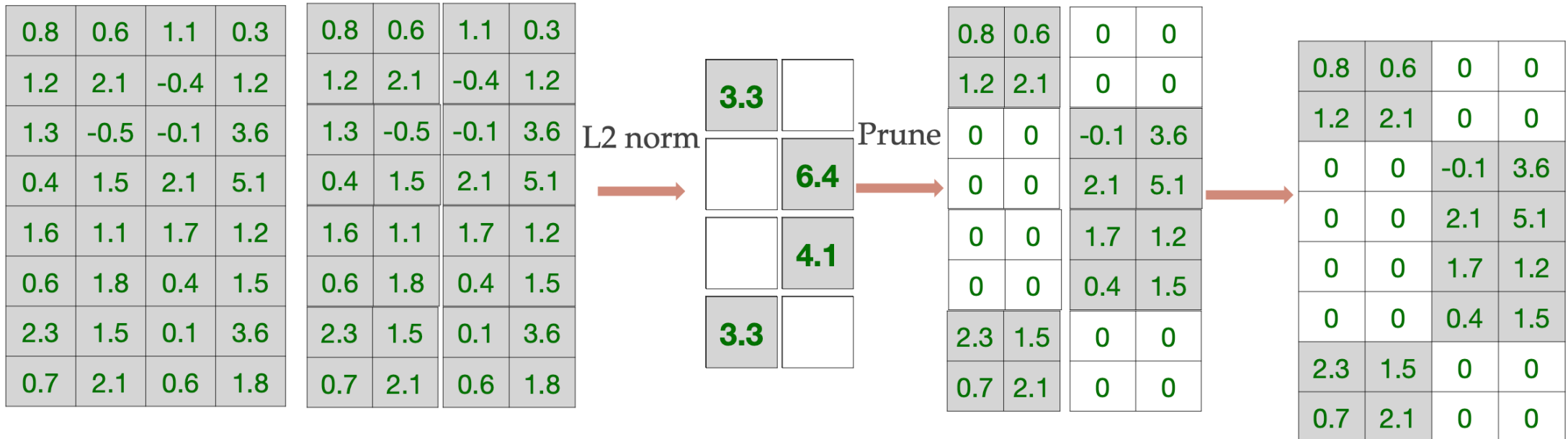
Regularizer

# Prune the model as fine-tuning

- Initialization: start with pre-trained model
- Check the current epoch falls into the pre-defined milestones. If yes,
  - Divide weight matrix
  - Compute l2-norm of each tile and update tile penalty factor
  - Update the model loss
  - Train Transformer model
  - **Prune weights based on L2 norm**
  - Retrain the non-zero entries for several epochs

$$\min f(\{\mathbf{W}^k\}_{k=1}^N, \{\mathbf{b}^k\}_{k=1}^N) + \lambda \sum_{k=1}^N \sum_{i=1}^p \sum_{j=1}^q \beta_{ij}^k \|\mathbf{W}_{ij}^k\|_2,$$

# Prune weights based on L2 norm



# Efficient computing on sparse models

**Irregular**

0	0	6	0
0	7	0	2
0	8	0	0
5	2	0	0

**Row**










4	3	6	9
0	0	0	0
4	8	3	2
0	0	0	0

**Column pruning**

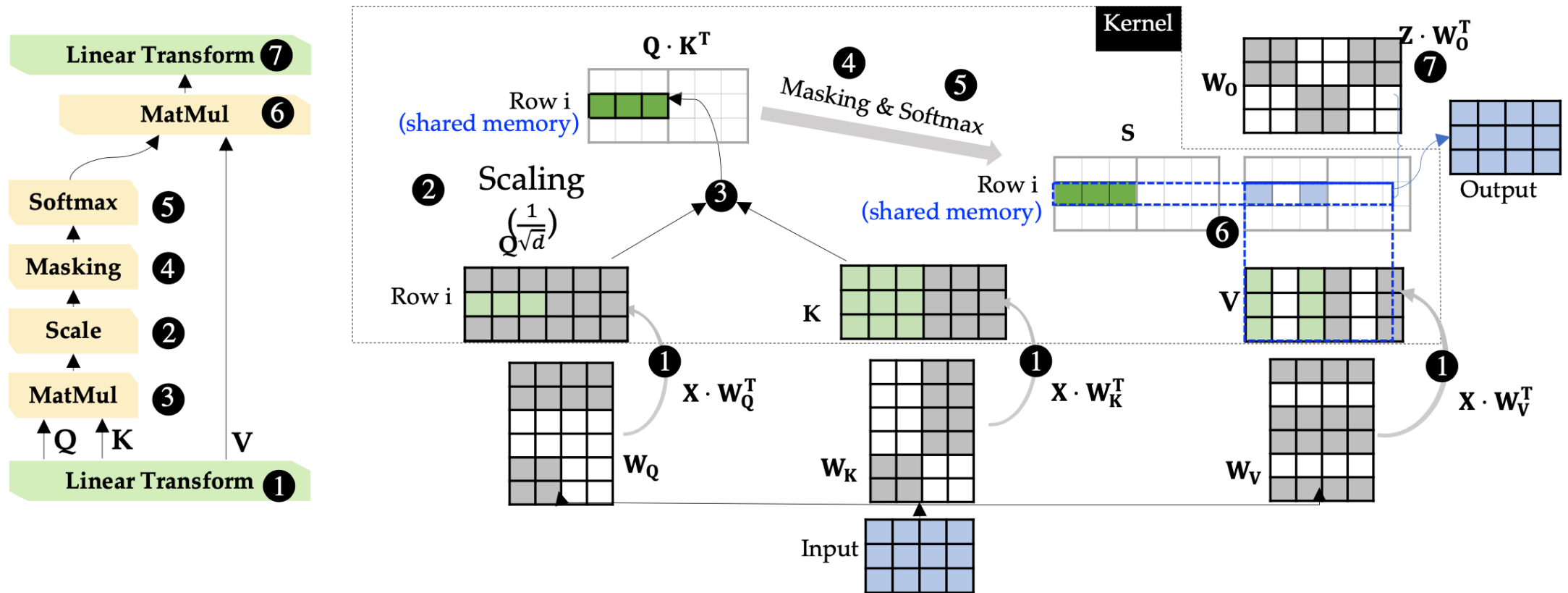
4	0	6	0
8	0	6	0
4	0	3	0
5	0	4	0

**Tensor-tile Pruning**

0	0	6	9
0	0	6	2
4	8	0	0
5	2	0	0

	Irregular	Row	Column	Tensor-tile 
<b>Accuracy</b>				
<b>Latency</b>				

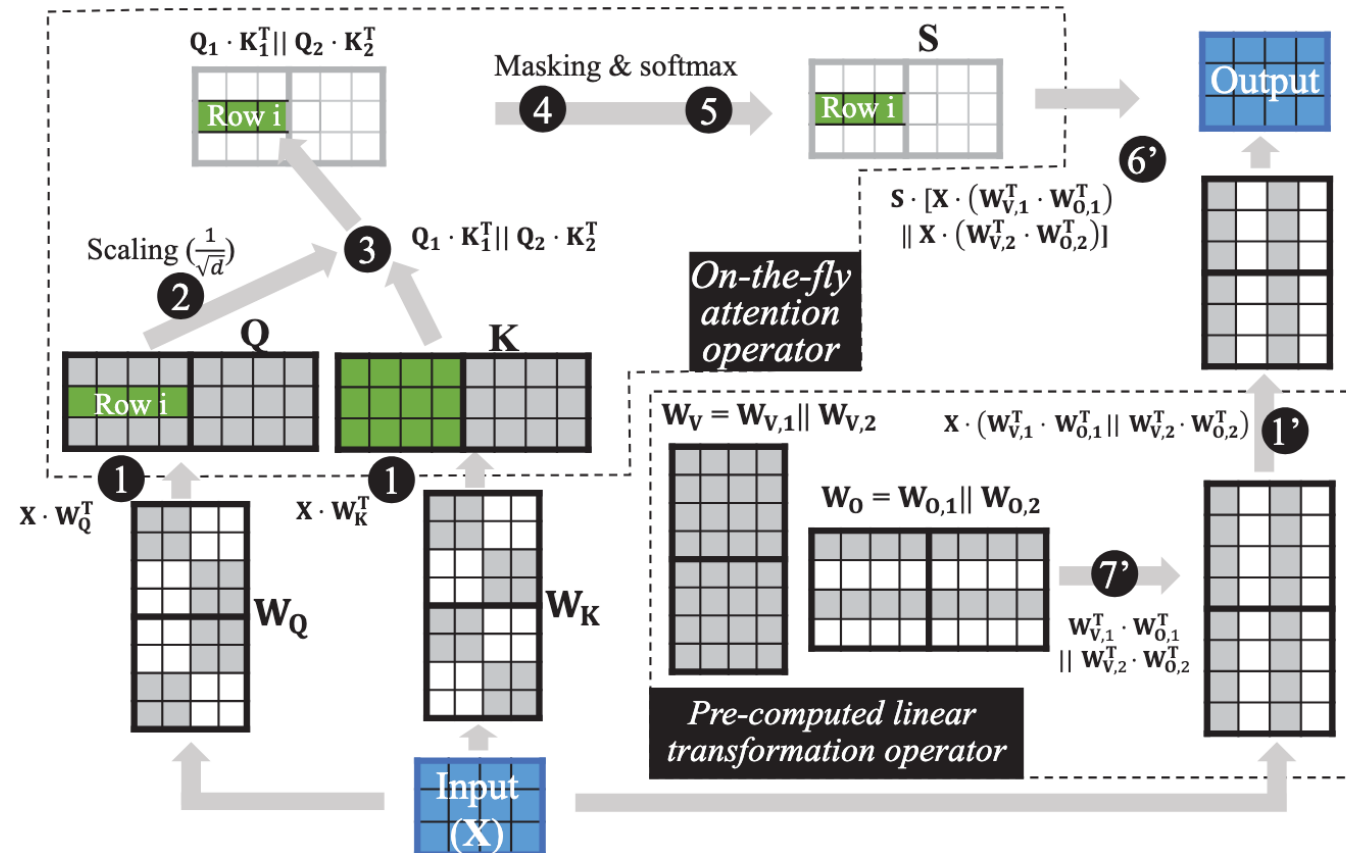
# Attention-aware pruning



$W_Q$  &  $W_K$ : Tile-based pruning

If not pre-computed linear transformation: column pruned  $W_V$ , tensor tile pruned  $W_O$

# Attention-aware pruning

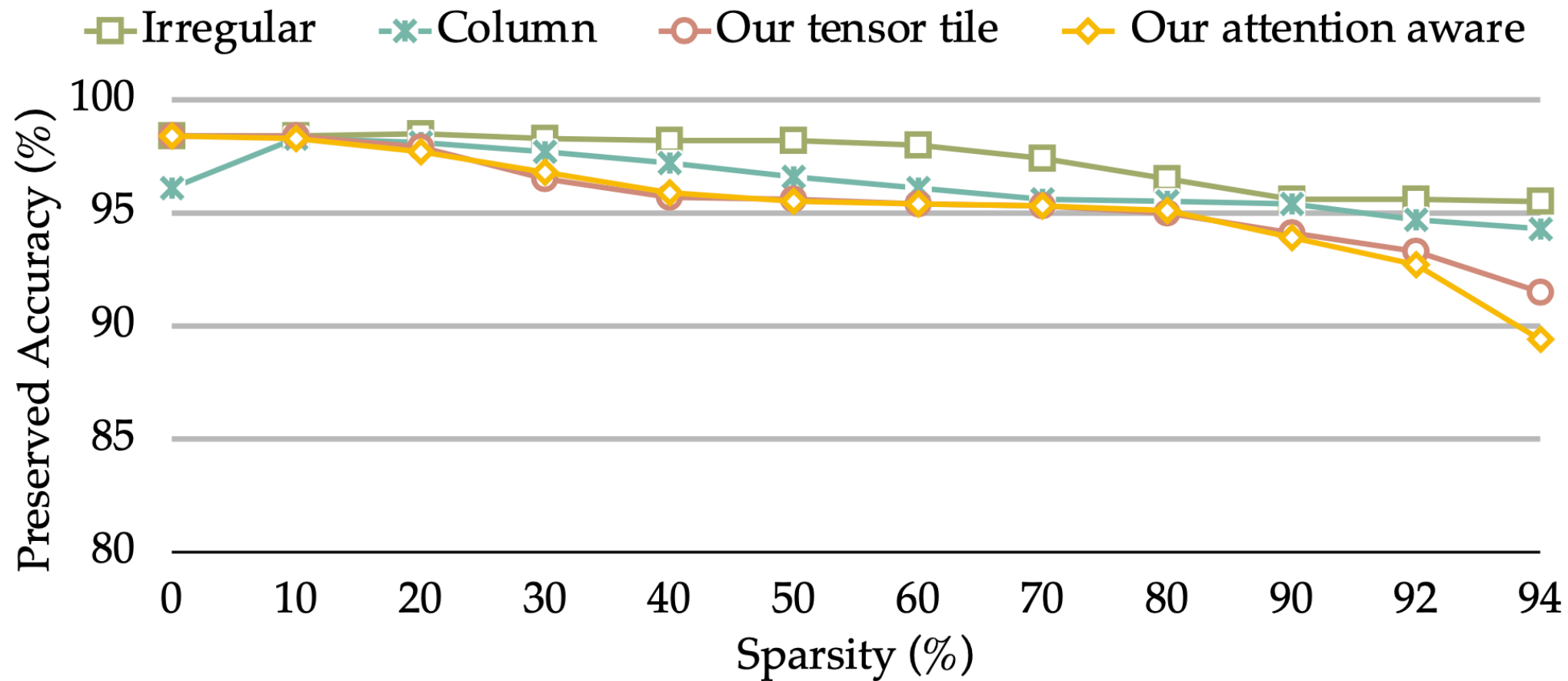


or. (b) On-the-fly attention operator w/ pre-computed linear transformation operator.

If pre-computed linear transformation:

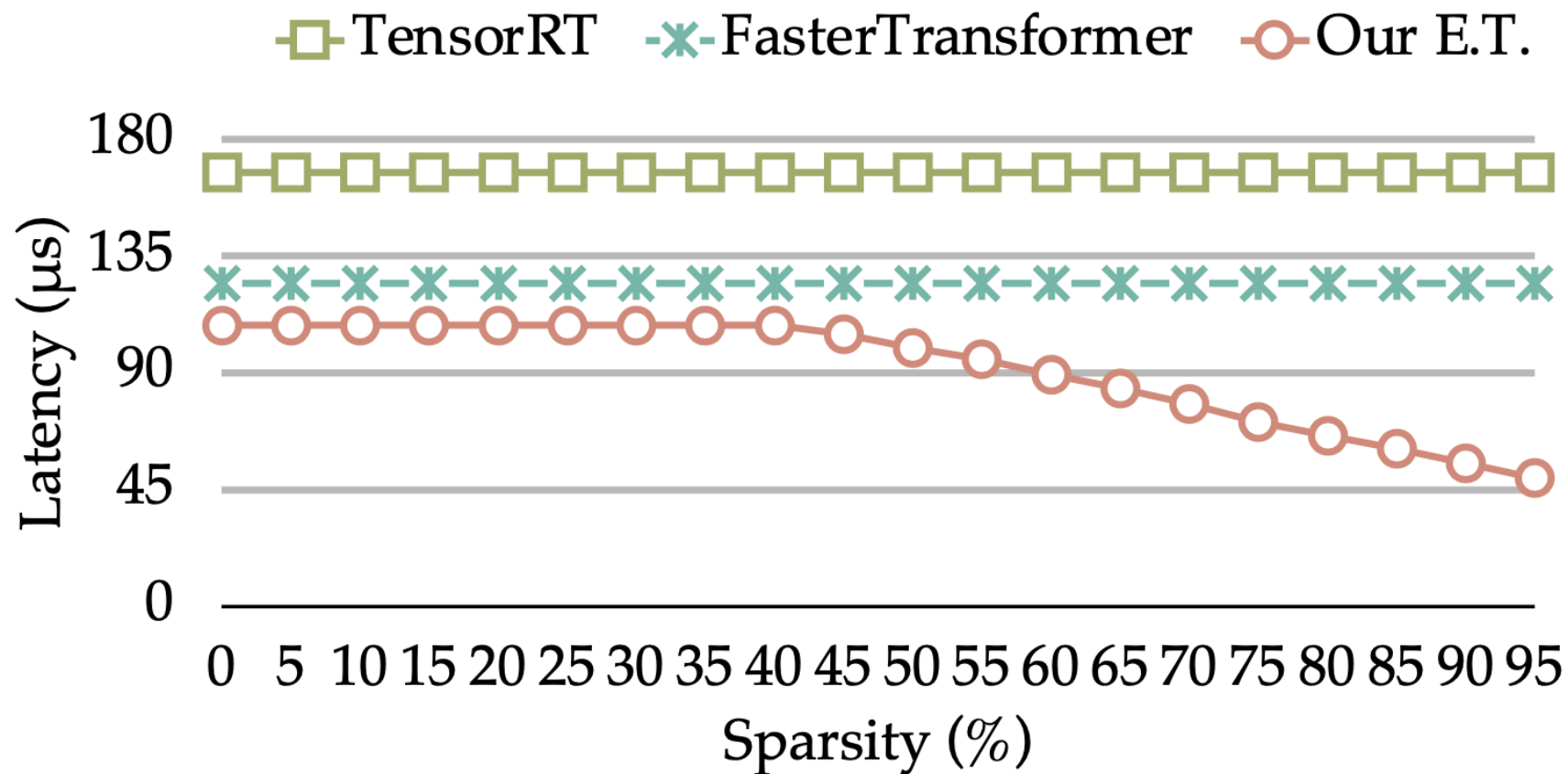
row pruned  $W_O$  (to benefit step 6), dense  $W_V$  (no sparsity changes from pruning, prevent pruning others)

# Evaluate pruning algorithms





# Compare with state-of-the-art



# Discussion

- V100

→ Other hardware platforms

- Inference

→ Training

- Combined with other kernel optimizations

# References

- <https://dl.acm.org/doi/10.1145/3458817.3476138>