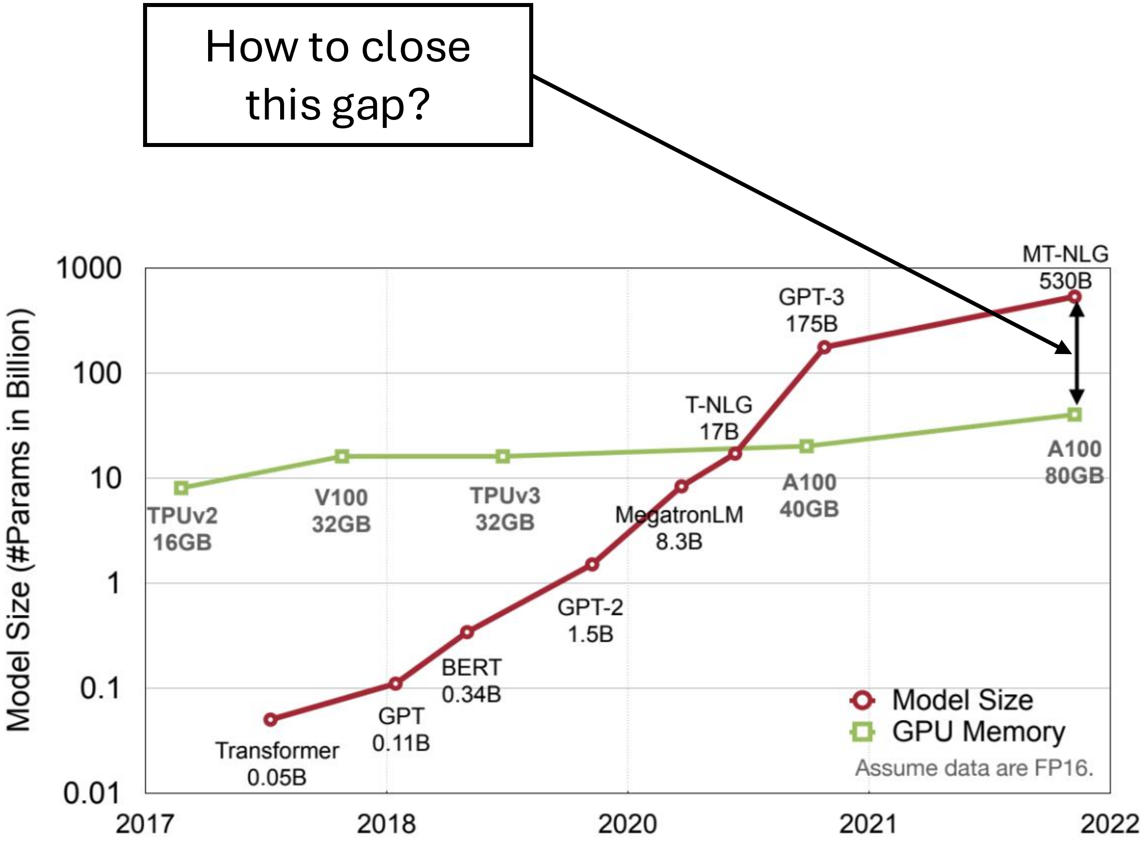
Acceleration

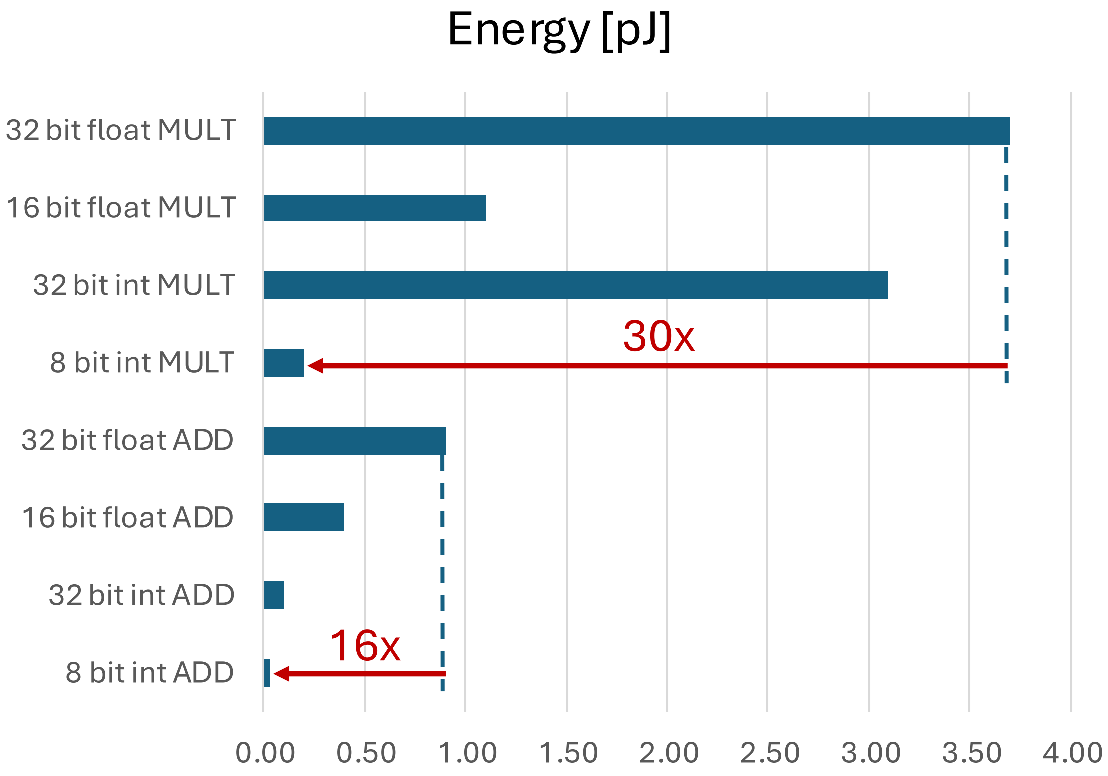
Nikita Kiselev

Deep Learning, Intelligent Systems

December 16, 2025

Introduction





We should make deep learning more efficient...

Xiao G. et al. Smoothquant: Accurate and efficient post-training quantization for large language models, 2023

Contents

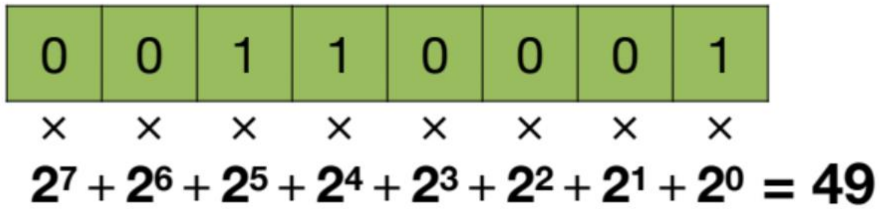
- Quantization
- Pruning
- Distillation
- KV-Cache
- Flash Attention

Quantization

Numerical Data Types: Integer

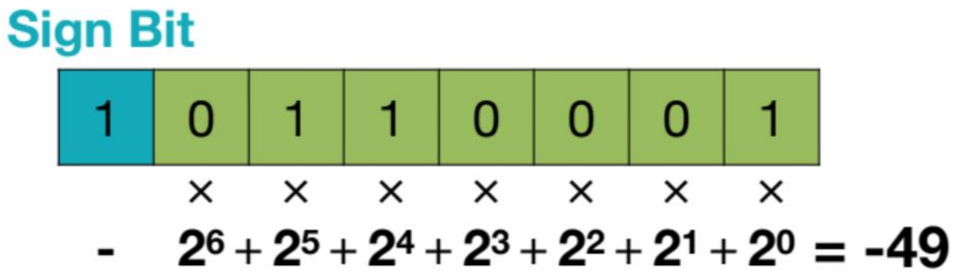
Unsigned Integer

- n-bit range: $[0, 2^n - 1]$



Signed Integer

- n-bit range: $[-2^{n-1} + 1, 2^{n-1} - 1]$
- Both 000…00 and 100…00 represent 0



Fixed-Point Number

0	0	1	1	0	0	0	1
×	×	×	×	×	×	×	×

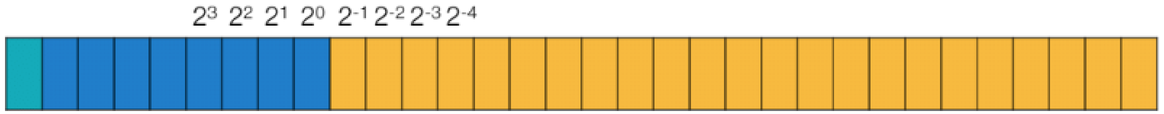
$$-2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = 3.0625$$

0	0	1	1	0	0	0	1
×	×	×	×	×	×	×	×

$$(-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) \times 2^{-4} = 49 \times 0.0625 = 3.0625$$

Floating-Point Number

Example: 32-bit floating-point number in IEEE 754

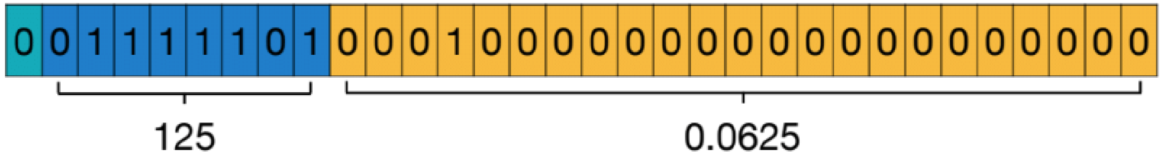


Sign 8 bit Exponent **23 bit Fraction** (significant / mantissa)

$$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127} \longleftarrow \text{Exponent Bias} = 127 = 2^{8-1}-1$$

How to represent **0.265625**?

$\mathbf{0.265625} = 1.0625 \times 2^{-2} = (1 + \underline{0.0625}) \times 2^{\underline{125}-127}$



Floating-Point Number

Exponent Width → Range; Fraction Width → Precision

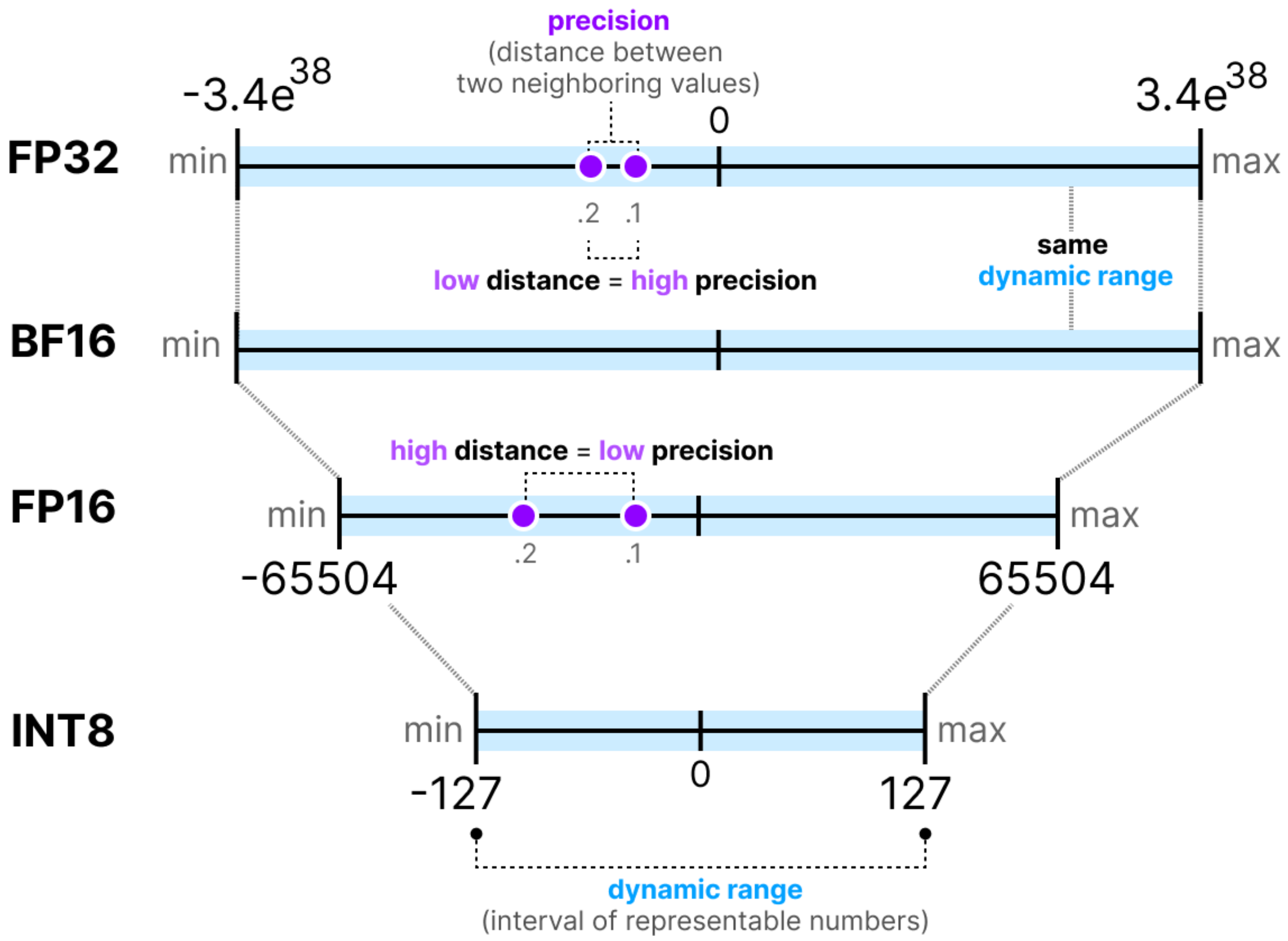
IEEE 754 Single Precision 32-bit Float (IEEE FP32)

IEEE 754 Half Precision 16-bit Float (IEEE FP16)

Google Brain Float (BF16)

	Exponent (bits)	Fraction (bits)	Total (bits)
IEEE 754 Single Precision 32-bit Float (IEEE FP32)	8	23	32
IEEE 754 Half Precision 16-bit Float (IEEE FP16)	5	10	16
Google Brain Float (BF16)	8	7	16

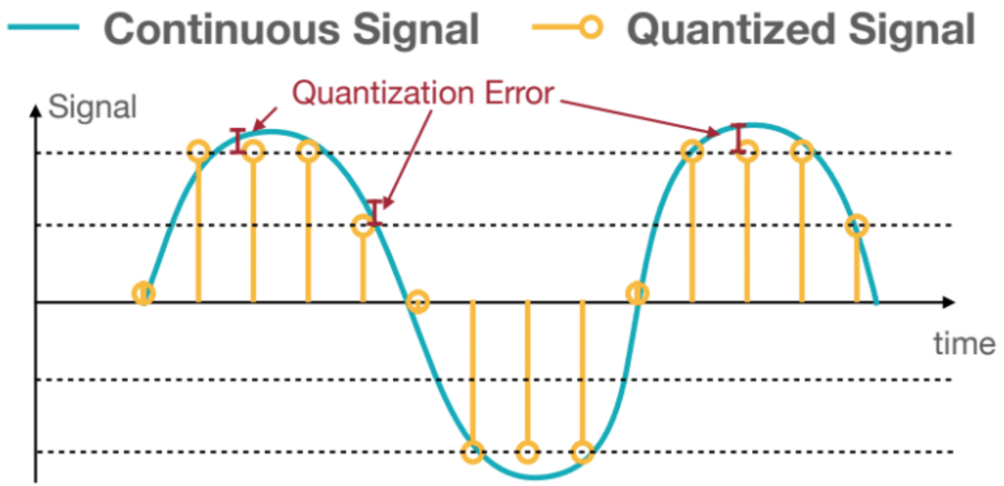
Dynamic Range and Precision

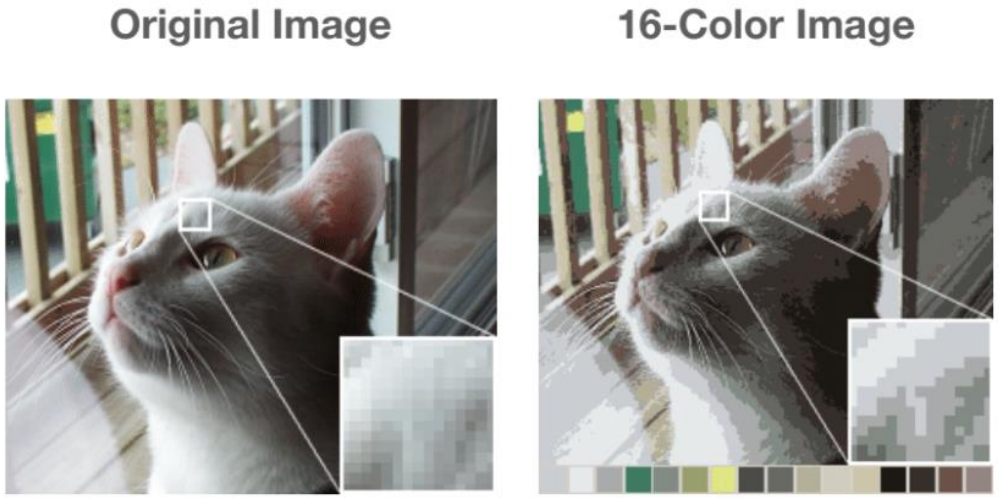


Maarten Grootendorst. A Visual Guide to Quantization, 2024

What is Quantization?

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set





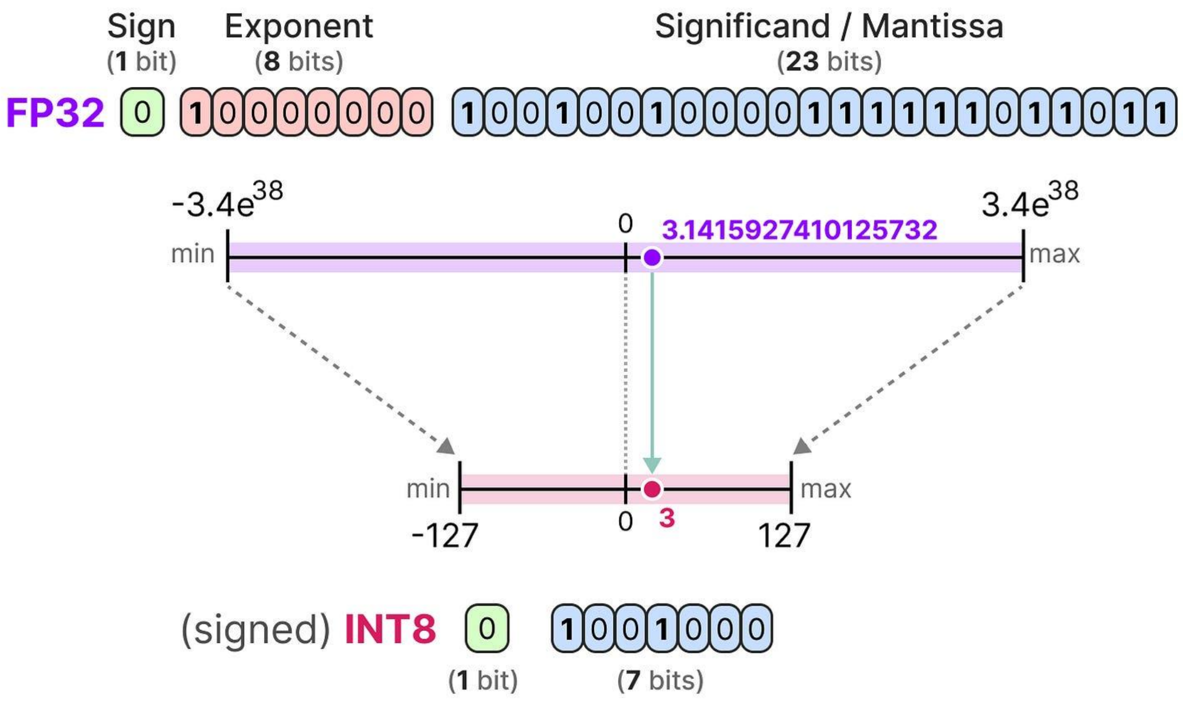
Images are in the public domain.

“Palettization”

Sing Han et al. TinyML and Efficient Deep Learning Computing

https://efficientml.ai

Model Weights Quantization



$$memory = \frac{(\#\text{bits per number})}{8} \times (\#\text{params})$$

$$\textbf{64-bits} = \frac{64}{8} \times 70B \approx \textbf{560}\ GB$$

$$\textbf{32-bits} = \frac{32}{8} \times 70B \approx \textbf{280}\ GB$$

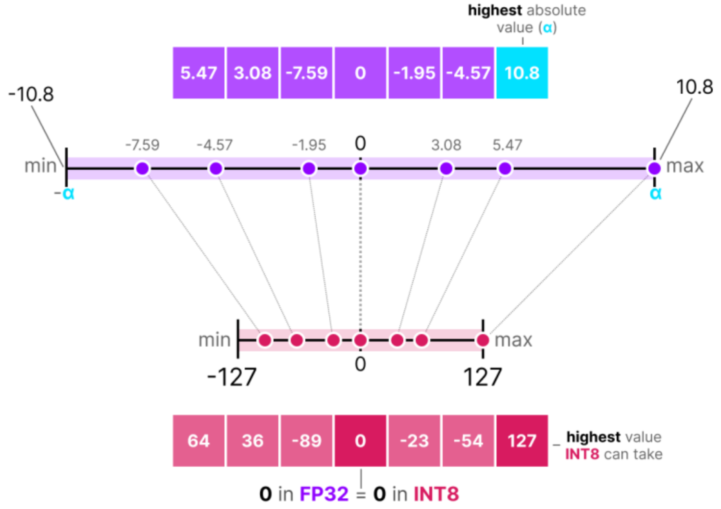
$$\textbf{16-bits} = \frac{16}{8} \times 70B \approx \textbf{140}\ GB$$

Depending on the hardware, integer-based calculations might be faster that floating-point calculations but this isn't always the case. However, computations are generally faster when using fewer bits.

Maarten Grootendorst. A Visual Guide to Quantization, 2024

Quantization Formula

Symmetric

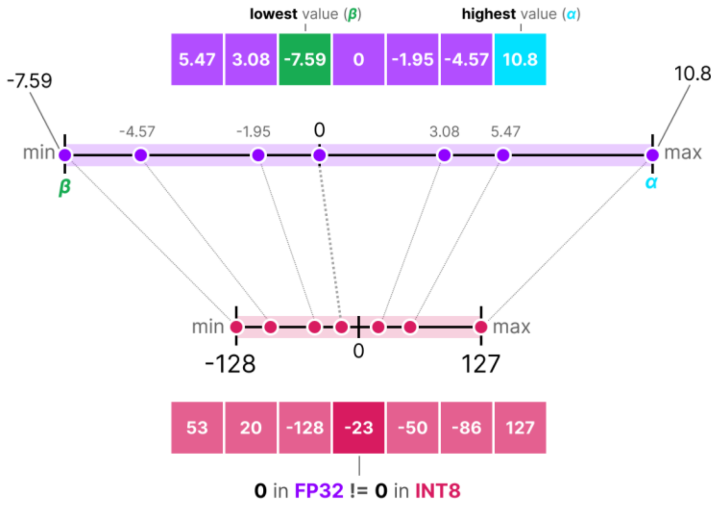


$$s = \frac{2^{b-1} - 1}{\alpha}$$

$$x_{\text{quantized}} = \text{round}(s \cdot x)$$

$$x_{\text{dequantized}} = \frac{x_{\text{quantized}}}{s}$$

Asymmetric



$$s = \frac{2^{b-1} - (-2^{b-1} + 1)}{\alpha - \beta}$$

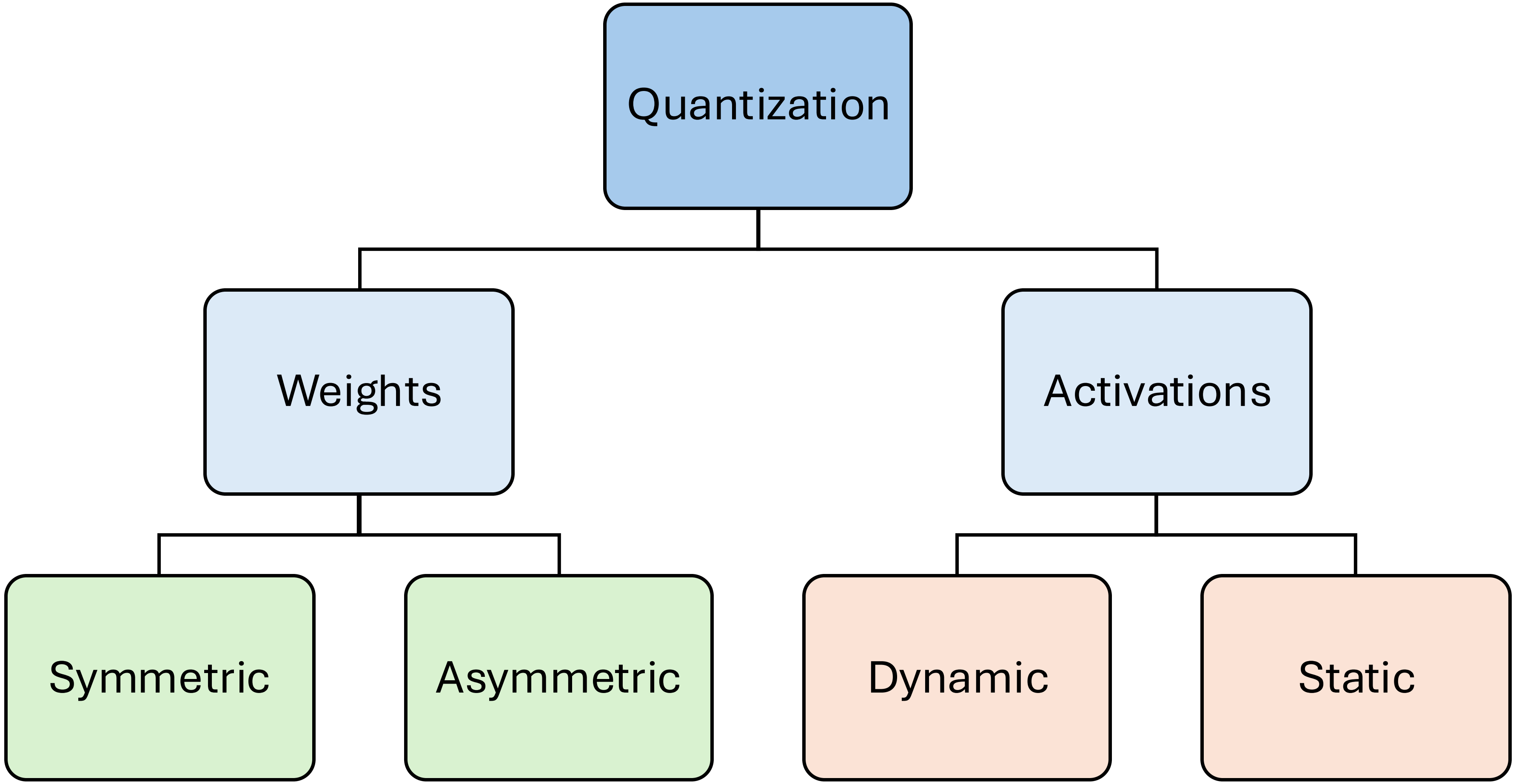
$$z = \text{round}(-s \cdot \beta) - 2^{b-1}$$

$$x_{\text{quantized}} = \text{round}(s \cdot x + z)$$

$$x_{\text{dequantized}} = \frac{x_{\text{quantized}} - z}{s}$$

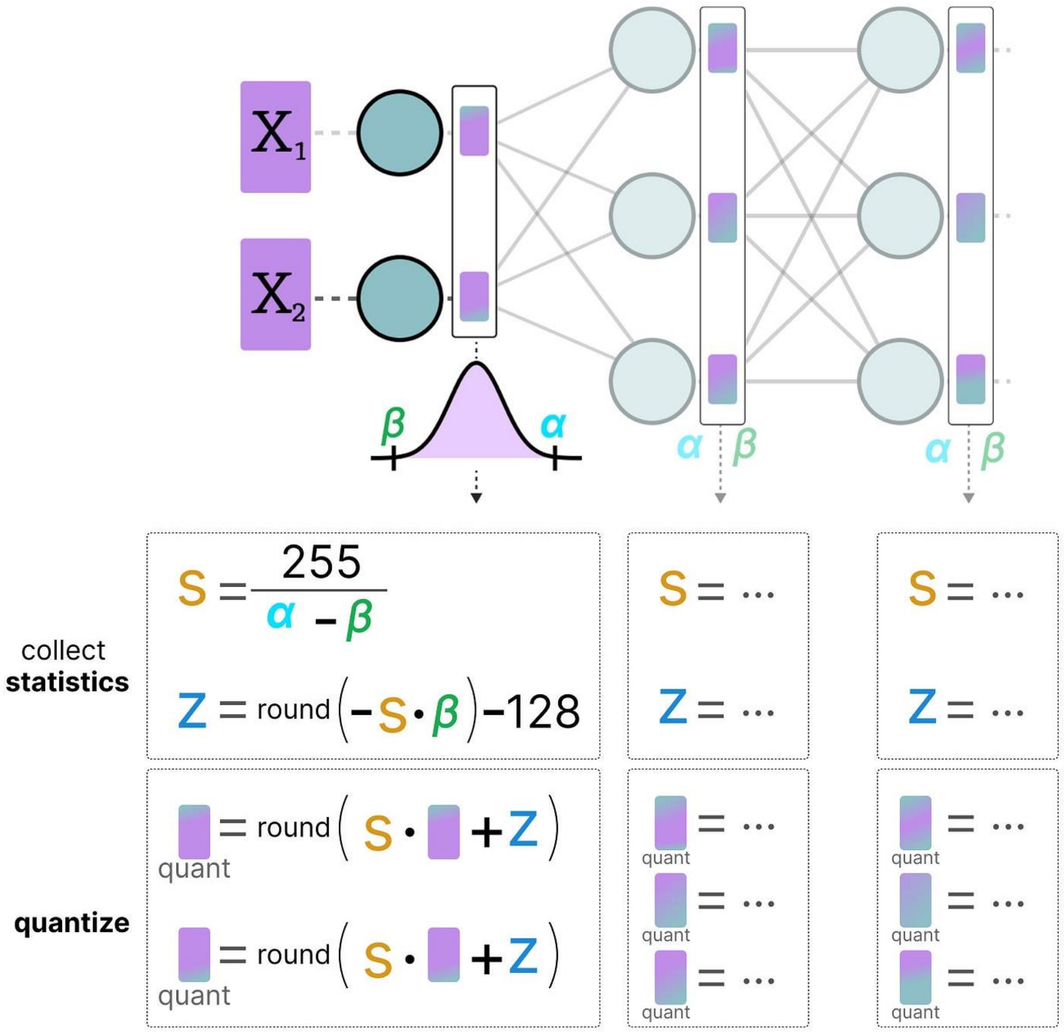
Maarten Grootendorst. A Visual Guide to Quantization, 2024

Post-Training Quantization



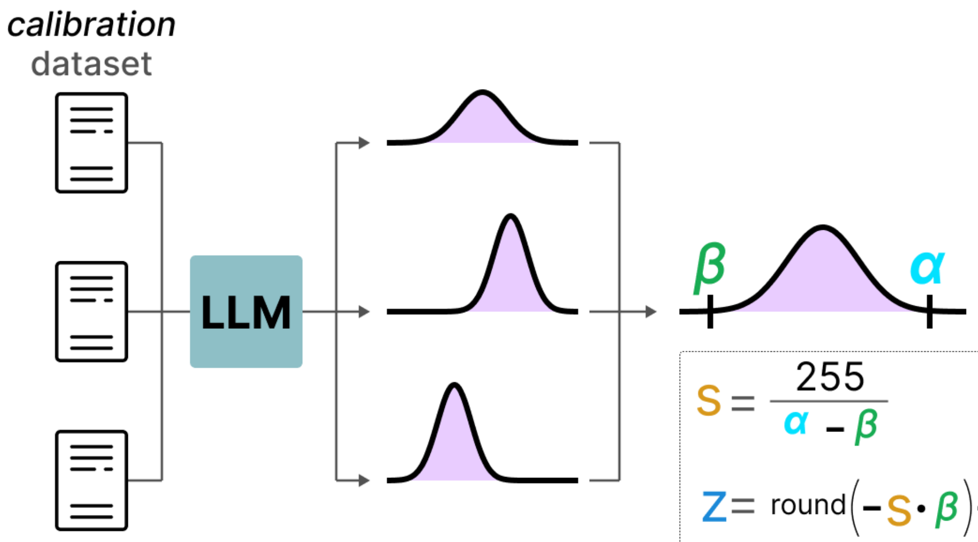
Dynamic Quantization

1. After data passes a hidden layer, its activations are collected
2. This distribution of activations is then used to calculate the zeropoint and scale factor
3. The process is repeated each time data passes through new later



Maarten Grootendorst. A Visual Guide to Quantization, 2024

Static Quantization



$$S = \frac{255}{\alpha - \beta}$$

$$Z = \text{round}(-S \cdot \beta) - 128$$

1. Use **calibration dataset** to collect these potential distributions
2. Dynamic quantization is more accurate, but increase compute time
3. In contrast, static quantization is less accurate, but is faster

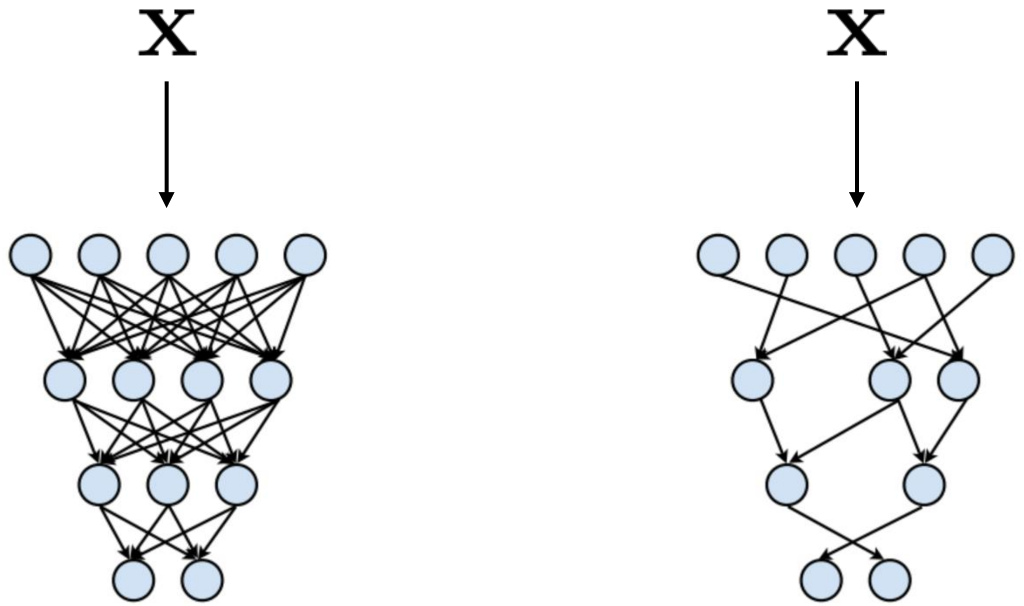
Pruning

Neural Network Pruning

$$\arg\min_{\boldsymbol{\theta}_p} \mathcal{L}_{\boldsymbol{\theta}_p}(\mathbf{x})$$

$$s.t. \|\boldsymbol{\theta}_p\|_0 \leq N$$

- $\mathcal{L}$ represents objective function for neural network training
- $\mathbf{x}$ is input, $\boldsymbol{\theta}$ is original weights, $\boldsymbol{\theta_p}$ is pruned weights
- $||\boldsymbol{\theta_p}||_0$ calculates the #nonzeros in $\boldsymbol{\theta_p}$, and N is the target of #nonzeros



$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}}(\mathbf{x})$$

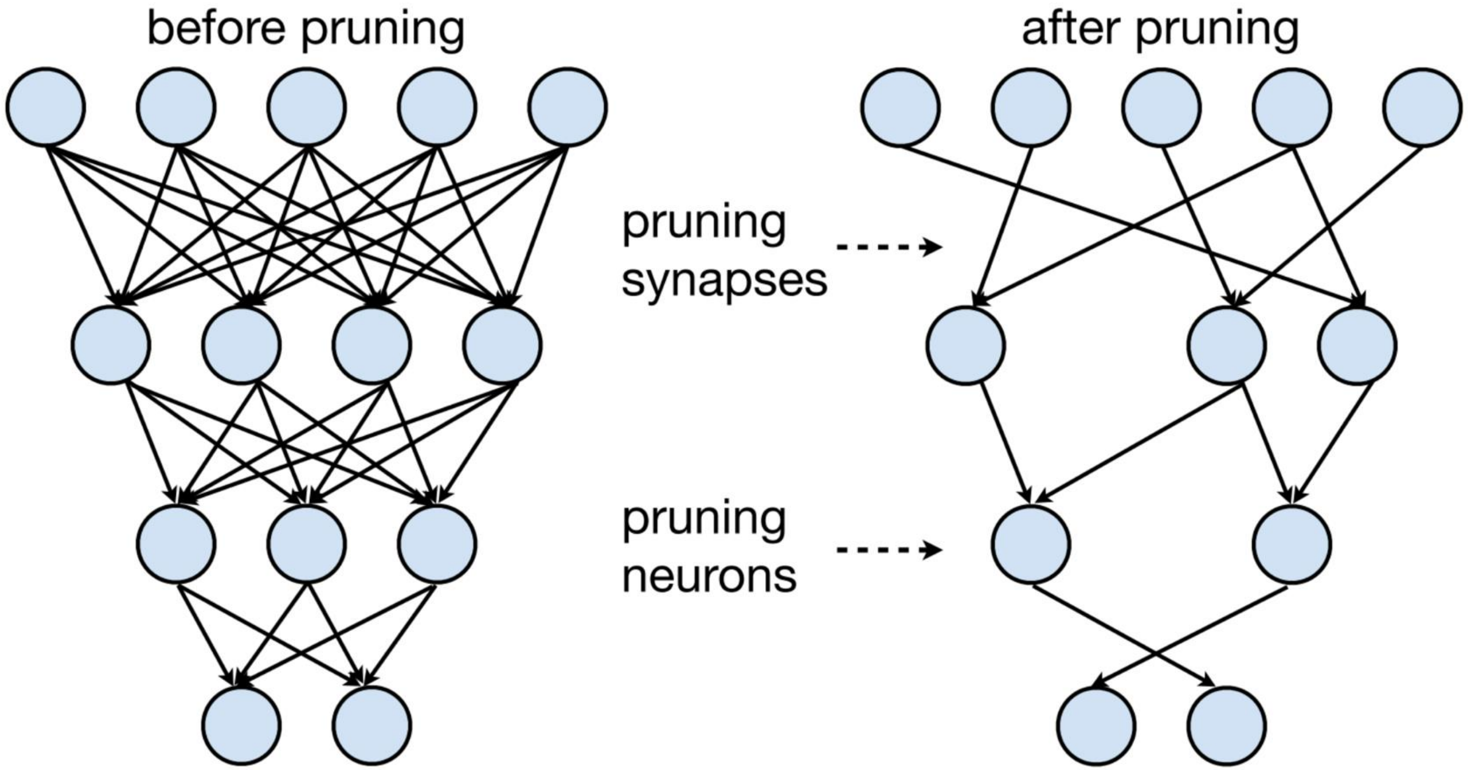
$$\arg\min_{\boldsymbol{\theta}_p} \mathcal{L}_{\boldsymbol{\theta}_p}(\mathbf{x})$$

$$s.t. \|\boldsymbol{\theta}_p\|_0 \leq N$$

Han S. et al. Learning both weights and connections for efficient neural network, 2015

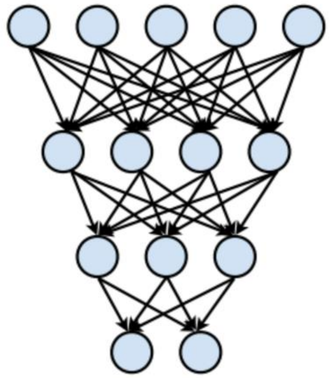
Neural Network Pruning

Make neural network smaller by removing synapses and neurons

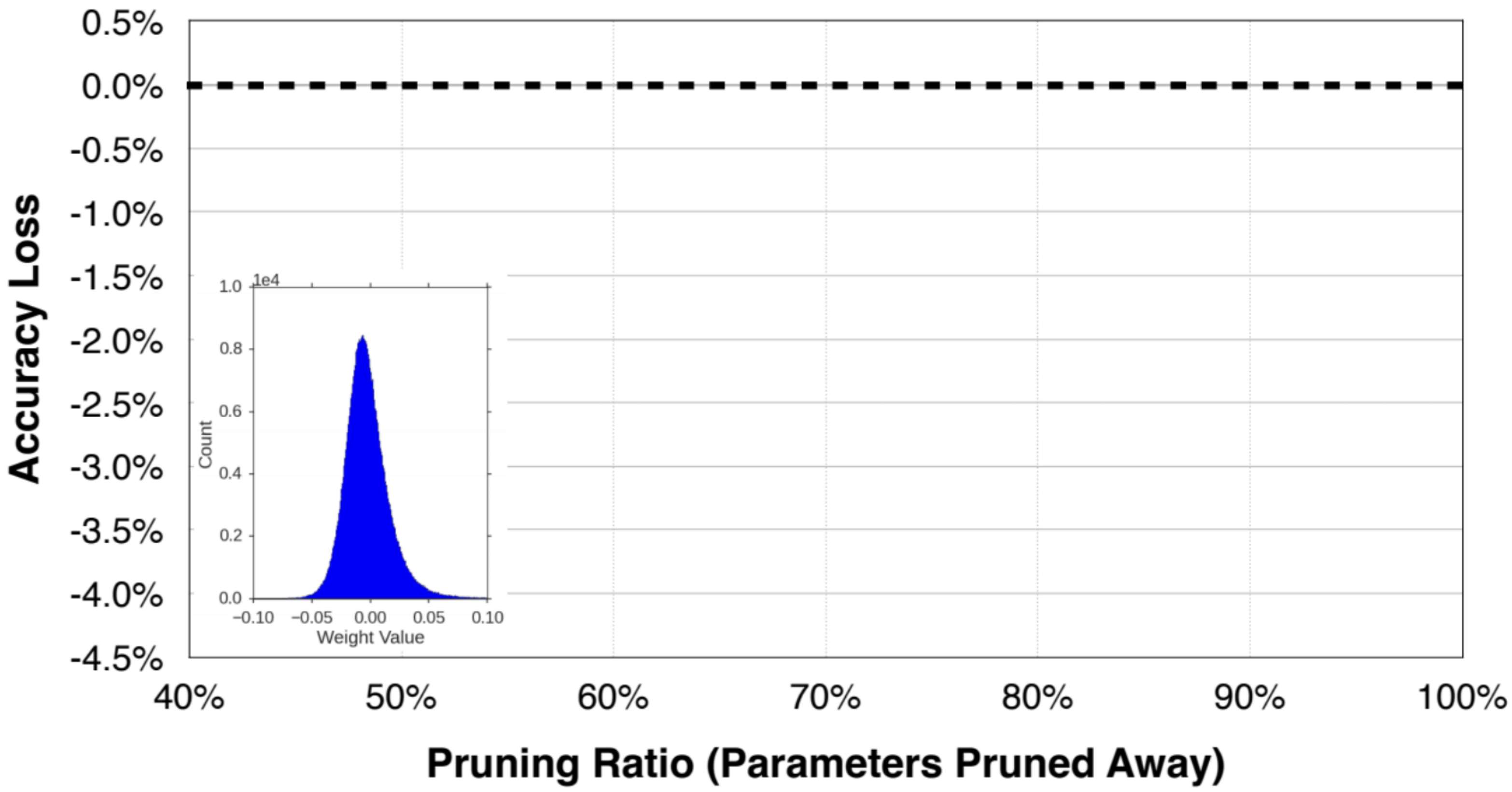


Han S. et al. Learning both weights and connections for efficient neural network, 2015

Neural Network Pruning

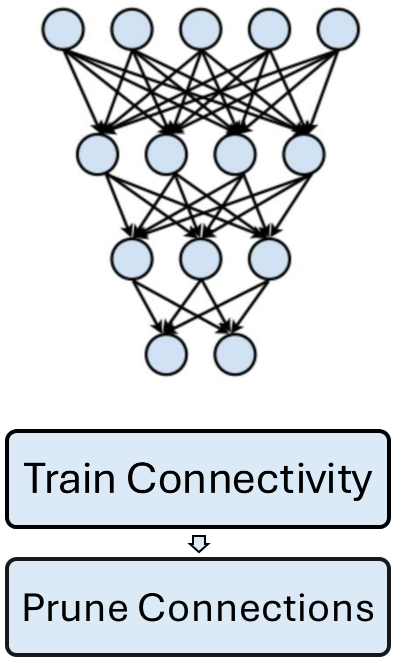


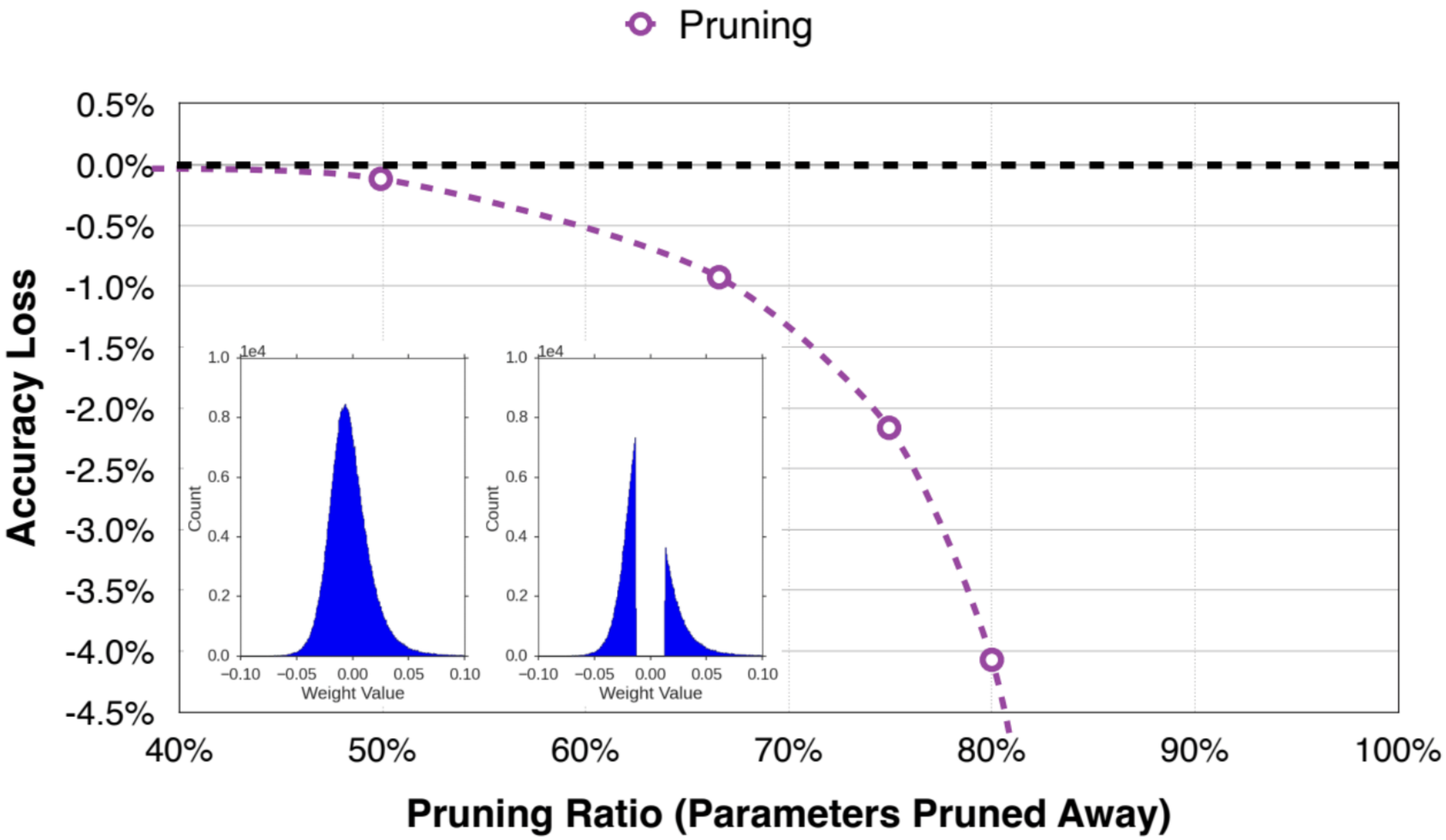
Train Connectivity



Han S. et al. Learning both weights and connections for efficient neural network, 2015

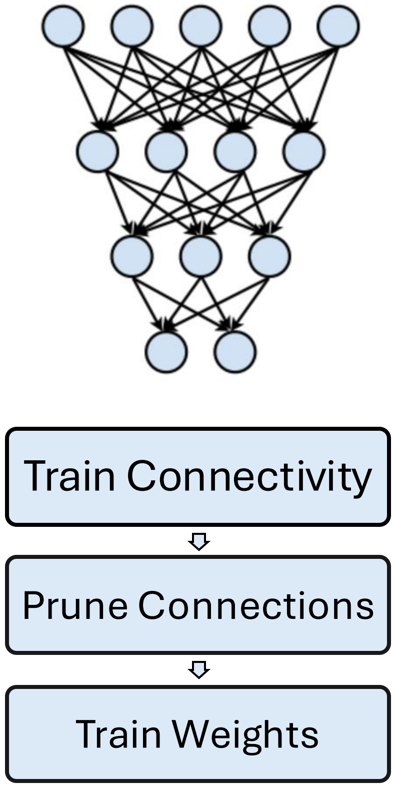
Neural Network Pruning

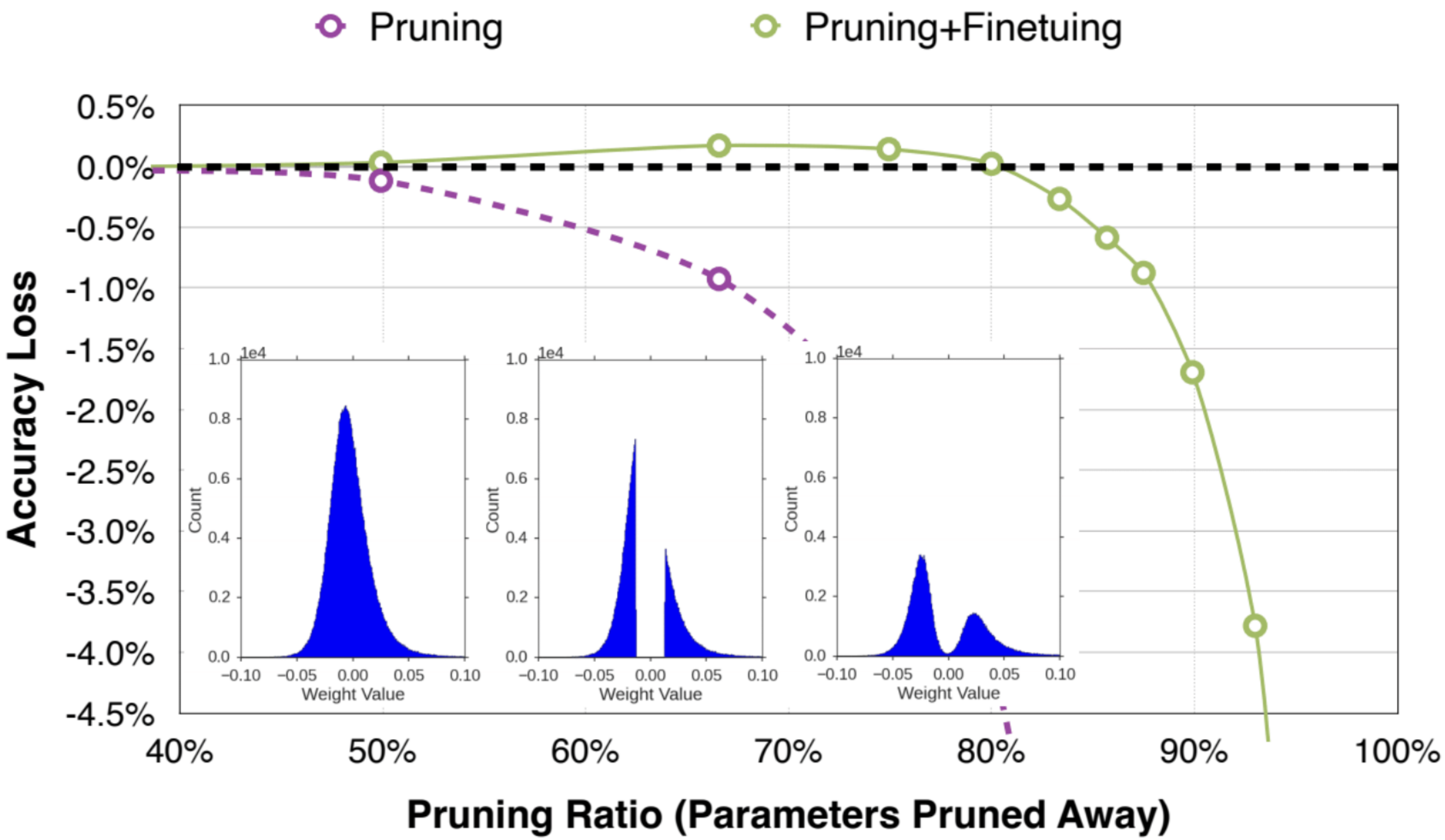




Han S. et al. Learning both weights and connections for efficient neural network, 2015

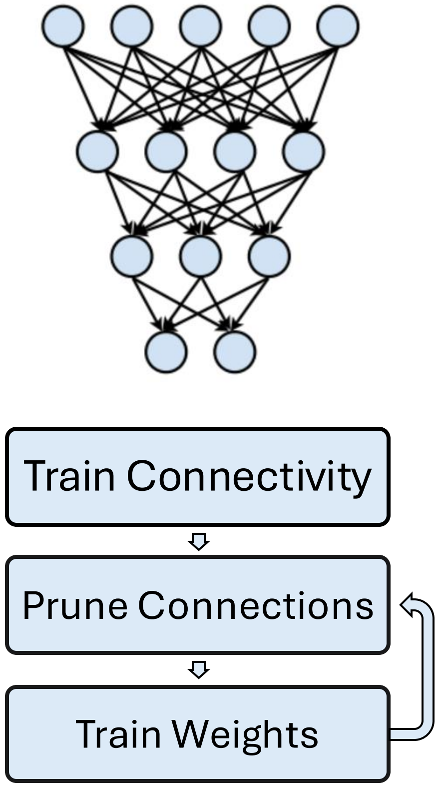
Neural Network Pruning

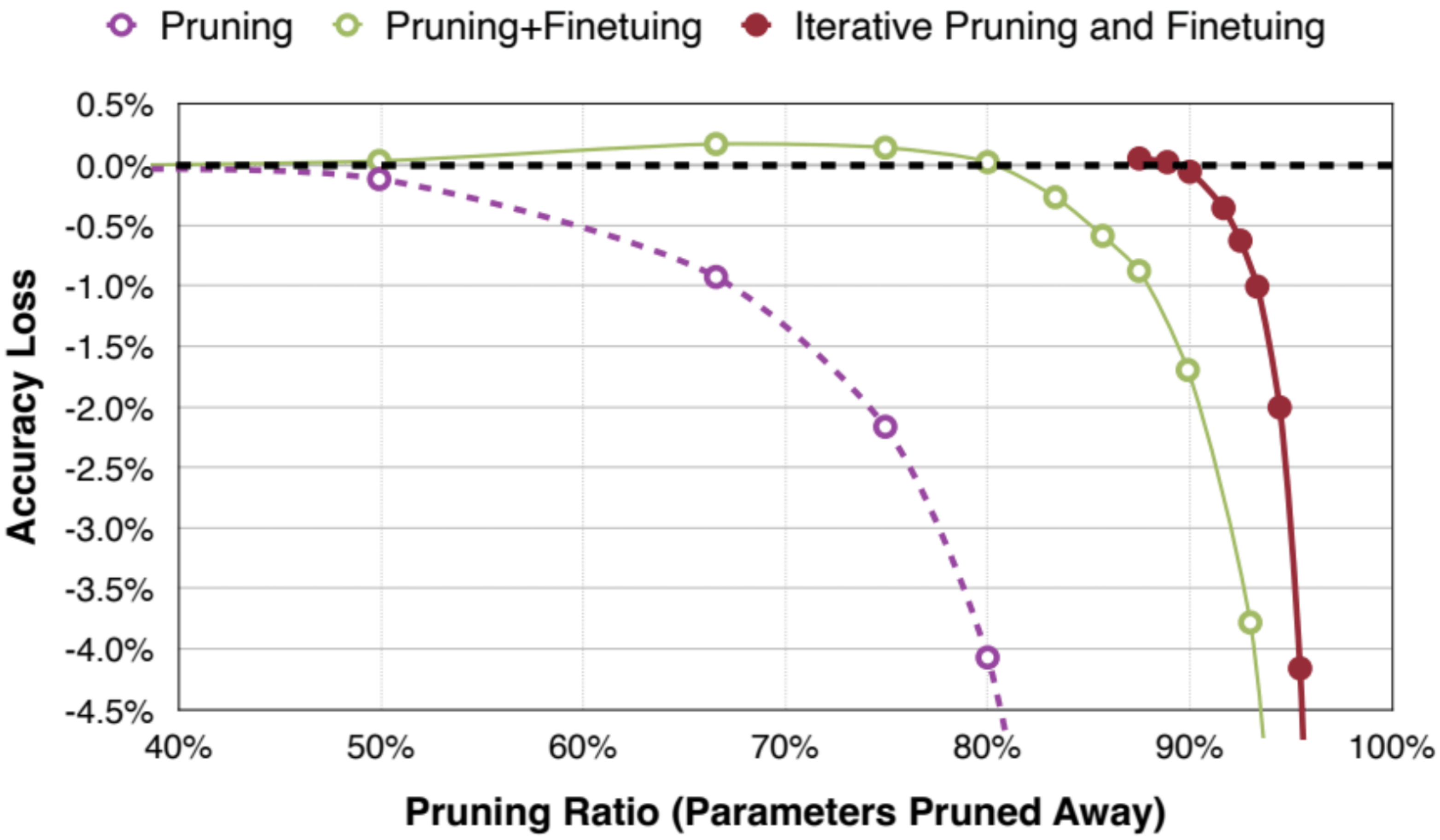




Han S. et al. Learning both weights and connections for efficient neural network, 2015

Neural Network Pruning

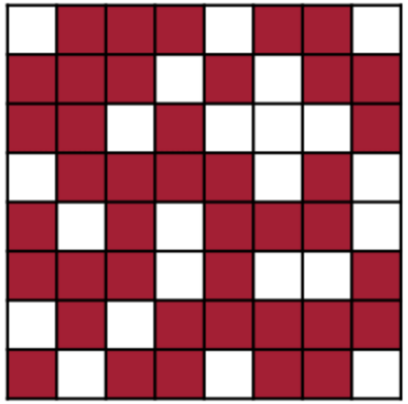


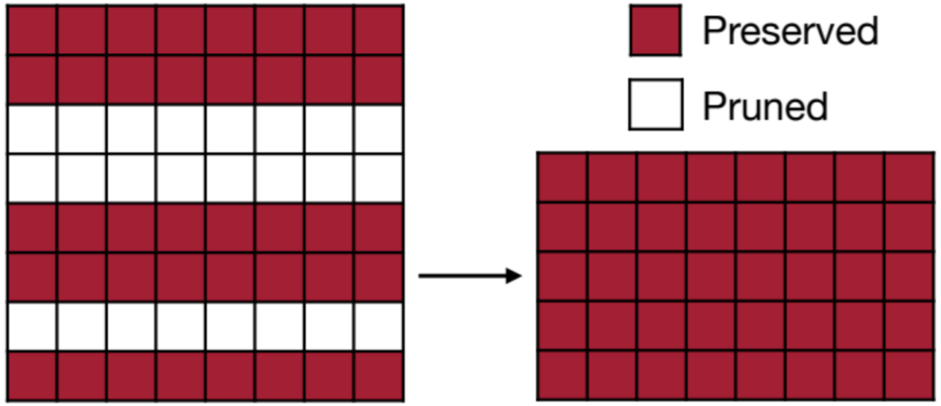


Han S. et al. Learning both weights and connections for efficient neural network, 2015

Pruning at Different Granularities

A simple example of 2D weight matrix





Fine-grained / Unstructured

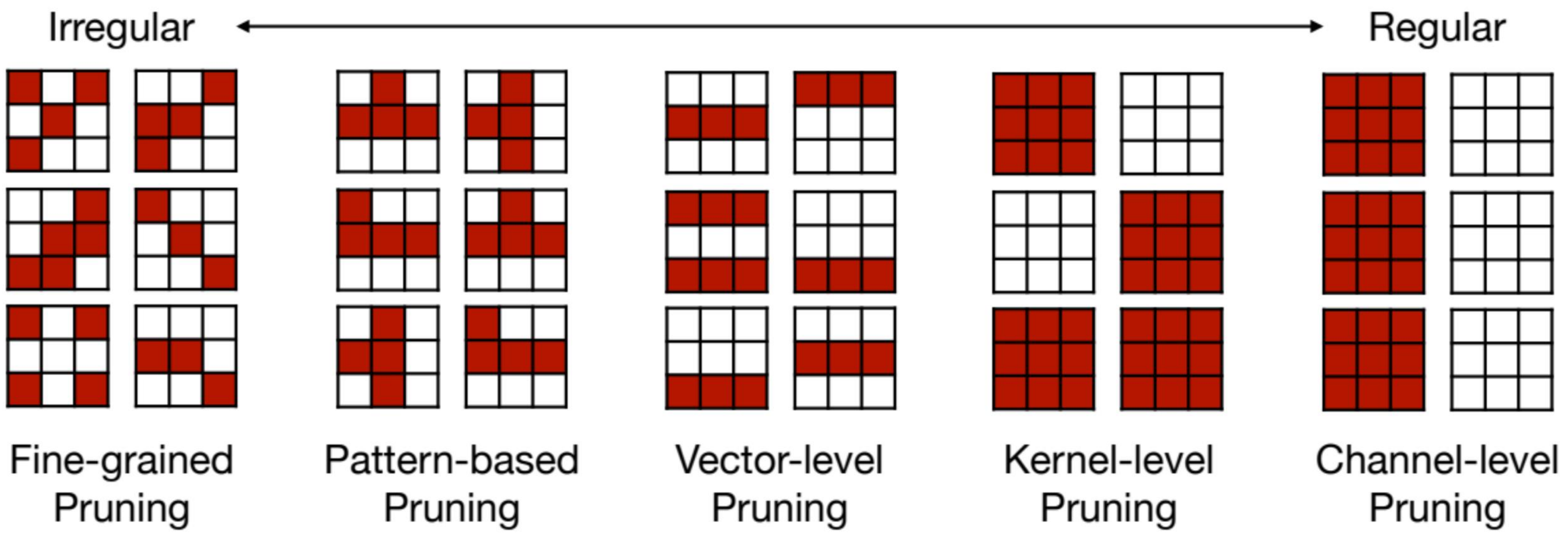
- More flexible pruning index choice
- Hard to accelerate (irregular)

Coarse-grained / Structured

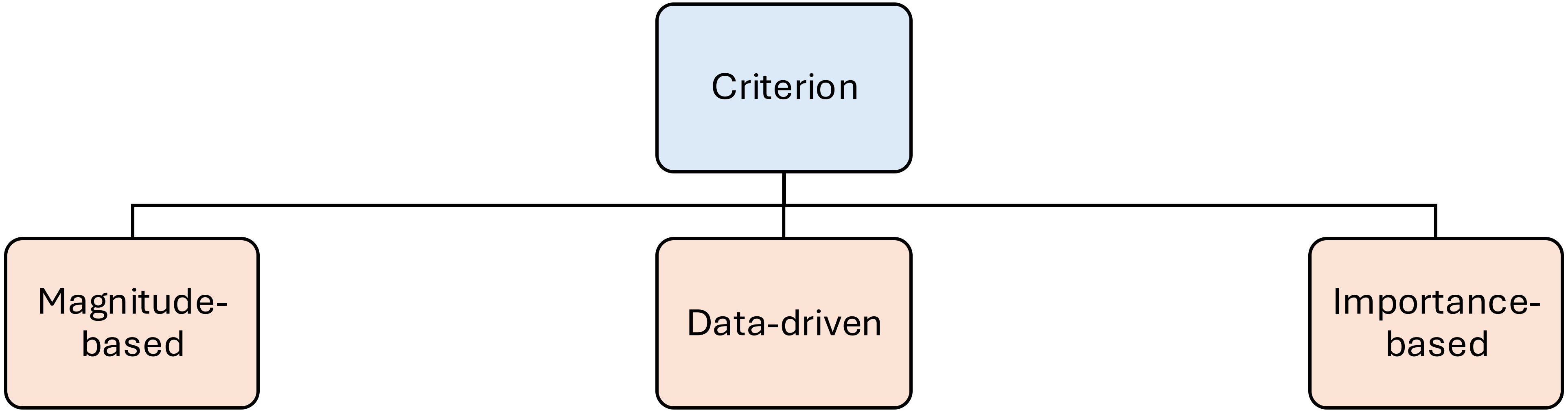
- Less flexible pruning index choice (a subset of the fine-grained case)
- Easy to accelerate (just a smaller matrix!

Pruning at Different Granularities

The case of convolutional layers



Pruning Criterions



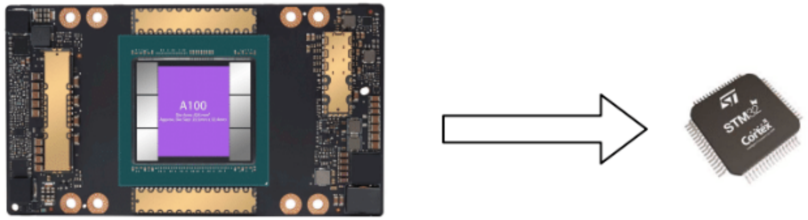
Remove weights with minimal absolute values

1. Neuron value changes are too small – replace with constant
2. Based on neuron activation on dataset
3. Combine neurons with high correlation

Remove neurons with minimal impact on loss

Distillation

Challenge: limited hardware resources



	Cloud AI	Tiny AI
Computation (fp32)	19.5 TFLOPS	MFLOPs
Memory	80GB	256kB
Neural Network	ResNet ViT-Large …	MCUNet MobileNetV2-Tiny …

Neural network must be tiny to run efficiently on tiny edge devices.
How to train tiny model with the help of large model?

Tiny models are hard to train

Tiny models underfit large datasets, how to help them…?

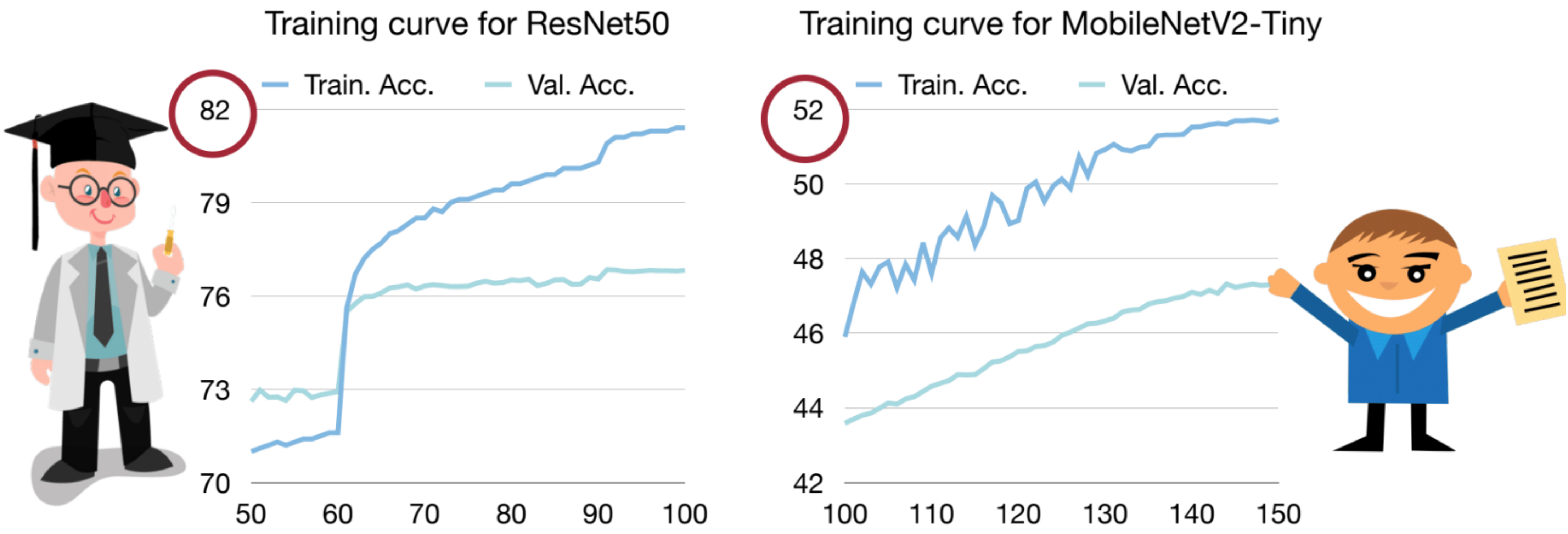
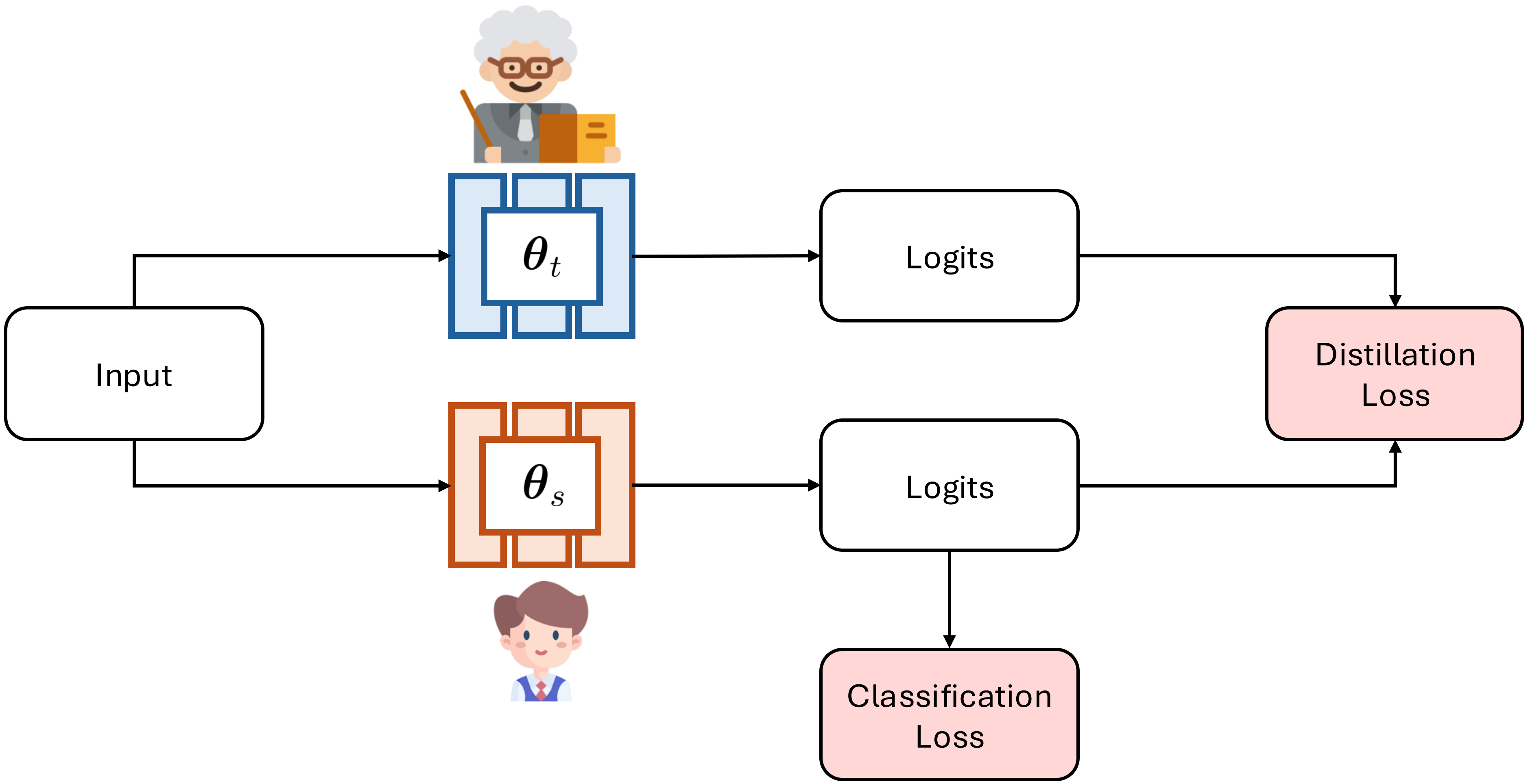
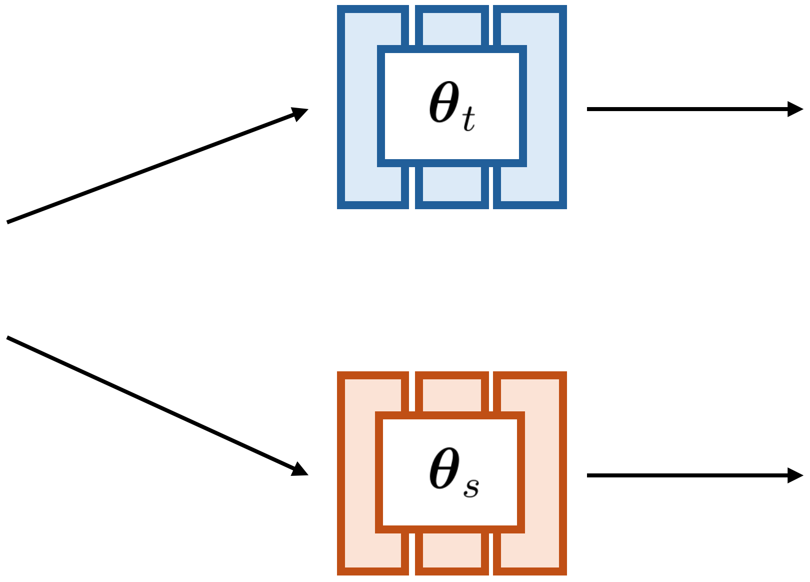


Illustration of Knowledge Distillation



Intuition of Knowledge Distillation

Matching prediction probabilities between teacher and student



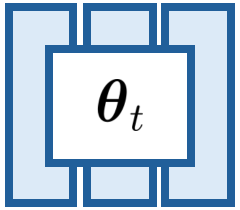
	Logits	Probabilities
Cat	5	0.982
Dog	1	0.017

	Logits	Probabilities
Cat	3	0.731
Dog	2	0.269

The student model is less confident

Intuition of Knowledge Distillation

Concept of temperature



$$\frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}$$

	Logits	Probabilities (T=1)	Probabilities (T=10)
Cat	5	0.982	0.599
Dog	1	0.017	0.401

$$\frac{\exp(z_i/T)}{\sum_{j=1}^{K} \exp(z_j/T)}$$

Formal Definition of KD

Neural networks typically use a softmax function to generate the **logits** z_i to class **probabilities** $p(z_i, T) = \frac{\exp(z_i/T)}{\sum_{j=1}^{k} \exp(z_j/T)}$

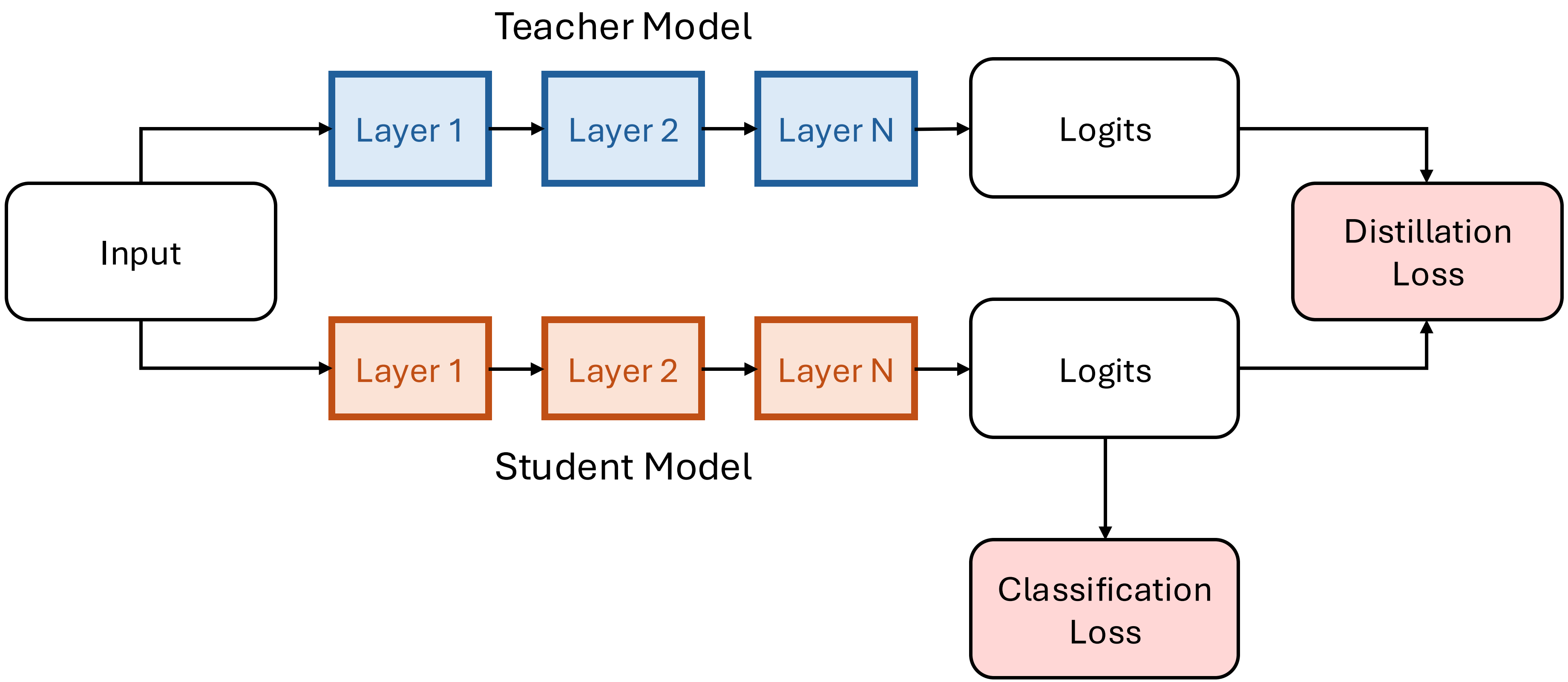
Temperature is normally set to 1

The goal of knowledge distillation is to **align the class probability distribution from teacher and student networks**

What to match?

1. Output logits
2. Intermediate weights
3. Intermediate features

Matching output logits



Cross Entropy Loss

$$\mathbb{E}(-p_t \log p_s)$$

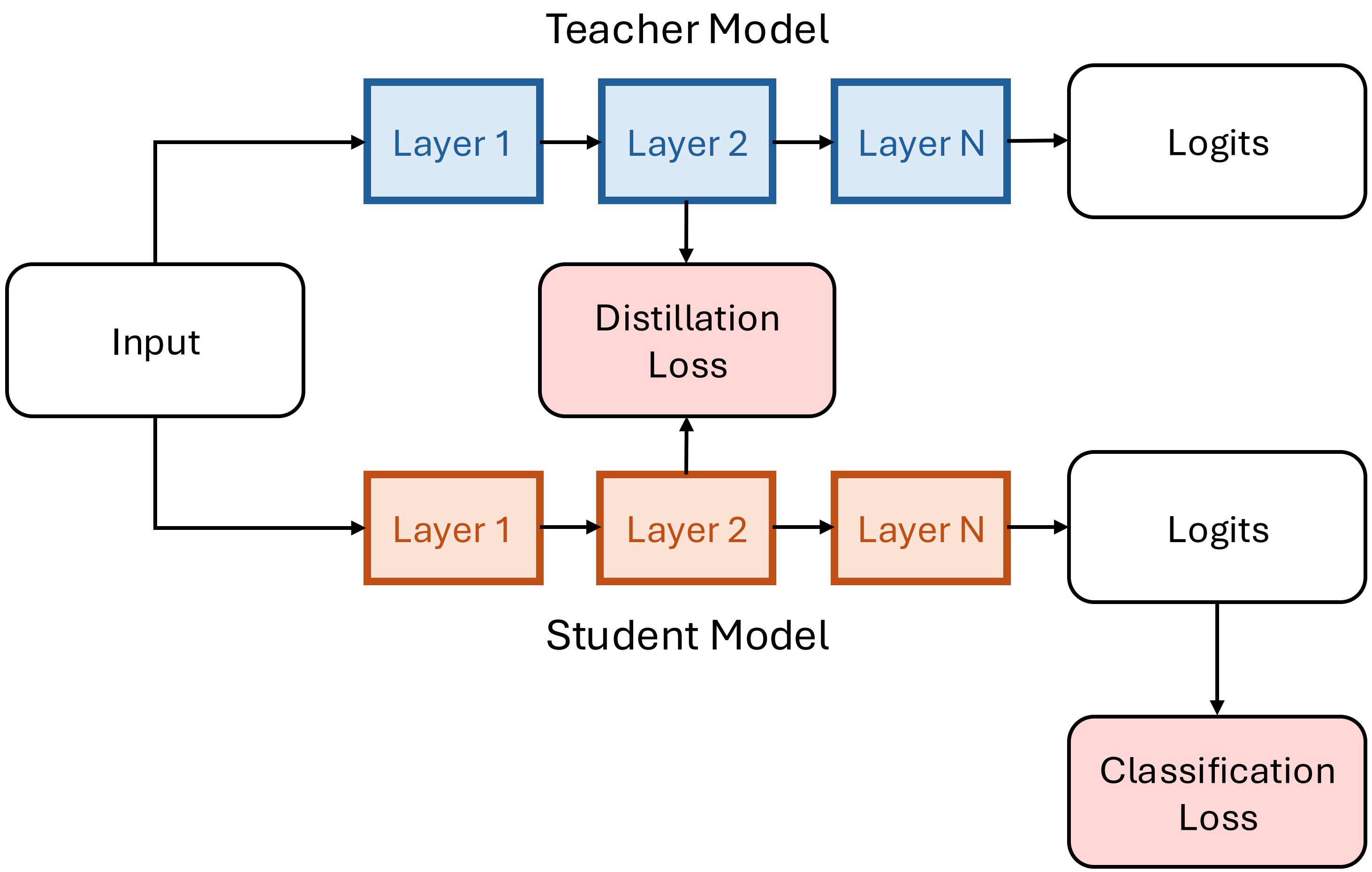
L2 Loss

$$\mathbb{E}\|p_t - p_s\|_2^2$$

What to match?

1. Output logits
2. Intermediate weights
3. Intermediate features

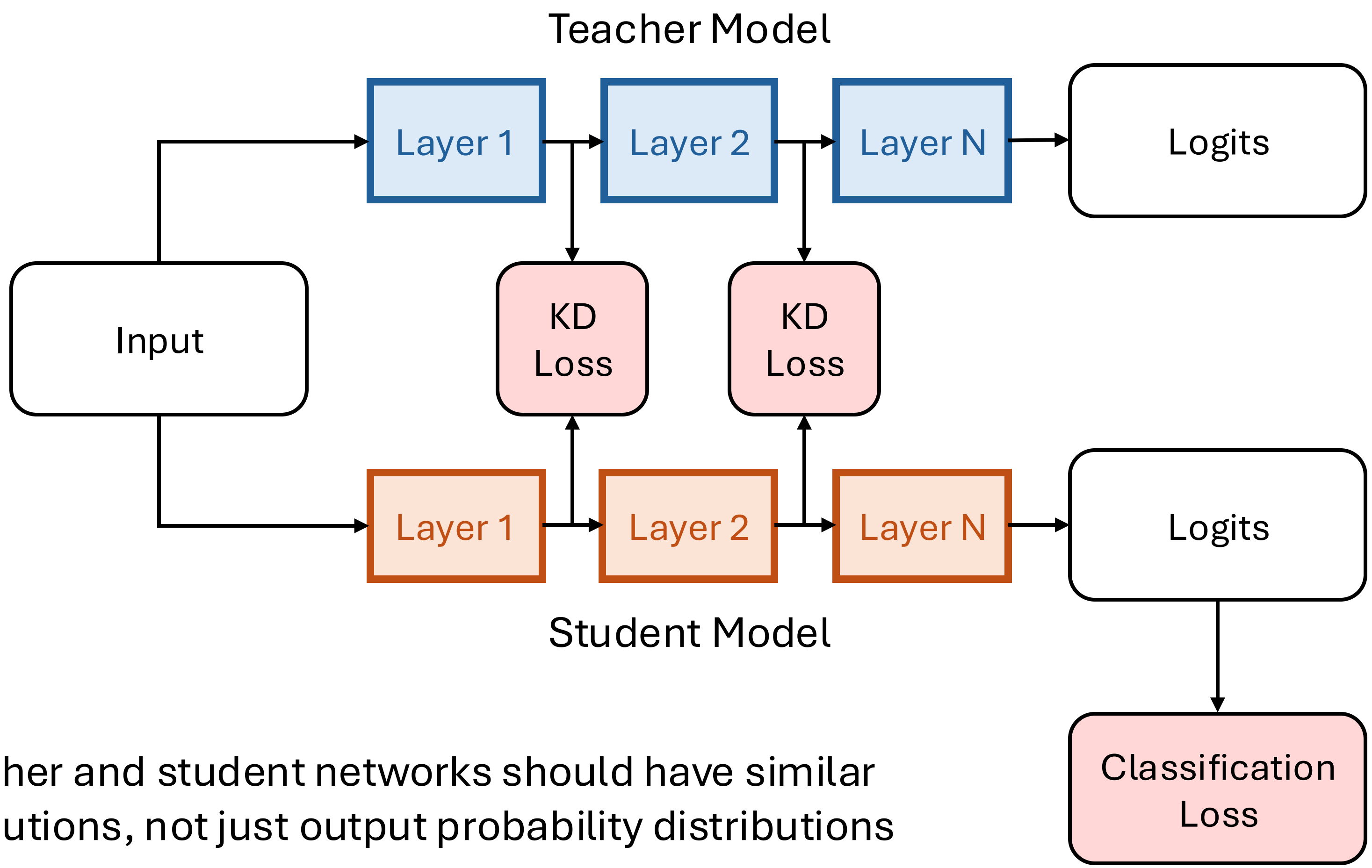
Matching intermediate weights



What to match?

1. Output logits
2. Intermediate weights
3. Intermediate features

Matching intermediate features

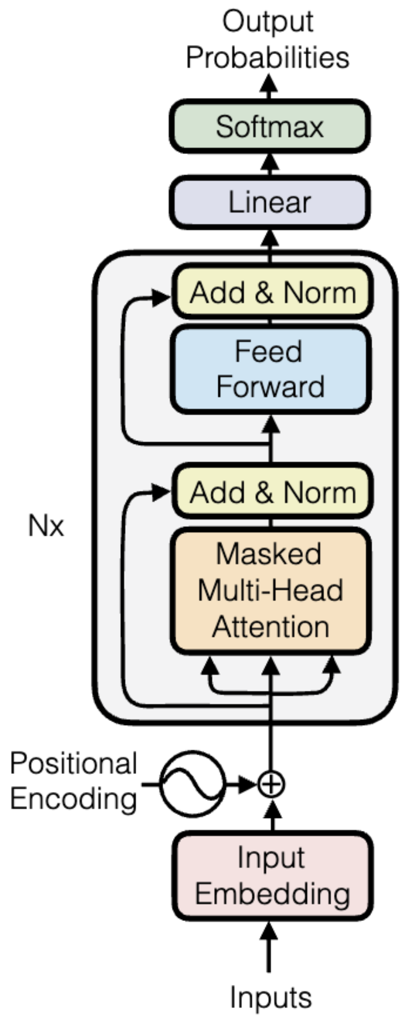


Intuition: teacher and student networks should have similar feature distributions, not just output probability distributions

TIME FOR A BREAK

KV-Cache

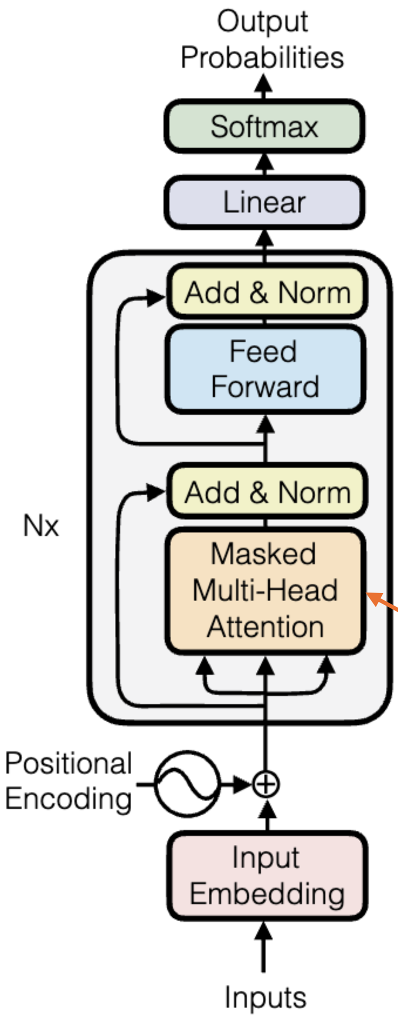
Preliminaries



- GPT uses a Transformer Decoder that generates text autoregressively – one token at a time
- Each step takes all previous tokens as input and predicts the next token's probability

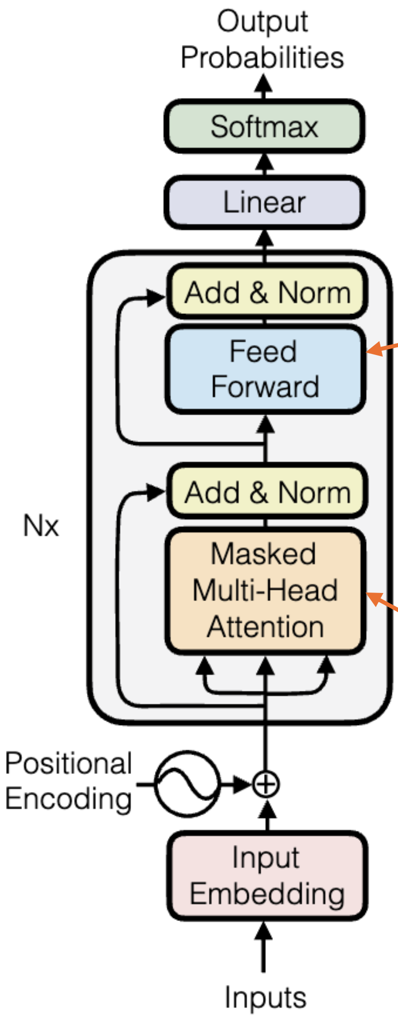
Problem: without optimization, GPT must recompute attention for all previous tokens at every step

Transformer Decoder



Self-Attention (masked): tokens look at the previous tokens
queries, keys, values are computed from hidden states

Transformer Decoder

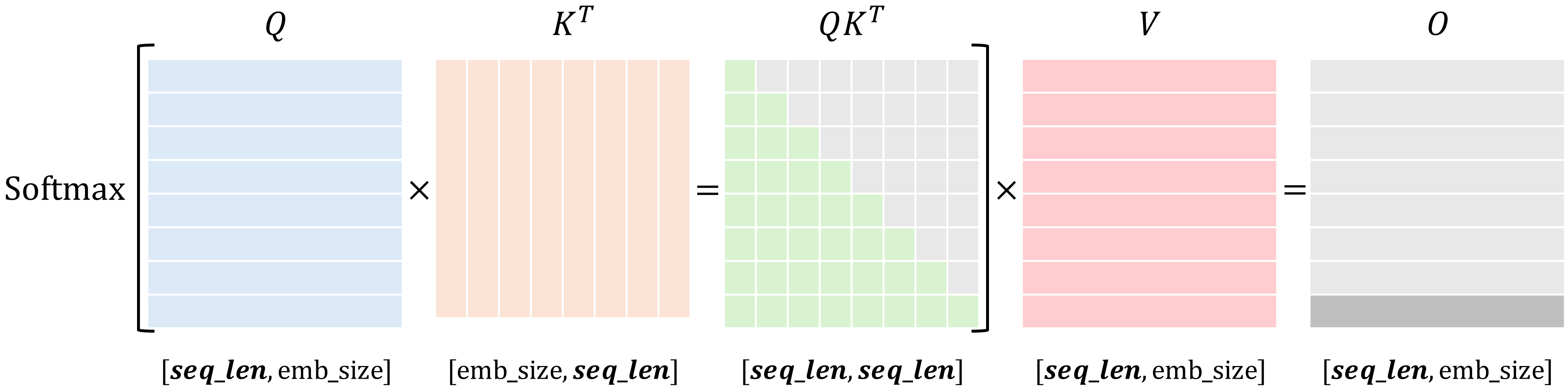


Feed-Forward: after taking information from other tokens, take a moment to think and process this information

Self-Attention (masked): tokens look at the previous tokens queries, keys, values are computed from hidden states

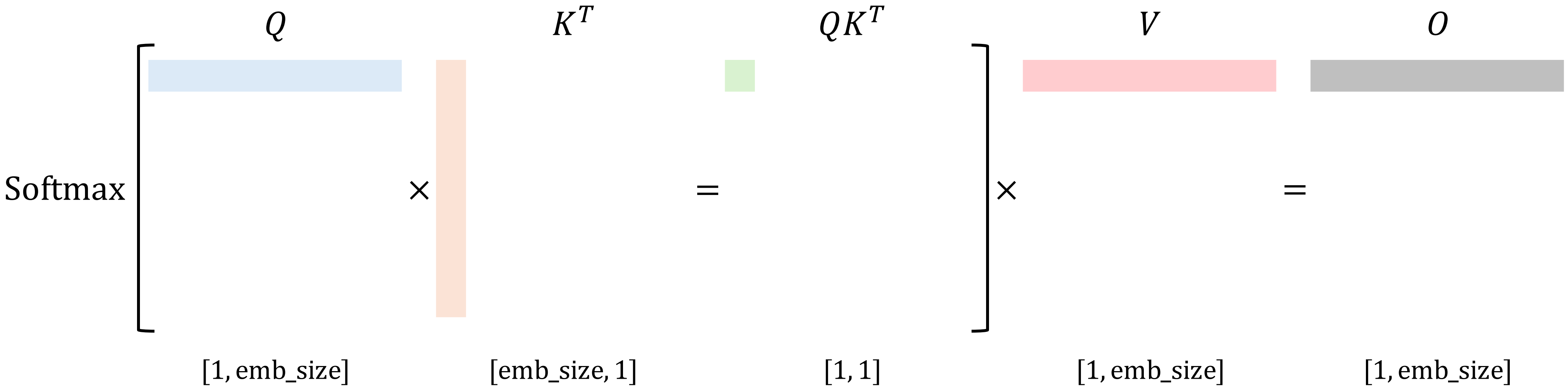
Scaled dot-product Attention

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



Scaled dot-product Attention

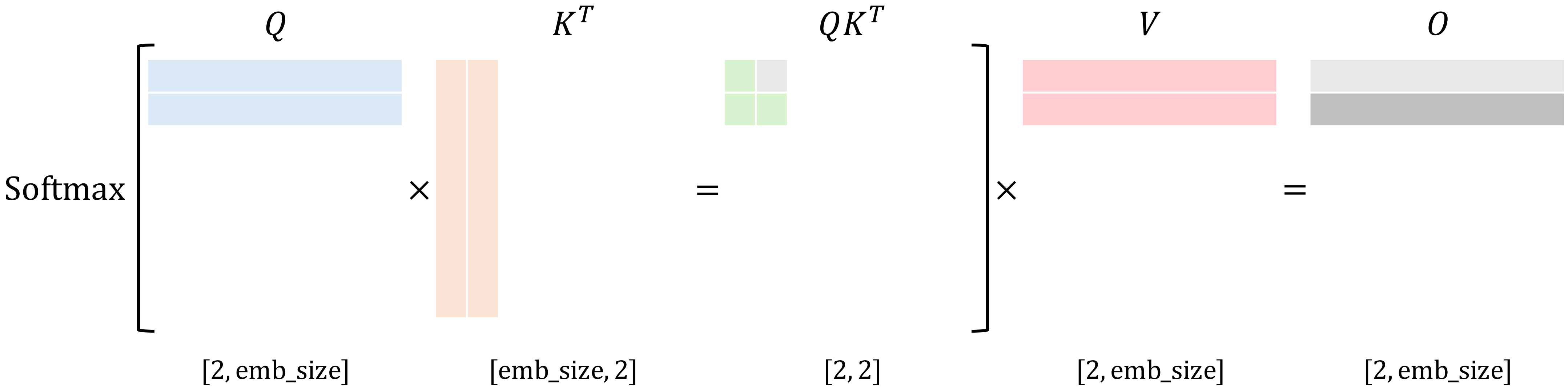
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers

Scaled dot-product Attention

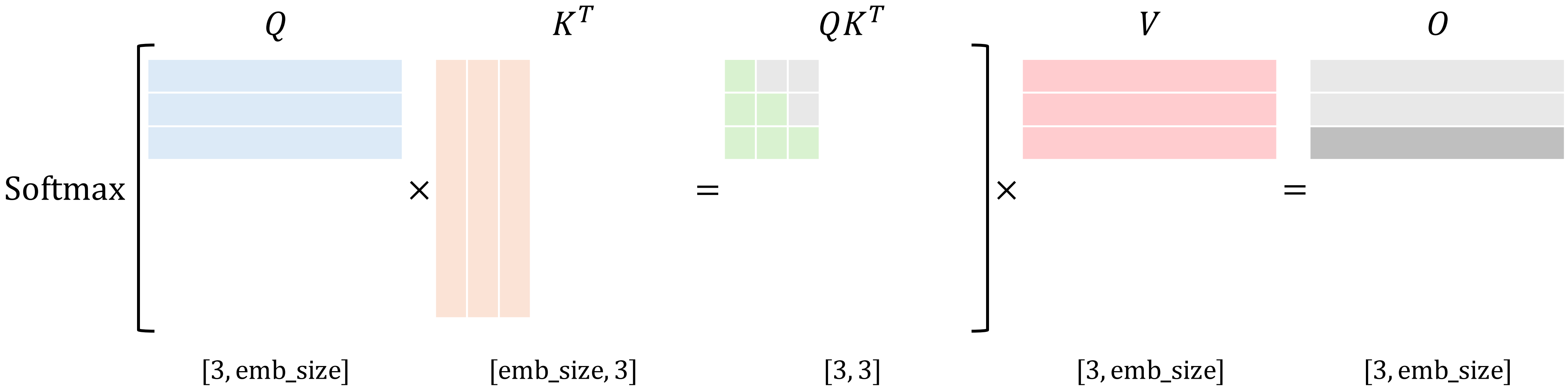
$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict

Scaled dot-product Attention

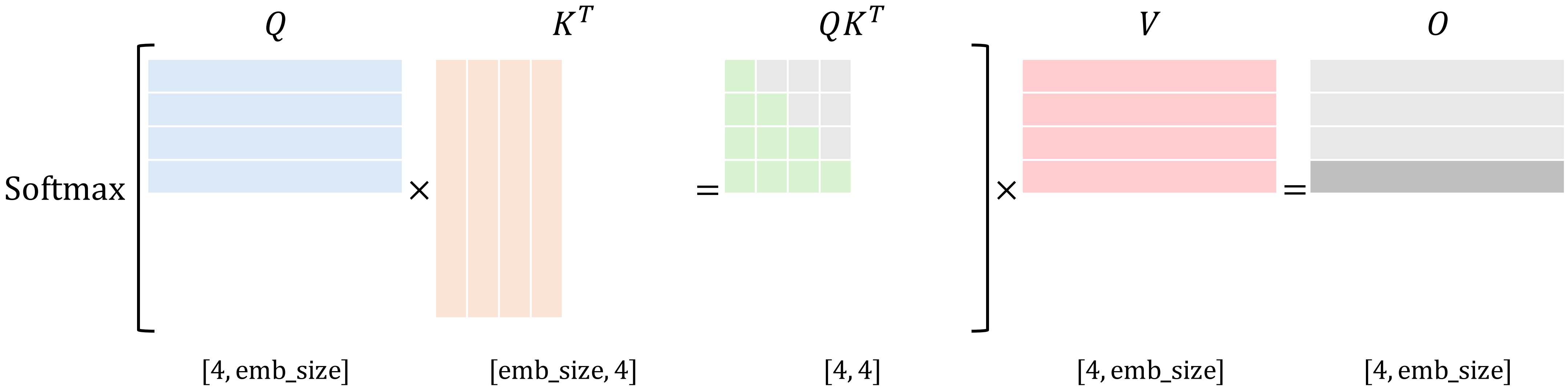
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow's

Scaled dot-product Attention

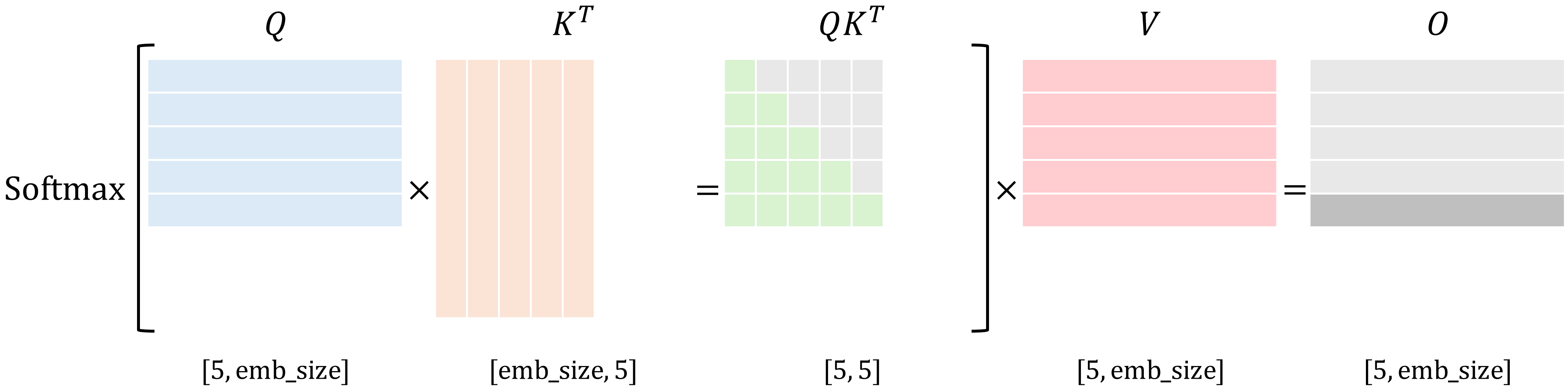
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow’s words

Scaled dot-product Attention

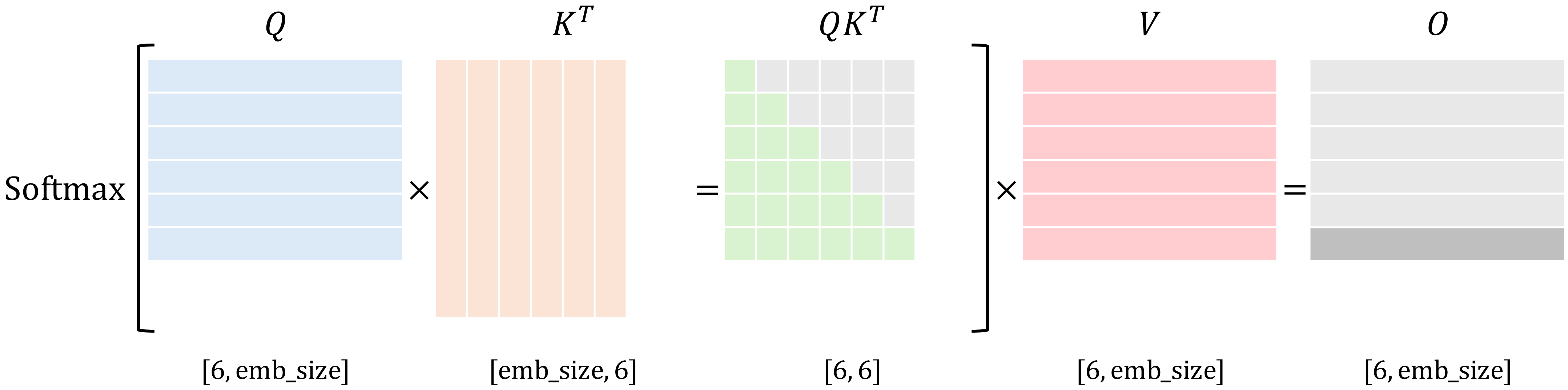
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow's words from

Scaled dot-product Attention

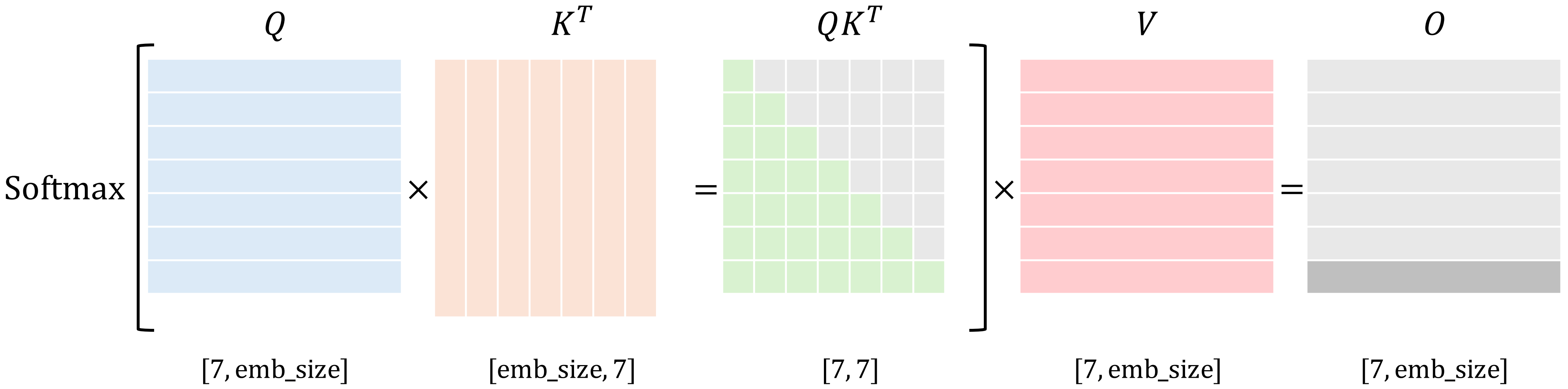
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow's words from today's

Scaled dot-product Attention

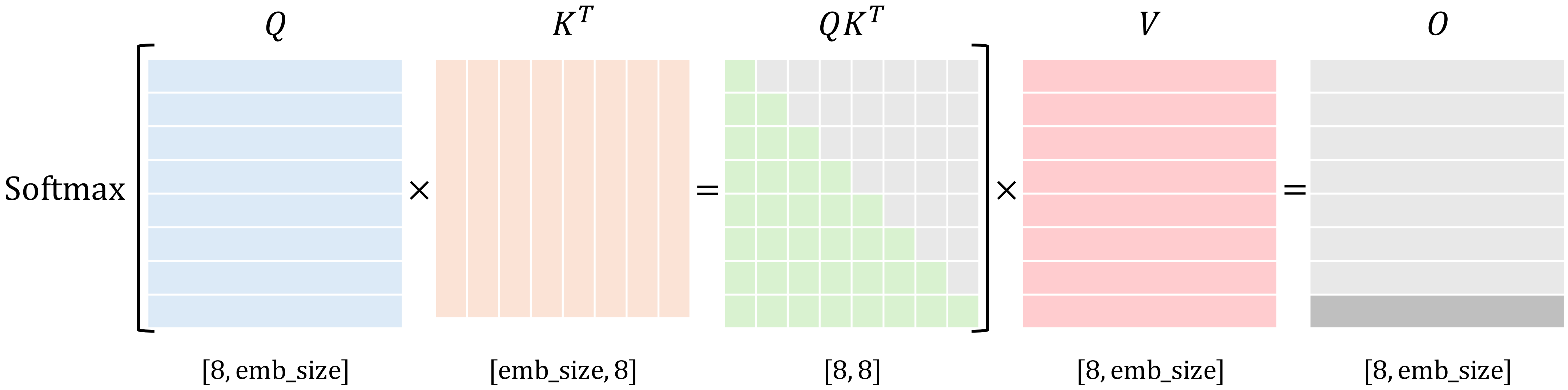
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow's words from today's hidden

Scaled dot-product Attention

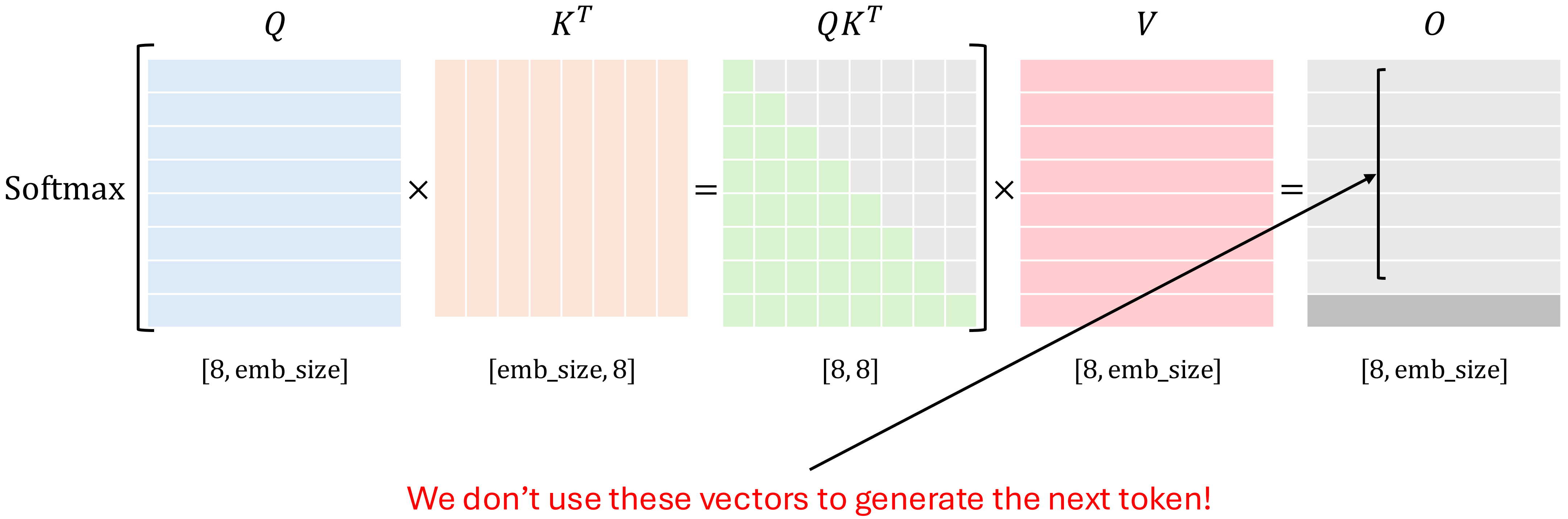
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow's words from today's hidden context

Scaled dot-product Attention

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



KV-Cache

Cache the past keys and values once → reuse them for all future queries:

$$\mathbf{K}_{\text{cache}} = [k_1, k_2, \ldots, k_{t-1}]^T, \quad \mathbf{V}_{\text{cache}} = [v_1, v_2, \ldots, v_{t-1}]^T$$

Then for the next token:

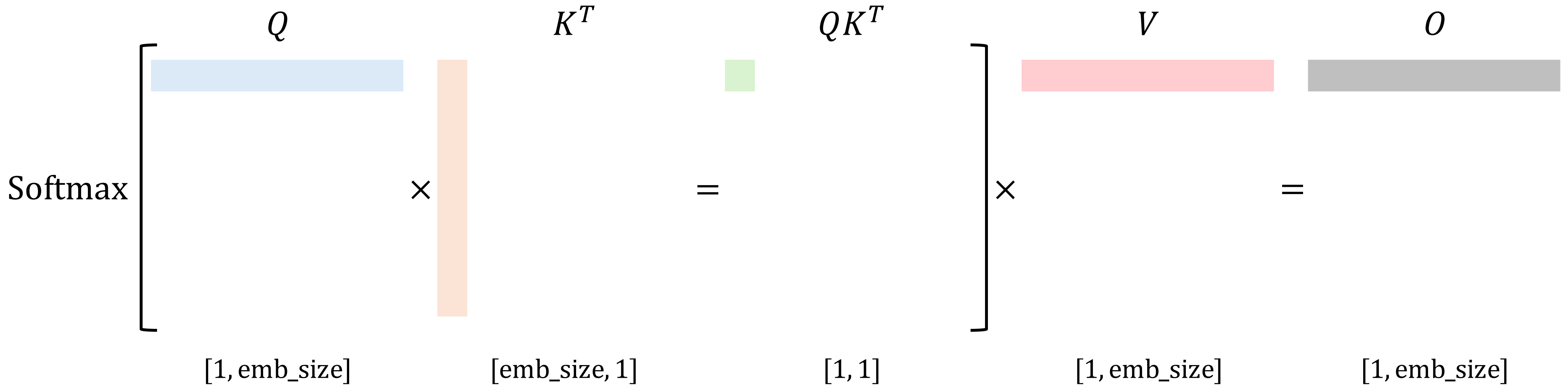
$$o_t = \text{Softmax}\left(\frac{q_t^T \cdot [\mathbf{K}_{\text{cache}}^T, k_t]}{\sqrt{d_k}}\right)[\mathbf{V}_{\text{cache}}, v_t]^T$$

Complexity per step:

$$\mathcal{O}(t^2) \rightarrow \mathcal{O}(t)$$

Scaled dot-product Attention w/ KV-Cache

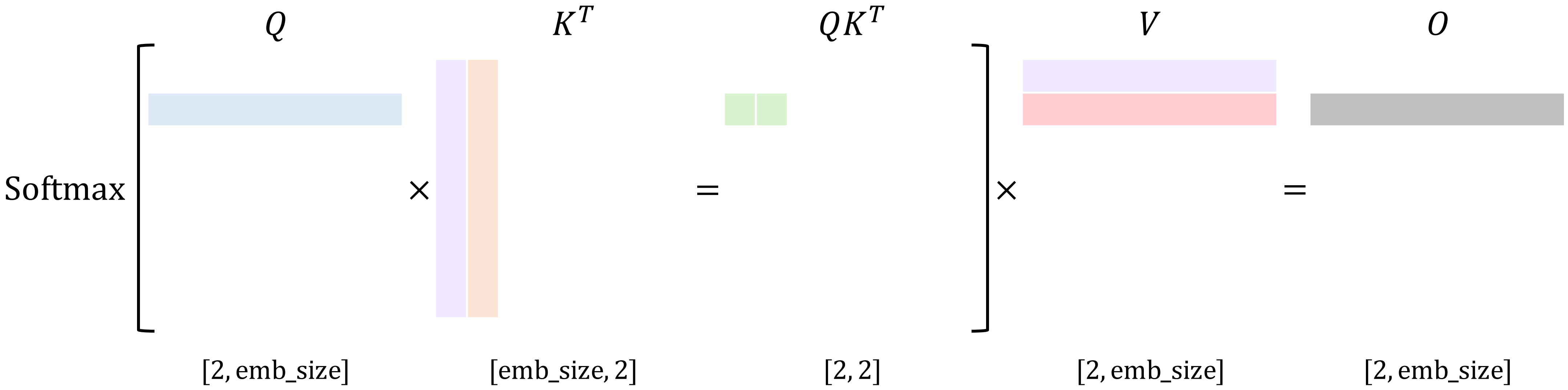
$$\mathrm{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers

Scaled dot-product Attention w/ KV-Cache

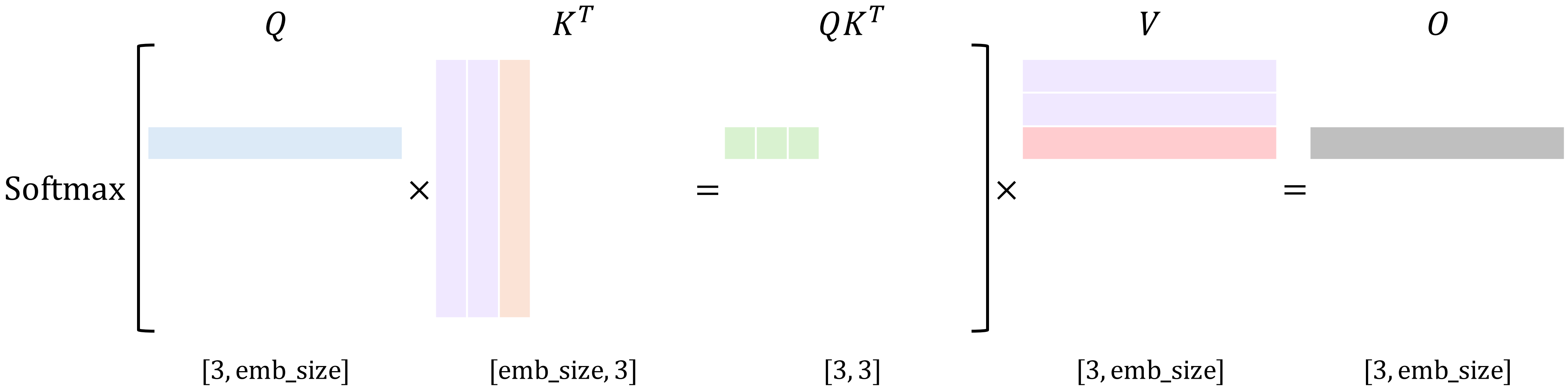
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict

Scaled dot-product Attention w/ KV-Cache

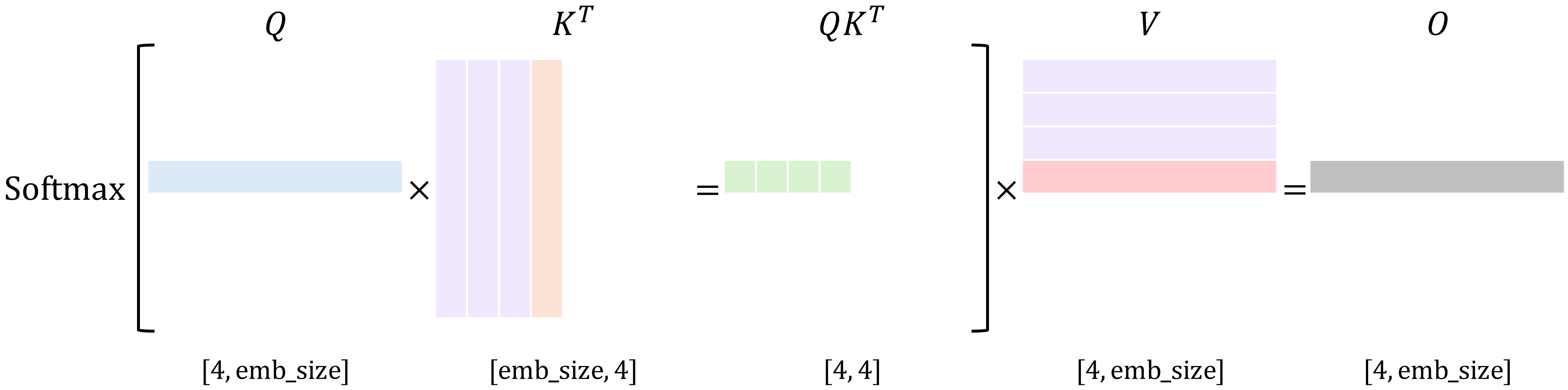
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow’s

Scaled dot-product Attention w/ KV-Cache

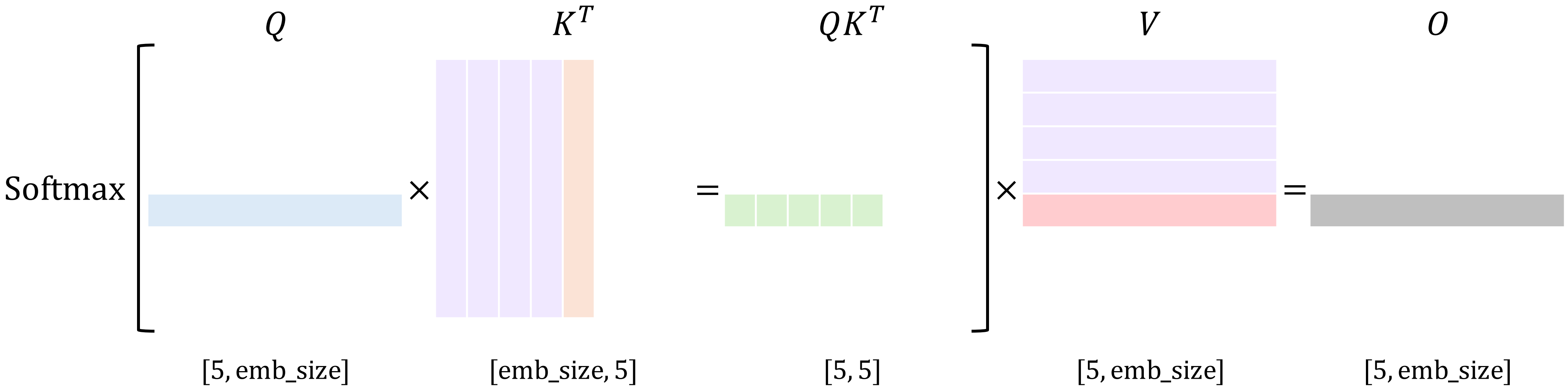
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow’s words

Scaled dot-product Attention w/ KV-Cache

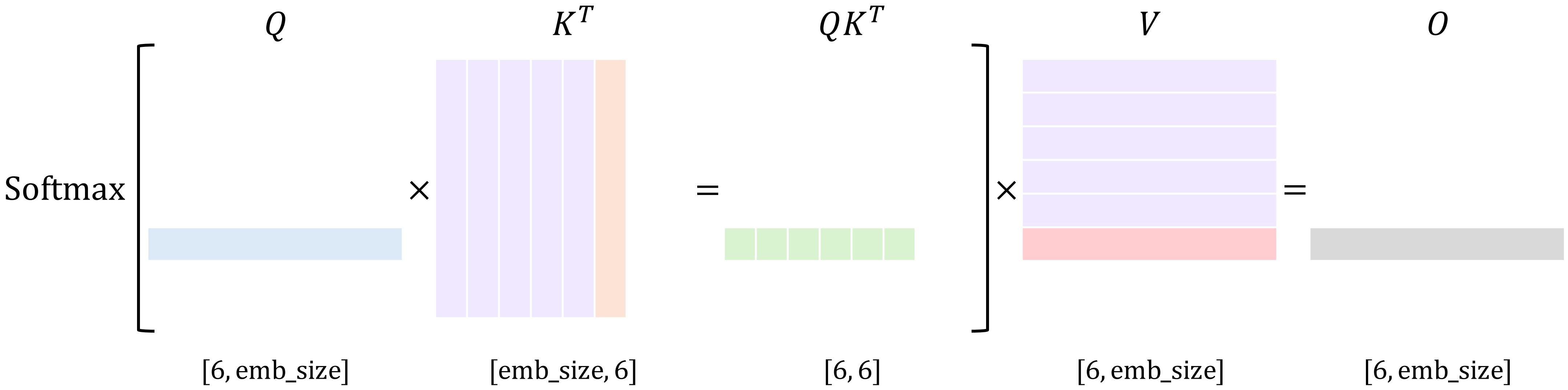
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow's words from

Scaled dot-product Attention w/ KV-Cache

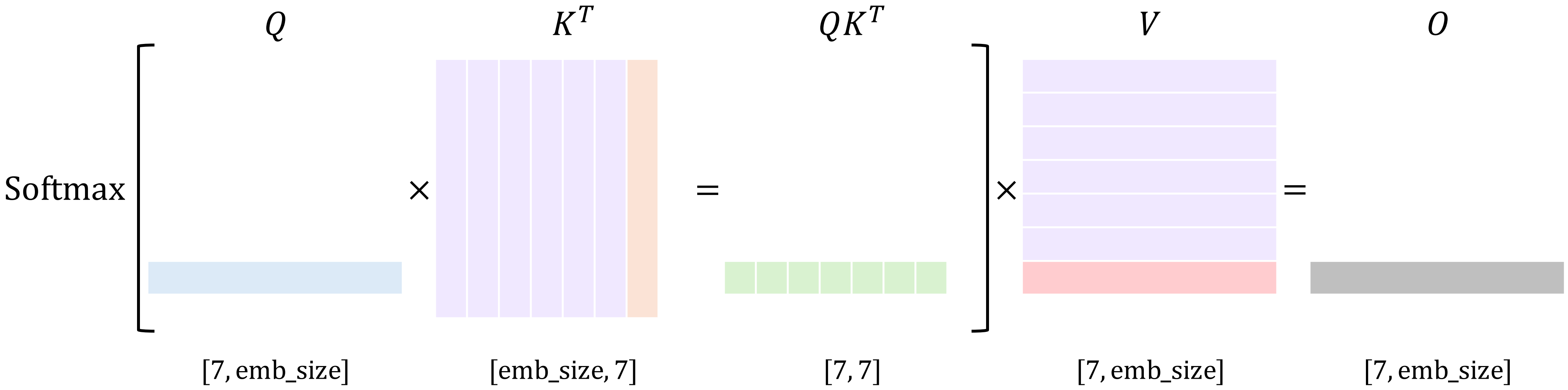
$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow's words from today's

Scaled dot-product Attention w/ KV-Cache

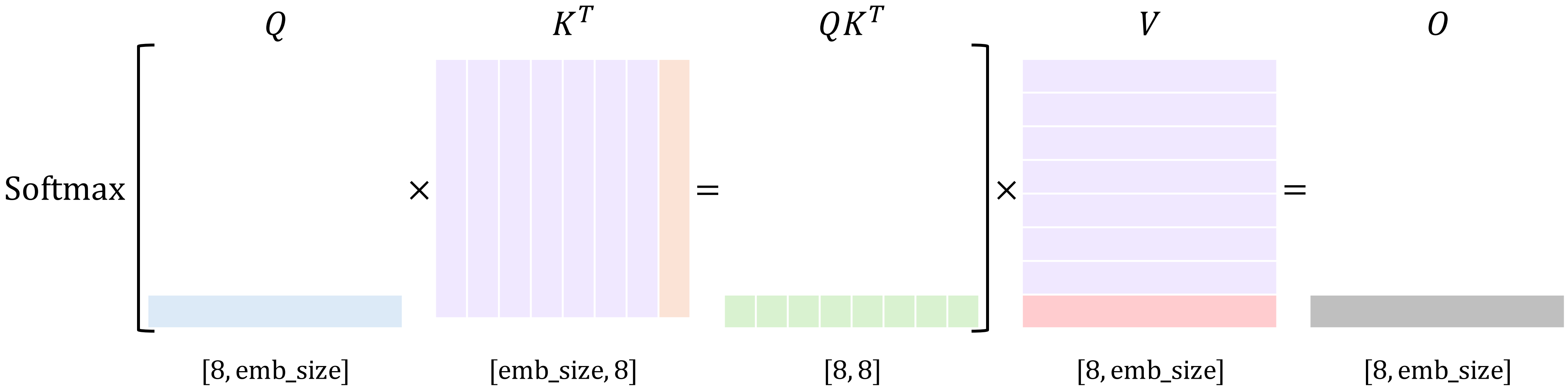
$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow’s words from today’s hidden

Scaled dot-product Attention w/ KV-Cache

$$\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$



<bos> Transformers predict tomorrow's words from today's hidden context

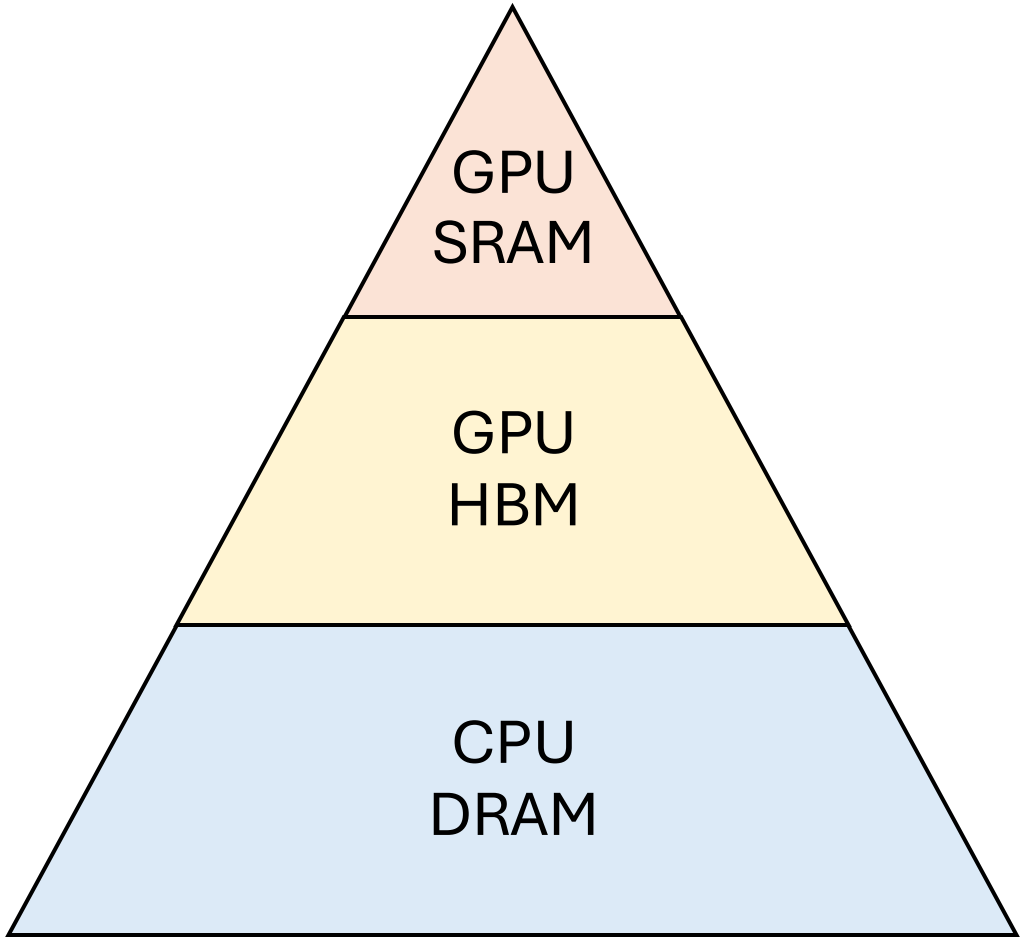
Implementation Example

```
from transformers import AutoModelForCausalLM, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("HuggingFaceTB/SmolLM2-1.7B")
model = AutoModelForCausalLM.from_pretrained("HuggingFaceTB/SmolLM2-1.7B").cuda()

tokens = tokenizer.encode("The red cat was", return_tensors="pt").cuda()
output = model.generate(
    tokens, max_new_tokens=300, use_cache=True # by default is set to True
)
output_text = tokenizer.batch_decode(output, skip_special_tokens=True)[0]
```

With KV-Cache	Standard Inference	Speedup
11.7 s	1 min 1 s	~5.21x times faster

Flash Attention

Memory Types



Bandwidth	Size
19 TB/s	20 MB
1.5 TB/s	40 GB
12.8 GB/s	> 1 TB

Attention Step-by-Step

$$\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

$$S = QK^T$$

$$A = \mathrm{Softmax}(S)$$

$$O = AV$$

Why is Self-Attention slow?

High
Bandwidth
Memory
(HBM)

$$S = QK^T$$

$$A = \text{Softmax}(S)$$

$$O = AV$$

Why is Self-Attention slow?

High
Bandwidth
Memory
(HBM)

Load Q, K by blocks from HBM

Write S to HBM

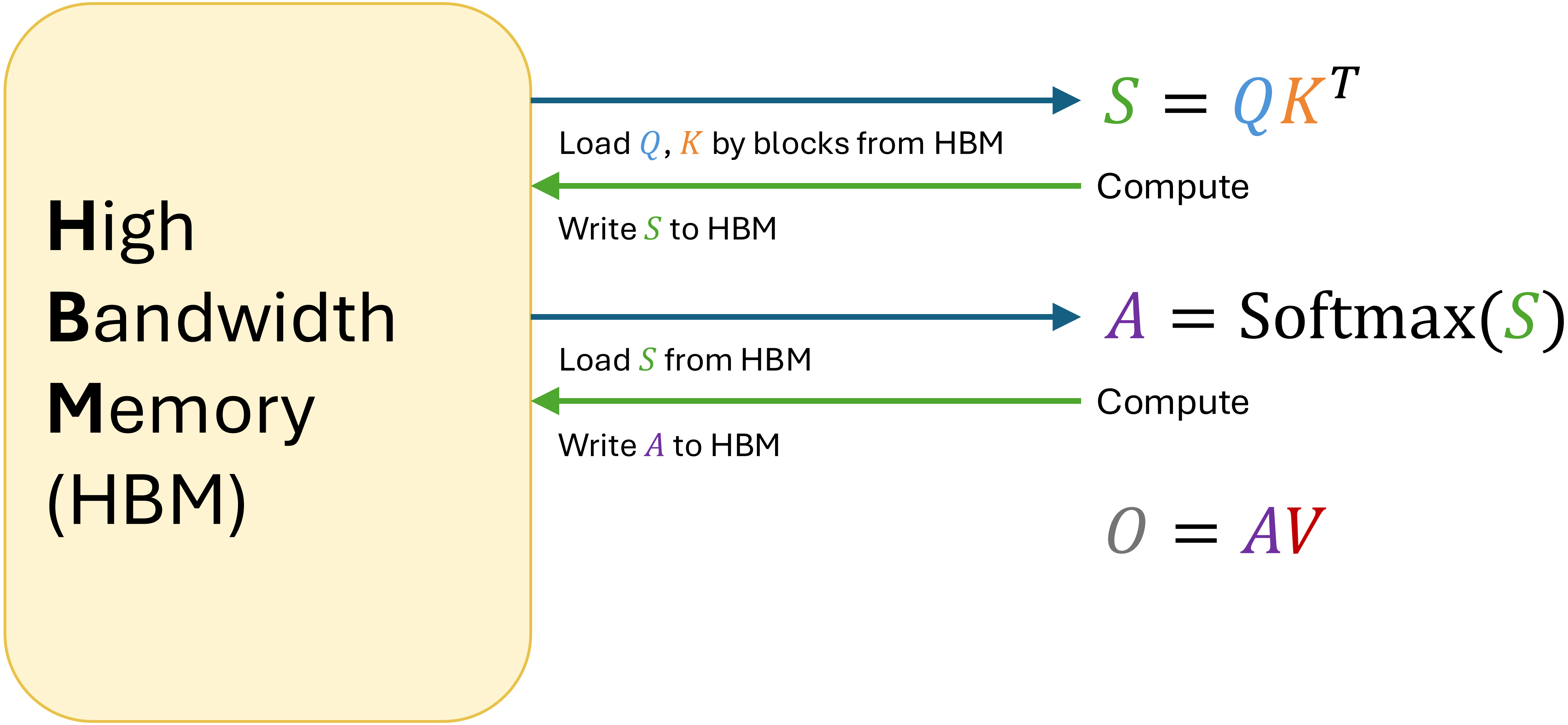
$$S = QK^T$$

Compute

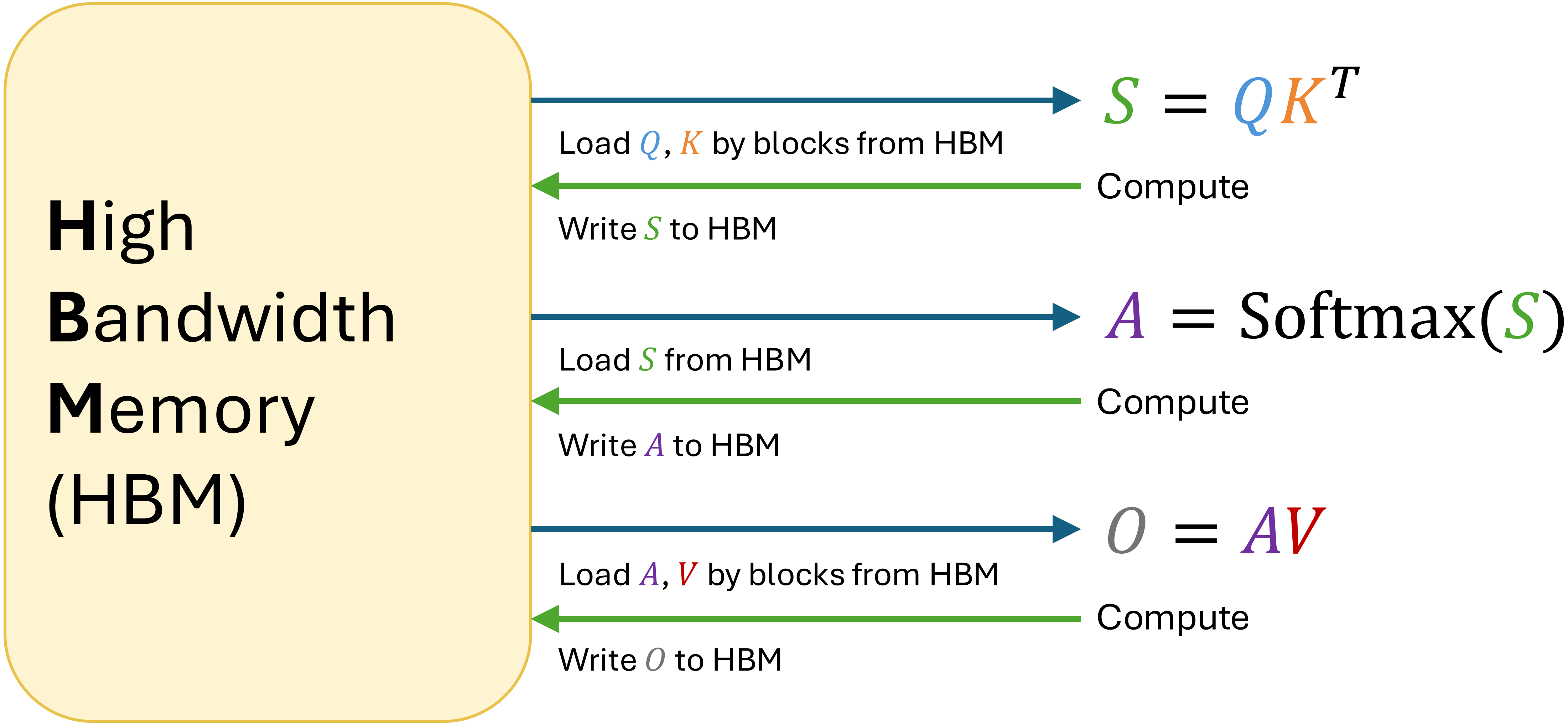
$$A = \mathrm{Softmax}(S)$$

$$O = AV$$

Why is Self-Attention slow?



Why is Self-Attention slow?



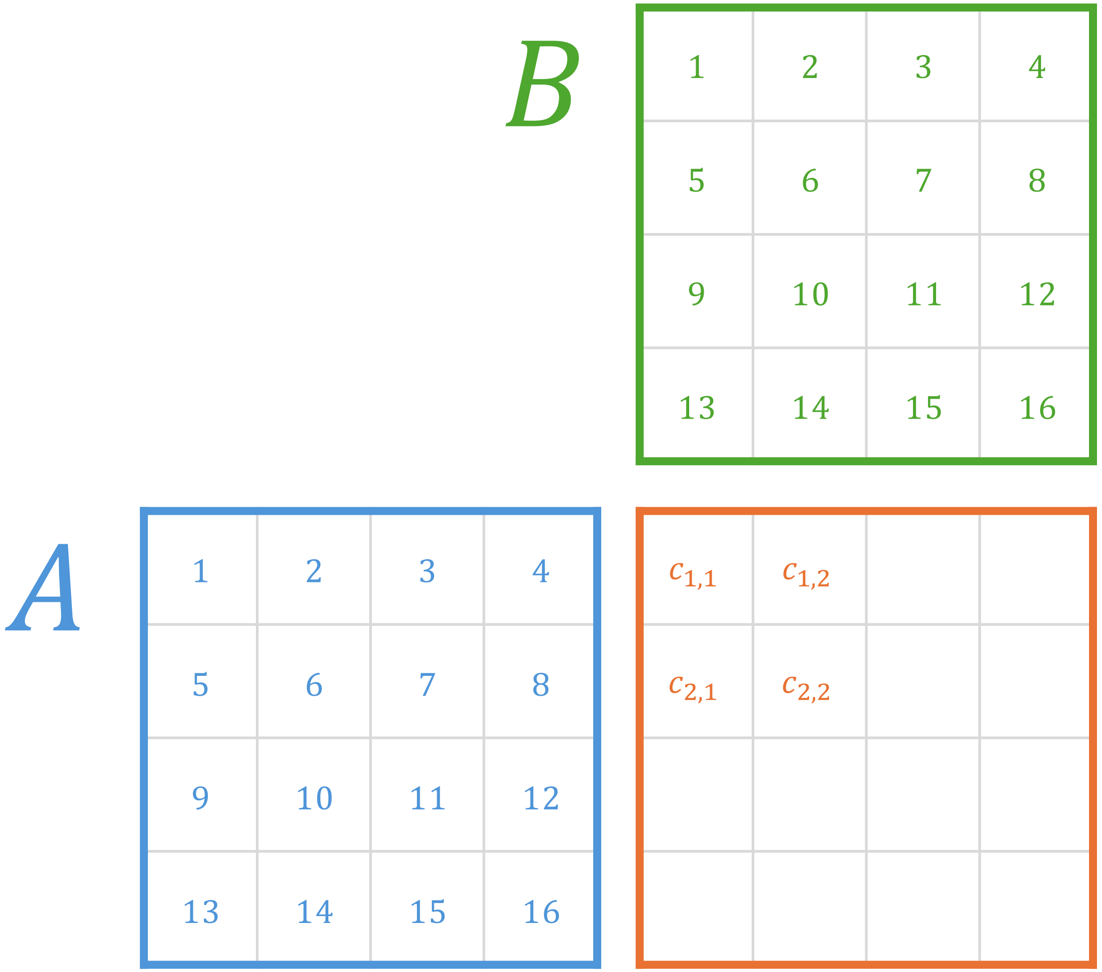
IO-aware Algorithm – Tiling

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \qquad \begin{bmatrix} c_{1,1} & c_{1,2} & & \\ c_{2,1} & c_{2,2} & & \\ & & & \\ & & & \end{bmatrix}$$

$$C = A \times B$$

IO-aware Algorithm – Tiling



$$C_{1,1} = 1 \times 1 + 2 \times 5 + 3 \times 9 + 4 \times 13$$

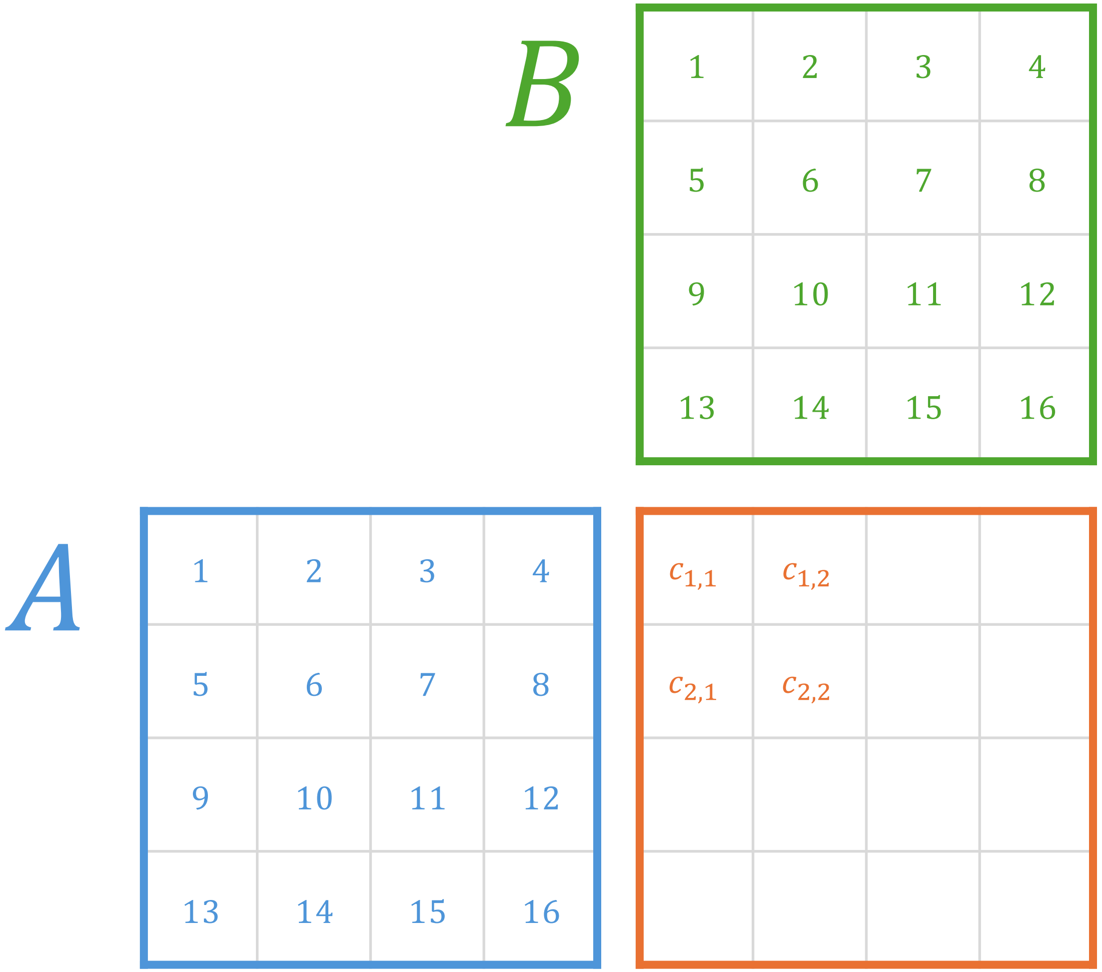
$$C_{1,2} = 1 \times 2 + 2 \times 6 + 3 \times 10 + 4 \times 14$$

$$C_{2,1} = 5 \times 1 + 6 \times 5 + 7 \times 9 + 8 \times 13$$

$$C_{2,2} = 5 \times 2 + 6 \times 6 + 7 \times 10 + 8 \times 14$$

$$C = A \times B$$

IO-aware Algorithm – Tiling



Without tiling: 32 memory accesses

$$C_{1,1} = 1 \times 1 + 2 \times 5 + 3 \times 9 + 4 \times 13$$

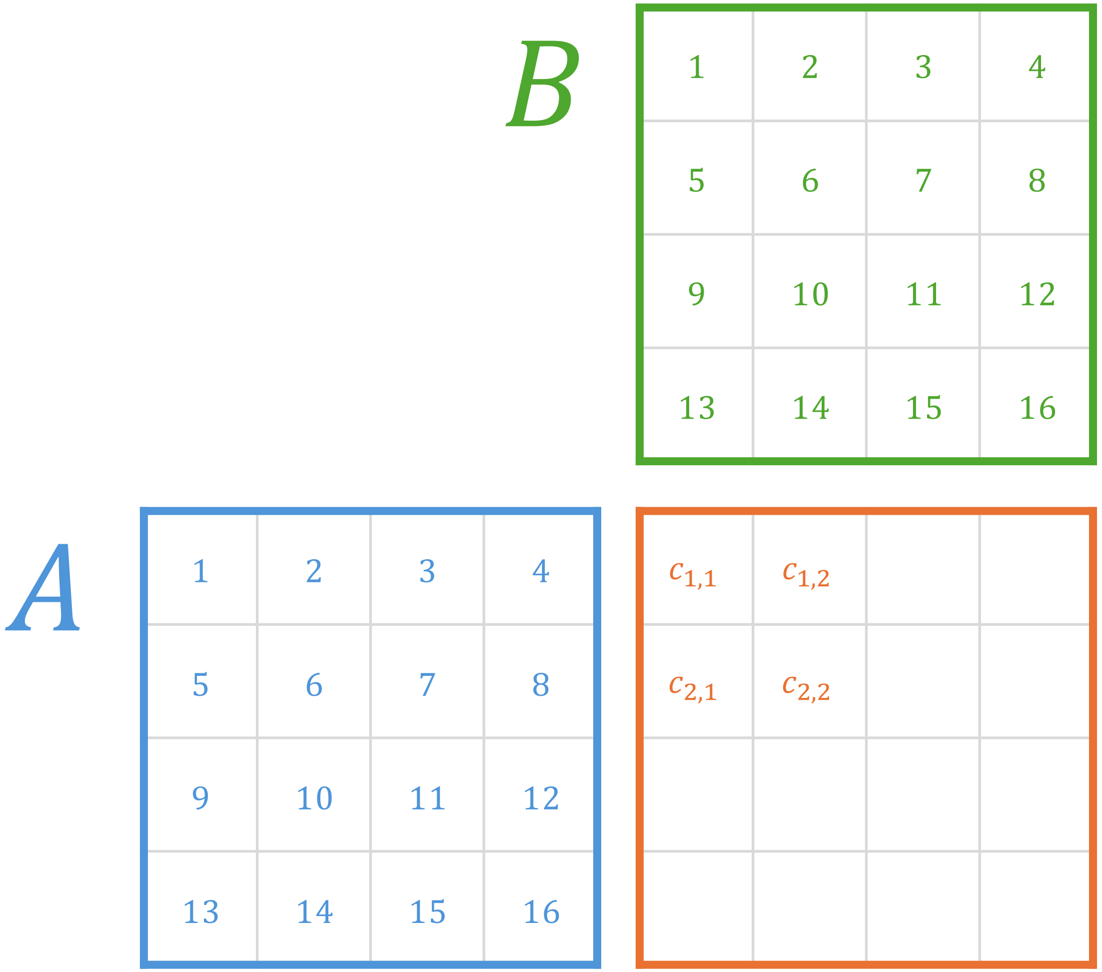
$$C_{1,2} = 1 \times 2 + 2 \times 6 + 3 \times 10 + 4 \times 14$$

$$C_{2,1} = 5 \times 1 + 6 \times 5 + 7 \times 9 + 8 \times 13$$

$$C_{2,2} = 5 \times 2 + 6 \times 6 + 7 \times 10 + 8 \times 14$$

$$C = A \times B$$

IO-aware Algorithm – Tiling



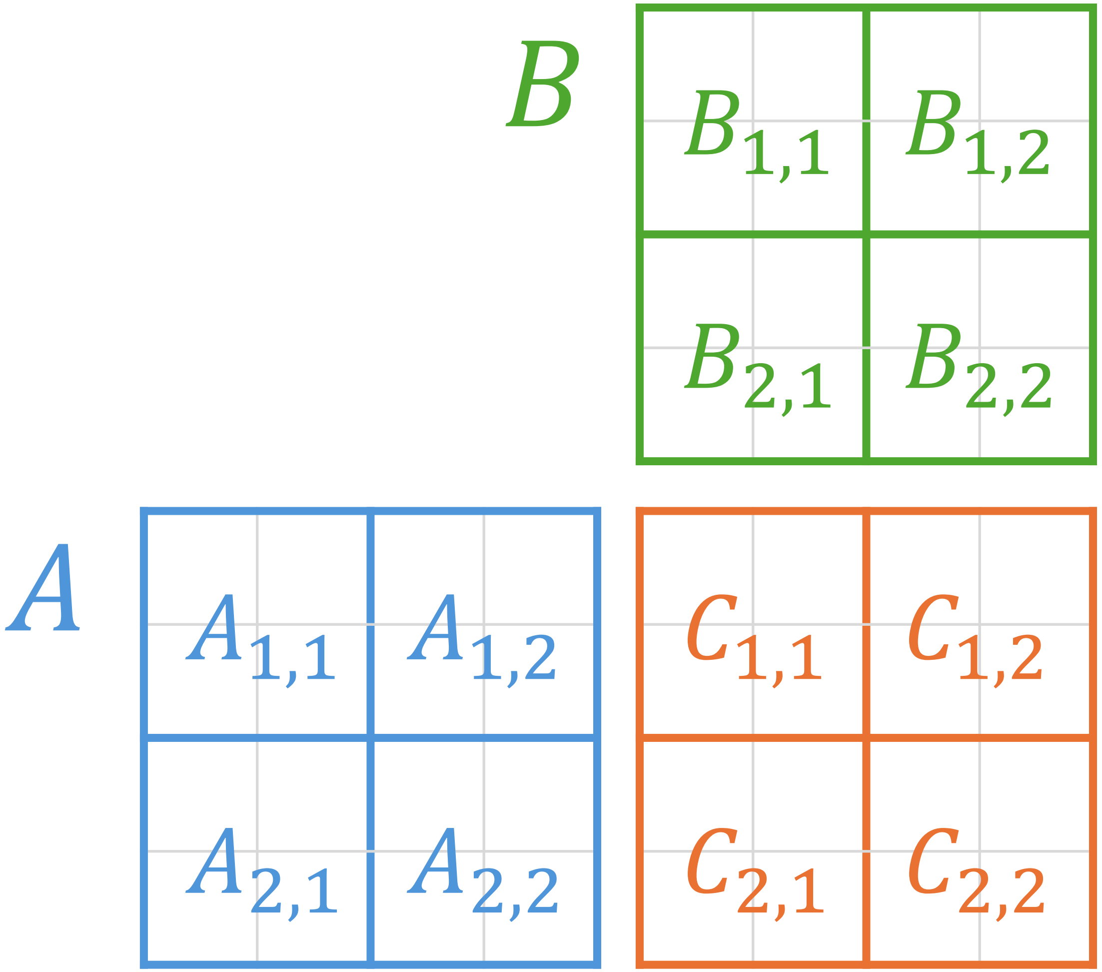
Without tiling: 32 memory accesses
With tiling: 16 memory accesses

$N \times N$ block $\rightarrow 1/N$ memory access

$$\begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 9 & 10 \\ 13 & 14 \end{bmatrix}$$

$$C = A \times B$$

IO-aware Algorithm – Tiling



$$C = A \times B$$

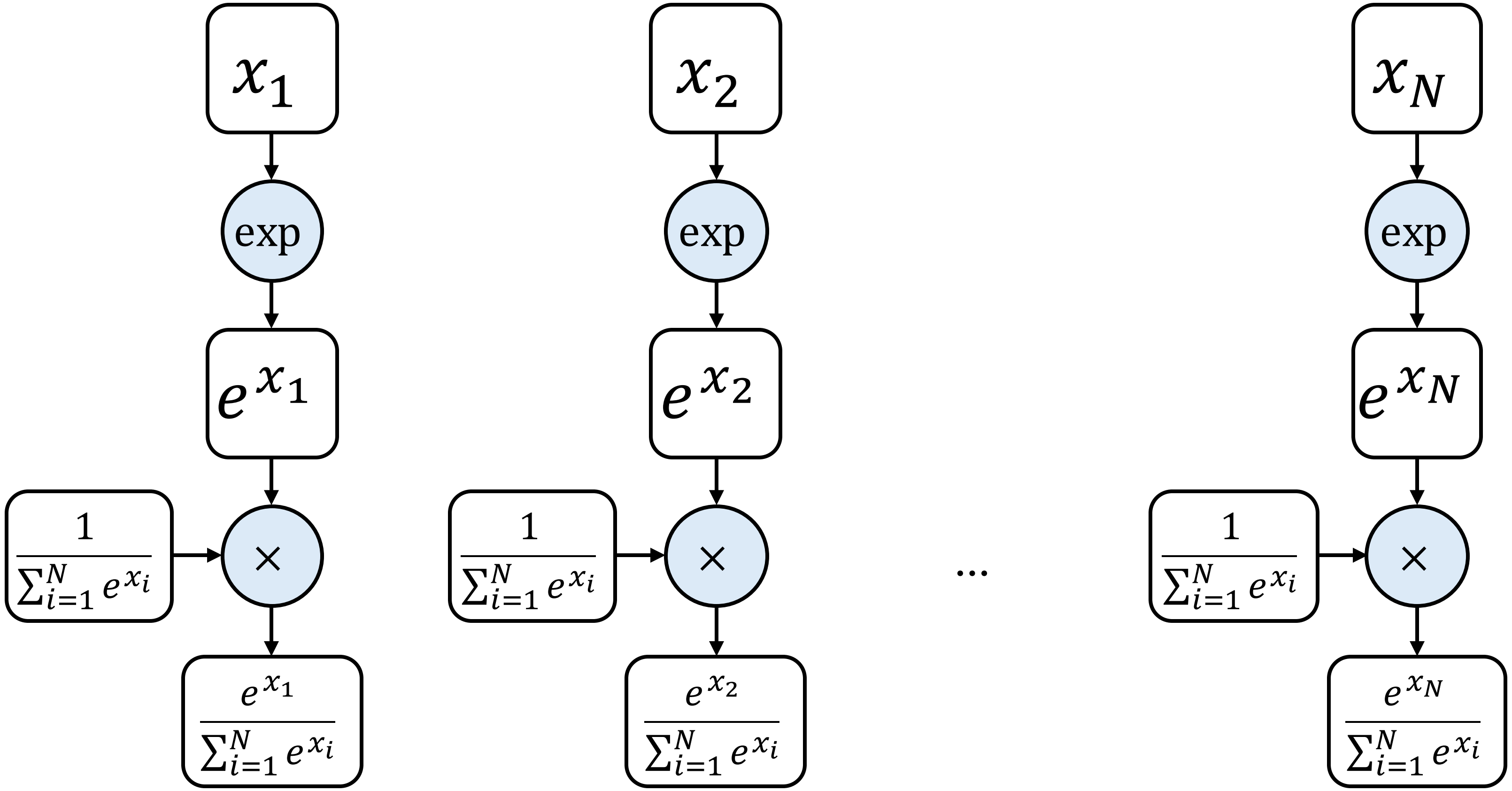
How to make efficient?

$$S = QK^T$$

$$A = \mathrm{Softmax}(S)$$

$$O = AV$$

Softmax

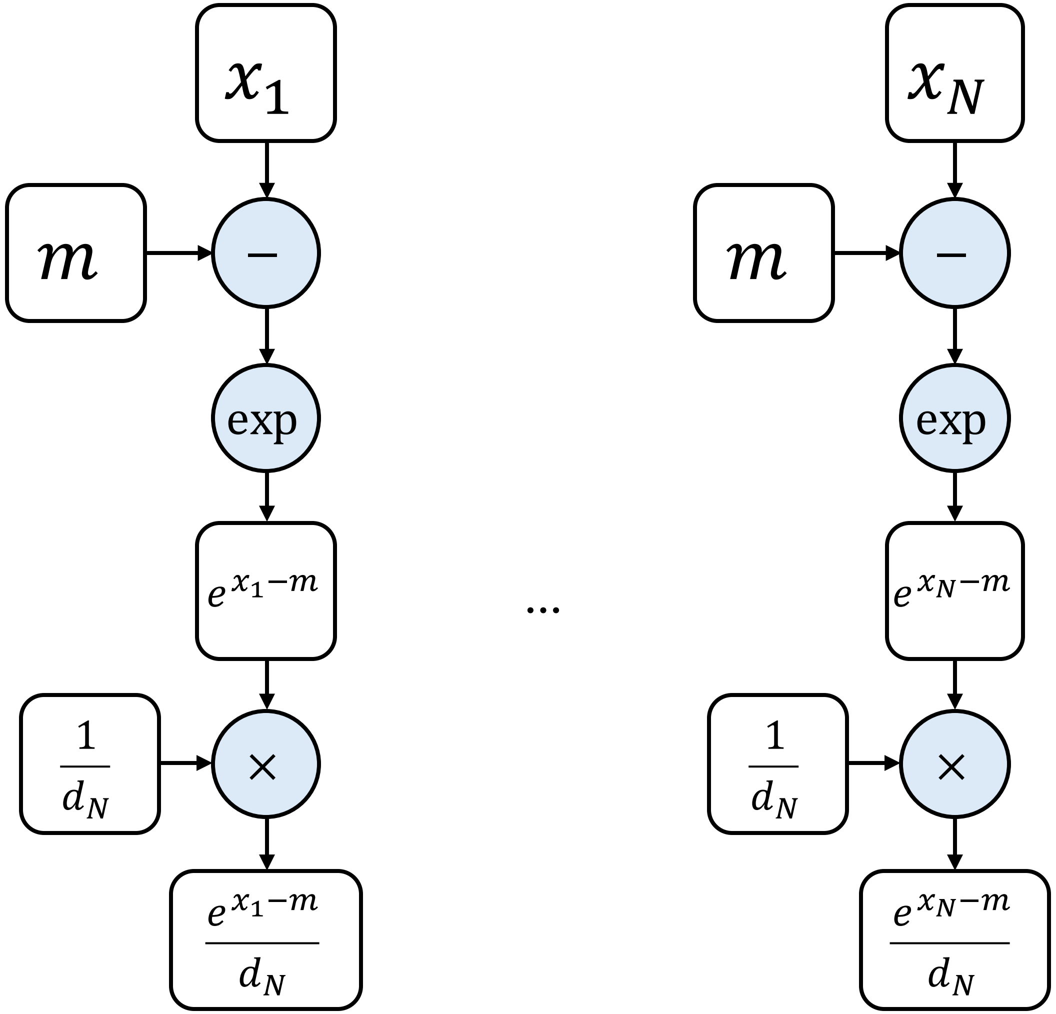


Runtime Warning: overflow encountered in exp y = torch.exp(x)

Safe Softmax

$$m = max(x_1, x_2, \dots, x_N)$$

$$d_N = \sum_{i=1}^{N} e^{x_i - m}$$



Safe Softmax

$$m_N = max(x_1, x_2, \dots, x_N)$$

$$d_N = \sum_{i=1}^{N} e^{x_i - m_N}$$

$m_0 = -\infty$

for $i = 1, \dots, N$ **do**

$\quad m_i = \max(m_{i-1}, x_i)$

$d_0 = 0$

for $i = 1, \dots, N$ **do**

$\quad d_i = d_{i-1} + e^{x_i - m_N}$

for $i = 1, \dots, N$ **do**

$\quad a_i = e^{x_i - m_N} / d_N$

3 loops

Online Softmax

$m_0 = -\infty$

for $i = 1, \ldots, N$ **do**

$\quad m_i = \max(m_{i-1}, x_i)$

$d_0 = 0$

for $i = 1, \ldots, N$ **do**

$\quad d_i = d_{i-1} + e^{x_i - m_N}$

for $i = 1, \ldots, N$ **do**

$\quad a_i = e^{x_i - m_N} / d_N$

$$d_i = \sum_{j=1}^{i} e^{x_j - m_N} \qquad d'_i = \sum_{j=1}^{i} e^{x_j - m_i} \qquad d'_N = d_N$$

$$d'_i = \underbrace{\left(\sum_{j=1}^{i} e^{x_j - m_{i-1}}\right)}_{d'_{i-1}} e^{m_{i-1} - m_i} + e^{x_i - m_i}$$

Online Softmax

$m_0 = -\infty$

for $i = 1, \dots, N$ **do**

$\quad m_i = \max(m_{i-1}, x_i)$

$d_0 = 0$

for $i = 1, \dots, N$ **do**

$\quad d_i = d_{i-1} + e^{x_i - m_N}$

for $i = 1, \dots, N$ **do**

$\quad a_i = e^{x_i - m_N} / d_N$

2 loops

$m_0 = -\infty$

$d_0 = 0$

for $i = 1, \dots, N$ **do**

$\quad m_i = \max(m_{i-1}, x_i)$

$\quad d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$

for $i = 1, \dots, N$ **do**

$\quad a_i = e^{x_i - m_N} / d'_N$

Online Softmax

$m_0 = -\infty$

$d_0 = 0$

for $i = 1, \ldots, N$ **do**

$\quad x_i = qk_i^T$

$\quad m_i = \max(m_{i-1}, x_i)$

$\quad d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$

$o_0 = 0$

for $i = 1, \ldots, N$ **do**

$\quad a_i = e^{x_i - m_N} / d'_N$

$\quad o_i = o_{i-1} + a_i v_i$

$$S = QK^T$$

$$A = \text{Softmax}(S)$$

$$O = AV$$

Online Softmax

$m_0 = -\infty$

$d_0 = 0$

for $i = 1, \dots, N$ **do**

$x_i = q k_i^T$

$m_i = \max(m_{i-1}, x_i)$

$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$

$o_0 = 0$

for $i = 1, \dots, N$ **do**

$a_i = e^{x_i - m_N} / d'_N$

$o_i = o_{i-1} + a_i v_i$

$S = QK^T$

$A = \text{Softmax}(S)$

$O = AV$

We can do better!

Let's do the same trick:

$$o_i = \sum_{j=1}^{i} \frac{e^{x_j - m_N}}{d'_N} v_j \longrightarrow o'_i = \sum_{j=1}^{i} \frac{e^{x_j - m_i}}{d'_i} v_j$$

$$o_N = o'_N$$

Flash Attention ⚡

$$m_0 = -\infty$$
$$d_0 = 0$$
$$o_0 = 0$$

for $i = 1, \ldots, N$ **do**

$$x_i = q k_i^T$$
$$m_i = \max(m_{i-1}, x_i)$$
$$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$$
$$o'_i = o'_{i-1} \frac{d'_{i-1}}{d'_i} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d'_i} v_i$$

return o'_N

$$S = QK^T$$
$$A = \text{Softmax}(S)$$
$$O = AV$$

Flash Attention ⚡

for $i = 1, \dots, N$ **do**

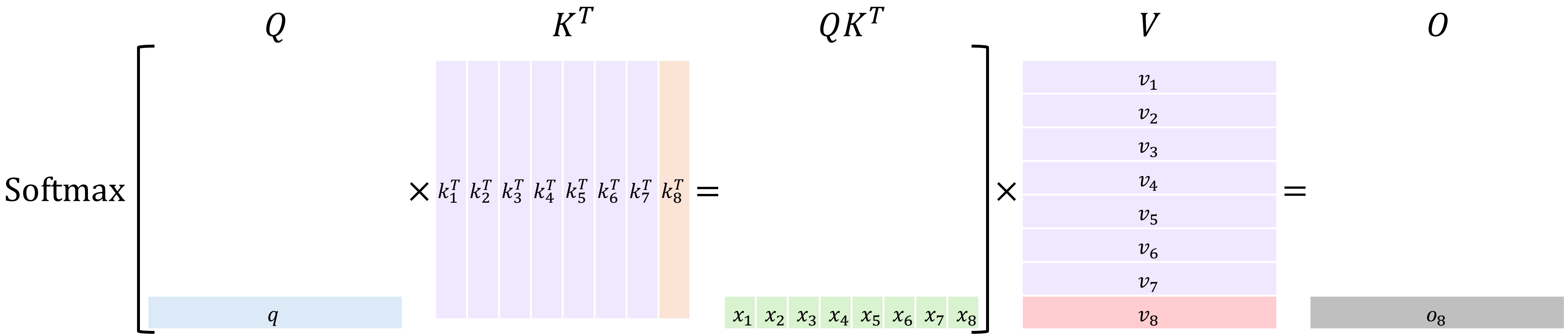
$$x_i = q k_i^T$$

$$m_i = \max(m_{i-1}, x_i)$$

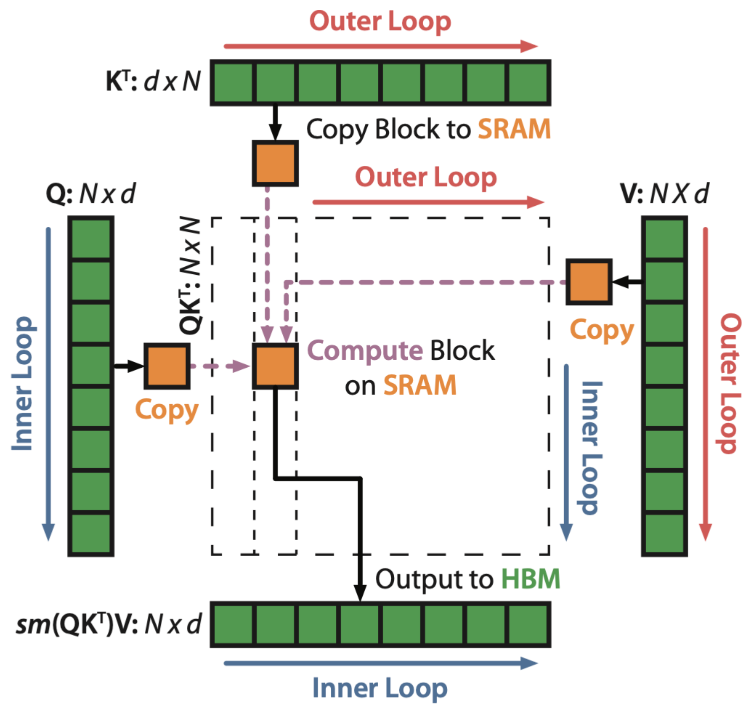
$$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$$

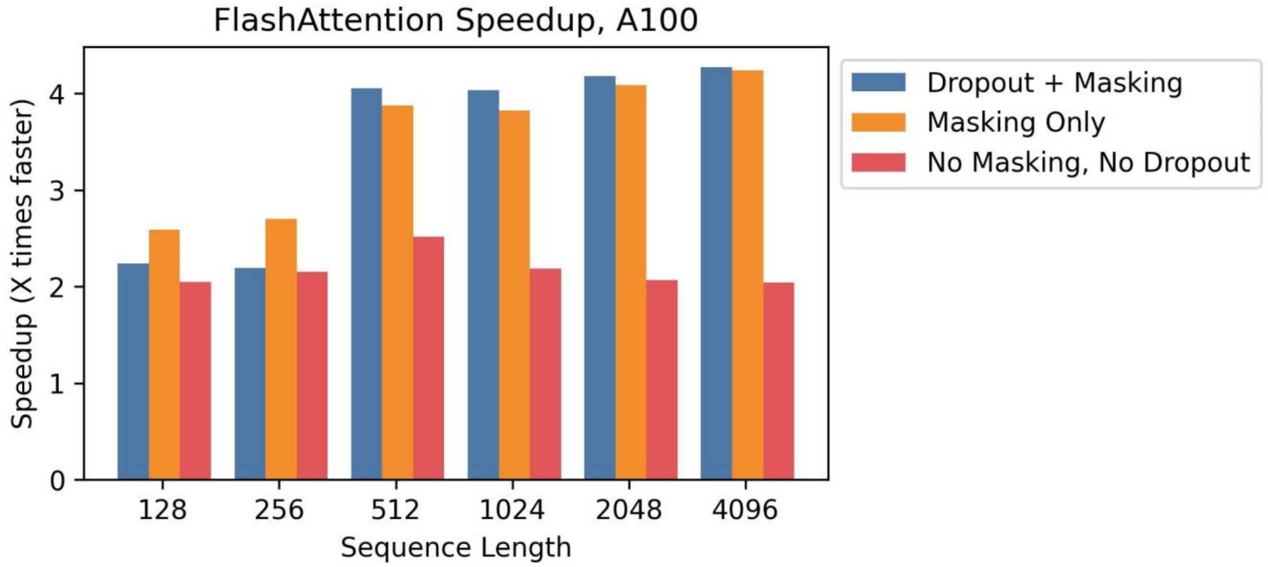
$$o'_i = o'_{i-1} \frac{d'_{i-1}}{d'_i} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d'_i} v_i$$

return o'_N



Flash Attention ⚡





Dao T. et al. FlashAttention: Fast and memory-efficient exact attention with io-awareness, 2022

Recap

- Quantization
- Pruning
- Distillation
- KV-Cache
- Flash Attention

Course Overview

- Introduction to Neural Networks
  - MLP
  - Backpropagation
  - Initialization
  - Regularization
  - CNN
- Natural Language Processing
  - Word Embeddings
  - RNN, LTSM
  - Attention
  - Transformer
- Computer Vision
  - Image Classification
  - Object Detection
  - Segmentation
- Reinforcement Learning
  - Multi-armed Bandits
  - Monte Carlo Methods
  - Policy Improvement
- Generative Models
  - Autoregression
  - VAE
  - GAN
  - Diffusion
  - Flow Matching
- Multimodality
  - CLIP
  - BLIP
  - VLM
- Acceleration
  - Quantization
  - Pruning
  - Distillation
  - KV-Cache
  - Flash Attention

Game Rules

- 5 Homeworks = **70 points**
- Oral Exam = **30 points**
- Maximum Points: 70 + 30 = **100 points**

Final Grade: $\min(\mathrm{round}(\#\mathrm{points}, 10), 10)$

Thanks for your $\mathrm{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$