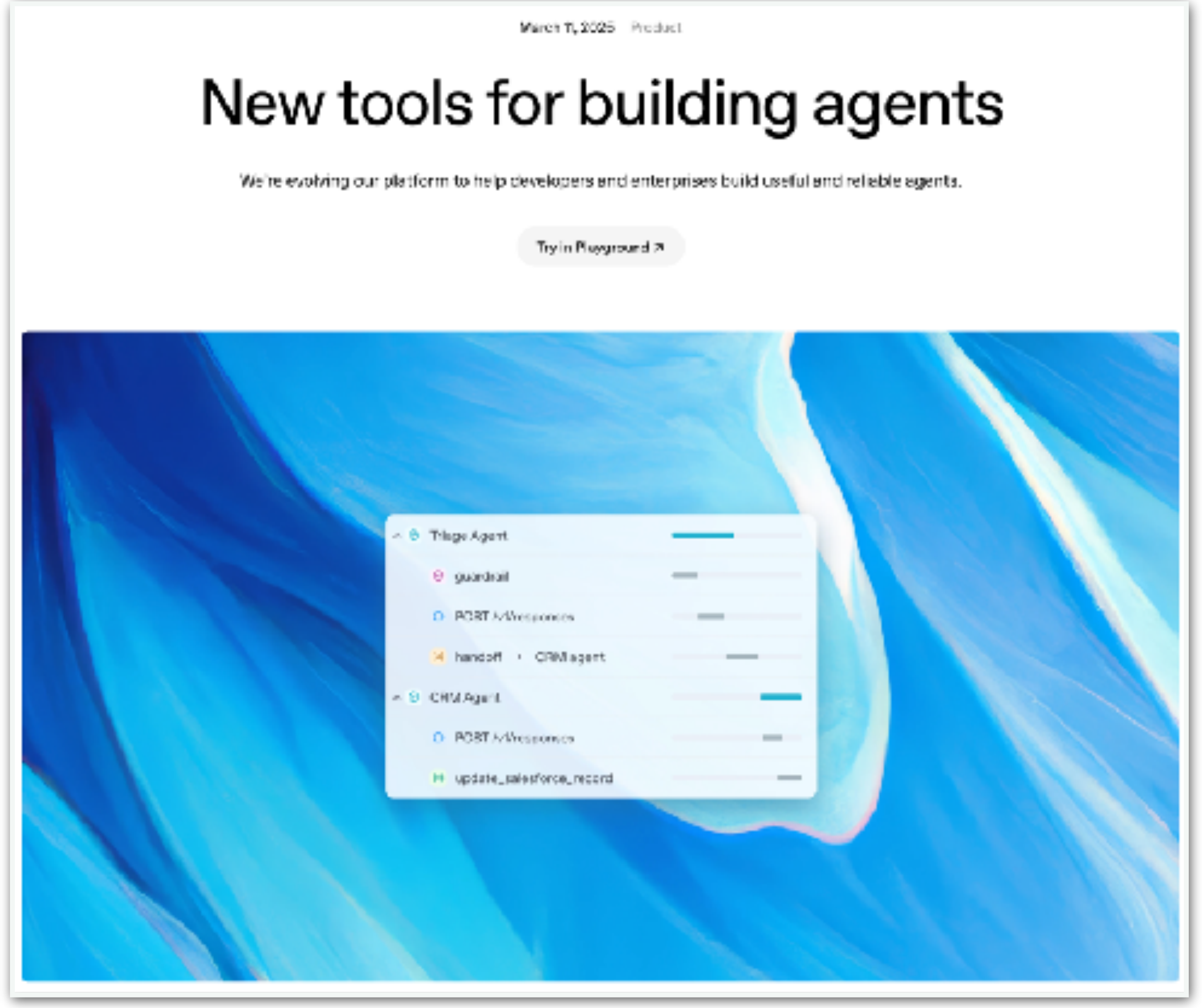
COMP 3361 Natural Language Processing

Lecture 12: Attention and Transformers (cont.)

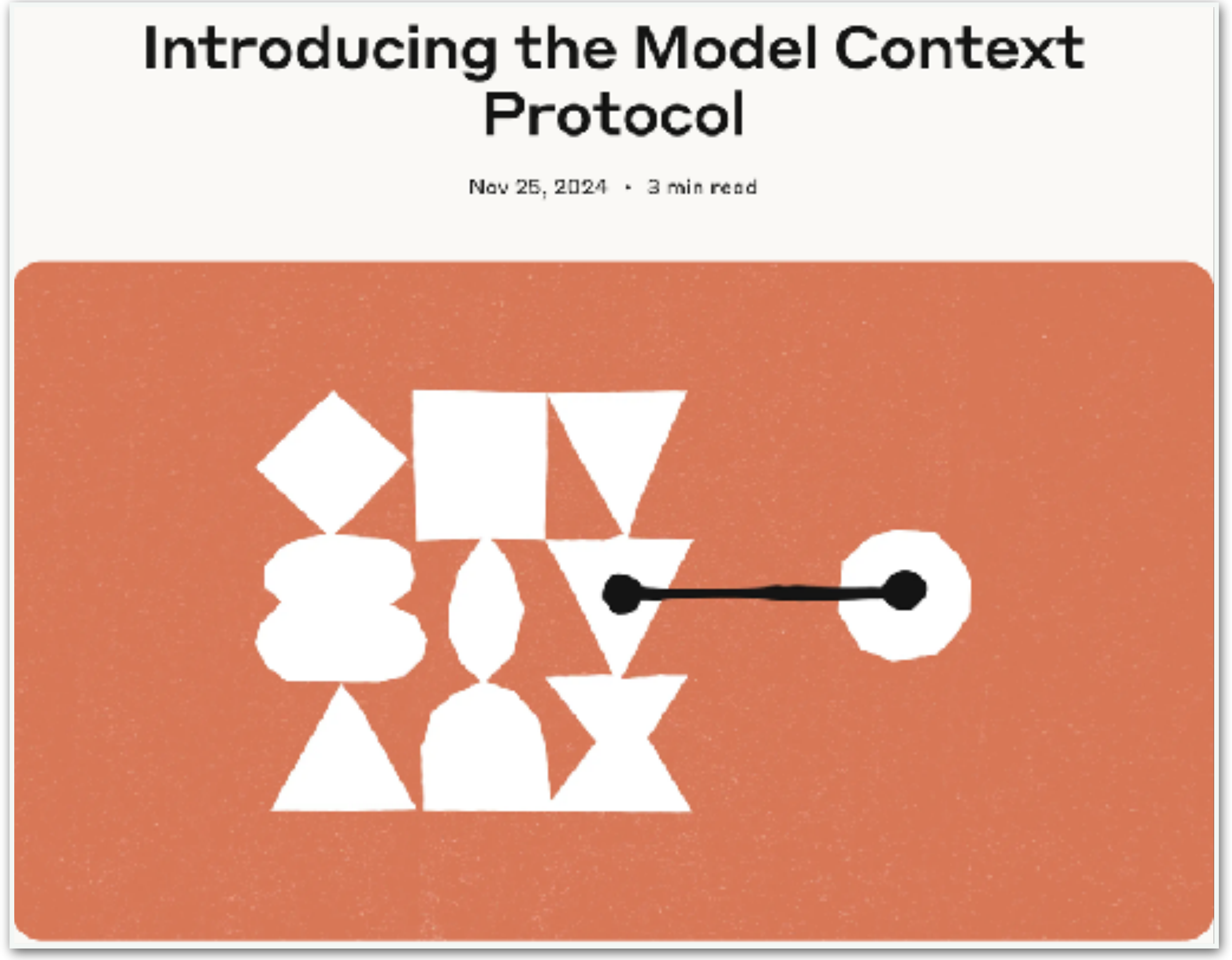
Spring 2025

Many materials from CSE447@UW (Liwei Jiang), COS 484@Princeton, and CS224n@Stanford with special thanks!

Latest AI news



[OpenAI Agents SDK](https://openai.github.io/openai-agents-python/)

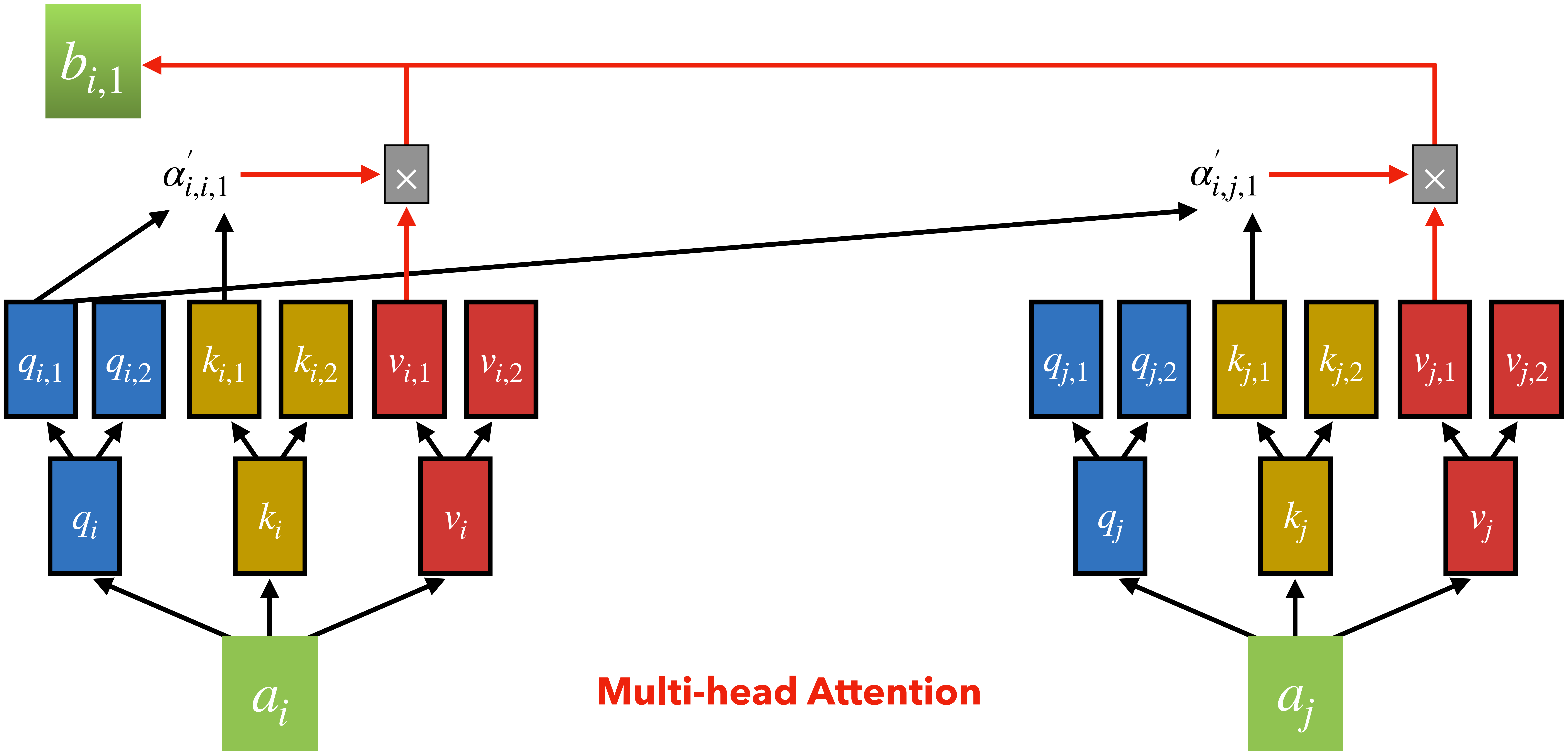


[Anthropic Model Context Protocol](https://norahsakal.com/blog/mcp-vs-api-model-context-protocol-explained)

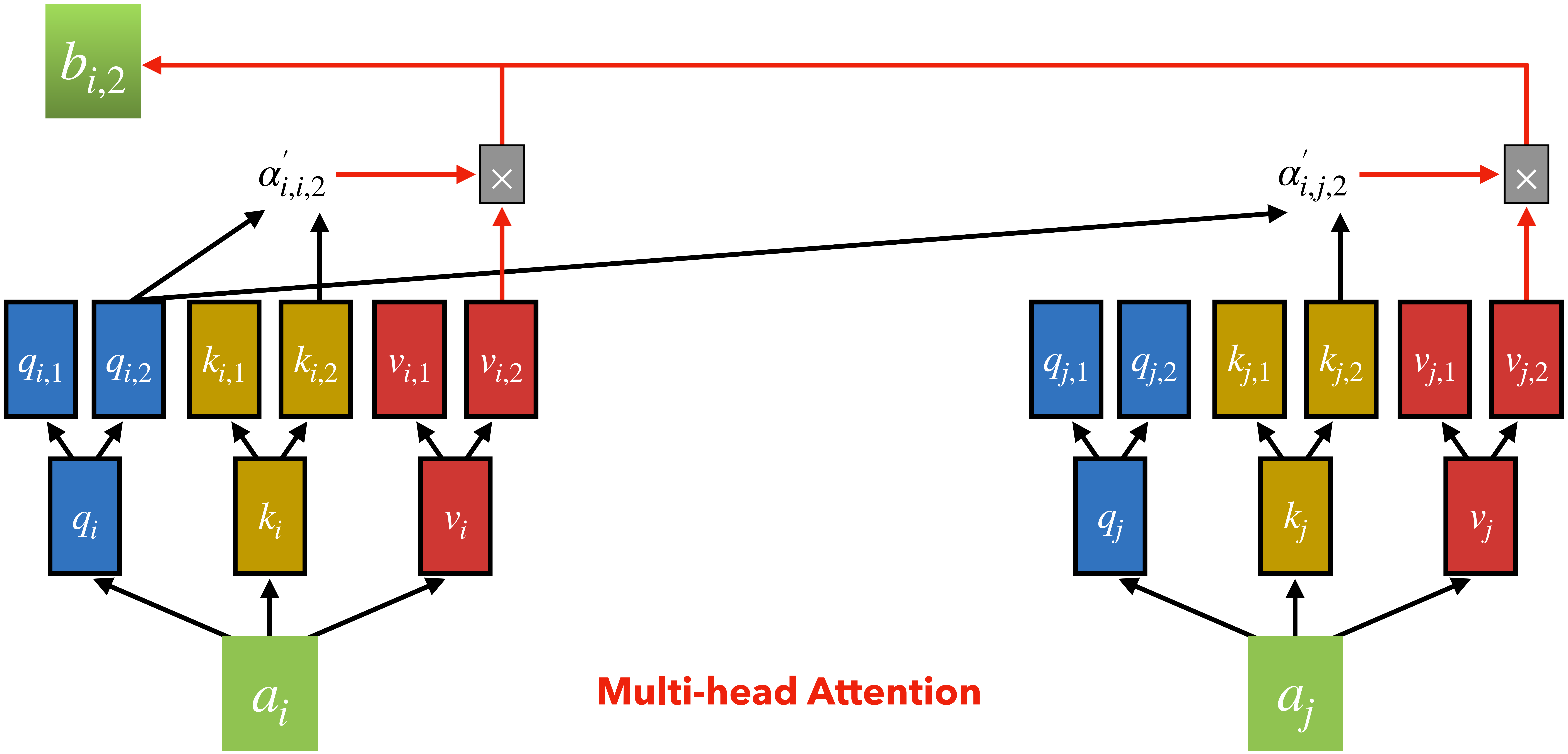
Google Gemini Robotics

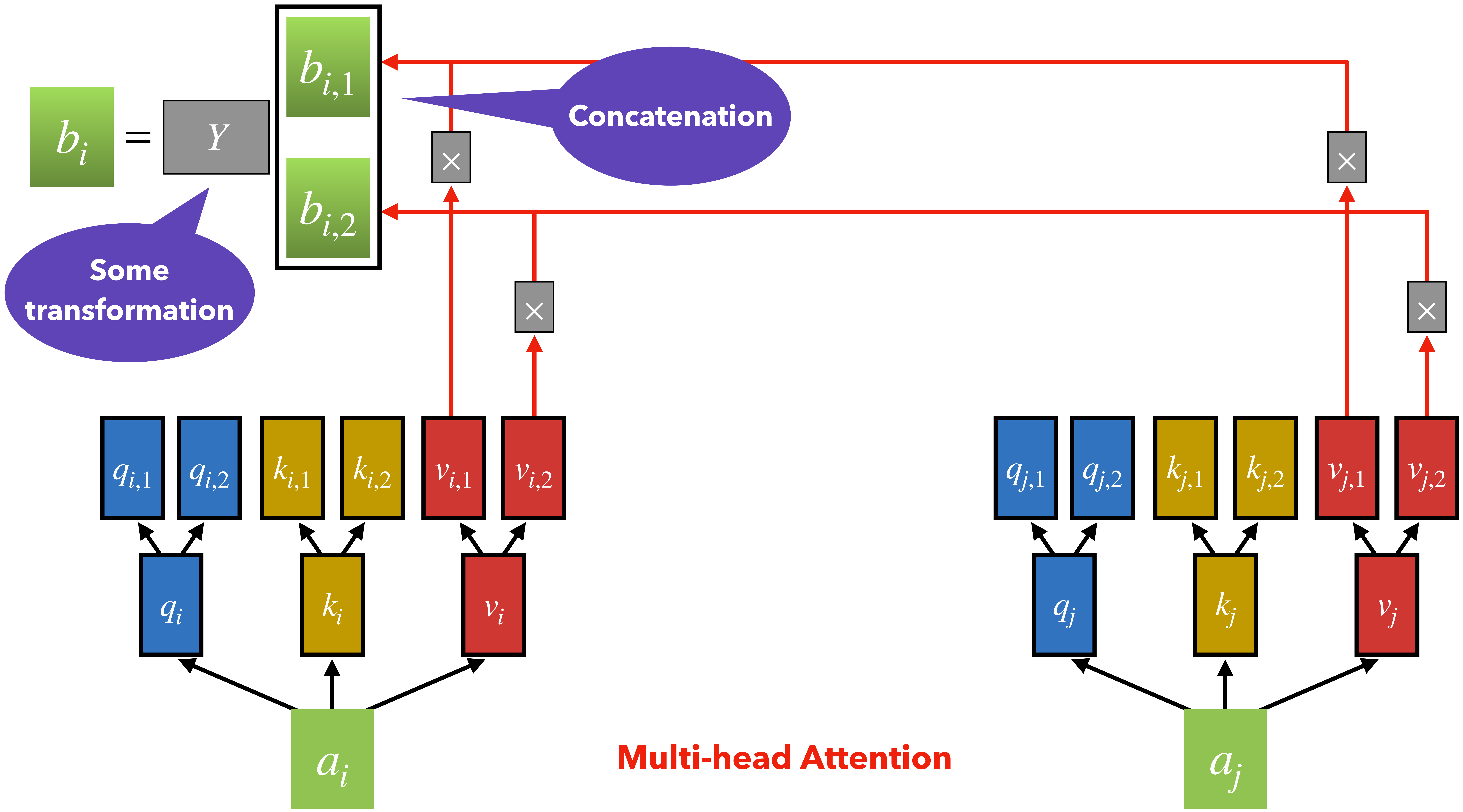
https://openai.github.io/openai-agents-python/
https://norahsakal.com/blog/mcp-vs-api-model-context-protocol-explained

Multi-Head Attention: Walk-through



Multi-Head Attention: Walk-through





b_i = Y
$b_{i,1}$
$b_{i,2}$
Concatenation
Some transformation
$q_{i,1}$ $q_{i,2}$ $k_{i,1}$ $k_{i,2}$ $v_{i,1}$ $v_{i,2}$
$q_{j,1}$ $q_{j,2}$ $k_{j,1}$ $k_{j,2}$ $v_{j,1}$ $v_{j,2}$
q_i k_i v_i
q_j k_j v_j
a_i
a_j
Multi-head Attention

Recall the Matrices Form of Self-Attention

$Q = I\ W_Q$

$K = I\ W_K$

$V = I\ W_V$

$I = \{a_1, \ldots, a_n\} \in \mathbb{R}^{n \times d}$, where $a_i \in \mathbb{R}^d$

$W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$

$Q, K, V \in \mathbb{R}^{n \times d}$

$A = Q\ K^T$

$A = I\ W_Q\ (I\ W_K)^T = I\ W_Q\ W_K^T\ I^T$

$A' = \text{softmax}(A)$

$A', A \in \mathbb{R}^{n \times n}$

$O = A'\ V$

$O \in \mathbb{R}^{n \times d}$

Multi-head Attention in Matrices

- Multiple attention "heads" can be defined via multiple W_Q, W_K, W_V matrices
- Let $W_Q^l, W_K^l, W_V^l \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and l ranges from 1 to h.
- Each attention head performs attention independently:
  - $O^l = \text{softmax}(I\ W_Q^l\ {W_K^l}^T\ I^T)\ I\ W_V^l$
- Concatenating different O^l from different attention heads.
  - $O = [O^1; \ldots; O^n]\ Y$, where $Y \in \mathbb{R}^{d \times d}$

The Matrices Form of Multi-head Attention

$Q^l = I\, W_Q^l$

$K^l = I\, W_K^l$

$V^l = I\, W_V^l$

- $I = \{a_1, \ldots, a_n\} \in \mathbb{R}^{n \times d}$, where $a_i \in \mathbb{R}^d$
- $W_Q^l, W_K^l, W_V^l \in \mathbb{R}^{d \times \frac{d}{h}}$
- $Q^l, K^l, V^l \in$?

$A^l = Q^l\, K^{l^T}$

$A^{l'} = \mathrm{softmax}(A^l)$

- $A^{l'}, A^l \in \mathbb{R}$?

$O^l = A^{l'}\, V^l$

- $O^l \in \mathbb{R}$?

$O = [O^1; \ldots; O^h]\, Y$

- $Y \in \mathbb{R}^{d \times d}$
- $[O^1; \ldots; O^h] \in$?
- $O \in \mathbb{R}$?

Dimensions?

The Matrices Form of Multi-head Attention

$$Q^l = I\, W_Q^l$$
$$K^l = I\, W_K^l$$
$$V^l = I\, W_V^l$$

$I = \{a_1, \ldots, a_n\} \in \mathbb{R}^{n \times d}$, where $a_i \in \mathbb{R}^d$

$W_Q^l, W_K^l, W_V^l \in \mathbb{R}^{d \times \frac{d}{h}}$

$Q^l, K^l, V^l \in \mathbb{R}^{n \times \frac{d}{h}}$

$$A^l = Q^l\, K^{l^T}$$
$$A^{l'} = \text{softmax}(A^l)$$

$A^{l'}, A^l \in \mathbb{R}^{n \times n}$

$$O^l = A^{l'}\, V^l$$

$O^l \in \mathbb{R}^{n \times \frac{d}{h}}$

$$O = [O^1; \ldots; O^h]\; Y$$

$Y \in \mathbb{R}^{d \times d}$

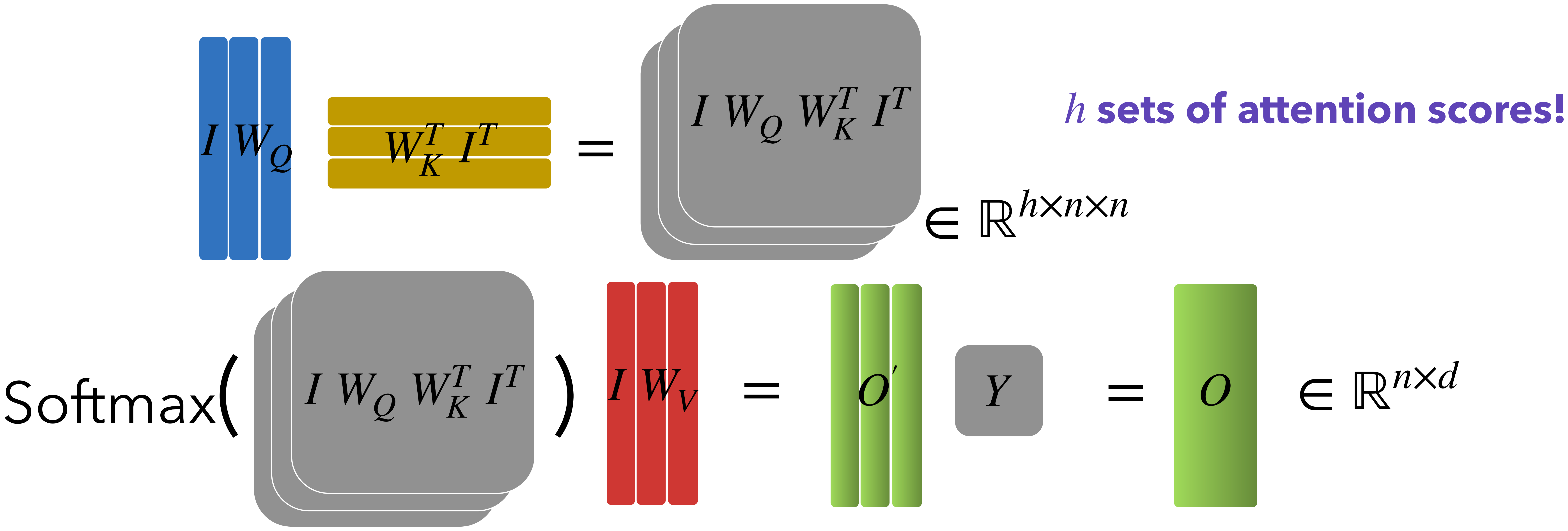
$[O^1; \ldots; O^h] \in \mathbb{R}^{n \times d}$

$O \in \mathbb{R}^{n \times d}$

Dimensions?

Multi-head Attention is Computationally Efficient

- Even though we compute h many attention heads, it's not more costly.
  - We compute $I\ W_Q \in \mathbb{R}^{n \times d}$, and then reshape to $\mathbb{R}^{n \times h \times \frac{d}{h}}$.
  - Likewise for $I\ W_K$ and $I\ W_V$.
  - Then we transpose to $\mathbb{R}^{h \times n \times \frac{d}{h}}$; **now the head axis is like a batch axis.**
  - Almost everything else is identical. All we need to do is to reshape the tensors!



Scaled Dot Product

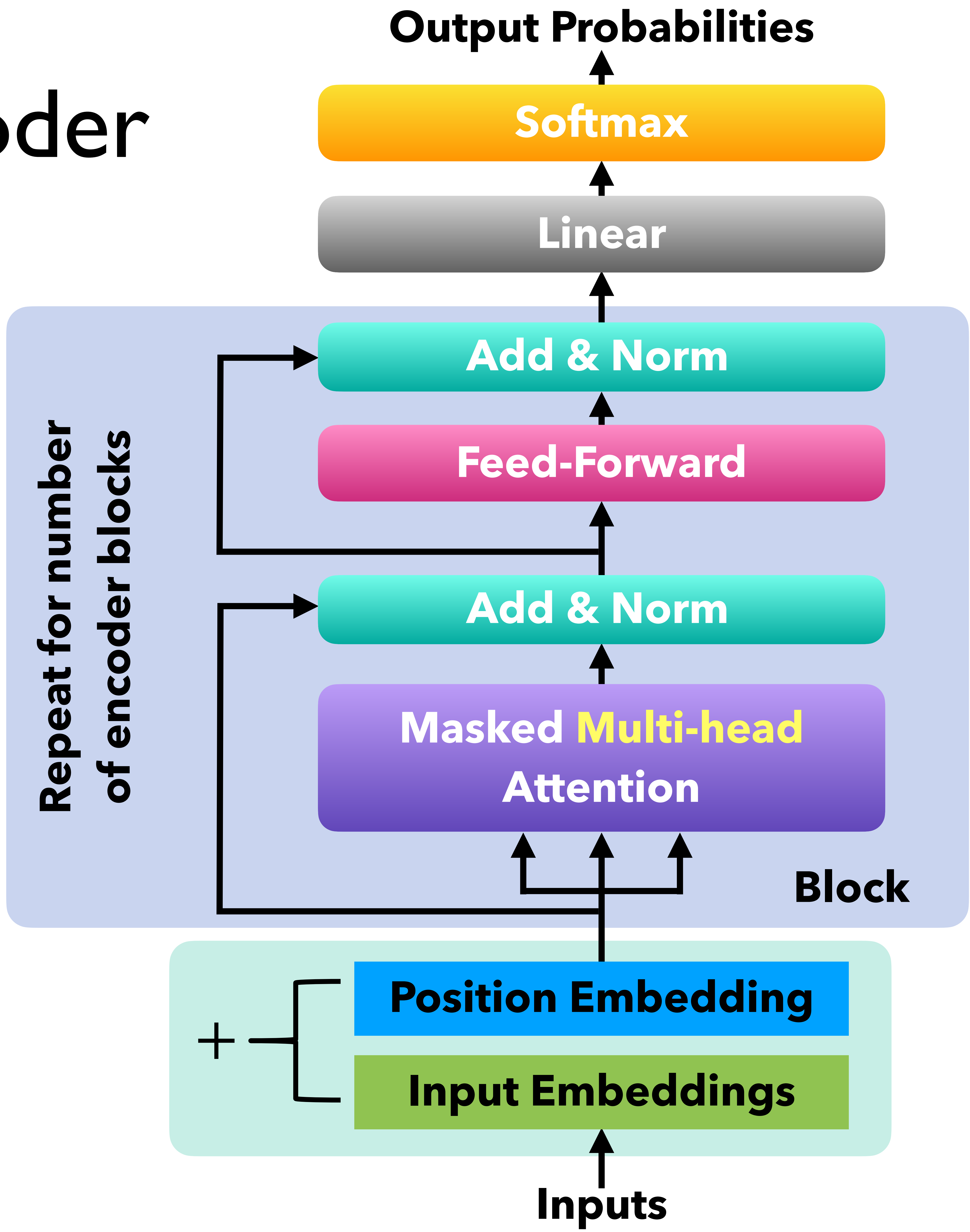
- **"Scaled Dot Product"** attention aids in training.
- When dimensionality d becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.

- Instead of the self-attention function we've seen:
  - $O^l = \text{softmax}(I\, W_Q^l\, {W_K^l}^T\, I^T)\, I\, W_V^l$
- **We divide the attention scores by** $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h (the dimensionality divided by the number of heads).

$$O^l = \textbf{softmax}\left(\frac{I\, W_Q^l\, {W_K^l}^T\, I^T}{\sqrt{d/h}}\right) I\, W_V^l$$

The Transformer Decoder

- Now that we've replaced self-attention with multi-head self-attention, we'll go through two **optimization tricks**:
  - ***Residual connection ("Add")***
  - ***Layer normalization ("Norm")***

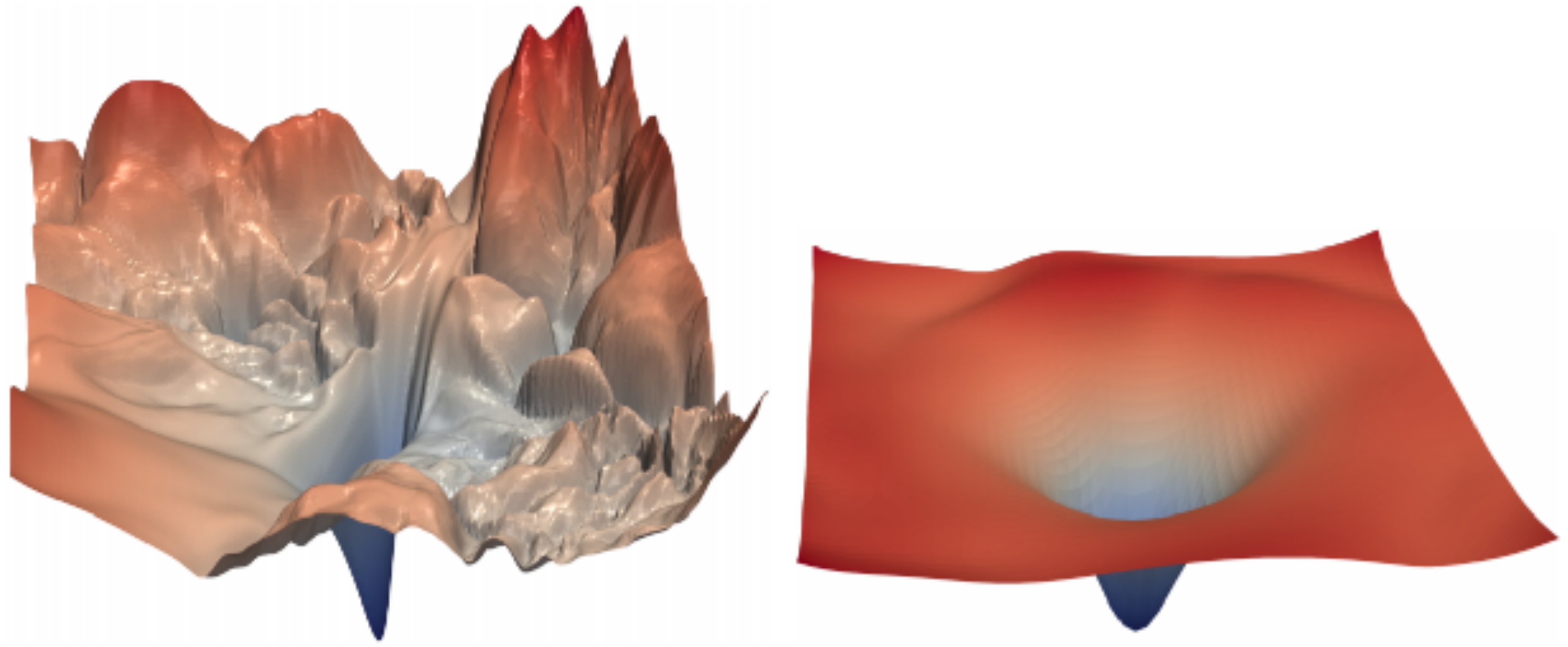


Residual Connections

- Residual connections are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)

  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)

  - Gradient is great through the residual connection; it's 1!
  - Bias towards the identity function!



[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

Layer Normalization

- Layer normalization is a trick to help models train faster.
- **Idea:** cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within each layer.
  - LayerNorm's success may be due to its normalizing gradients [Xu et al., 2019]
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.
- Let $\mu = \sum_{j=1}^{d} x_j$; this is the mean; $\mu \in \mathbb{R}$.
- Let $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^{d} \left(x_j - \mu\right)^2}$; this is the standard deviation; $\sigma \in \mathbb{R}$.
- Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)
- Then layer normalization computes:

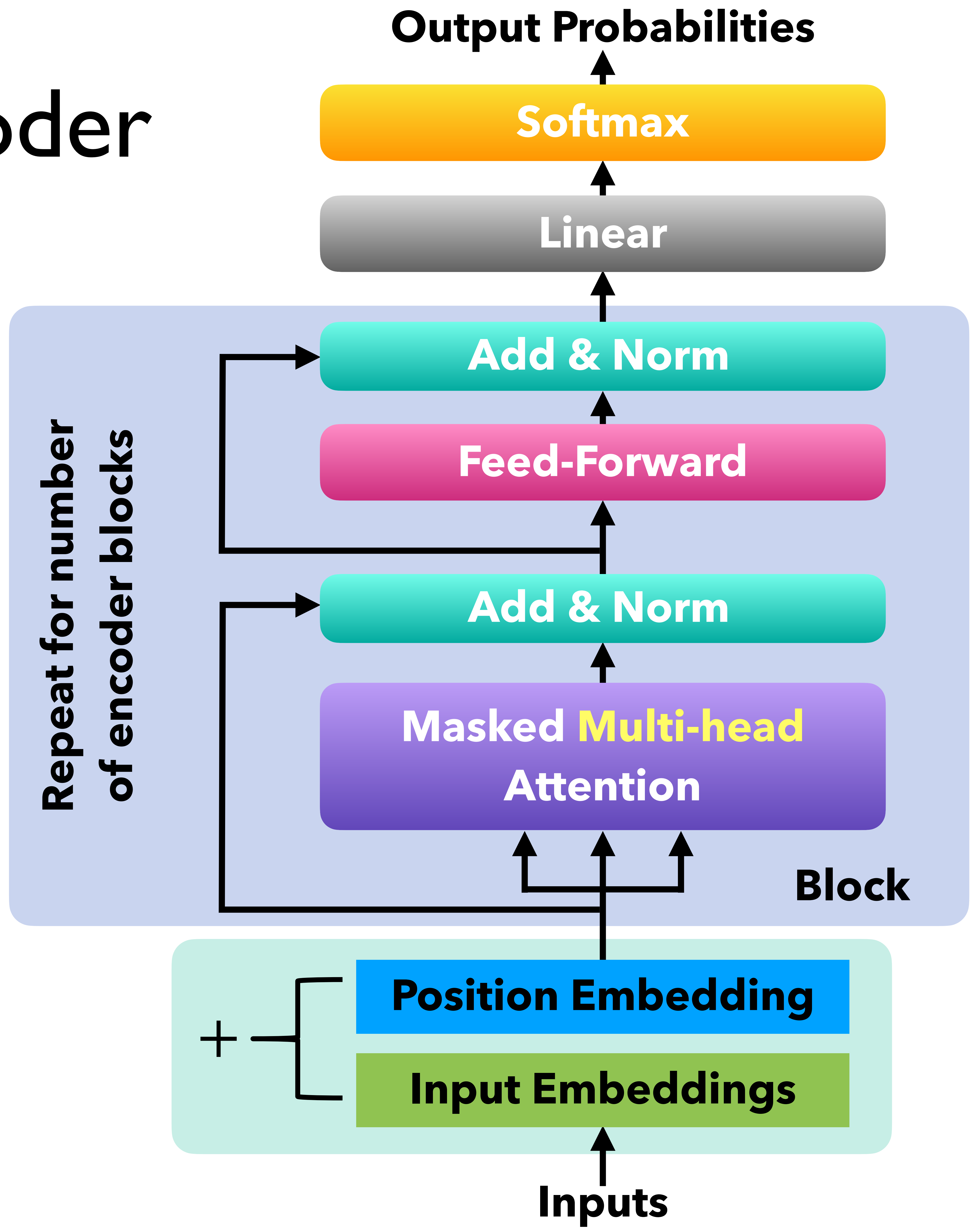
Normalize by scalar mean and variance

- $\text{output} = \frac{x - \mu}{\sqrt{\sigma} + \epsilon} * \gamma + \beta$

Modulate by learned element-wise gain and bias

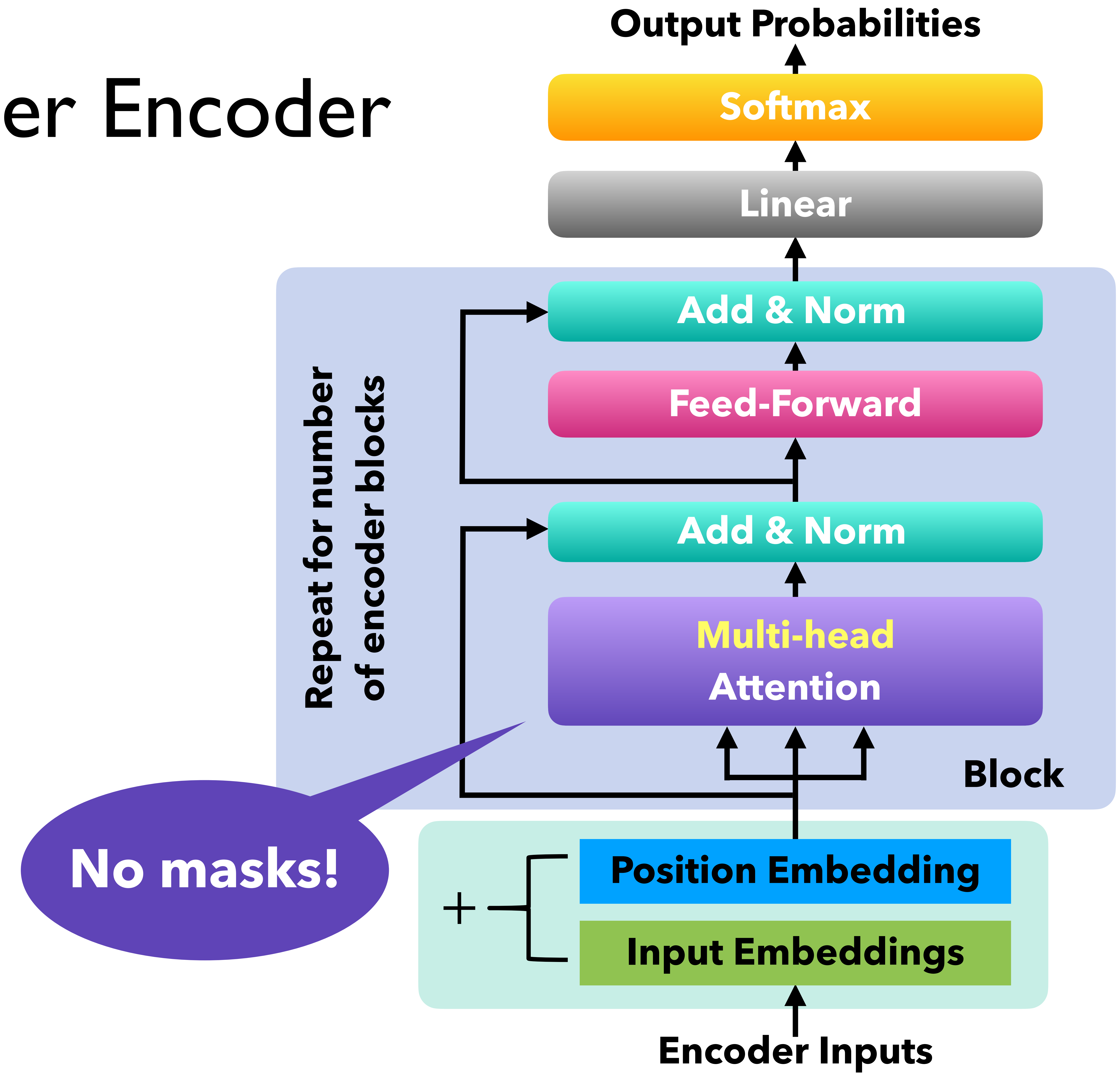
The Transformer Decoder

- The Transformer Decoder is a stack of Transformer Decoder **Blocks**.
- Each Block consists of:
  - Masked Multi-head Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm



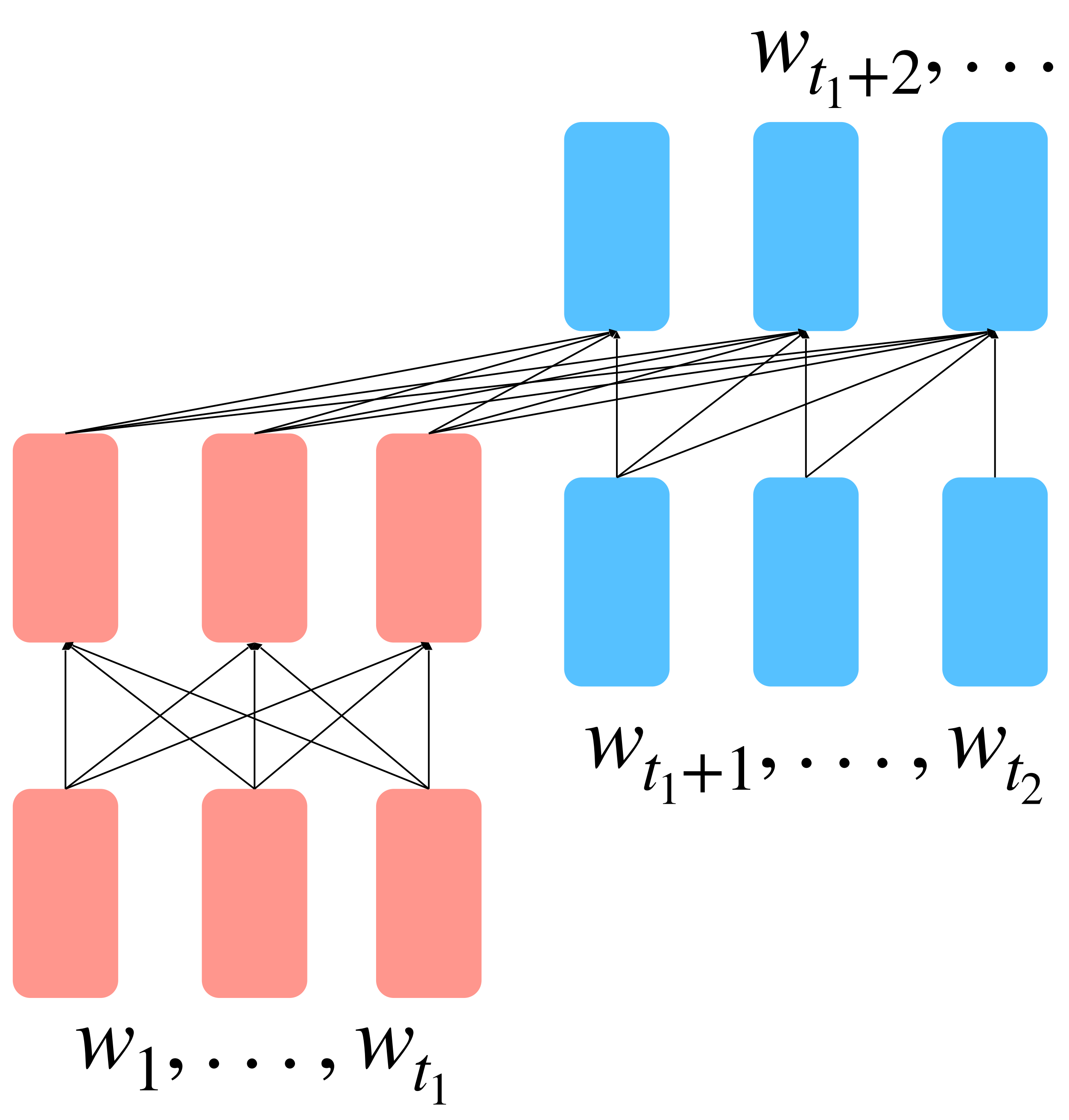
The Transformer Encoder

- The Transformer **Decoder** constrains to **unidirectional** context, as for language models.
- What if we want **bidirectional** context, like in a bidirectional RNN?
- We use **Transformer Encoder** – the ONLY difference is that we **remove the masking** in self-attention.

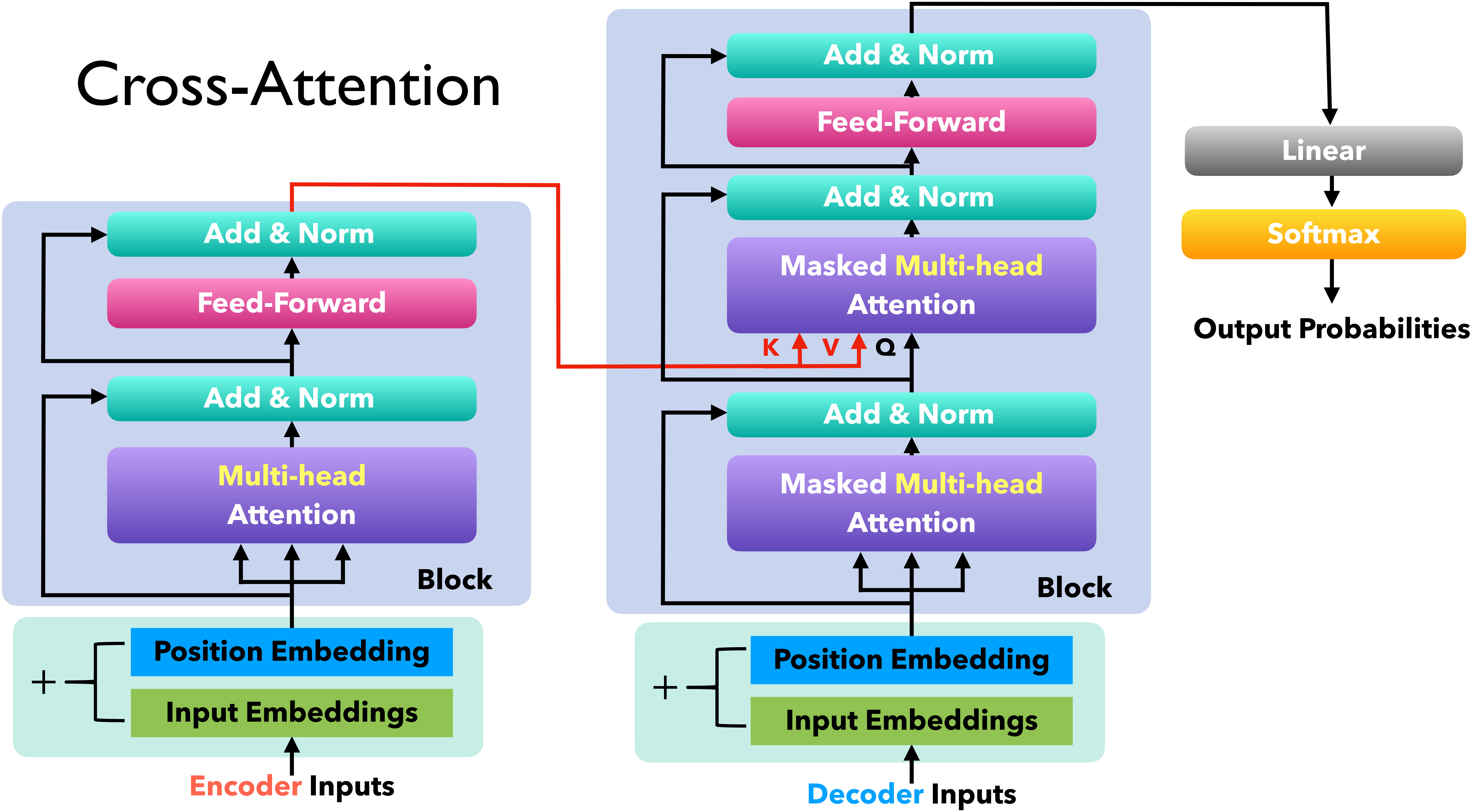


The Transformer Encoder-Decoder

- More on Encoder-Decoder models will be introduced in the next lecture!
- Right now we only need to know that it processes the source sentence with a **bidirectional** model (**Encoder**) and generates the target with a **unidirectional** model (**Decoder**).
- The Transformer Decoder is modified to perform **cross-attention** to the output of the Encoder.



Cross-Attention



Cross-Attention Details

- **Self-attention:** queries, keys, and values come from the same source.
- **Cross-Attention:** *keys* and *values* are from **Encoder** (like a memory); *queries* are from **Decoder**.
- Let $h_1, \ldots, h_n$ be output vectors from the Transformer **encoder**, $h_i \in \mathbb{R}^d$.
- Let $z_1, \ldots, z_n$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$.
- **Keys** and **values** from the **encoder**:
  - $k_i = W_K h_i$
  - $v_i = W_V h_i$
- **Queries** are drawn from the **decoder**:
  - $q_i = W_Q z_i$

Transformers: pros and cons

- **Easier to capture long-range dependencies**: we draw attention between every pair of words!
- **Easier to parallelize:**

$$Q = XW^Q \qquad K = XW^K \qquad V = XW^V$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

- **Are positional encodings enough to capture positional information?**

Otherwise self-attention is an unordered function of its input

- **Quadratic computation in self-attention**

Can become very slow when the sequence length is large

Quadratic computation as a function of sequence length

$Q = XW^Q \qquad K = XW^K \qquad V = XW^V$

$n \times d_q$ $\qquad d_k \times n$ $\qquad n \times d_v$

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

Need to compute n^2 pairs of scores (= dot product) O(n^2d)

RNNs only require $O(nd^2)$ running time:

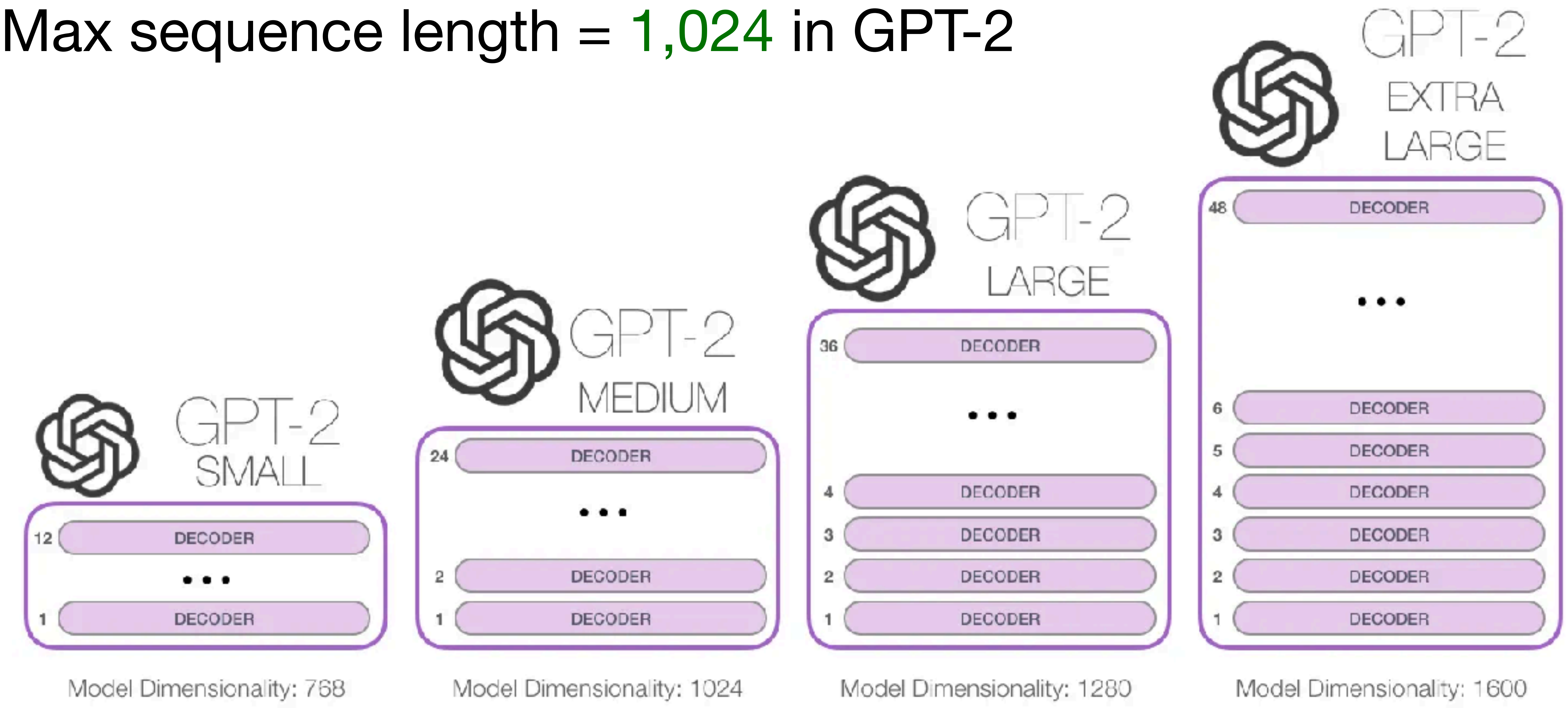
$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

(assuming input dimension = hidden dimension = d)

Quadratic computation as a function of sequence length

Need to compute n^2 pairs of scores (= dot product) O(n^2d)

Max sequence length = 1,024 in GPT-2

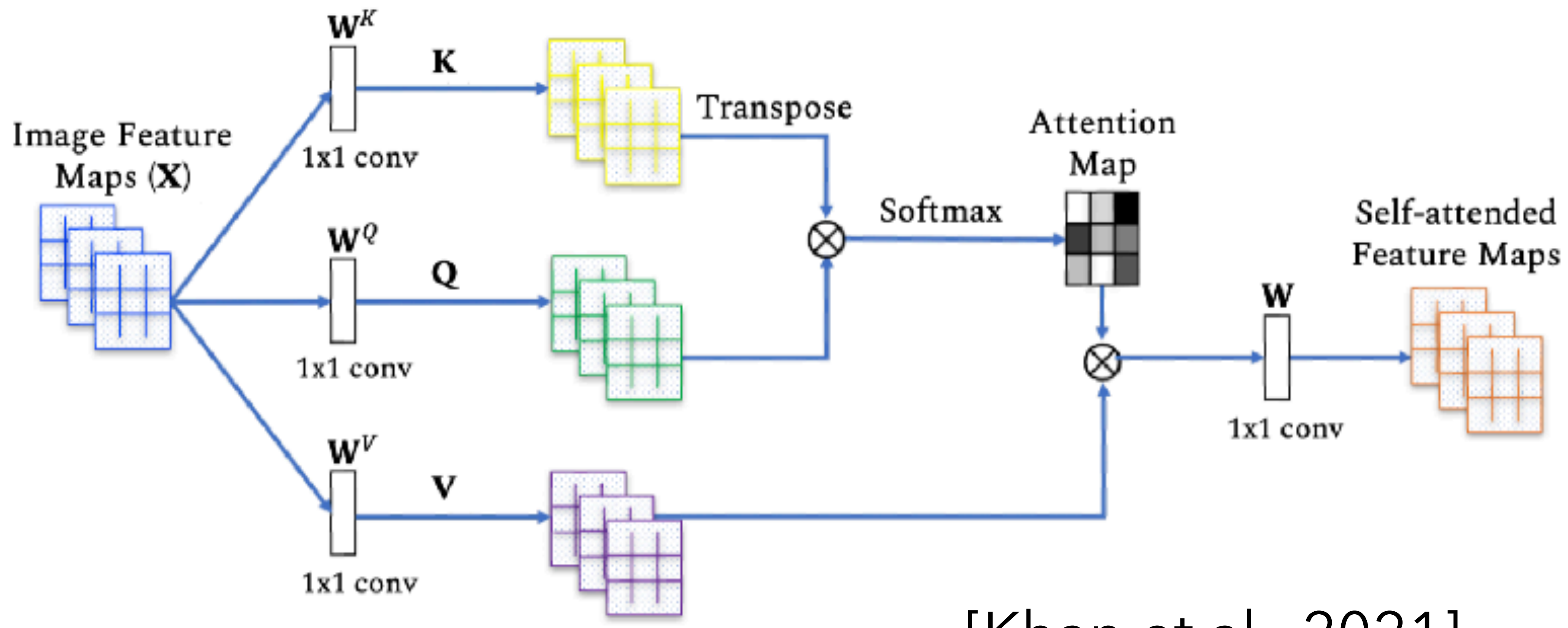


What if we want to scale $n \geq 50{,}000$? For example, to work on long documents?

The Revolutionary Impact of Transformers

- **Almost all current-day leading language models** use Transformer building blocks.
  - E.g., GPT1/2/3/4, T5, Llama 1/2, BERT, … almost anything we can name
  - Transformer-based models dominate nearly all NLP leaderboards.

- Since Transformer has been popularized in language applications, computer vision also adapted Transformers, e.g., **Vision Transformers**.



[Khan et al., 2021]

What's next after Transformers?