Deep Learning for NLP

Module 2: Transformers and Pretraining

Teaching: Khyathi Chandu

Acknowledgements

- Slide deck adapted from Taylor Sorenson, Liwei Jiang, Yejin Choi.
- Many thanks to John Hewitt for sharing his amazing slide deck on Transformers for us to adapt from!
- Also thanks to the online course from Hung-yi Lee, for the walk-through of self-attention and multi-head attention!

What are we learning?

- How to model language computationally?
  - Attention: Self-attention, cross-attention, multi-headed attention
  - Working of transformers: diving into all components
  - Encoder/decoder models
  - Coding exercise (fine-tuning)
- What is pre-training and how is it performed?
  - Why pretraining?
  - Finetuning and catastrophic forgetting
  - Pre-training paradigms and architectures
  - Training objectives

Modeling language

Attention Is All You Need (NeurIPS 2017)

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

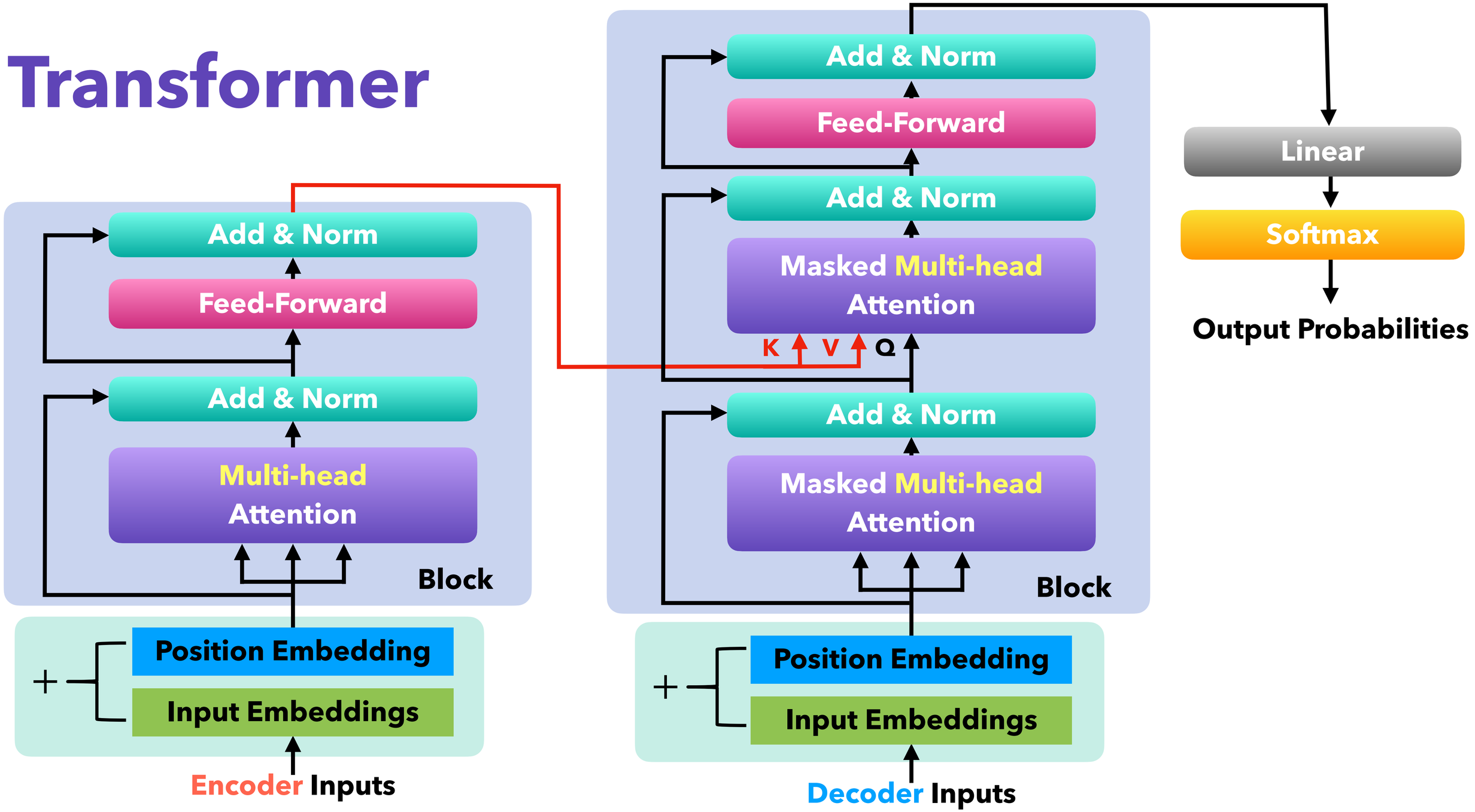
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

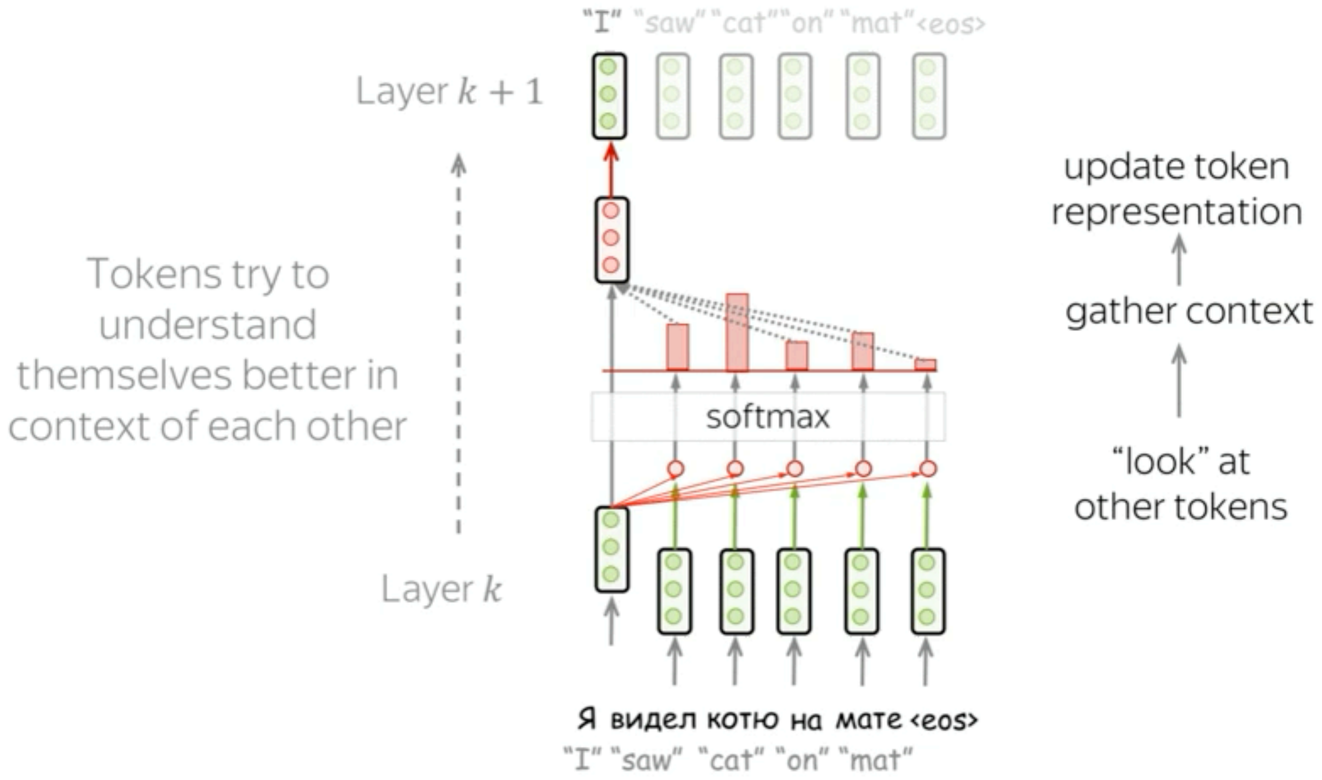
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformer



The New De Facto Method: **Attention**

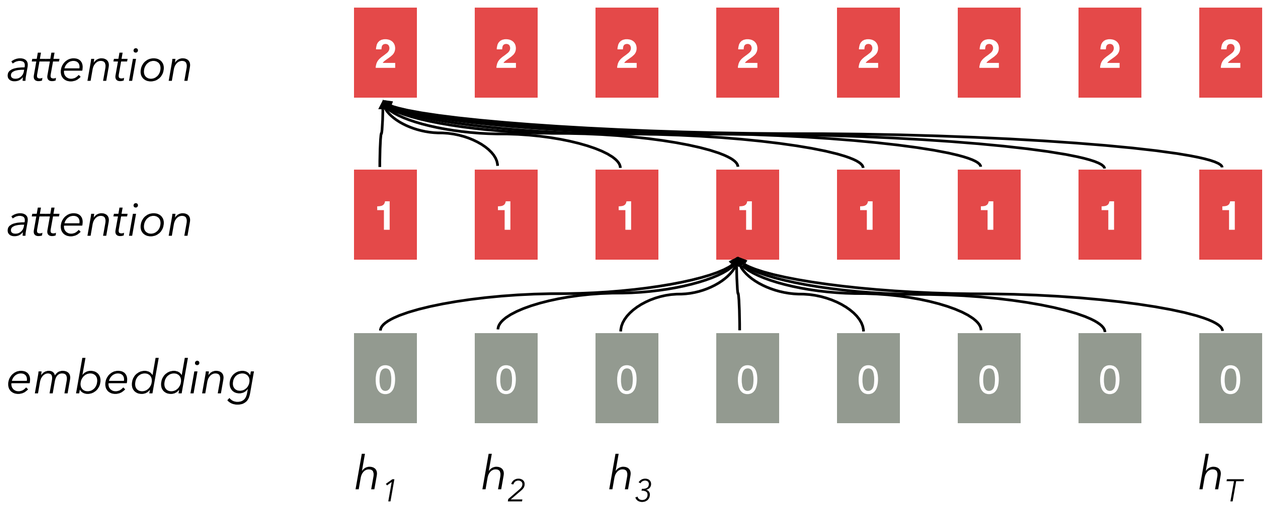
Instead of deciding the next token solely based on the previously seen tokens, **each token will "look at" all input tokens at the same to decide which ones are most important** to decide the next token.



In practice, the actions of all tokens are done in parallel!

Building the Intuition of **Attention**

- **Attention** treats each token's representation as a **query** to access and incorporate information from **a set of values**.
  - Today we look at attention **within a single sequence**.
- Number of unparallelizable operations does **NOT** increase with sequence length.

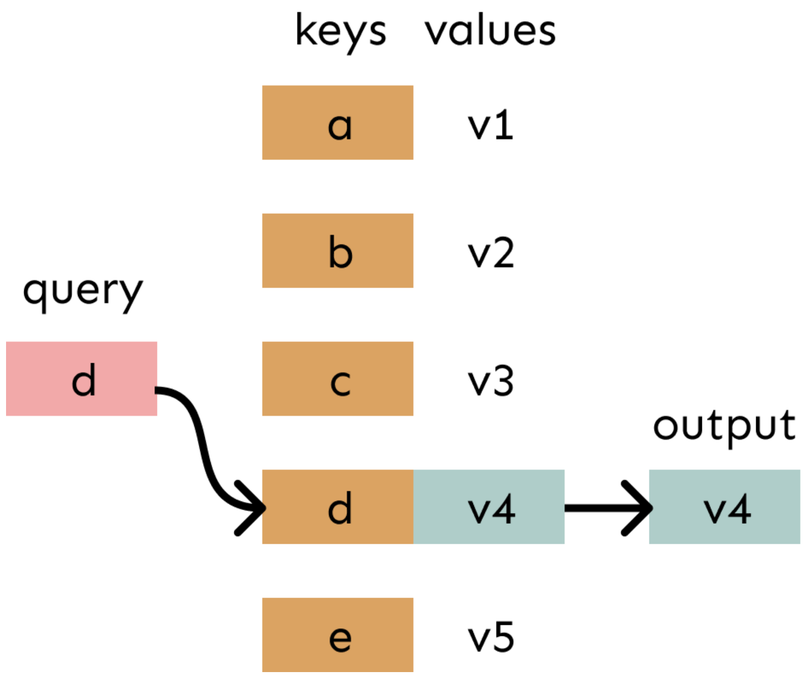


All tokens attend to all tokens in previous layer; most arrows here are omitted

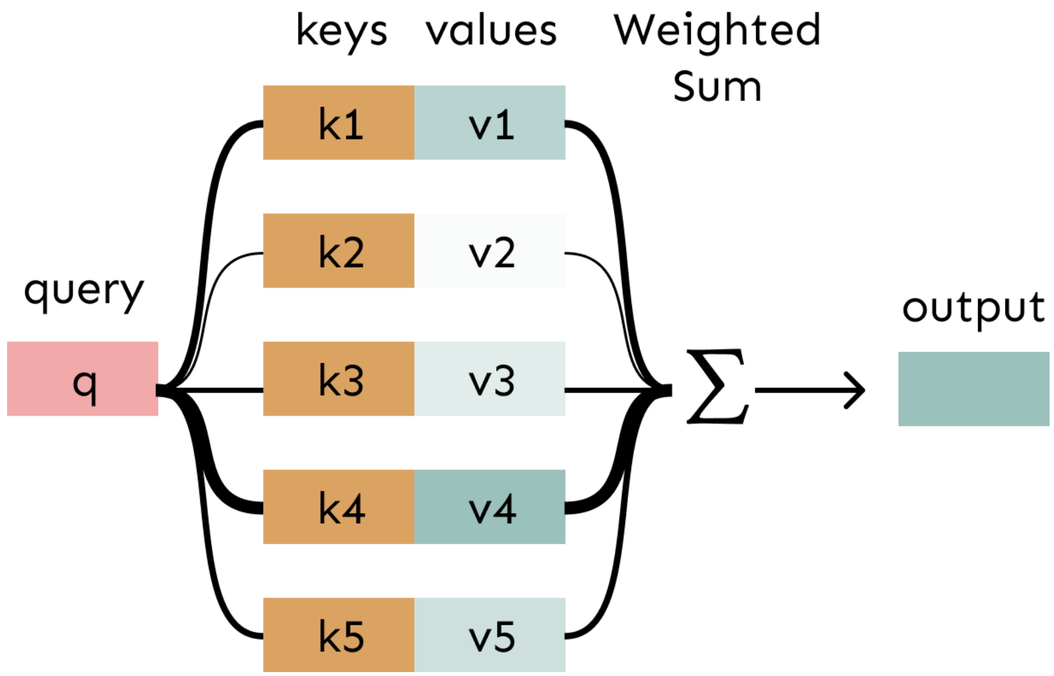
Attention as a soft, averaging lookup table

We can think of **attention** as performing **fuzzy lookup** in a key-value store.

In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



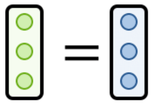
In **attention**, the **query** matches all **keys** ***softly***, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



Self-Attention: Basic Concepts

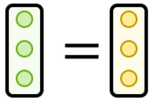
[Lena Viota Blog]

Each vector receives t

$[W_Q] \times$  $=$

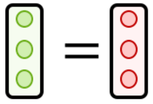
Query: vector **from** which the atten

"Hey there, do you have

$[W_K] \times$  $=$

Key: vector **at** which the query looks to con

"Hi, I have this informa

$[W_V] \times$  $=$

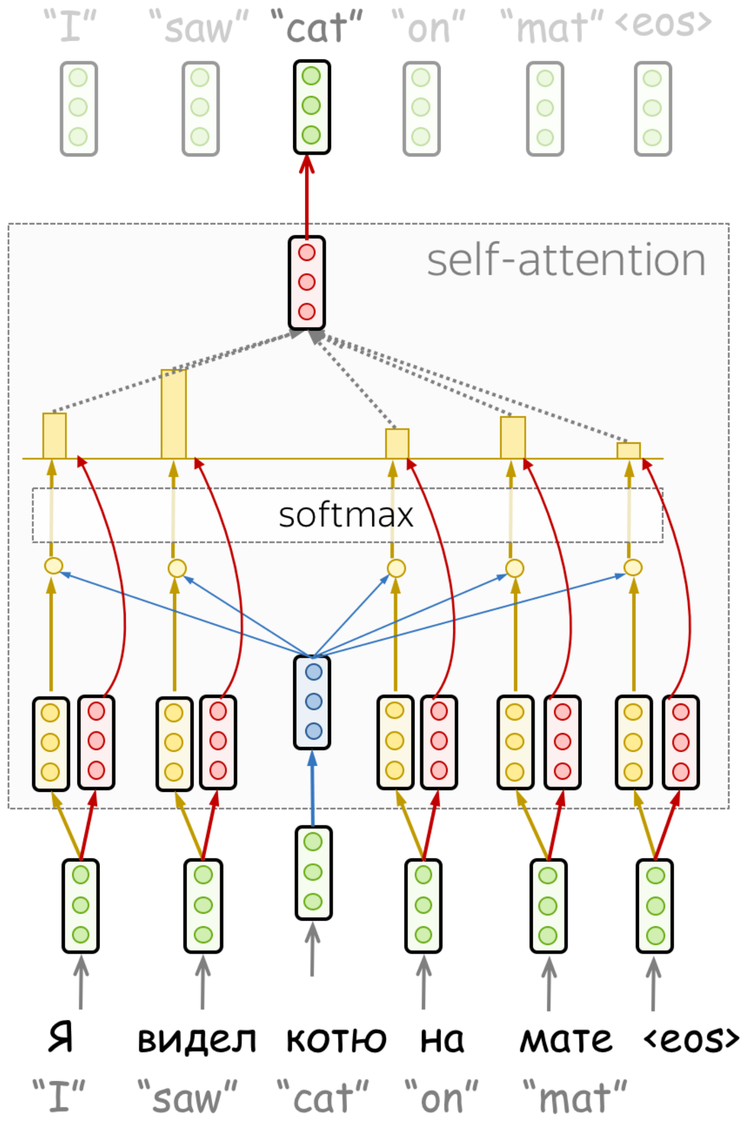
Value: t attention output

"Here's the information I have!"

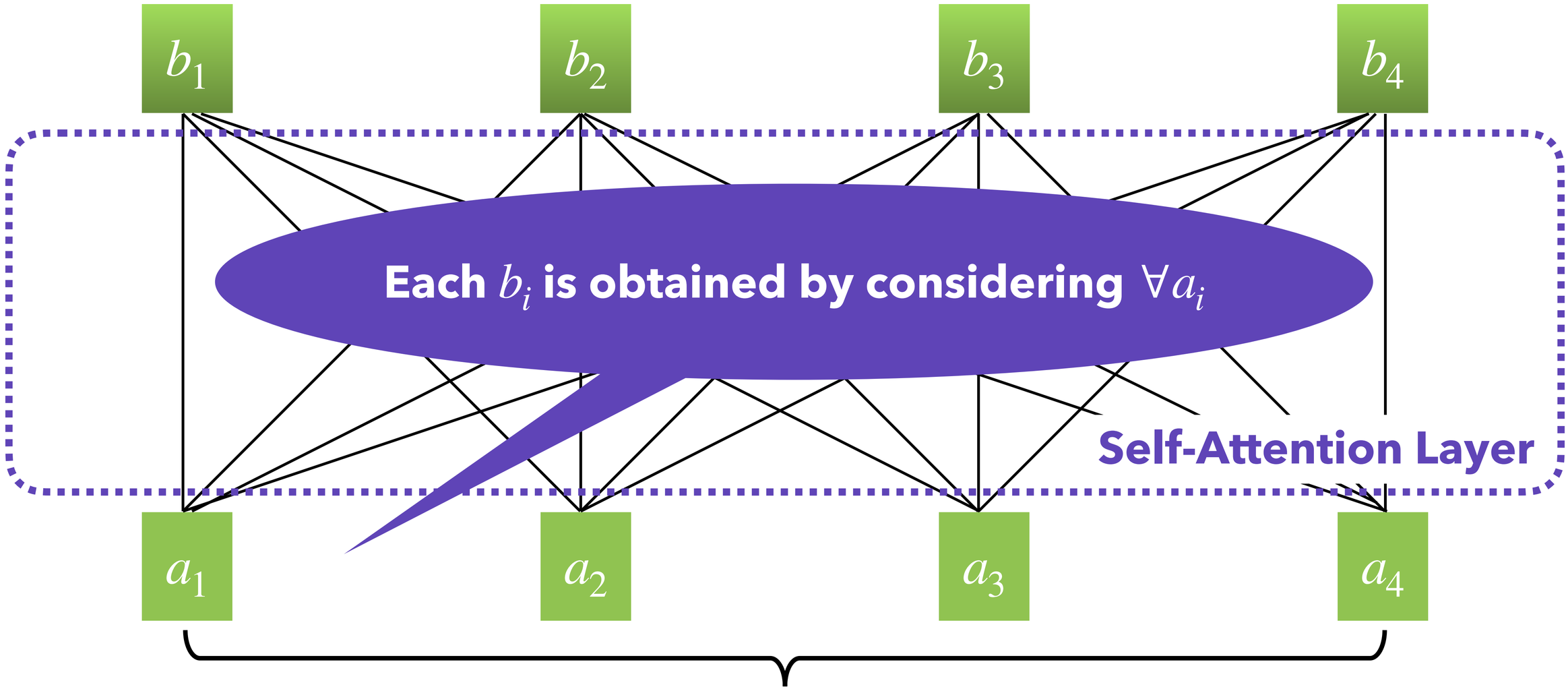
Query: asking for information

Key: saying that it has some information

Value: giving the information

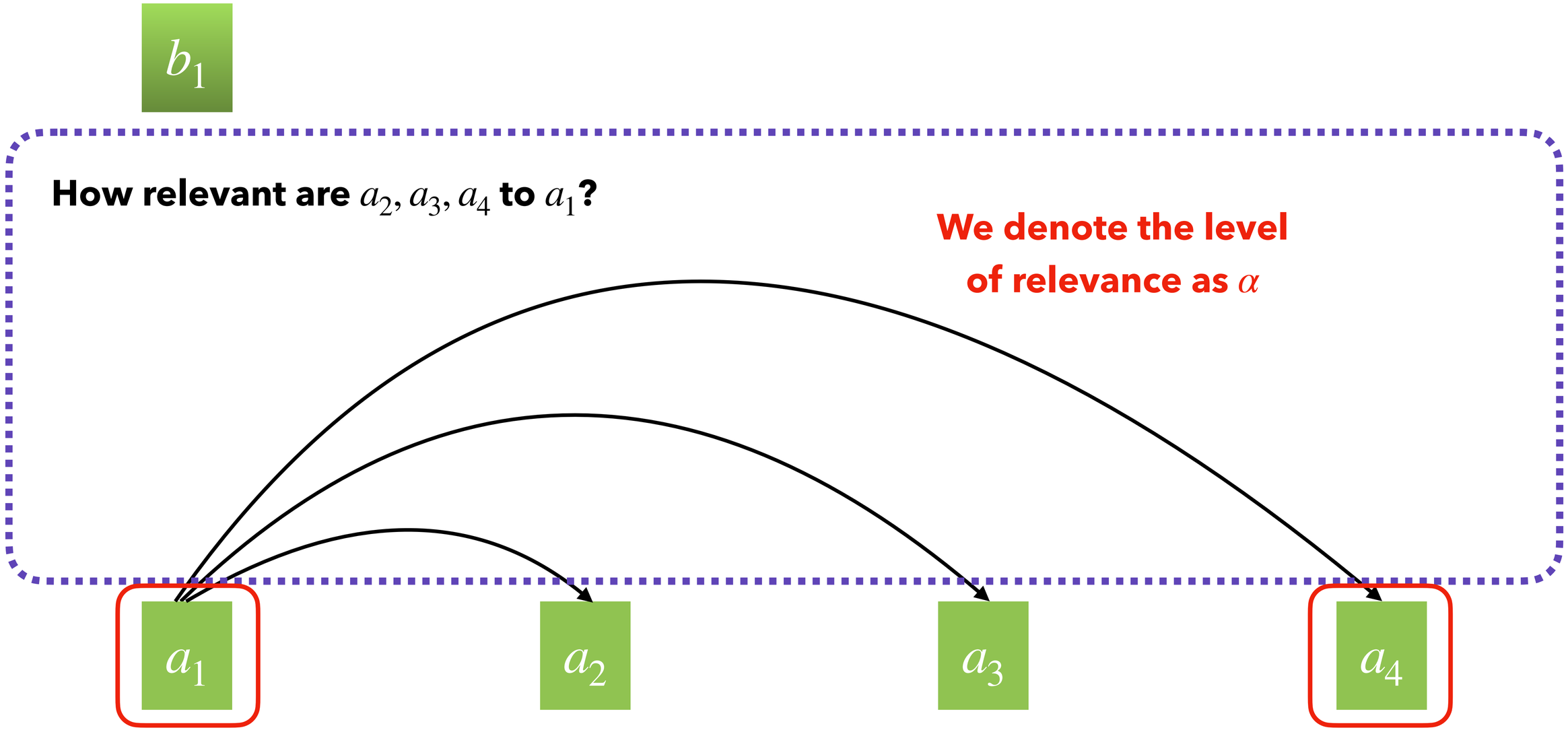


Self-Attention: Walk-through

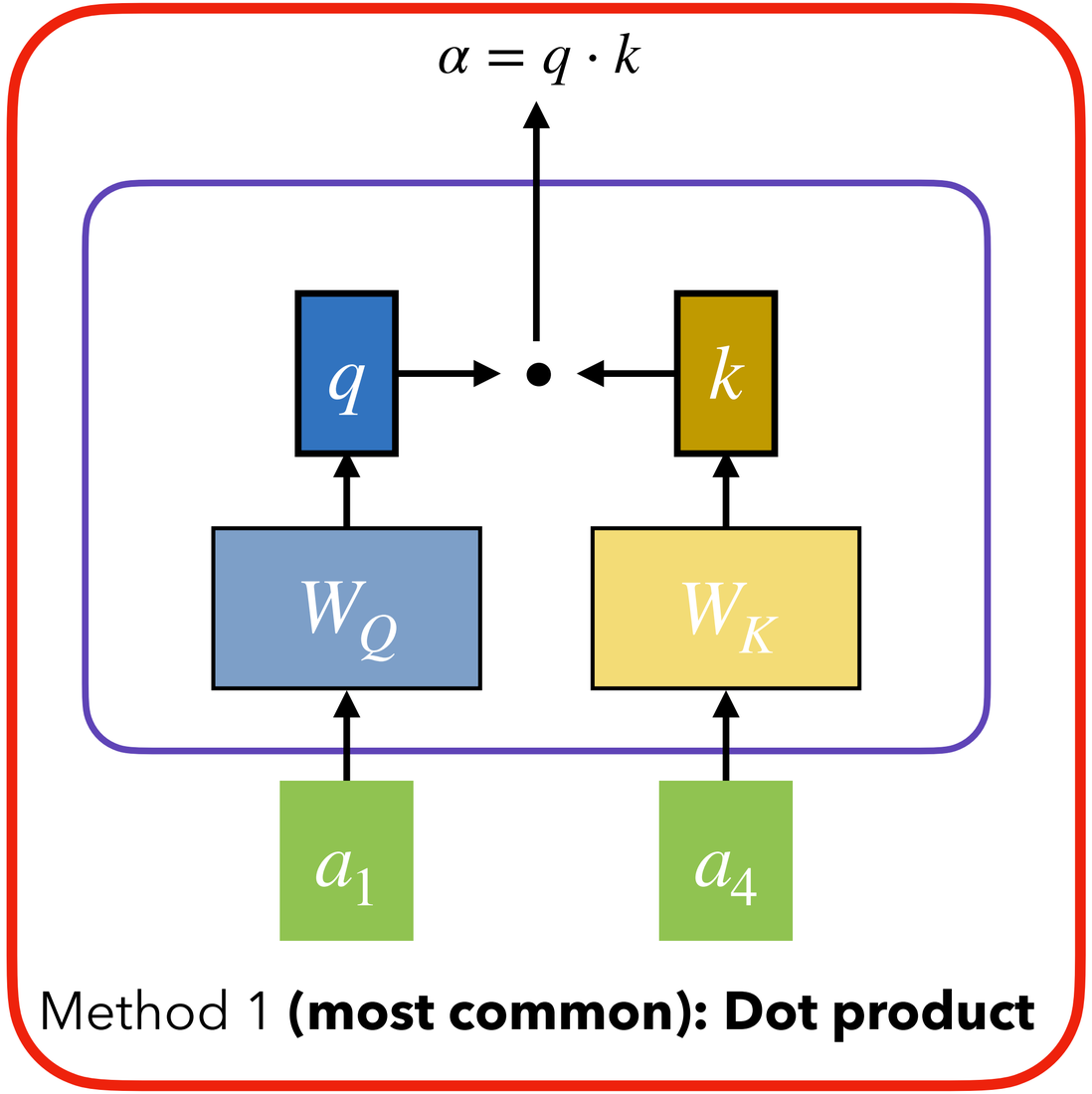


Can be either **input** or a **hidden layer**

Self-Attention: Walk-through

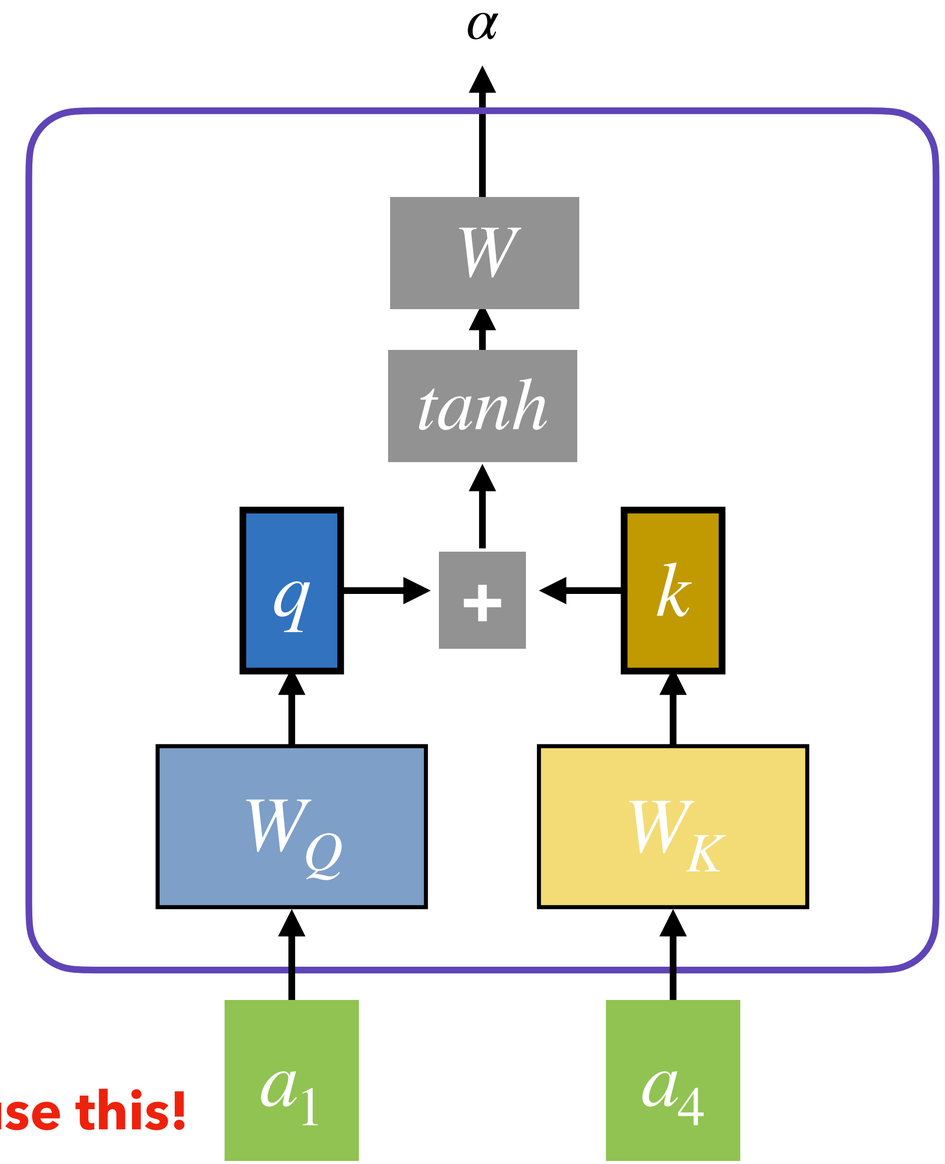


How to compute α?



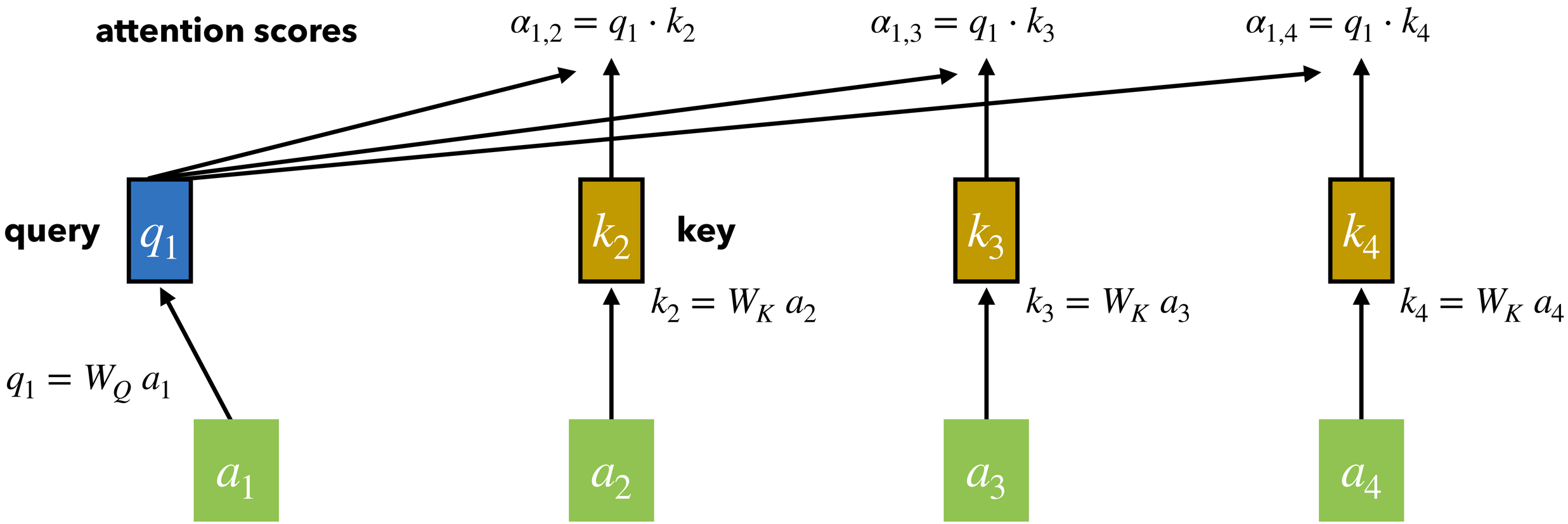
Method 1 **(most common): Dot product**

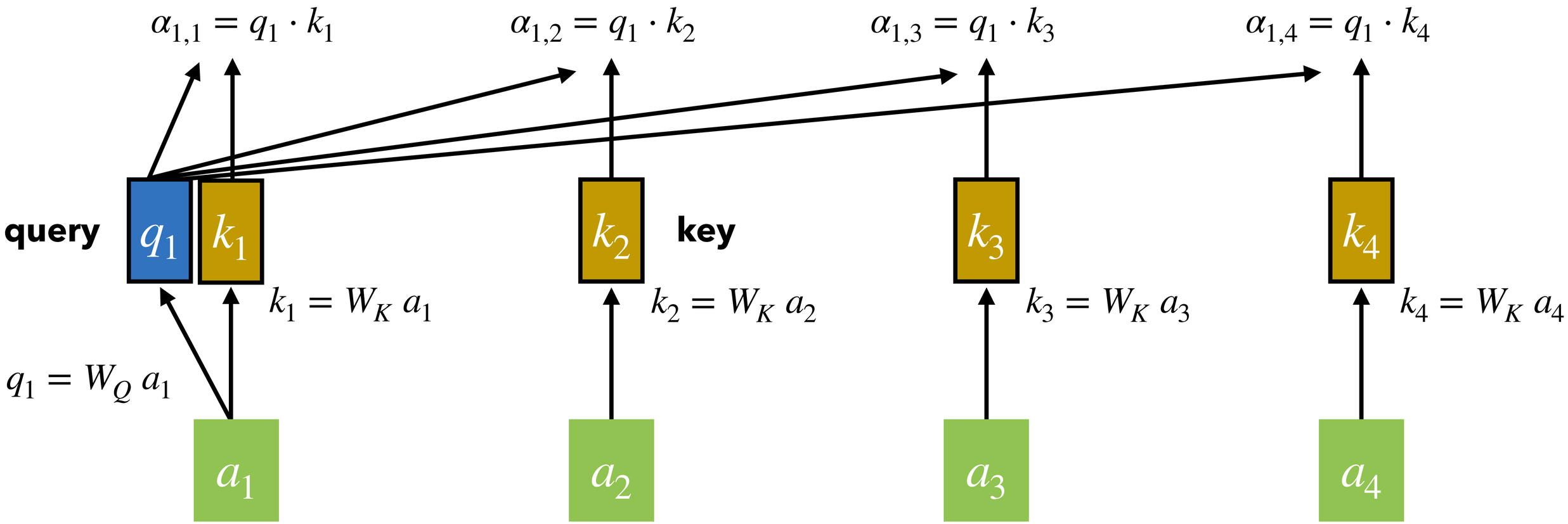
We'll use this!



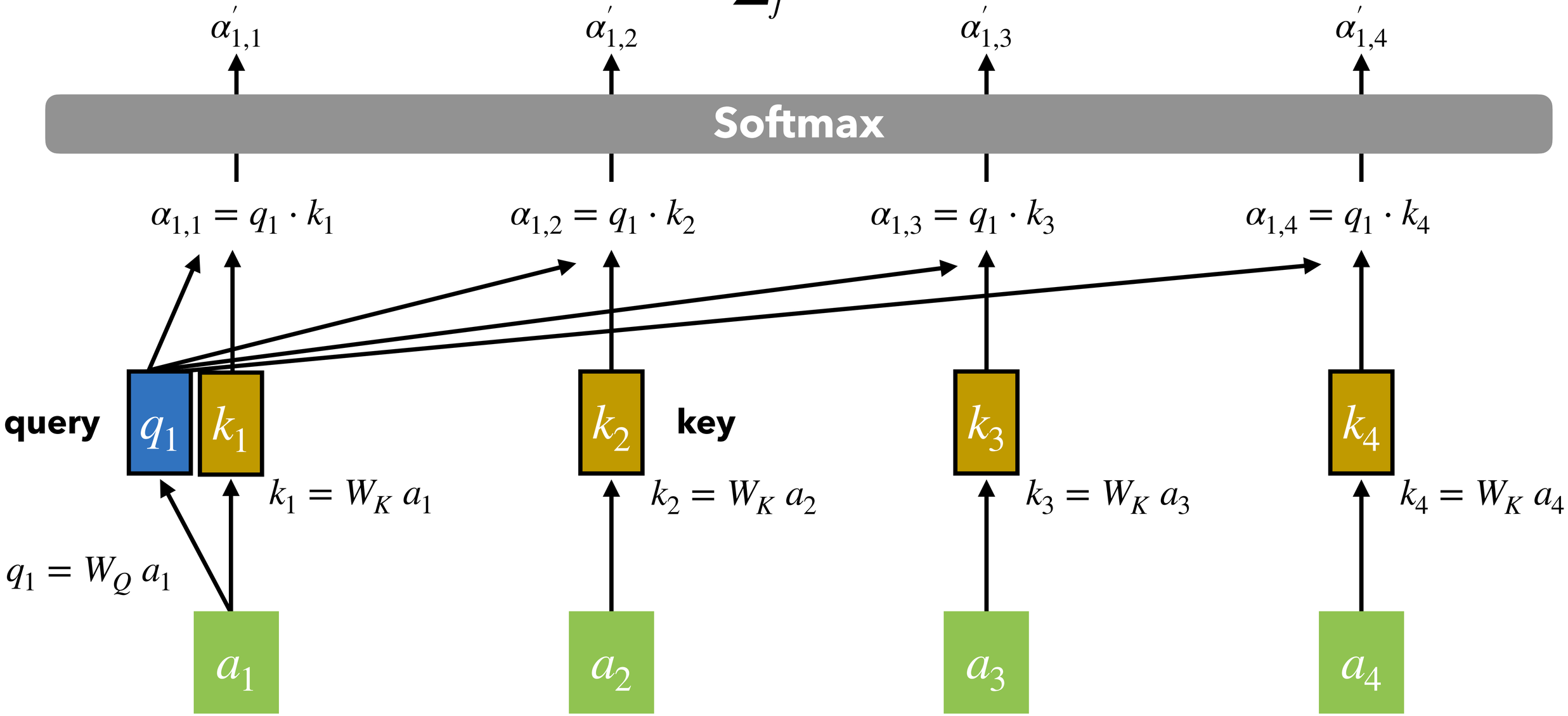
Method 2: **Additive**

Self-Attention: Walk-through



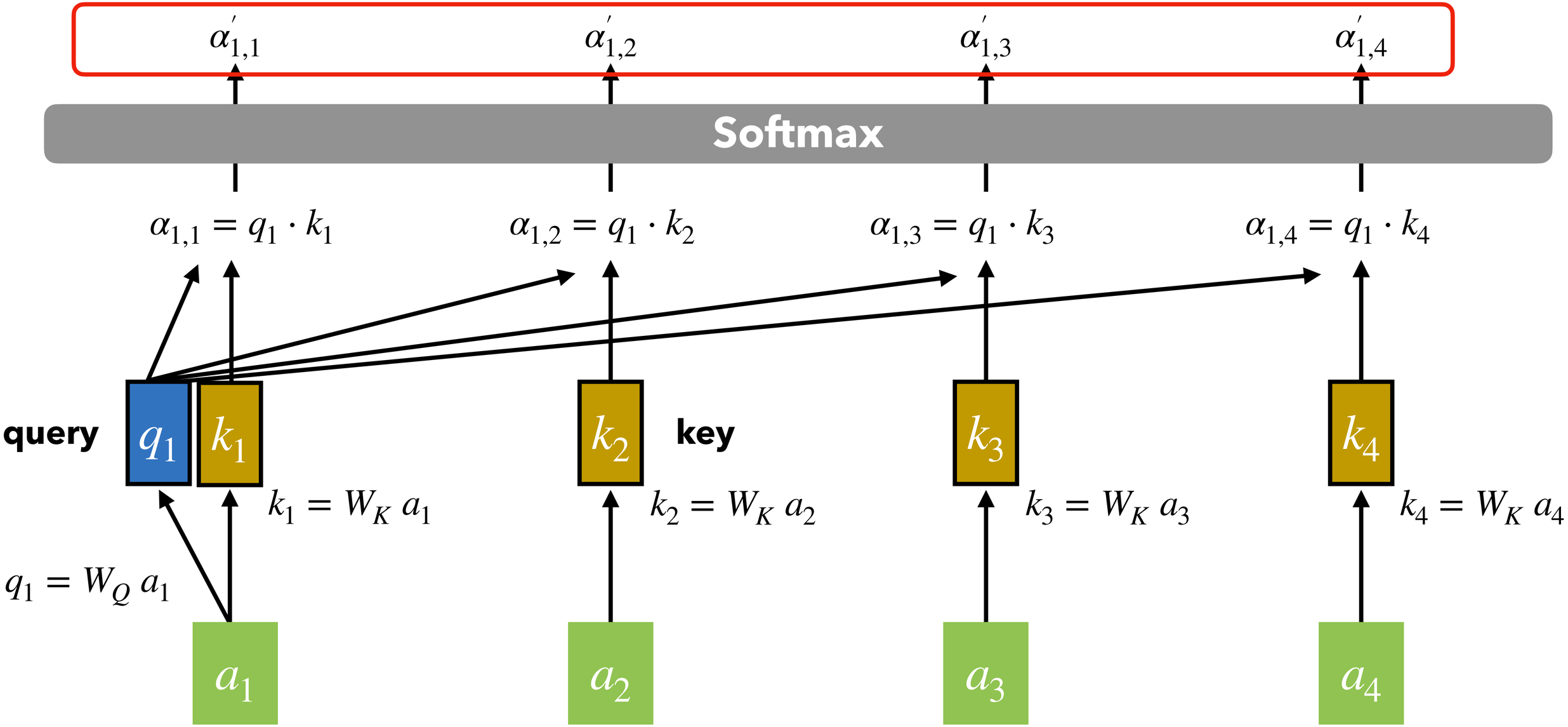

$\alpha_{1,1} = q_1 \cdot k_1$
$\alpha_{1,2} = q_1 \cdot k_2$
$\alpha_{1,3} = q_1 \cdot k_3$
$\alpha_{1,4} = q_1 \cdot k_4$
query
q_1
k_1
k_2
key
k_3
k_4
$k_1 = W_K\, a_1$
$k_2 = W_K\, a_2$
$k_3 = W_K\, a_3$
$k_4 = W_K\, a_4$
$q_1 = W_Q\, a_1$
a_1
a_2
a_3
a_4

$$\alpha'_{1,i} = \frac{e^{\alpha_{1,i}}}{\sum_j e^{\alpha_{1,j}}}$$



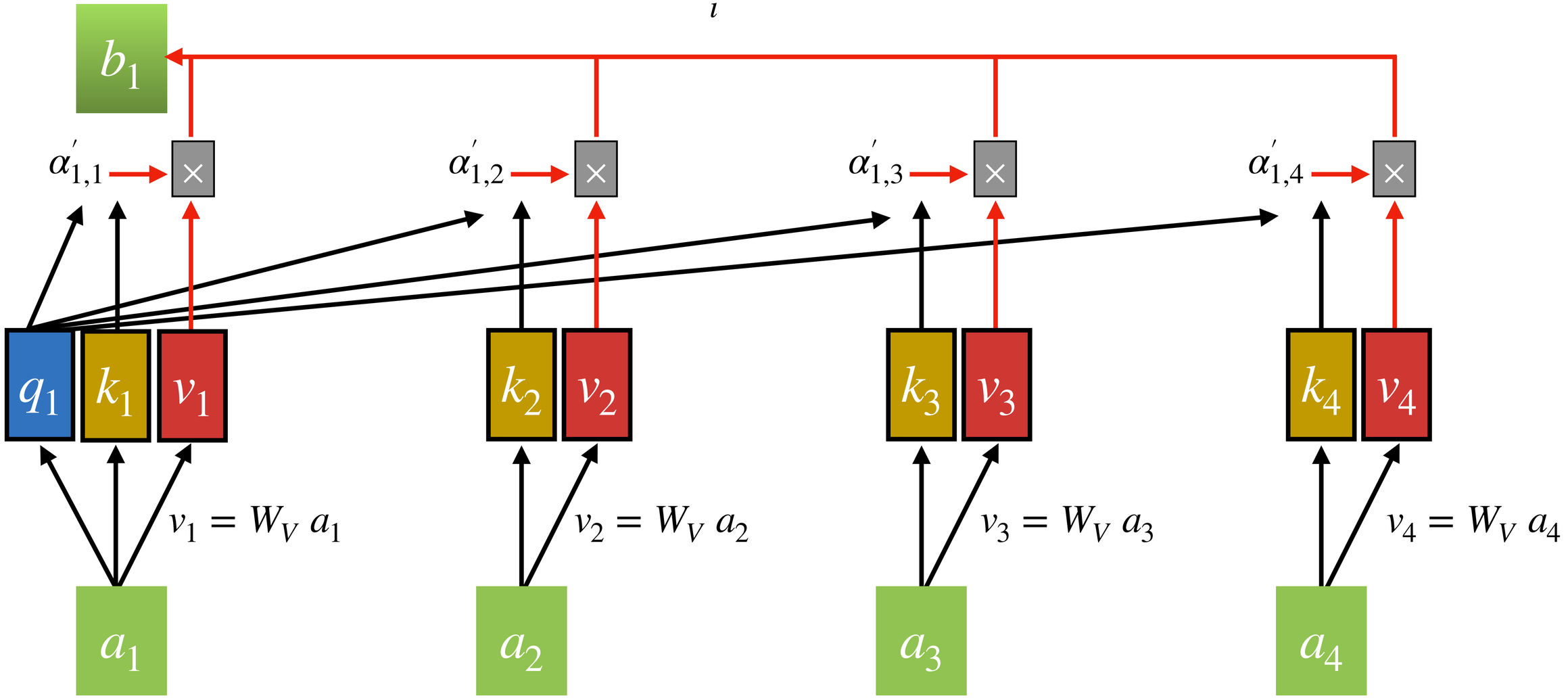
Denote how relevant each token are to a_1!

Use attention scores to extract information



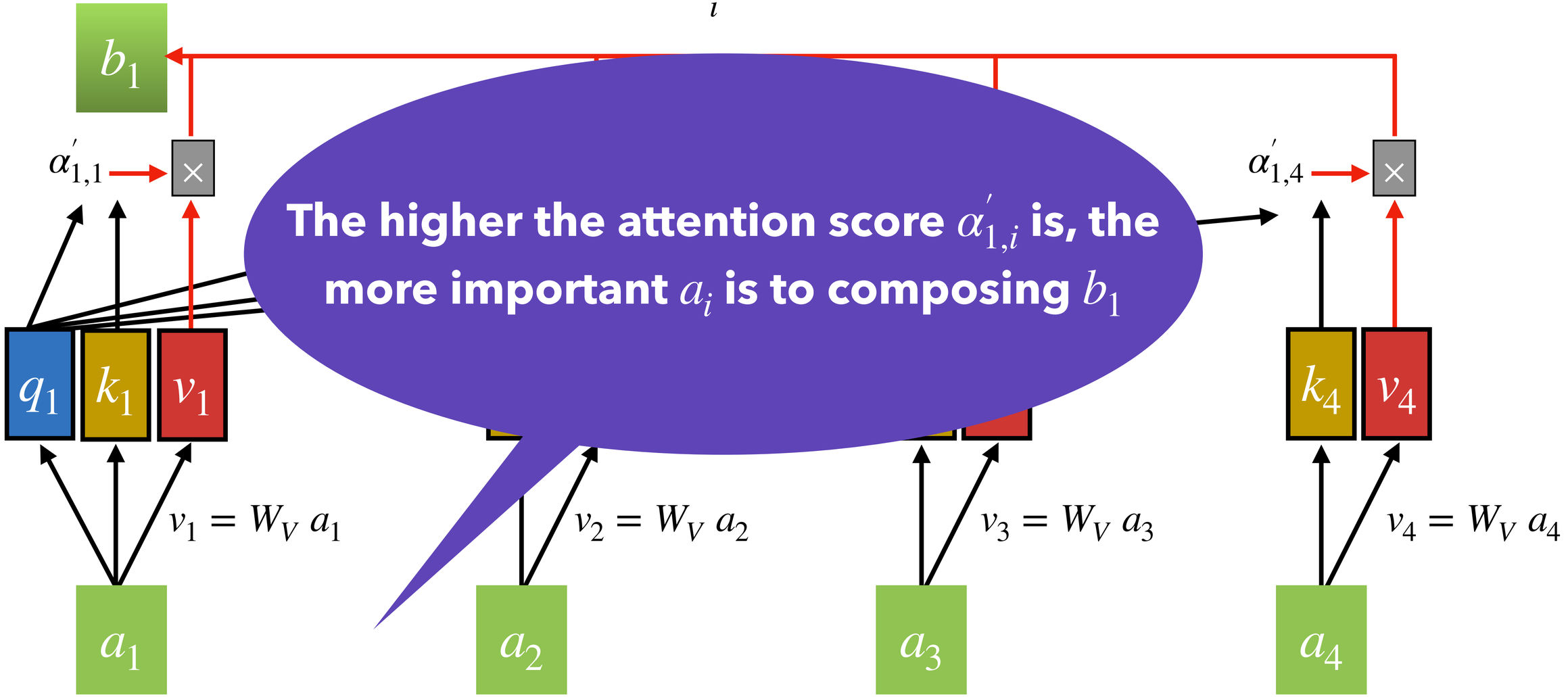
Use attention scores to extract information

$$b_1 = \sum_i \alpha'_{1,i}\, v_i$$



Use attention scores to extract information

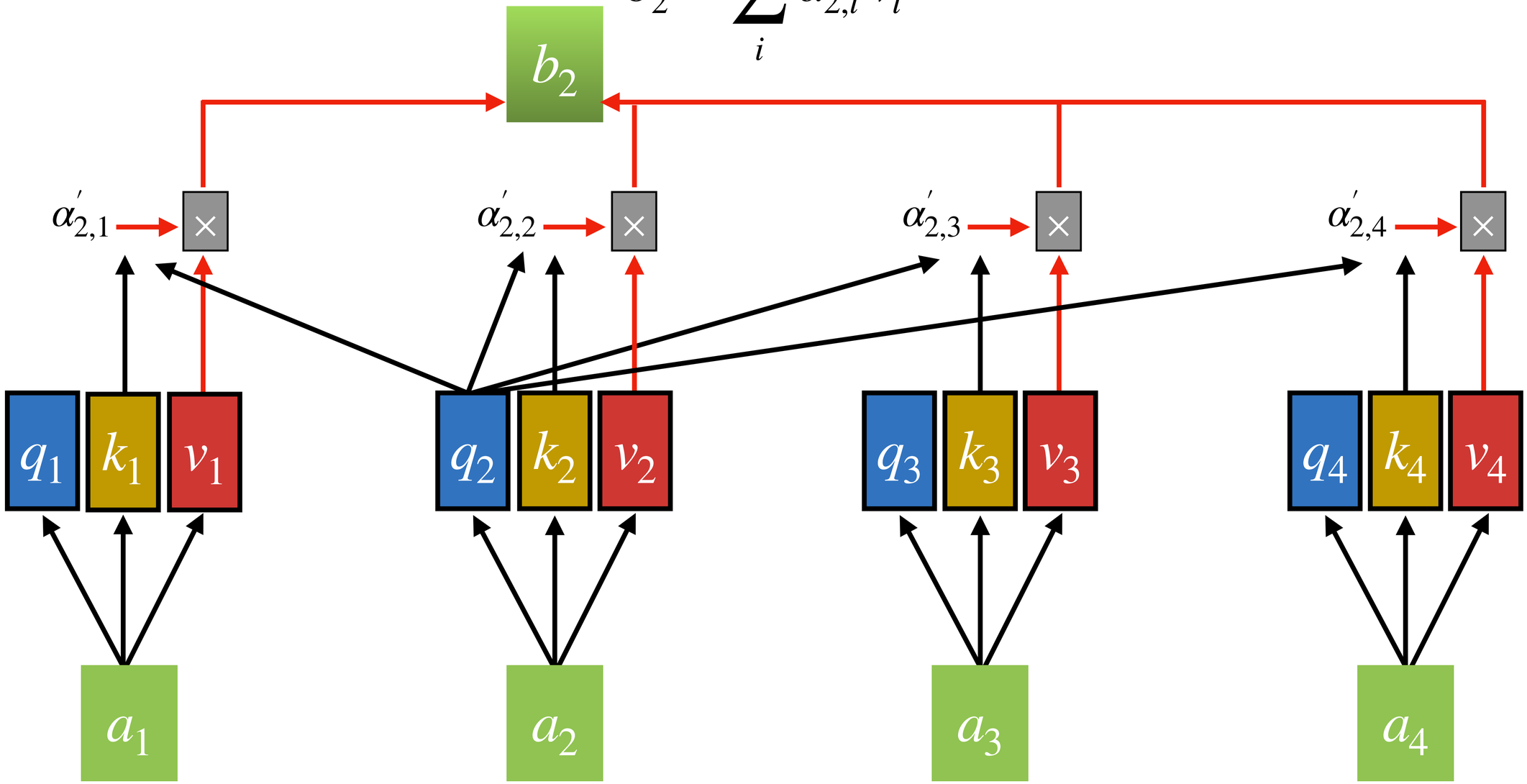
$$b_1 = \sum_i \alpha'_{1,i}\, v_i$$



The higher the attention score $\alpha'_{1,i}$ is, the more important a_i is to composing b_1

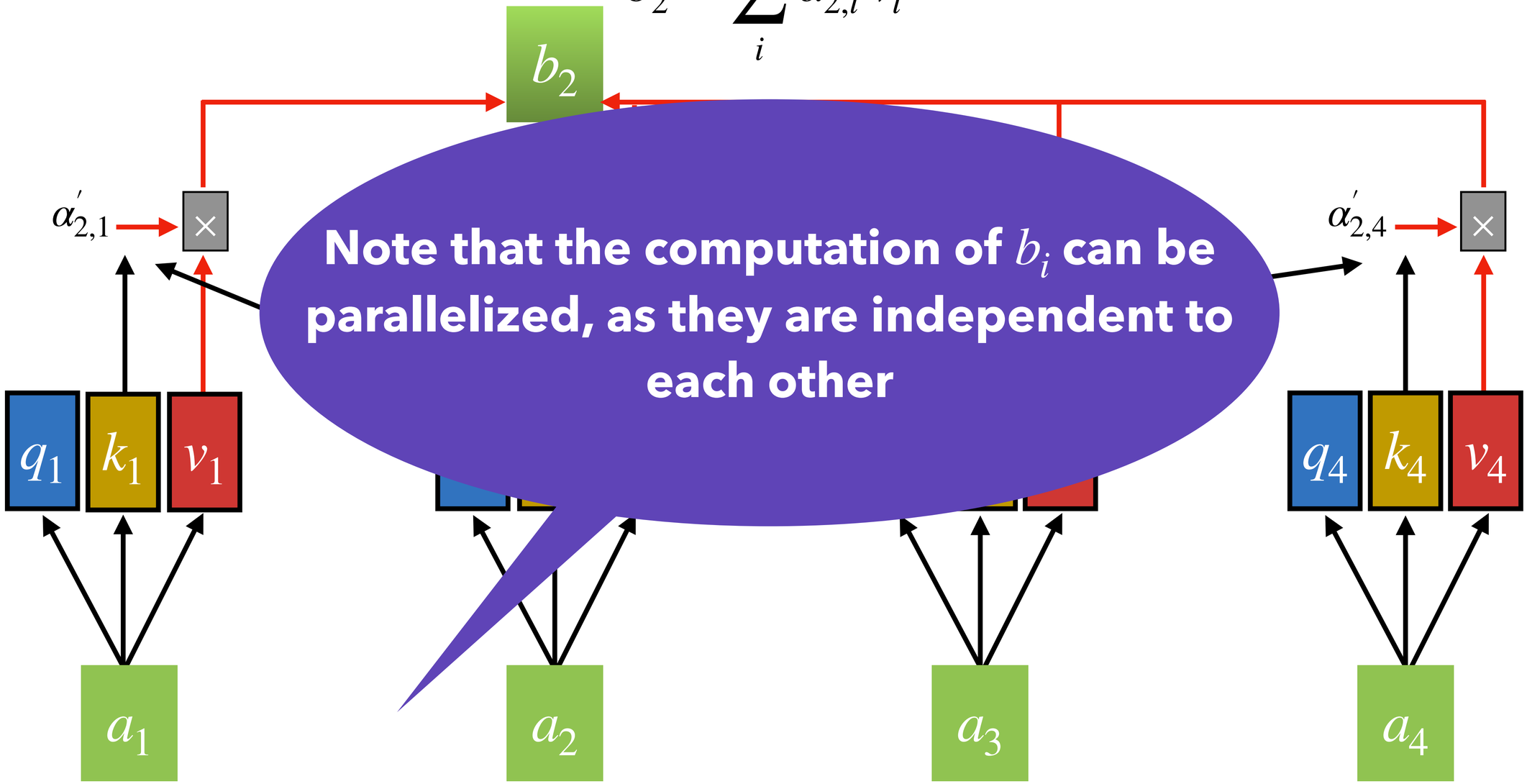
Repeat the same calculation for all a_i to obtain b_i

$$b_2 = \sum_i \alpha'_{2,i} \, v_i$$



Repeat the same calculation for all a_i to obtain b_i

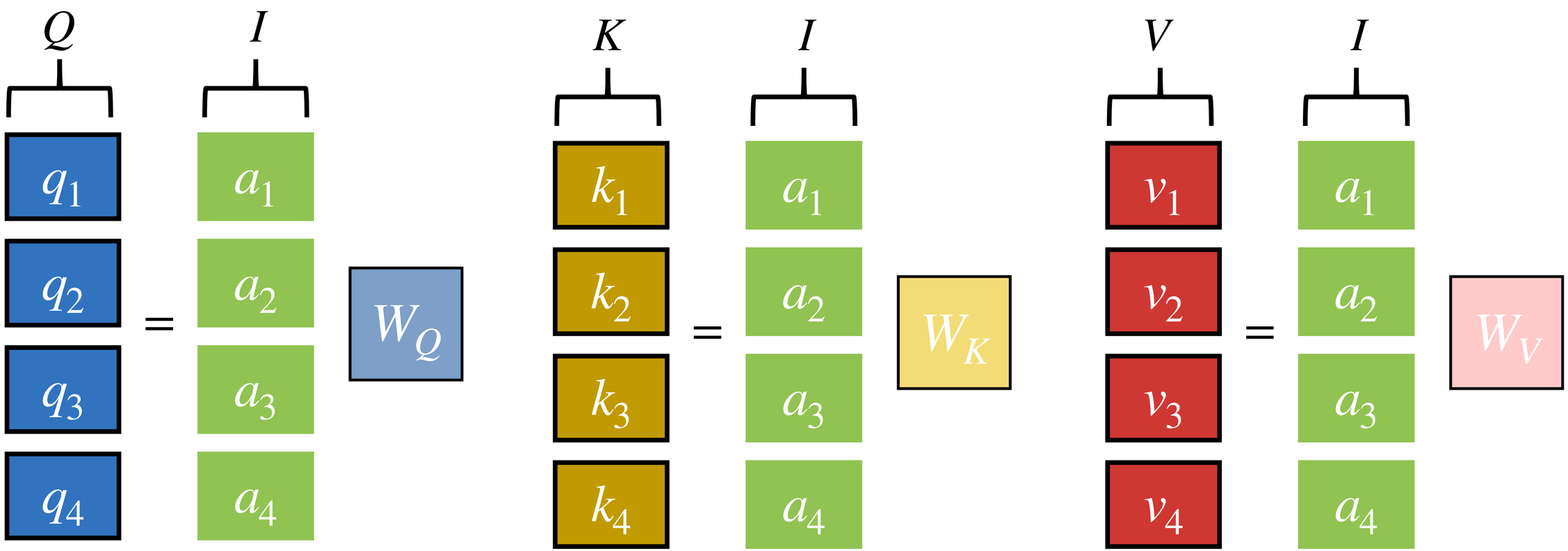
$$b_2 = \sum_i \alpha'_{2,i} \, v_i$$



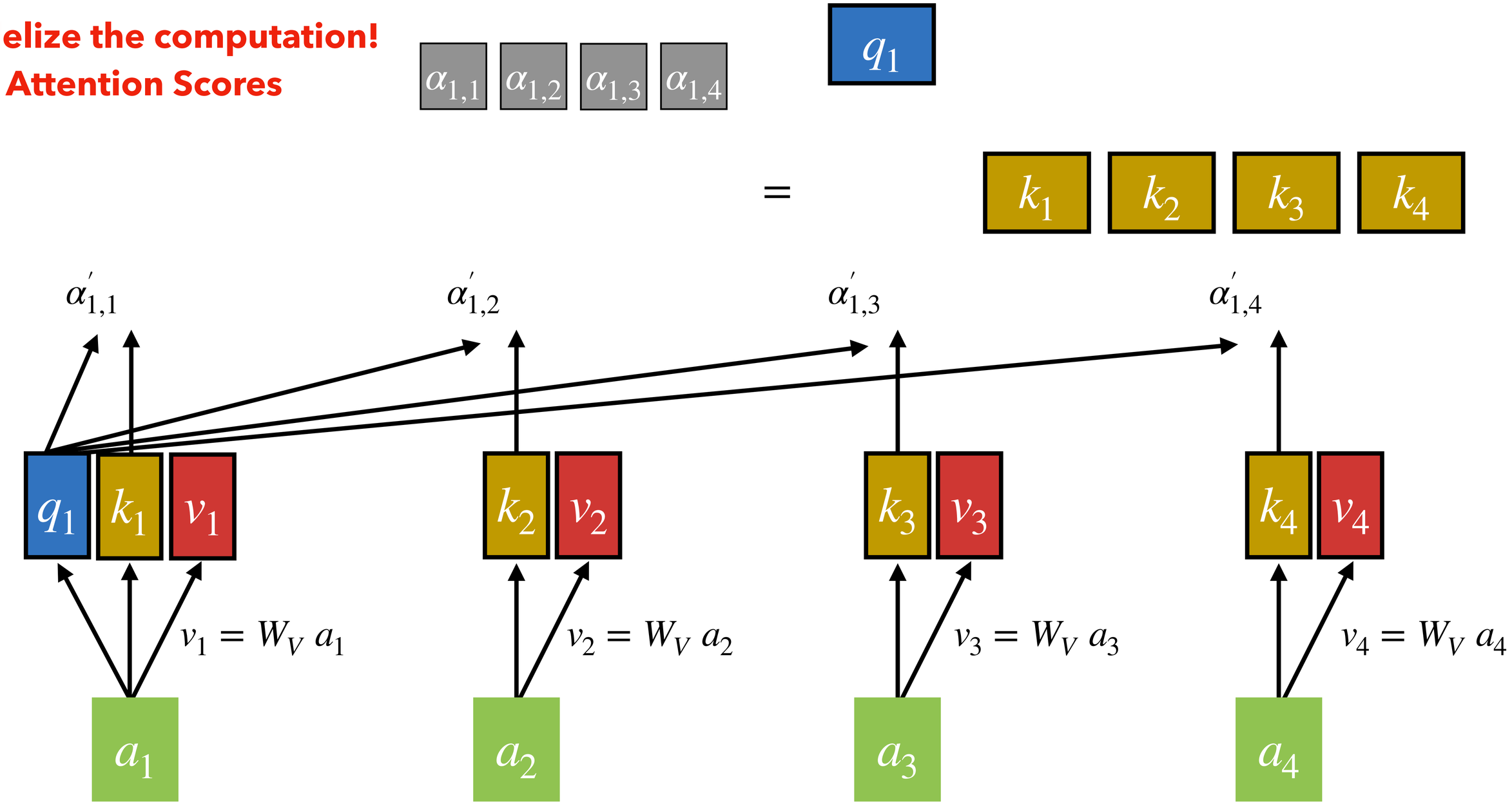
Note that the computation of b_i can be parallelized, as they are independent to each other

Parallelize the computation!

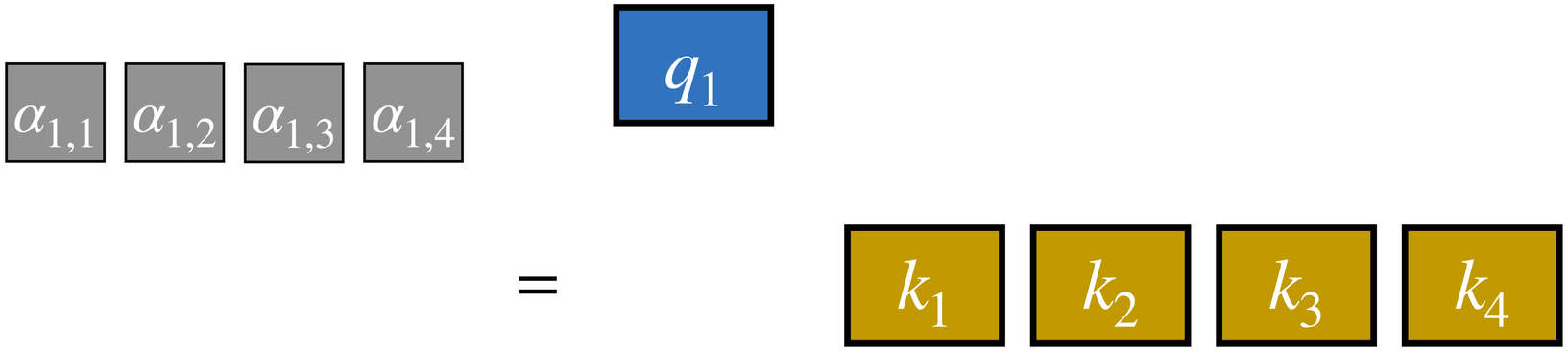
QKV



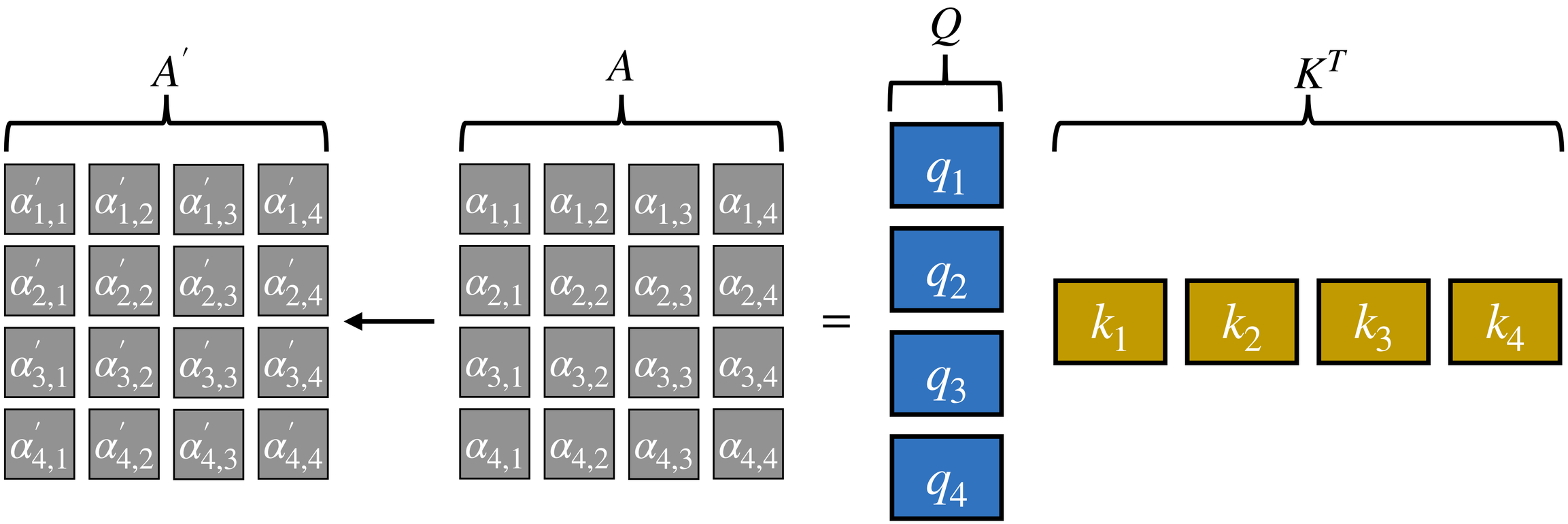
Parallelize the computation!
Attention Scores

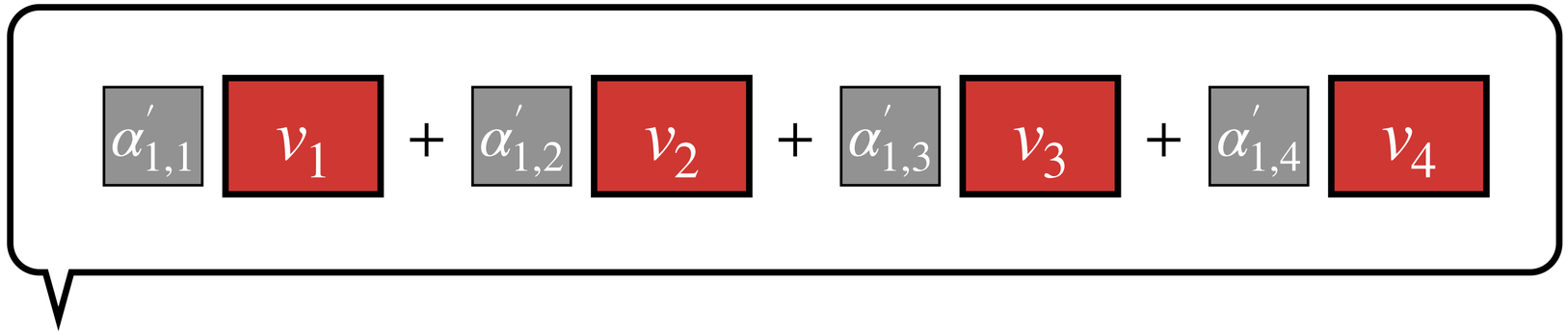


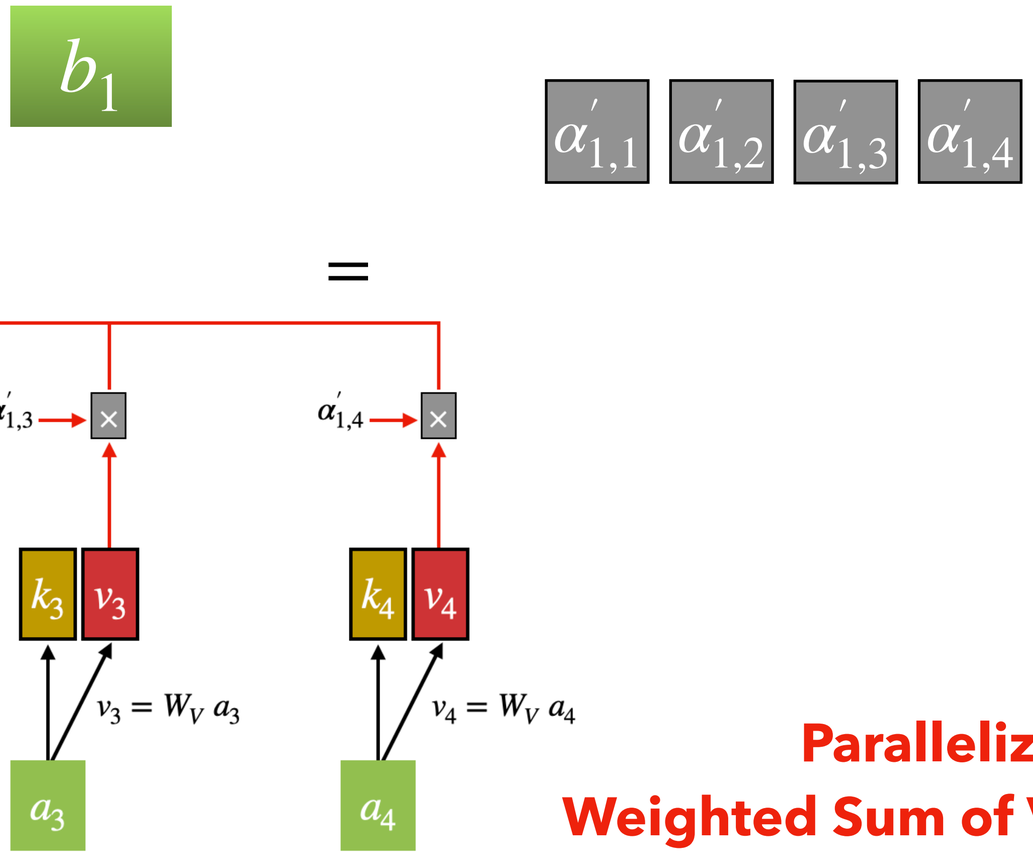
Parallelize the computation!
Attention Scores



Parallelize the computation!
Attention Scores



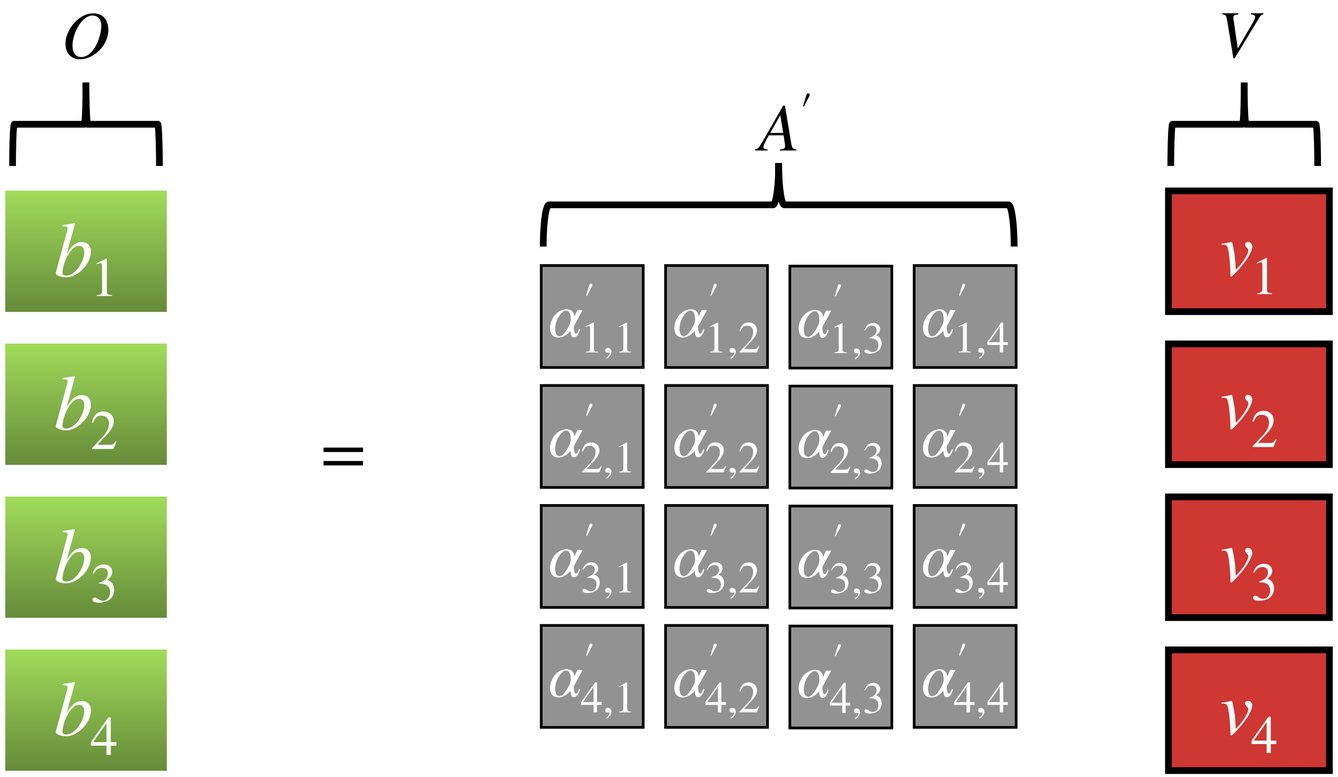




Parallelize the computation!

Weighted Sum of Values with Attention Scores

Parallelize the computation!



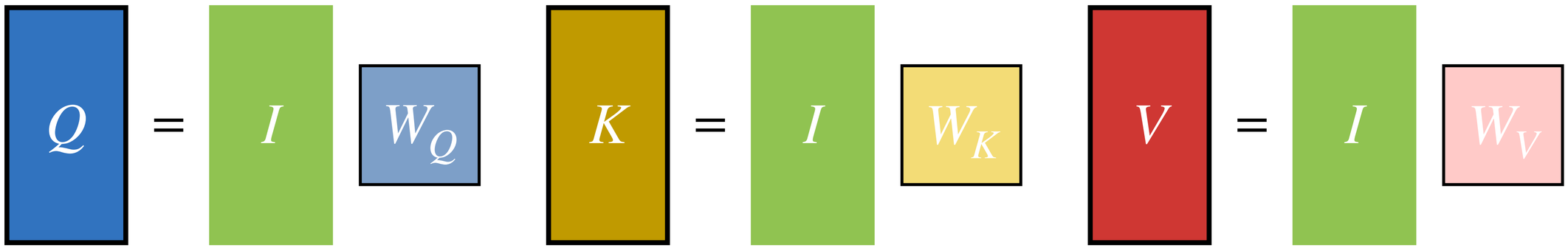
Parallelize the computation!

Weighted Sum of Values with Attention Scores

$$Q = I\ W_Q$$

$$K = I\ W_K$$

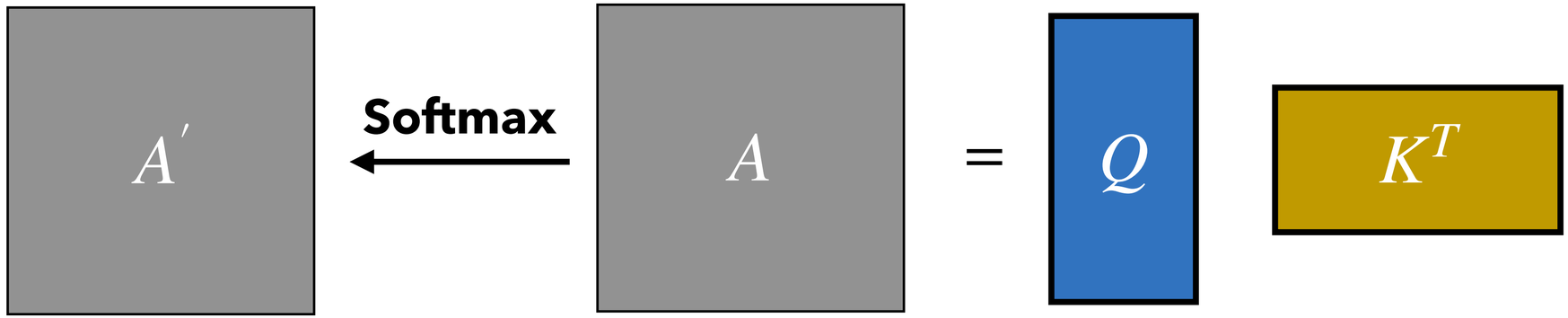
$$V = I\ W_V$$



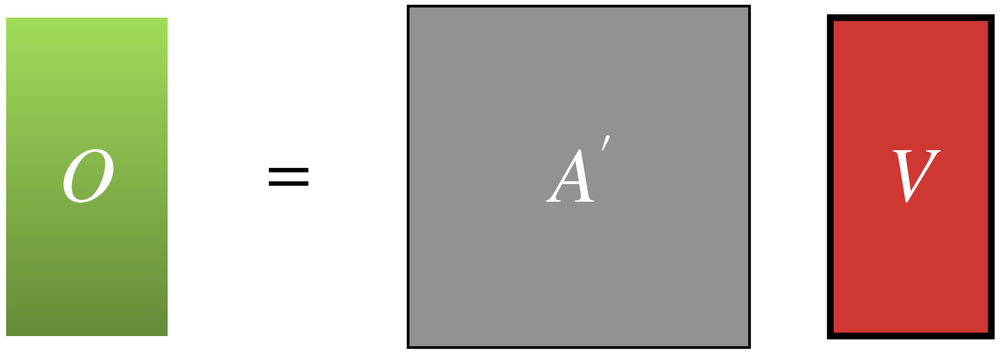
$$A = Q\ K^T$$

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

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



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



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 $a_i \in \mathbb{R}^d$

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

$Q, K, V \in$?

$$A = Q\ K^T$$
$$A = I\ W_Q\ (I\ W_K)^T = I\ W_Q\ W_K^T\ I^T$$
$$A^{'} = \text{softmax}(A)$$

$A^{'}, A \in$?

Dimensions?

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

$O \in$?

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 $a_i \in \mathbb{R}^d$

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

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

$$A = Q\ K^T$$
$$A = I\ W_Q\ (I\ W_K)^T = I\ W_Q\ W_K^T\ I^T$$
$$A^{'} = \text{softmax}(A)$$

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

Dimensions?

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

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

Self-Attention: Summary

Let $w_{1:n}$ be a sequence of words in vocabulary V, like *Steve Jobs founded Apple*.

For each w_i, let $a_i = Ew_i$, where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix.

1. Transform each word embedding with weight matrices W_Q, W_K, W_V, each in $\mathbb{R}^{d \times d}$

$$q_i = W_Q\, a_i \text{ (queries)} \qquad k_i = W_K\, a_i \text{ (keys)} \qquad v_i = W_V\, a_i \text{ (values)}$$

2. Compute pairwise similarities between keys and queries; normalize with softmax

$$\alpha_{i,j} = k_j\, q_i \qquad \alpha'_{i,j} = \frac{e^{\alpha_{i,j}}}{\sum_j e^{\alpha_{i,j}}}$$

3. Compute output for each word as weighted sum of values

$$b_i = \sum_j \alpha'_{i,j}\, v_j$$

Limitations and Solutions of Self-Attention

No Sequence Order	→	Position Embedding
No Nonlinearities	→	Adding Feed-forward Networks
Looking into the Future	→	Masking

Limitations and Solutions of Self-Attention

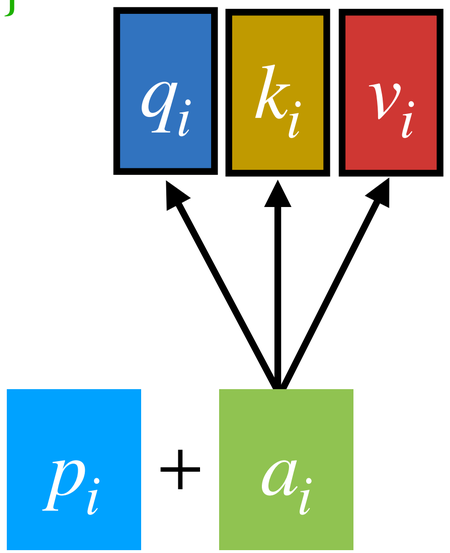
No Sequence Order	→	Position Embedding
No Nonlinearities	→	Adding Feed-forward Networks
Looking into the Future	→	Masking

No Sequence Order → Position Embedding

- All tokens in an input sequence are **simultaneously** fed into self-attention blocks. Thus, there's no difference between tokens at different positions.
  - **We lose the position info!**

- **How do we bring the position info back, just like in RNNs?**
  - **Representing each sequence index as a vector:** $p_i \in \mathbb{R}^d$, for $i \in \{1,\ldots,n\}$

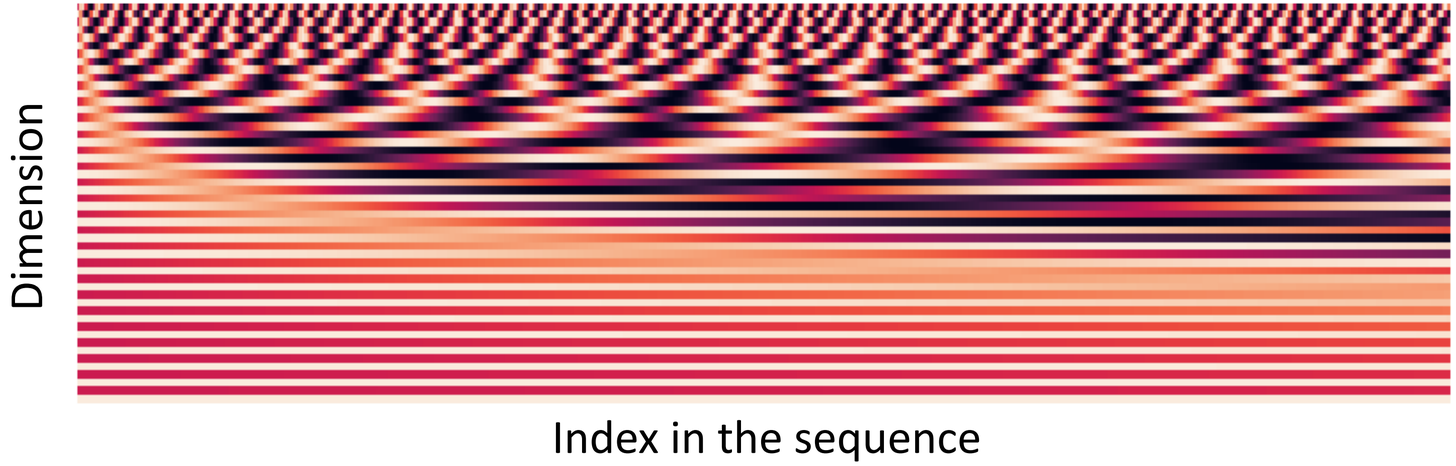
- **How to incorporate the position info into the self-attention blocks?**
  - **Just add the p_i to the input:** $\hat{a}_i = a_i + p_i$
  - where a_i is the embedding of the word at index i.
  - In deep self-attention networks, we do this at the **first layer**.
  - We can also concatenate a_i and p_i, but more commonly we add them.



Position Representation Vectors via Sinusoids

Sinusoidal Position Representations (from the original Transformer paper): concatenate sinusoidal functions of varying periods.

$$\boldsymbol{p}_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

- Periodicity indicates that maybe "absolute position" isn't as important
- Maybe can extrapolate to longer sequences as periods restart!

- Not learnable; also the extrapolation doesn't really work!

Learnable Position Representation Vectors

Learned absolute position representations: p_i contains learnable parameters.

- Learn a matrix $p \in \mathbb{R}^{d \times n}$, and let each p_i be a column of that matrix
- Most systems use this method.

- **Flexibility:** each position gets to be learned to fit the data

- Cannot extrapolate to indices outside $1,\ldots,n$.

Sometimes people try more flexible representations of position:

- Relative linear position attention [Shaw et al., 2018]
- Dependency syntax-based position [Wang et al., 2019]

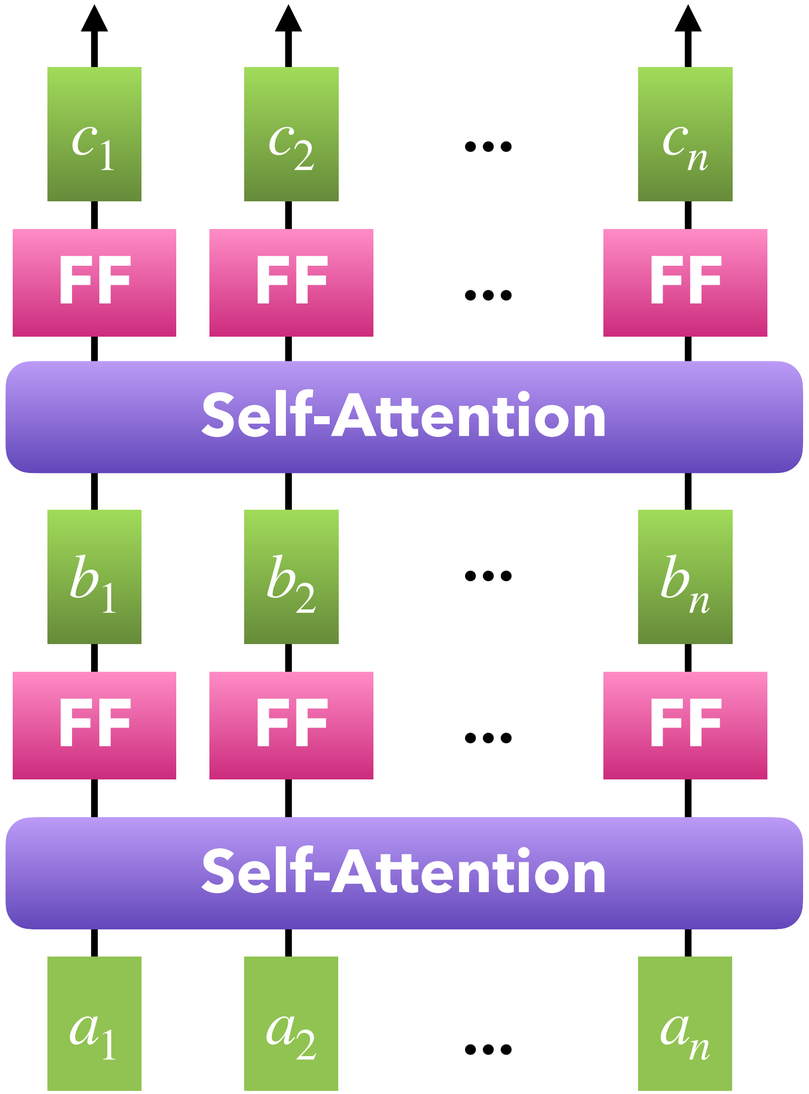
Limitations and Solutions of Self-Attention

No Sequence Order	→	Position Embedding
No Nonlinearities	→	Adding Feed-forward Networks
Looking into the Future	→	Masking

No Nonlinearities → Add Feed-forward Networks

There are **no element-wise nonlinearities** in self-attention; stacking more self-attention layers just re-averages value vectors.

Easy Fix: add a feed-forward network to post-process each output vector.



Limitations and Solutions of Self-Attention

No Sequence Order	→	Position Embedding
No Nonlinearities	→	Adding Feed-forward Networks
Looking into the Future	→	Masking

Looking into the Future → Masking

- In **decoders** (language modeling, producing the next word given previous context), we need to ensure we **don't** peek at the future.
- At every time-step, we could change the set of keys and queries to include only past words. **(Inefficient!)**
- To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$.

$$\alpha_{i,j} = \begin{cases} q_i \, k_j, \; j \leq i \\ -\infty, \; j > i \end{cases}$$

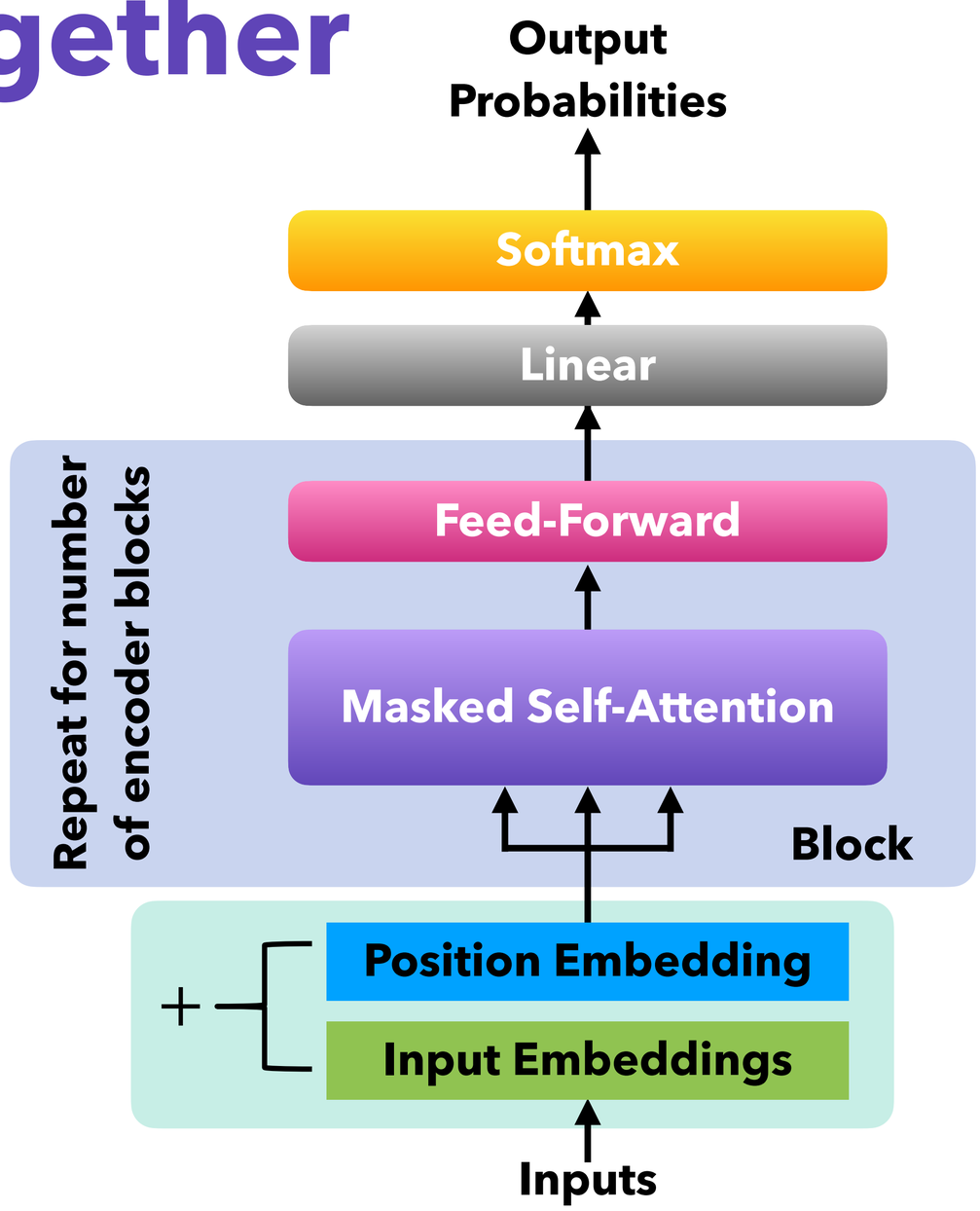
We can look at these (not greyed out) words

For encoding these words

	[START]	The	chef	who
[START]		$-\infty$	$-\infty$	$-\infty$
The			$-\infty$	$-\infty$
chef				$-\infty$
who				

Now We Put Things Together

- **Self-attention**
  - The basic computation
- **Positional Encoding**
  - Specify the sequence order
- **Nonlinearities**
  - Adding a feed-forward network at the output of the self-attention block
- **Masking**
  - Parallelize operations (looking at all tokens) while not leaking info from the future

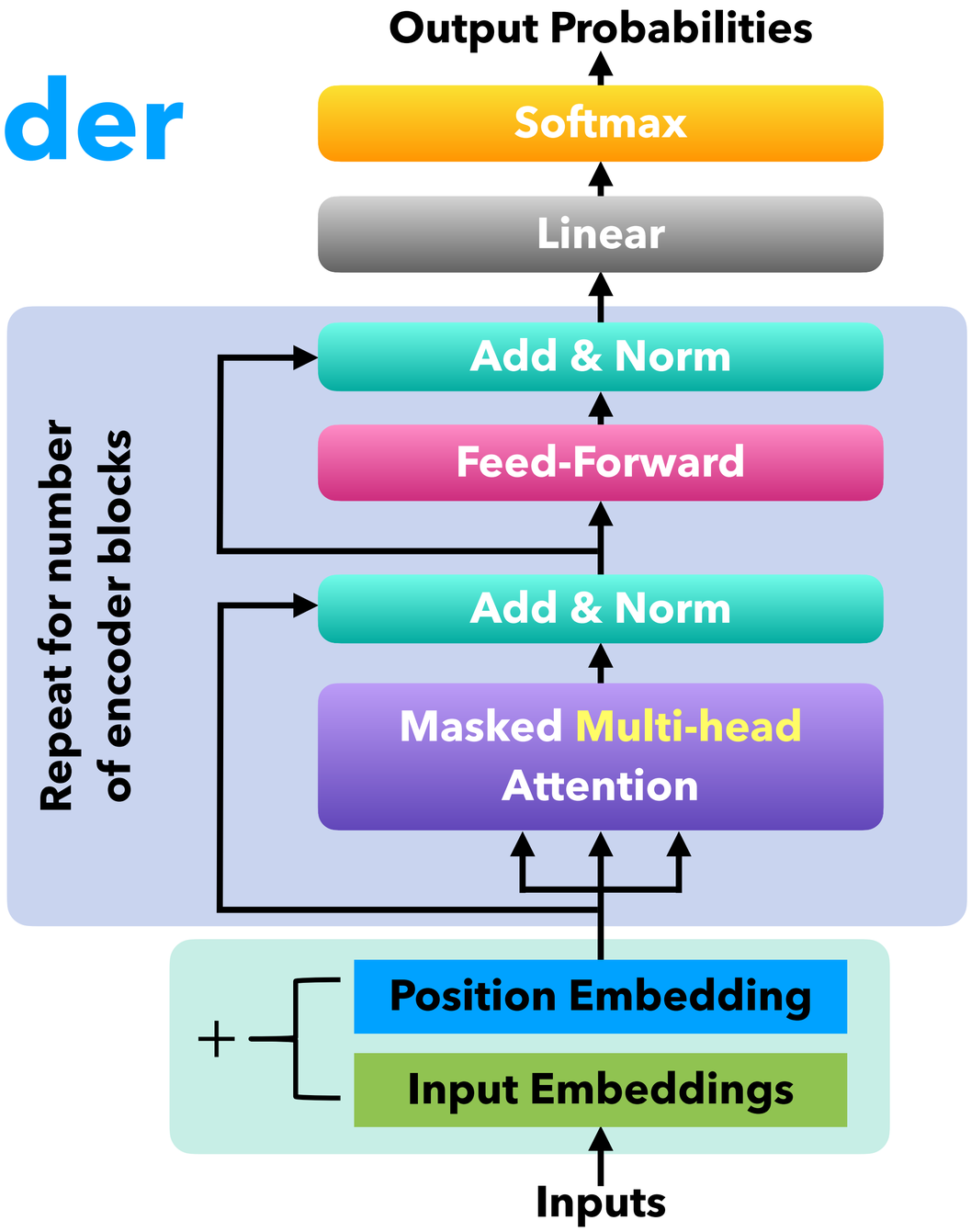


The Transformer Decoder

- A **Transformer decoder** is what we use to build systems like language models.

- It's a lot like our minimal self-attention architecture, but with a few more components.
  - ***Residual connection ("Add")***
  - ***Layer normalization ("Norm")***

- Replace self-attention with **multi-head self-attention**.

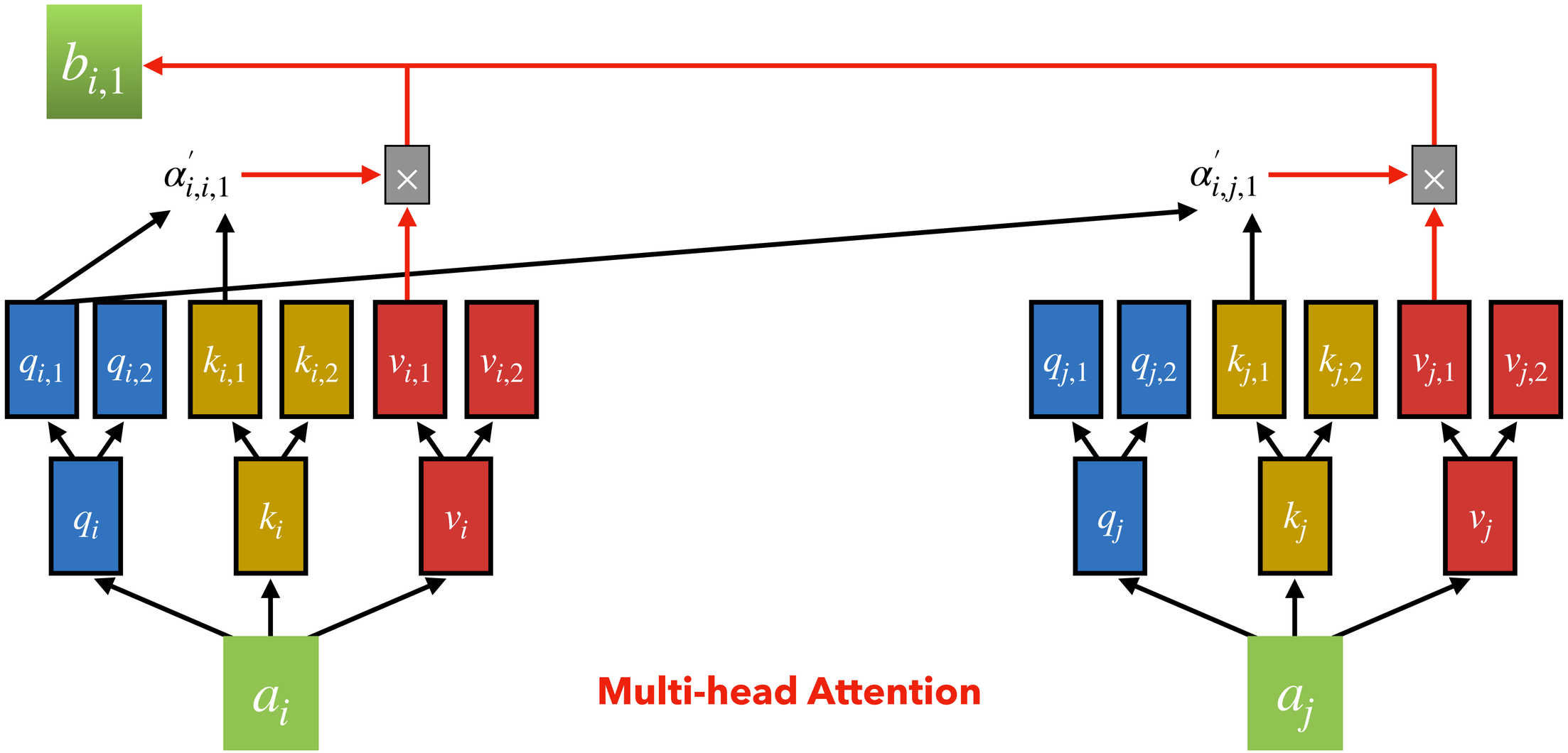


Why Multi-head Attention?

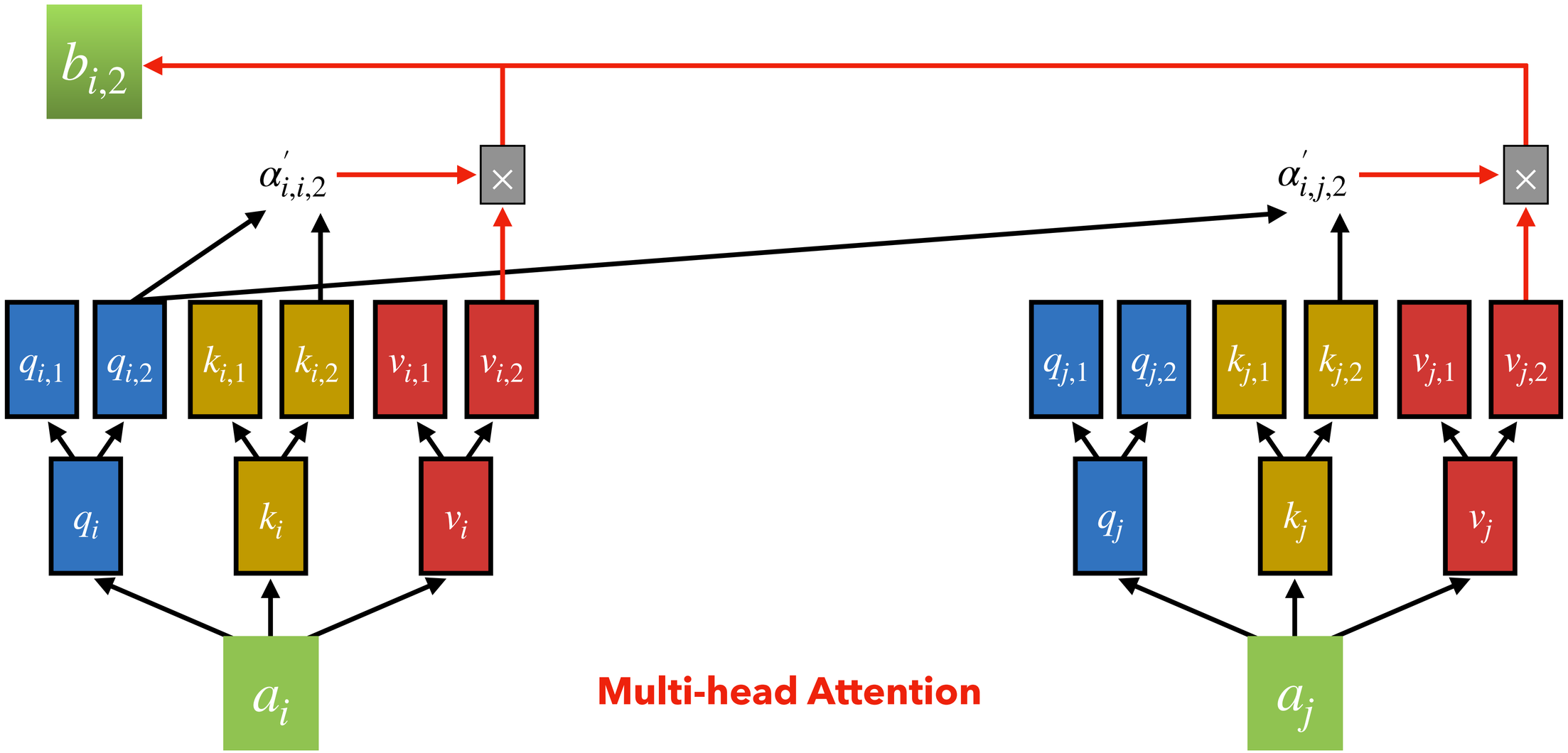
What if we want to look in multiple aspects in the sentence at once?

Instead of having only one attention head, we can create multiple sets of (queries, keys, values) independent from each other!

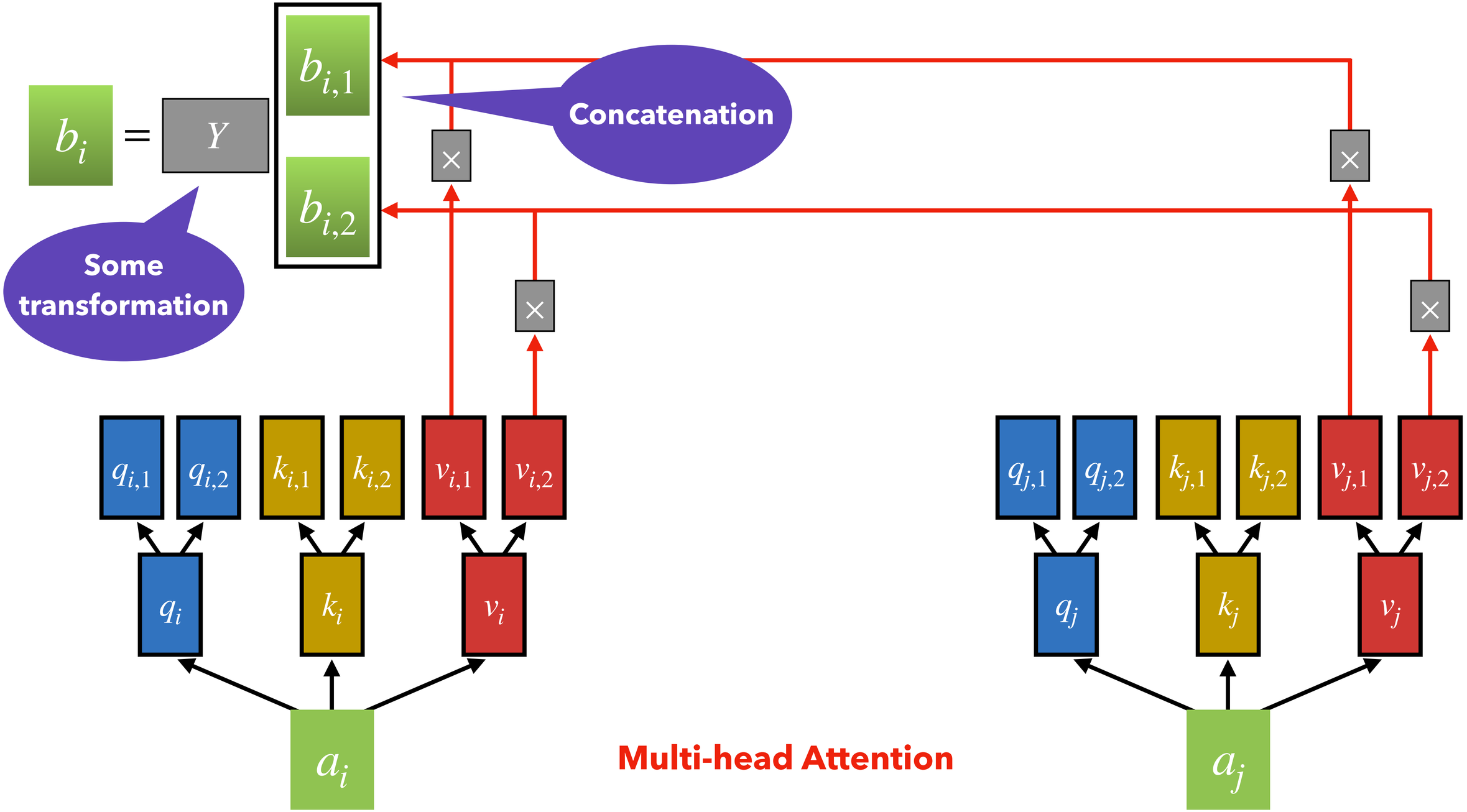
Multi-Head Attention: Walk-through



Multi-head Attention



Multi-head Attention



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 $a_i \in \mathbb{R}^d$

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

$Q^l, K^l, V^l \in$?

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

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

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

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

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

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

$[O^1; \ldots; O^h] \in$?

$O \in \mathbb{R}$?

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 $a_i \in \mathbb{R}^d$

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

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

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

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

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

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

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

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

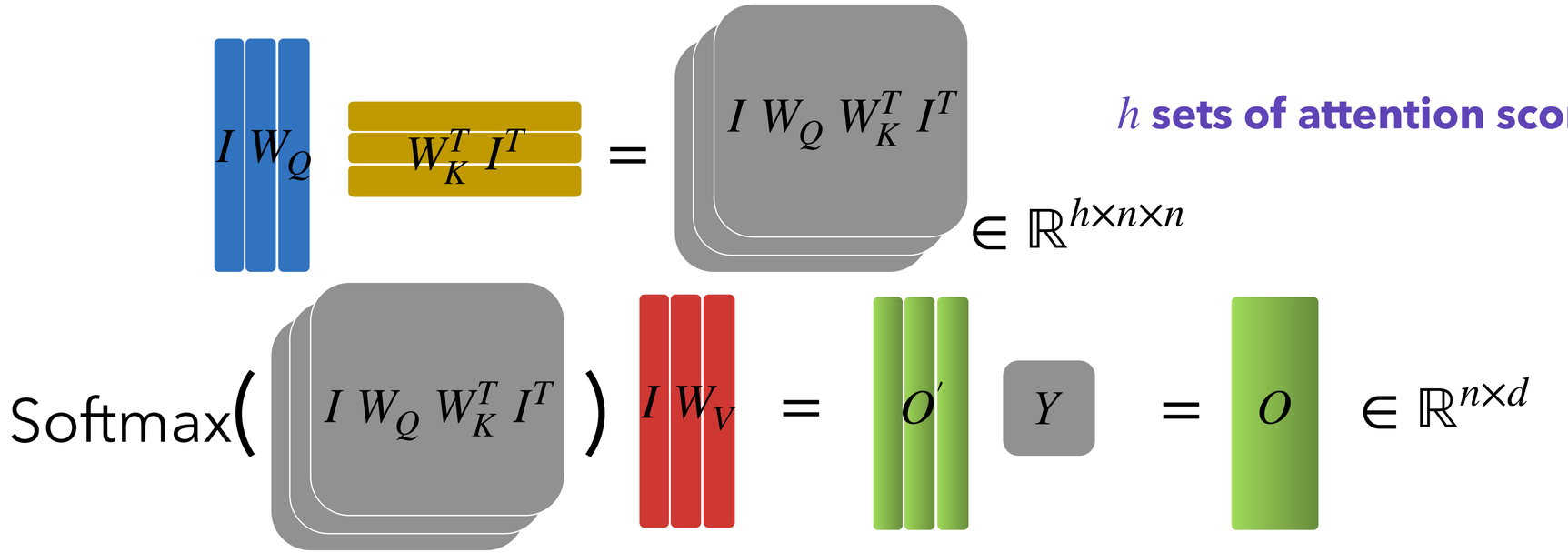
$[O^1; \ldots; O^h] \in \mathbb{R}^{n \times d}$

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

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!



h sets of attention scores!

Scaled Dot Product [Vaswani et al., 2017]

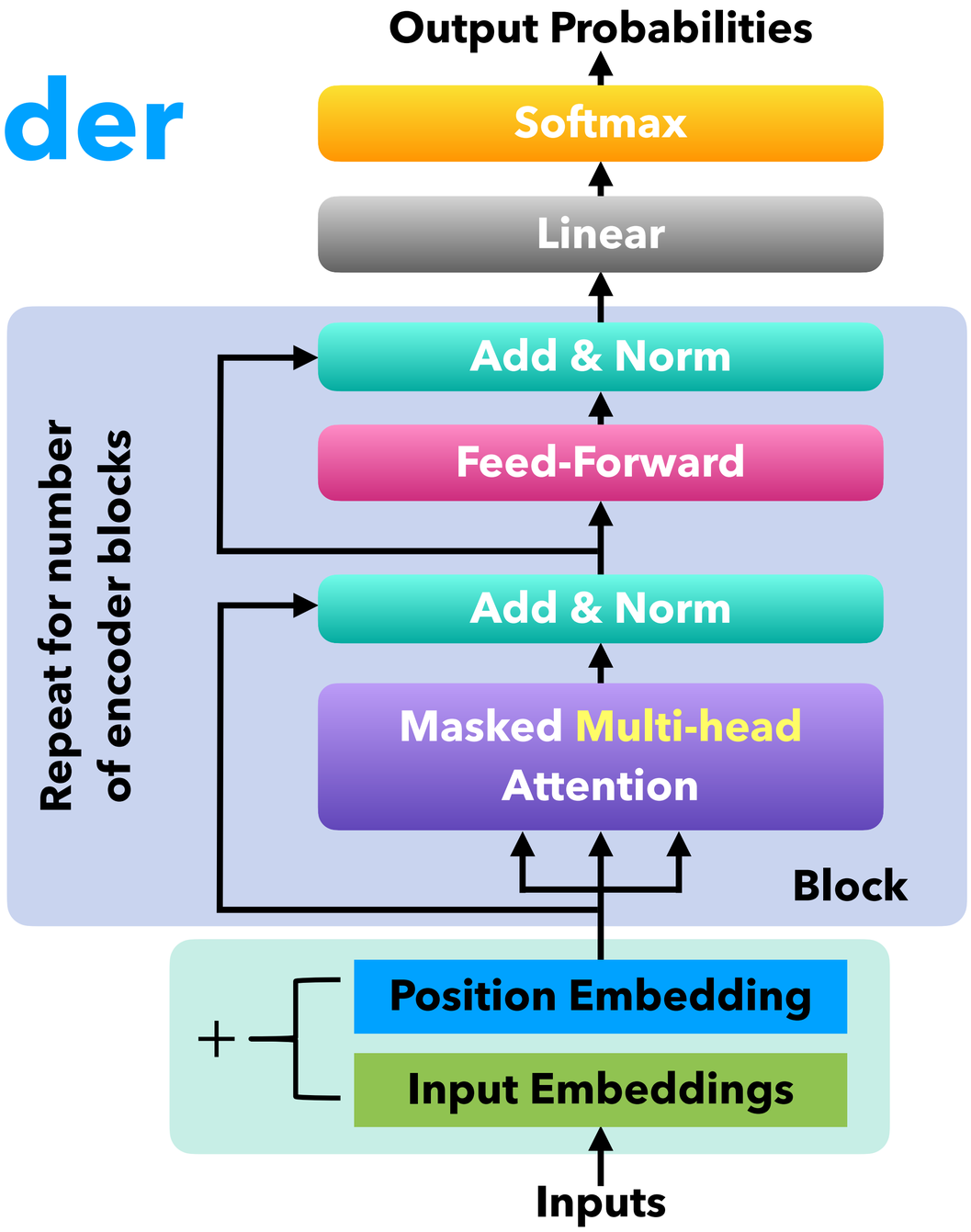
- **"Scaled Dot Product"** attention aids in training.
- When dimensionality d becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.

- 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")***

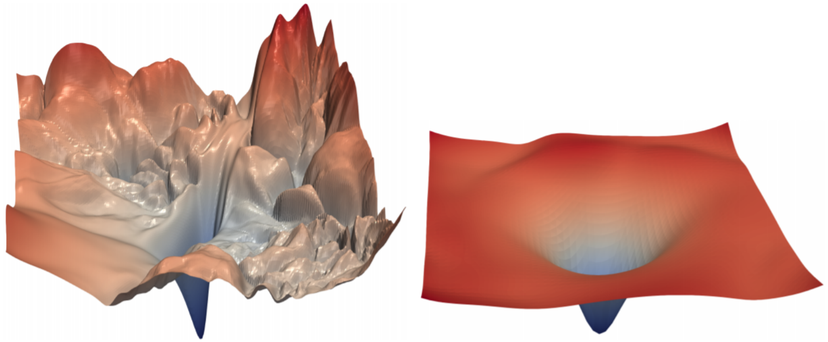


Residual Connections [He et al., 2016]

- 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 [Ba et al., 2016]

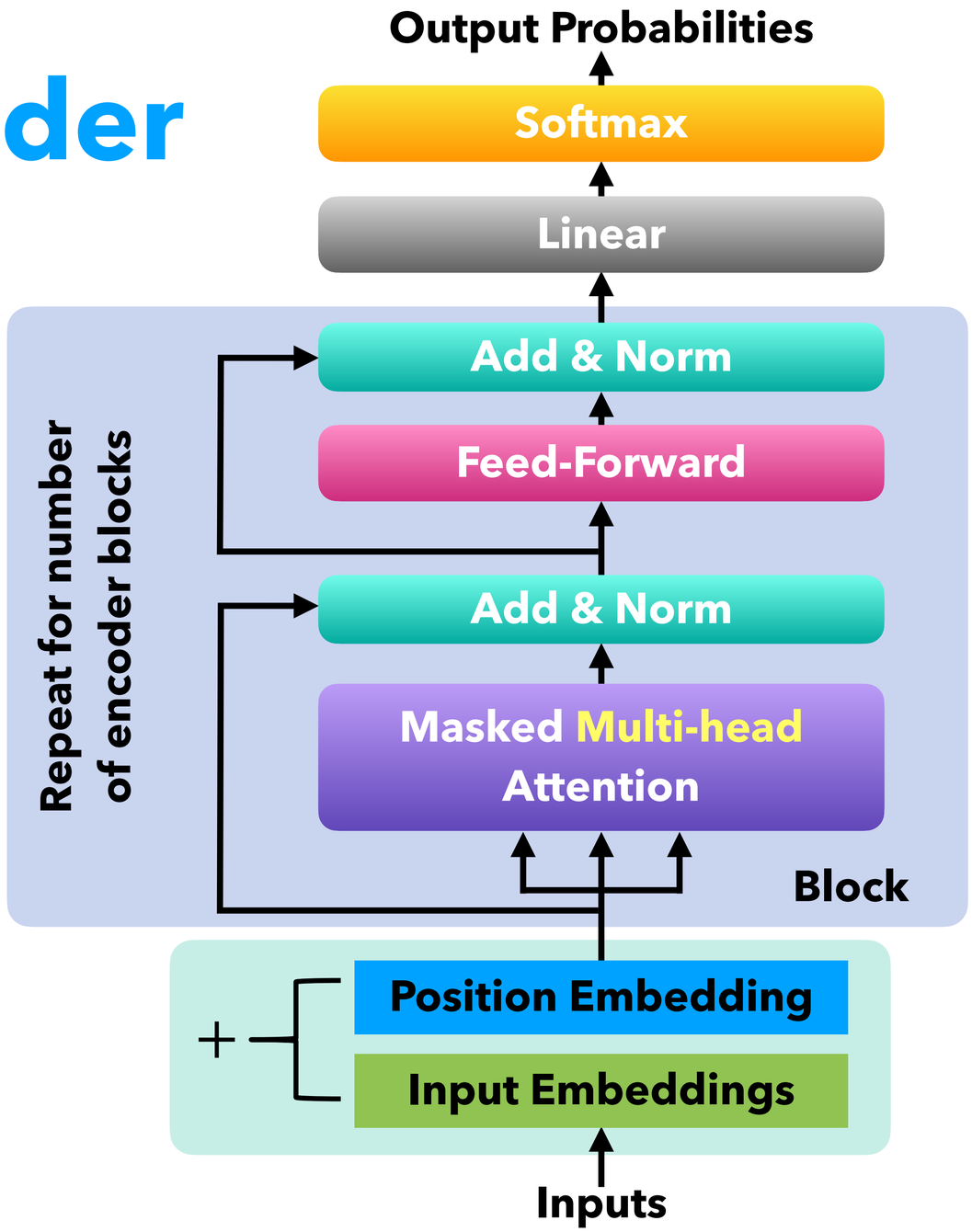
- Layer normalization is a trick to help models train faster.
- **Idea:** cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within each layer.
  - LayerNorm's success may be due to its normalizing gradients [Xu et al., 2019]
- Let $x \in \mathbb{R}^d$ 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:
- $$\text{output} = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned element-wise gain and bias

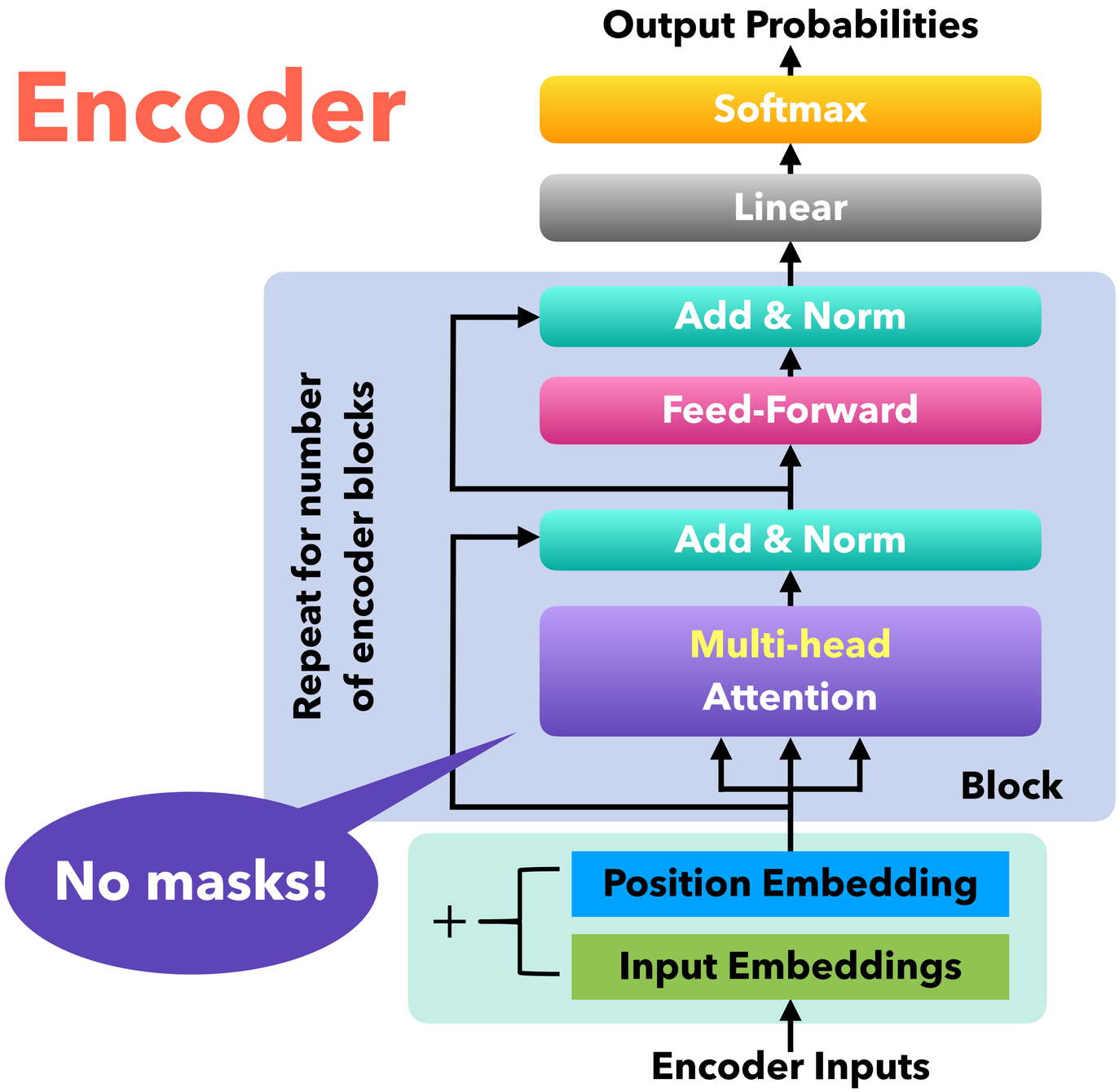
The Transformer Decoder

- 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



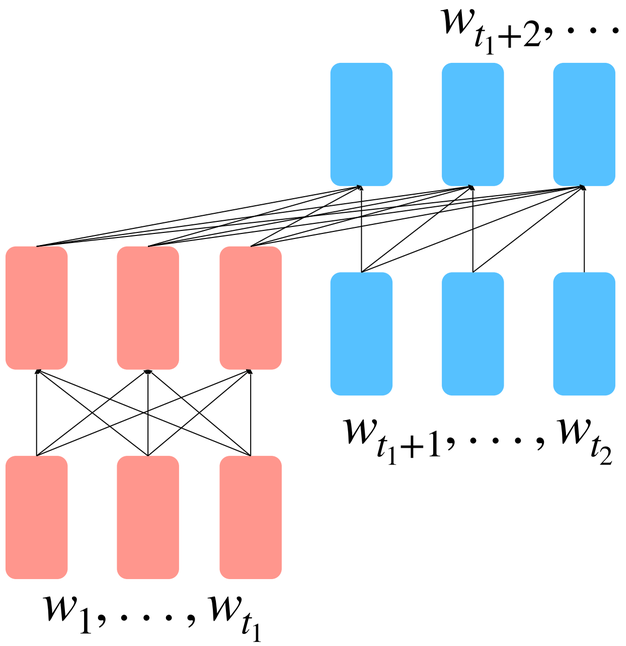
The Transformer Encoder

- 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.

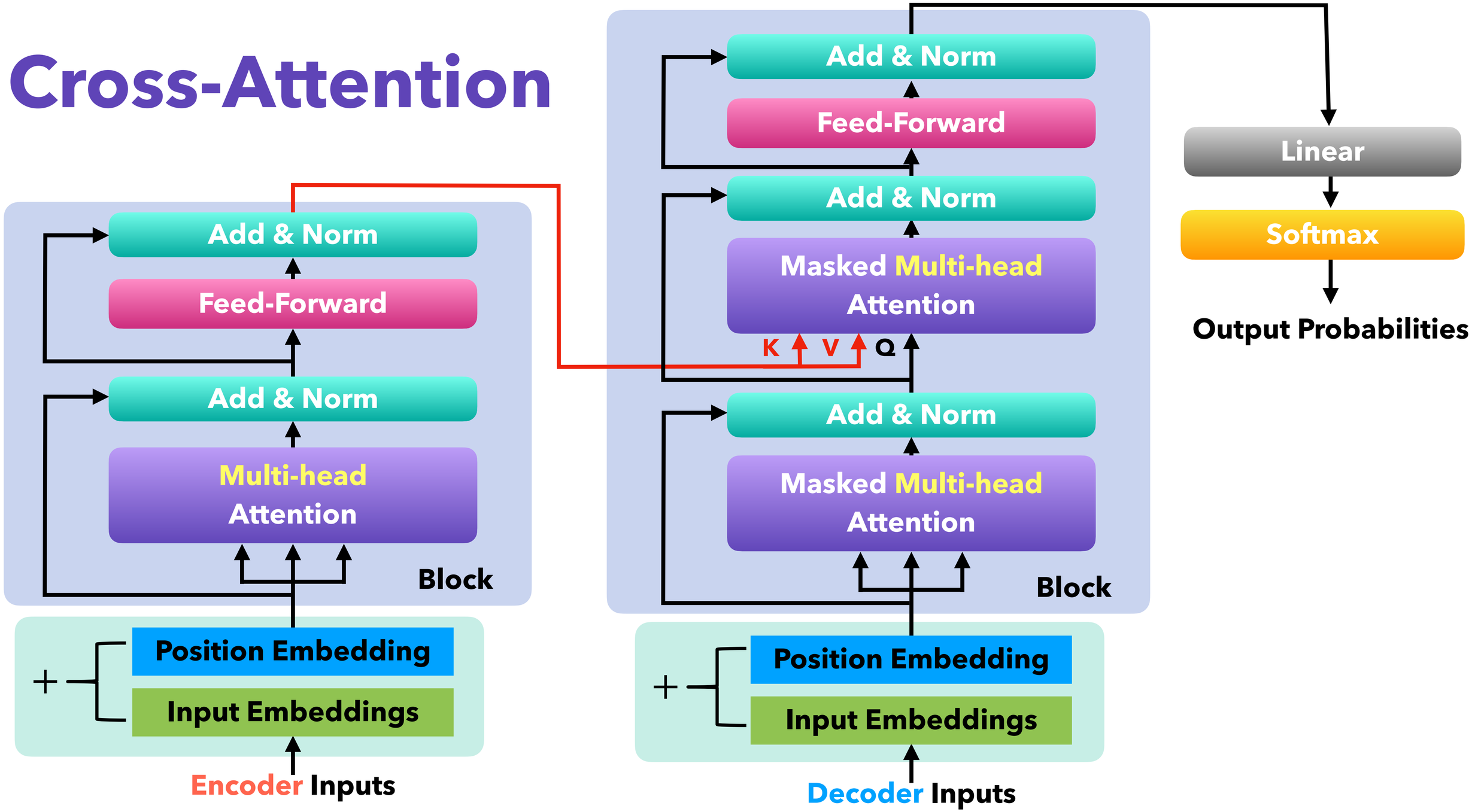


The Transformer Encoder-Decoder

- 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



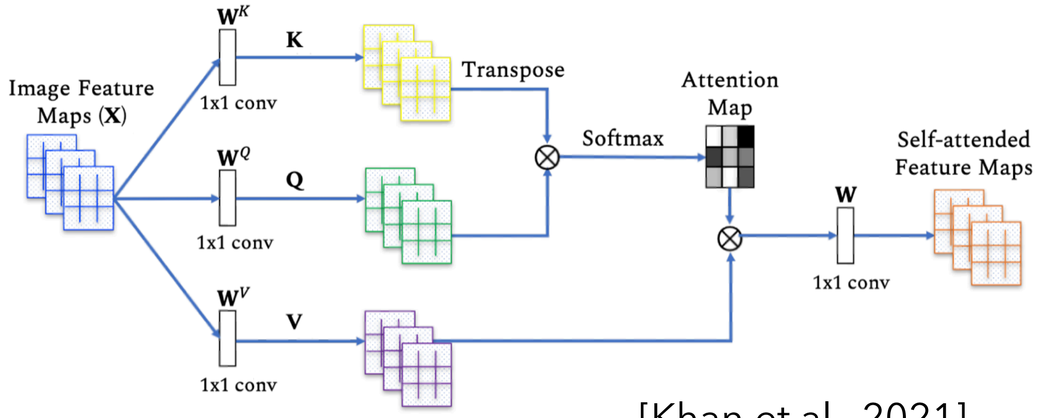
Cross-Attention Details

- **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**:
  - $q_i = W_Q z_i$

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.

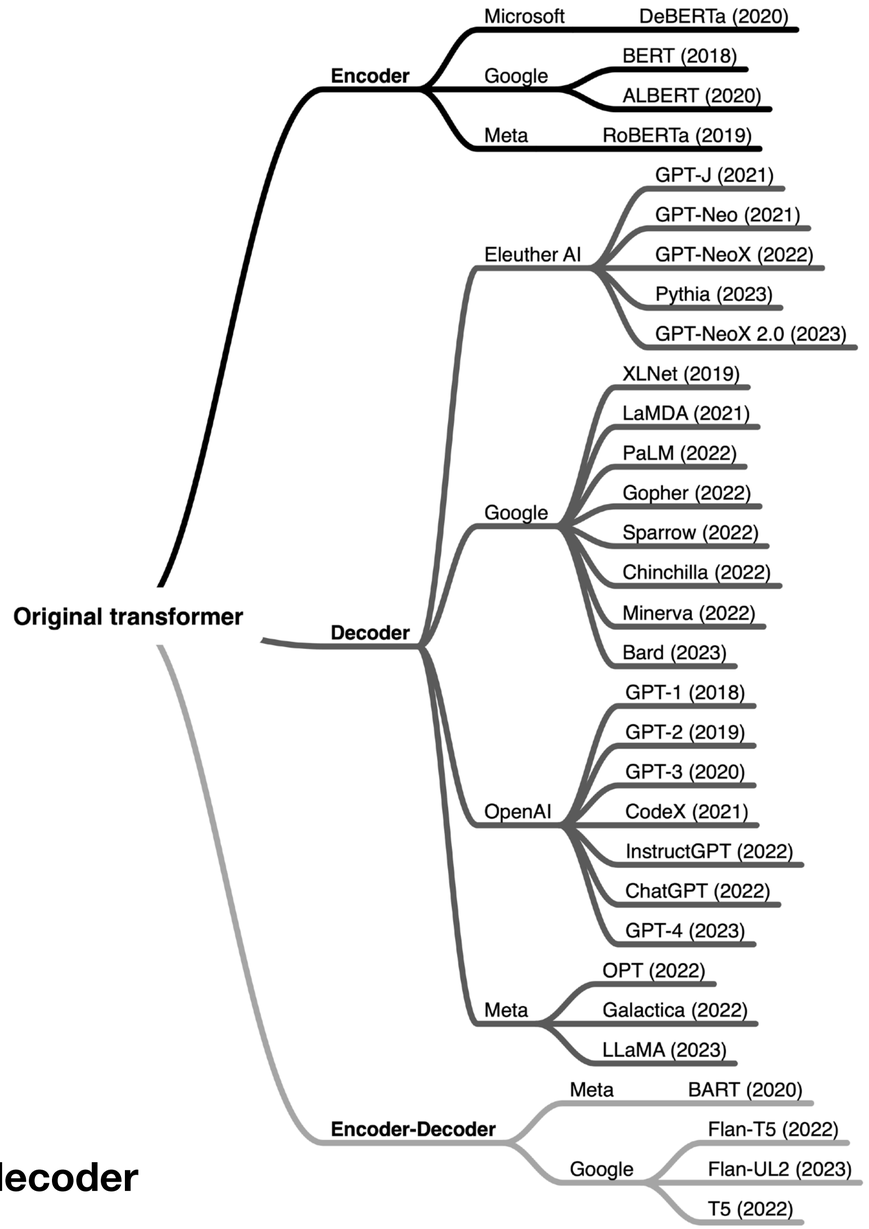
- Since Transformer has been popularized in language applications, computer vision also adapted Transformers, e.g., **Vision Transformers**.



[Khan et al., 2021]

What's next after Transformers?

Transformer types and examples



https://magazine.sebastianraschka.com/p/understanding-encoder-and-decoder

Pretraining

Consider the the task of **Sentiment Analysis**

Food Review: "I recently had the pleasure of dining at Fusion Bites, and the experience was nothing short of spectacular. The menu boasts an exciting blend of global flavors, and each dish is a masterpiece in its own right."

Say that we are given a dataset of 100K food reviews with sentiment labels, **how do we train a model to perform sentiment analysis over unseen food reviews?**

We can directly train a randomly initialized model to take in food review texts and output "positive" or "negative" sentiment labels.

Consider the the task of **Sentiment Analysis**

Food Review: “I recently had the pleasure of dining at Fusion Bites, and the experience was nothing short of spectacular. The menu boasts an exciting blend of global flavors, and each dish is a masterpiece in its own right.”

Movie Review: "The narrative unfolds with a steady pace, showcasing a blend of various elements. While the performances are competent, and the cinematography captures the essence of the story, the overall impact falls somewhere in the middle."

If we are instead given **movie reviews** to classify, can we use the same system trained from food reviews to predict the sentiment?

May NOT generalize well due to distributional shift!

Lots of Information in Raw Texts

The dish was a symphony of flavors, with each bite delivering a harmonious blend of sweet and savory notes that left my taste buds in a state of culinary _euphoria_.

The dish fell short of expectations, as the flavors lacked depth and the texture was disappointingly bland, leaving me with a sense of culinary _letdown_.

Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was _disappointing_.

Despite a promising premise, the movie failed to live up to its potential, as the plot felt disjointed, the characters lacked depth, and the pacing left me disengaged, resulting in a rather _amazing_ cinematic experience.

Lots of Information in Raw Texts

Verb	I went to Hawaii for snorkeling, hiking, and whale _watching_.
Preposition	I walked across the street, checking for traffic __over__ my shoulders.
Commonsense	I use ___knife___ and fork to eat steak.
Time	Ruth Bader Ginsburg was born in ___1933___.
Location	University of Washington is located at __Seattle__, Washington.
Math	I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, __34__.
Chemistry	Sugar is composed of carbon, hydrogen, and __oxygen__.

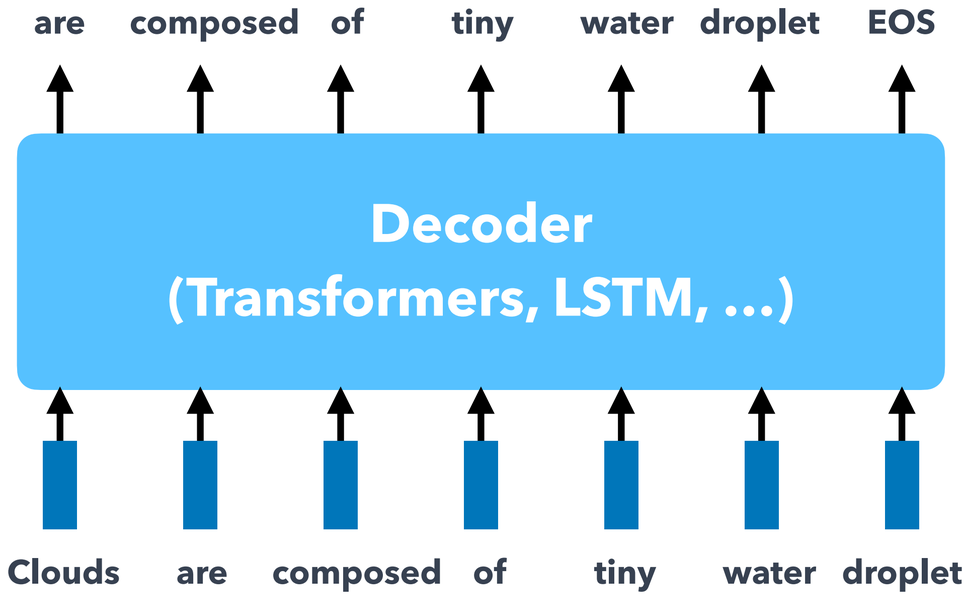
...

How to Harvest Underlying Patterns, Structures, and Semantic Knowledge from Raw Texts?

Pre-training! (aka self-supervised learning)

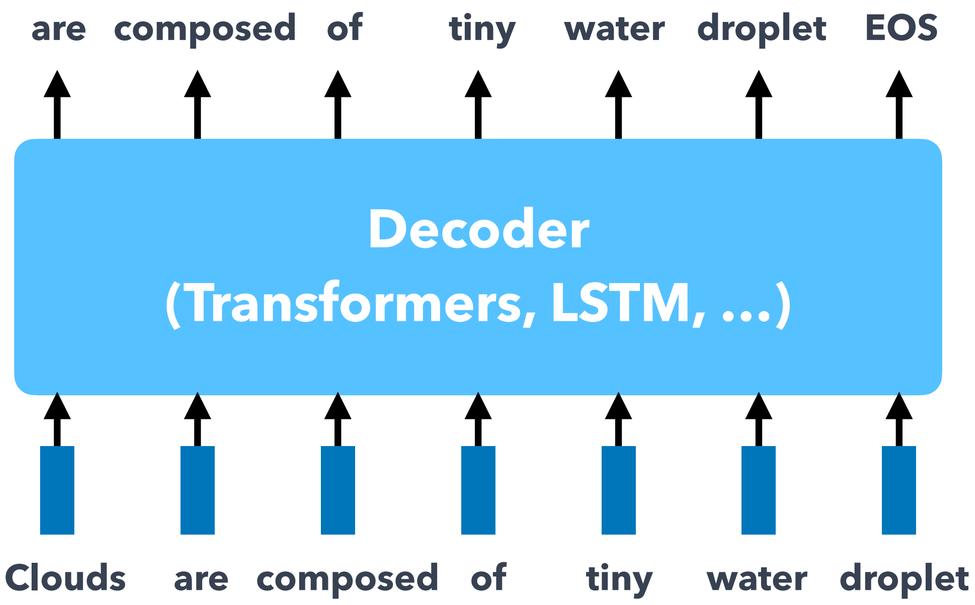
Self-supervised Pre-training for Learning Underlying Patterns, Structures, and Semantic Knowledge

- Pre-training through **language modeling** [Dai and Le, 2015]
  - Model $P_\theta(w_t \mid w_{1:t-1})$, the probability distribution of the next word given previous contexts.
  - **There's lots of (English) data for this!** E.g., books, websites.
  - **Self-supervised** training of a neural network to perform the language modeling task with massive raw text data.
  - Save the network parameters to reuse later.



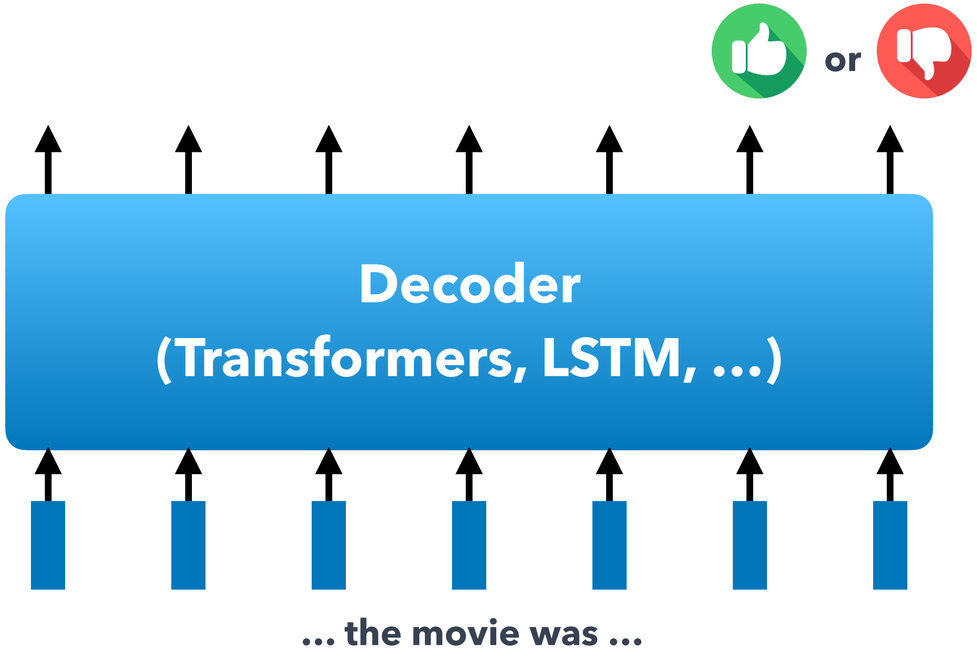
Supervised Fine-tuning for Specific Tasks

Step 1:
Pre-training



Abundant data; learn general language

Step 2:
Fine-tuning



Limited data; adapt to the task

Advantages of Pre-training & Fine-tuning

- **Leveraging rich underlying information** from abundant raw texts.
- **Reducing the reliance of task-specific labeled data** that is difficult or costly to obtain.
- **Initializing model parameters** for more **generalizable** NLP applications.
- **Saving training cost** by providing a reusable model checkpoints.
- **Providing robust representation** of language contexts.

Caveat: Catastrophic Forgetting

- **Sequentially** pre-train then fine-tune may result in **catastrophic forgetting**, meaning that **while adapting to the new fine-tuning task, the model may lose previously learned information.**
- However, as modern language models are becoming larger in size and are pre-trained on massive raw text, they do encode tremendous amount of valuable information. **Thus, it's generally still more helpful to leverage information learned from the pre-training stage, than training on a task completely from scratch.**

3 Pre-training Paradigms/Architectures

Encoder

- E.g., BERT, RoBERTa, DeBERTa, …
- **Autoencoder** model
- **Masked** language modeling

Encoder-Decoder

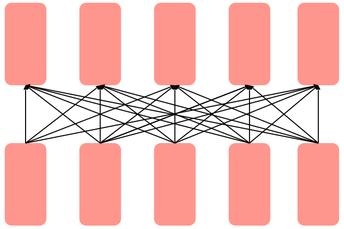
- E.g., T5, BART, …
- **seq2seq** model

Decoder

- E.g., GPT, GPT2, GPT3, …
- **Autoregressive** model
- **Left-to-right** language modeling

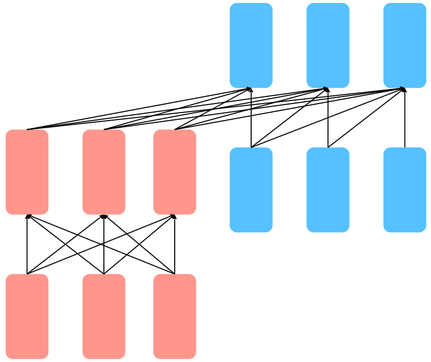
3 Pre-training Paradigms/Architectures

Encoder



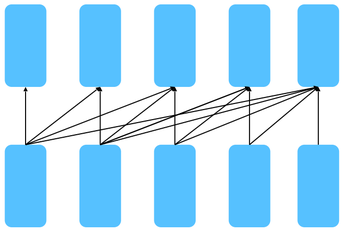
- Bidirectional; can condition on the future context

Encoder-Decoder



- Map two sequences of different length together

Decoder

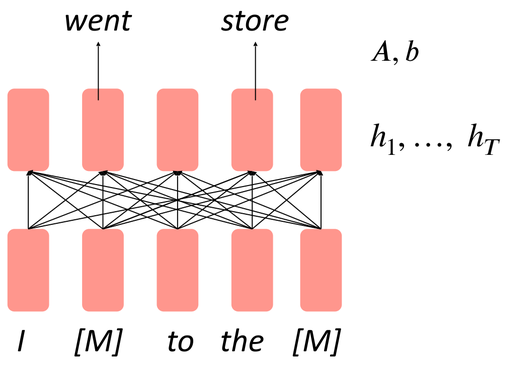


- Language modeling; can only condition on the past context

Encoder: Training Objective

[Devlin et al., 2018]

- How to encode information from both **bidirectional** contexts?
- General Idea: **text reconstruction!**
  - Your time is limited so don't waste it living someone else's life. Don't be trapped by dogma which is living with the results of other people's thinking. – Steve Jobs



$$h_1, \ldots, h_T = \text{Encoder}(w_1, \ldots, w_T)$$

$$y_i \sim Aw_i + b$$

Only add loss terms from the masked tokens. If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x \mid \tilde{x})$. Called **Masked Language model (MLM)**.

Encoder: BERT

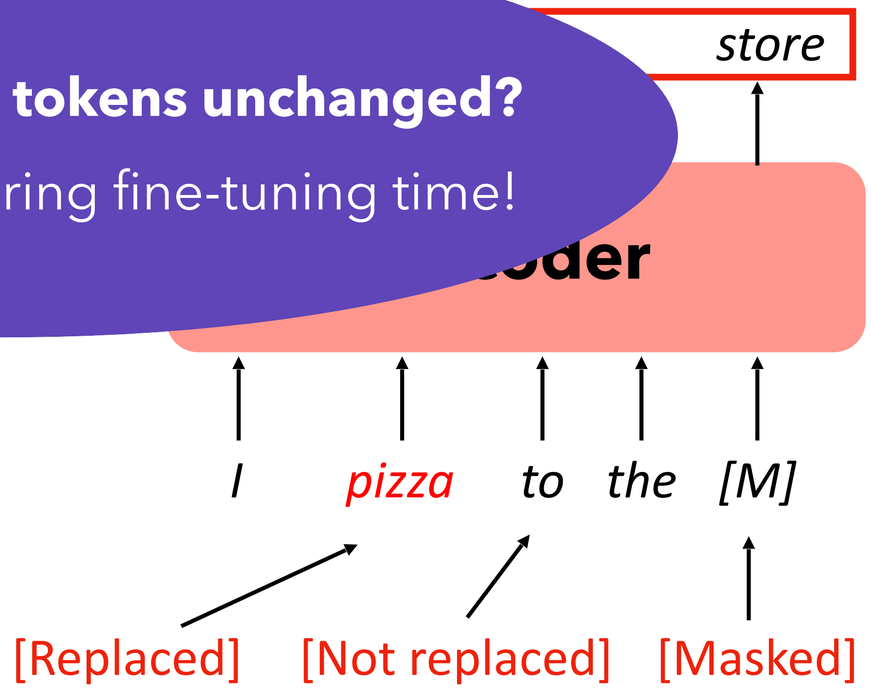
Bidirectional **E**ncoder **R**epresentations from **T**ransformers

[Devlin et al., 2018]

- **2 Pre-training Objectives:**
  - **Masked LM: Choose a random 15% of toke predict.**
    - For each chosen token:
      - Replace it with **[MASK]**
      - Replace it with a **random tok**
      - Leave it **unchanged** 10% of the time (but still predict it!).
  - **Next Sentence Prediction (NSP)**
    - 50% of the time two adjacent sentences are in the correct order.
    - **This actually hurts model learning based on later work!**

WHY keeping some tokens unchanged?

There's no [MASK] during fine-tuning time!



Encoder: BERT

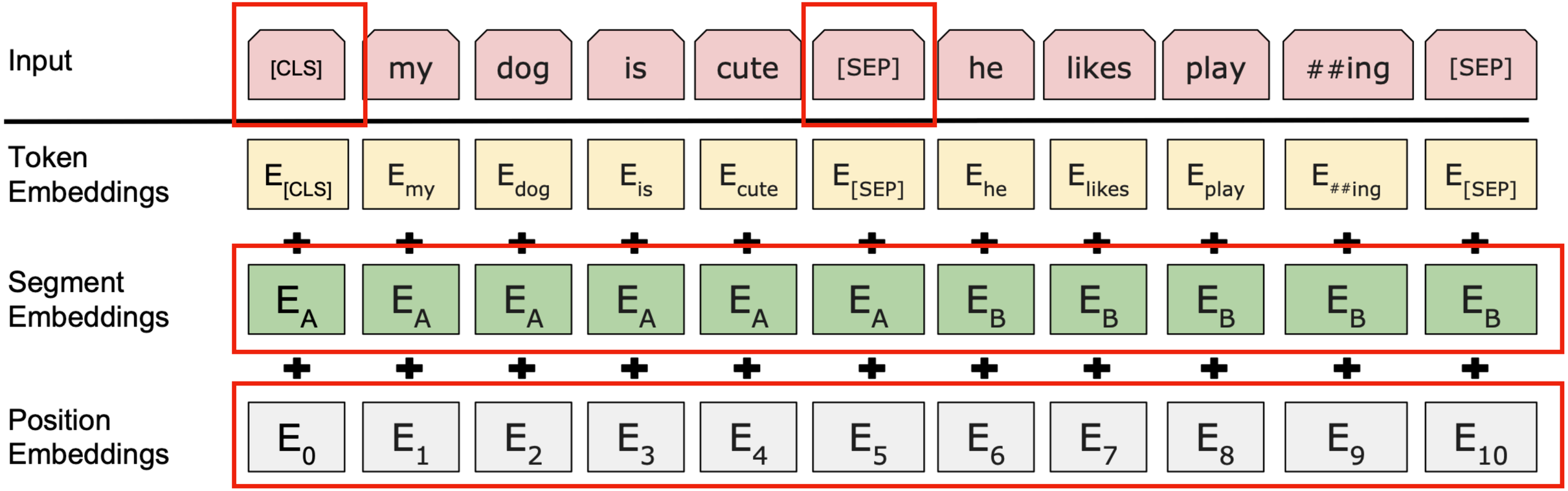
Bidirectional **E**ncoder **R**epresentations from **T**ransformers

[Devlin et al., 2018]

Special token added to the beginning of each input sequence

Special token to separate sentence A/B

Final embedding is the sum of all three!



Learned embedding to every token indicating whether it belongs to sentence A or sentence B

Position of the token in the entire sequence

Encoder: BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

[Devlin et al., 2018]

- **SOTA at the time on a wide range of tasks after fine-tuning!**

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

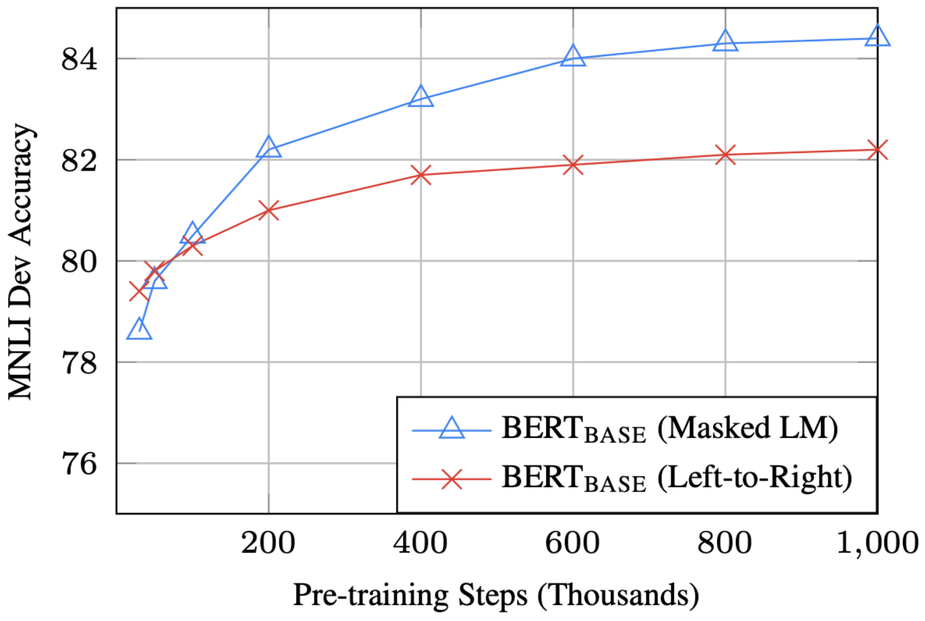
- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI:** natural language inference over question answering data
- **SST-2:** sentiment analysis
- **CoLA:** corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B:** semantic textual similarity
- **MRPC:** microsoft paraphrase corpus
- **RTE:** a small natural language inference corpus

Encoder: BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers [Devlin et al., 2018]

SWAG

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0



- **Two Sizes of Models**
  - **Base:** 110M, 4 Cloud TPUs, 4 days
  - **Large:** 340M, 16 Cloud TPUs, 4 days
  - Both models can be fine-tuned with single GPU
  - The larger the better!

- MLM converges slower than Left-to-Right at the beginning, but out-performers it eventually

Encoder: RoBERTa

[Liu et al., 2019]

- **Original BERT is significantly undertrained!**
- More data (16G => 160G)
- Pre-train for longer
- Bigger batches
- Removing the next sentence prediction (NSP) objective
- Training on longer sequences
- Dynamic masking, randomly masking out different tokens
- A larger byte-level BPE vocabulary containing 50K sub-word units

All around better than BERT!

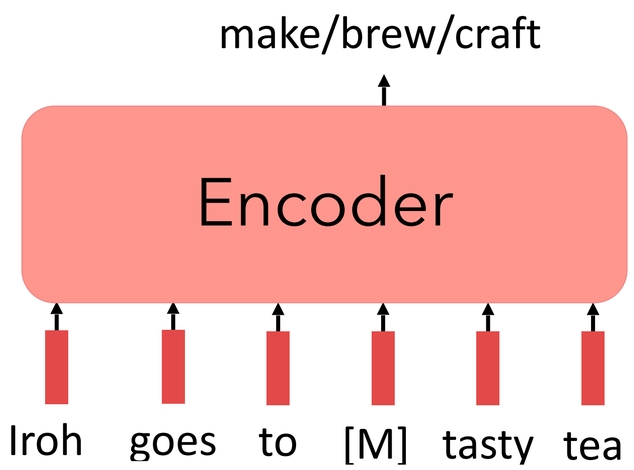
Encoder: Other Variations of BERT

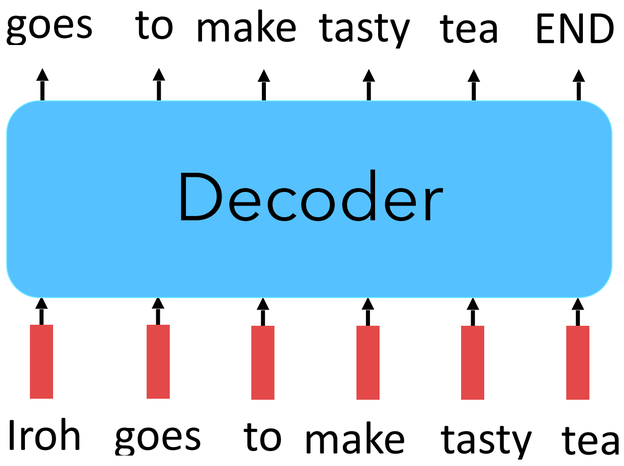
- **ALBERT [Lan et al., 2020]**: incorporates two parameter reduction techniques that lift the major obstacles in scaling pre-trained models
- **DeBERTa [He et al., 2021]:** decoding-enhanced BERT with disentangled attention
- **SpanBERT [Joshi et al., 2019]:** masking contiguous spans of words makes a harder, more useful pre-training task
- **ELECTRA [Clark et al., 2020]:** corrupts texts by replacing some tokens with plausible alternatives sampled from a small generator network, then train a discriminative model that predicts whether each token in the corrupted input was replaced by a generator sample or not.
- **DistilBERT [Sanh et al., 2019]:** distilled version of BERT that's 40% smaller
- **TinyBERT [Jiao et al., 2019]:** distill BERT for both pre-training & fine-tuning
- **...**

Encoder: Pros & Cons

- Consider both left and right context
- Capture intricate contextual relationships

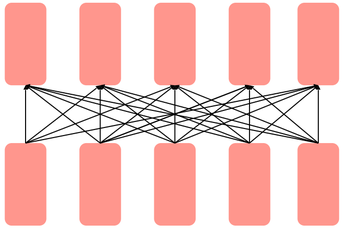
- Not good at generating open-text from left-to-right, one token at a time





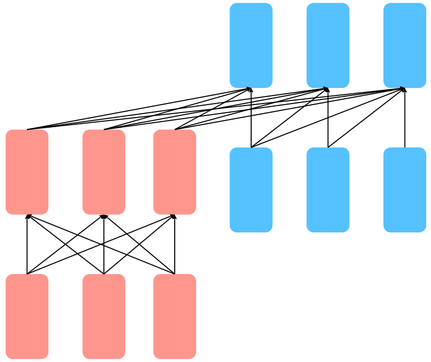
3 Pre-training Paradigms/Architectures

Encoder



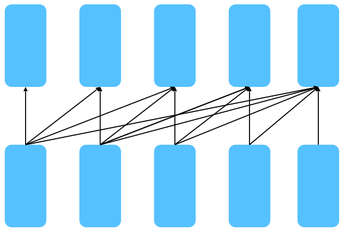
- Bidirectional; can condition on the future context

Encoder-Decoder



- Map two sequences of different length together

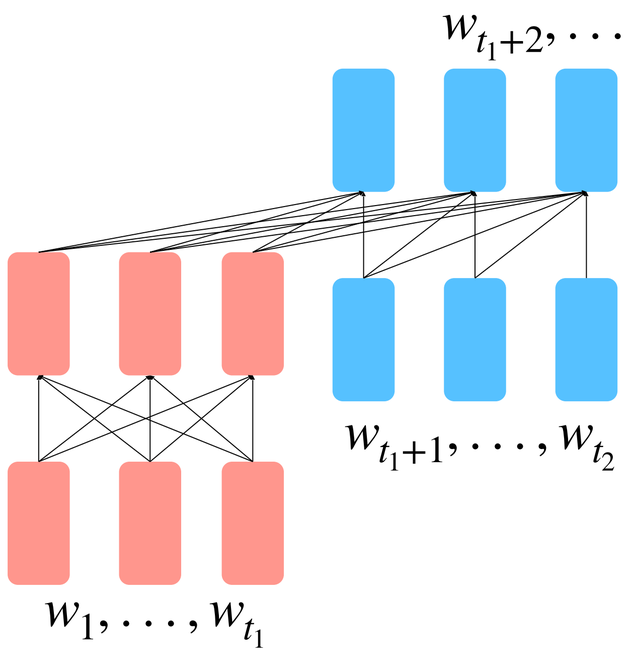
Decoder



- Language modeling; can only condition on the past context

Encoder-Decoder: Architecture

- Moving towards **open-text generation**…
- **Encoder** builds a representation of the source and gives it to the **decoder**
- **Decoder** uses the source representation to generate the target sentence
- The **encoder** portion benefits from **bidirectional** context; the **decoder** portion is used to train the whole model through **language modeling**



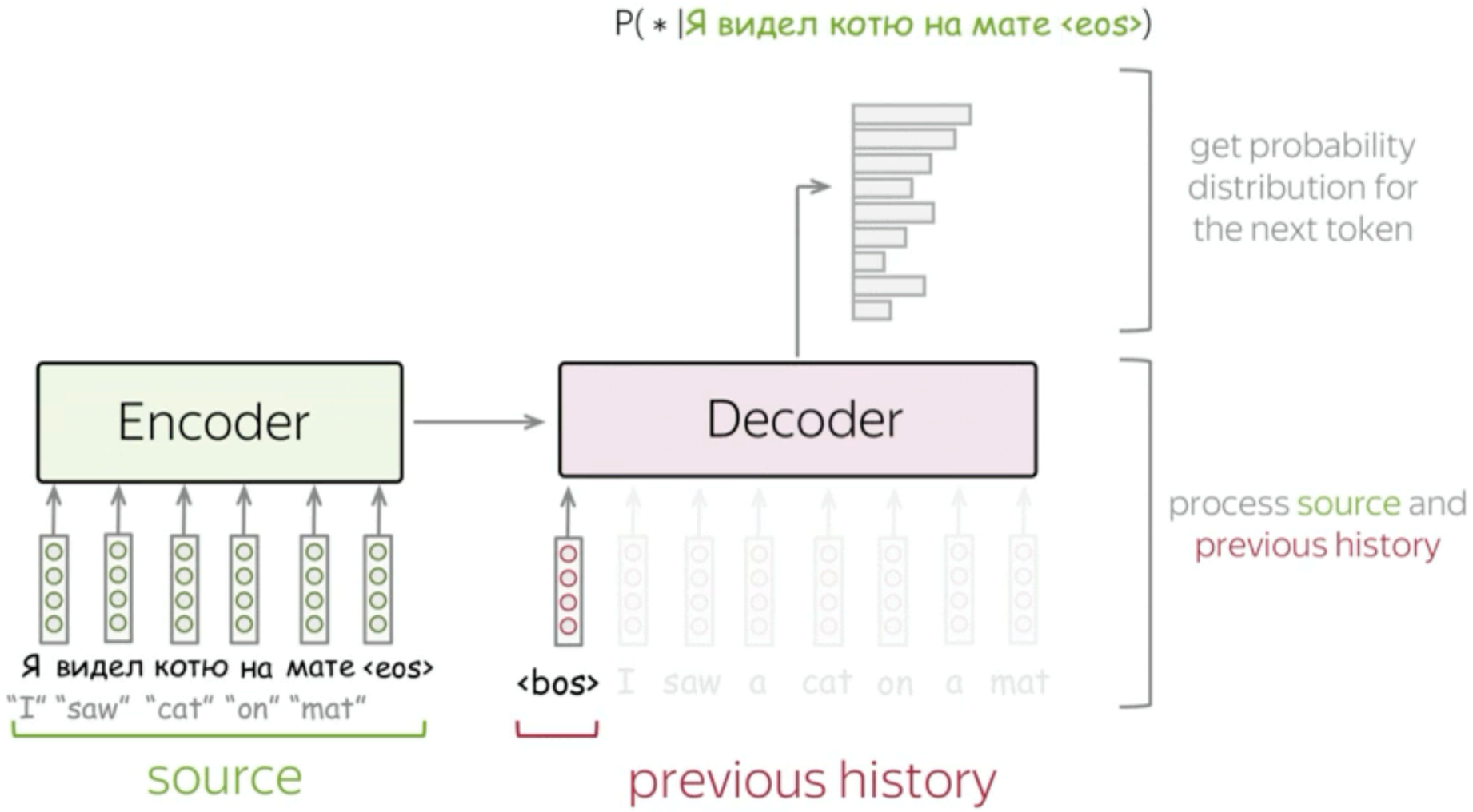
$$h_1, \ldots, h_{t_1} = \text{Encoder}(w_1, \ldots, w_{t_1})$$

$$h_{t_1+1}, \ldots, h_{t_2} = \text{Decoder}(w_{t_1+1}, \ldots, w_{t_2}, h_1, \ldots, h_{t_1})$$

$$y_i \sim Ah_i + b, i > t$$

[Raffel et al., 2018]

Encoder-Decoder: An Machine Translation Example



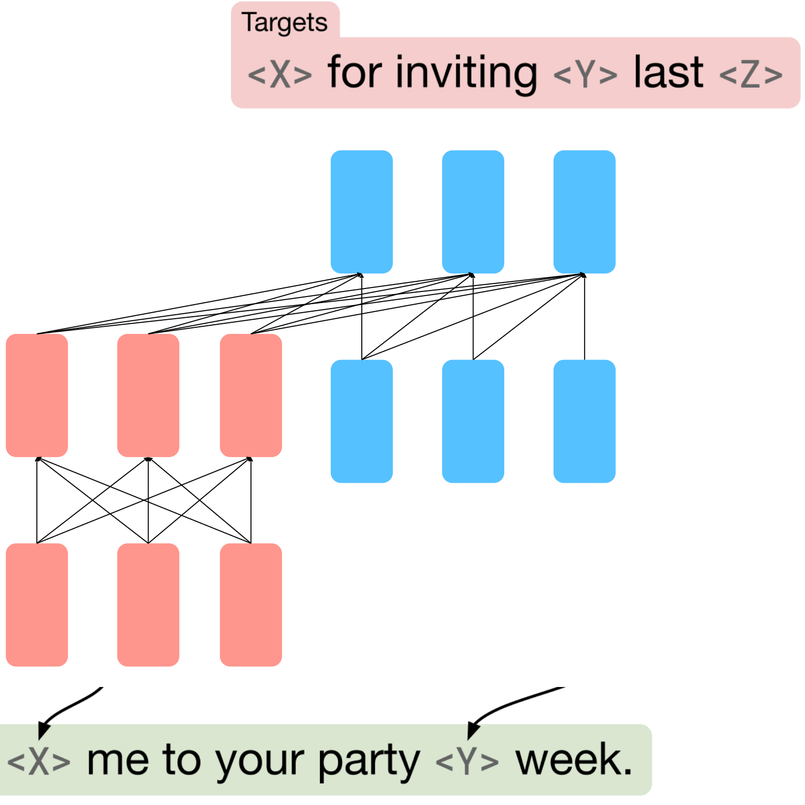
[Lena Viota Blog]

Encoder-Decoder: Training Objective

- **T5 [Raffel et al., 2018]**
- **Text span corruption (denoising):** Replace different-length spans from the input with unique placeholders (e.g., <extra_id_0>); decode out the masked spans.
  - Done during **text preprocessing**: training uses **language modeling** objective at the decoder side

Original text

Thank you ~~for inviting~~ me to your party ~~last~~ week.



Encoder-Decoder: T5

[Raffel et al., 2018]

- **Encoder-decoders** works better than decoders
- **Span corruption (denoising)** objective works better than language modeling

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Encoder-Decoder: T5

[Raffel et al., 2018]

- **Text-to-Text:** convert NLP tasks into input/output text sequences
- **Dataset:** Colossal Clean Crawled Corpus (C4), 750G text data!
- **Various Sized Models:**
  - Base (222M)
  - Small (60M)
  - Large (770M)
  - 3B
  - 11B
- **Achieved SOTA with scaling & purity of data**

[Google Blog]

T5

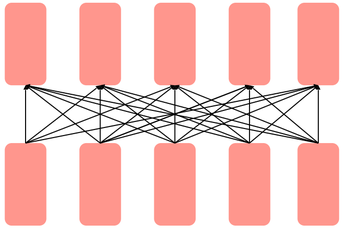
Encoder-Decoder: Pros & Cons

- A nice middle ground between leveraging **bidirectional** contexts and **open-text** generation
- Good for **multi-task** fine-tuning

- Require more **text wrangling**
- **Harder to train**
- **Less flexible** for natural language generation

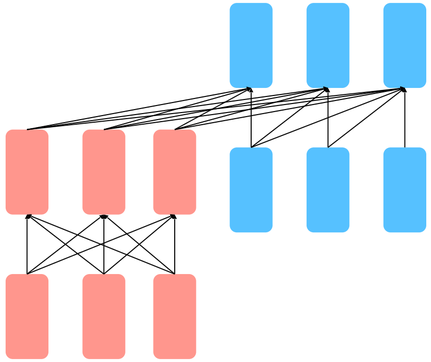
3 Pre-training Paradigms/Architectures

Encoder



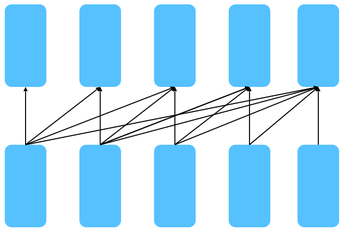
- Bidirectional; can condition on the future context

Encoder-Decoder



- Map two sequences of different length together

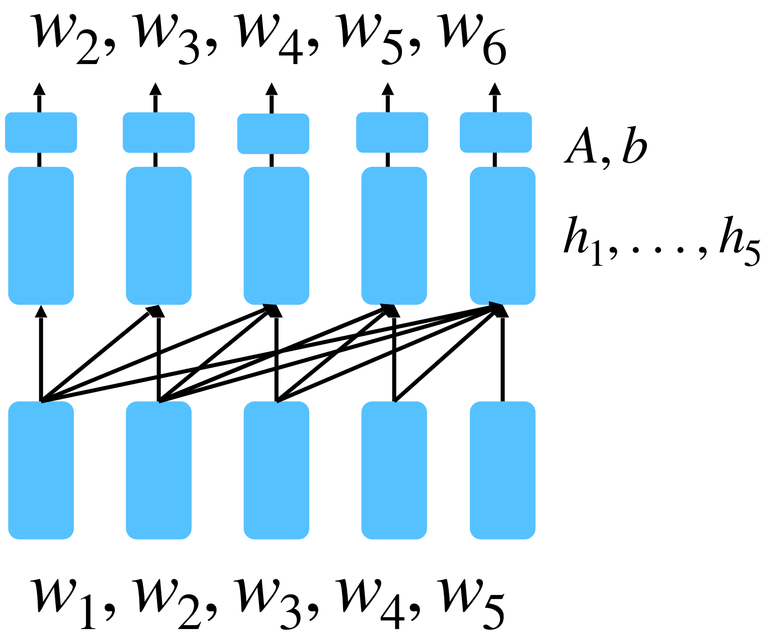
Decoder



- Language modeling; can only condition on the past context

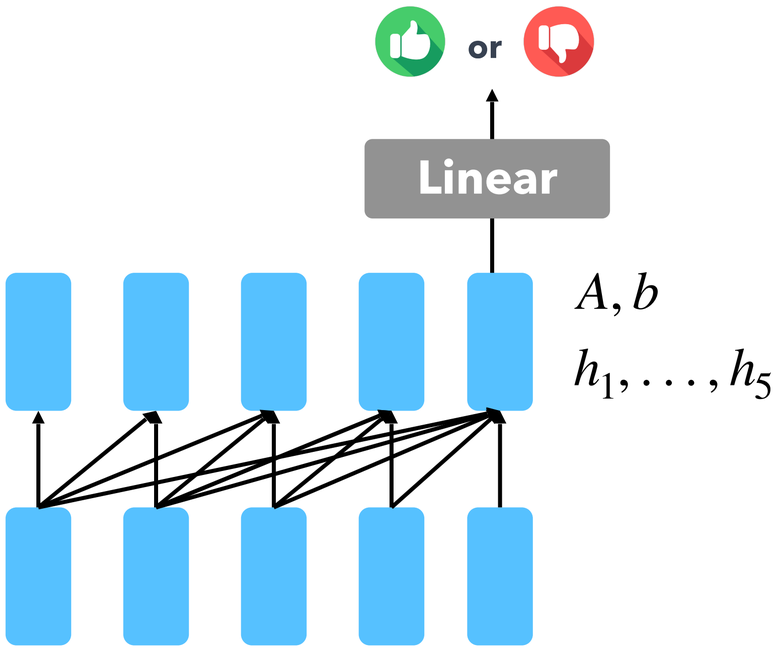
Decoder: Training Objective

- Many most famous generative LLMs are **decoder-only**
  - e.g., GPT1/2/3/4, Llama1/2
- **Language modeling!** Natural to be used for **open-text generation**
- **Conditional LM:** $p(w_t \mid w_1, \ldots, w_{t-1}, x)$
  - Conditioned on a source context x to generate from left-to-right
- Can be fine-tuned for **natural language generation (NLG)** tasks, e.g., dialogue, summarization.



Decoder: Training Objective

- Customizing the pre-trained model for downstream tasks:
  - Add a **linear layer** on top of the last hidden layer to make it a classifier!
  - During fine-tuning, trained the randomly **initialized linear layer**, along with **all parameters** in the neural net.

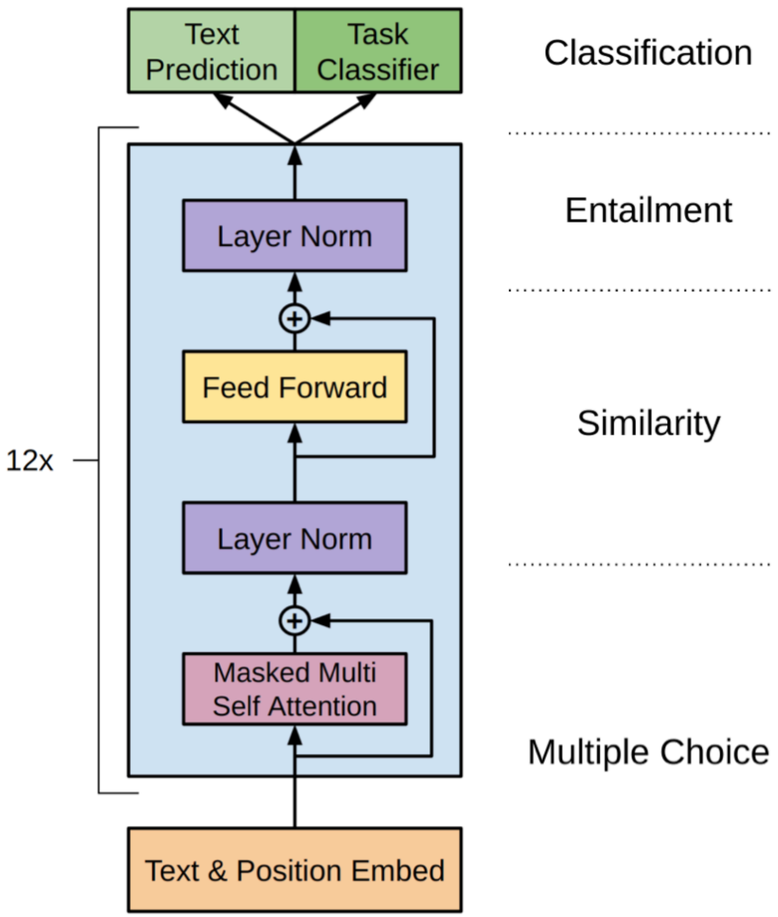


Is Santa Claus real figure?

Decoder: GPT

Generative **P**re-trained **T**ransformer

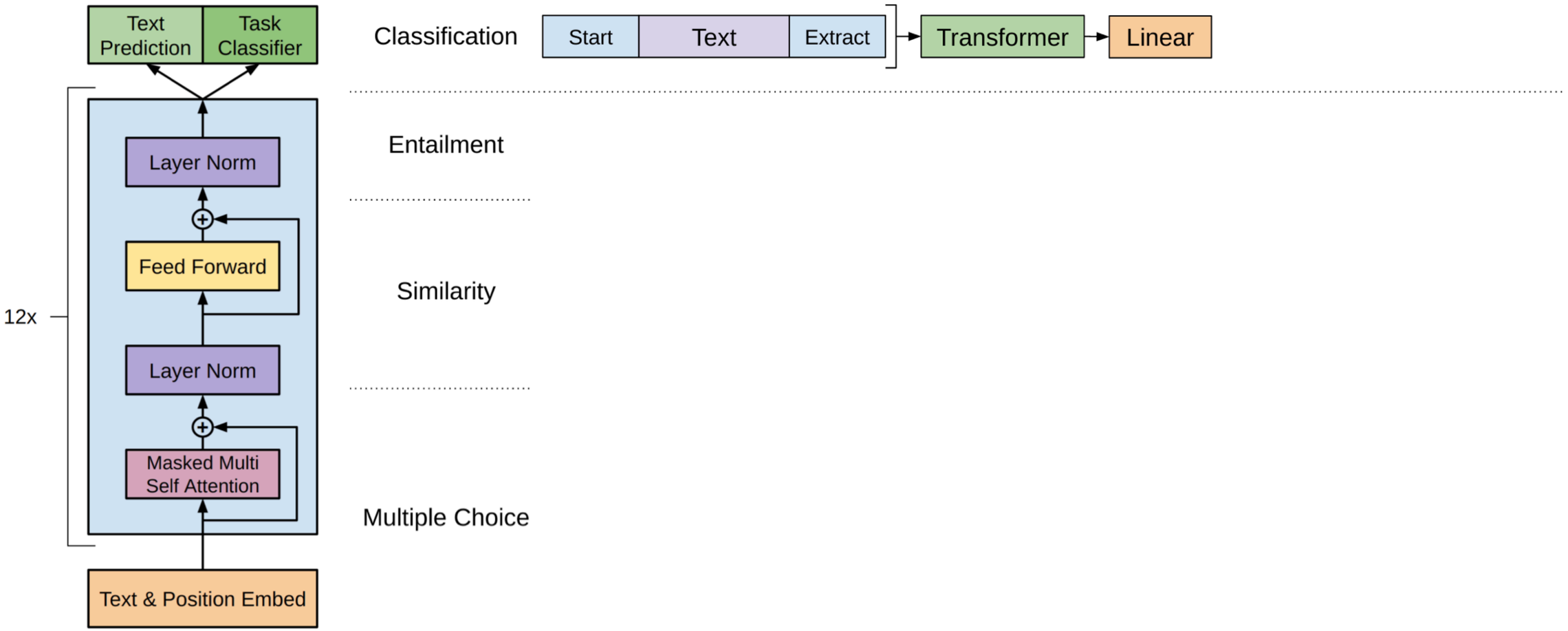
[Radford et al., 2018]



Decoder: GPT

Generative **P**re-trained **T**ransformer

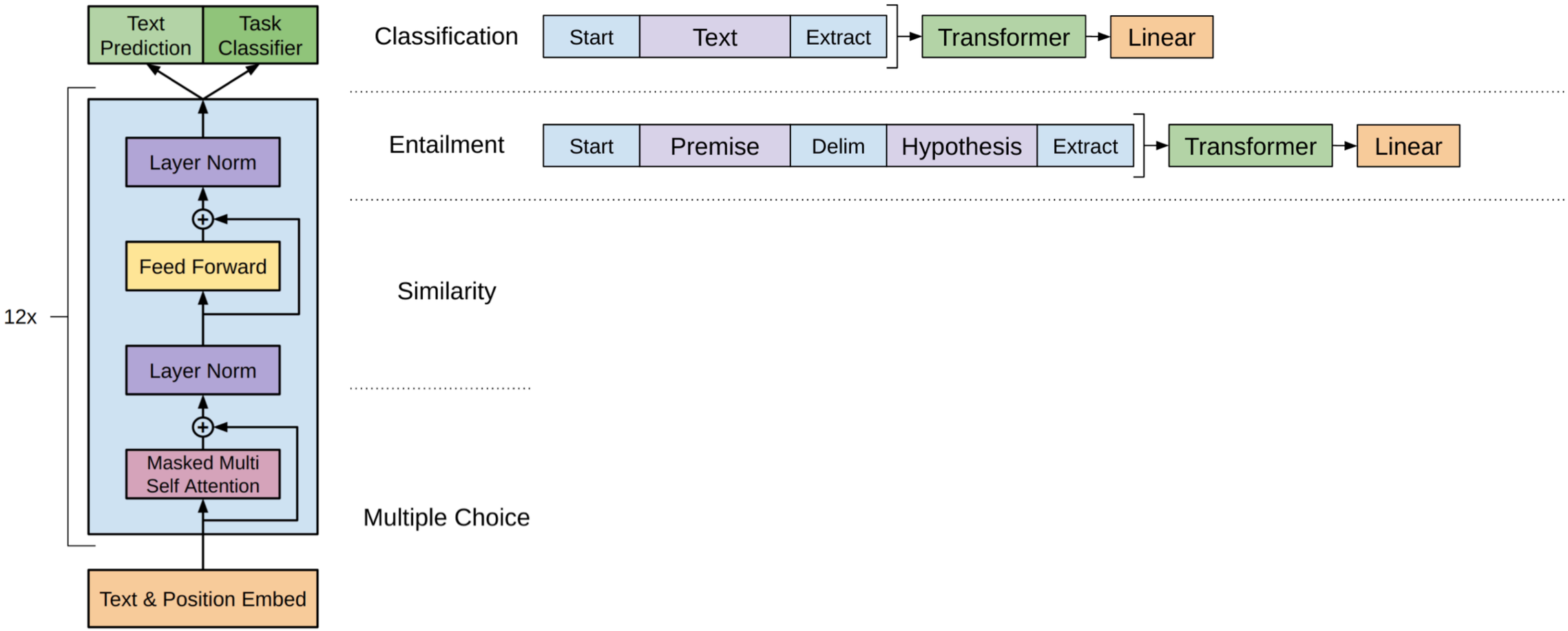
[Radford et al., 2018]



Decoder: GPT

Generative **P**re-trained **T**ransformer

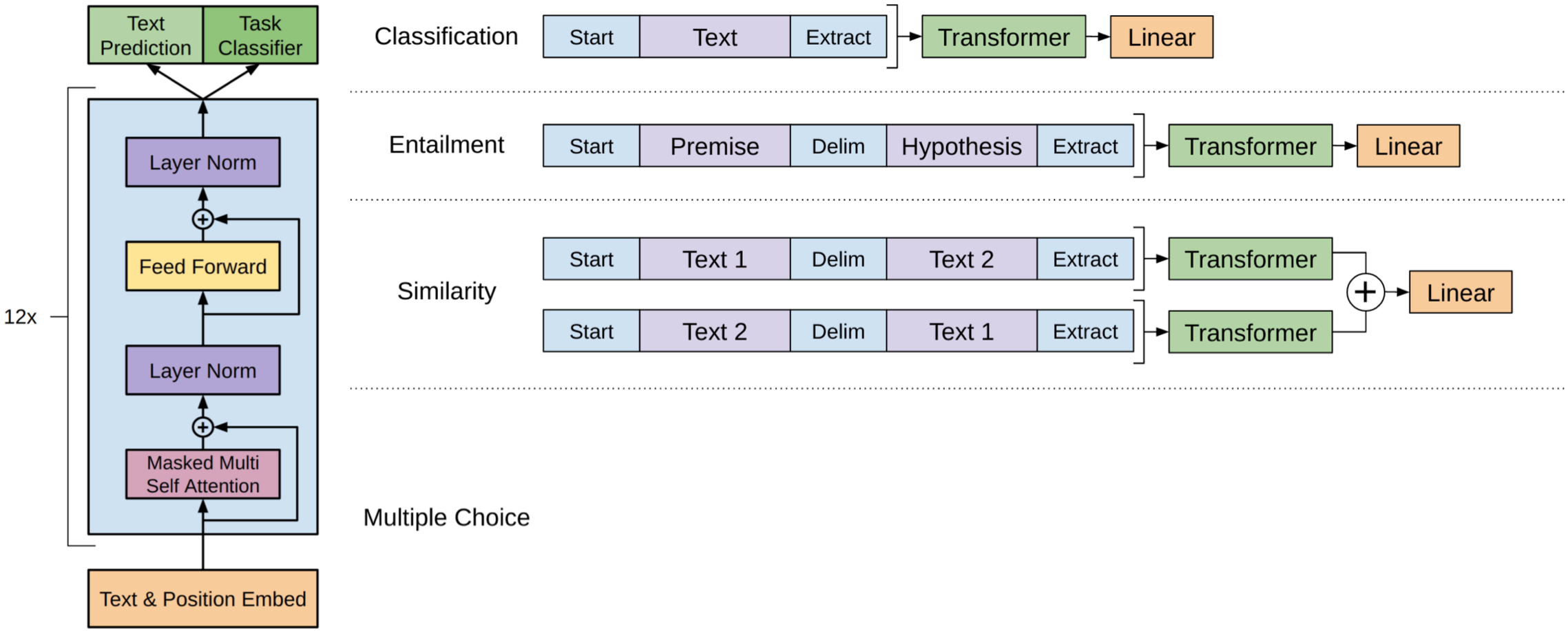
[Radford et al., 2018]



Decoder: GPT

Generative **P**re-trained **T**ransformer

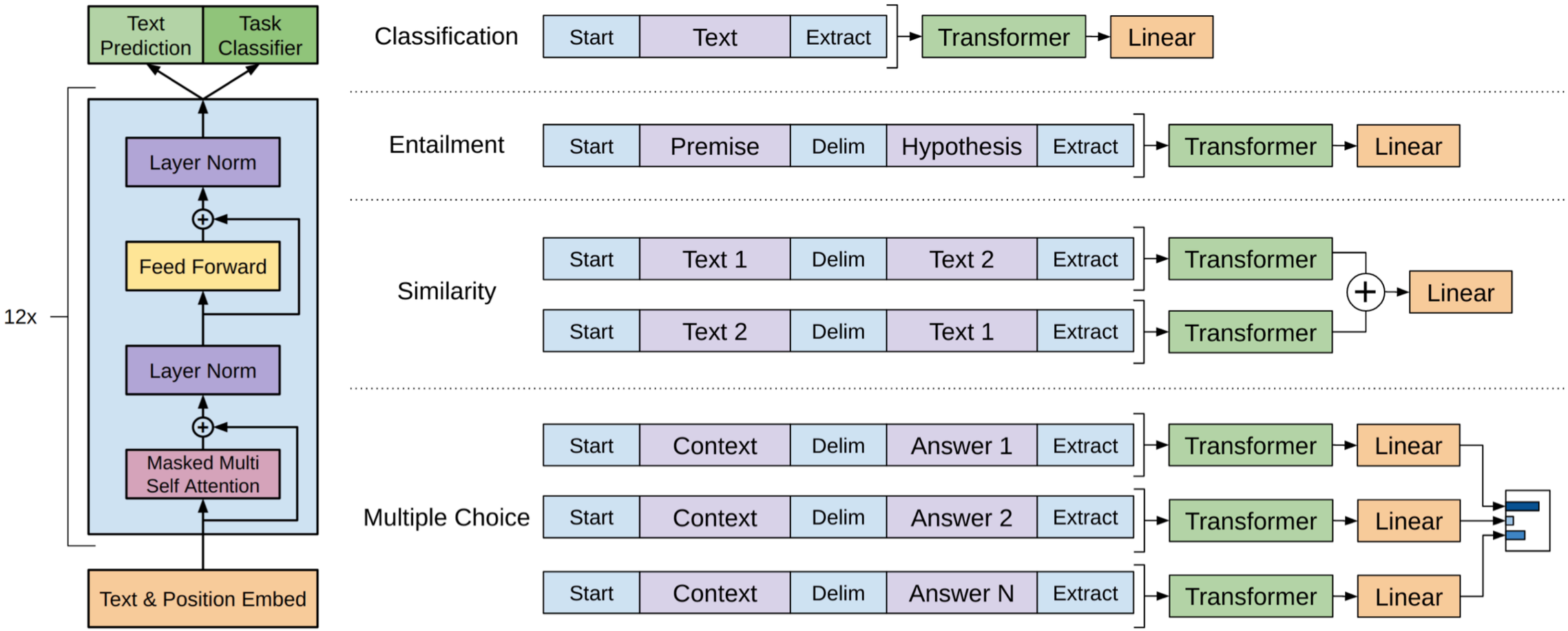
[Radford et al., 2018]



Decoder: GPT

Generative **P**re-trained **T**ransformer

[Radford et al., 2018]



How to pick a proper architecture for a given task?

- Right now **decoder-only** models seem to dominant the field at the moment
  - e.g., GPT1/2/3/4, Mistral, Llama1/2
- T5 (seq2seq) works well with multi-tasking
- **Picking the best model architecture remains an open research question!**

Thank you!