Transformers and Pre-trained Language Models

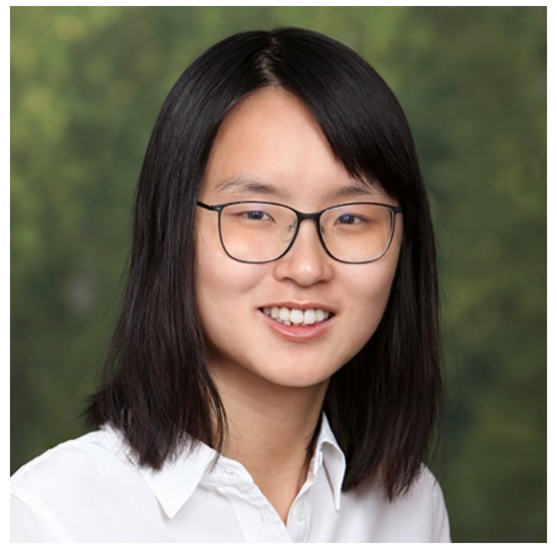
Danqi Chen

Princeton Language and Intelligence
Princeton University

July 15th, 2024

(Many slides are adapted from Princeton COS484 and Stanford CS224N course materials)

Introduction



Adithya Bhaskar

Howard Chen (w/ Karthik Narasimhan)

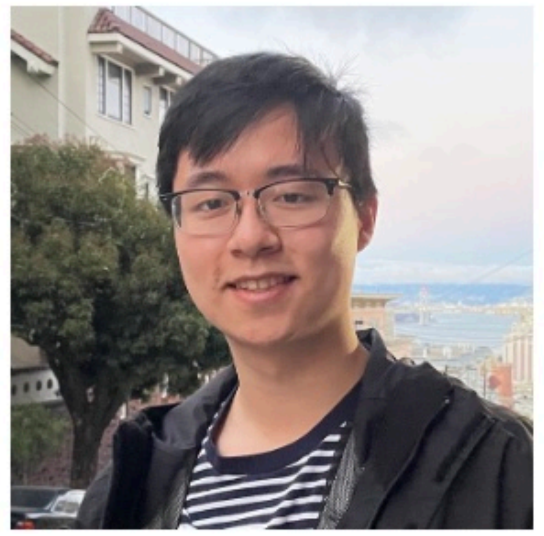
Dan Friedman

Tianyu Gao

Lucy He (w/ Peter Henderson)

Alexander Wettig

Mengzhou Xia



Howard Yen

Zexuan Zhong

Our research focuses on ***training***, ***adapting*** and ***understanding*** large language models

See more at https://www.cs.princeton.edu/~danqic/

Lecture plan

Part I. **Transformers**

Focus: innovations and key designs in neural **architectures**

30min coffee break

Part II. **Pre-trained language models**

Focus: training **objectives** & **data**, downstream **adaptations**

Lecture plan

- **Fundamentals (70%)** - I will walk through the most important ideas in NLP and LLMs in the past 5+ years (Transformers, pre-training, in-context learning, RLHF, …)

- **How do these ideas evolve and lead to state-of-the-art models? (15%)** - I will highlight recent improvements and developments

- **Cutting-edge research topics (15%)** - What research topics do we study in 2024? I will briefly discuss some of the works from my research group too

Part I.

Transformers

Transformers

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

(Vaswani et al., 2017)

What is attention?

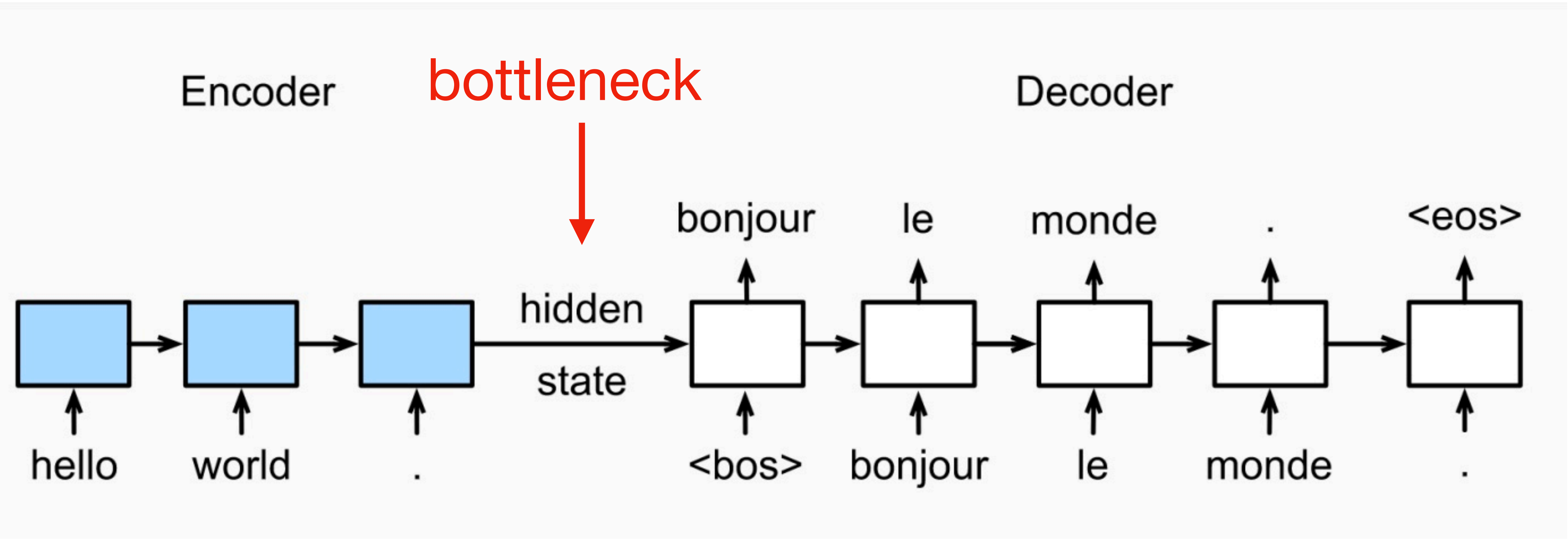
NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

(Bahdanau et al., 2015)

- Attention is a technique to address the “bottleneck” issue in the seq2seq model, originally designed for machine translation



What is attention?

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

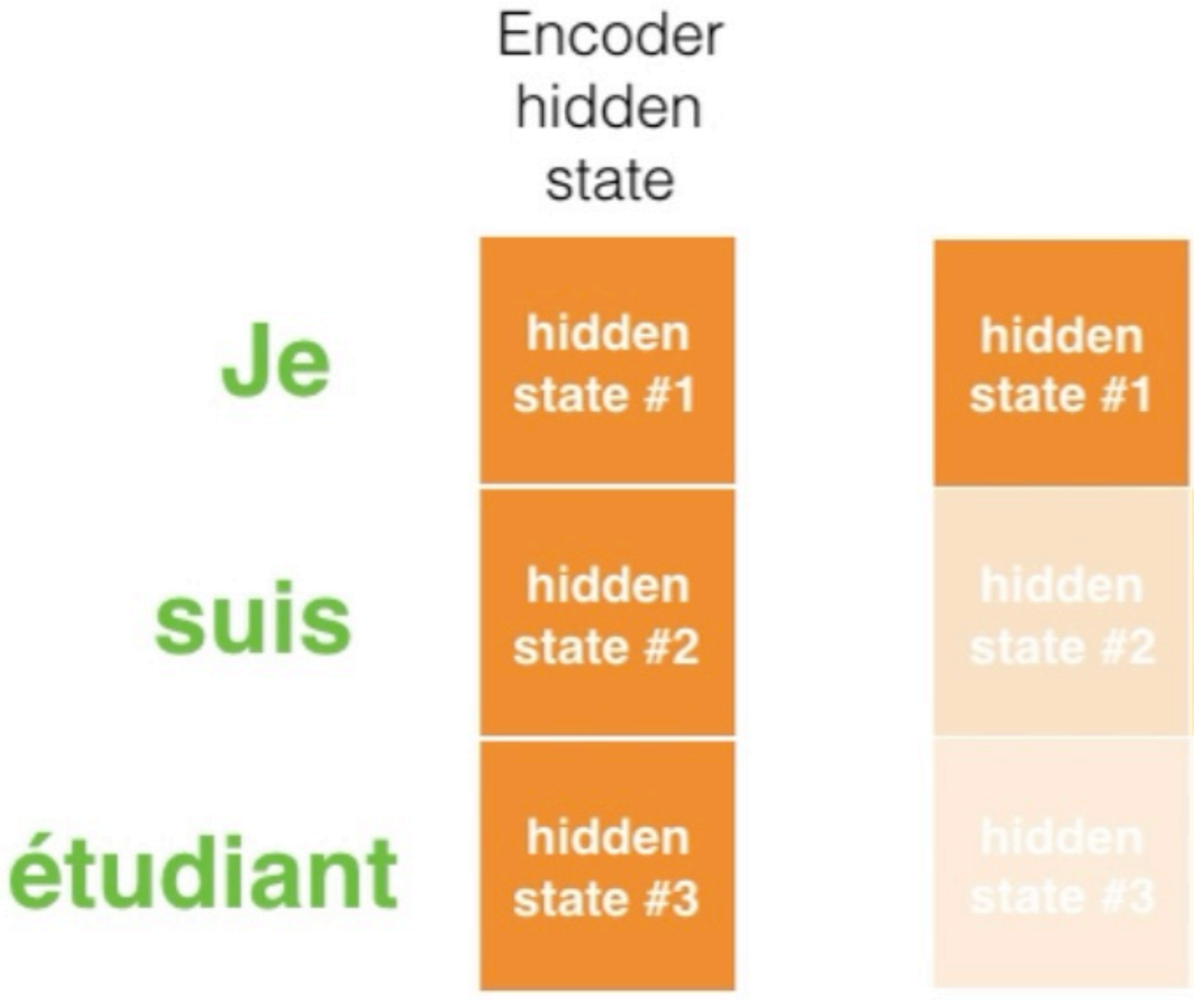
KyungHyun Cho **Yoshua Bengio***
Université de Montréal

(Bahdanau et al., 2015)

- Attention is a technique to address the “bottleneck” issue in the seq2seq model, originally designed for machine translation

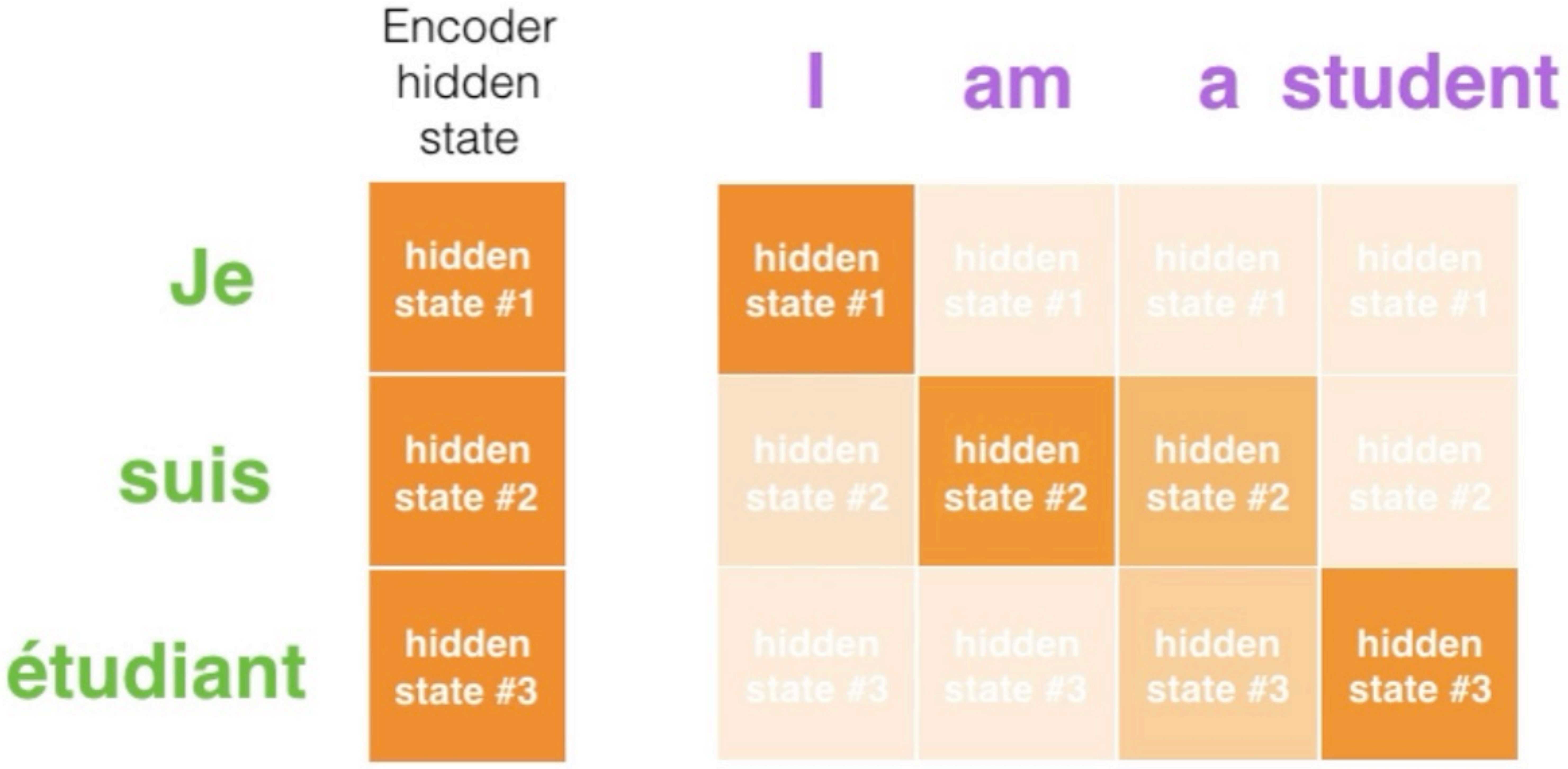
- **Key idea** 💡: At each time step during decoding, focus on only a particular part of source sentence
  - This depends on **decoder’s** current hidden state h_t^{dec}
  - Usually implemented as a probability distribution over the hidden states of the encoder (h_i^{enc})

Attention for seq2seq models



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention for seq2seq models



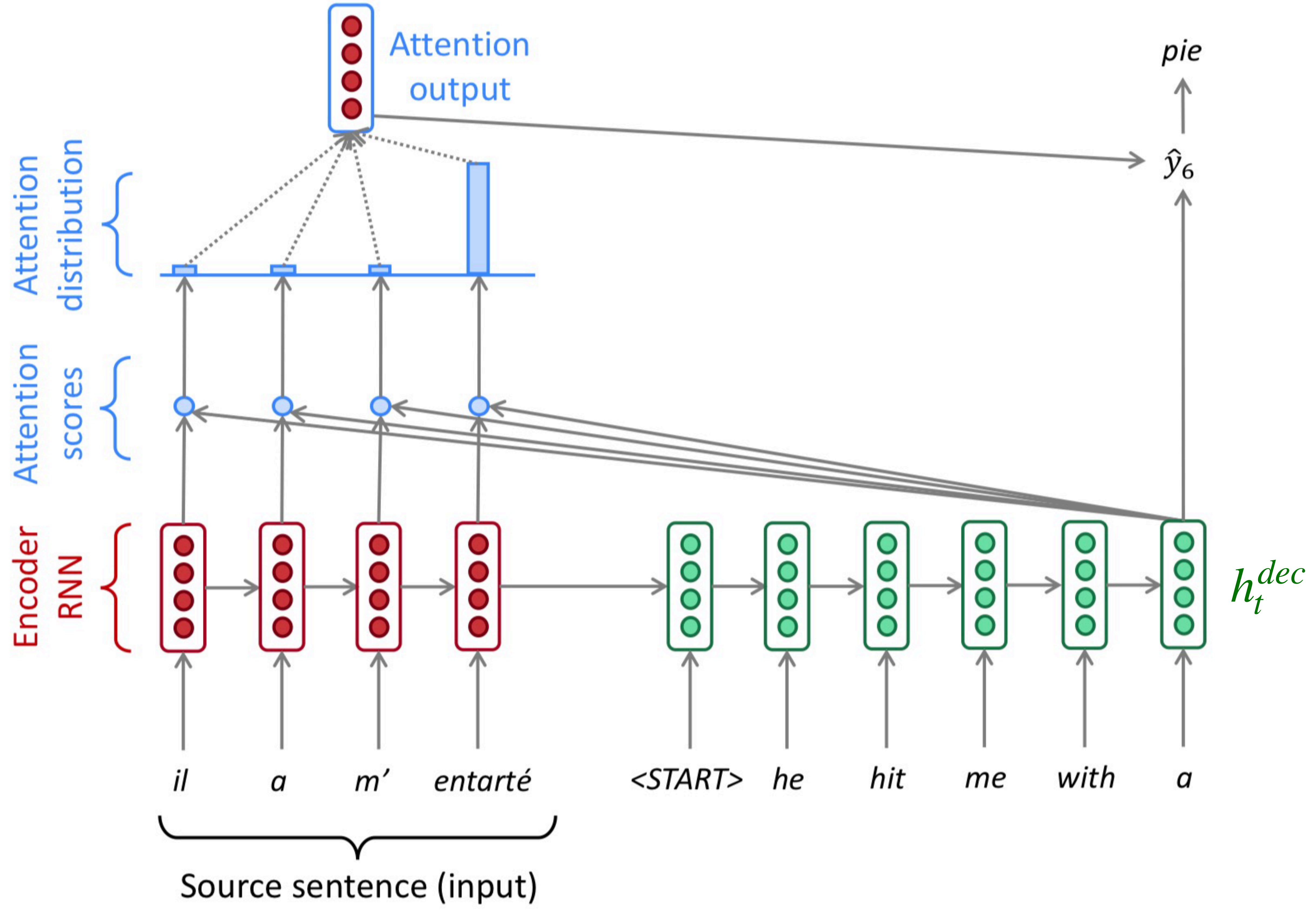
Attention learns the notion of **alignment**

“Which source words are more relevant to the current target word?”

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention for seq2seq models

$h_1^{enc}, \ldots, h_n^{enc}$ and h_t^{dec} are hidden states from encoder and decoder RNNs



- Encoder hidden states: $h_1^{enc}, \ldots, h_n^{enc}$

  (n: # of words in source sentence)

- Decoder hidden state at time t: h_t^{dec}

- Attention scores:

  $$e^t = [g(h_1^{enc}, h_t^{dec}), \ldots, g(h_n^{enc}, h_t^{dec})] \in \mathbb{R}^n$$

- Attention distribution:

  $$\alpha^t = \mathrm{softmax}(e^t) \in \mathbb{R}^n$$

- Weighted sum of encoder hidden states:

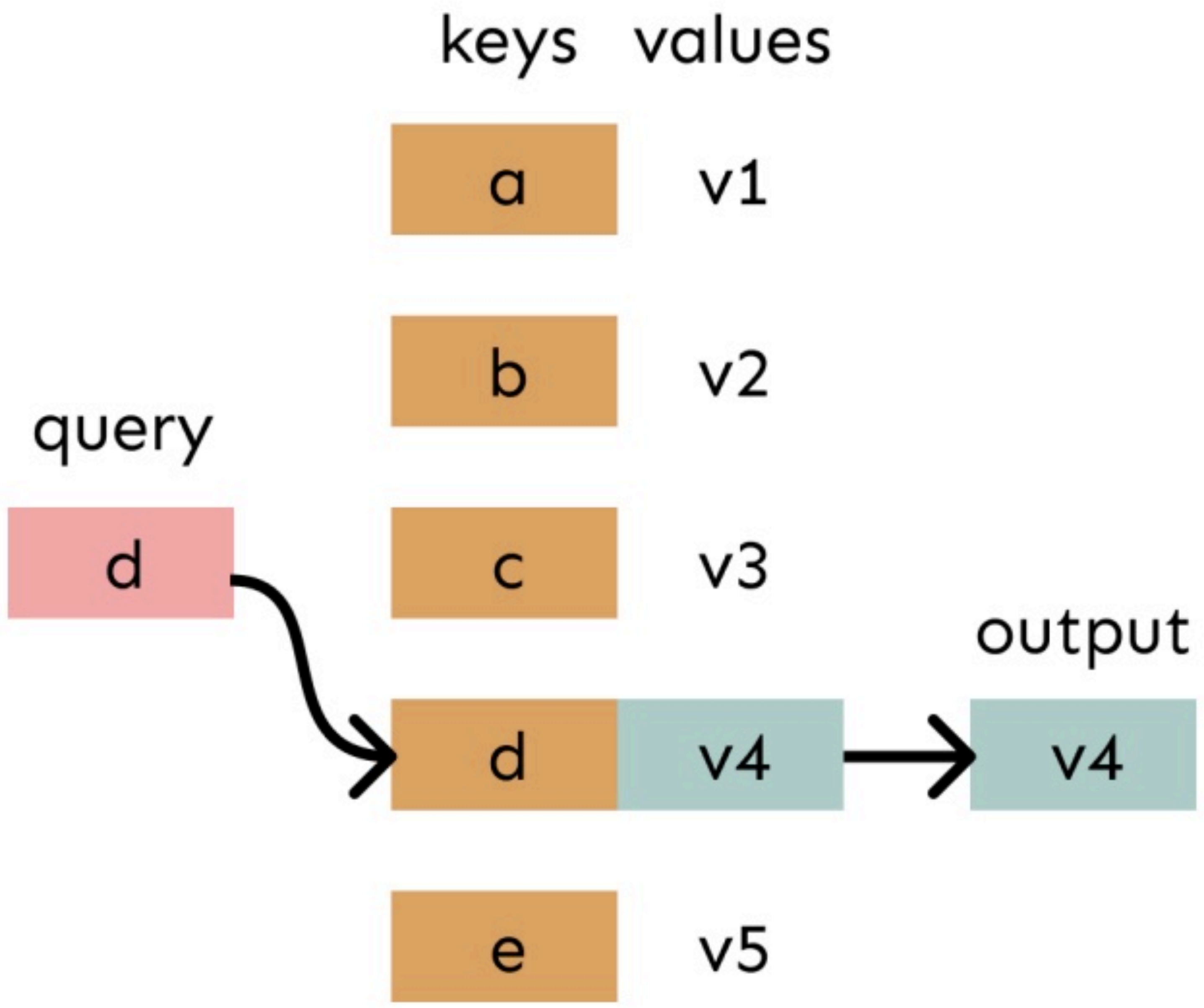
  $$o_t = \sum_{i=1}^{n} \alpha_i^t h_i^{enc} \in \mathbb{R}^h$$

Combine o_t and h_t^{dec} to predict next word

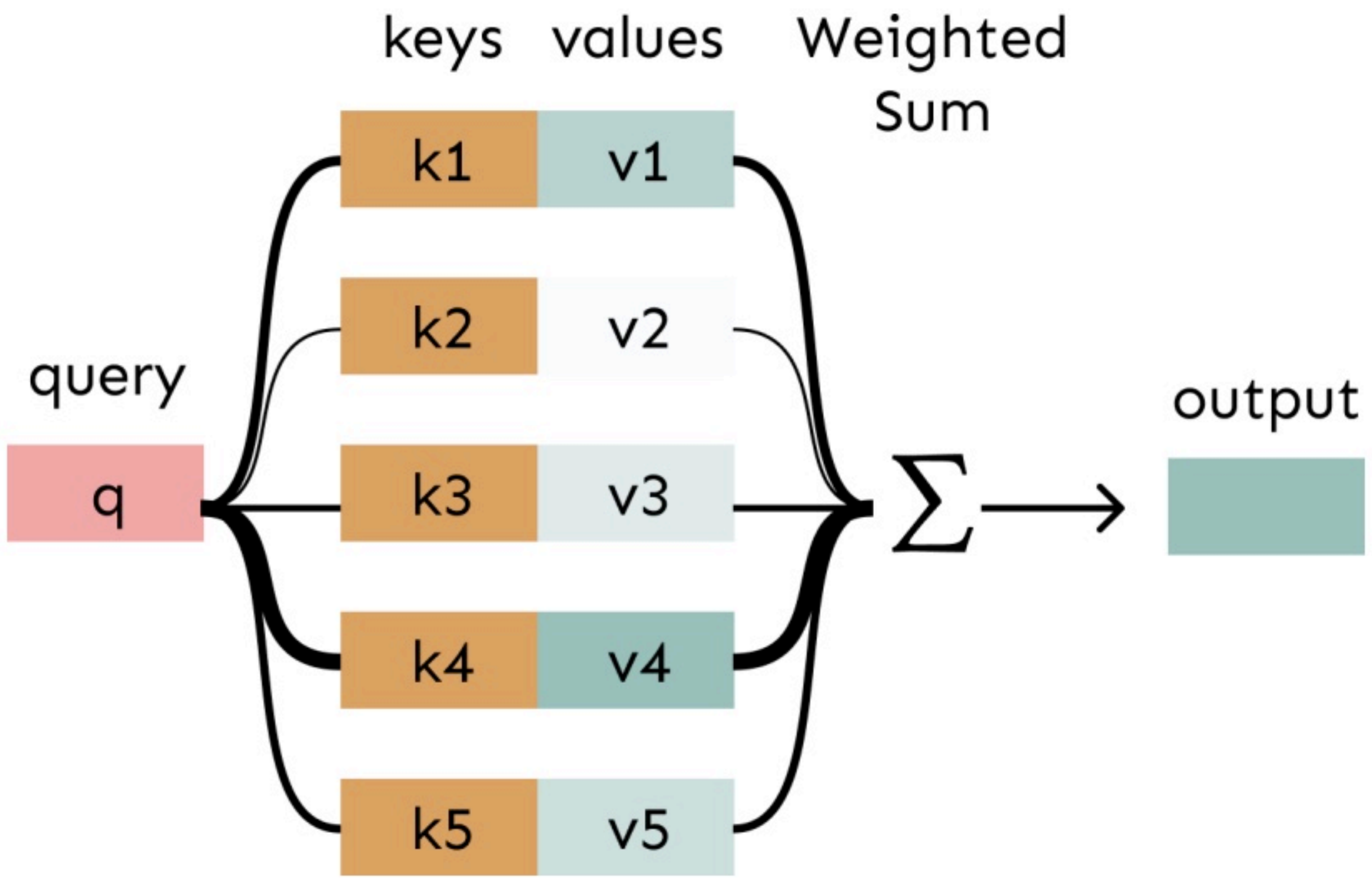
Attention as a soft, averaging lookup table

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

Lookup table: a table of keys that map to values. The query matches one of the keys, returning its value.

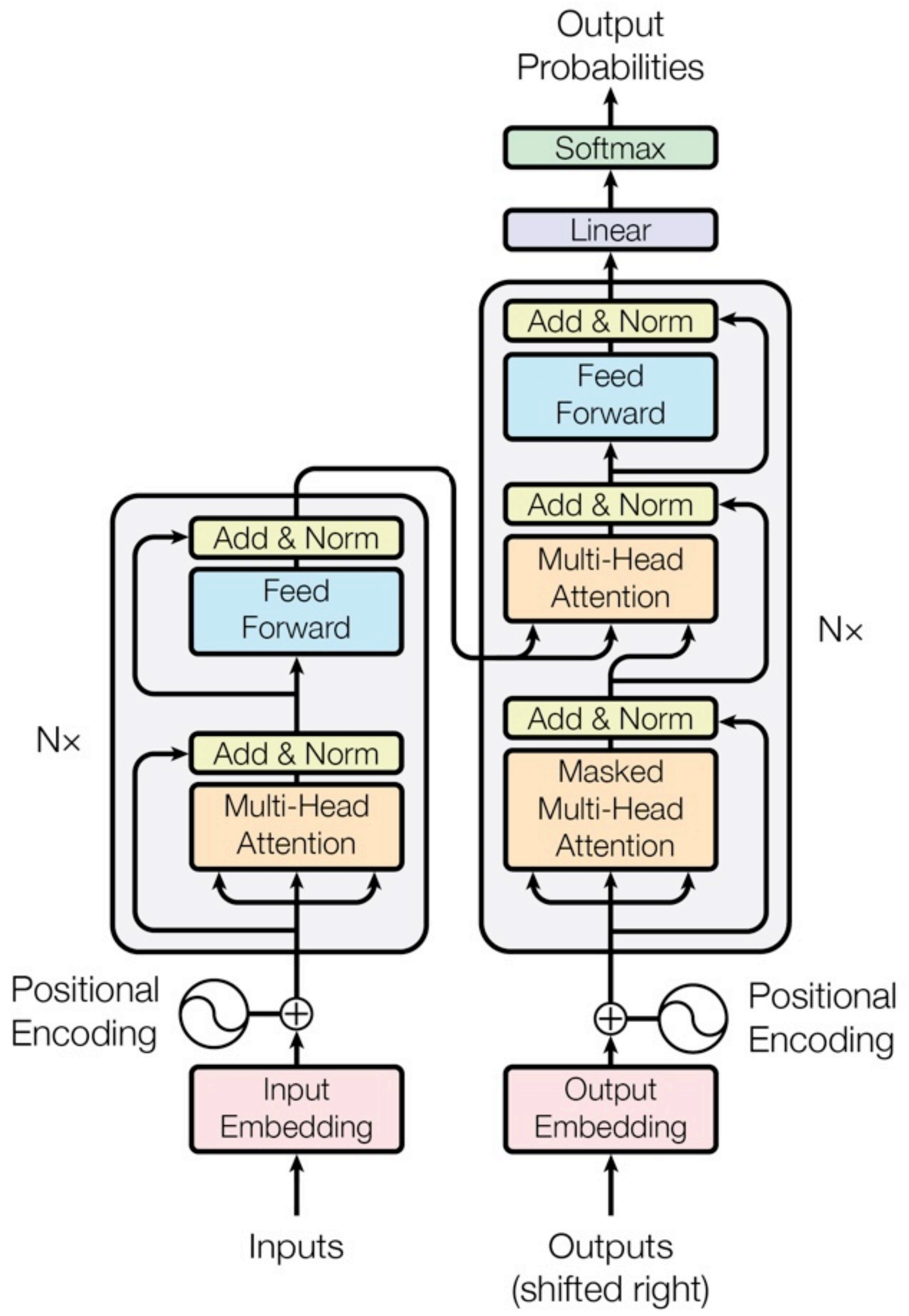


Attention: The query matches to all keys softly to a weight between 0 and 1. The keys' values are multiplied by the weights and summed.



(In the case of NMT, key = value)

Transformer encoder-decoder



(Vaswani et al., 2017)

- Transformer encoder + Transformer decoder: a replacement for seq2seq + attention based on RNNs
- First designed and experimented on NMT

1997 RNNs / LSTMs

↓

2014 seq2seq

↓

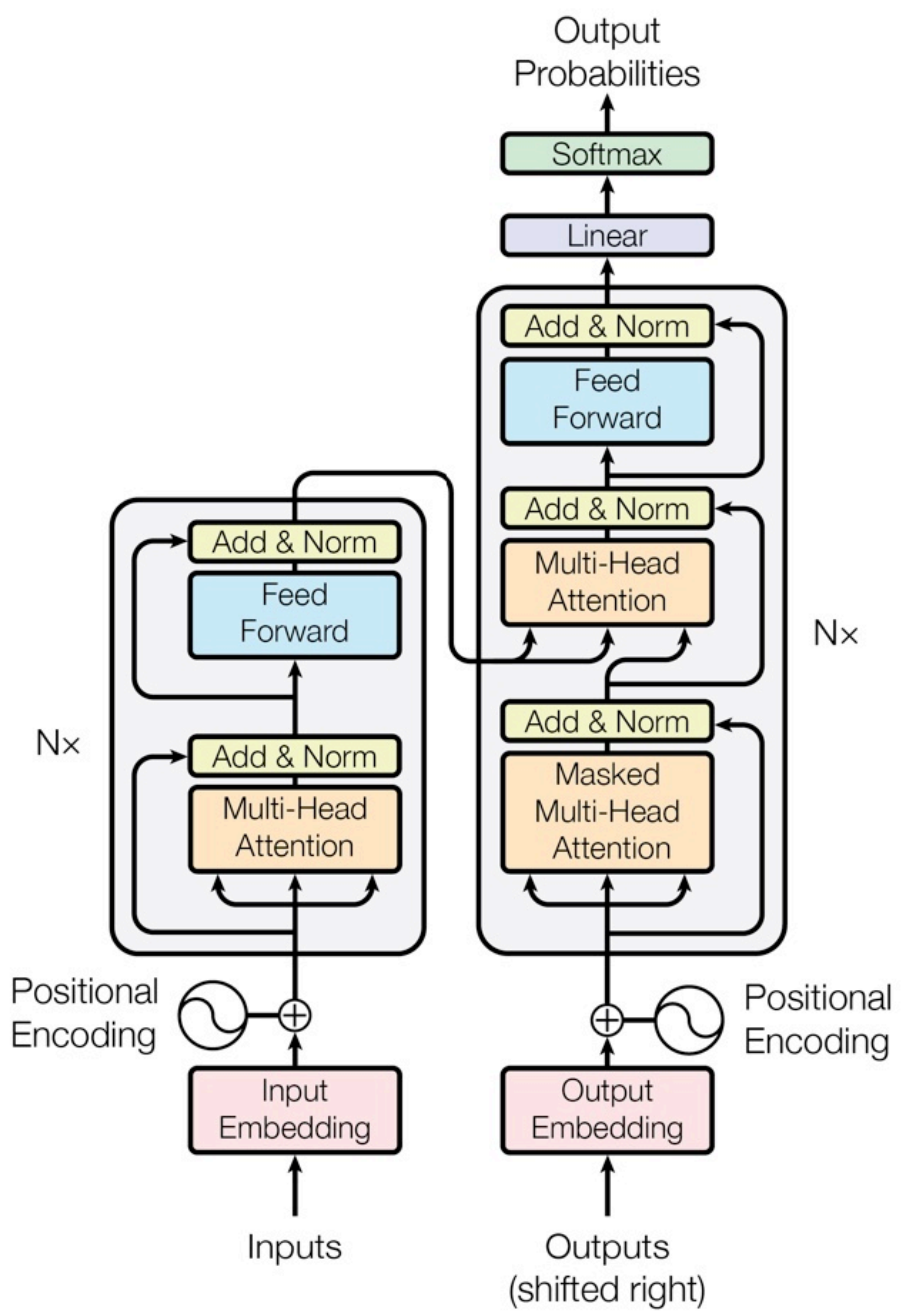
2015 seq2seq + attention

↓

2017 attention only - Transformer encoder-decoder

Transformers (both encoders and decoders) have become the default neural architectures in modeling languages!

Transformer encoder-decoder



(Vaswani et al., 2017)

- Transformer encoder = a stack of **encoder layers**
- Transformer decoder = a stack of **decoder layers**

Transformer encoder: BERT, RoBERTa, ELECTRA

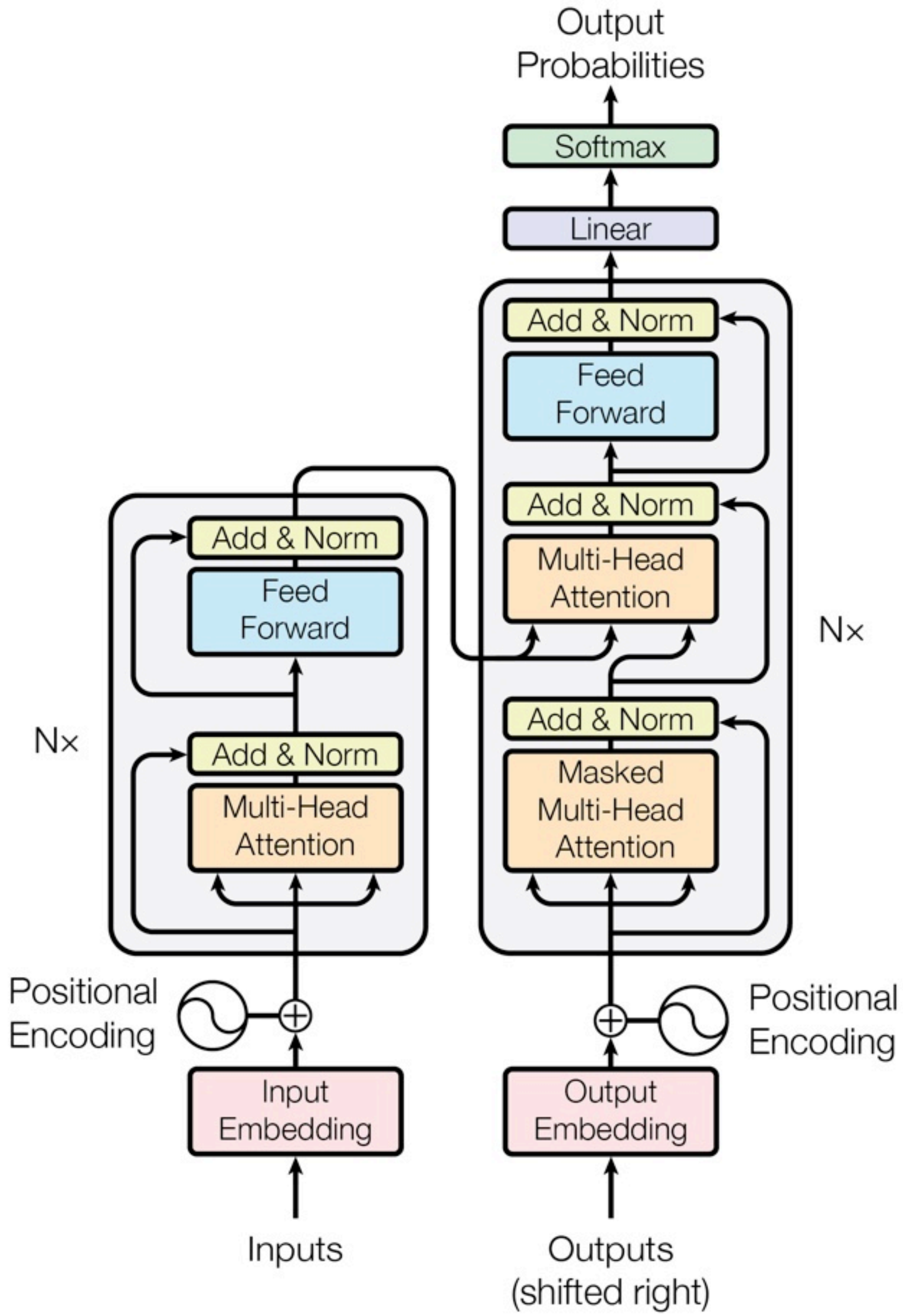
Transformer decoder: GPT-n, ChatGPT, Gemini, Claude, LLaMA, Mistral, …

Transformer encoder-decoder: T5, BART

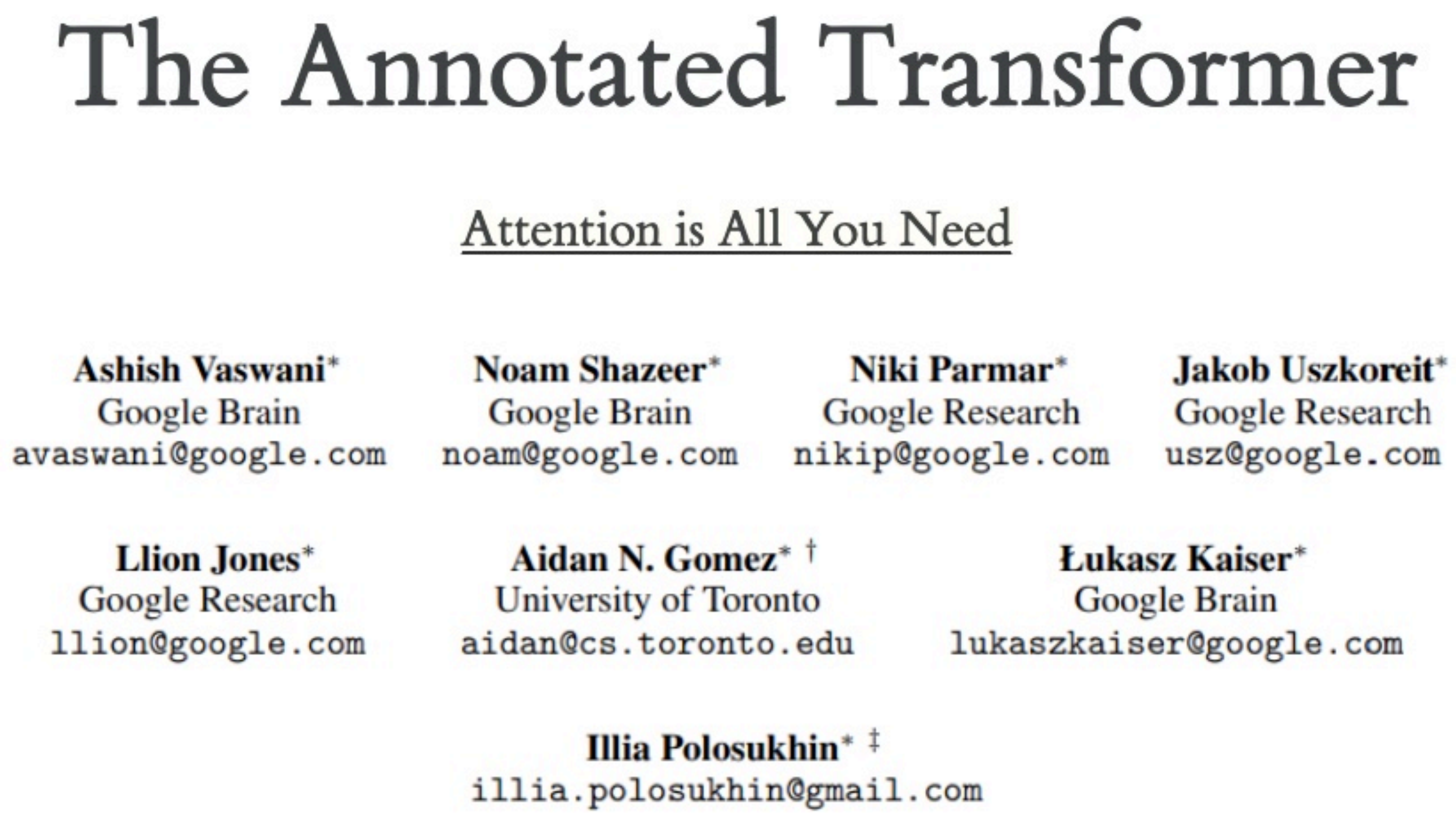
- Key innovation: **self-attention, multi-head**
- Transformers don't have any recurrence structures!

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^h$$

Transformers: roadmap



- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder



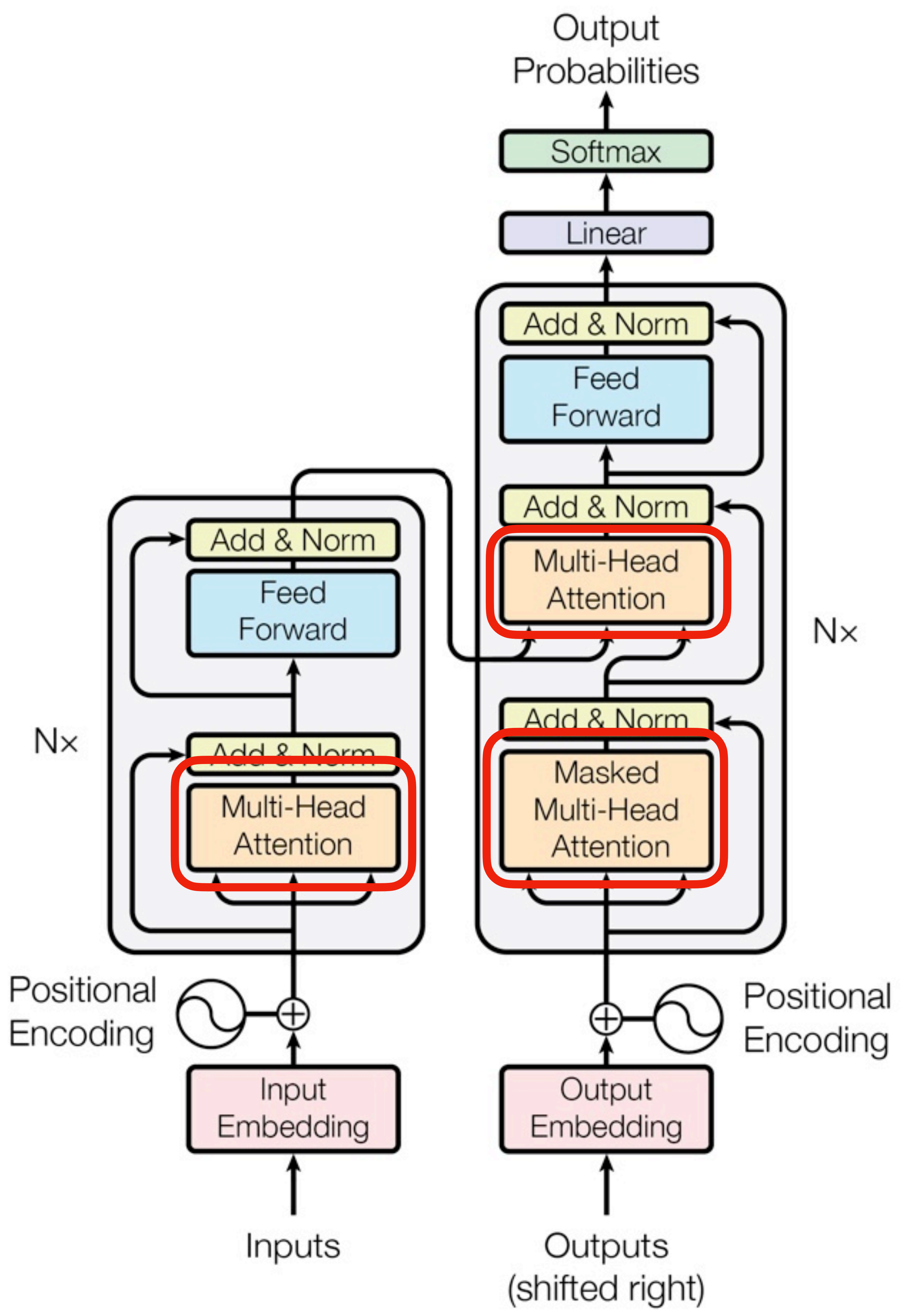
The Annotated Transformer

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

https://nlp.seas.harvard.edu/annotated-transformer/

Transformers: roadmap



- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers

General form of attention

- A more general form: use a set of **keys** and **values** $(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_n, \mathbf{v}_n)$, $\mathbf{k}_i \in \mathbb{R}^{d_k}$, $\mathbf{v}_i \in \mathbb{R}^{d_v}$, **keys** are used to compute the attention scores and **values** are used to compute the output vector

- Attention always involves the following steps:
  - Computing the **attention scores** $\mathbf{e} = g(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}^n$
  - Taking softmax to get **attention distribution** α:

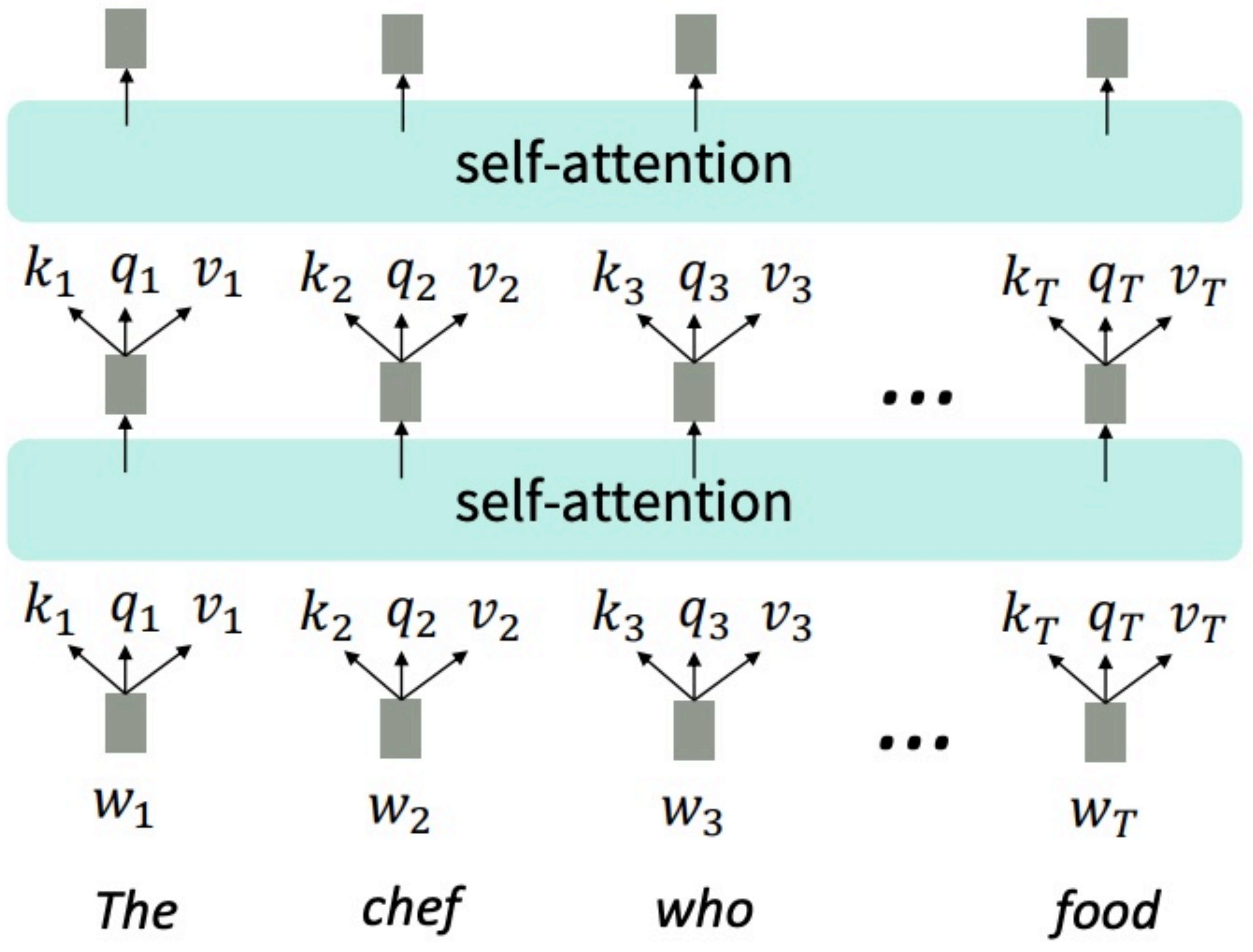
$$\alpha = \mathrm{softmax}(\mathbf{e}) \in \mathbb{R}^n$$

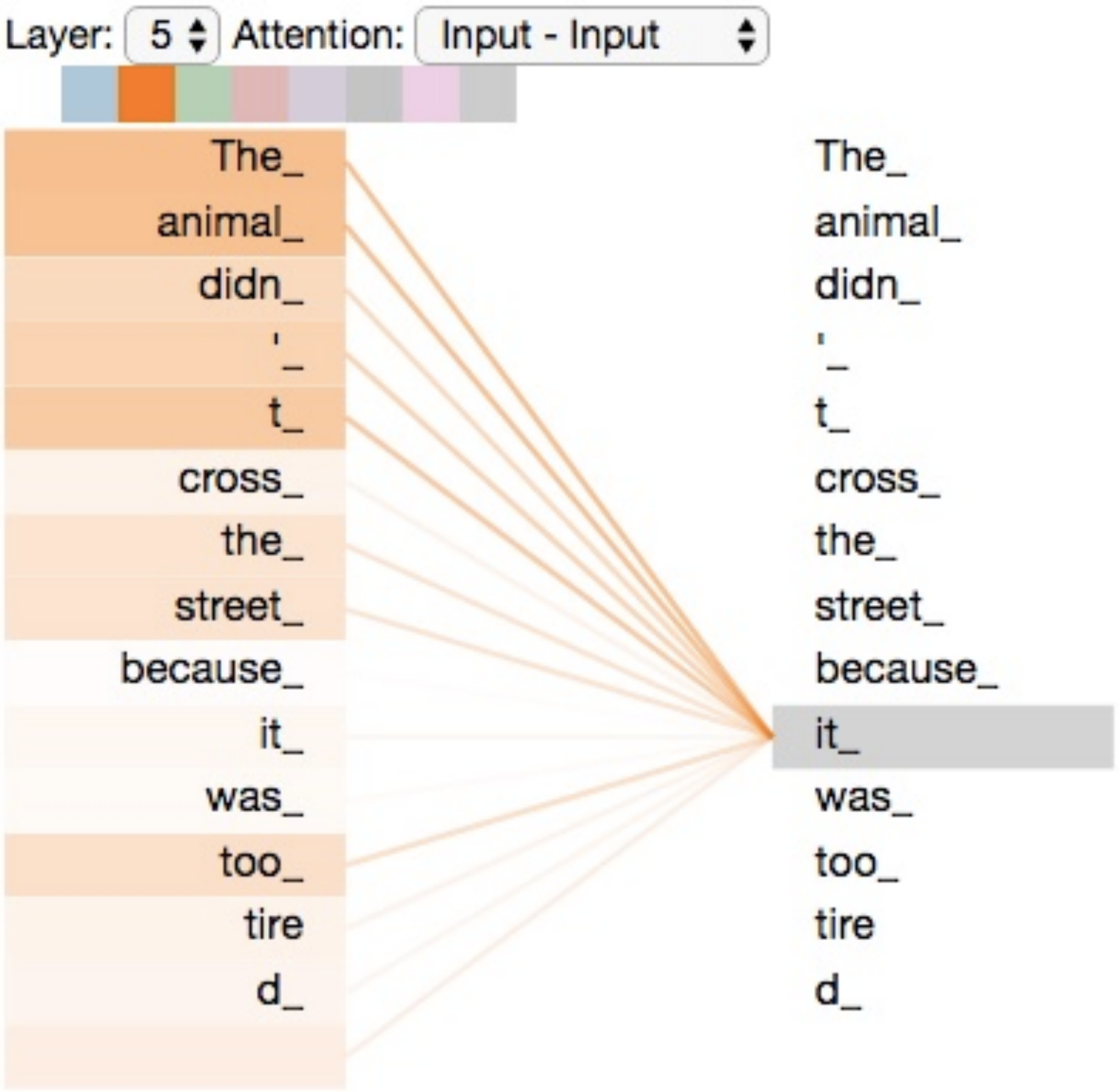
  - Using attention distribution to take **weighted sum** of values:

$$\mathbf{o} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i \in \mathbb{R}^{d_v}$$

Self-attention

- In NMT, **query** = decoder's hidden state, **keys** = **values** = encoder's hidden states
- Self-attention = attention from the sequence to **itself**
- Self-attention: let's use each word in a sequence as the **query**, and all other words in the sequence as **keys** and **values**.





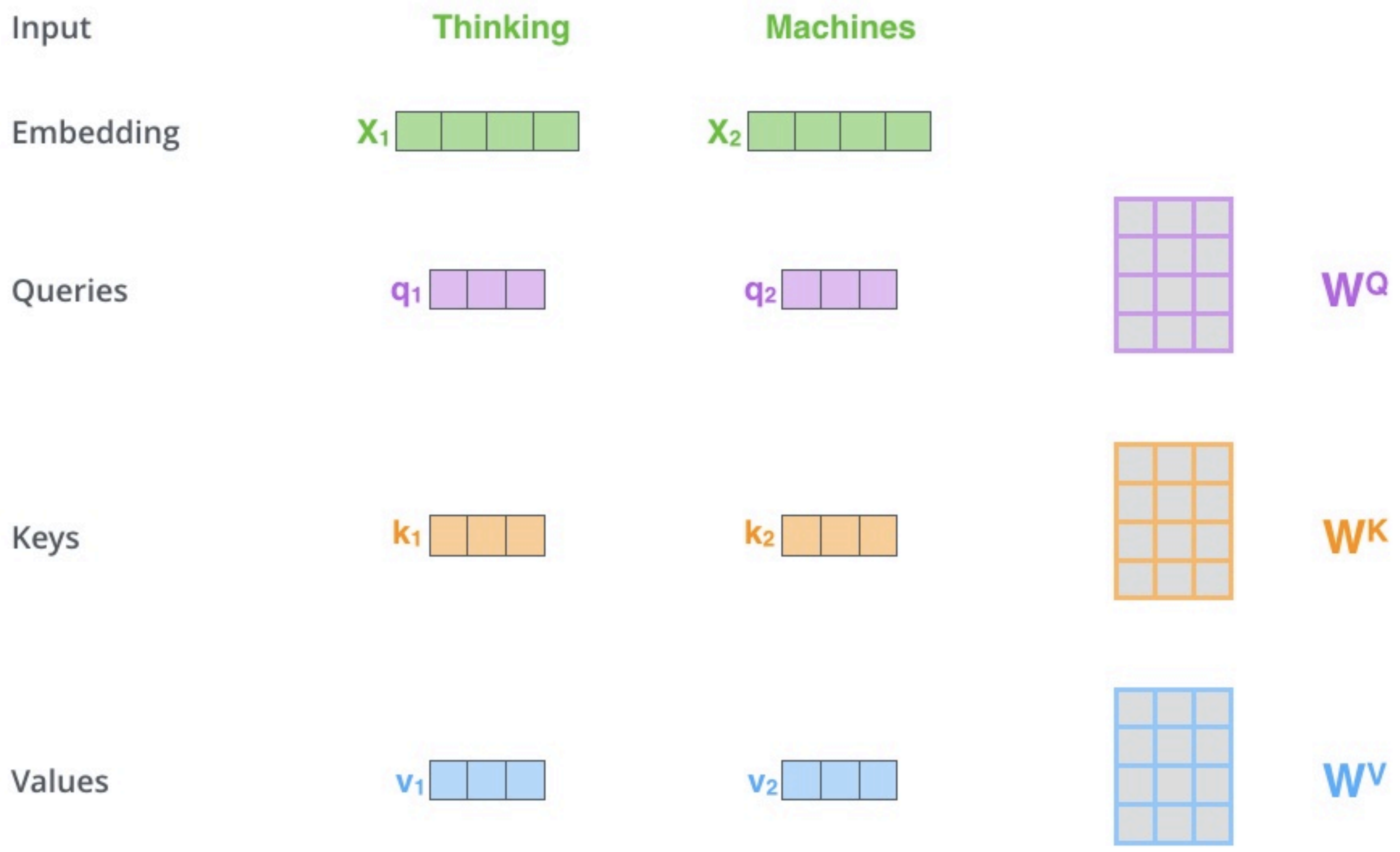
https://jalammar.github.io/illustrated-transformer/

Self-attention

Step #1: Transform each input vector into three vectors: **query**, **key**, and **value** vectors

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q \in \mathbb{R}^{d_q} \qquad \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K \in \mathbb{R}^{d_k} \qquad \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V \in \mathbb{R}^{d_v}$$

$$\mathbf{W}^Q \in \mathbb{R}^{d_{in} \times d_q} \qquad \mathbf{W}^K \in \mathbb{R}^{d_{in} \times d_k} \qquad \mathbf{W}^V \in \mathbb{R}^{d_{in} \times d_v}$$



Note that we use row vectors here;
It is also common to write

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i \in \mathbb{R}^{d_q}$$

for $\mathbf{x}_i$ = a column vector

https://jalammar.github.io/illustrated-transformer/

Self-attention

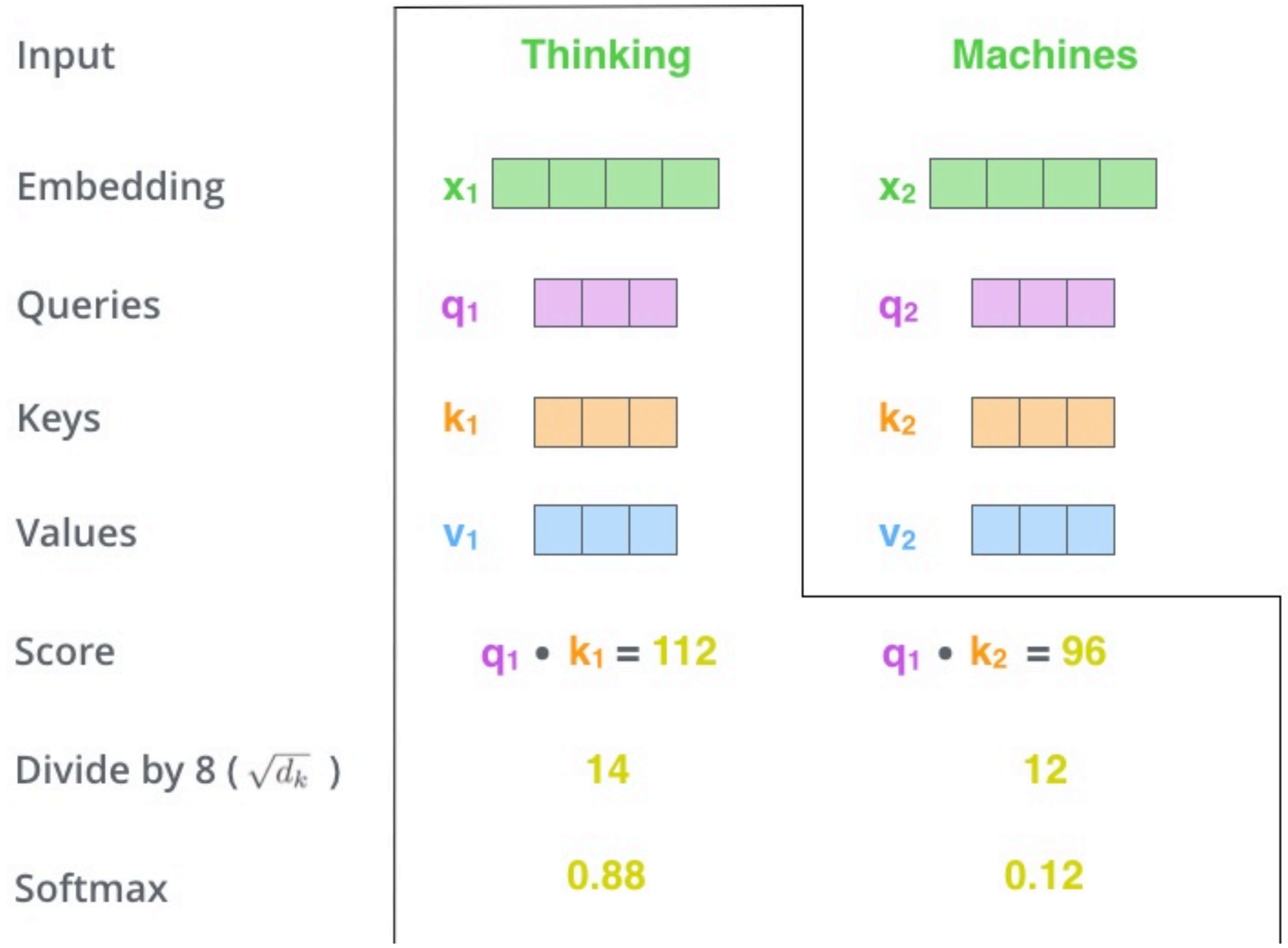
Step #2: Compute pairwise similarities between keys and queries; normalize with softmax

For each $\mathbf{q}_i$, compute attention scores and attention distribution:

$$\alpha_{i,j} = \text{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}\right)$$

aka. "scaled dot product"

It must be $d_q = d_k$ in this case



Q. Why scaled dot product?

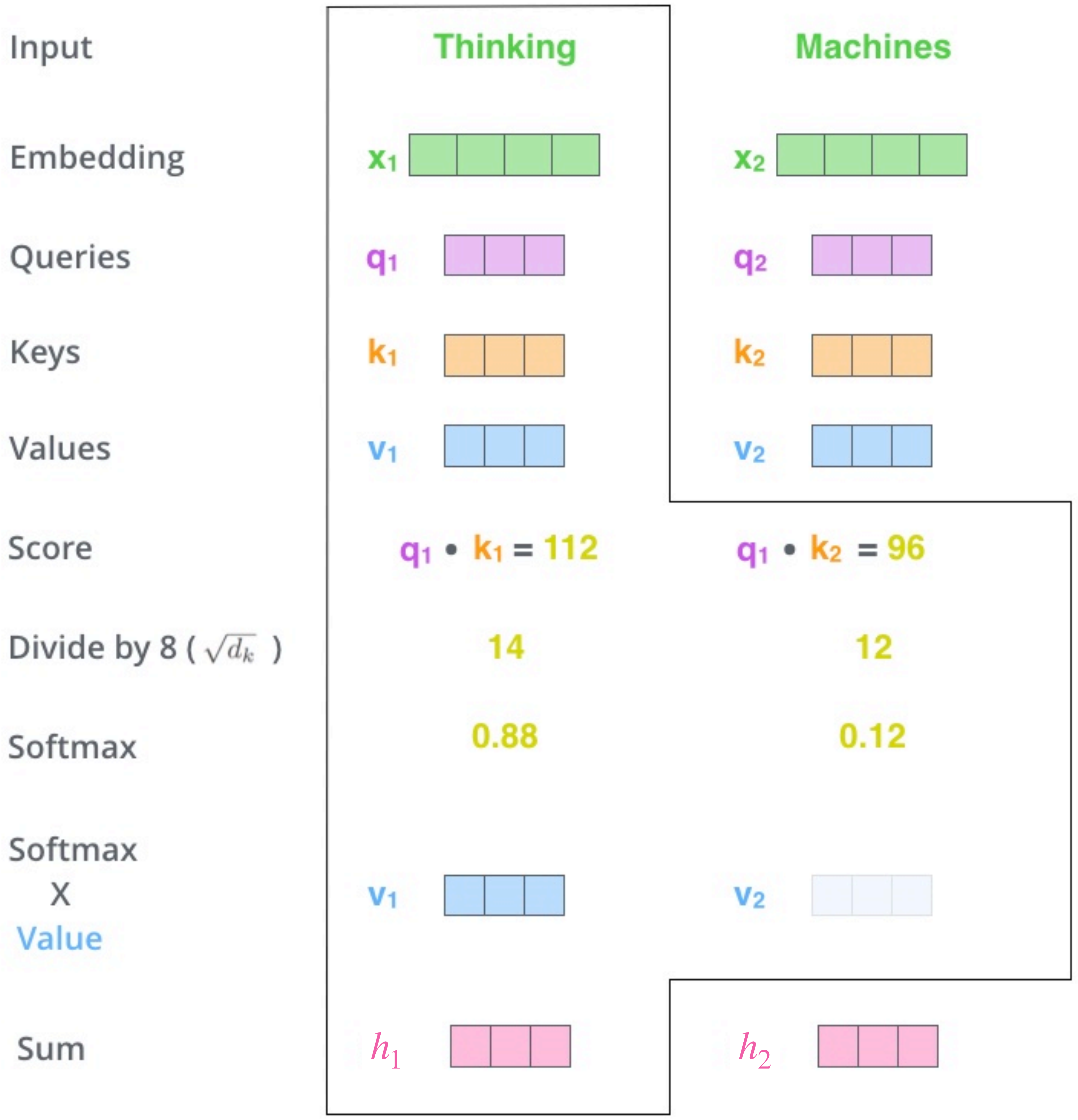
To avoid the dot product to become too large for larger d_k; scaling the dot product by $\dfrac{1}{\sqrt{d_k}}$ is easier for optimization

https://jalammar.github.io/illustrated-transformer/

Self-attention

Step #3: Compute output for each input as weighted sum of values

$$\mathbf{h}_i = \sum_{j=1}^{n} \alpha_{i,j} \mathbf{v}_j \in \mathbb{R}^{d_v}$$



https://jalammar.github.io/illustrated-transformer/

Self-attention

What would be the output vector for the word "Thinking" approximately?

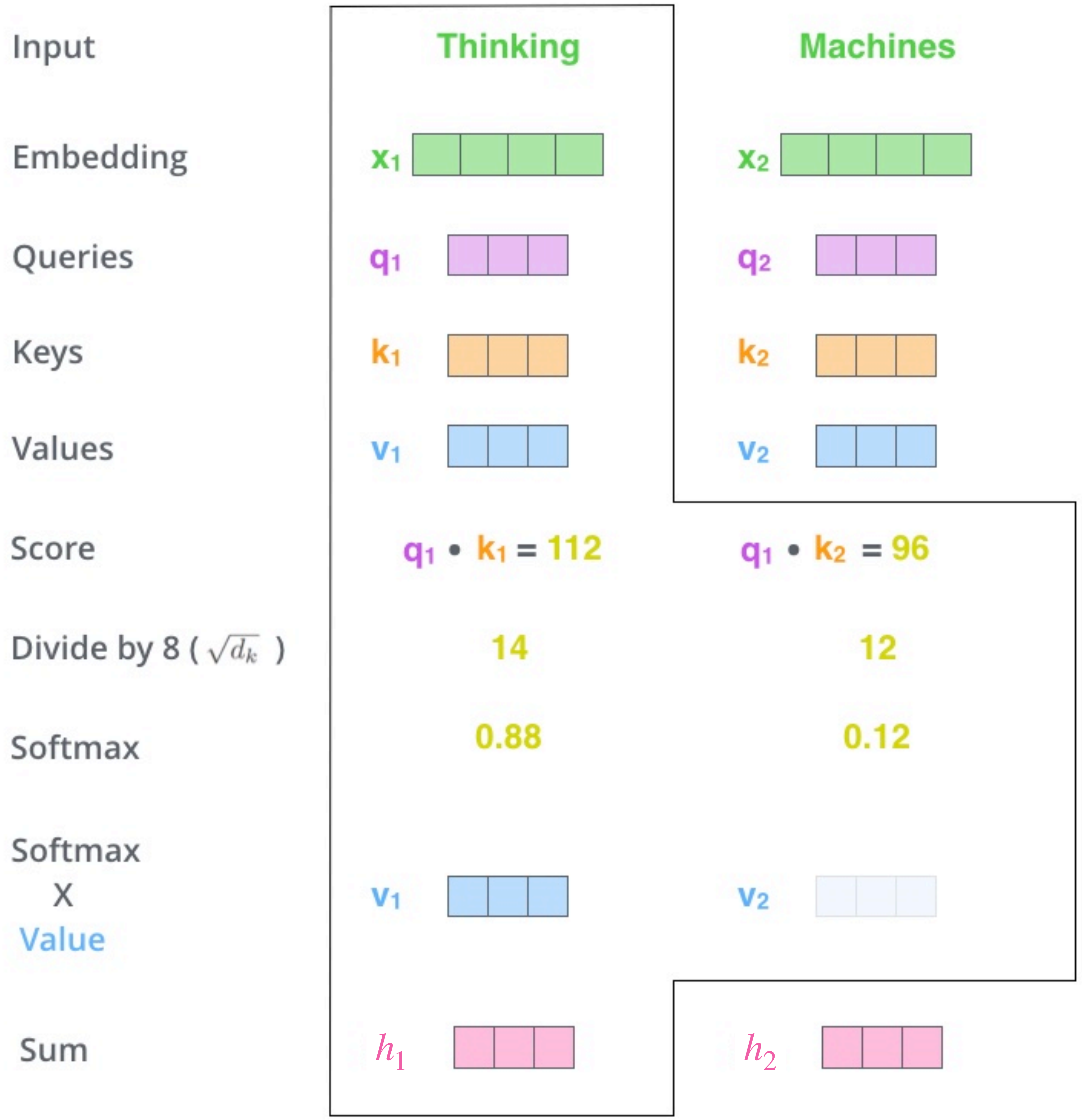
(A) $0.5\mathbf{v}_1 + 0.5\mathbf{v}_2$

(B) $0.54\mathbf{v}_1 + 0.46\mathbf{v}_2$

(C) $0.88\mathbf{v}_1 + 0.12\mathbf{v}_2$

(D) $0.12\mathbf{v}_1 + 0.88\mathbf{v}_2$

(C) is correct.



https://jalammar.github.io/illustrated-transformer/

Self-attention: matrix notations

$X \in \mathbb{R}^{n \times d_{in}}$ (n = input length)

$$Q = XW^Q, K = XW^K, V = XW^V$$

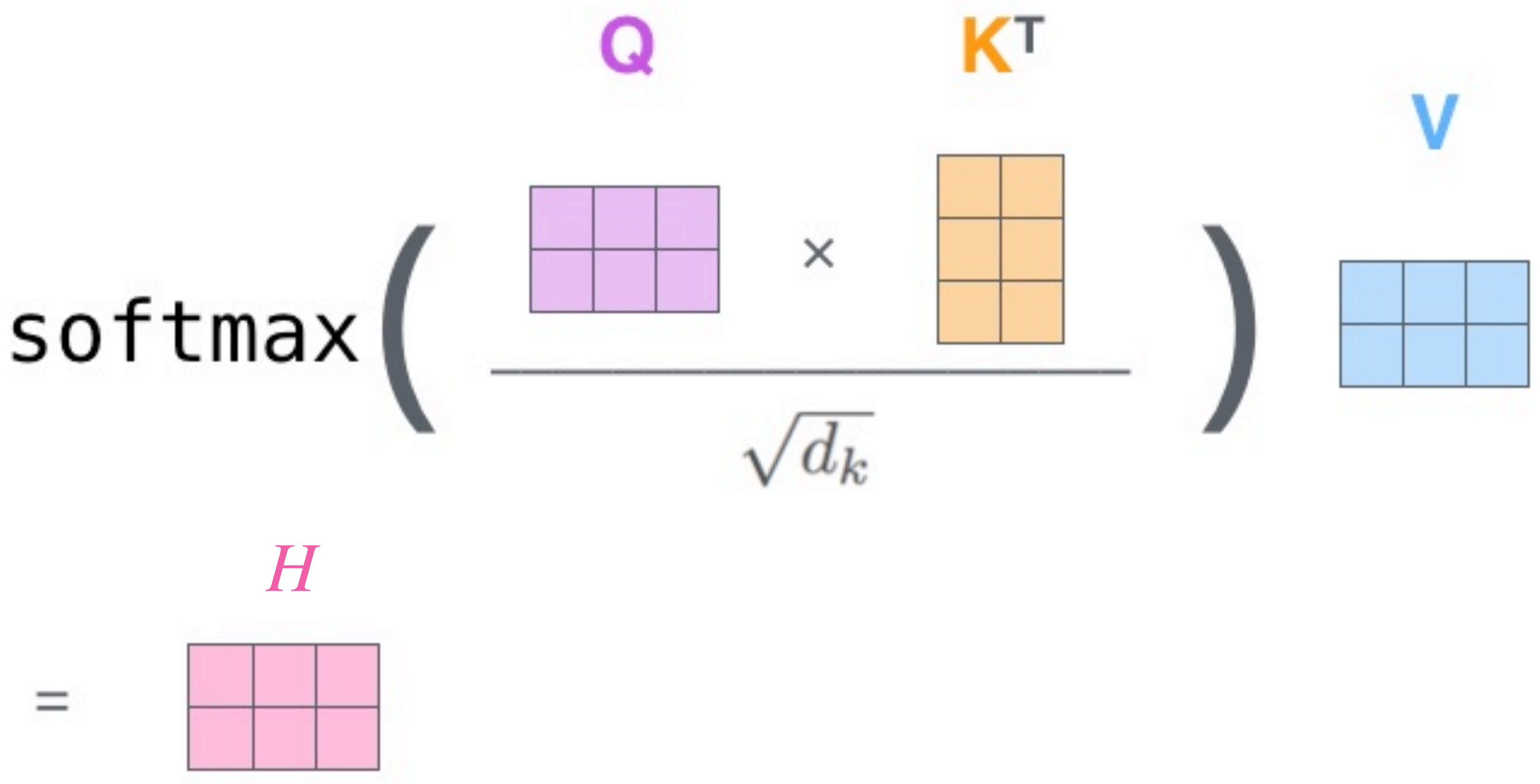
where $W^Q \in \mathbb{R}^{d_{in} \times d_q}, W^K \in \mathbb{R}^{d_{in} \times d_k}, W^V \in \mathbb{R}^{d_{in} \times d_v}$

$n \times d_q$ $\quad$ $d_k \times n$

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

$n \times d_v$

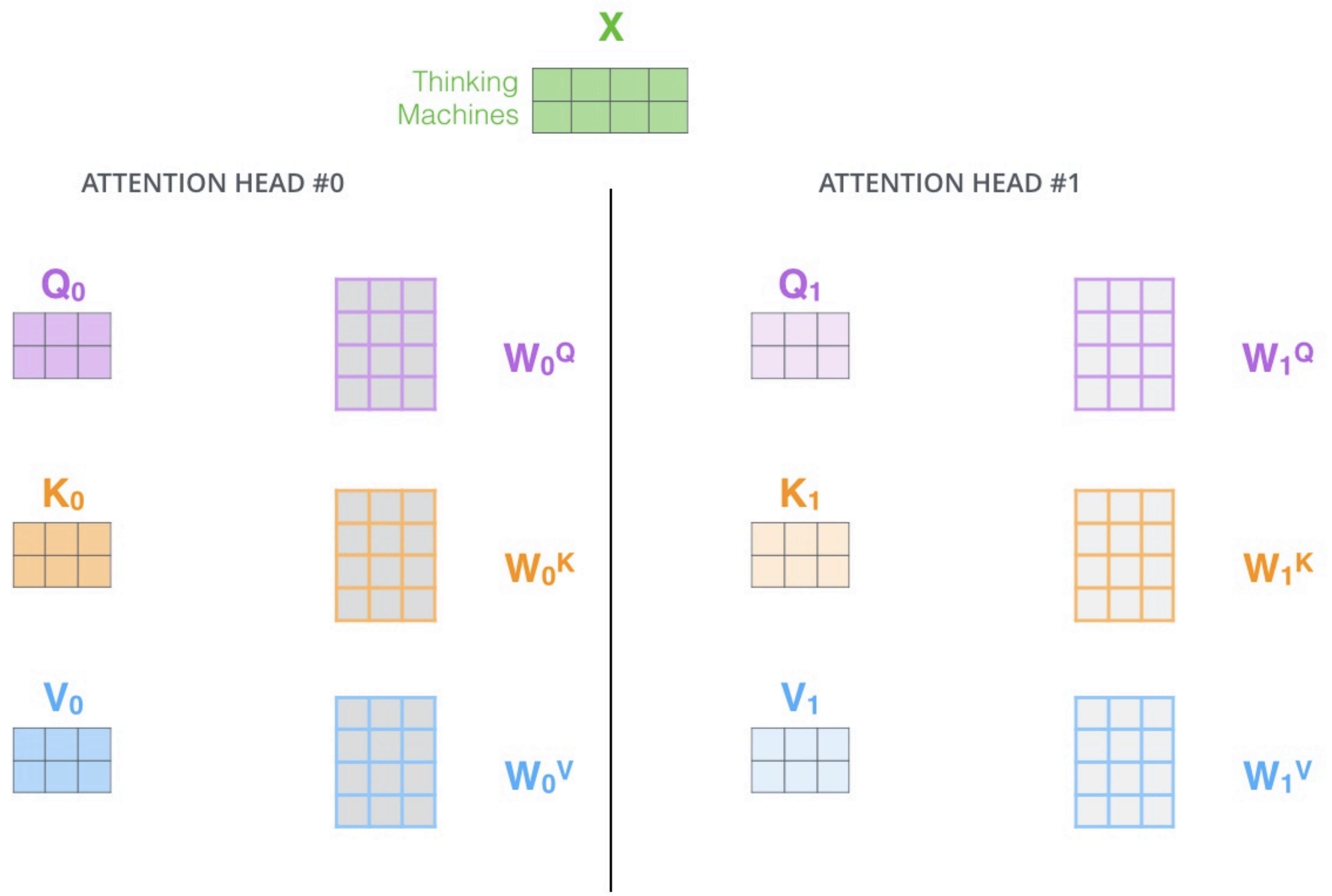
Q: What is this softmax operation?

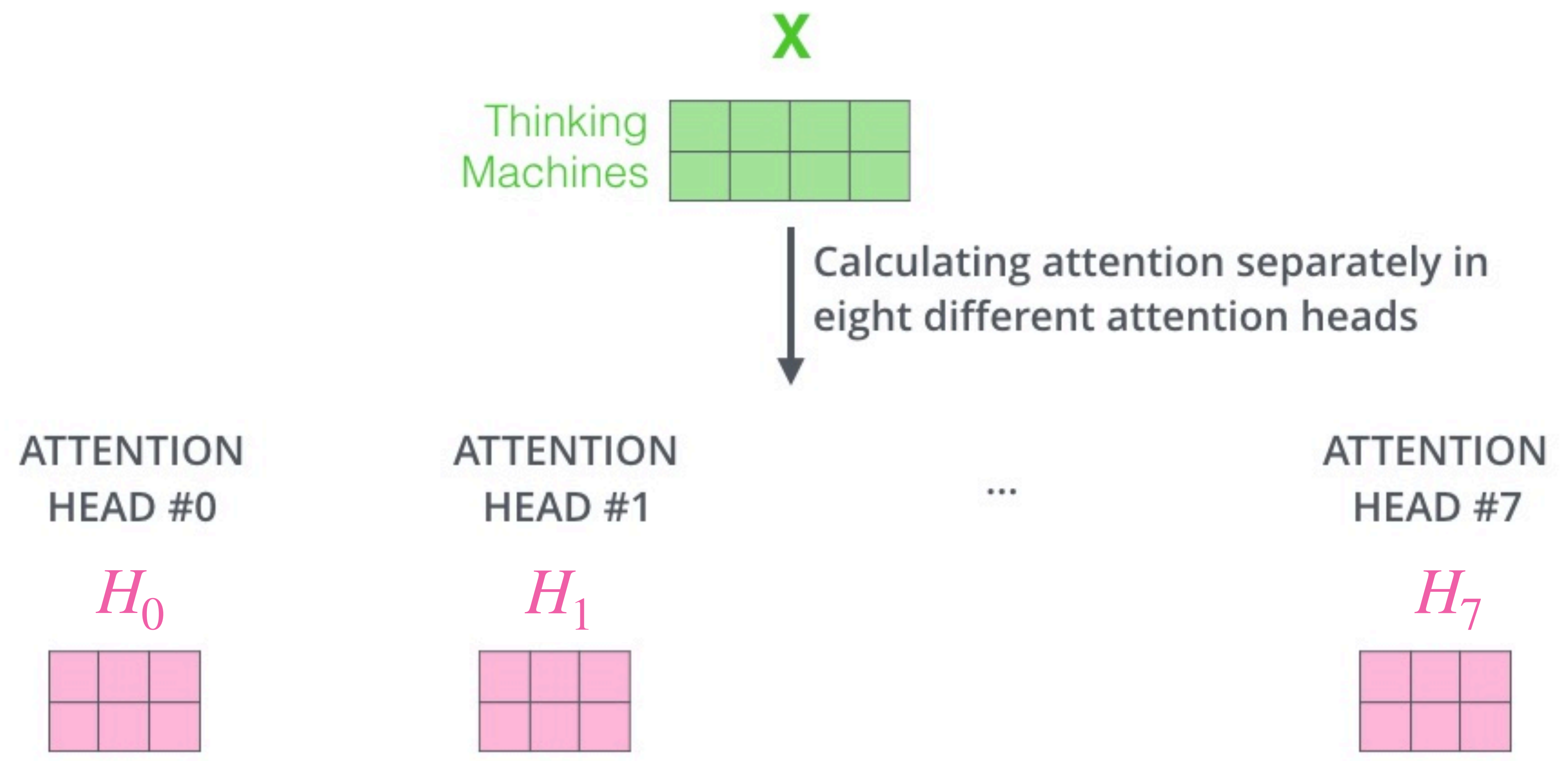


Multi-head attention

"The Beast with Many Heads"

- It is better to use multiple attention functions instead of one!
  - Each attention function ("head") can focus on different positions.
- It gives the attention layer multiple "representation subspaces"





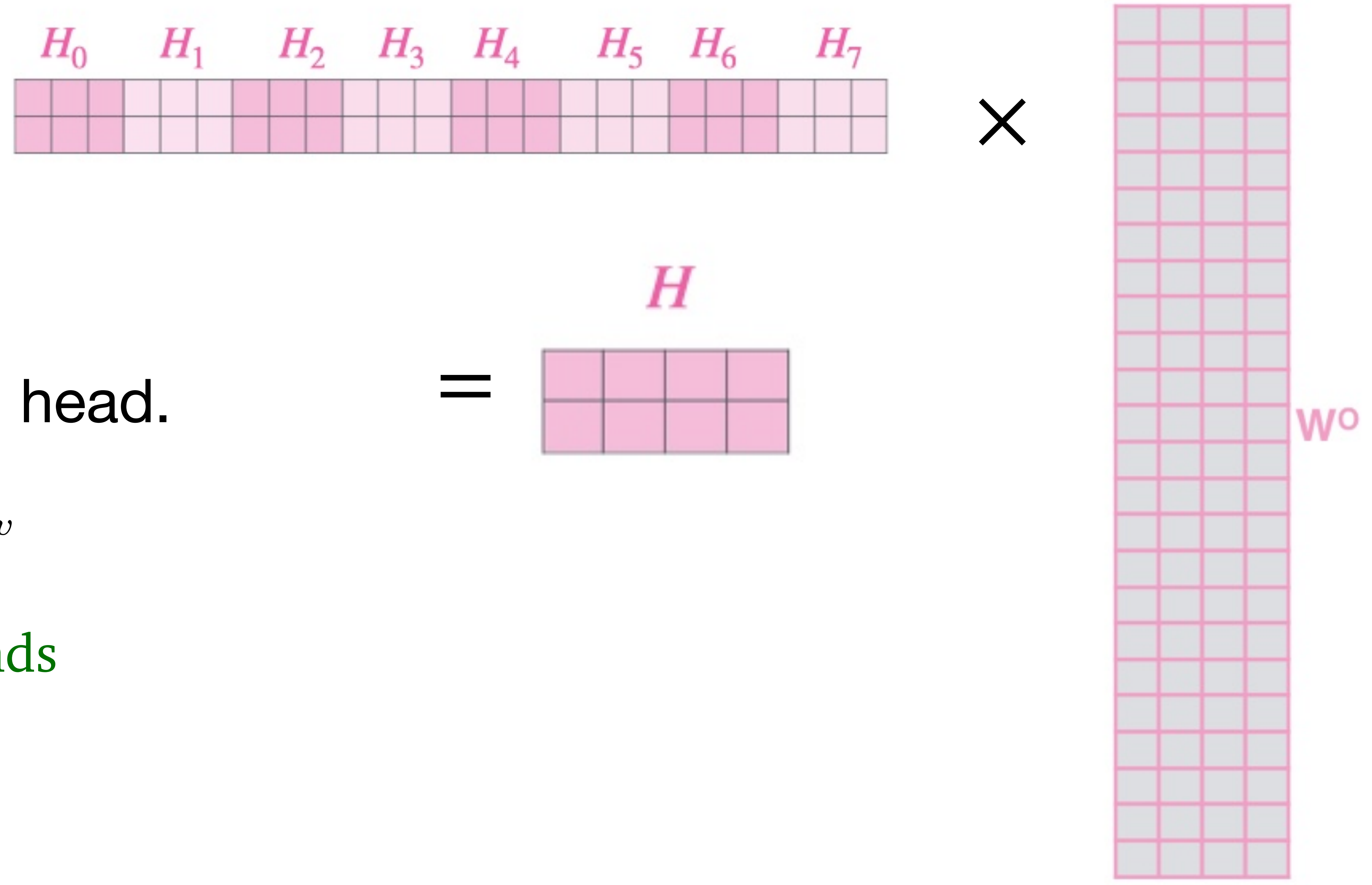
https://jalammar.github.io/illustrated-transformer/

Multi-head attention

"The Beast with Many Heads"

Finally, we just concatenate all the heads and apply an output projection matrix.

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\mathrm{head}_i = \mathrm{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$



- In practice, we use a ***reduced*** dimension for each head.

$$W_i^Q \in \mathbb{R}^{d_{in} \times d_q}, W_i^K \in \mathbb{R}^{d_{in} \times d_k}, W_i^V \in \mathbb{R}^{d_{in} \times d_v}$$

$$d_q = d_k = d_v = d/m \quad d = \text{hidden size, } m = \text{\# of heads}$$

$$W^O \in \mathbb{R}^{d \times d_{out}}$$

- The total computational cost is similar to that of single-head attention with full dimensionality.

https://jalammar.github.io/illustrated-transformer/

Multi-head attention

"The Beast with Many Heads"

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_\mathrm{h})W^O$$
$$\mathrm{head}_i = \mathrm{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

- We can think of multi-head attention (MHA) layer as an abstraction layer that maps a sequence of input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_{in}}$ to a sequence of *n* vectors: $\mathbf{h}_1, \ldots, \mathbf{h}_n \in \mathbb{R}^{d_{out}}$

If we stack multiple layers, usually $d_{in} = d_{out} = d$

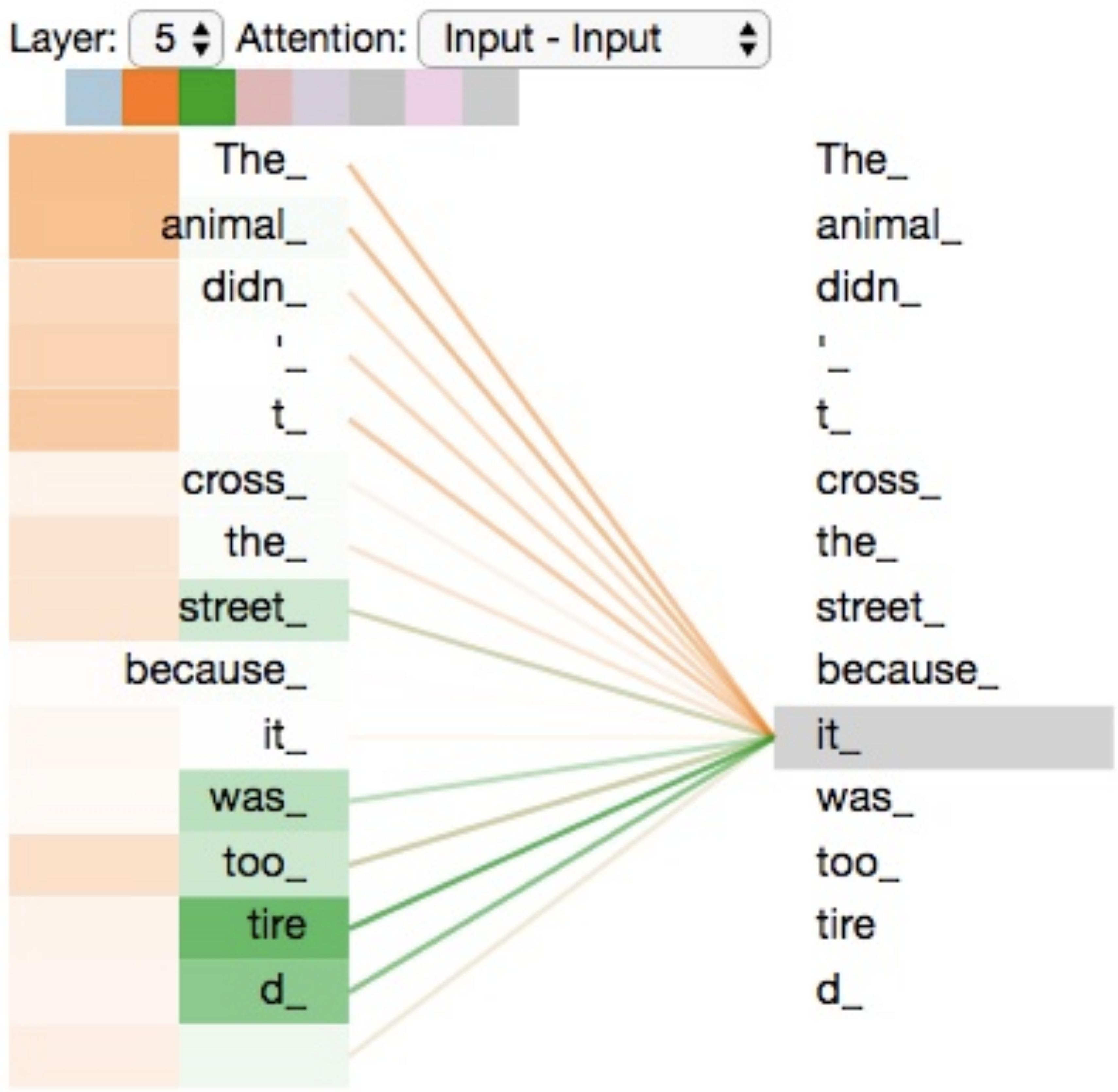
- The same abstraction as RNNs - used as a drop-in replacement for an RNN layer

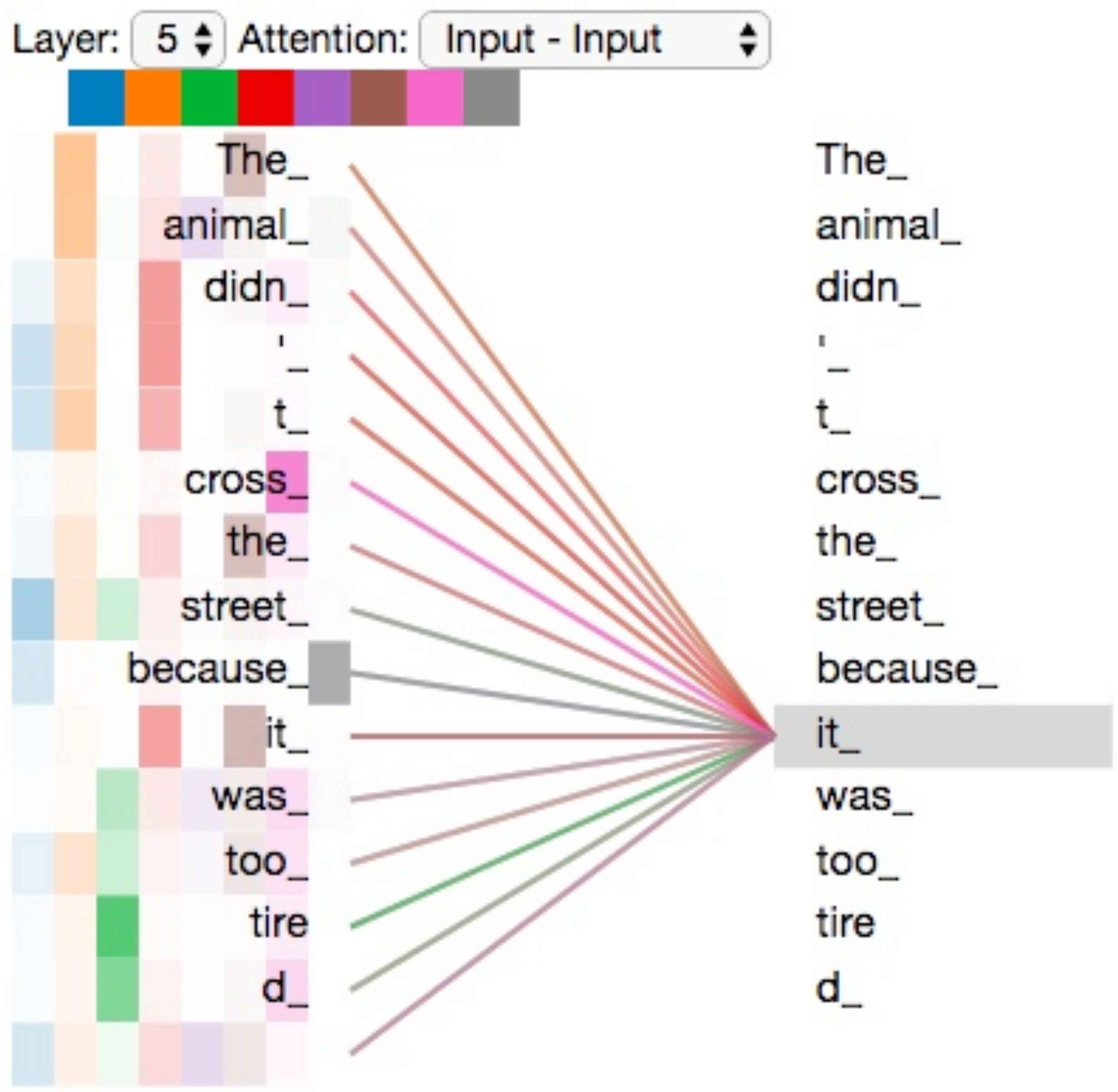
$$\mathbf{h}_t = f(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^{d_{out}}$$

Much easier to parallelize, more expensive to scale up to longer sequences!

https://jalammar.github.io/illustrated-transformer/

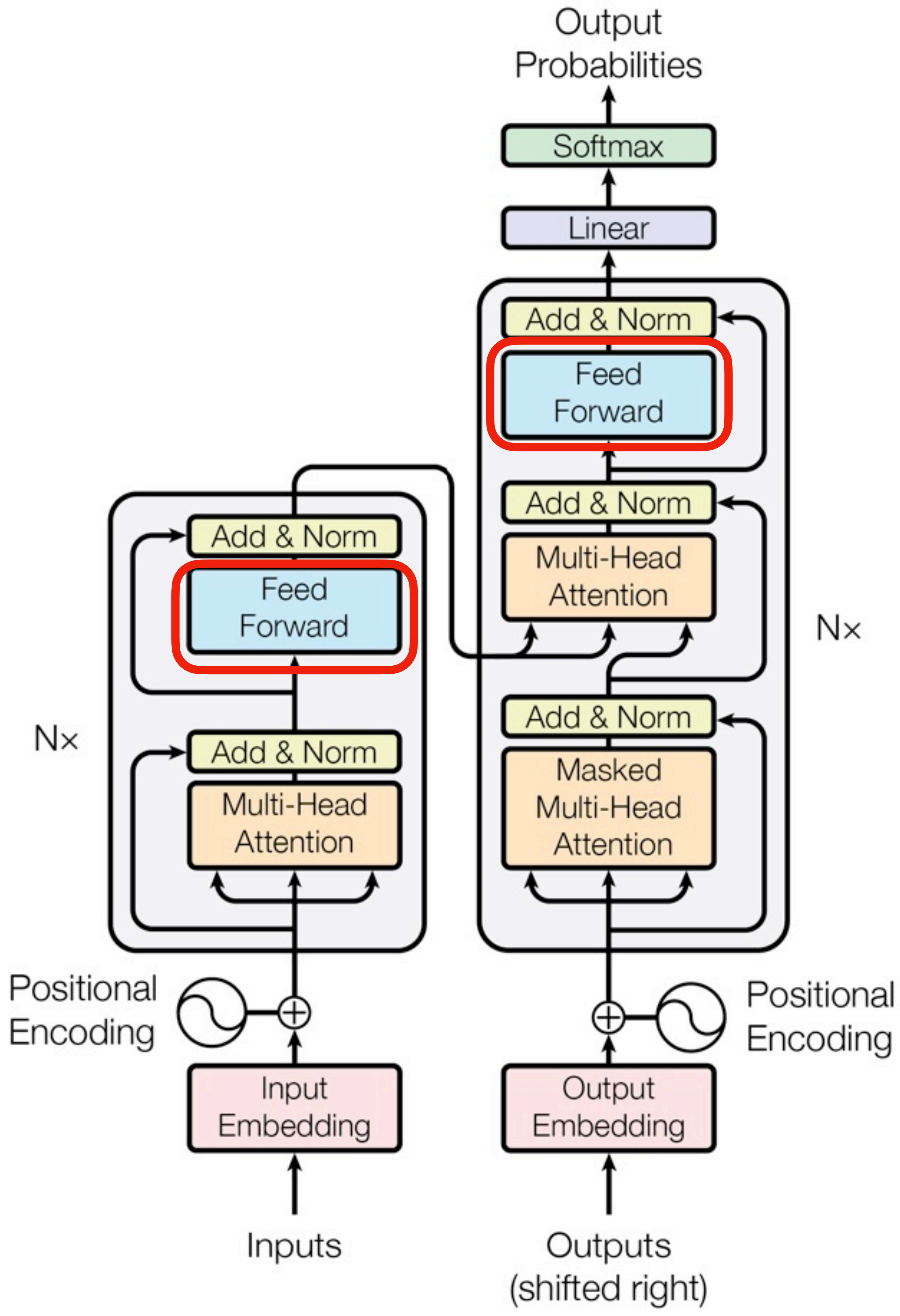
What does multi-head attention learn?





https://github.com/jessevig/bertviz

Transformers: roadmap



- Self-attention and multi-head self-attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers

Adding nonlinearities: Feed-forward layers

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

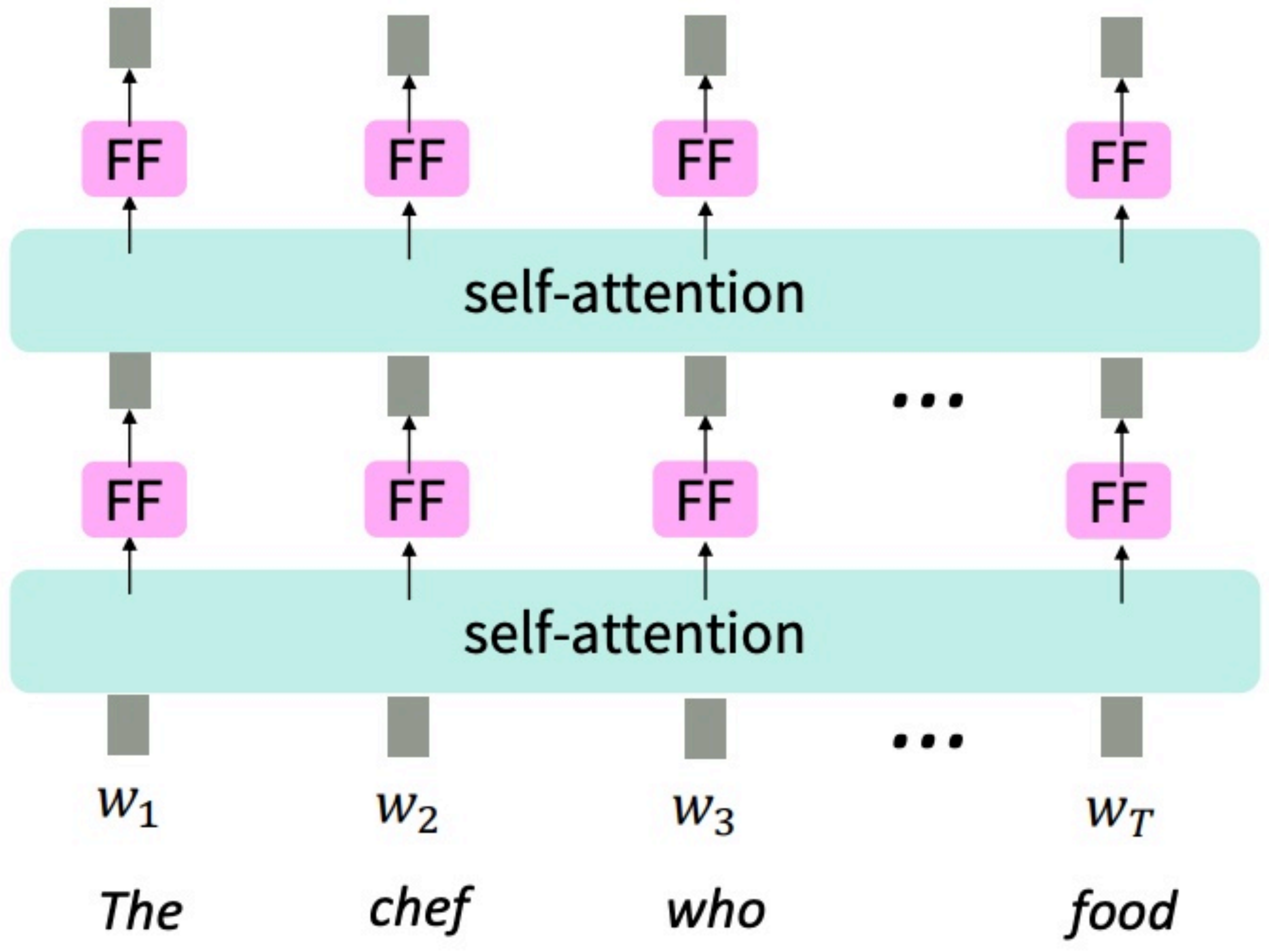
- Simple fix: add a feed-forward network to post-process each output vector

$$\mathrm{FFN}(\mathbf{x}_i) = \mathrm{ReLU}(\mathbf{x}_i \mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

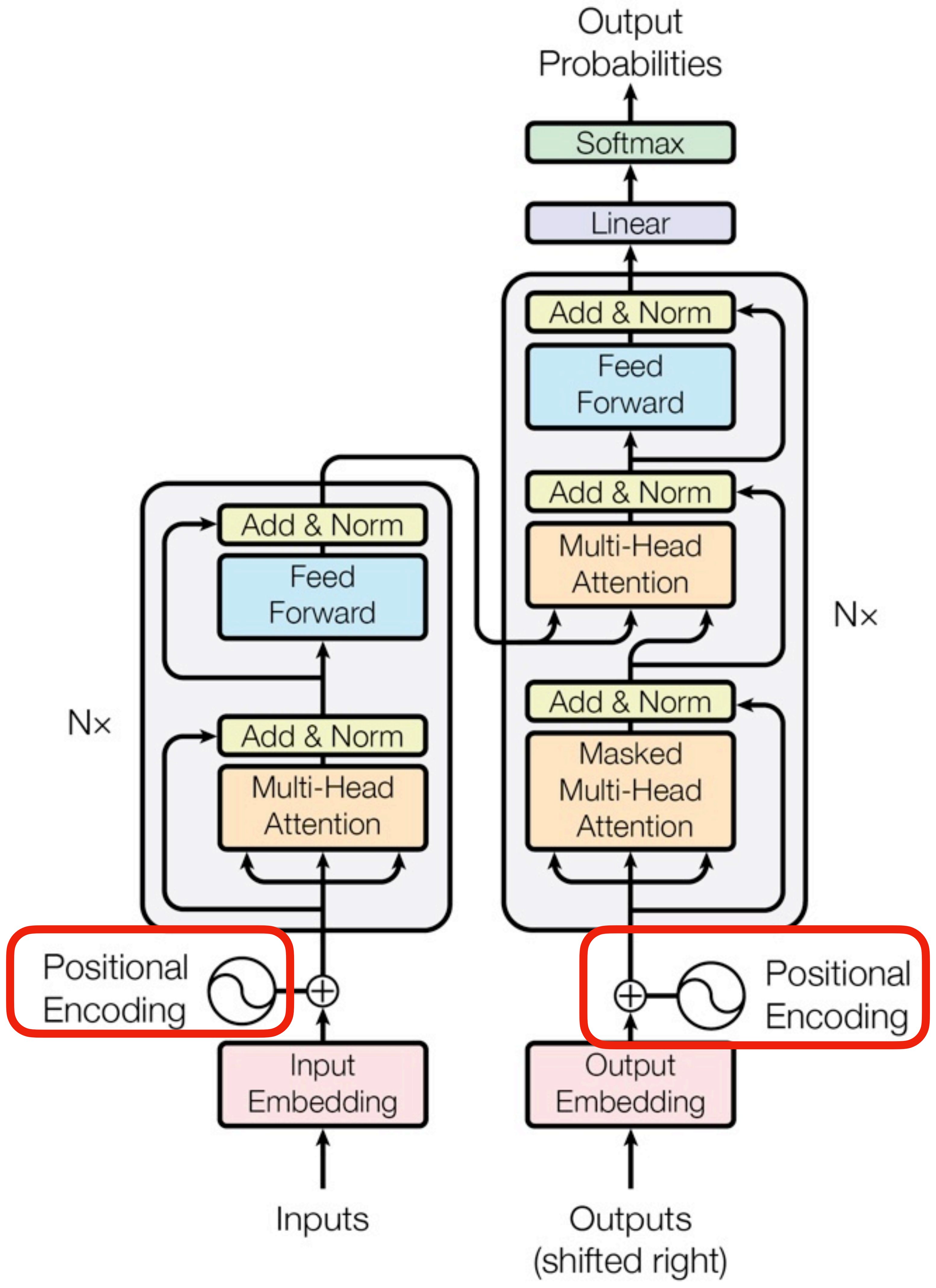
$$\mathbf{W}_1 \in \mathbb{R}^{d \times d_{ff}}, \mathbf{b}_1 \in \mathbb{R}^{d_{ff}}$$

$$\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d}, \mathbf{b}_2 \in \mathbb{R}^{d}$$

Usually, $d_{ff} = 4d$



Transformers: roadmap



- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers

Modeling order information: positional encoding

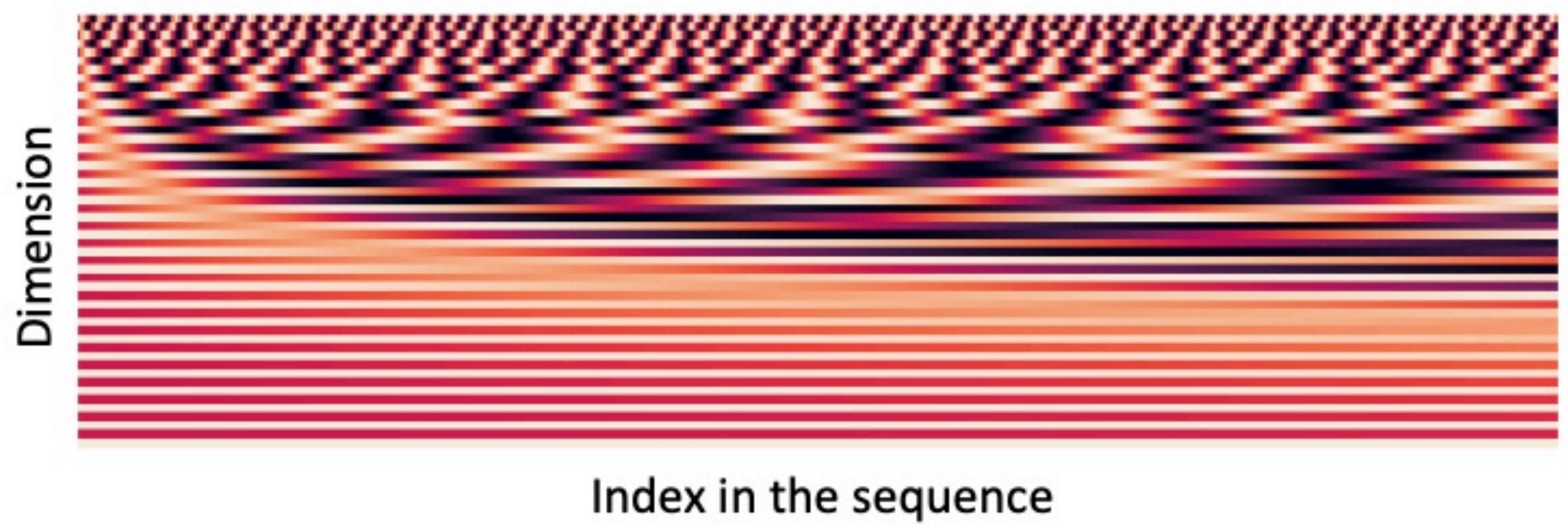
- Unlike RNNs, self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values

- Solution: Add **positional embeddings** to the input embeddings: $\mathbf{p}_i \in \mathbb{R}^d$ for $i = 1,2,\ldots,n$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{p}_i$$

- **Sinusoidal positional embeddings**: sine and cosine functions of different frequencies:

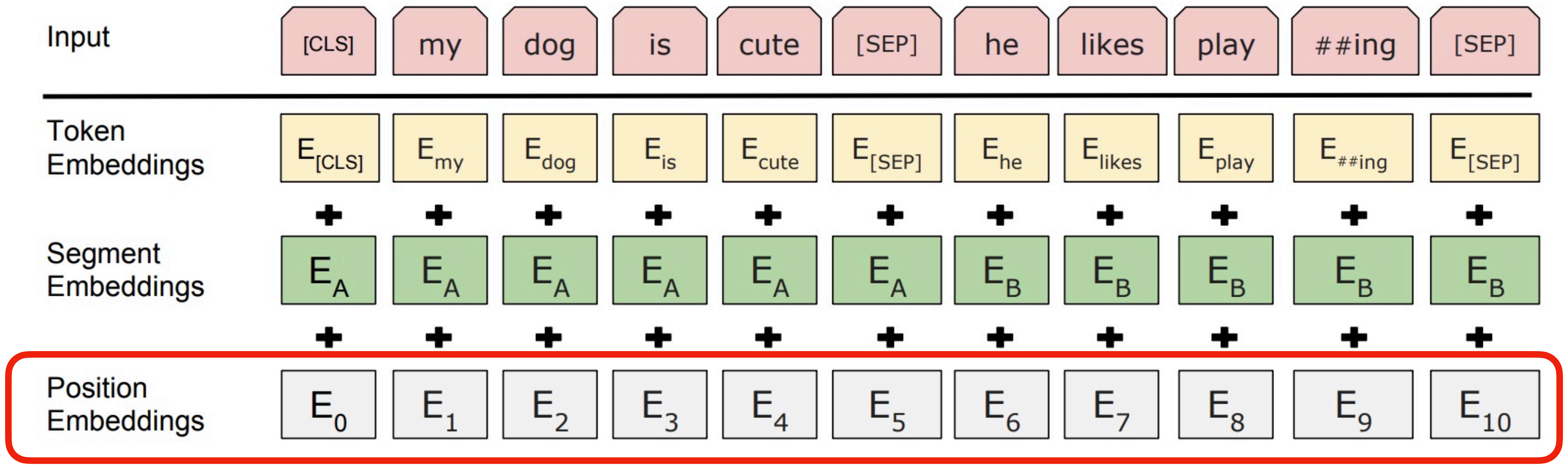
$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



  - **Pros**: Periodicity + can extrapolate to longer sequences
  - **Cons**: Not learnable

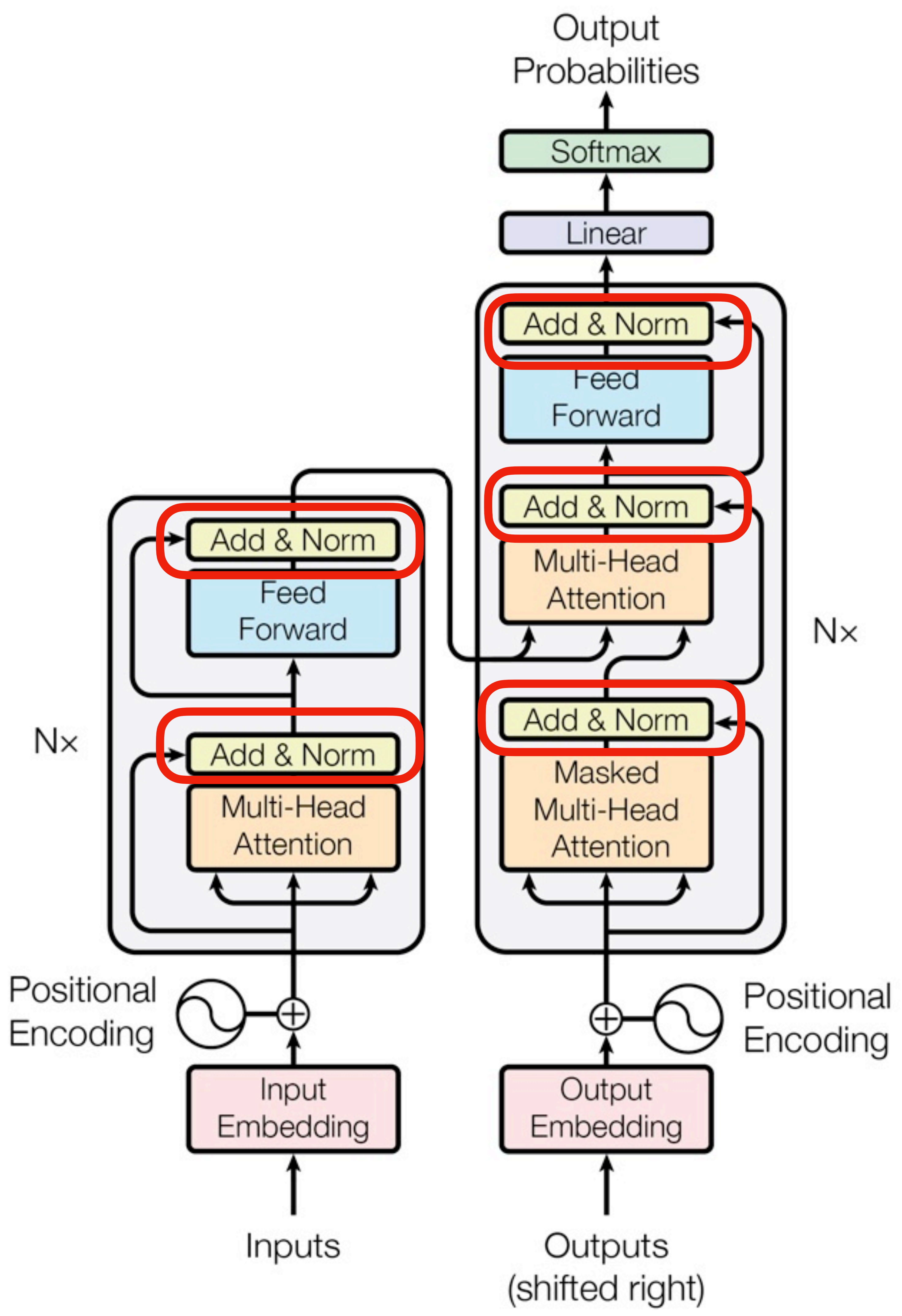
Modeling order information: positional encoding

- **Absolute positional embeddings:** let all $\mathbf{p}_i$ be learnable parameters
    - $P \in \mathbb{R}^{d \times L}$ for L = max sequence length
- **Pros**: each position gets to be learned to fit the data
- **Cons**: can't extrapolate to indices outside of max sequence length L
- **Examples:** BERT, GPT-1, GPT-2, GPT-3, OPT



(Devlin et al., NAACL 2019)

Transformers: roadmap



- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers

How to make Transformers work for ***deep*** NNs?

Add & Norm: $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$

Residual connections (He et al., CVPR 2016)

Instead of $X^{(i)} = \mathrm{Layer}(X^{(i-1)})$ (i represents the layer)

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

This prevents the network from "forgetting" or distorting important information as it is processed by many layers.

How to make Transformers work for ***deep*** NNs?

Add & Norm: $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$

Layer normalization (Ba et al., 2016)

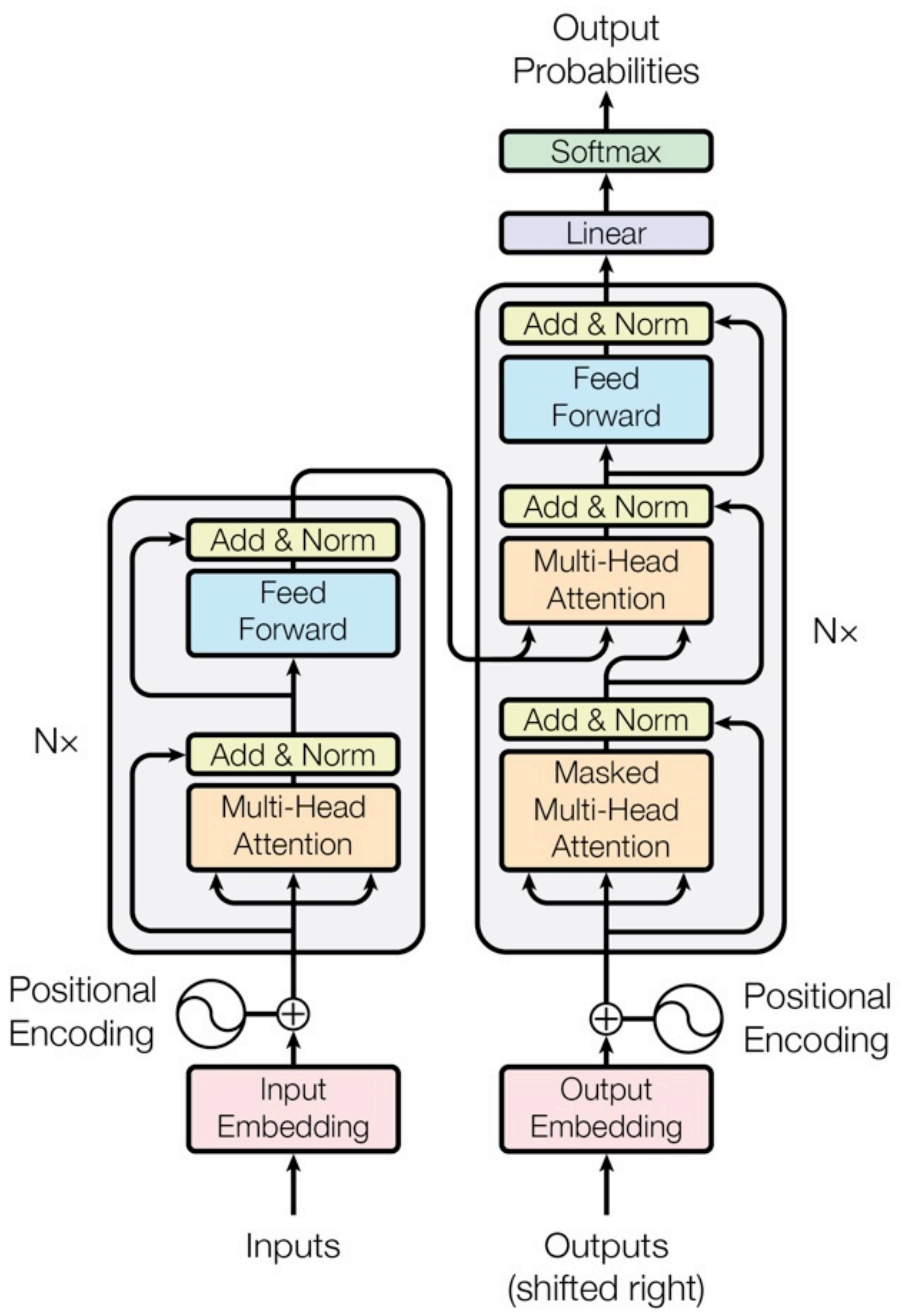
Problem: Difficult to train the parameters of a given layer because its input from the layer beneath keeps shifting.

Solution: Reduce variation by **normalizing** to zero mean and standard deviation of one **within each layer**.

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

$\gamma, \beta \in \mathbb{R}^d$ are learnable parameters

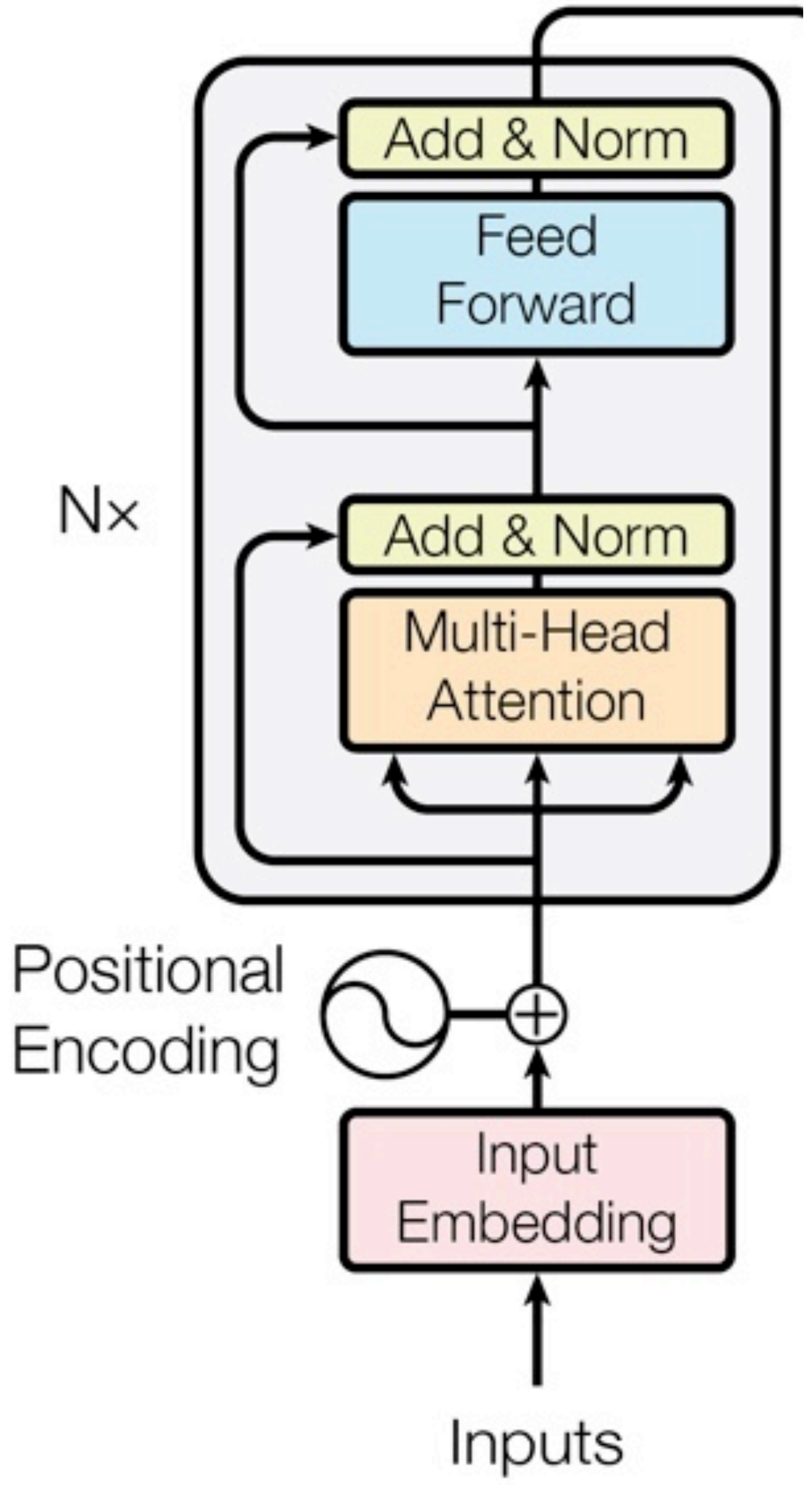
Transformers: roadmap



- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers

Let's put things together - Transformer encoder



From the bottom to the top:

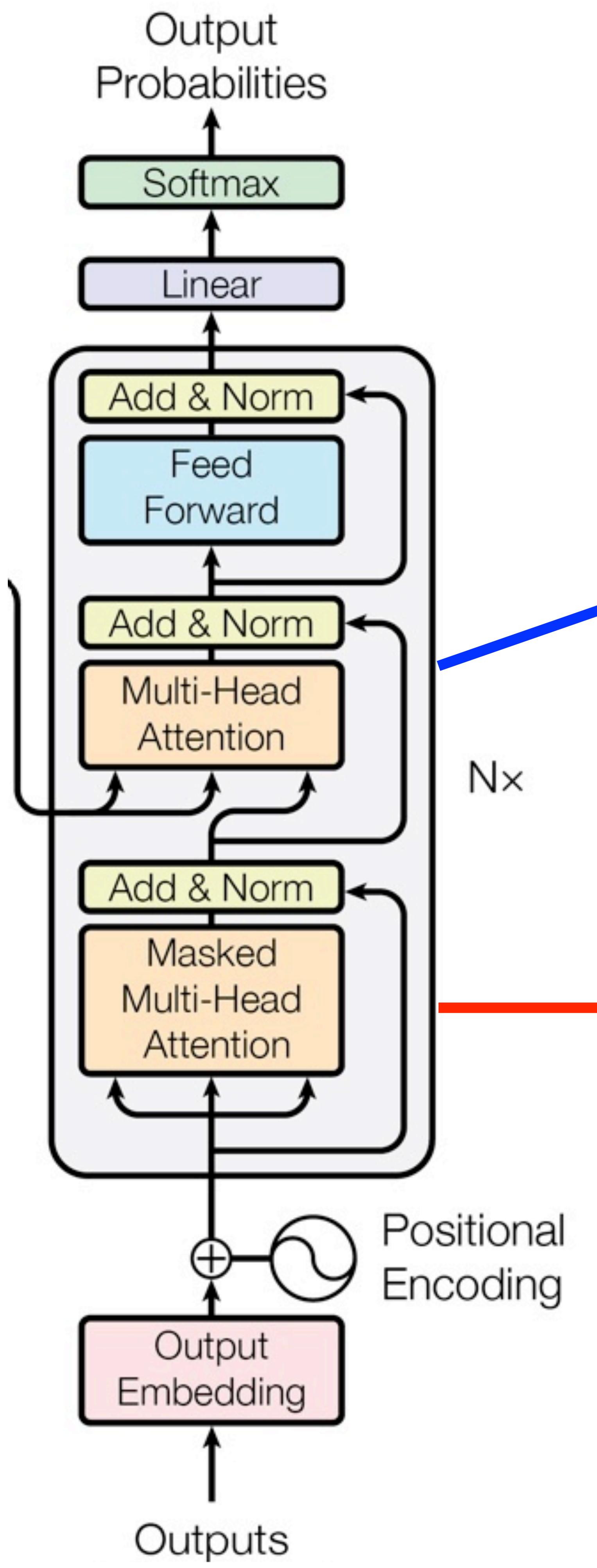
- Input embedding
- Positional encoding
- A stack of Transformer encoder layers

Transformer encoder is a stack of N layers, which consists of two sub-layers:

- Multi-head attention layer
- Feed-forward layer

$$\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_{in}} \longrightarrow \mathbf{h}_1, \ldots, \mathbf{h}_n \in \mathbb{R}^{d_{out}}$$

Let's put things together - Transformer decoder



Cross-attention between source and target sequence

Self-attention within target sequence

From the bottom to the top:

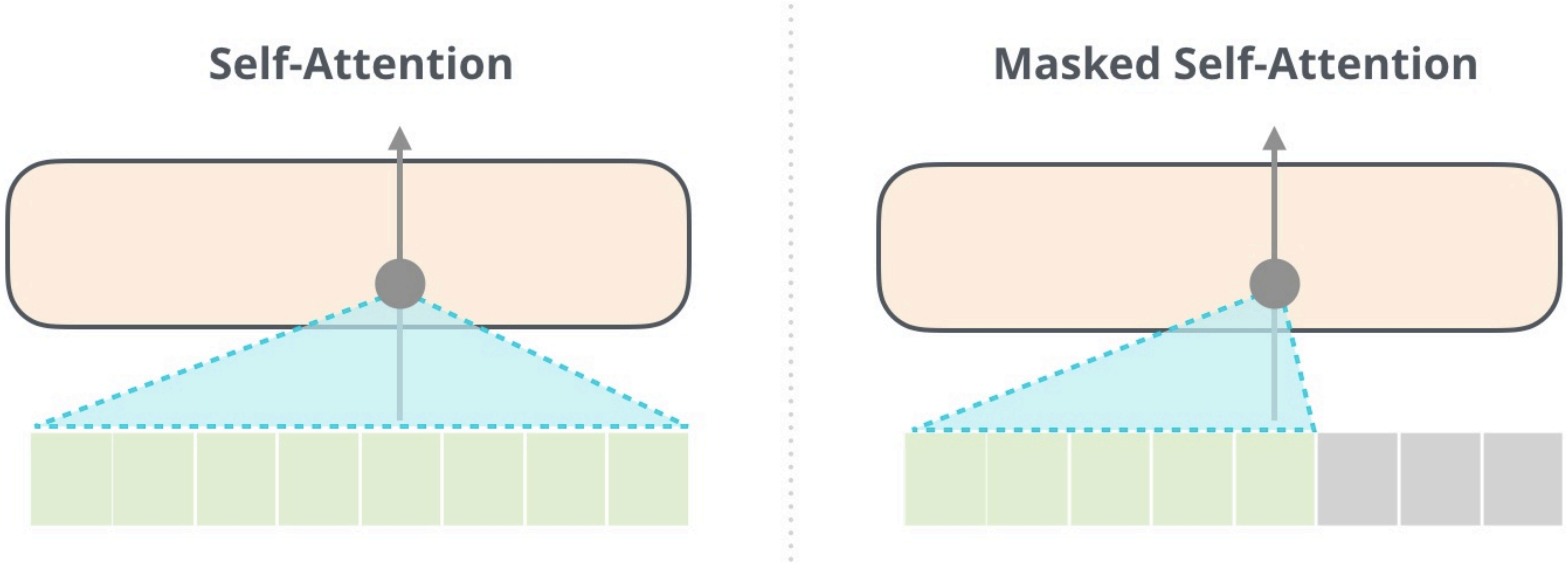
- Output embedding
- Positional encoding
- A stack of Transformer decoder layers
- Linear + softmax

Transformer decoder is a stack of N layers, which consists of **three** sub-layers:

- Masked multi-head attention
- Multi-head cross-attention
- Feed-forward layer
- (w/ Add & Norm between sub-layers)

Masked multi-head self-attention

- Key: You can’t see the future text for the decoder!



- Solution: for every $\mathbf{q}_i$, only attend to $\{(\mathbf{k}_j, \mathbf{v}_j)\}, j \leq i$ How to implement this? Masking!

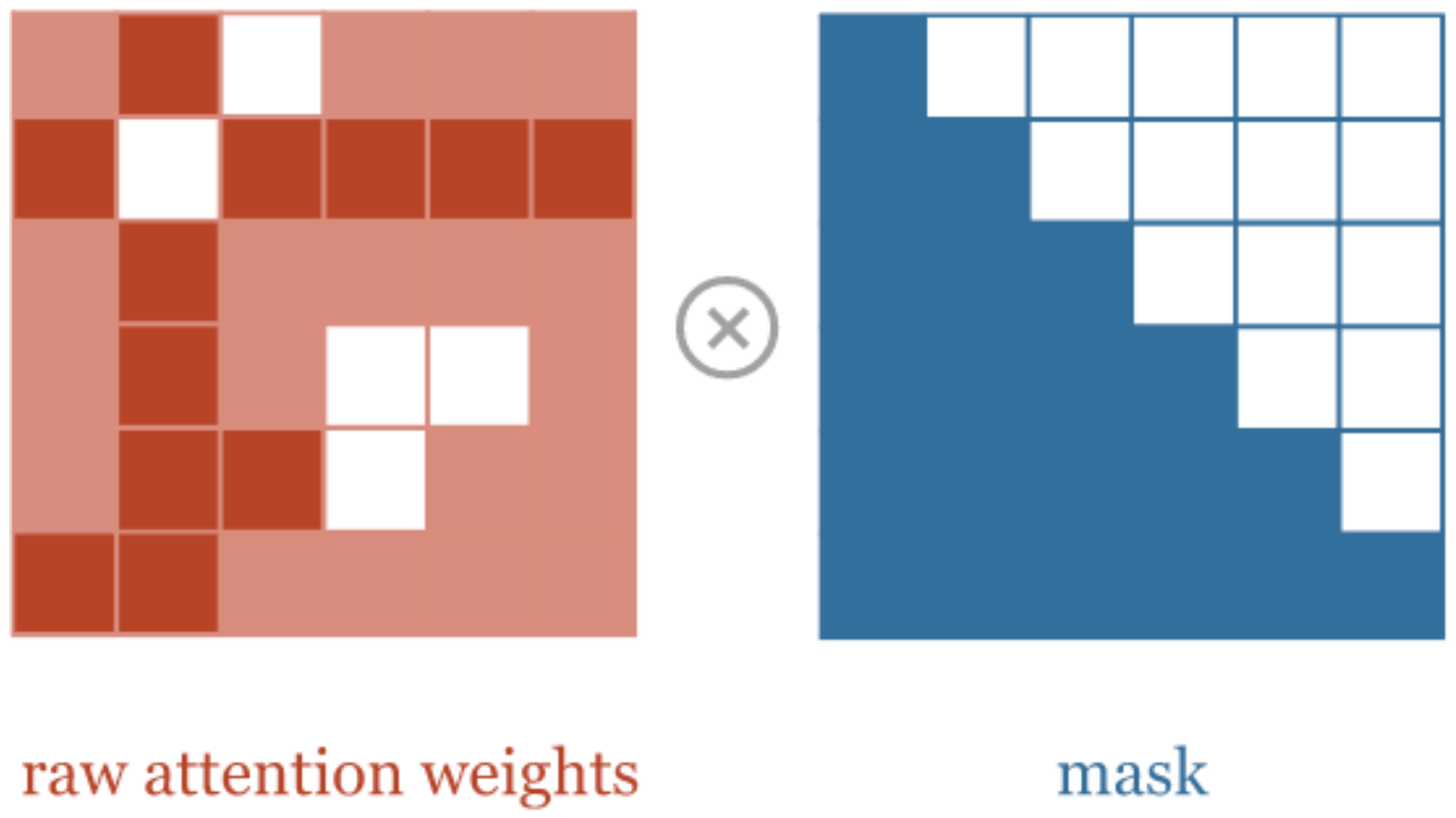
https://jalammar.github.io/illustrated-gpt2/

Masked multi-head self-attention

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q, \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K, \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V$$

$$e_{i,j} = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}, \forall j = 1, \ldots, n$$

$$\alpha_i = \mathrm{softmax}(\mathbf{e}_i)$$



Efficient implementation: compute attention as we normally do, mask out attention to future words by setting attention scores to $-\infty$

```
dot = torch.bmm(queries, keys.transpose(1, 2))

indices = torch.triu_indices(t, t, offset=1)
dot[:, indices[0], indices[1]] = float('-inf')

dot = F.softmax(dot, dim=2)
```

http://peterbloem.nl/blog/transformers

Masked multi-head self-attention

The following matrix denotes the values of $\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$ for $1 \leq i \leq n, 1 \leq j \leq n \ (n = 4)$

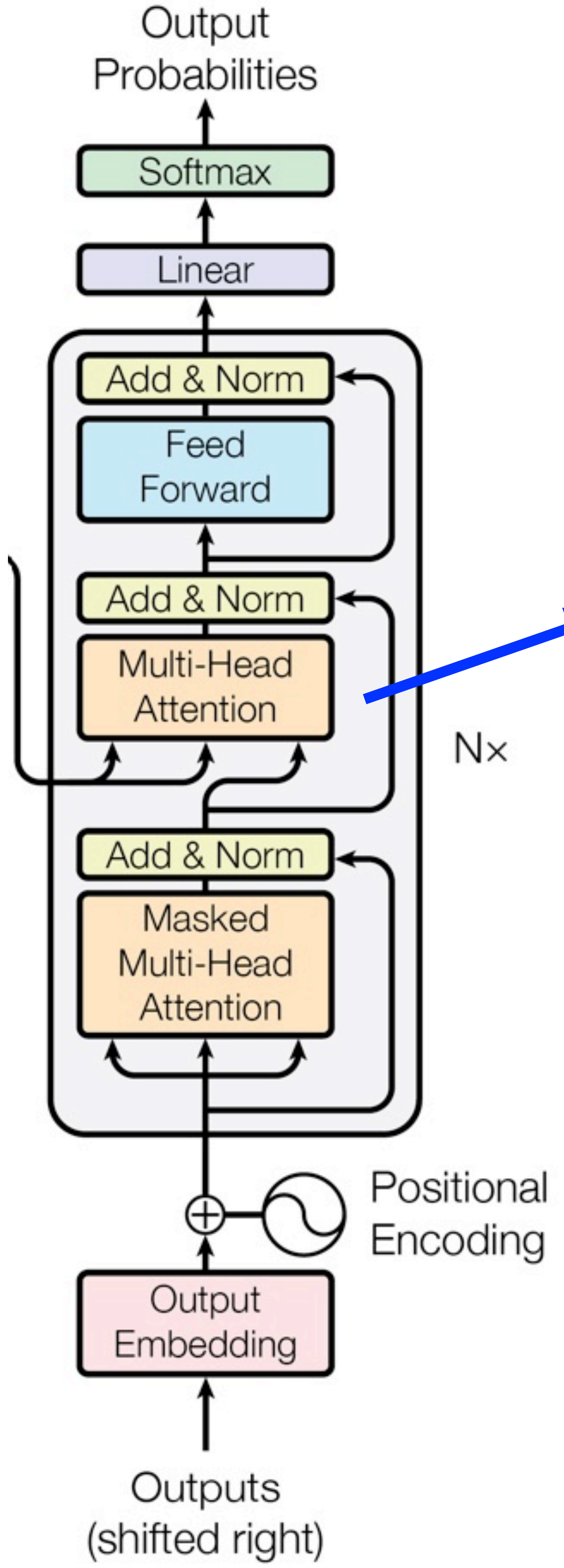
1	0	-1	-1
1	1	-1	0
0	1	1	-1
-1	-1	2	1

What should be the value of $\alpha_{2,2}$ in masked attention?

(A) 0
(B) 0.5
(C) $\frac{e}{2e + e^{-1} + 1}$
(D) 1

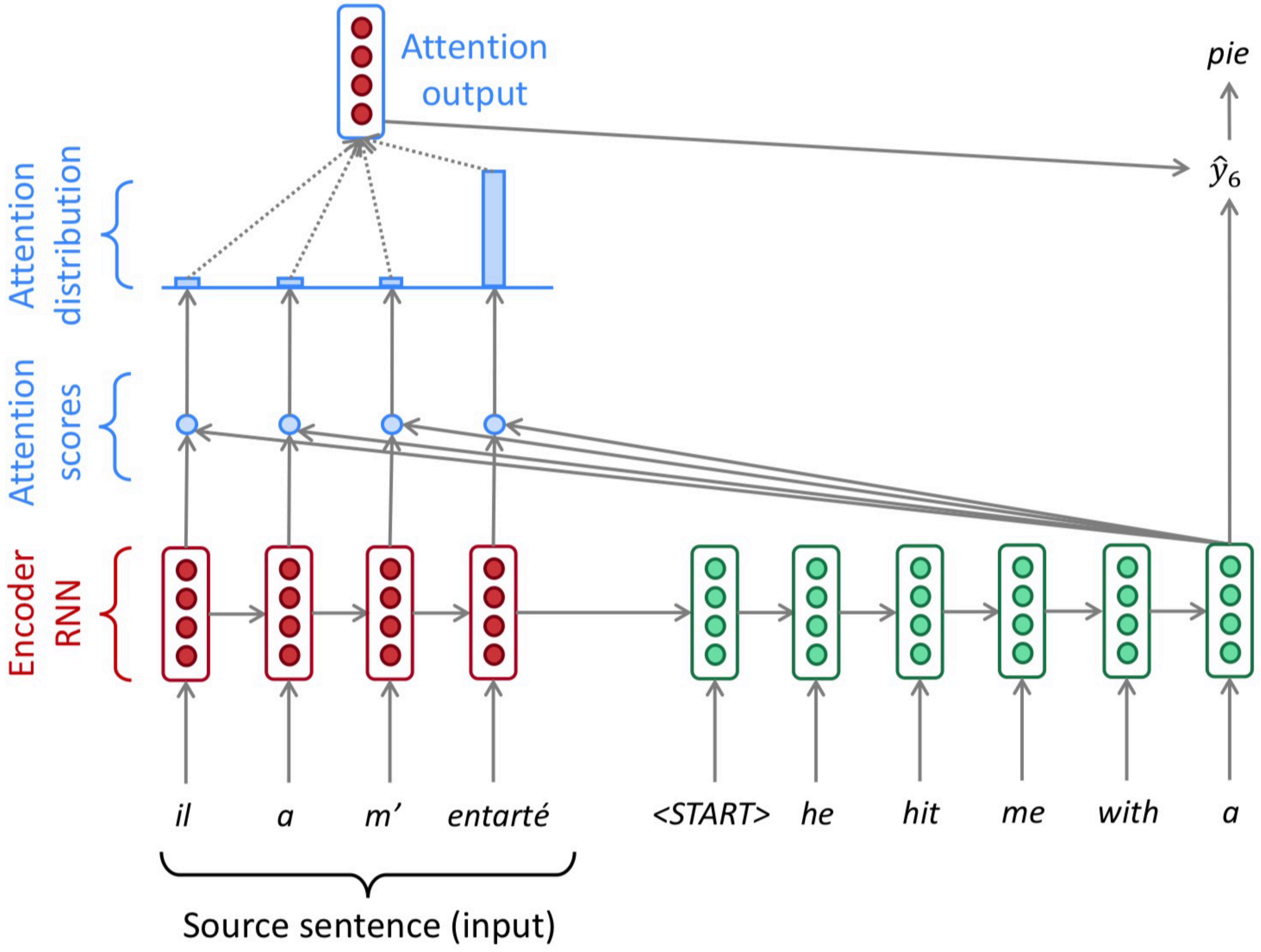
The correct answer is (B)

Multi-head cross-attention



Cross-attention between source and target sequence

Similar as the attention in seq2seq model!



Multi-head cross-attention

Self-attention:

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q, \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K, \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V$$

$$e_{i,j} = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}, \forall j = 1, \ldots, n$$

$$\alpha_i = \text{softmax}(\mathbf{e}_i)$$

$$\mathbf{h}_i = \sum_{j=1}^{n} \alpha_{i,j} \mathbf{v}_j$$

Cross-attention:

(always from the top layer)

$\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_m$: hidden states from encoder

$\mathbf{x}_1, \ldots, \mathbf{x}_n$: hidden states from decoder

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q \quad i = 1,2,\ldots,n$$

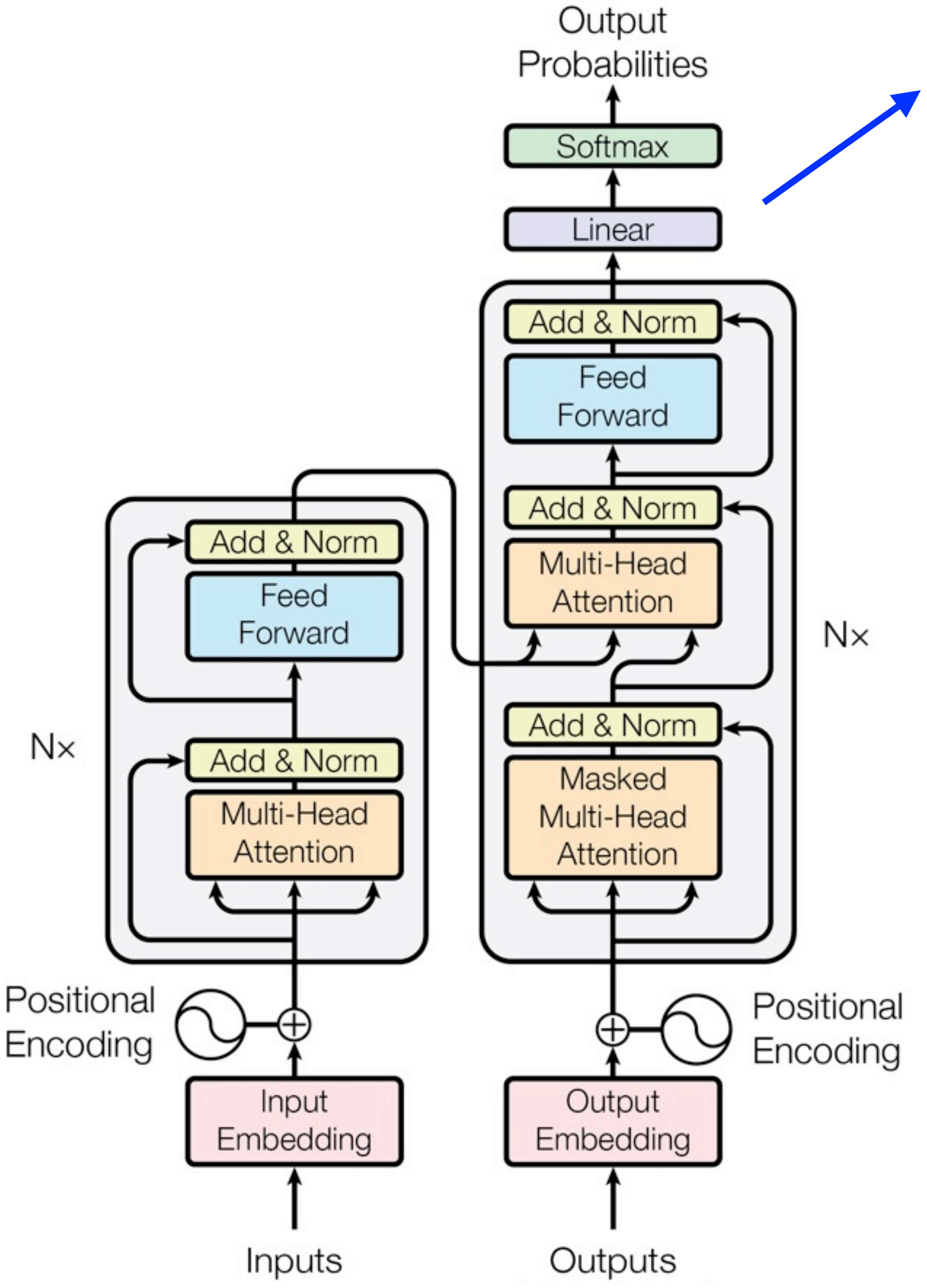
$$\mathbf{k}_j = \tilde{\mathbf{x}}_j \mathbf{W}^K, \mathbf{v}_j = \tilde{\mathbf{x}}_j \mathbf{W}^V \quad \forall j = 1,2,\ldots,m$$

$$e_{i,j} = \frac{\mathbf{q}_\mathbf{i} \cdot \mathbf{k}_j}{\sqrt{d_k}}, \forall j = 1, \ldots, m$$

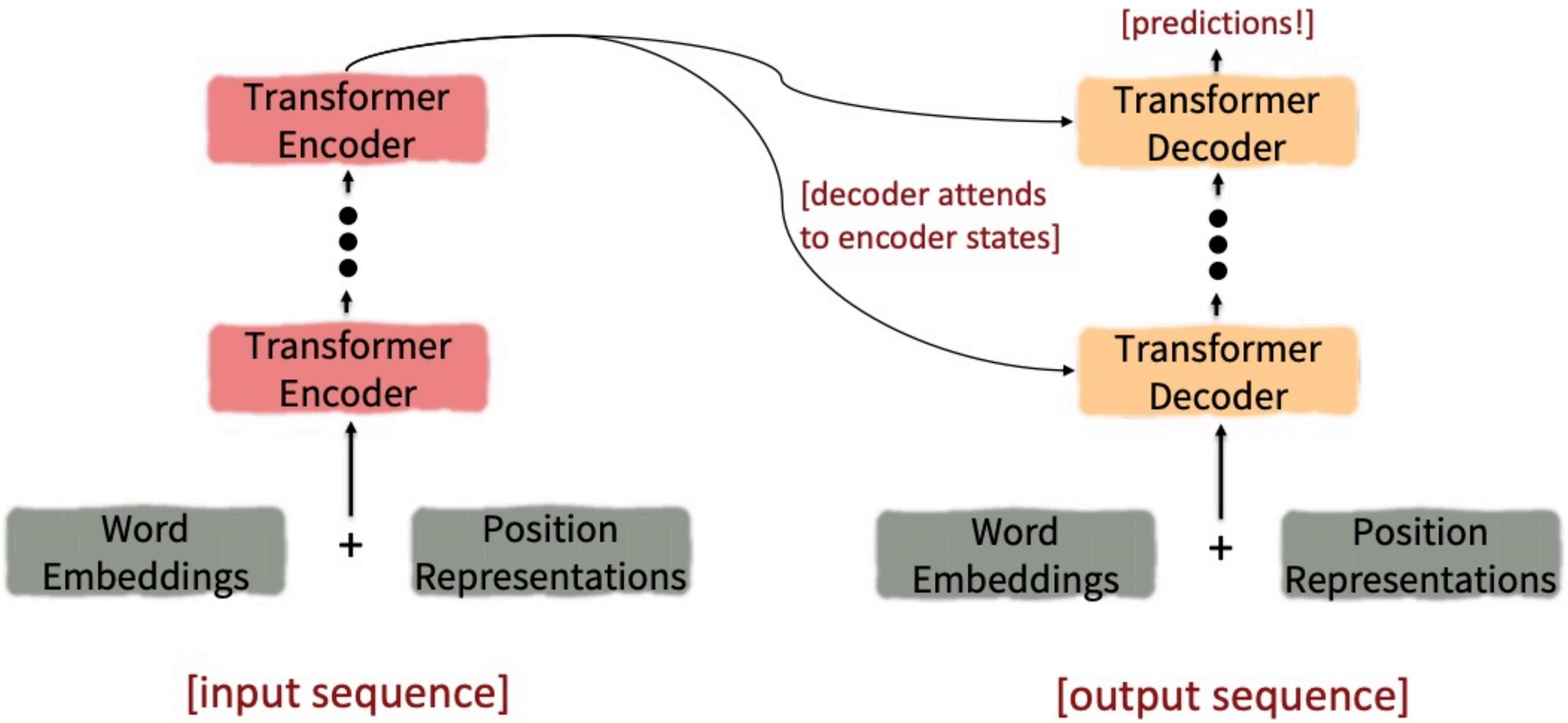
$$\alpha_i = \text{softmax}(\mathbf{e}_i)$$

$$\mathbf{h}_i = \sum_{j=1}^{m} \alpha_{i,j} \mathbf{v}_j$$

Transformer encoder-decoder



$\text{softmax}(\mathbf{W}_o \mathbf{h}_i)$



Training Transformer encoder-decoder models

The same as the way that we train seq2seq models!

- Training data: parallel corpus $\{(\mathbf{w}_i^{(s)}, \mathbf{w}_i^{(t)})\}$
- Minimize cross-entropy loss:

$$\sum_{t=1}^{T} -\log P(y_t \mid y_1, \ldots, y_{t-1}, \mathbf{w}^{(s)})$$

(denote $\mathbf{w}^{(t)} = y_1, \ldots, y_T$)

- Back-propagate gradients through both encoder and decoder

36M sentence pairs

French: bonjour le monde .

English: hello world .

Masked self-attention is the key!

This can enable parallelizable operations while NOT looking at the future

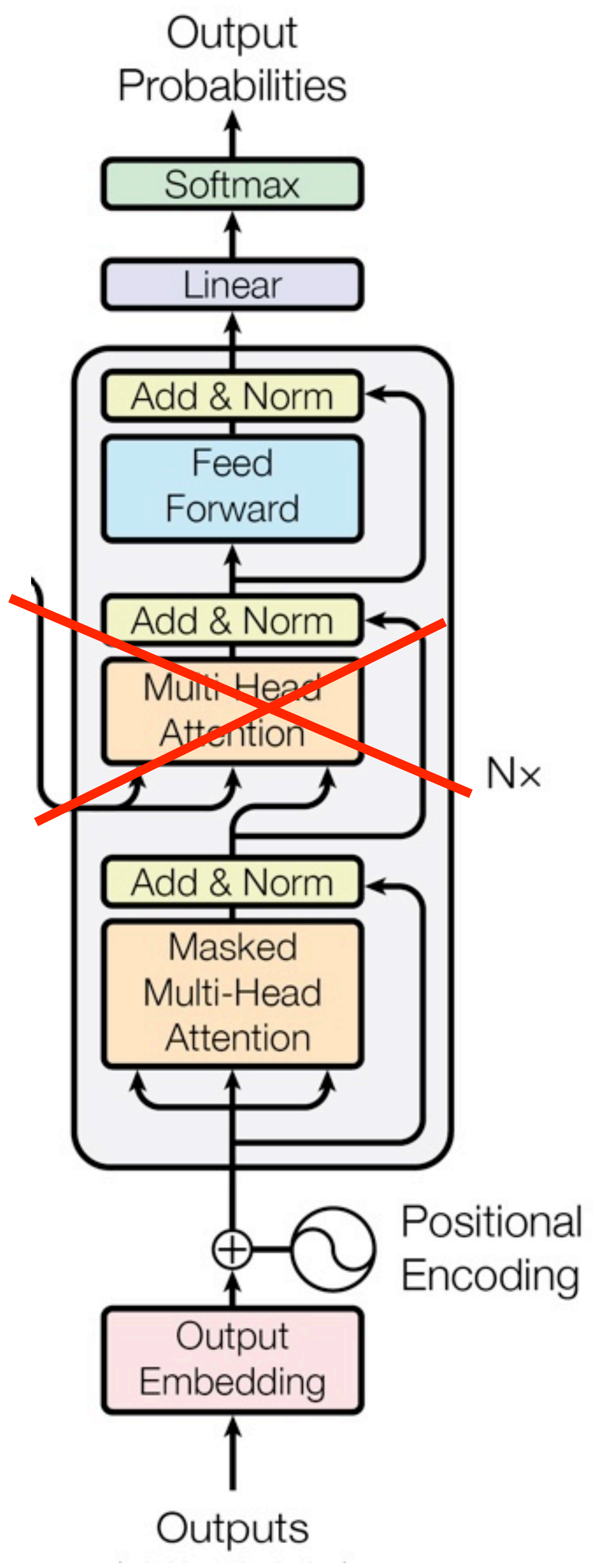
Empirical results with Transformers

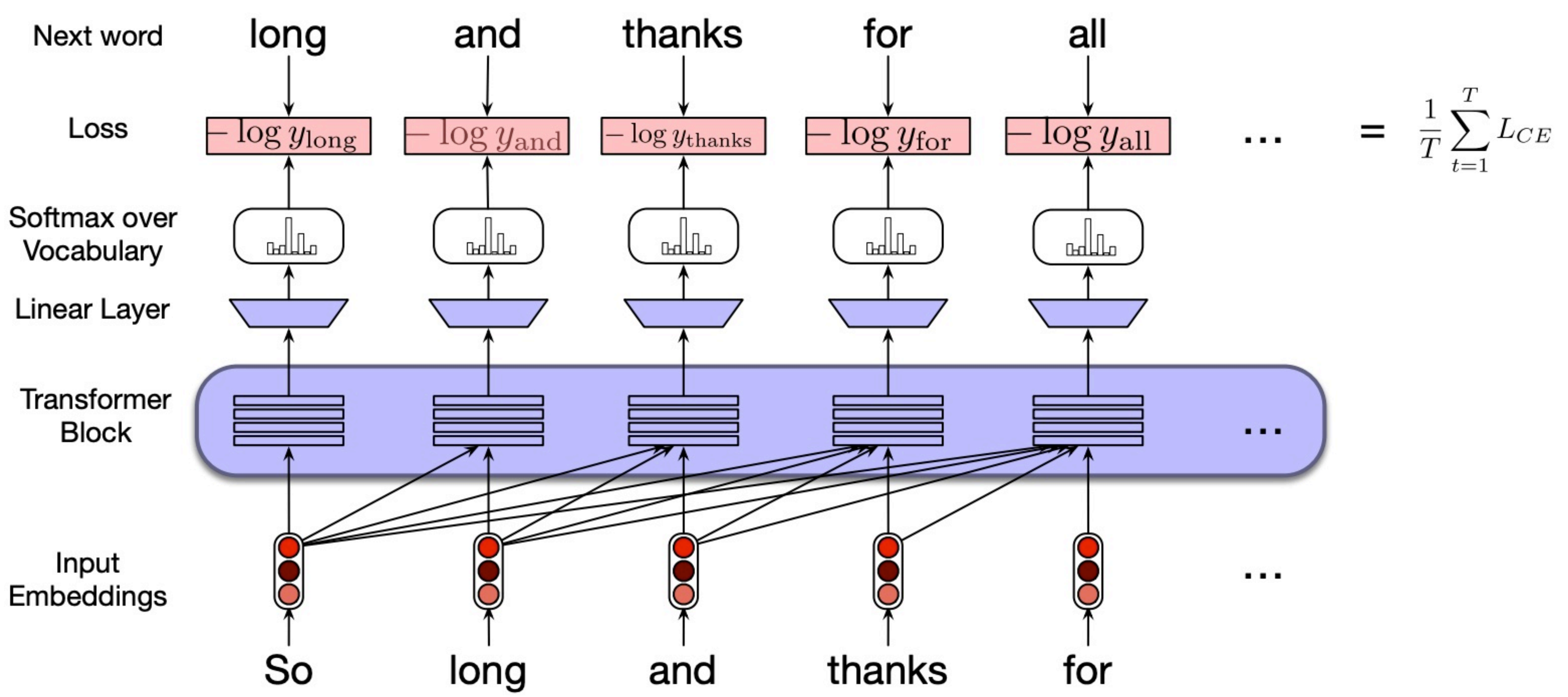
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

(Vaswani et al., 2017)

Transformer-based language models

The backbone of large language models (e.g., GPT/ChatGPT, Gemini, LLaMA, …)





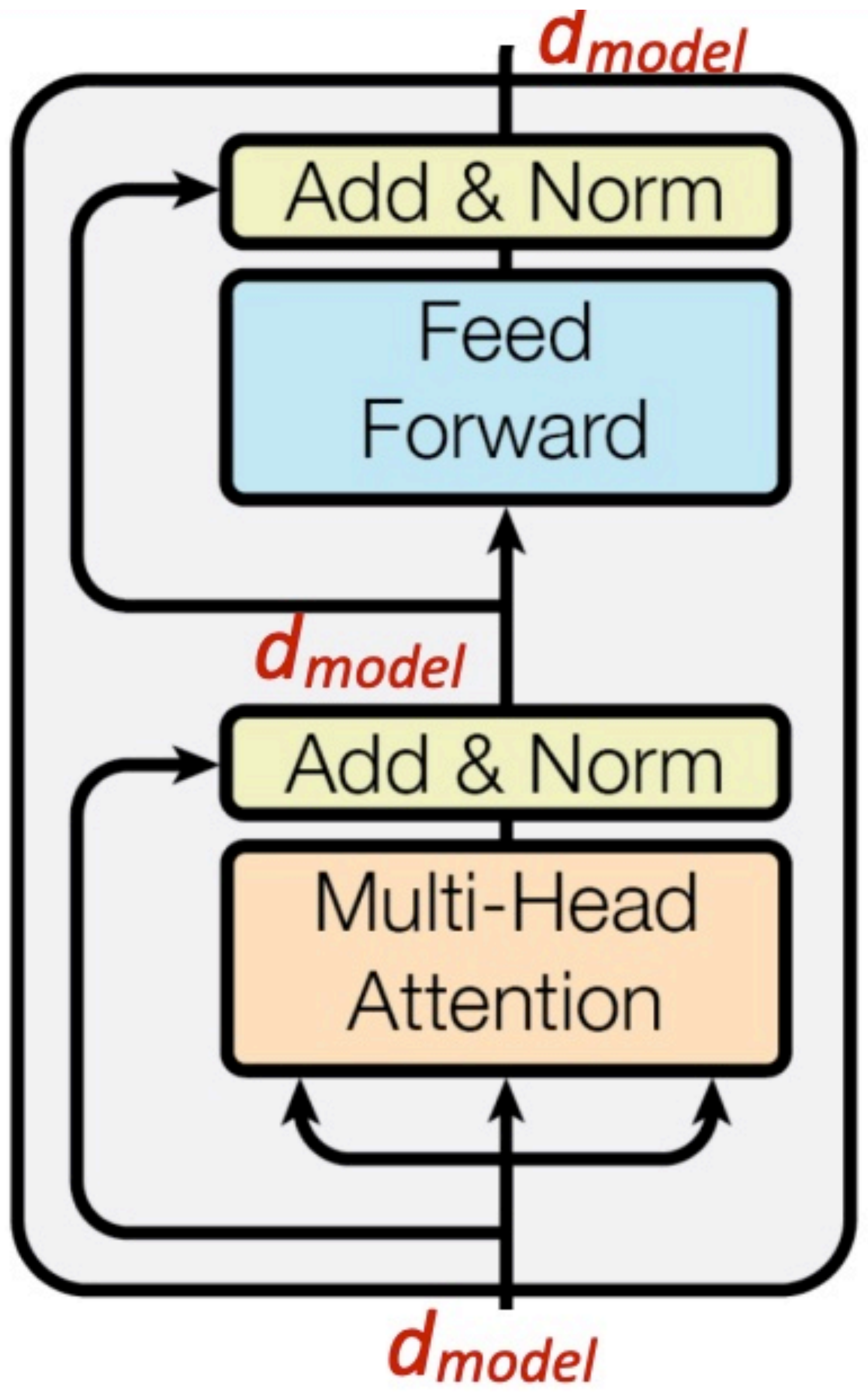
Transformer architecture specifications

	N	d_{model}	d_{ff}	h	d_k	d_v
base	6	512	2048	8	64	64

(Vaswani et al., 2017)

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}
GPT-3 Small	125M	12	768	12	64
GPT-3 Medium	350M	24	1024	16	64
GPT-3 Large	760M	24	1536	16	96
GPT-3 XL	1.3B	24	2048	24	128
GPT-3 2.7B	2.7B	32	2560	32	80
GPT-3 6.7B	6.7B	32	4096	32	128
GPT-3 13B	13.0B	40	5140	40	128
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128

(Brown et al., 2020)



Transformers: pros and cons

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

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

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

- **Are positional embeddings 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 becomes very long

Computational analysis of Transformers

Multi-head attention (MHA)

$$Q = XW^Q, K = XW^K, V = XW^V$$

$$\boxed{\text{Attention}(Q, K, V) = \text{softmax}(\frac{\underbrace{Q}_{n \times d}\underbrace{K^T}_{d \times n}}{\sqrt{d_k}})\underbrace{V}_{n \times d}}$$

$O(nd^2 + n^2d)$

Feed-forward layers (FFN)

$$\text{FFN}(\mathbf{x}_i) = \text{ReLU}(\mathbf{x}_i \underbrace{\mathbf{W}_1}_{d \times d_{ff}} + \mathbf{b}_1)\underbrace{\mathbf{W_2}}_{d_{ff} \times d} + \mathbf{b}_2$$

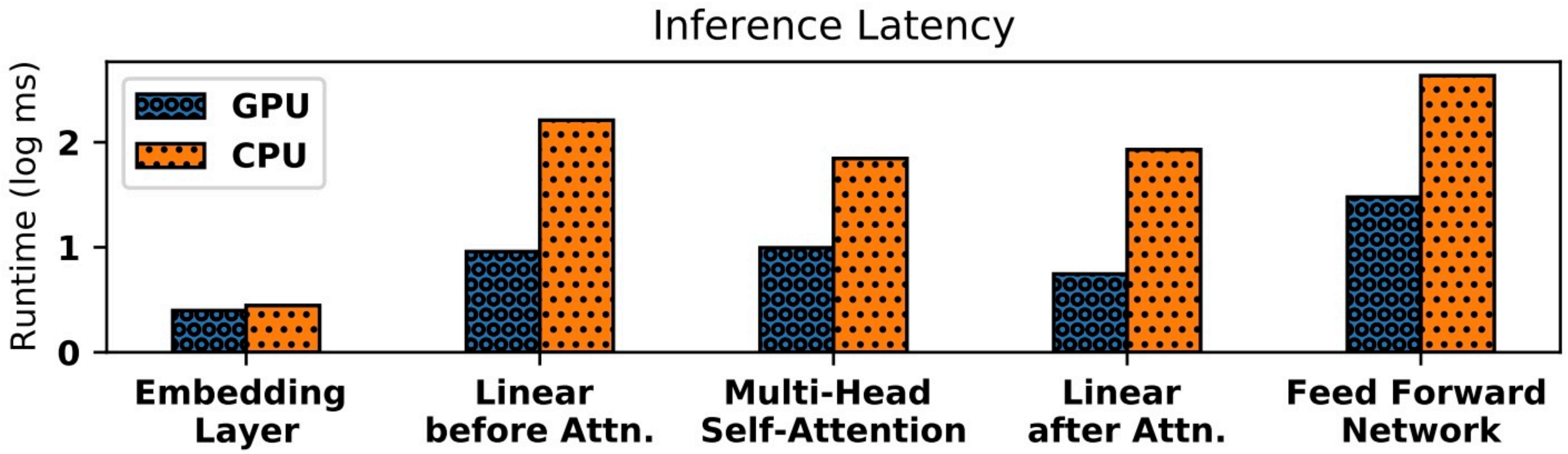
$O(nd^2)$

Note: RNNs only require $O(nd^2)$ time: $\mathbf{h}_t = f(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$

(assuming input dimension = hidden dimension = d)

Computational analysis of Transformers

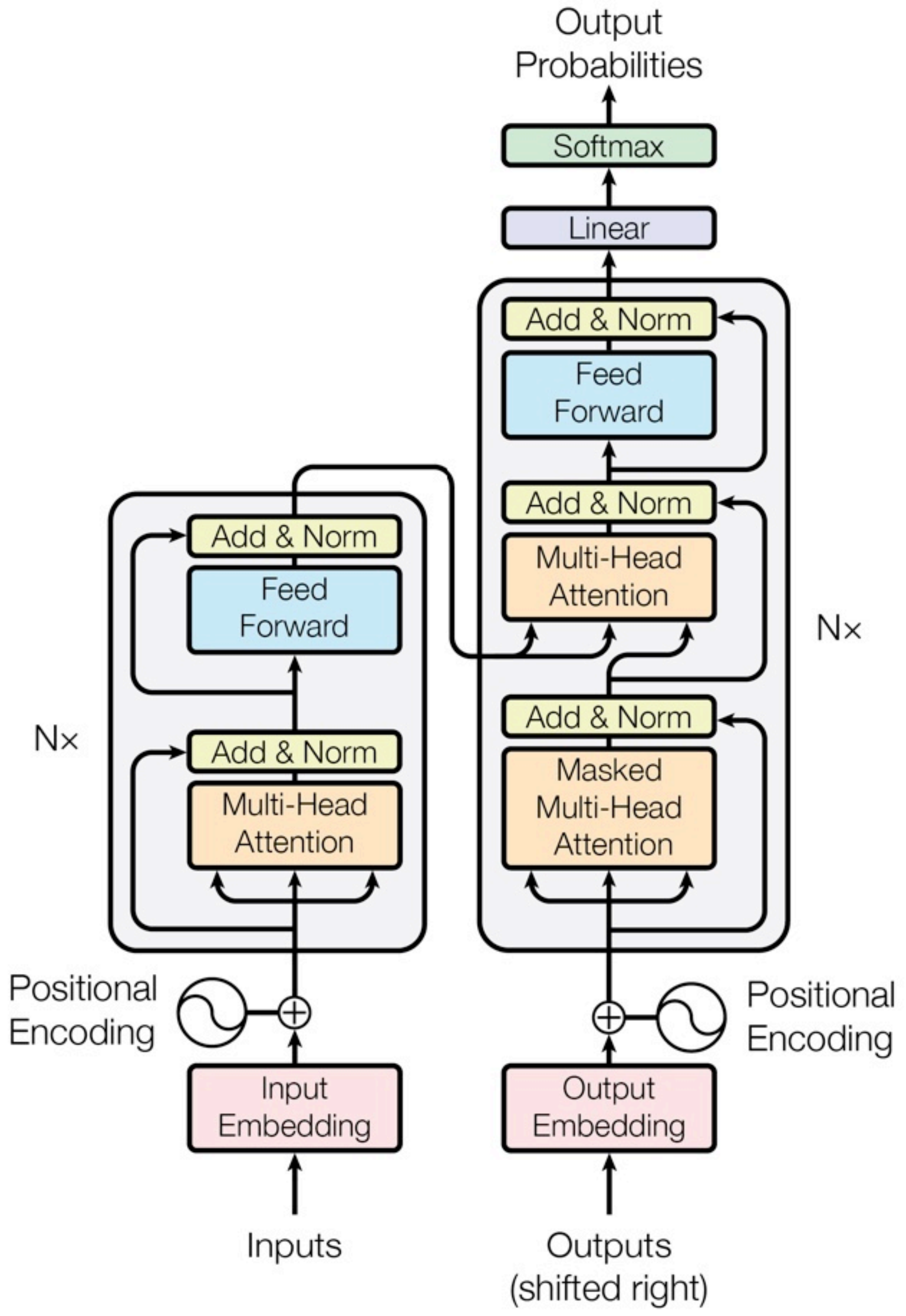
- For BERT-sized models ($n = 512, d = 768, d_{ff} = 4d$), 2/3 of parameters are FFNs.



(Ganesh et al., 2020)

- However, when sequence length becomes longer (e.g., > 50,000), the computation will be dominated by self-attention $O(n^2d)$
  - Numerous solutions have been proposed to address this issue
  - Long-context language modeling is still one of the most active research areas today

Transformers: roadmap



- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

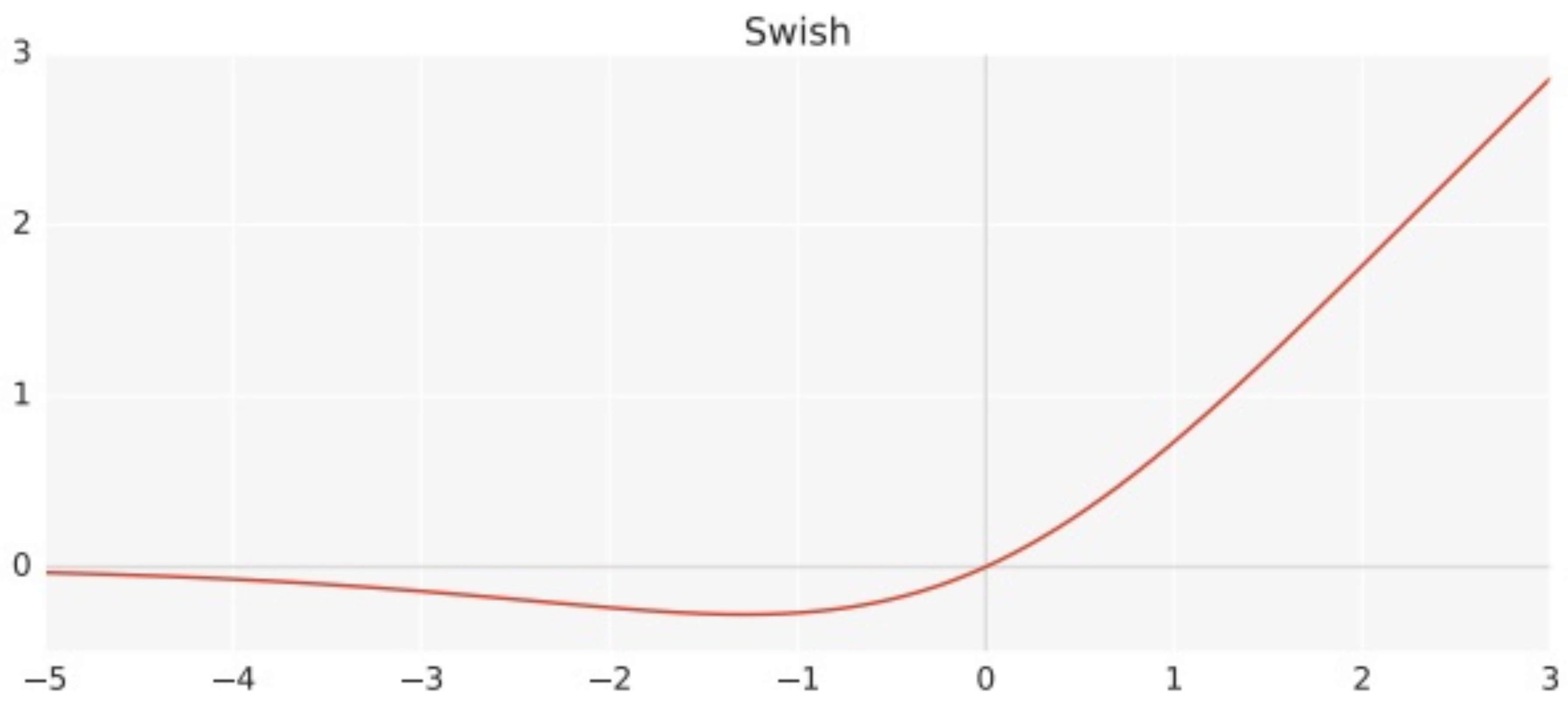
- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers

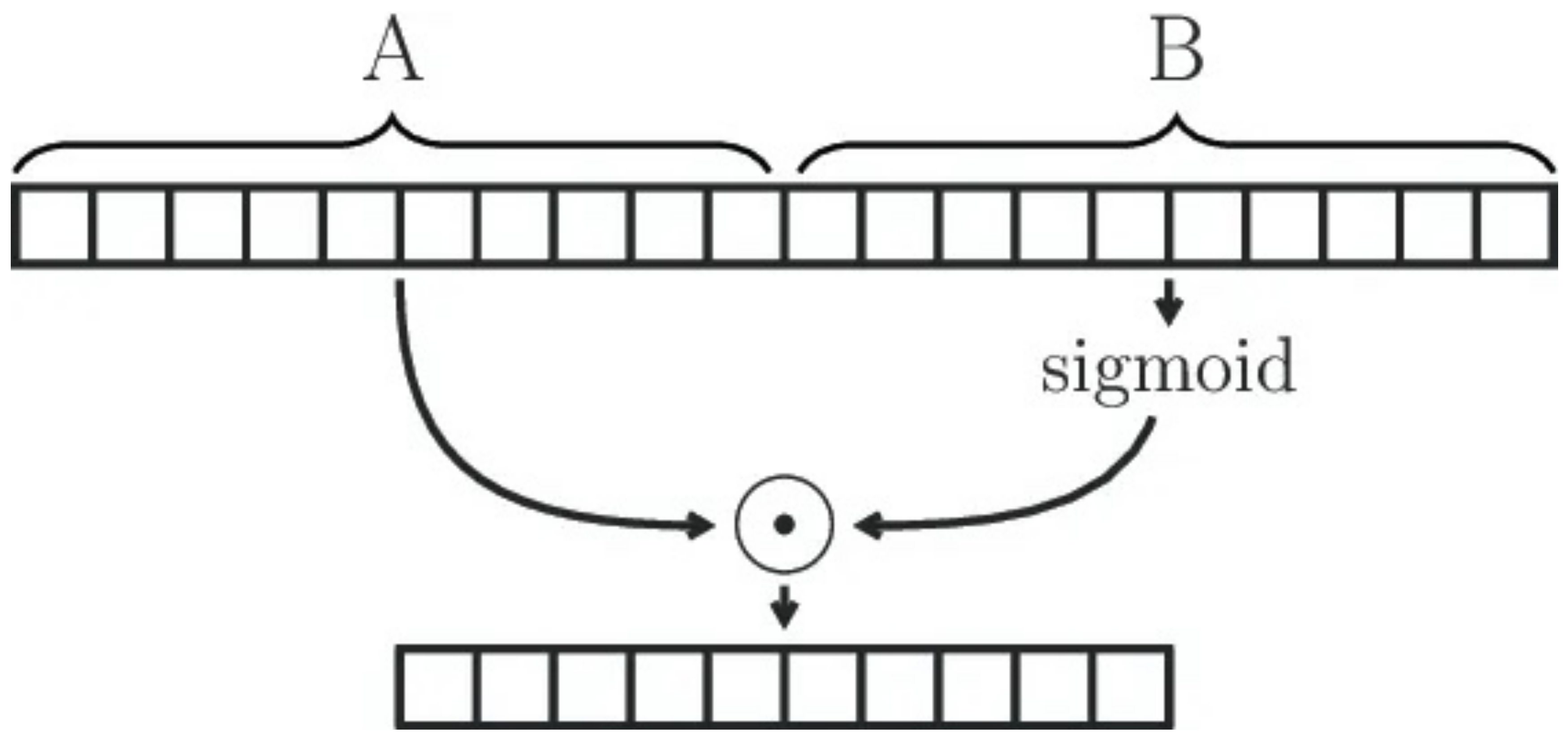
Major modifications since original Transformers

SwiGLU activation

SwiGLU = Swish + GLU

$$\mathrm{SwiGLU}(x, W, V, b, c, \beta) = \mathrm{Swish}_\beta(xW + b) \otimes (xV + c)$$





Swish(x) = $x \cdot \mathrm{sigmoid}(\beta x)$

https://azizbelaweid.substack.com/p/what-is-swiglu-how-to-implement-it

(Shazeer et al., 2020): GLU Variants Improve Transformer

SwiGLU activation

	Score Average	CoLA MCC	SST-2 Acc	MRPC F1	MRPC Acc	STSB PCC	STSB SCC	QQP F1	QQP Acc	MNLIm Acc	MNLImm Acc	QNLI Acc	RTE Acc
FFN_{ReLU}	83.80	51.32	94.04	**93.08**	**90.20**	89.64	89.42	89.01	91.75	85.83	86.42	92.81	80.14
FFN_{GELU}	83.86	53.48	94.04	92.81	**90.20**	89.69	89.49	88.63	91.62	85.89	86.13	92.39	80.51
$\text{FFN}_{\text{Swish}}$	83.60	49.79	93.69	92.31	89.46	89.20	88.98	88.84	91.67	85.22	85.02	92.33	81.23
FFN_{GLU}	84.20	49.16	94.27	92.39	89.46	89.46	89.35	88.79	91.62	86.36	86.18	92.92	**84.12**
$\text{FFN}_{\text{GEGLU}}$	84.12	53.65	93.92	92.68	89.71	90.26	90.13	89.11	91.85	86.15	86.17	92.81	79.42
$\text{FFN}_{\text{Bilinear}}$	83.79	51.02	**94.38**	92.28	89.46	90.06	89.84	88.95	91.69	**86.90**	**87.08**	92.92	81.95
$\text{FFN}_{\text{SwiGLU}}$	84.36	51.59	93.92	92.23	88.97	**90.32**	**90.13**	**89.14**	**91.87**	86.45	86.47	**92.93**	83.39
$\text{FFN}_{\text{ReGLU}}$	**84.67**	**56.16**	**94.38**	92.06	89.22	89.97	89.85	88.86	91.72	86.20	86.40	92.68	81.59
[Raffel et al., 2019]	83.28	53.84	92.68	92.07	88.92	88.02	87.94	88.67	91.56	84.24	84.57	90.48	76.28
ibid. stddev.	0.235	1.111	0.569	0.729	1.019	0.374	0.418	0.108	0.070	0.291	0.231	0.361	1.393

$$\text{SwiGLU}(x, W, V, b, c, \beta) = \text{Swish}_\beta(xW + b) \otimes (xV + c)$$

Notes: there are 3 projection matrices (up_project, down_project, gate_project), d_{ff} is reduced to $4d \times \frac{2}{3}$

Pre-normalization

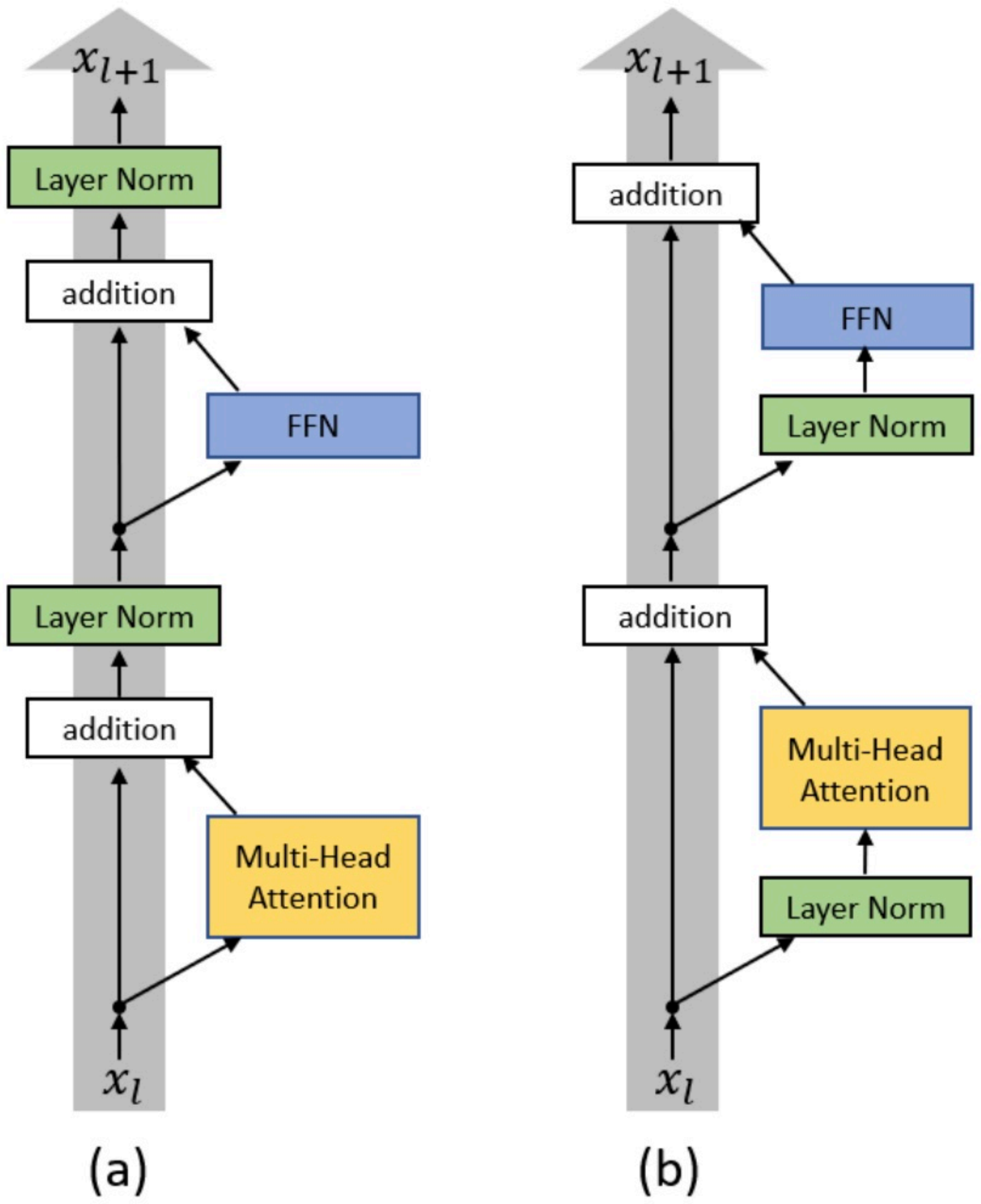


Image: (Xiong et al., 2020)

RMSNorm normalization function

$$\bar{a}_i = \frac{a_i}{\text{RMS}(\mathbf{a})} g_i, \quad \text{where } \text{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}.$$

(Zhang and Senrich, 2019)

Rotary positional embeddings

- **Relative positional embeddings** (T5 uses this!):

Self-Attention with Relative Position Representations

Peter Shaw
Google
petershaw@google.com

Jakob Uszkoreit
Google Brain
usz@google.com

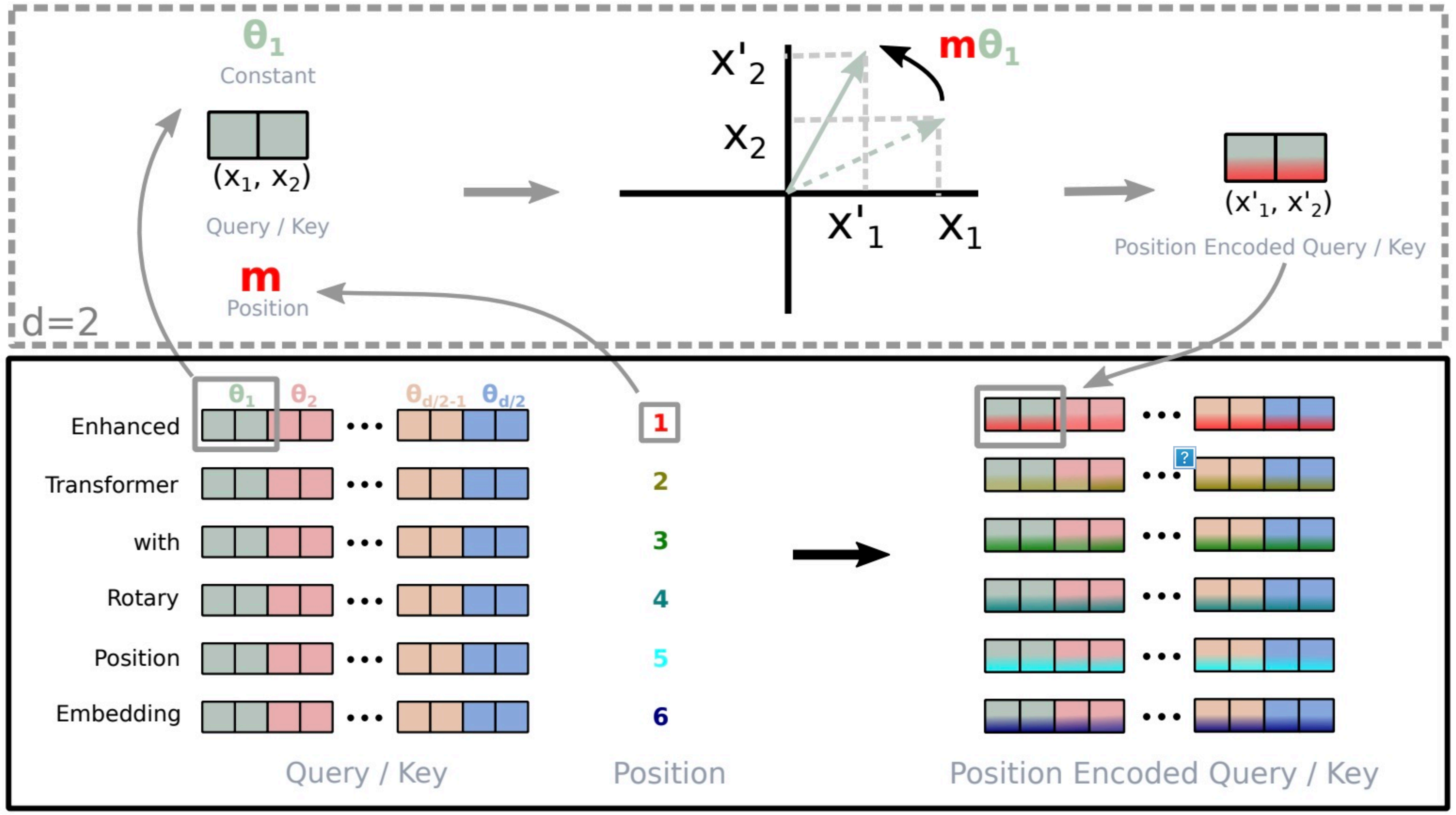
Ashish Vaswani
Google Brain
avaswani@google.com

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$

- Instead of focusing on absolute positions, relative positional embeddings concentrate on the **distances between pairs of tokens**
- Incorporating this relative positional information into attention directly

(Shaw et al., 2018) Self-Attention with Relative Position Representations

Rotary positional embeddings

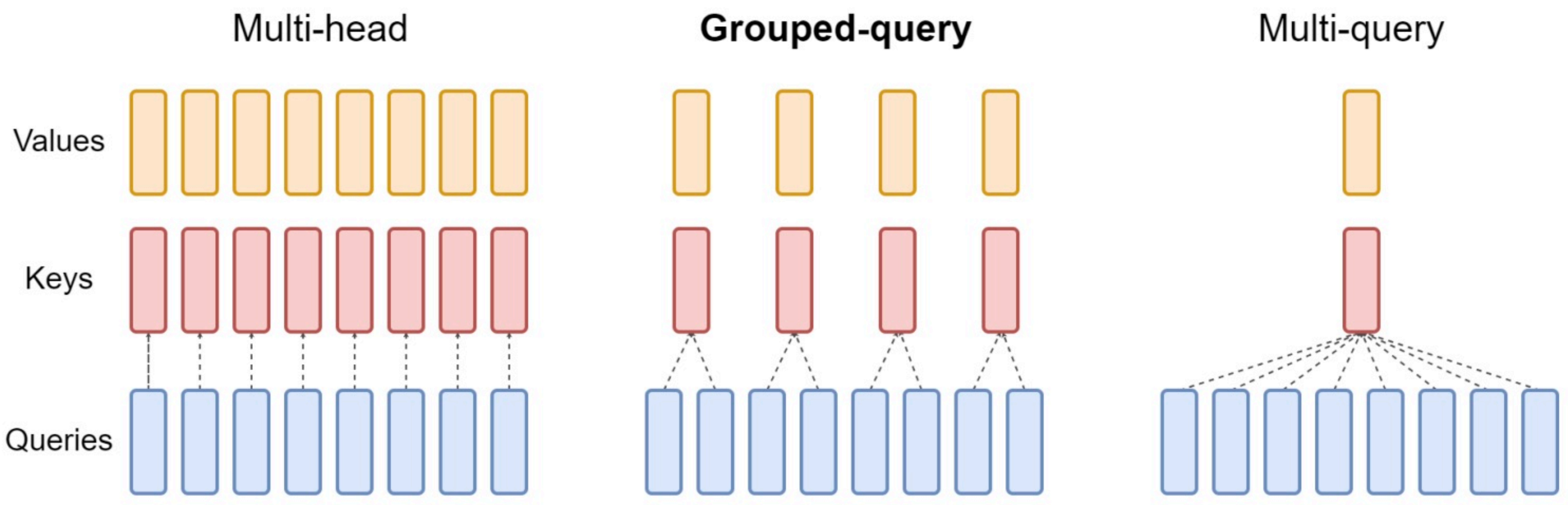


Unites both **absolute** and **relative** positional information

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

(Su et al., 2021) RoFormer: Enhanced Transformer with Rotary Position Embedding

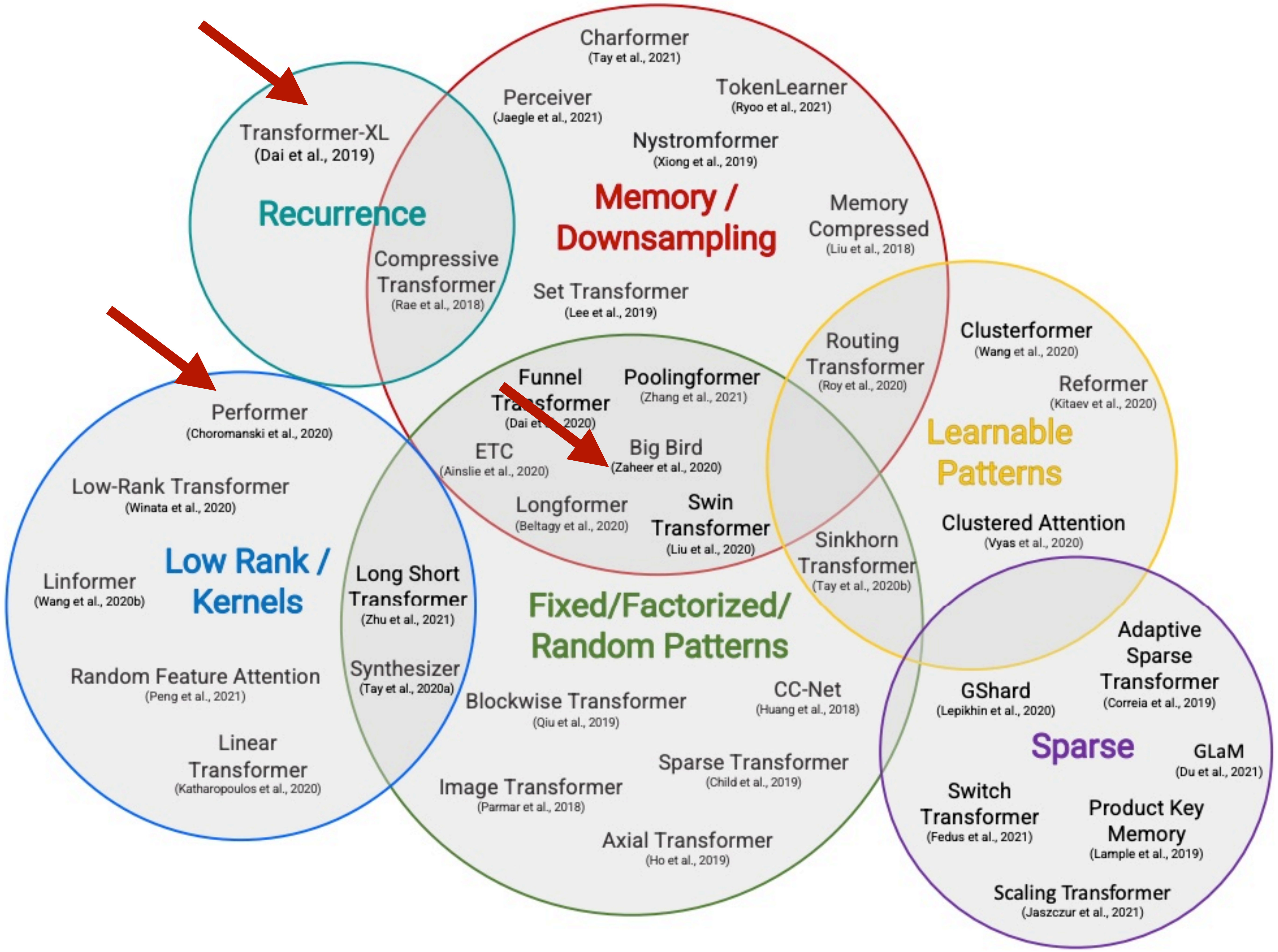
Grouped query attention (GQA)



(Ainslie et al., 2023) GQA: Training generalized multi-query transformer models from multi-head checkpoints.

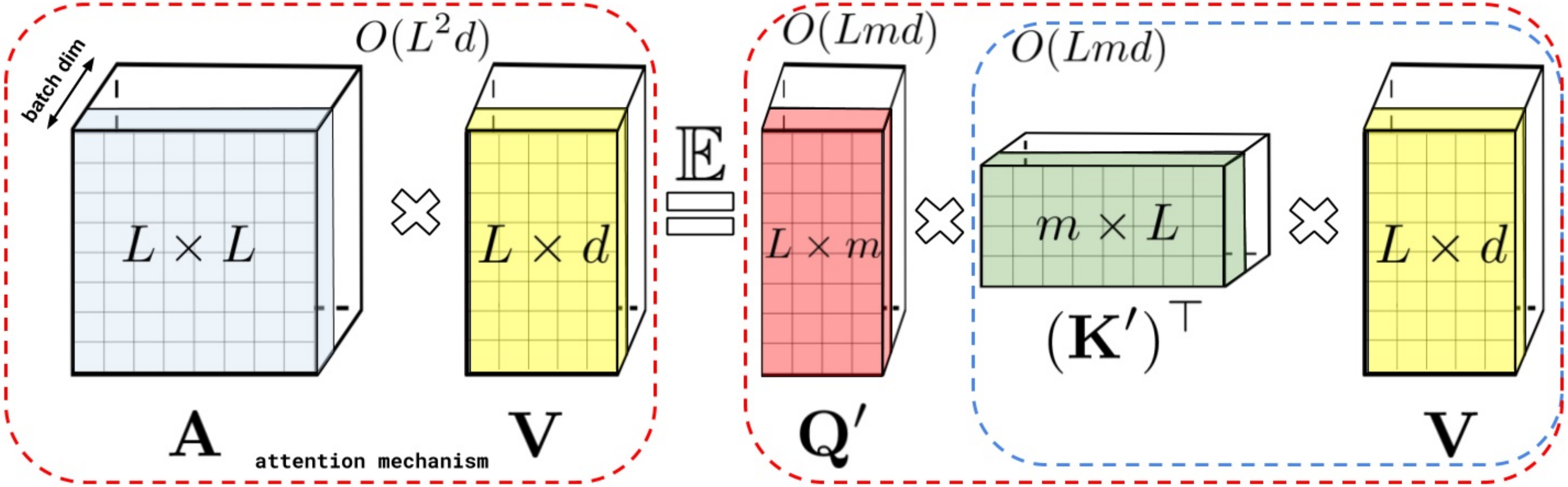
Architecture exploration beyond Transformers

Efficient Transformers



(Tay et al., 2020): Efficient Transformers: A Survey

Example: Performers



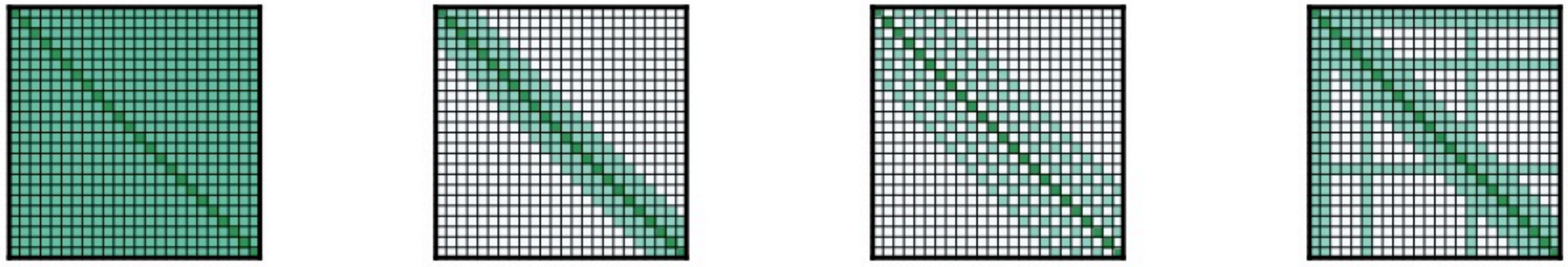
L: sequence length, m << *L*

Low-rank decomposition: Decompose A as the product of Q' and K' (random projection of origional keys and querys)

(Choromanski et al., 2020): Rethinking Attention with Performers

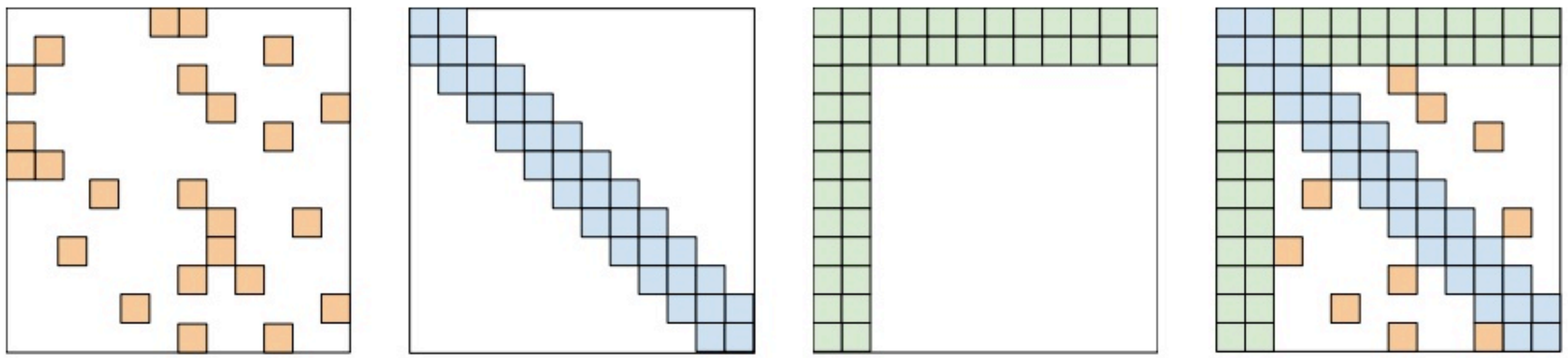
Example: Longformer / Big Bird

Sparse attention: only compute attention at particular positions



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

(Beltagy et al., 2020): Longformer: The Long-Document Transformer

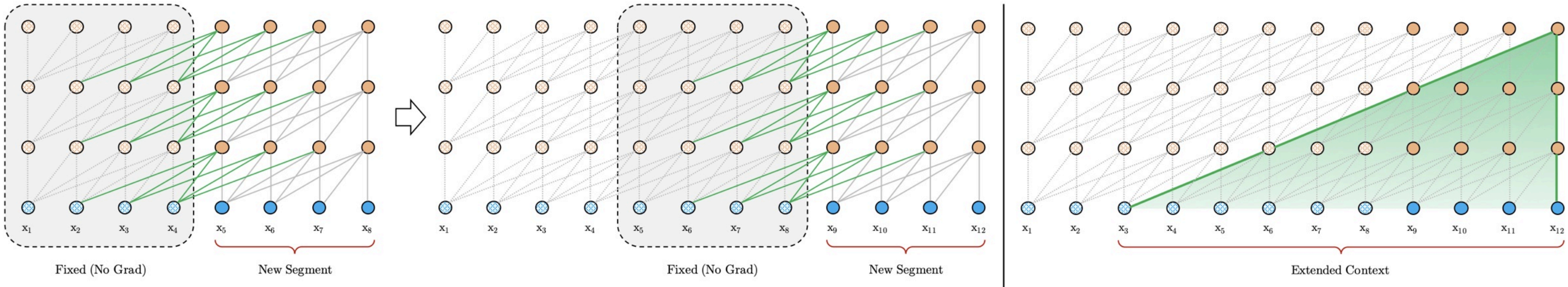


(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

(Zaheer et al., 2021): Big Bird: Transformers for Longer Sequences

Example: Transformer-XL

Segment-level recurrence with state reuse: hidden representations from previous segment will be cached as extended context (no back-propagation to those!)

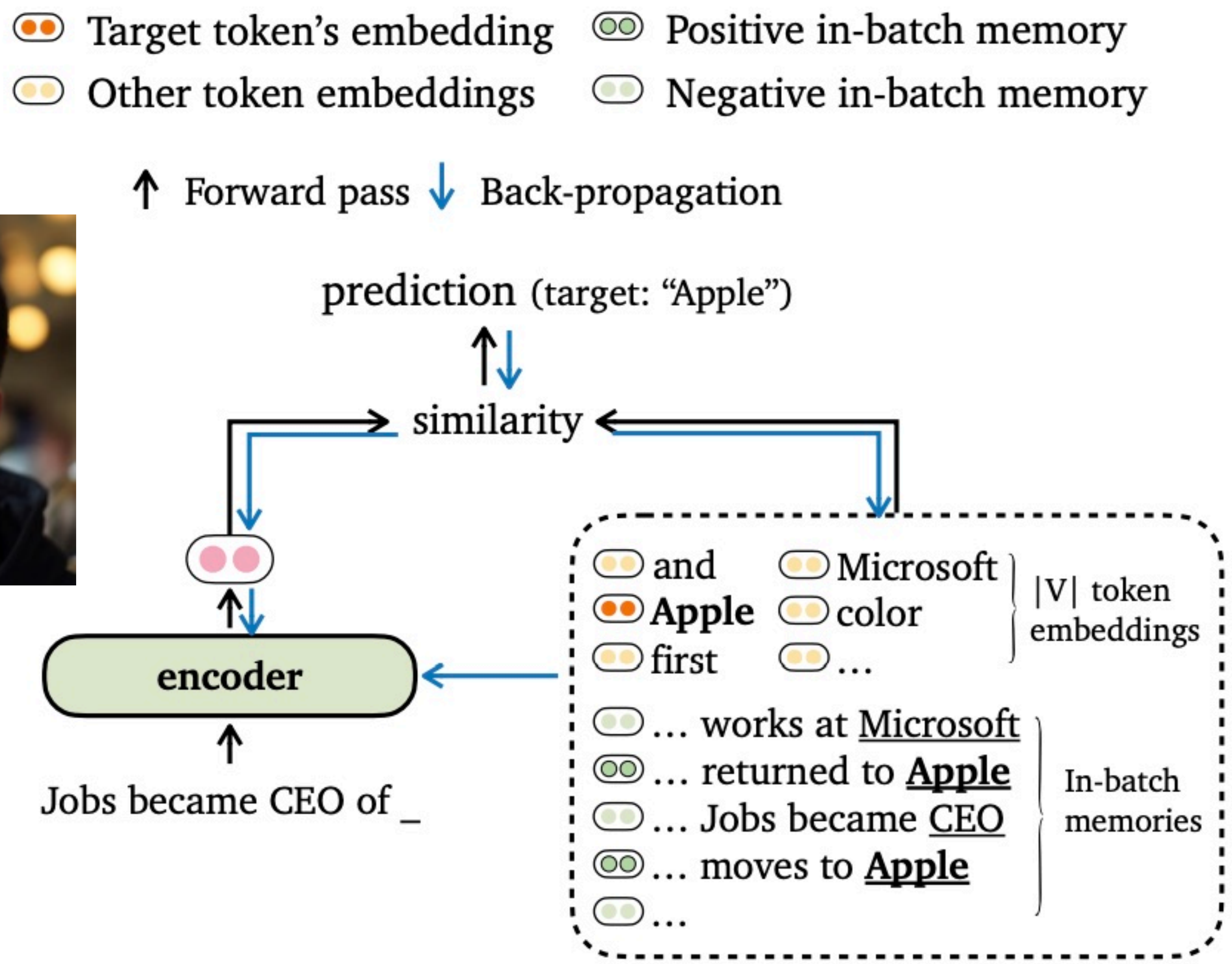


(a) Training phase. (b) Evaluation phase.

(Dai et al., 2019): Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context

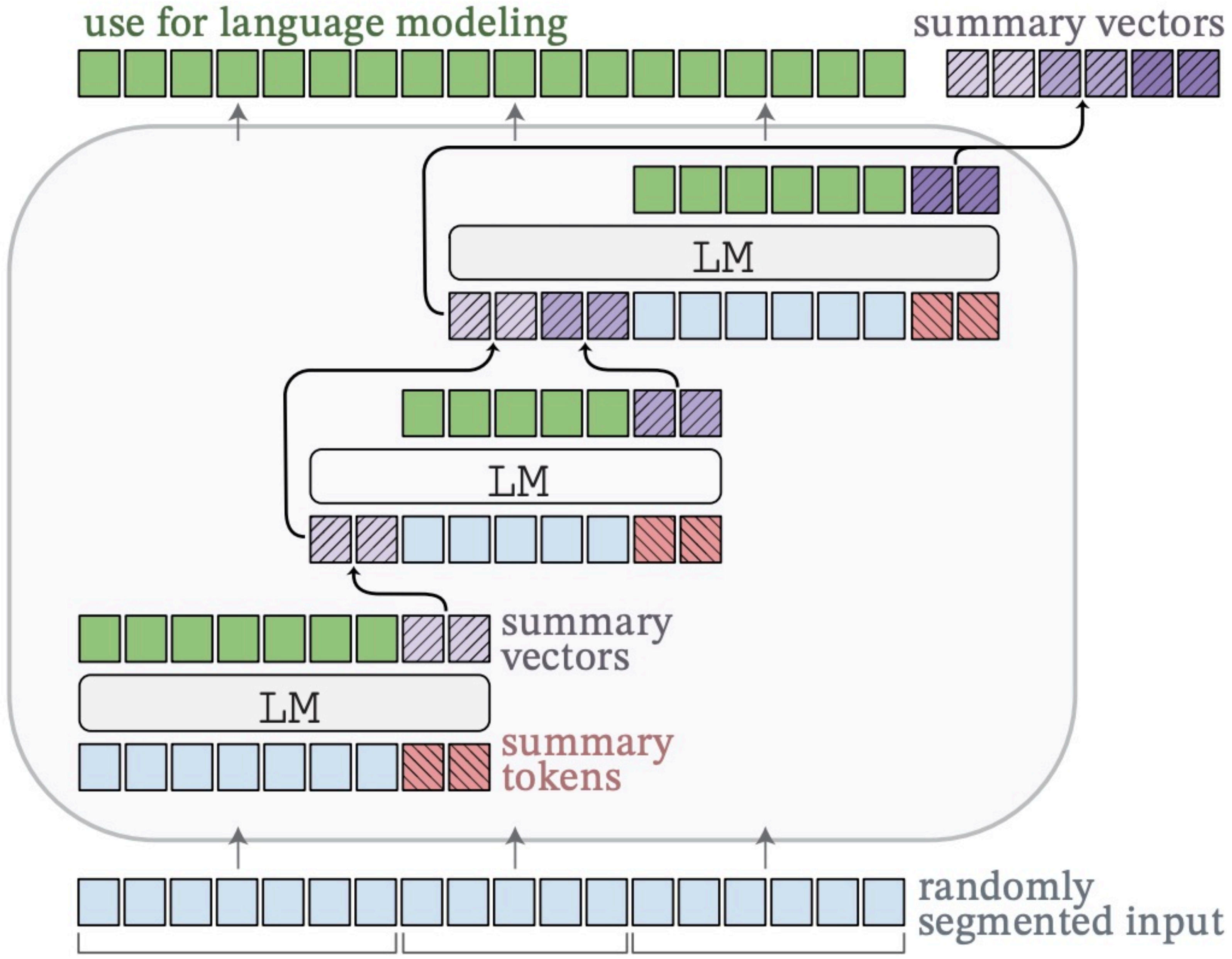
Research from my group

TRIME



(Zhong et al., EMNLP'22) Training Language Models with Memory Augmentation

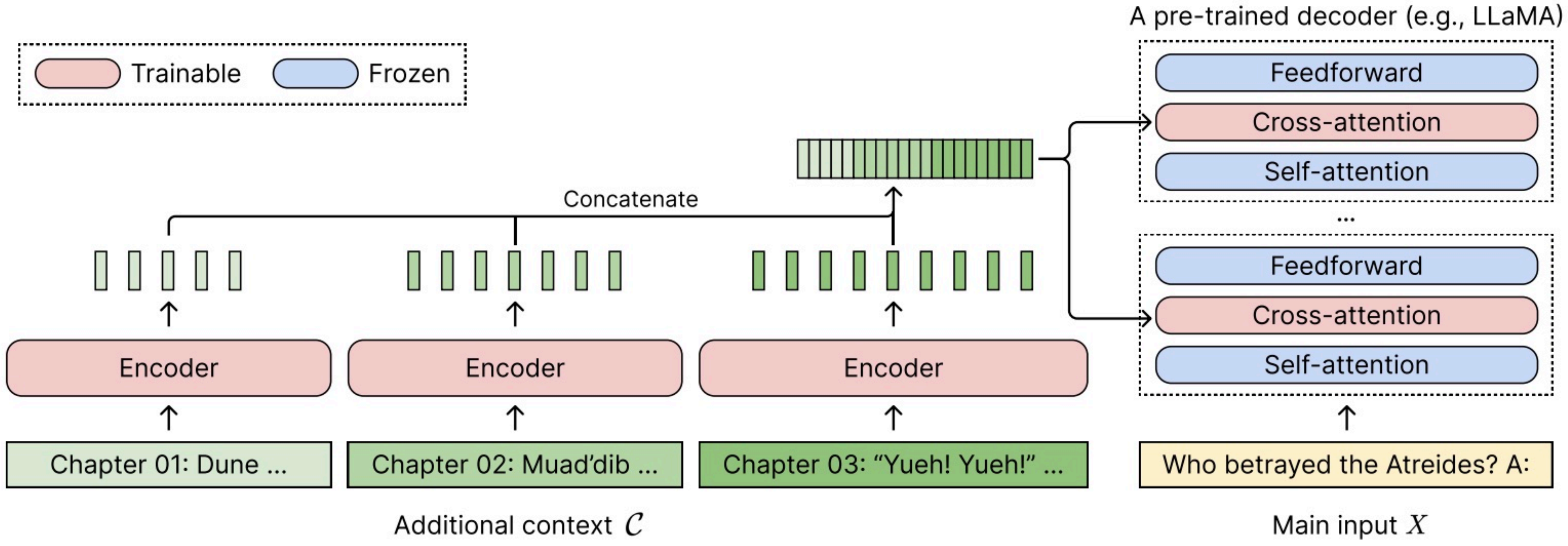
AutoCompressors

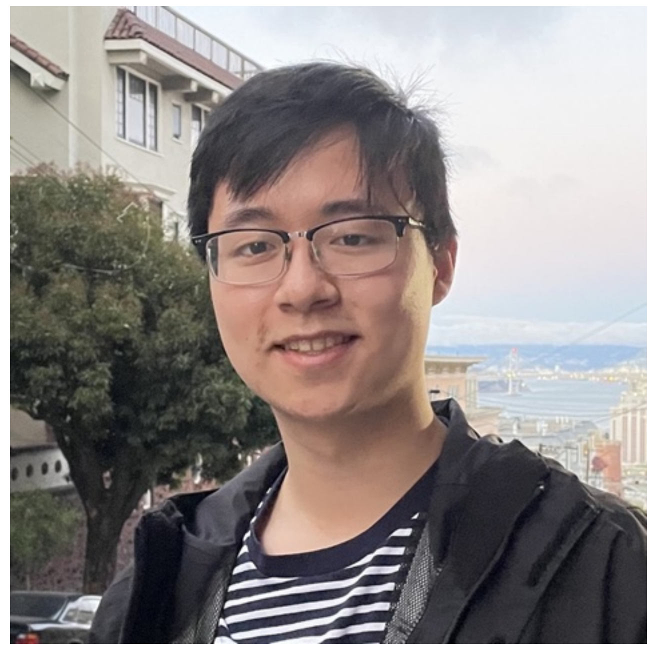


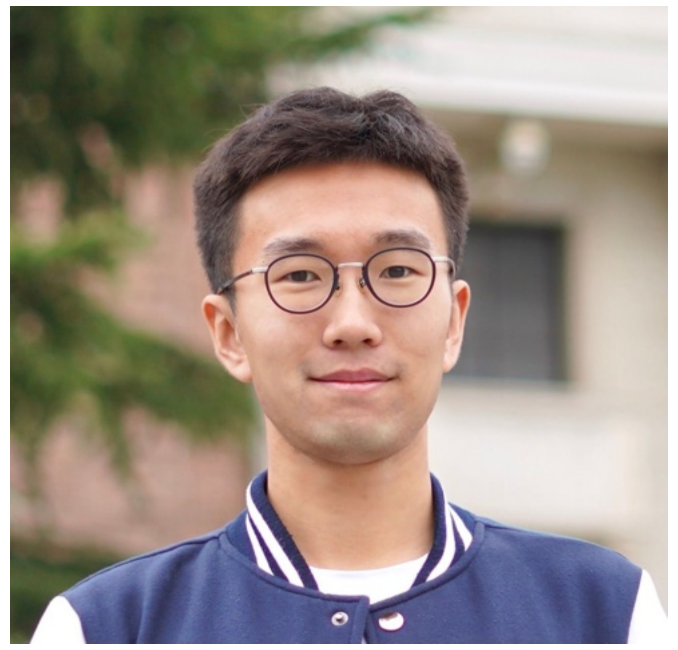
(Chevalier et al., EMNLP'23) Adapting Language Models to Compress Contexts

Research from my group

CEPE





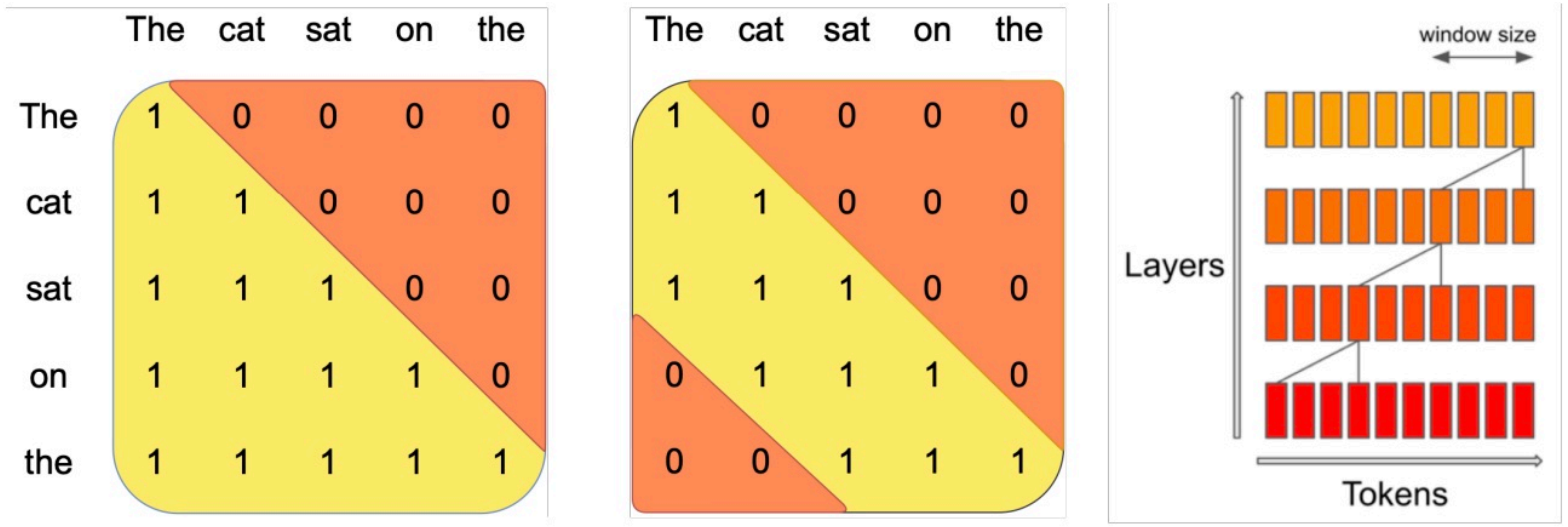


These architectures/techniques are generally applicable to both **long-context modeling** and **retrieval augmentation**!

(Yen et al., ACL'24) Long-Context Language Modeling with Parallel Context Encoding

Remarks on efficient Transformers

- A lot of exploration around 2019-2021: mostly approximation solutions of replacing the full quadratic attention. Few techniques have been adopted in state-of-the-art LLMs (exception: Mistral uses *Sliding Window Attention* to handle longer sequences).



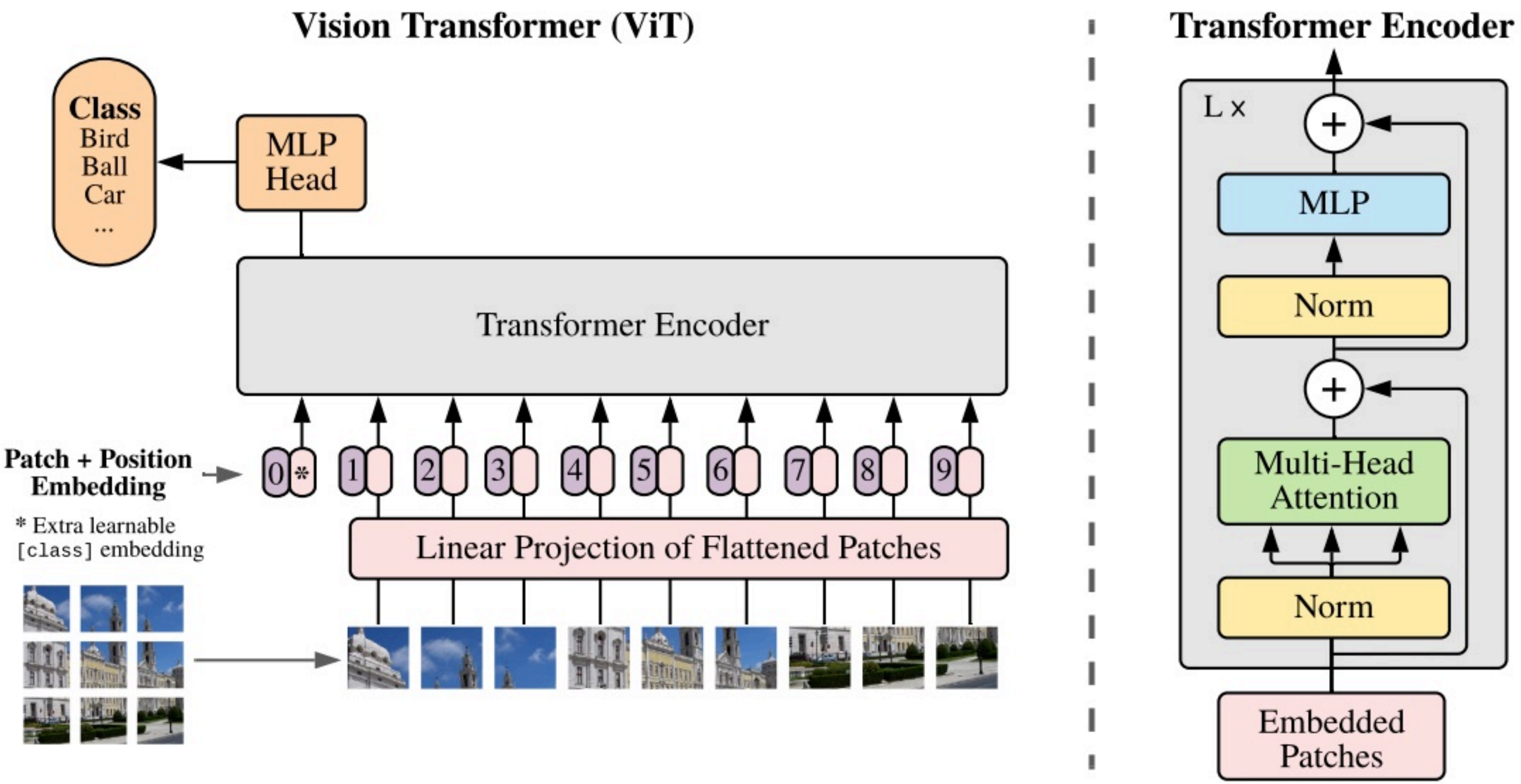
Vanilla Attention   Sliding Window Attention   Effective Context Length

(Jiang et al., 2023)

You can still do such approximations to speed up inference though!

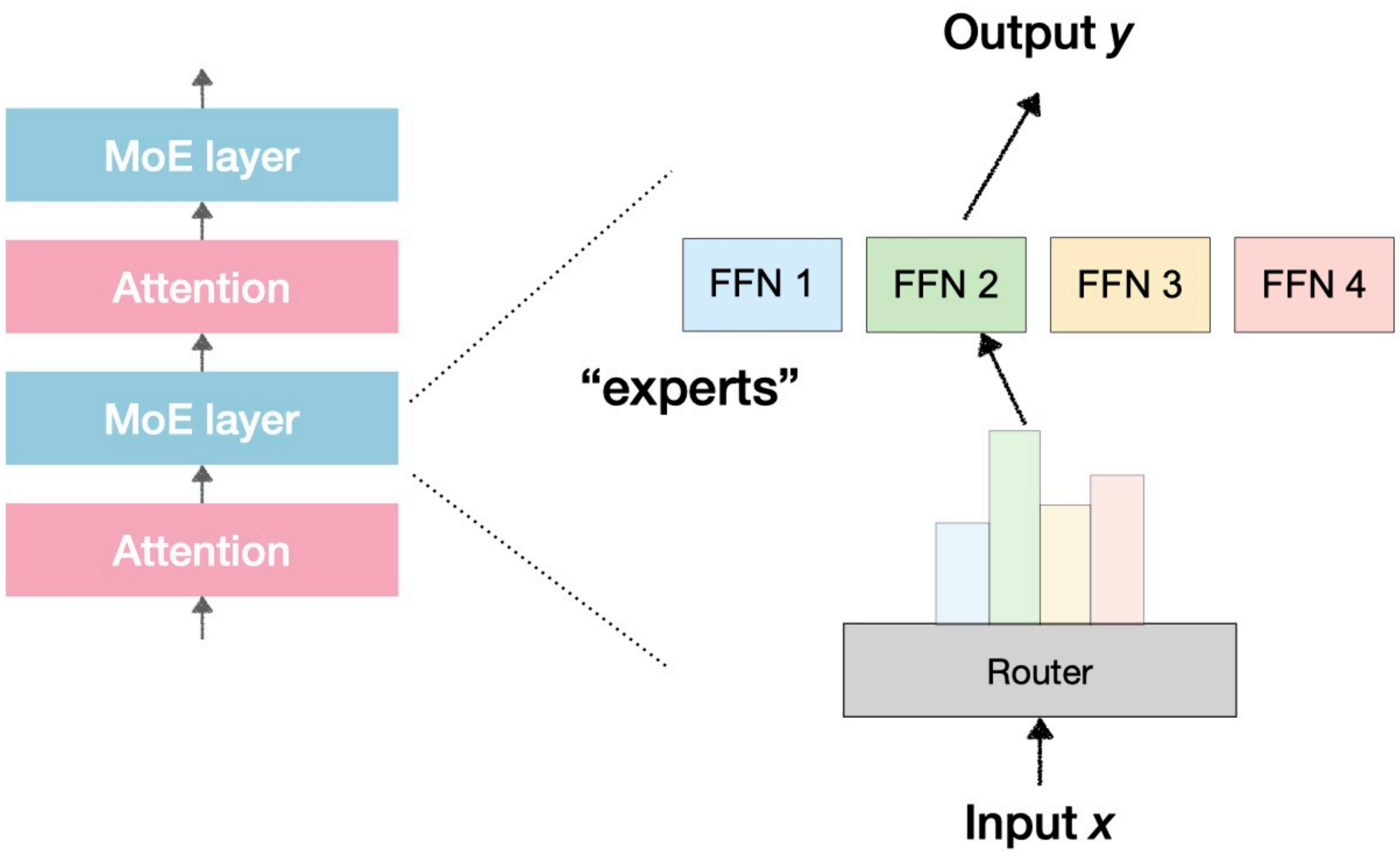
- On the other hand, many system-level advancements have been made to scale up Transformers to longer sequences **without approximation** e.g., FlashAttention (Dao et al., 2022)

Vision Transformer (ViT)

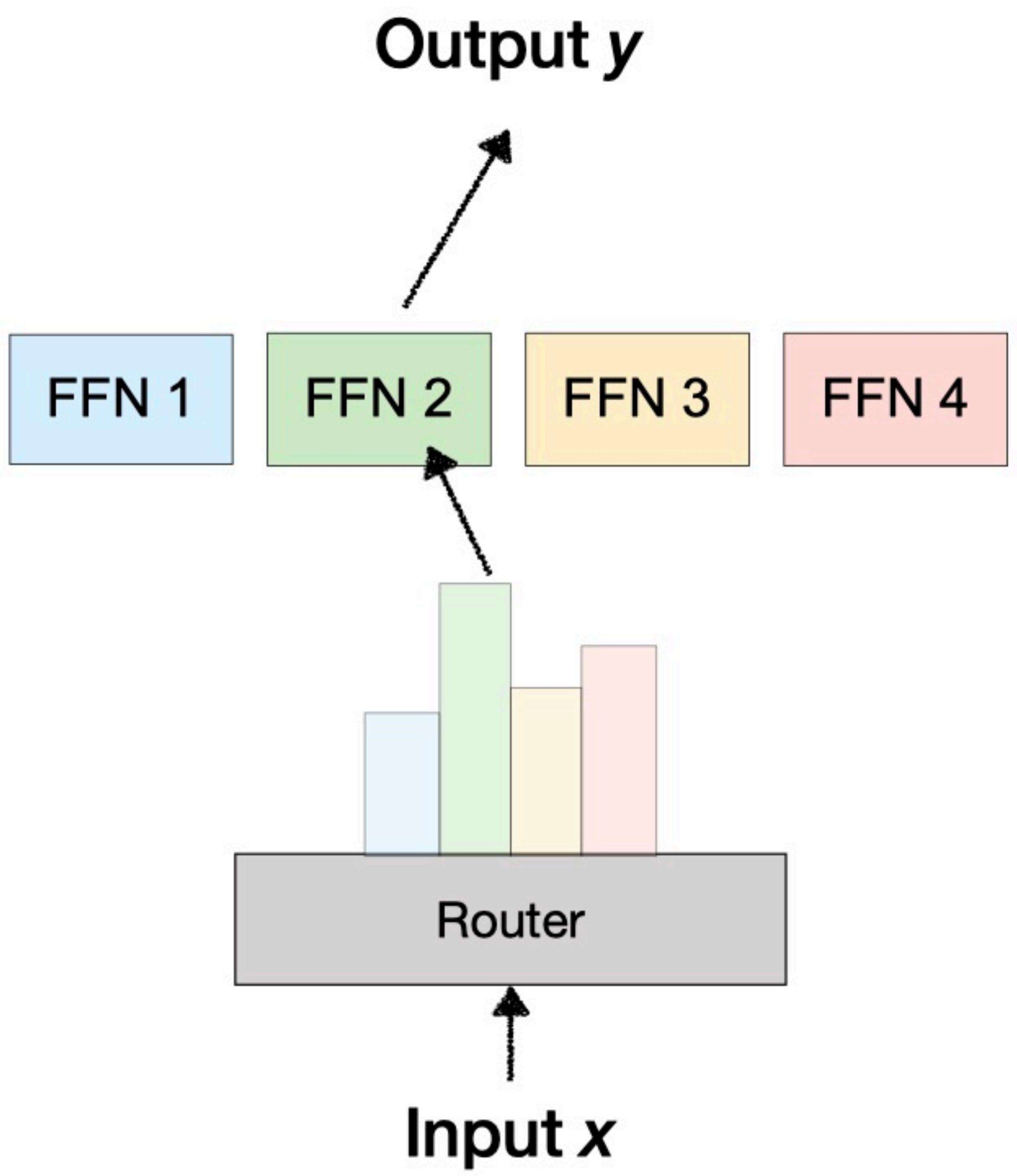


(Dosovitskiy et al., 2021): An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

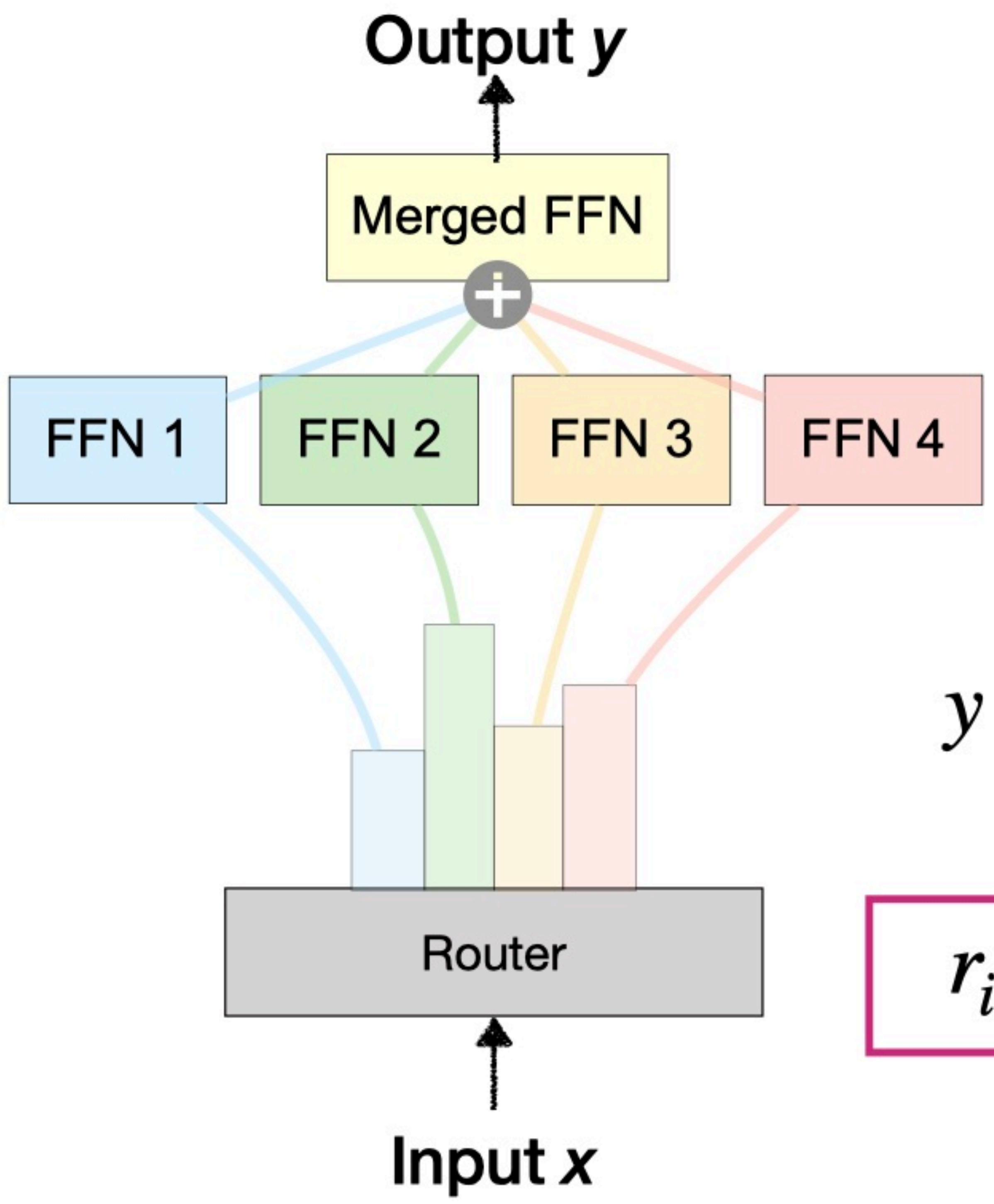
Mixture of experts (MoEs)



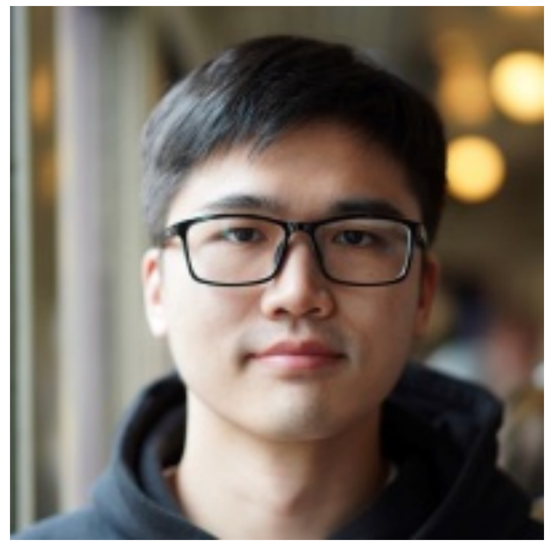
Mixture of experts (MoEs)



Conventional MoE



"Soft-merging" MoE



Merged FFN

$$y = \text{FFN}(x; \boxed{\sum_i r_i \theta_i})$$

$$\boxed{r_i = \text{softmax}(W_r x)}$$

Routing

(Zhong et al., 2024): Lory: Fully Differentiable Mixture-of-Experts for Autoregressive Language Model Pre-training

Part II. Pre-trained language models

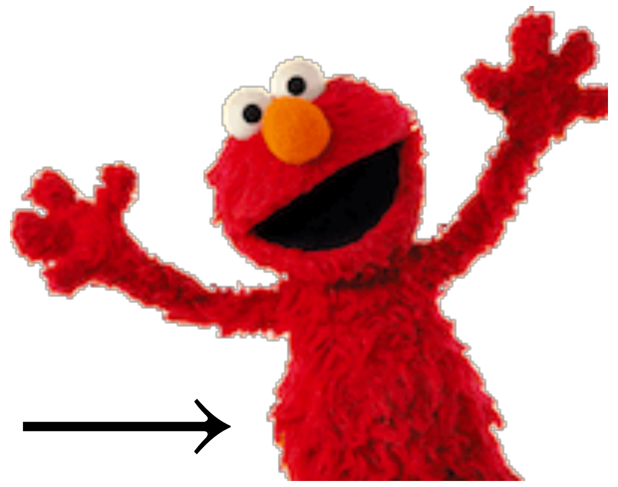
Roadmap

- **The BERT era**: pre-training and fine-tuning
- **The GPT-3 era**: prompting and in-context learning
- **The ChatGPT era**: supervised instruction tuning and RLHF

The BERT era: pre-training and fine-tuning

BERT = Bidirectional Encoder Representations from Transformers

LSTMs ⟶



ELMo

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
`{jacobdevlin,mingweichang,kentonl,kristout}@google.com`

⟵ Transformer decoder

GPT-1

BERT

(Devlin et al., 2019)

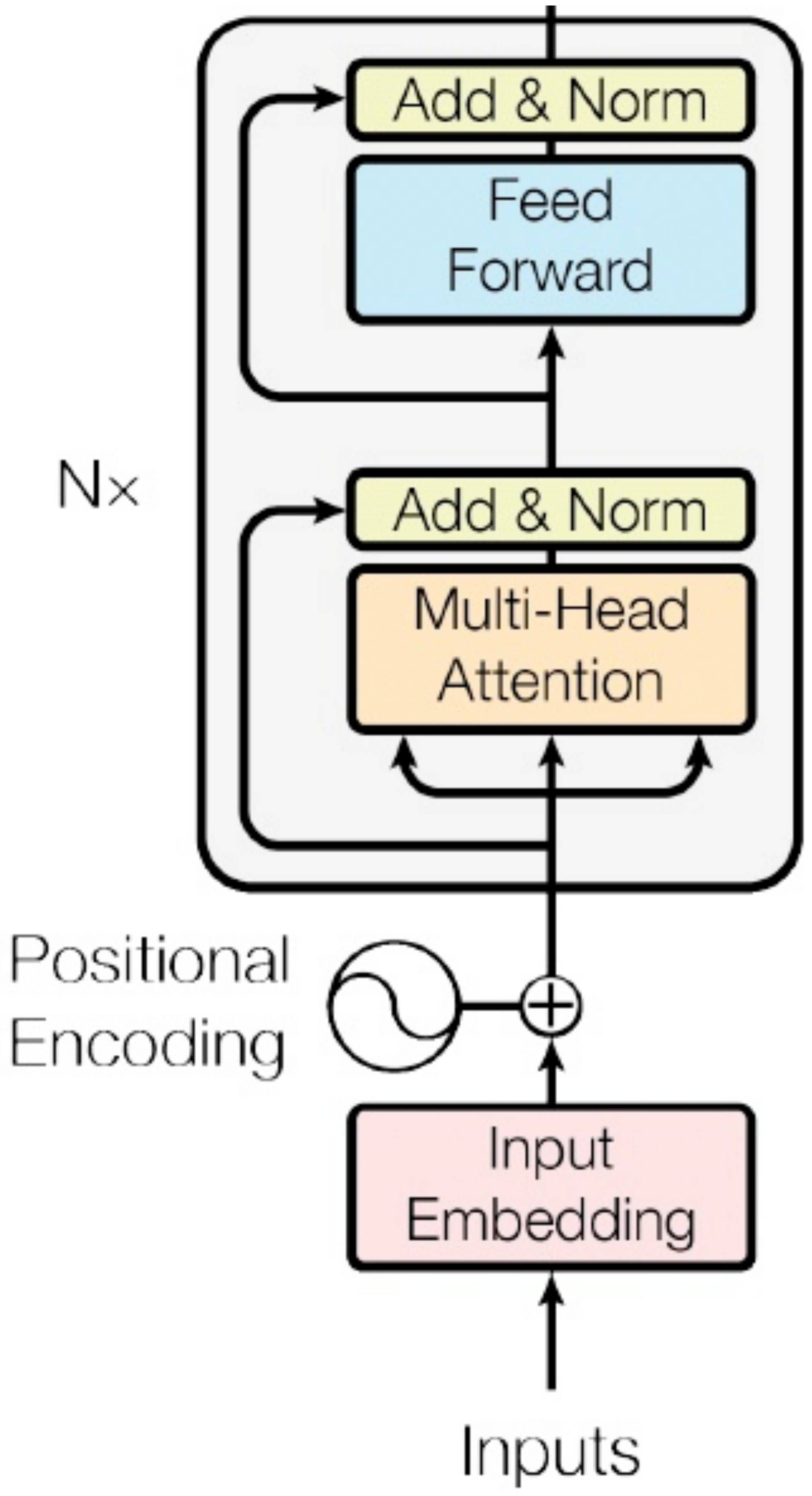
Input: a sequence of n words

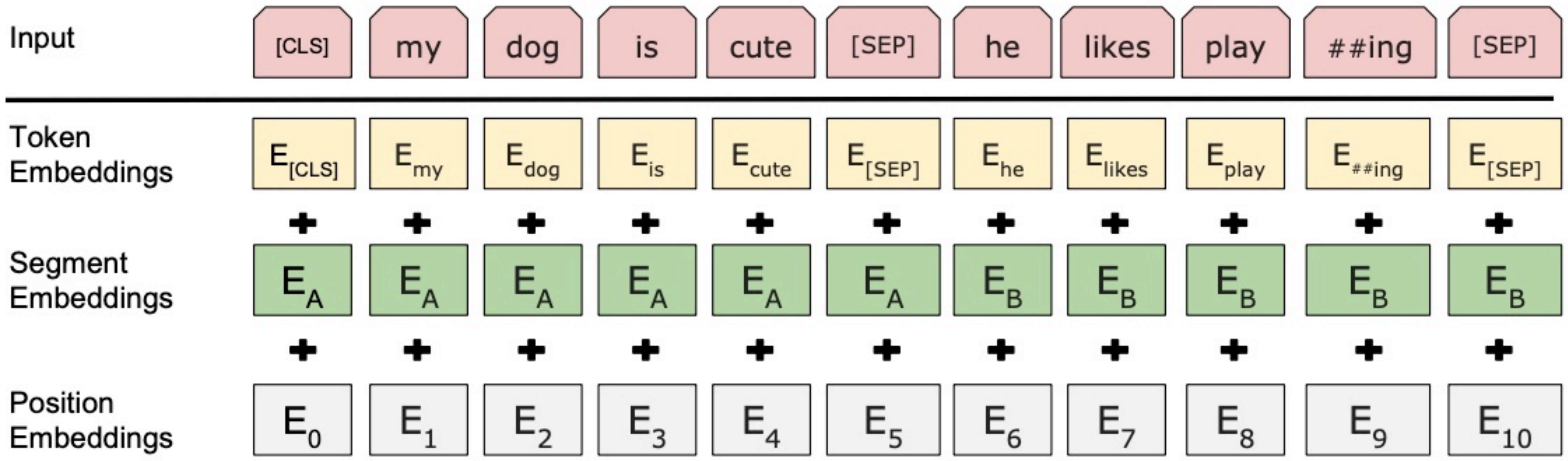
Output: a sequence of n vectors

aka. "**contextualized** word embeddings"

Each word doesn't have a fixed vector as in (static) word embeddings

Inside BERT: A Transformer encoder





- 12 or 24 layers of Transformer blocks
- Each block consists of a multi-head self-attention layer and a feedforward layer with residual connections

How is BERT pre-trained?

Two **pre-training** objectives:

- Masked language modeling (MLM)
- Next sentence prediction (NSP)

Masked language modeling

= mask out 15% of the inputs and then predict the masked words

store ↑ gallon ↑

the man went to the [MASK] to buy a [MASK] of milk

- Rather than *always* replacing the chosen words with `[MASK]`, the data generator will do the following:
- 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is hairy` → `my dog is [MASK]`
- 10% of the time: Replace the word with a random word, e.g., `my dog is hairy` → `my dog is apple`
- 10% of the time: Keep the word unchanged, e.g., `my dog is hairy` → `my dog is hairy`. The purpose of this is to bias the representation towards the actual observed word.

- Too little masking: too expensive to train
- Too much masking: not enough context

How is BERT pre-trained?

Two **pre-training** objectives:

- Masked language modeling (MLM)
- Next sentence prediction (NSP)

Next sentence prediction

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]

penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

Sample two segments of text (segment A and segment B) and predict whether the second segment is followed after the first one.

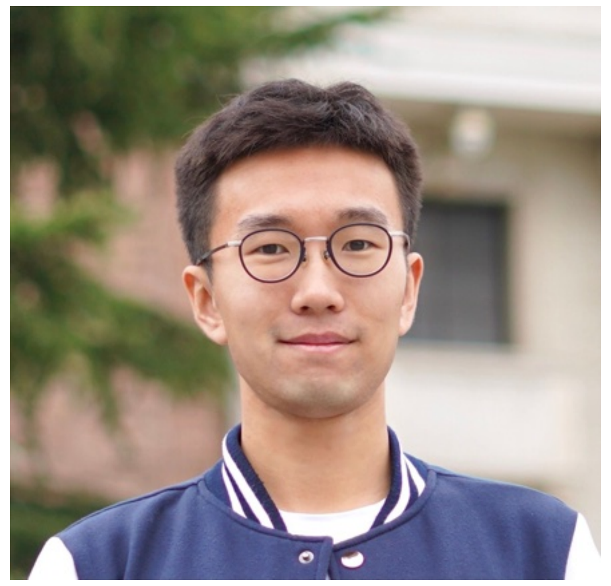
- 50% probability: a text segment of 512 tokens
- 50% probability: a text segment of 256 tokens, followed by another text segment of 256 tokens from a different document

The MLM loss and NSP loss are combined in pre-training

Remarks on BERT's pre-training objectives

- Later on, (Joshi et al., 2019; Liu et al., 2019) find that the next-sentence prediction objective unnecessary - RoBERTa doesn't use NSP at all.

- Understanding the role of masking rates (Why 15%?)



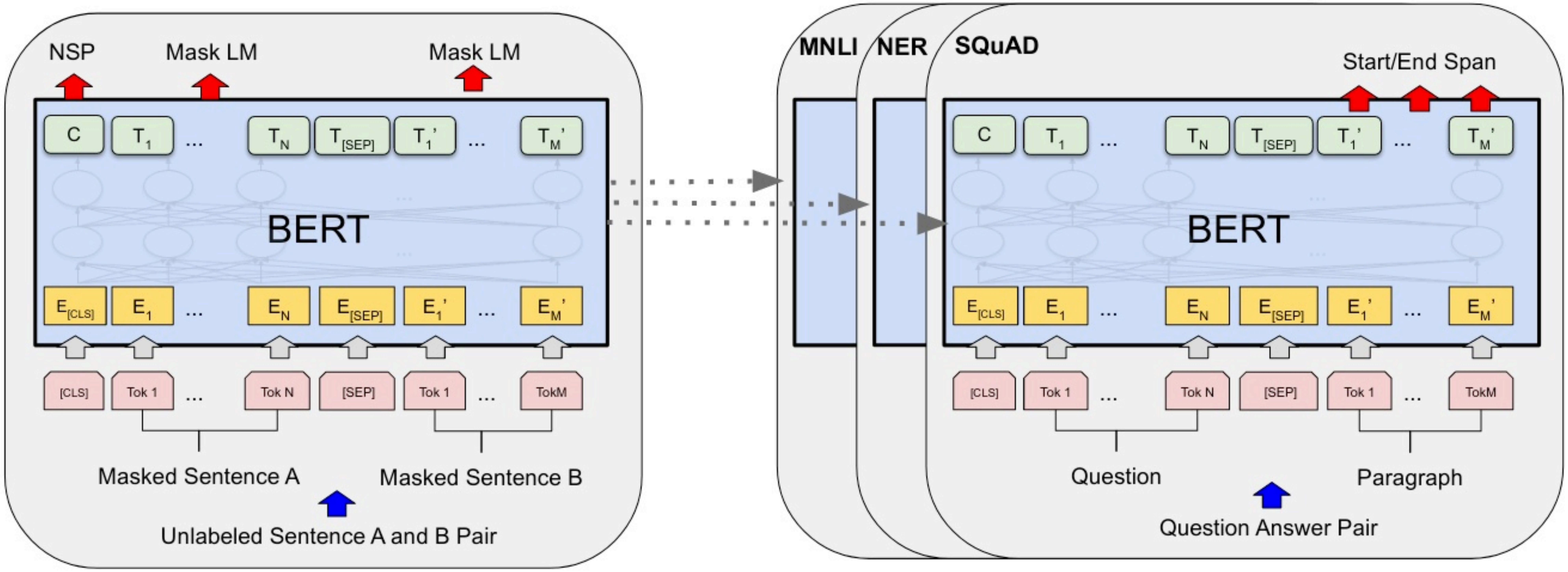
Should You Mask 15% in Masked Language Modeling?

Alexander Wettig* Tianyu Gao* Zexuan Zhong Danqi Chen
Department of Computer Science, Princeton University
{awettig,tianyug,zzhong,danqic}@cs.princeton.edu

(Wetting et al., 2023)

- The optimal masking rate should depend on *model sizes* and *masking strategies*
- Masking plays two distinct roles: corruption vs prediction

How is BERT used for downstream tasks?

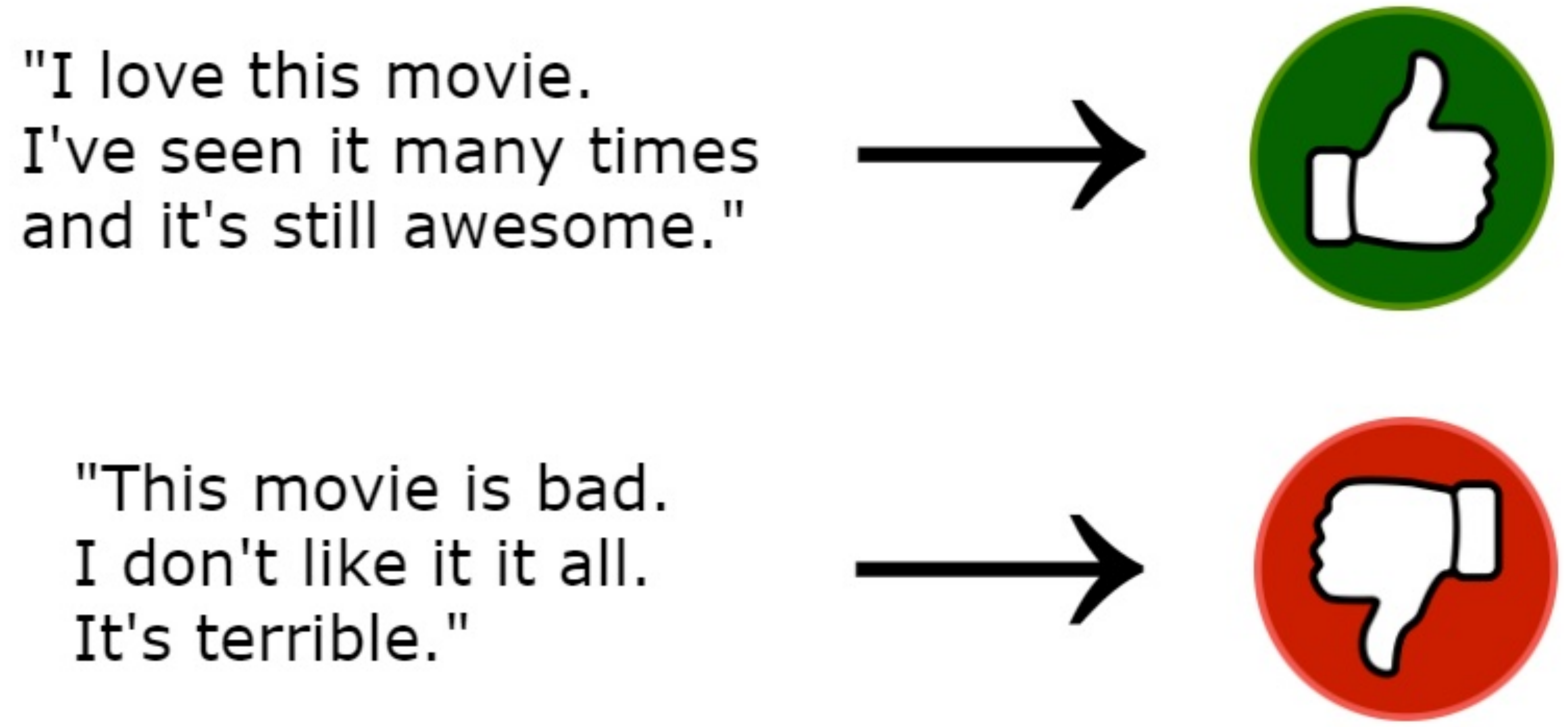


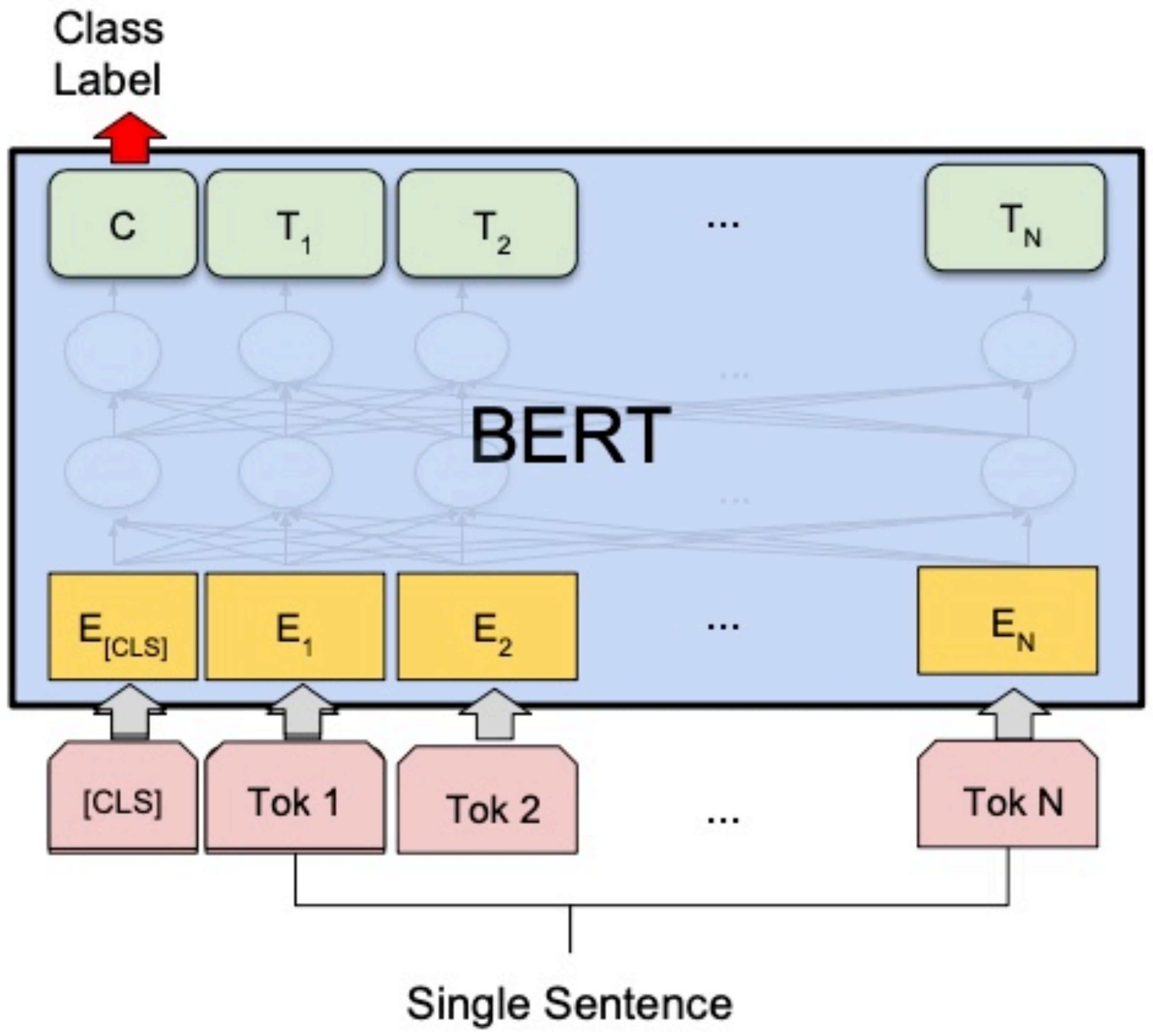
Pre-training

Fine-Tuning

The pre-trained BERT encoder can be used directly for downstream tasks with **minimal task-specific parameters**!

BERT for text classification





- Add a classifier on top of the [CLS] representation
- New parameters: $d \times |C|$, jointly trained with BERT parameters
  - $d =$ hidden dimension (e.g., 768)
  - $|C|$ = number of classes (e.g., 2)

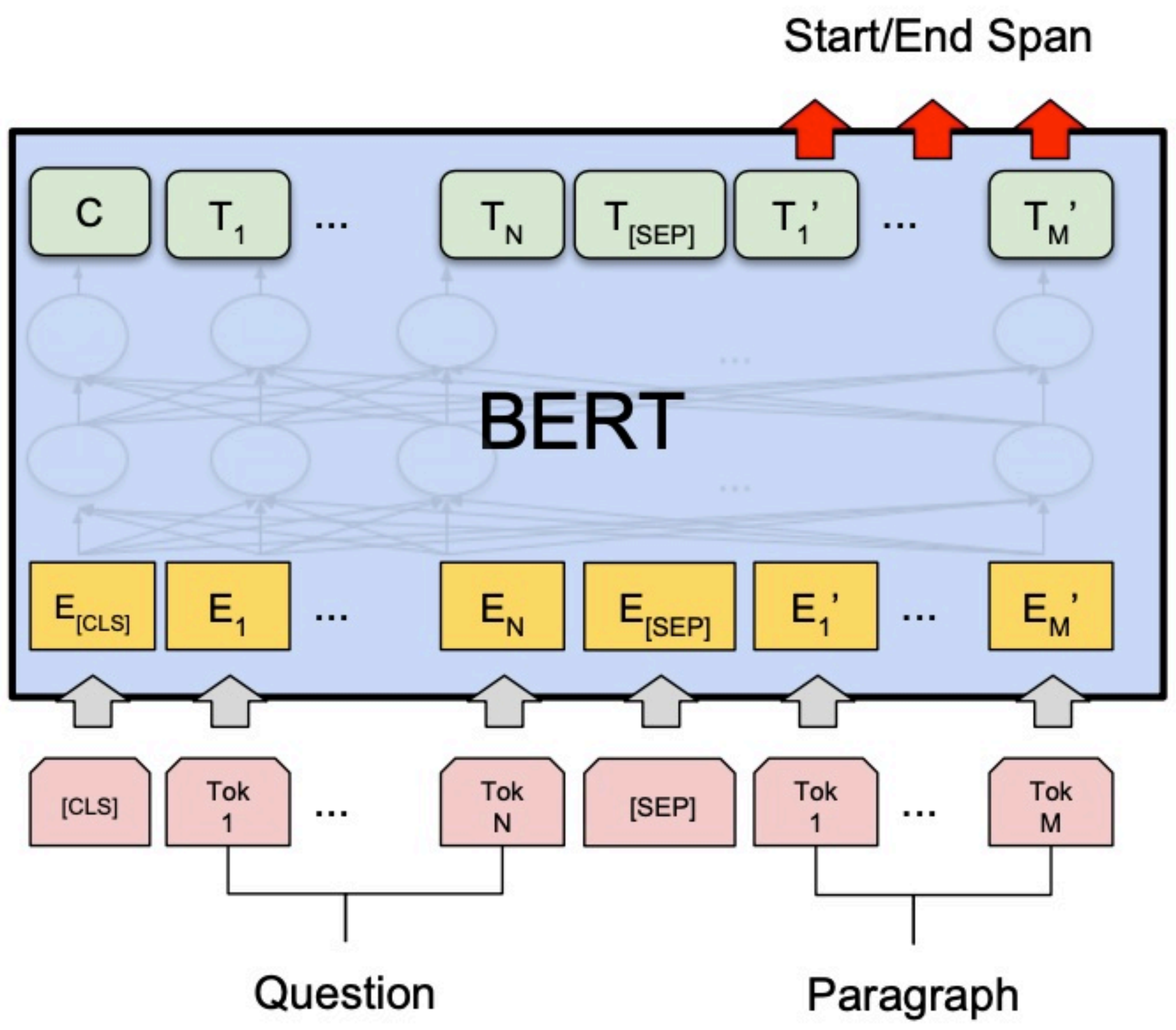
BERT for question answering

Stanford Question Answering Dataset (SQuAD)

Tesla was the fourth of five children. He had an older brother named Dane and three sisters, Milka, Angelina and Marica. Dane was killed in a horse-riding accident when Nikola was five. In 1861, Tesla attended the "Lower" or "Primary" School in Smiljan where he studied German, arithmetic, and religion. In 1862, the Tesla family moved to Gospić, Austrian Empire, where Tesla's father worked as a pastor. Nikola completed "Lower" or "Primary" School, followed by the "Lower Real Gymnasium" or "Normal School."

Q: What language did Tesla study while in school?

A: German



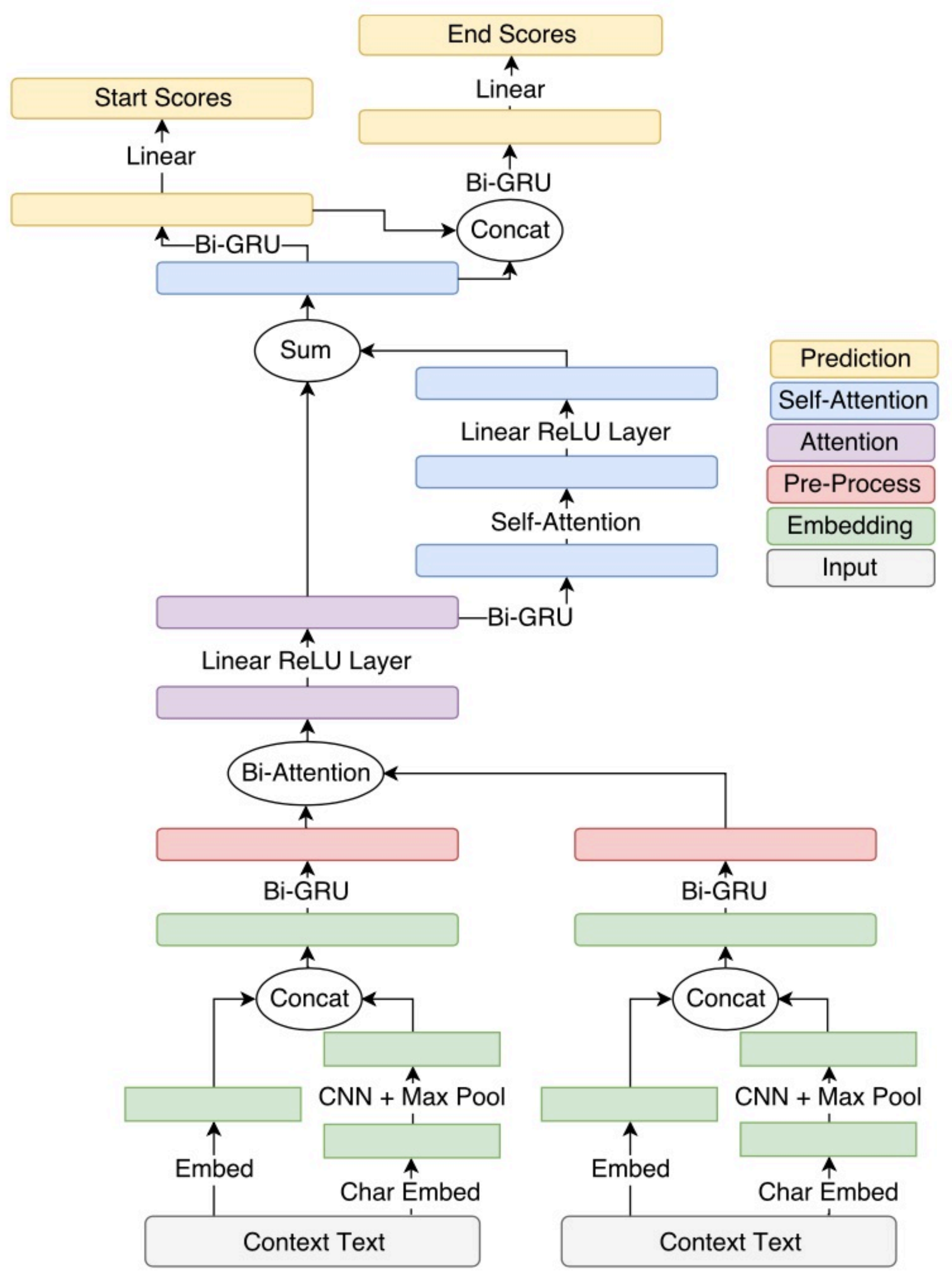
$$\mathcal{L} = -\log p_{\text{start}}(s^*) - \log p_{\text{end}}(e^*)$$

$$p_{\text{start}}(i) = \text{softmax}_i(\mathbf{w}_{\text{start}}^\intercal \mathbf{h}_i)$$

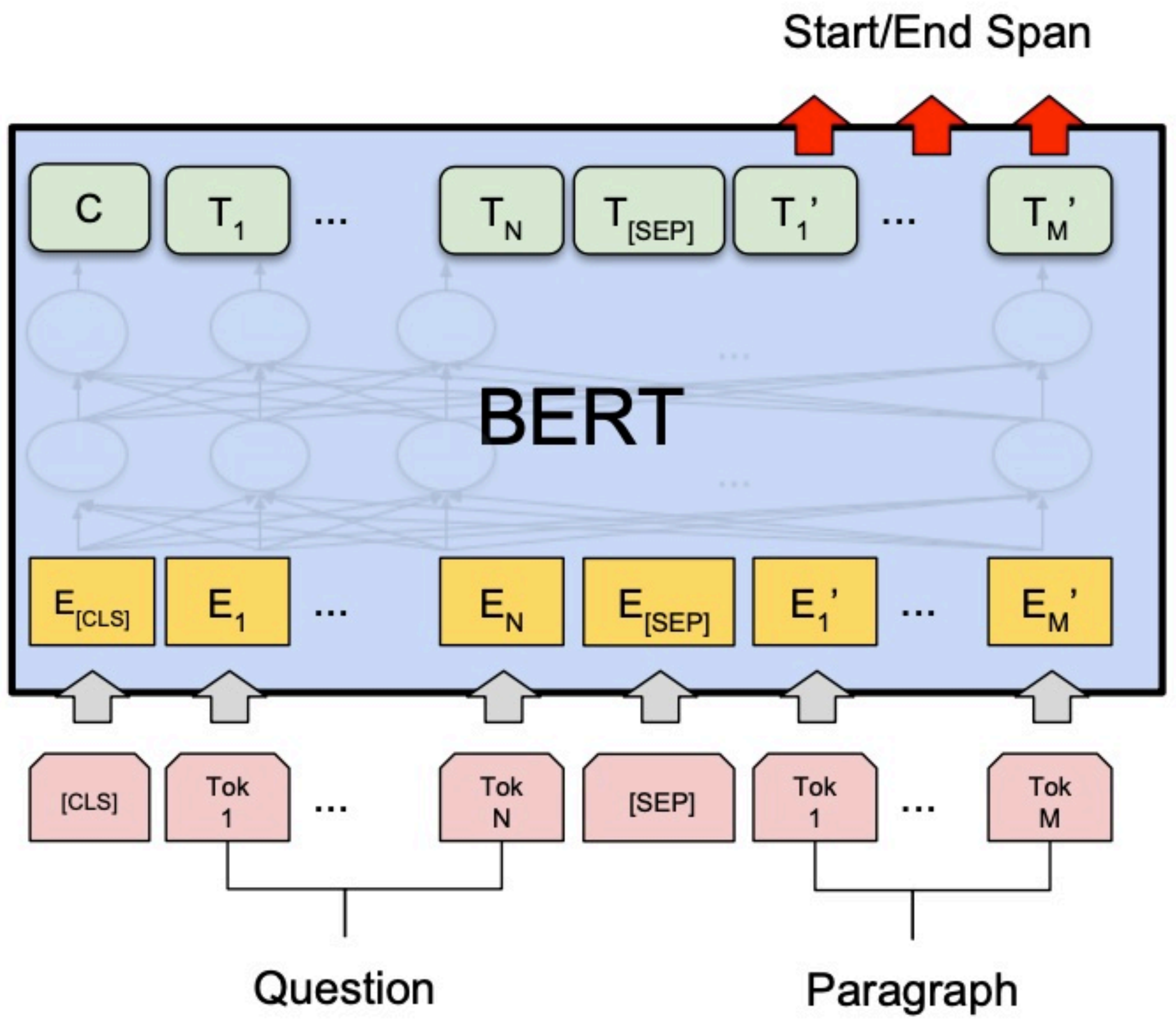
$$p_{\text{end}}(i) = \text{softmax}_i(\mathbf{w}_{\text{end}}^\intercal \mathbf{h}_i)$$

(Rajpurkar et al., 2016): SQuAD: 100,000+ Questions for Machine Comprehension of Text

Pre-BERT QA models vs BERT



Only the word embeddings at the input layer are pre-trained (2-3M parameters in total)



All the parameters are pre-trained except for a small number of task-specific parameters $\mathbf{w}_{\text{start}}, \mathbf{w}_{\text{end}}$

BERT has 110M or 330M parameters

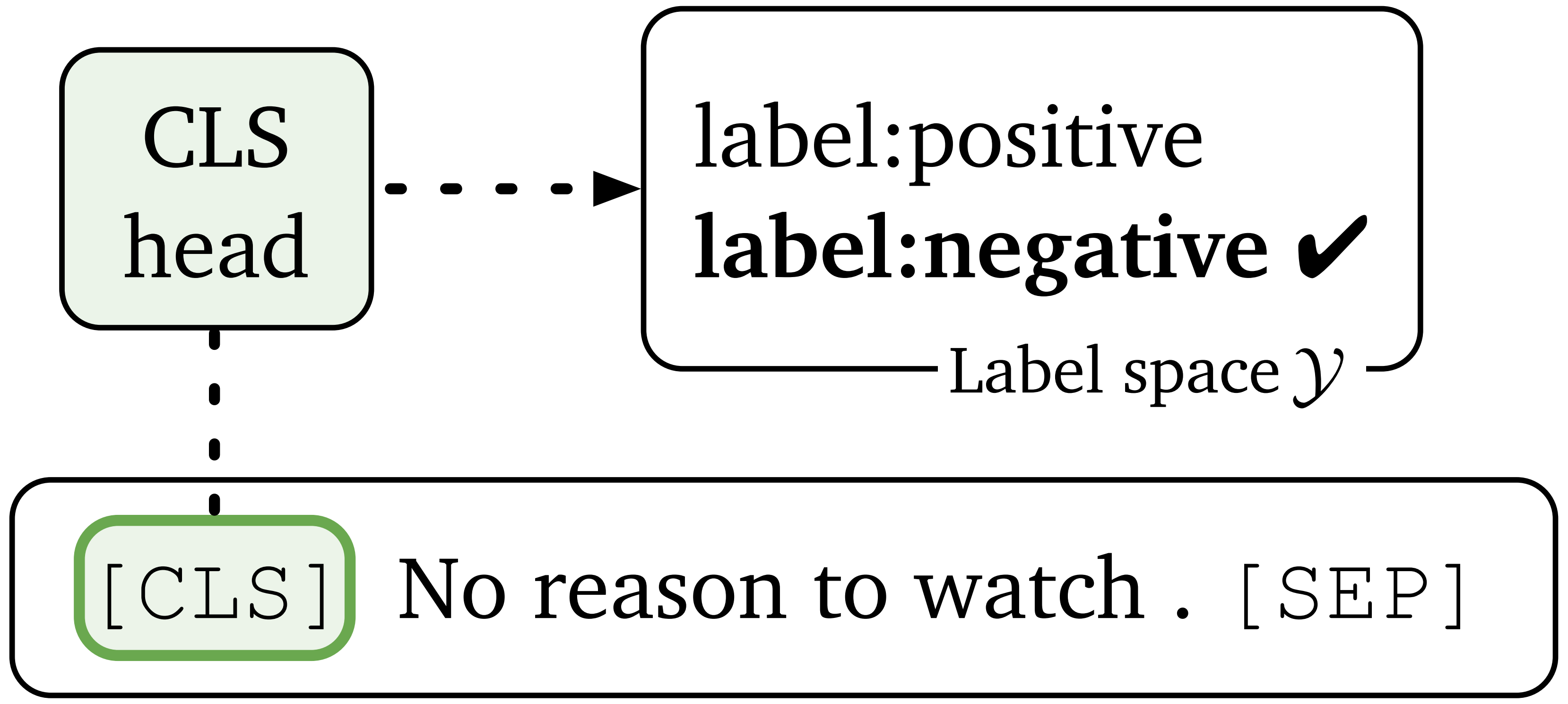
Prompt-based fine-tuning

Task: sentiment classification

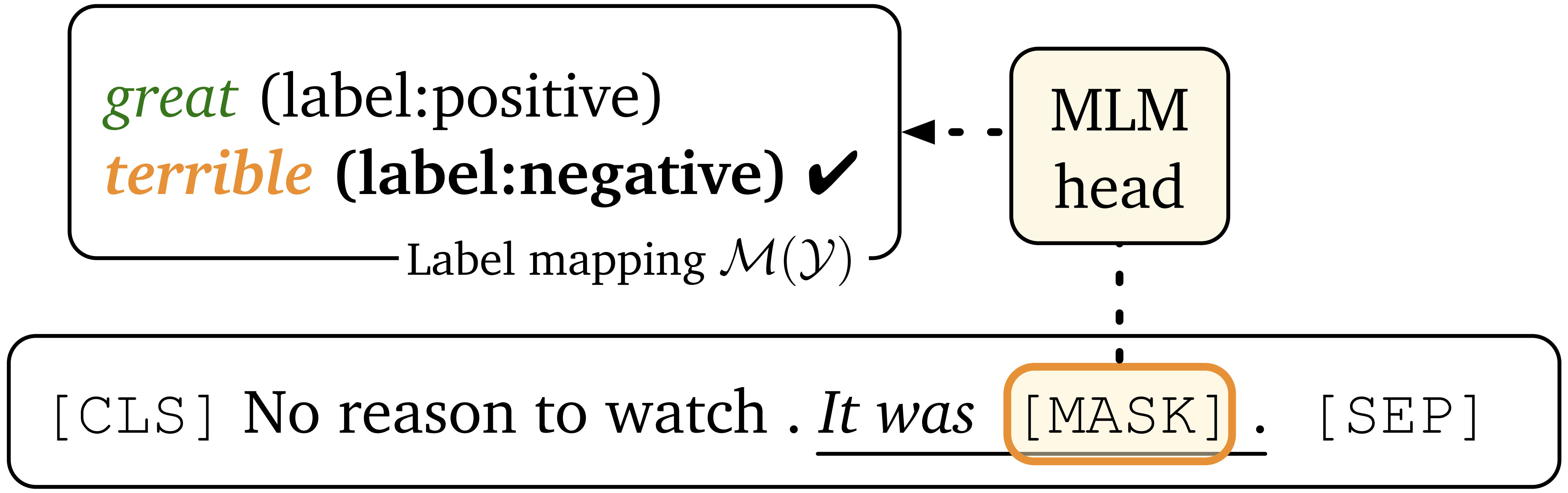
Input: “No reason to watch.”

Output: positive 👍 or negative 👎 ?


Label mapping



Fine-tuning



Prompt-based **fine-tuning**

81.4% (32 examples)
93.5% (67k examples; BERT-base)

92.7% (32 examples)

(Gao et al., 2021): Making Pre-trained Language Models Better Few-shot Learners

BERT: training cost

- BERT-base: 12 layers, n = 512, d = 768, 110M parameters
- BERT-large: 24 layers, n = 512, d = 1024, 330M parameters
- Trained on Wikipedia + BooksCorpus (3.3 billion tokens)
- Estimate: 6 days for BERT-base and 26 days for BERT-large on 8 Nvidia Titan-V GPUs (12Gb memory)

RESEARCH

SHARE

by Jacob Portes*, Alex Trott*, Daniel King, Sam Havens
on March 9, 2023

MosaicBERT: Pretraining BERT from Scratch for $20

https://www.mosaicml.com/blog/mosaicbert

(8x A100-80Gb GPUs)

RoBERTa

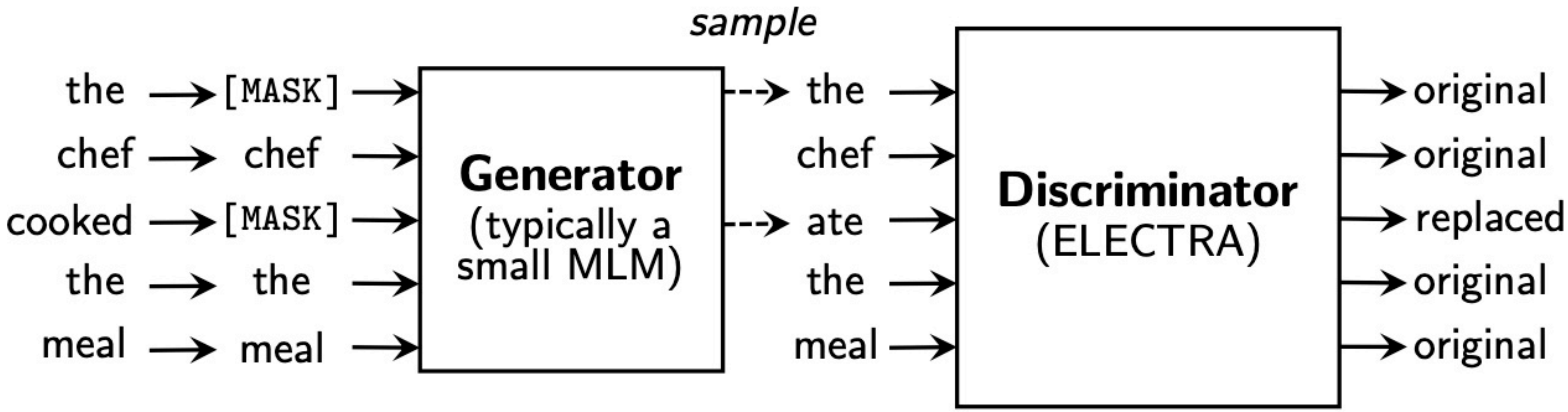
- BERT is still under-trained
- Removed the next-sentence prediction objective
- Trained longer with 10x data & bigger batch sizes
- Pre-trained on 1,024 V100 GPUs for one day in 2019

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

(Liu et al., 2019): RoBERTa: A Robustly Optimized BERT Pretraining Approach

ELECTRA

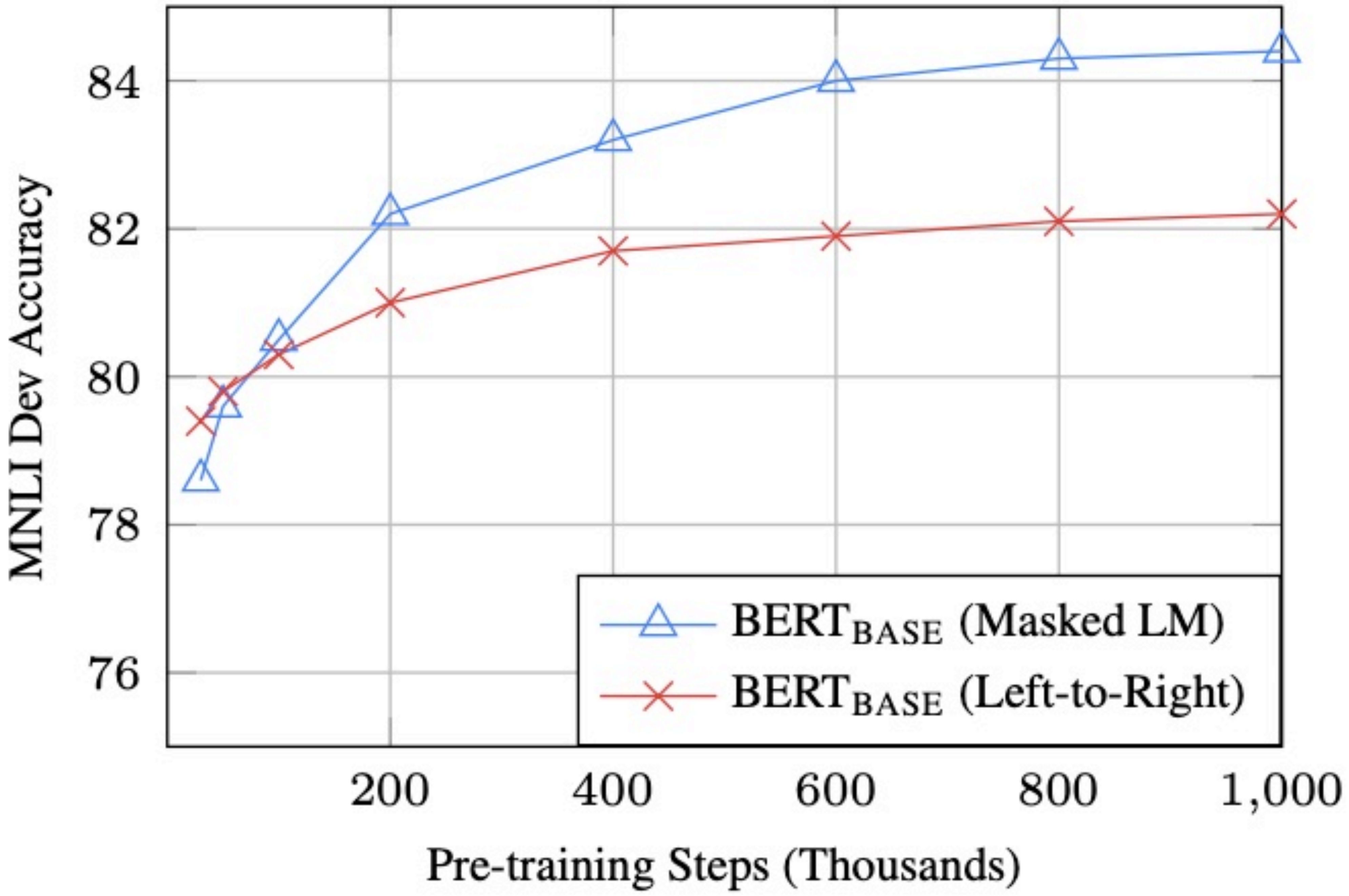
ELECTRA provides a more **efficient** training method,
because it predicts 100% of tokens (instead of 15%) every time



(Clark et al., 2020): ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators

BERT vs GPT-1 models

- Unlike GPT-1 (autoregressive language models), BERT/RoBERTa/ELECTRA can't generate text naturally!
- However, bidirectionality is important for natural language understanding tasks.



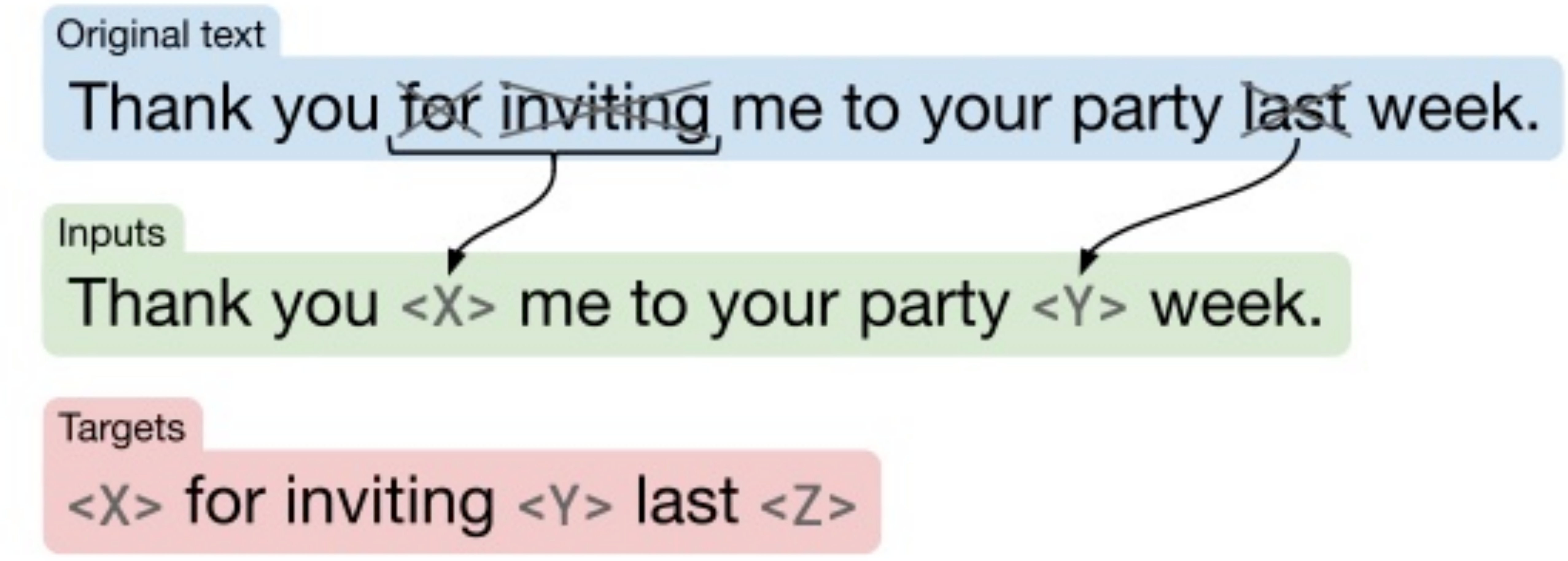
(Devlin et al., 2019)

Why not combine the best of both worlds?

T5: Text-to-text models

- T5 = **T**ext-**to**-**T**ext **T**ransfer **T**ransformer
- Transformer encoder-decoder architecture
  - Encoder: preserves bidirectionality
  - Decoder: good for generation!

Pre-training (always 15% span masking)



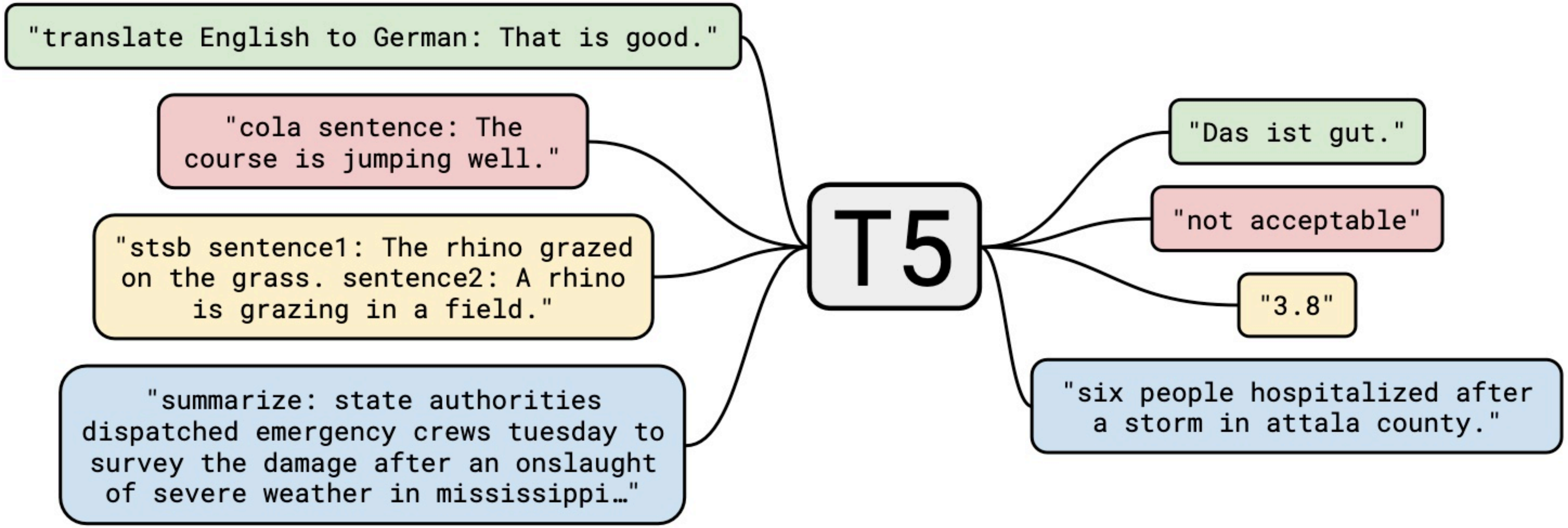
(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

T5: Text-to-text models

- T5 = **T**ext-**to**-**T**ext **T**ransfer **T**ransformer

Fine-tuning

All NLU tasks can be cast as text prediction tasks too!



(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

T5: Text-to-text models

- T5 = **T**ext-**to**-**T**ext **T**ransfer **T**ransformer

T5 comes in different sizes:

- t5-small.
- t5-base.
- t5-large.
- t5-3b.
- t5-11b.

Training corpus:
C4 = **C**olossal **C**lean **C**rawled **C**orpus

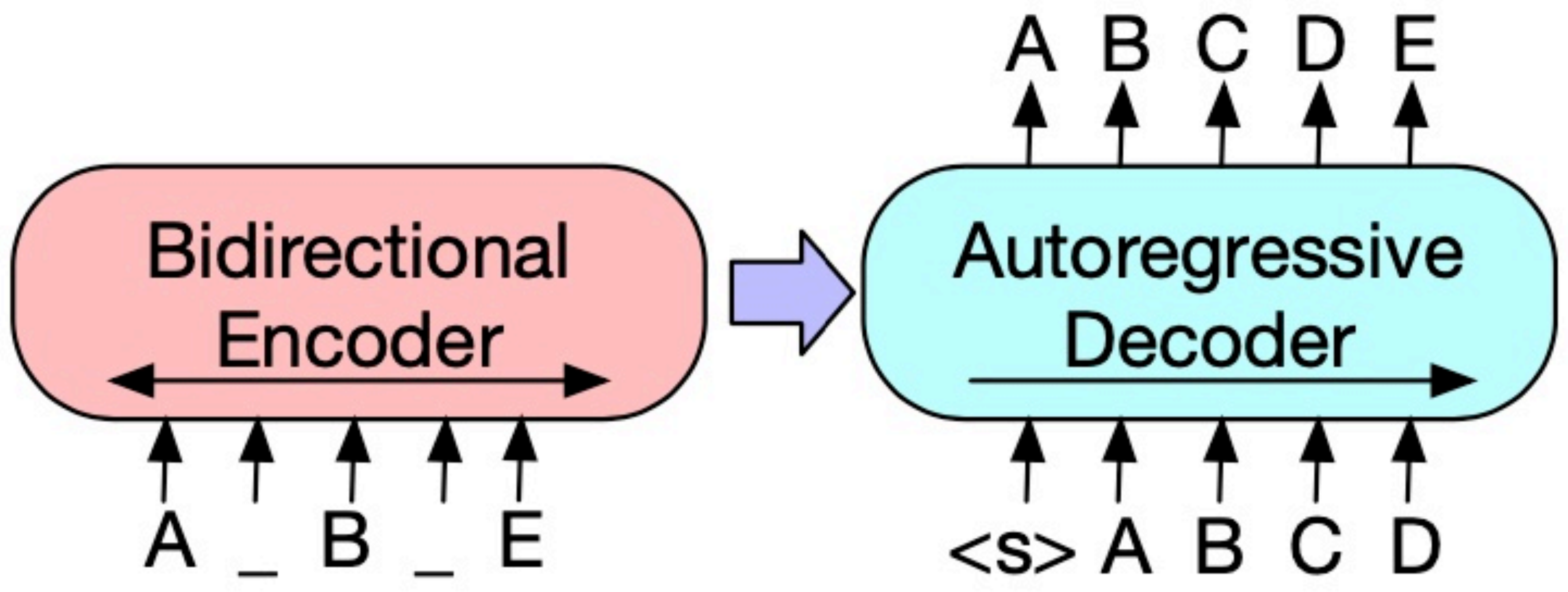
A cleaned version of Common Crawl

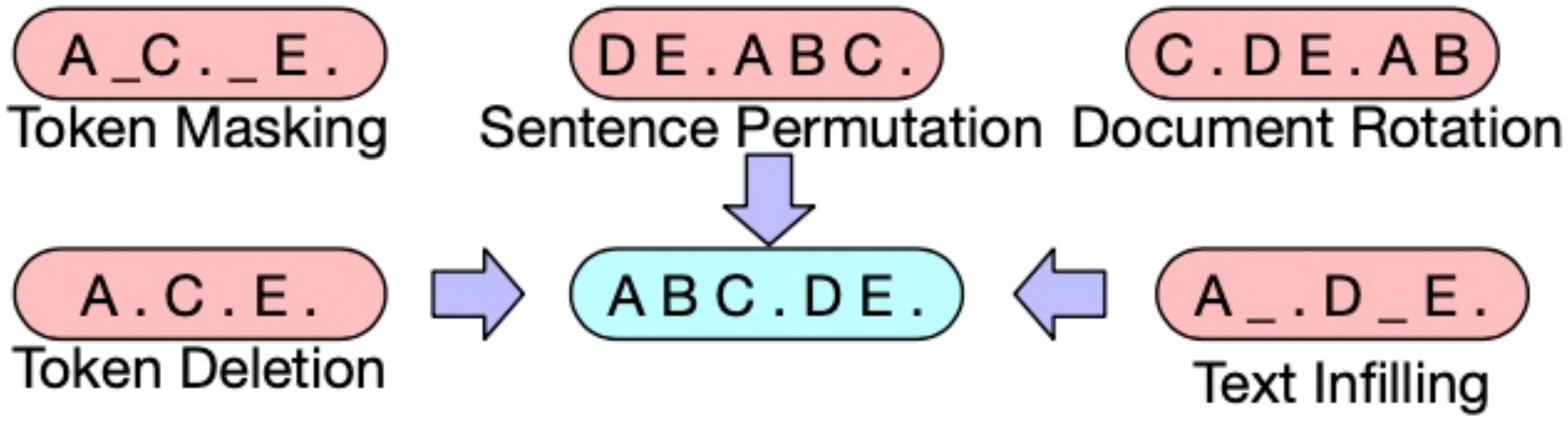
Trained on **34B** tokens

(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

BART: another text-to-text model

BART





(Lewis et al., 2020): BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension

The GPT-3 era:
prompting and in-context learning

GPT-3

Language Models are Few-Shot Learners

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*

Jared Kaplan† Prafulla Dhariwal Arvind Neelakantan Pranav Shyam

Girish Sastry Amanda Askell Sandhini Agarwal Ariel Herbert-Voss

Gretchen Krueger Tom Henighan Rewon Child Aditya Ramesh

Daniel M. Ziegler Jeffrey Wu Clemens Winter

Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray

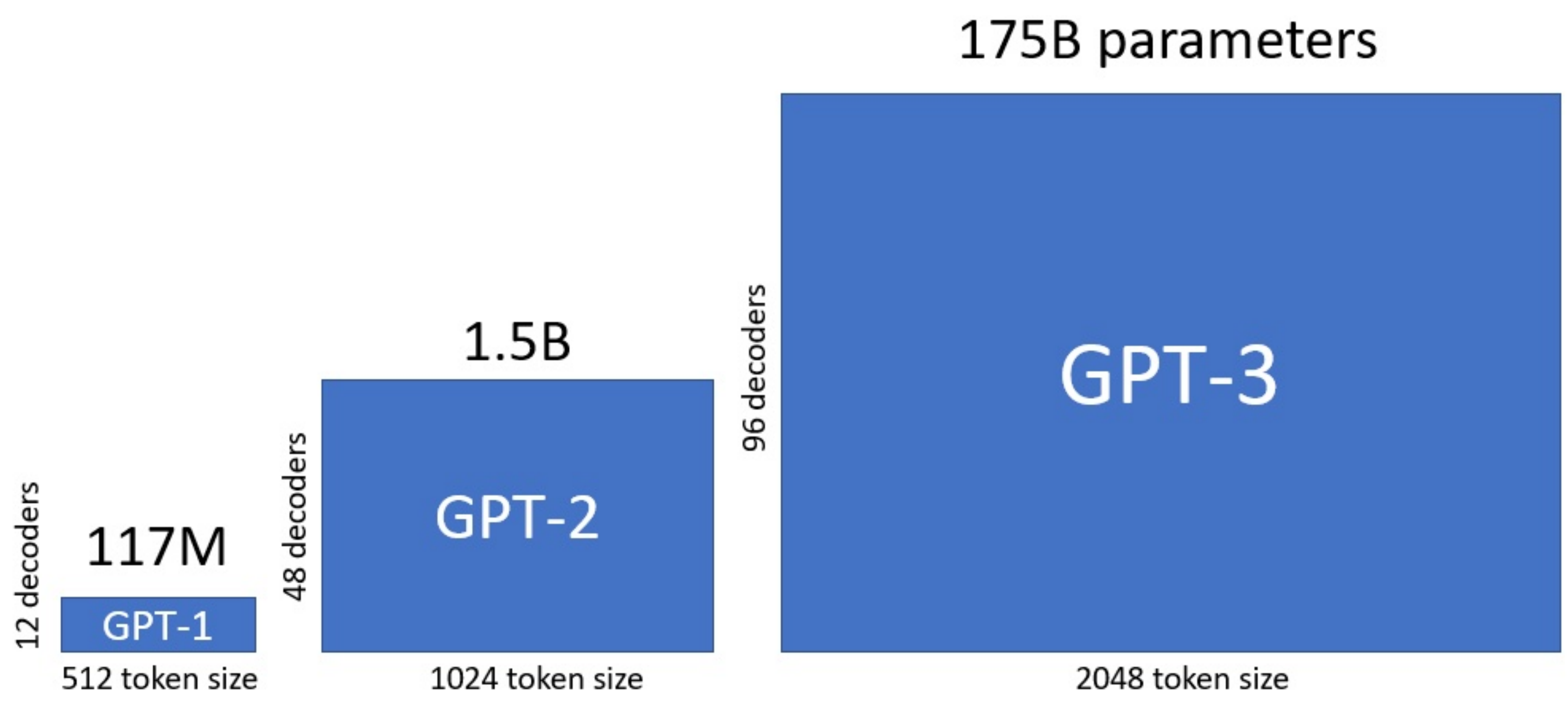
Benjamin Chess Jack Clark Christopher Berner

Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei

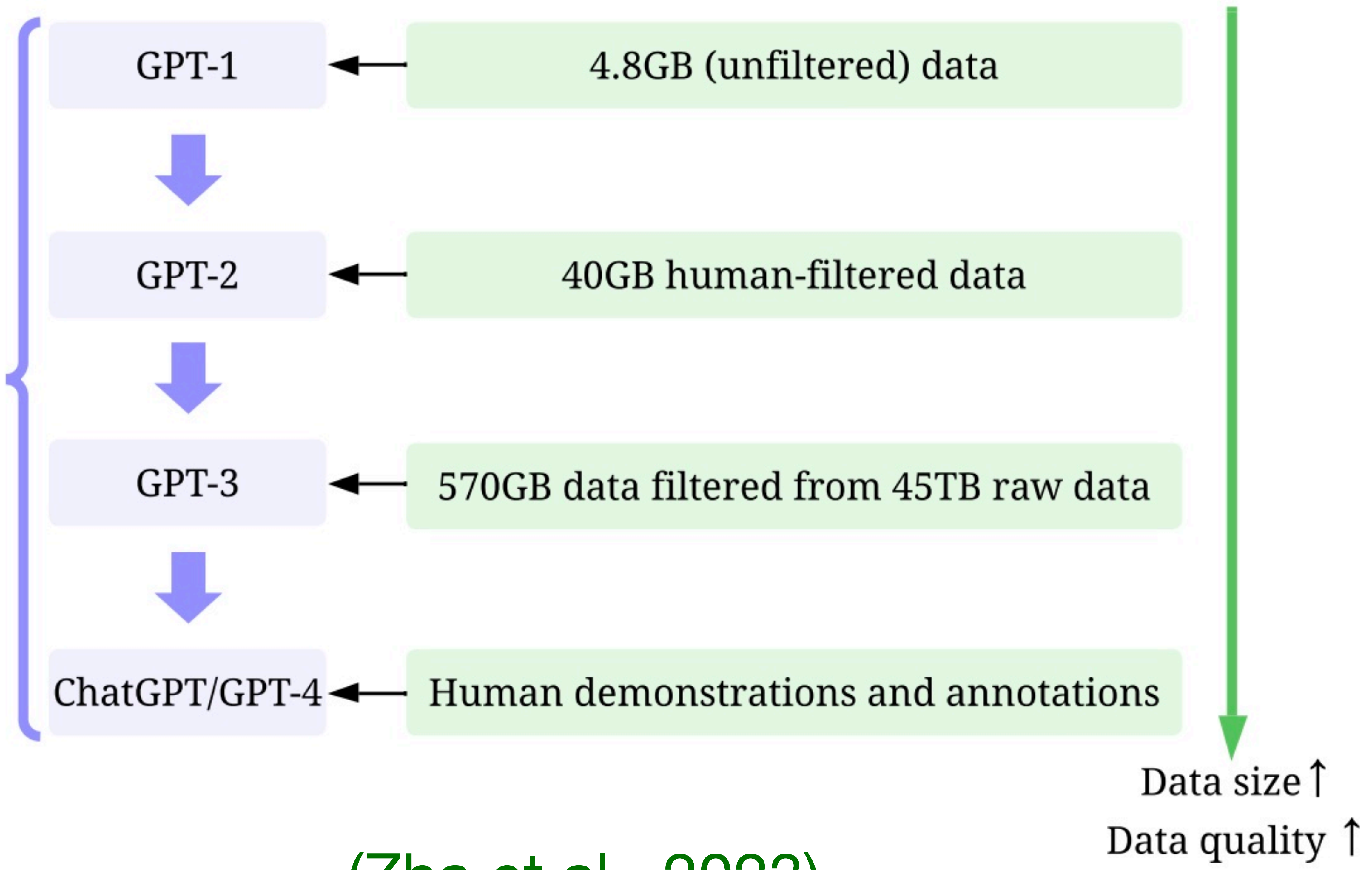
(Brown et al., 2020)

- From GPT-1 to GPT-2 to GPT-3
- **Transformer decoder** for autoregressive language modeling
- What's new?
  - Parameters ↑ Data ↑
  - These models can start generating human-like text!
  - You don't have to fine-tune these models - **prompting** works well!

GPT-1 vs GPT-2 vs GPT-3



https://medium.com/@YanAIx/



(Zha et al., 2023)

GPT-2 generation example

PROMPT (HUMAN-WRITTEN)	*A train carriage containing controlled nuclear materials was stolen in Cincinnati today. Its whereabouts are unknown.*
MODEL COMPLETION	The incident occurred on the downtown train line, which runs from Covington and Ashland stations. In an email to Ohio news outlets, the U.S. Department of Energy said it is working with the Federal Railroad Administration to find the thief. "The theft of this nuclear material will have significant negative consequences on public and environmental health, our workforce and the economy of our nation," said Tom Hicks, the U.S. Energy Secretary, in a statement. "Our top priority is to secure the theft and ensure it doesn't happen again." The stolen material was taken from the University of Cincinnati's Research Triangle Park nuclear research site, according to a news release from Department officials.

GPT-3's in-context learning

- Before GPT-3, most use of pre-trained language models are through ***fine-tuning***, on a reasonably-sized supervised dataset.
    - SST-2 has 67k examples, SQuAD has 88k (passage, answer, question) triples.

- GPT-3 shows that, with a very large **autoregressive** language model (**175B parameters**), the model can perform a task:
    - Using only **a few examples** **"Few-shot learning"**
    - **Without gradient updates:**

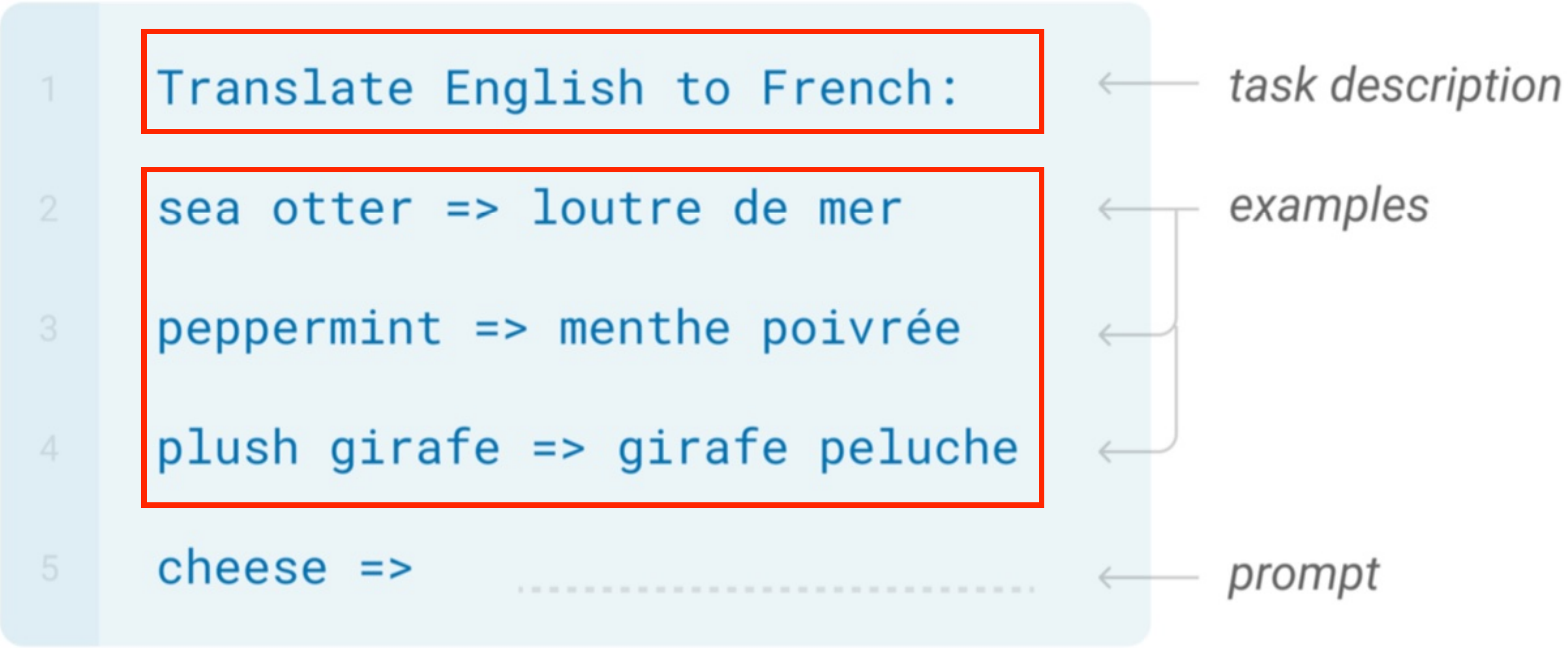
      the examples are only provided in the context! **"In-context learning"**

(Brown et al., 2020): Language Models are Few-Shot Learners

In-context learning

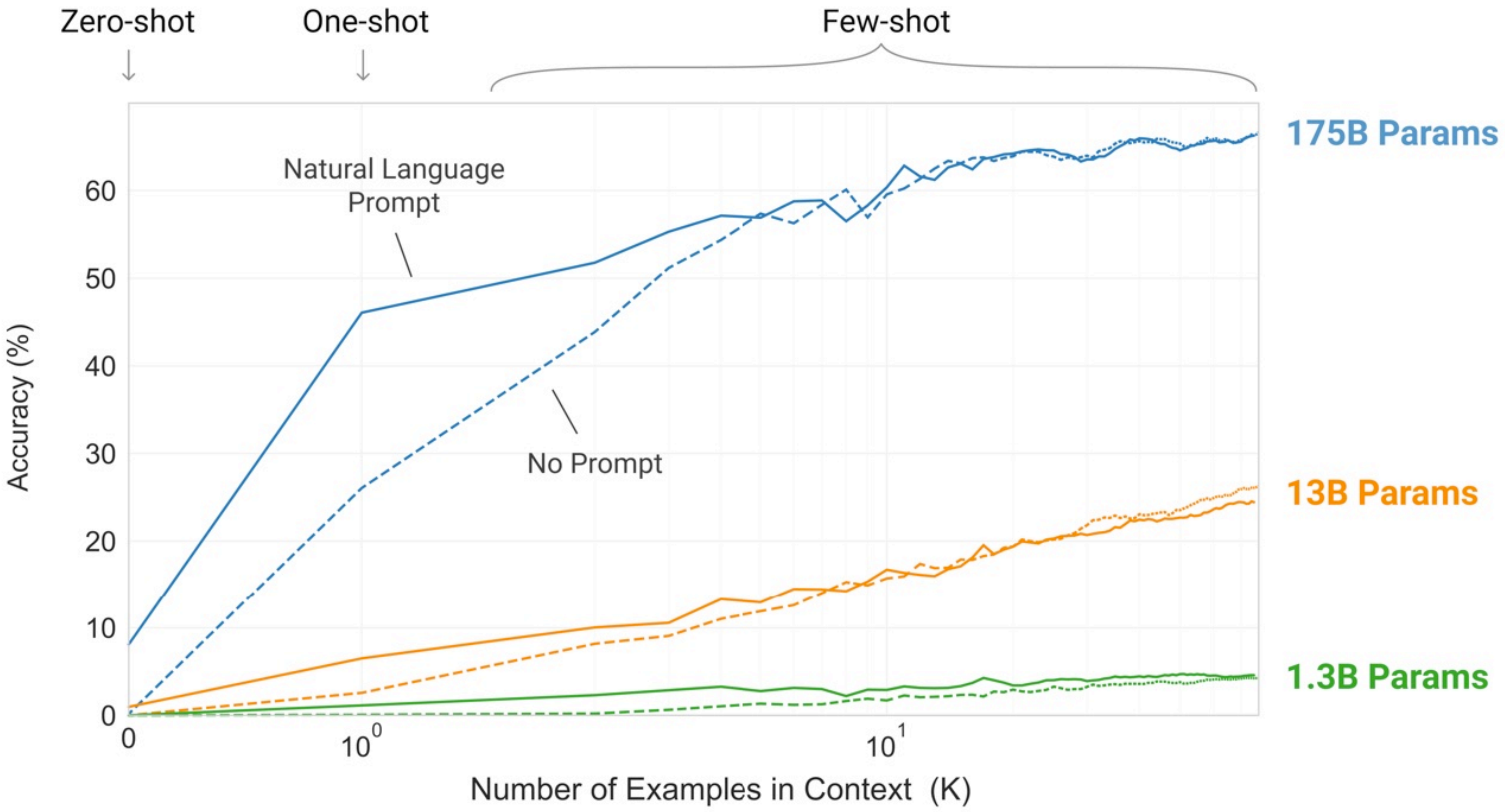
Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



(Brown et al., 2020): Language Models are Few-Shot Learners

In-context learning



(Brown et al., 2020): Language Models are Few-Shot Learners

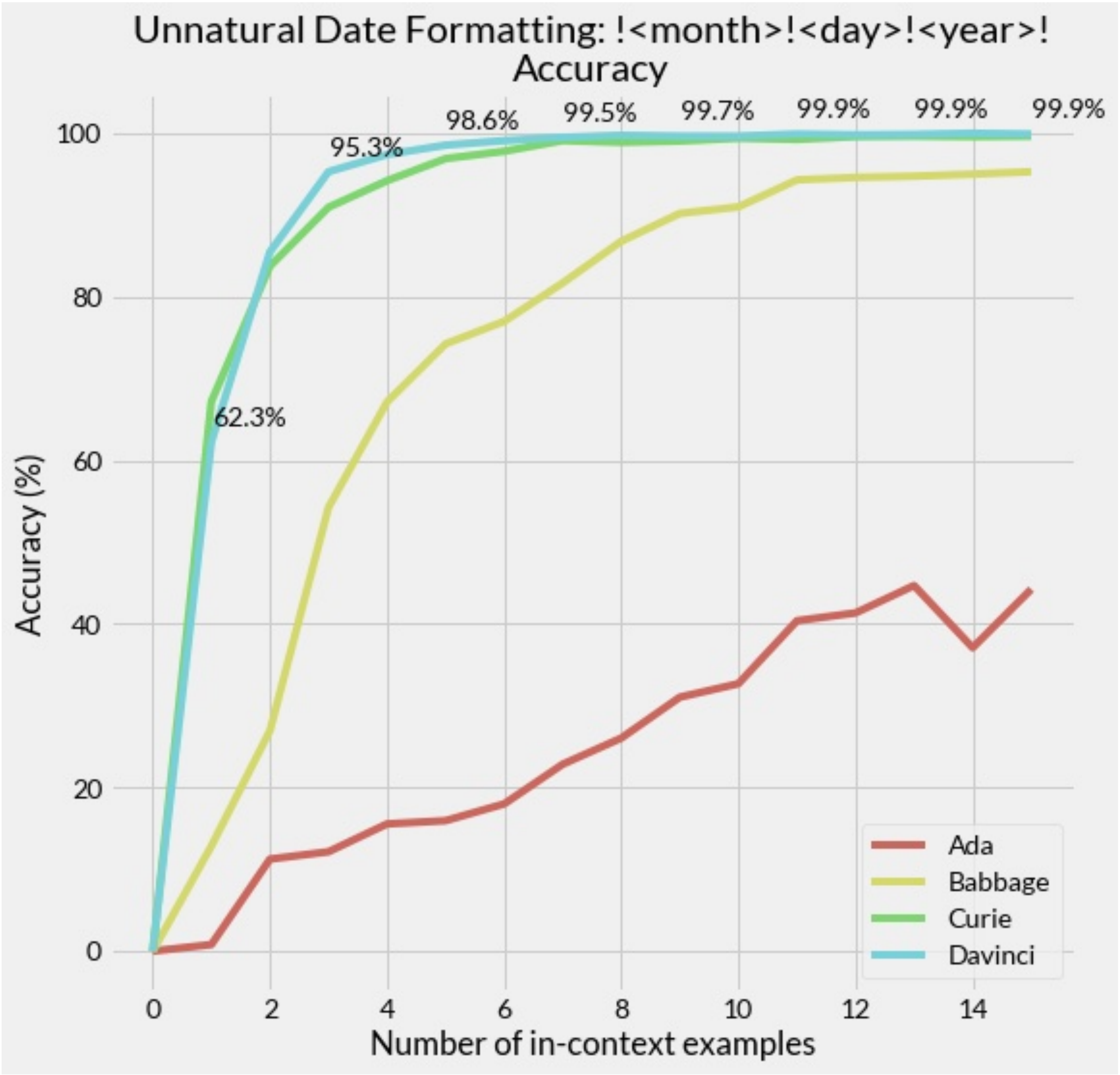
In-context learning

```
Input: 2014-06-01
Output: !06!01!2014!
Input: 2007-12-13
Output: !12!13!2007!
Input: 2010-09-23
Output: !09!23!2010!
Input: 2005-07-23
Output: !07!23!2005!
```

in-context examples

test example

model completion



http://ai.stanford.edu/blog/in-context-learning/

Understanding in-context learning

- **Hypothesis #1**: Transformers perform implicit gradient descent to update an “inner model”

Transformers Learn In-Context by Gradient Descent

Johannes von Oswald [1 2] Eyvind Niklasson [2] Ettore Randazzo [2] João Sacramento [1]
Alexander Mordvintsev [2] Andrey Zhmoginov [2] Max Vladymyrov [2]

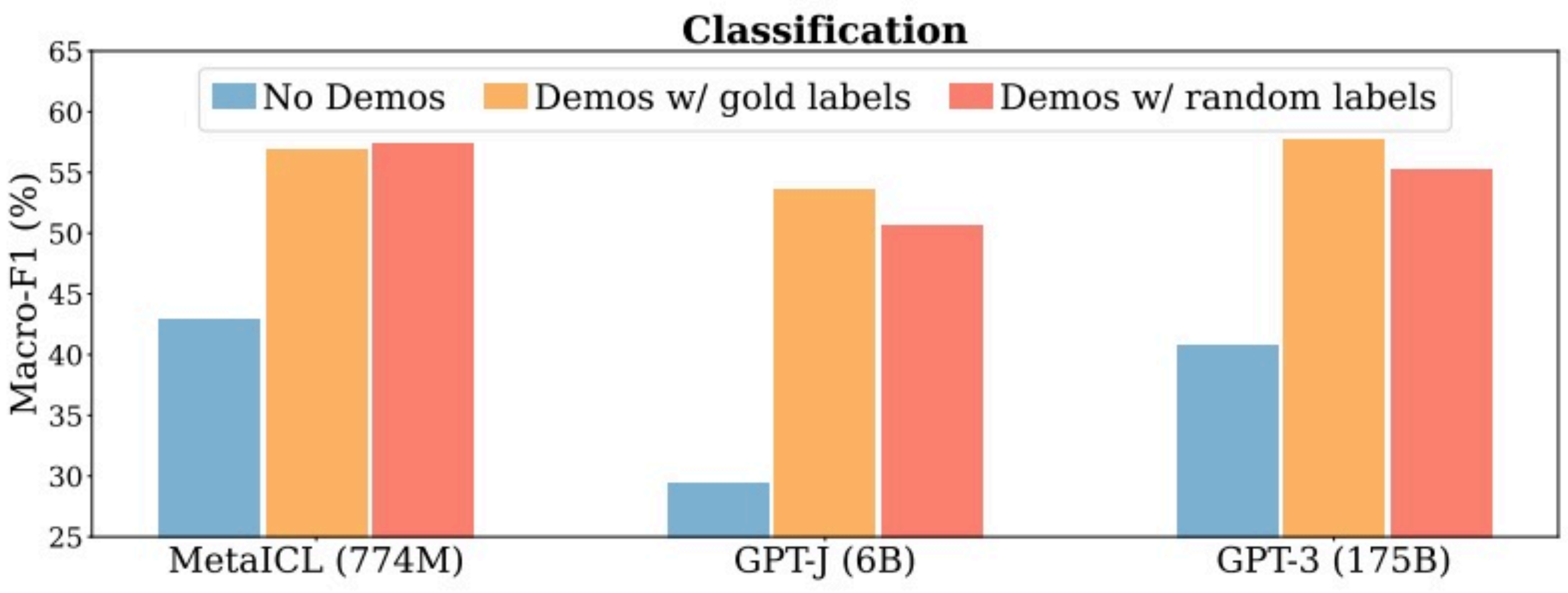
**Why Can GPT Learn In-Context?
Language Models Implicitly Perform Gradient Descent as Meta-Optimizers**

Damai Dai[†*] Yutao Sun[‖*] Li Dong[‡], Yaru Hao[‡], Shuming Ma[‡], Zhifang Sui[†], Furu Wei[‡]

- **Hypothesis #2**: Transformers learn tasks required for downstream applications during pre-training, and in-context demonstrations are only used to recognize which task is required

**Rethinking the Role of Demonstrations:
What Makes In-Context Learning Work?**

Sewon Min[1,2] Xinxi Lyu[1] Ari Holtzman[1] Mikel Artetxe[2]
Mike Lewis[2] Hannaneh Hajishirzi[1,3] Luke Zettlemoyer[1,2]
[1]University of Washington [2]Meta AI [3]Allen Institute for AI
{sewon,alrope,ahai,hannaneh,lsz}@cs.washington.ed
{artetxe,mikelewis}@meta.com



Ground-truth labels don’t matter!

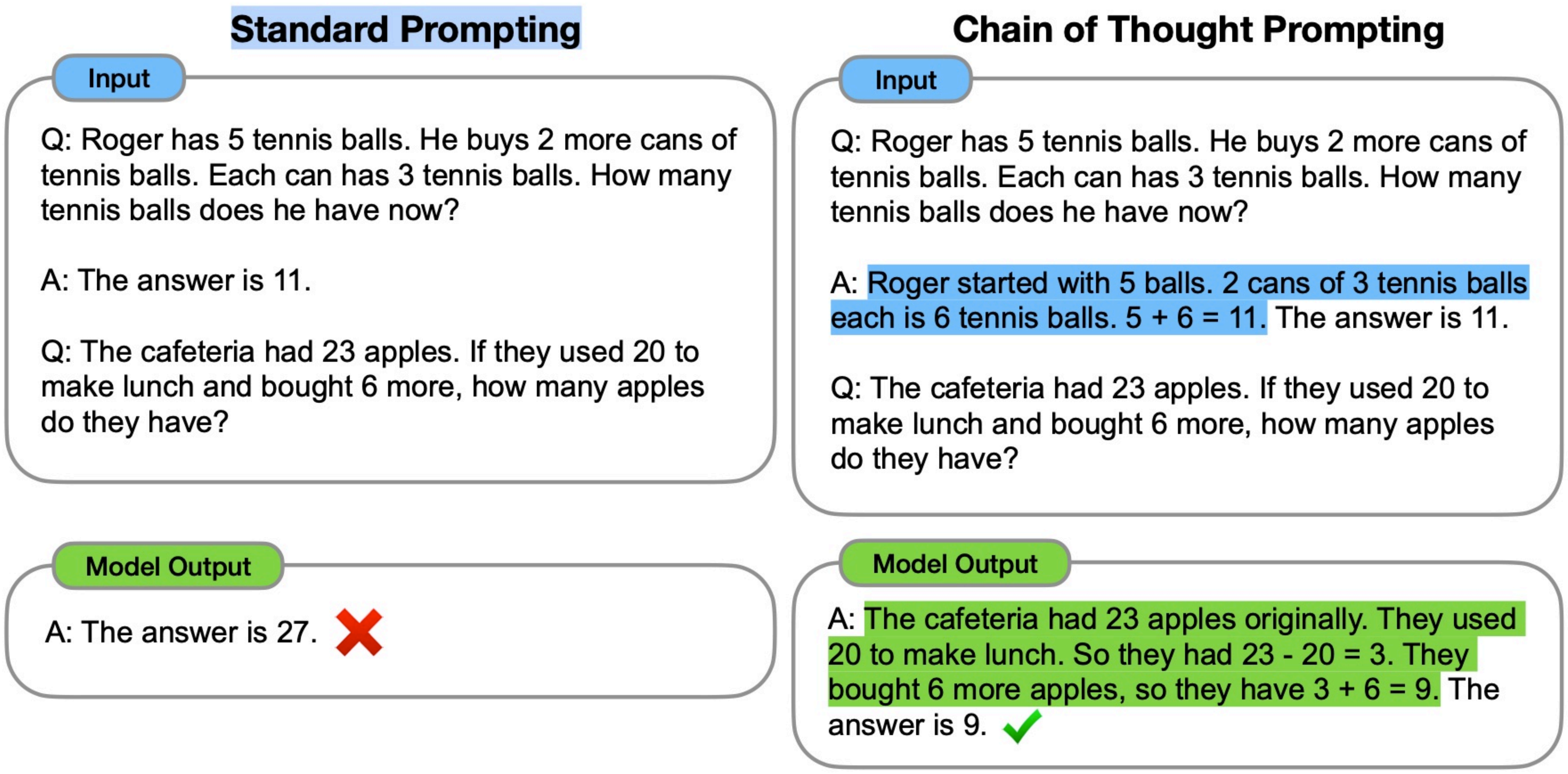
Understanding in-context learning

We disentangle In-context learning into two roles -
task recognition (TR) vs **task learning (TL)**

- TR: recognizes the task from demonstrations and applies LLMs' pre-trained priors
- TL: learns a new input-label mapping from demonstrations
- ICL performs both TR and TL, but TL emerges with **larger models** and **more demonstrations**

(Pan et al., 2023): What In-Context Learning "Learns" In-Context: Disentangling Task Recognition and Task Learning

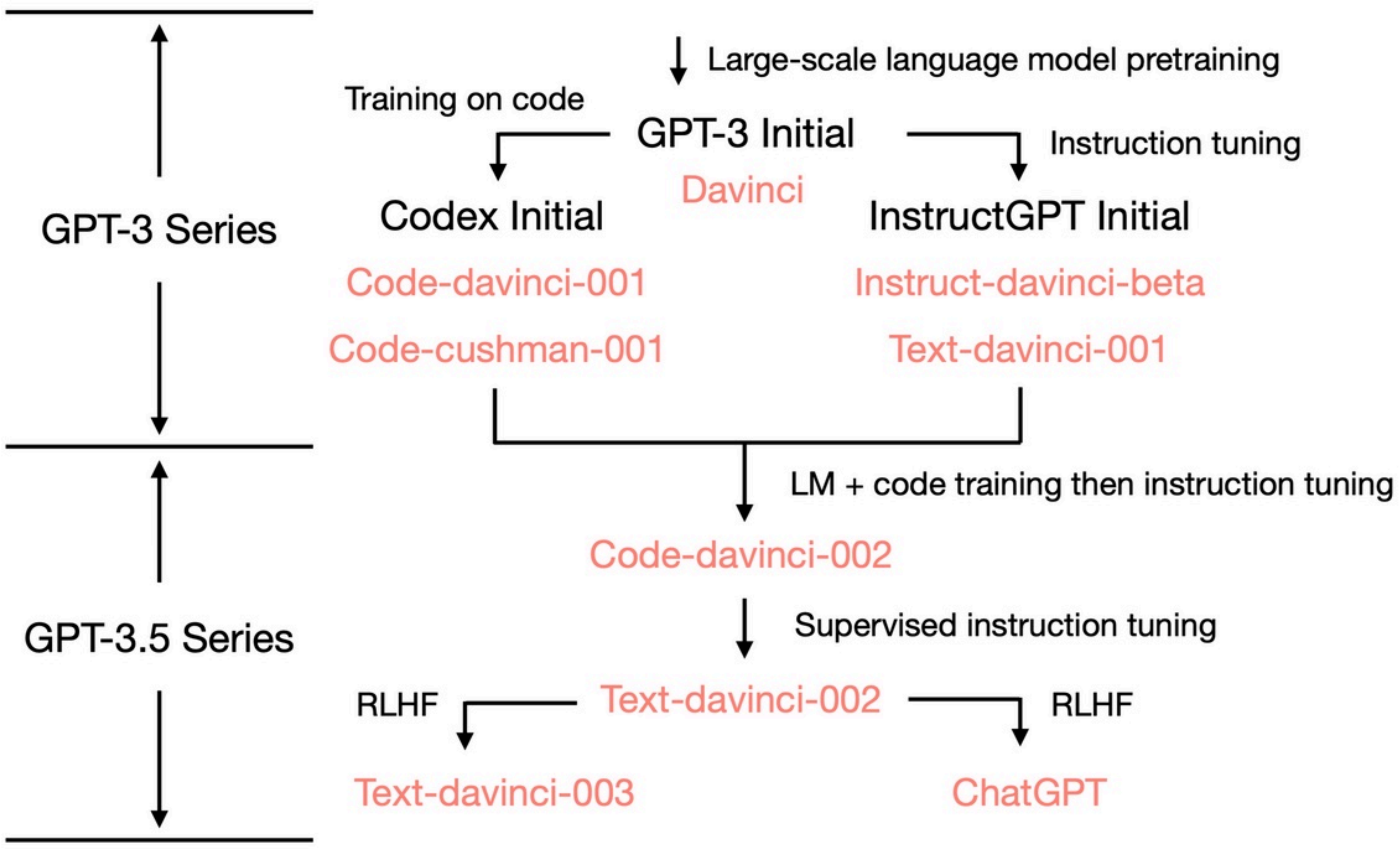
Chain-of-thought (CoT) prompting



(Wei et al., 2022): Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

The ChatGPT era:
Supervised instruction tuning and RLHF

From 2020's GPT-3 to 2022's ChatGPT



What's new?

- Code training
- Supervised instruction tuning
- RLHF = Reinforcement learning from human feedback

https://yaofu.notion.site/How-does-GPT-Obtain-its-Ability-Tracing-Emergent-Abilities-of-Language-Models-to-their-Sources-b9a57ac0fcf74f30a1ab9e3e36fa1dc1

Post-training pipeline of LLMs

Training language models to follow instructions with human feedback

Long Ouyang* **Jeff Wu*** **Xu Jiang*** **Diogo Almeida*** **Carroll L. Wainwright***

Pamela Mishkin* **Chong Zhang** **Sandhini Agarwal** **Katarina Slama** **Alex Ray**

John Schulman **Jacob Hilton** **Fraser Kelton** **Luke Miller** **Maddie Simens**

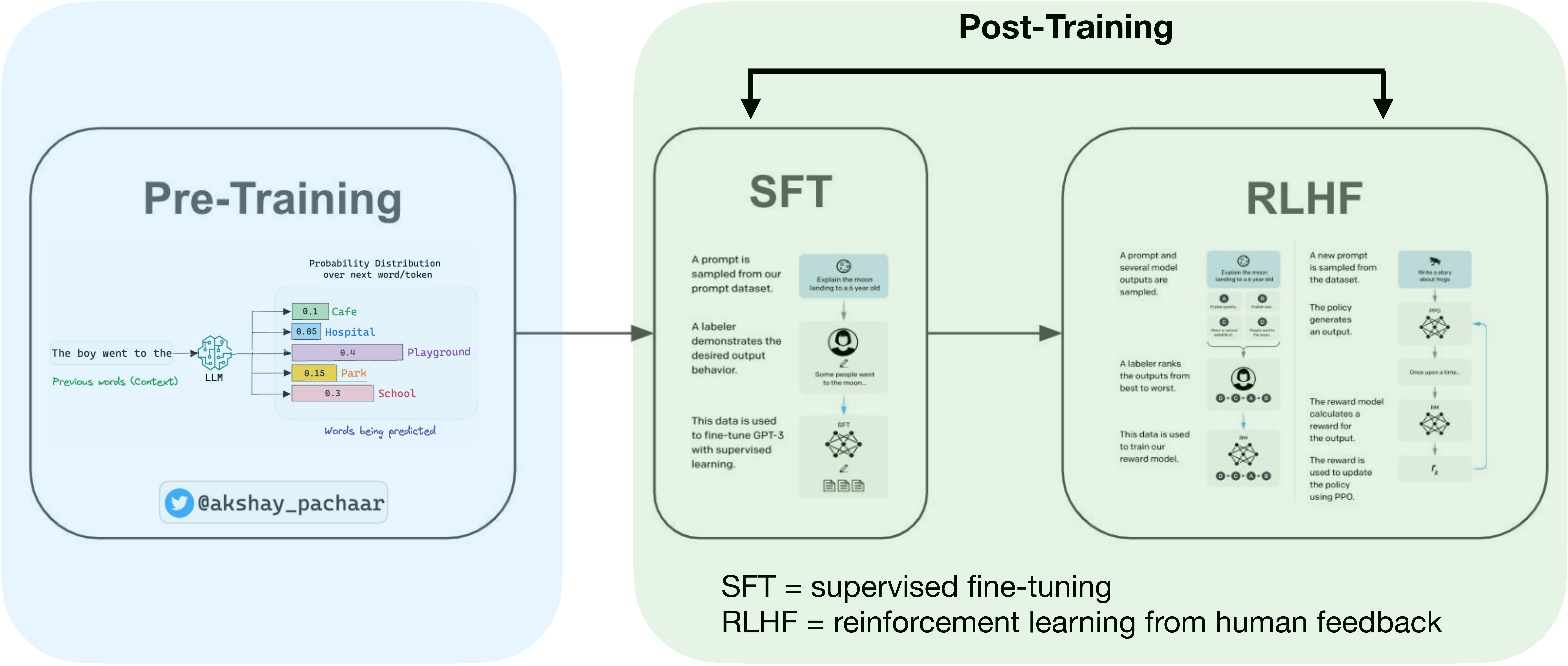
Amanda Askell† **Peter Welinder** **Paul Christiano***†

Jan Leike* **Ryan Lowe***

OpenAI

The InstructGPT paper

Post-training pipeline of LLMs



SFT = supervised fine-tuning
RLHF = reinforcement learning from human feedback

(Ouyang et al., 2022): Training language models to follow instructions with human feedback

Supervised fine-tuning (SFT)

- **Instruction data**: (**instruction**, **response**)
- **Objective**: next-token prediction conditioned on instruction

Can you write a short introduction about the relevance of the term "monopsony" in economics? Please use examples related to potential monopsonies in the labour market and cite relevant research.

"Monopsony" refers to a market structure where there is only one buyer for a particular good or service. In economics, this term is particularly relevant in the labor market, where a monopsony employer has significant power over the wages and working conditions of their employees. The presence of a monopsony can result in lower wages and reduced employment opportunities for workers, as the employer has little incentive to increase wages or provide better working conditions ……. understand the extent and impact of monopsonies on the economy and to develop policies to address this issue. References: Bivens, J., & Mishel, L. (2013). The Pay of Corporate Executives and Financial Professionals as Evidence of Rents in Top 1 Percent Incomes. Journal of Economic Perspectives, 27(3), 57-78.

Supervised fine-tuning (SFT)

Use-case	Prompt
Brainstorming	List five ideas for how to regain enthusiasm for my career
Generation	Write a short story where a bear goes to the beach, makes friends with a seal, and then returns home.
Rewrite	This is the summary of a Broadway play: """ {summary} """ This is the outline of the commercial for that play: """

Use-case	(%)
Generation	45.6%
Open QA	12.4%
Brainstorming	11.2%
Chat	8.4%
Rewrite	6.6%
Summarization	4.2%
Classification	3.5%
Other	3.5%
Closed QA	2.6%
Extract	1.9%

SFT data: only ~13k (written by labeller, not public)

(Ouyang et al., 2022): Training language models to follow instructions with human feedback

Collection of supervised fine-tuning data

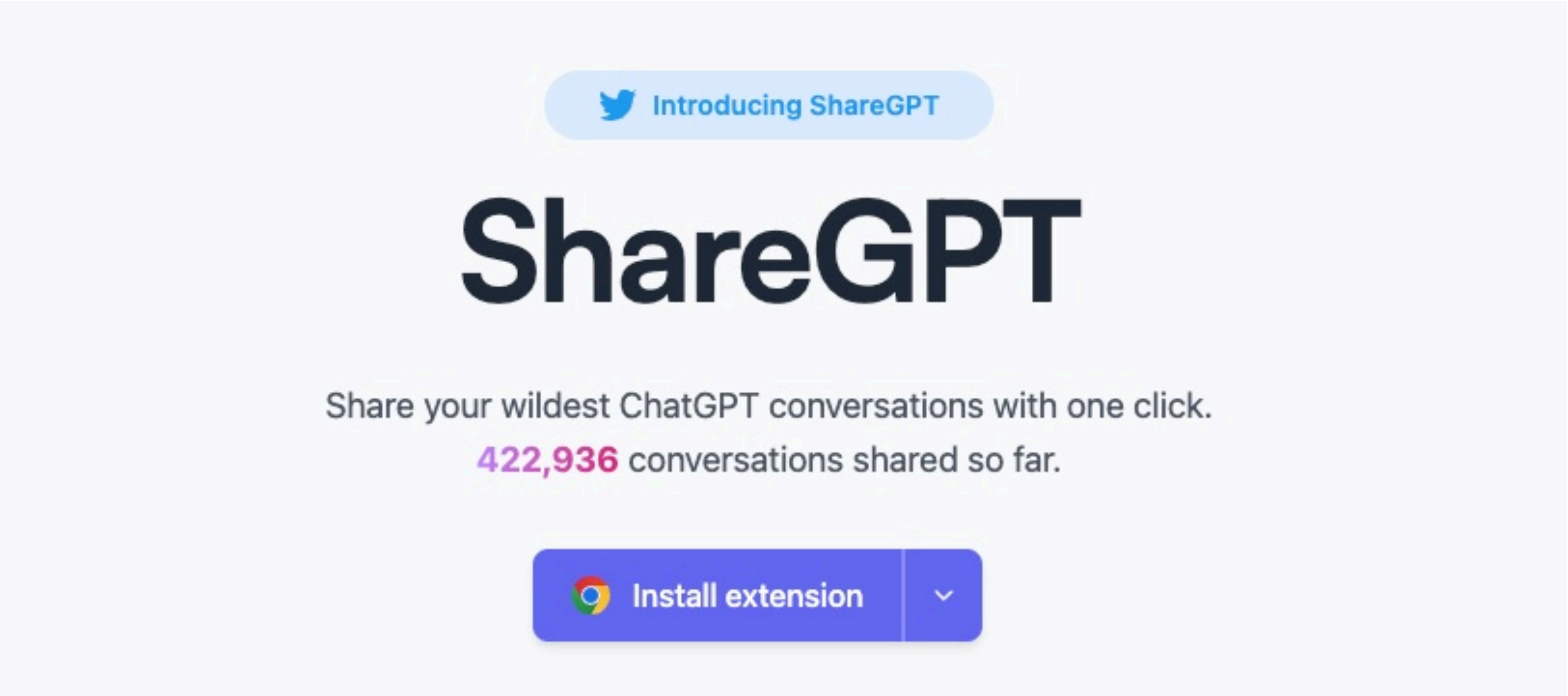
- Repurposed from existing supervised datasets
- Human-written instructions and responses

Super-NaturalInstructions (Wang et al., 2022)
Also: FLAN, T0++

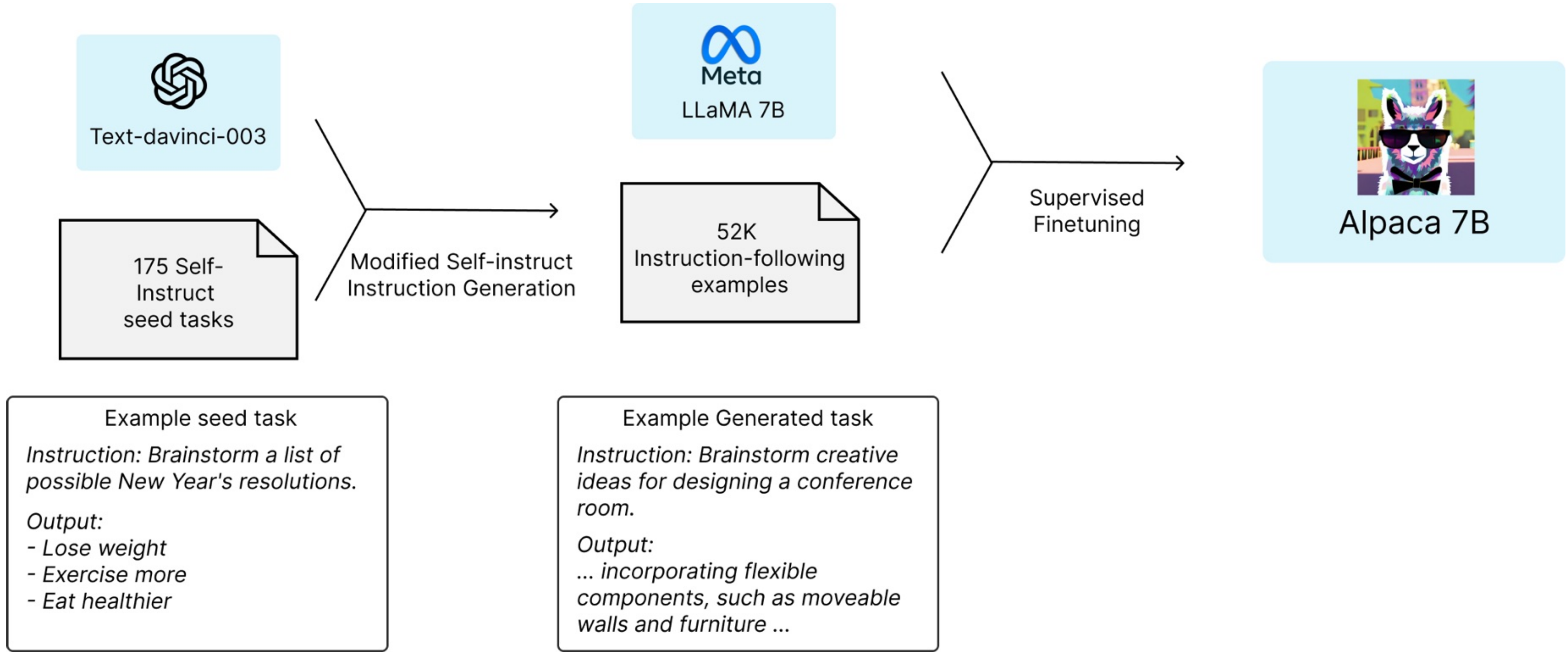
OpenAssistant
(Kopf et al., 2023)

Collection of supervised fine-tuning data

- Response distilled from GPT models
- Instructions can be generated by GPT models too, e.g., Self-Instruct (Wang et al., 2023)



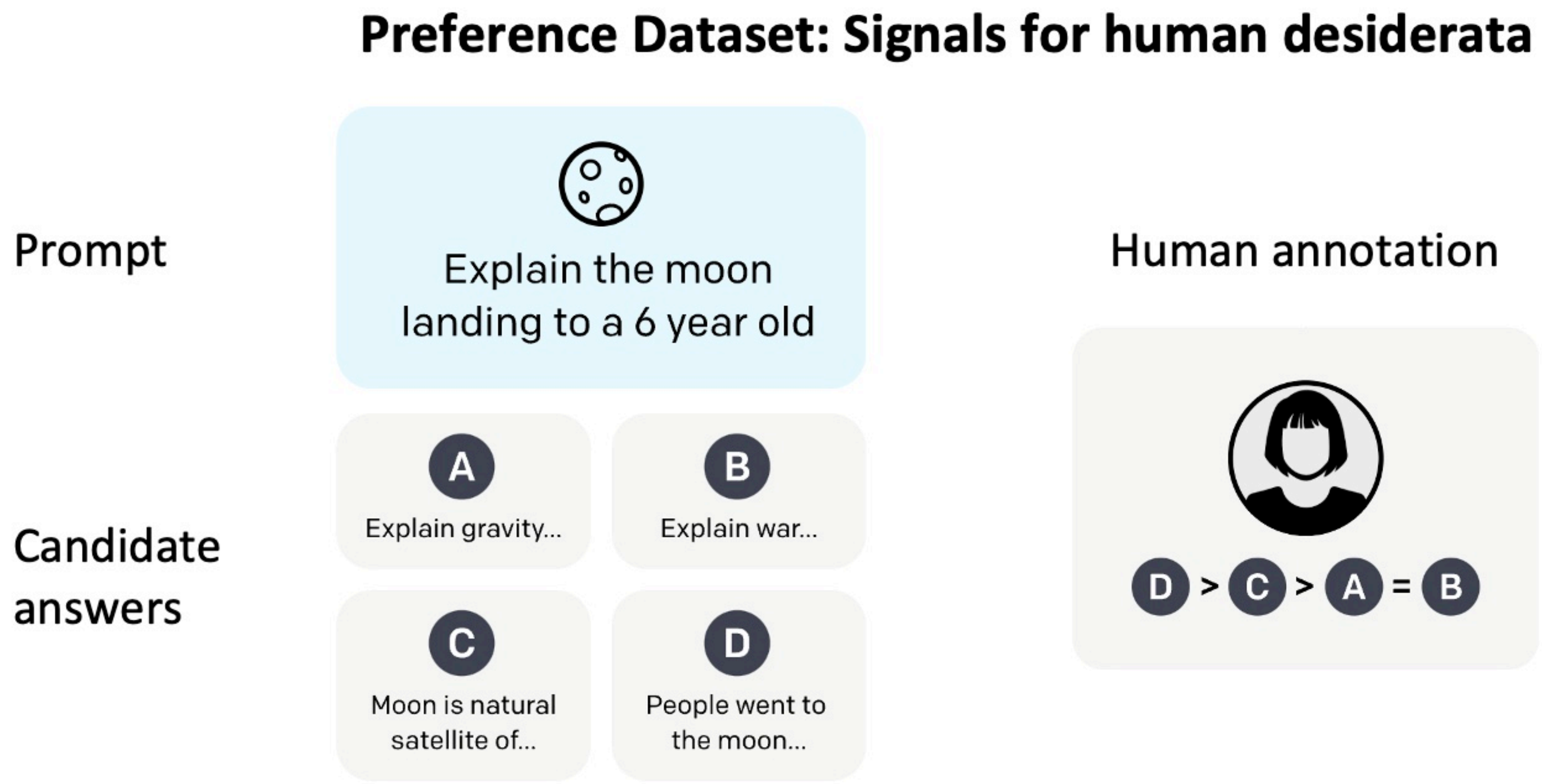
https://sharegpt.com/



Stanford Alpaca

Learning from human feedback

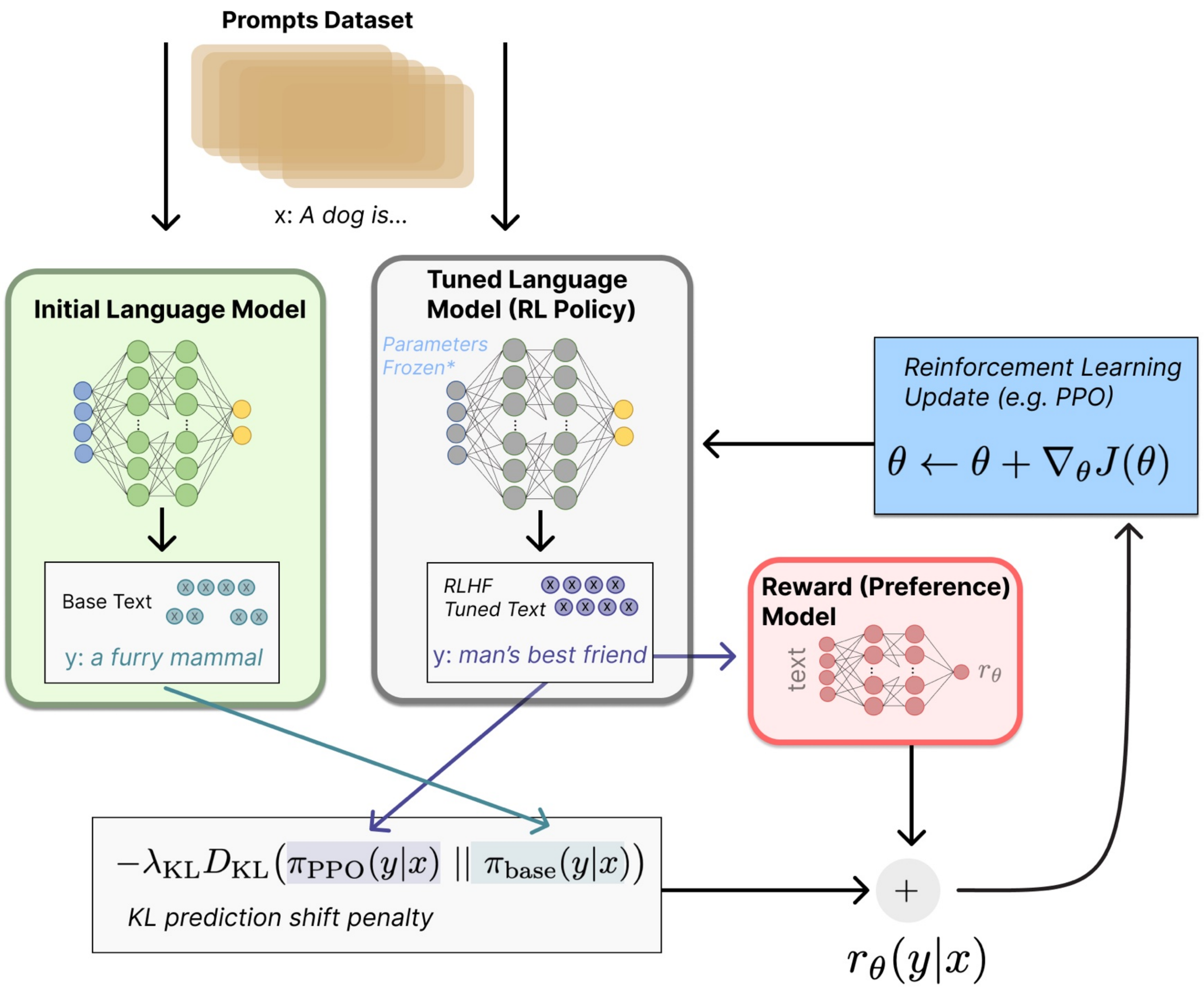
- **Preference data**: (instruction, winning response, losing response)



InstructGPT: 33k prompts, each with K (4~9) corresponding SFT model completions ranked by labellers

(Ouyang et al., 2022) Training language models to follow instructions with human feedback

Reinforcement learning from human feedback

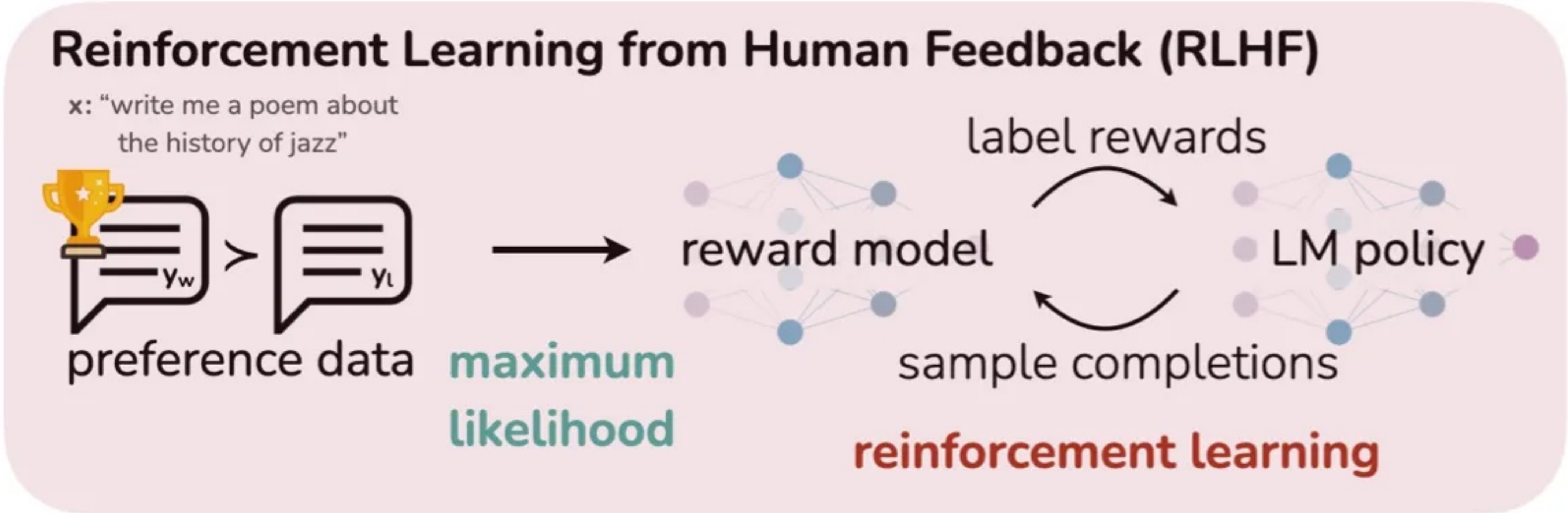


Multiple stages in training

- Reward model training
- Sampling from policy model
- Policy model update

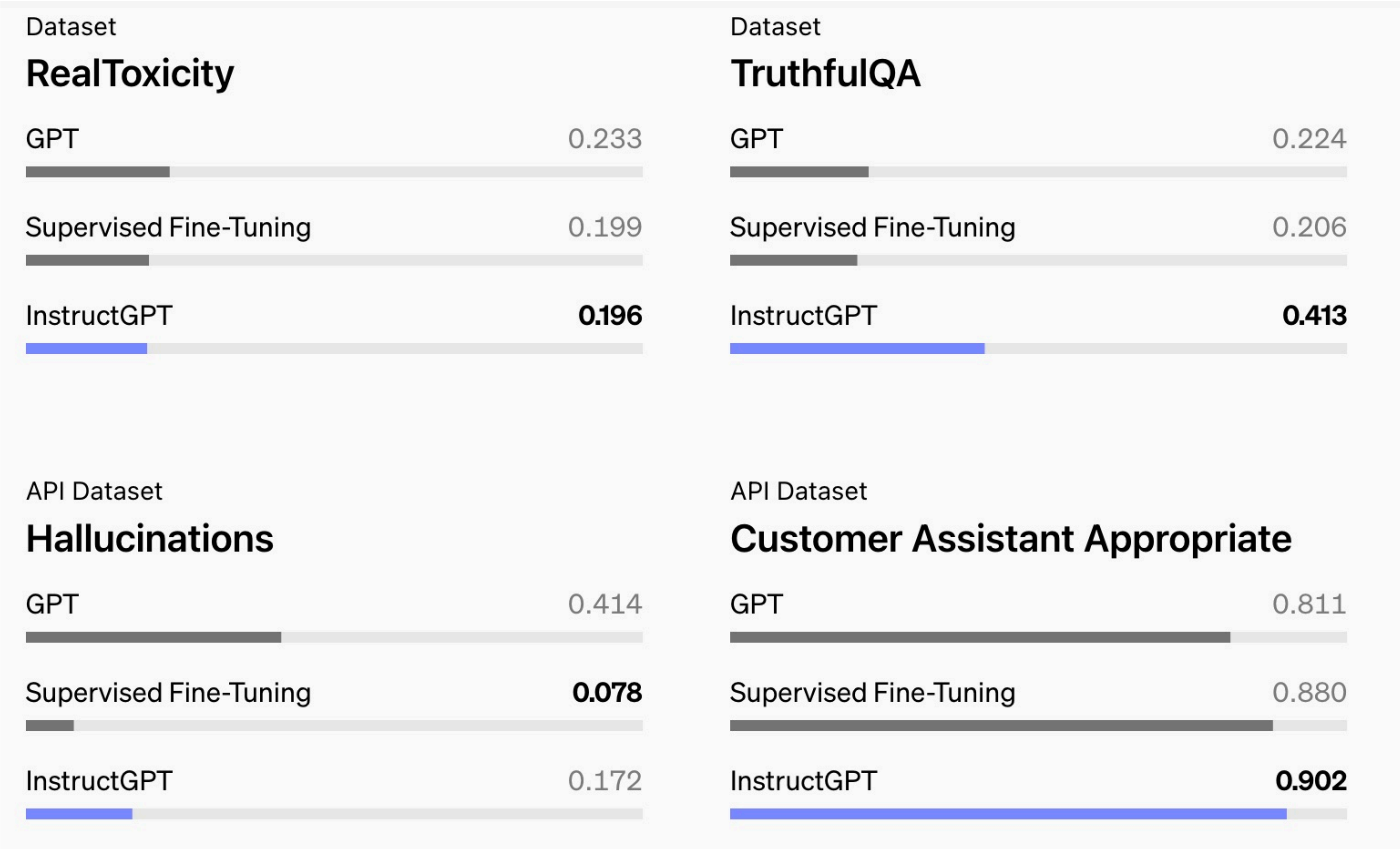
Multiple models involved

- Reward model
- Policy model
- Reference model



https://huggingface.co/blog/rlhf

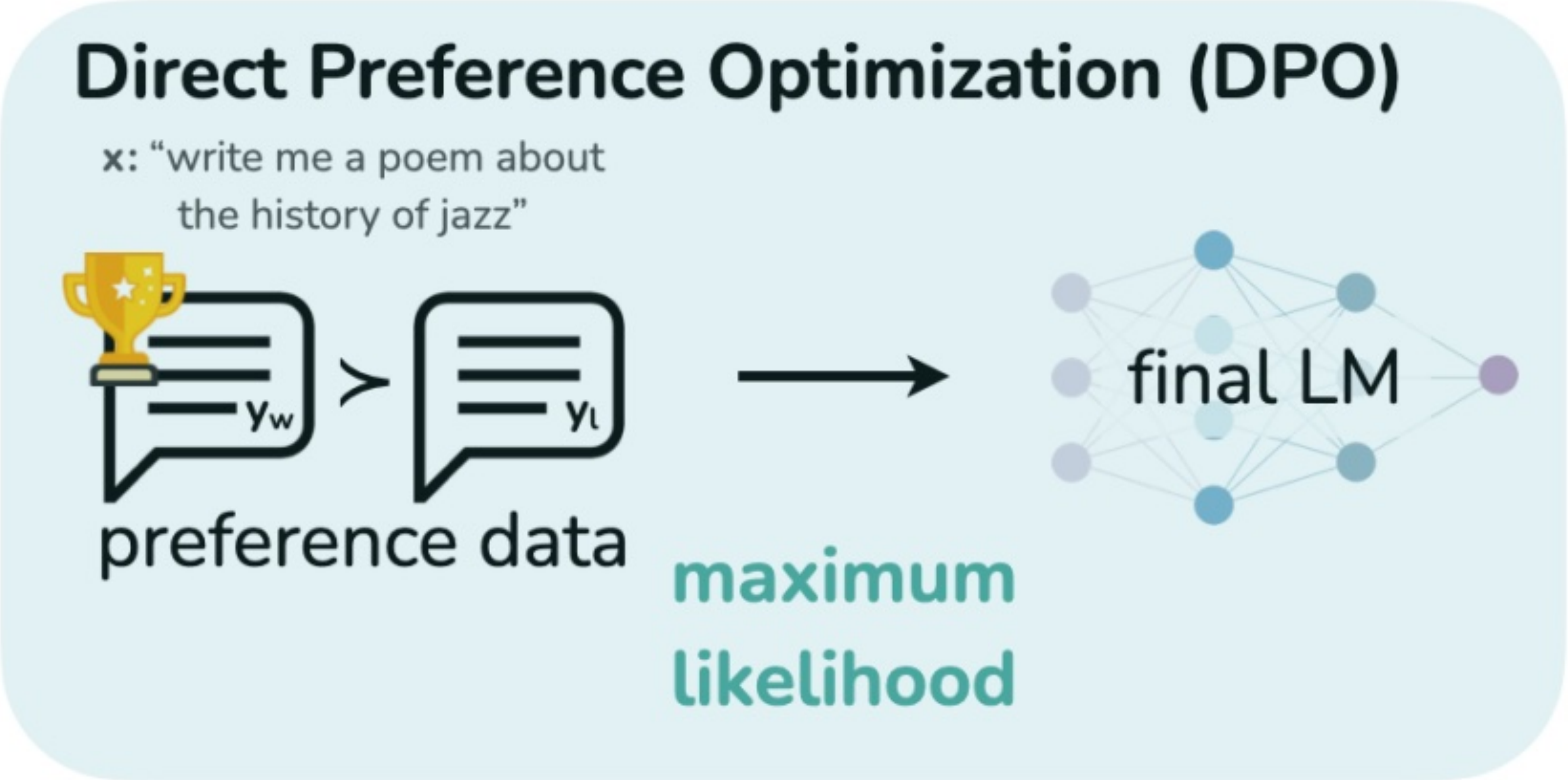
GPT-3 vs InstructGPT



https://openai.com/index/instruction-following/

Direct preference optimization (DPO)

Instead of training an explicit reward model, express reward in the form of policy model:



Implicit reward expression:

$$r(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x)$$

Bradley-Terry ranking objective:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x, y_w, y_l)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$

DPO objective:

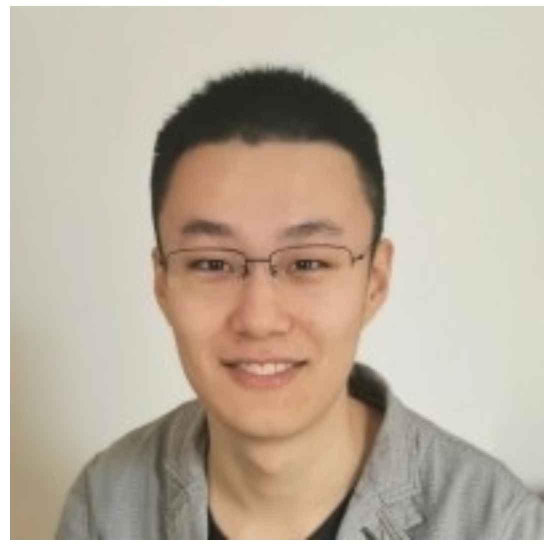
$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]$$

(Rafailov et al., 2023) Direct Preference Optimization: Your Language Model is Secretly a Reward Model

Simple preference optimization (SimPO)

$$\mathcal{L}_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\mathrm{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\mathrm{ref}}(y_l \mid x)}\right)\right]$$

$$\mathcal{L}_{\mathrm{SimPO}}(\pi_\theta) = -\mathbb{E}\left[\log \sigma\left(\frac{\beta}{|y_w|} \log \pi_\theta(y_w \mid x) - \frac{\beta}{|y_l|} \log \pi_\theta(y_l \mid x) - \gamma\right)\right]$$



- A simple **length-normalized** reward (reference-free!):

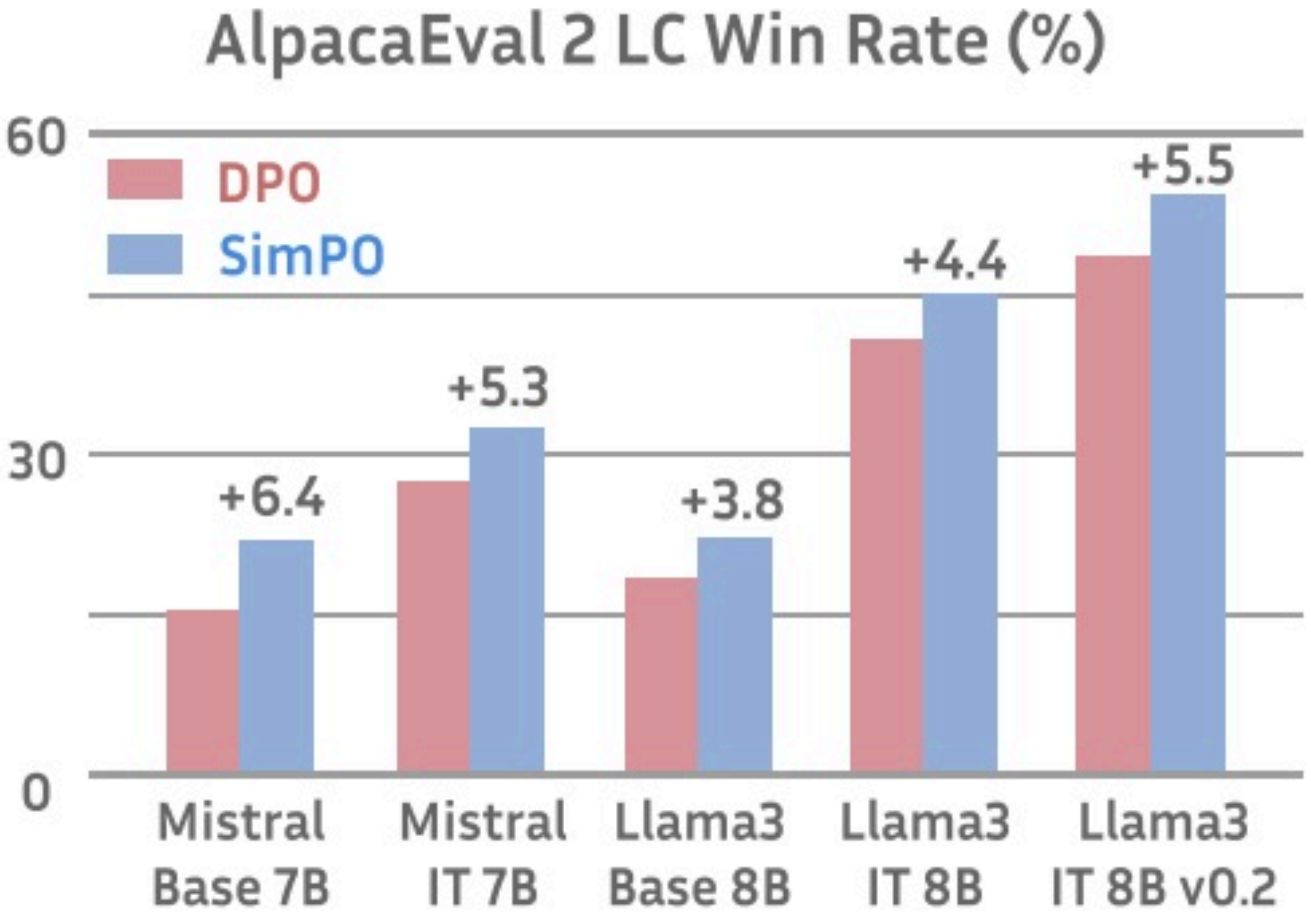
$$r_{\mathrm{SimPO}}(x, y) = \frac{\beta}{|y|} \log \pi_\theta(y \mid x) = \frac{\beta}{|y|} \sum_{i=1}^{|y|} \log \pi_\theta(y_i \mid x, y_{<i})$$

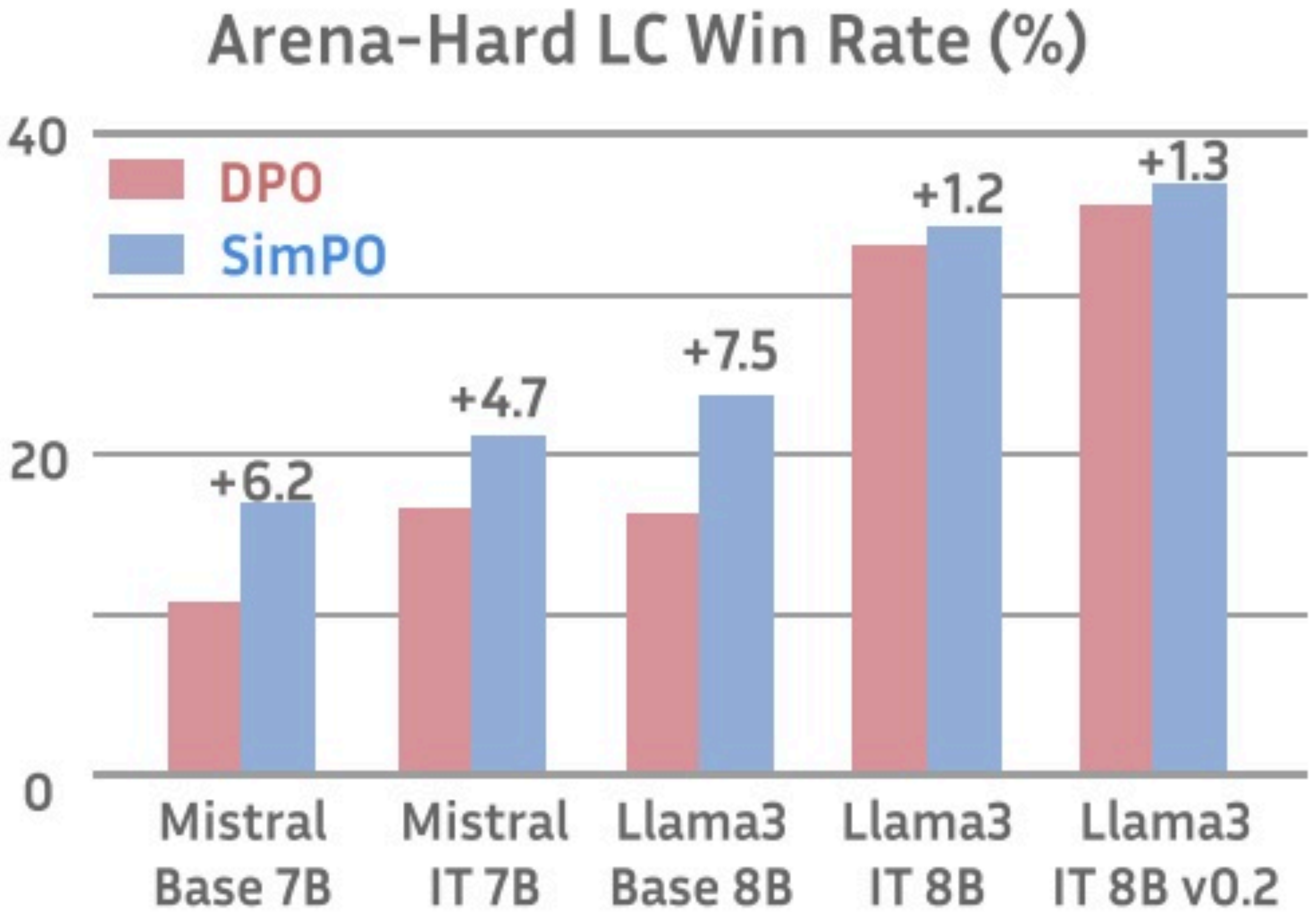
- Introducing target reward margin in Bradley-Terry objective:

$$p(y_w \succ y_l \mid x) = \sigma\left(r(x, y_w) - r(x, y_l) - \boxed{\gamma}\right)$$

(Meng et al., 2024) SimPO: Simple Preference Optimization with a Reference-Free Reward

Simple preference optimization (SimPO)





(Meng et al., 2024) SimPO: Simple Preference Optimization with a Reference-Free Reward