

# Deep Learning Basics

Lecture 8: Transformer

최성준 (고려대학교 인공지능학과)

# Sequential Model

- What makes sequential modeling a hard problem to handle?



Original sequence



Trimmed sequence



Omitted sequence

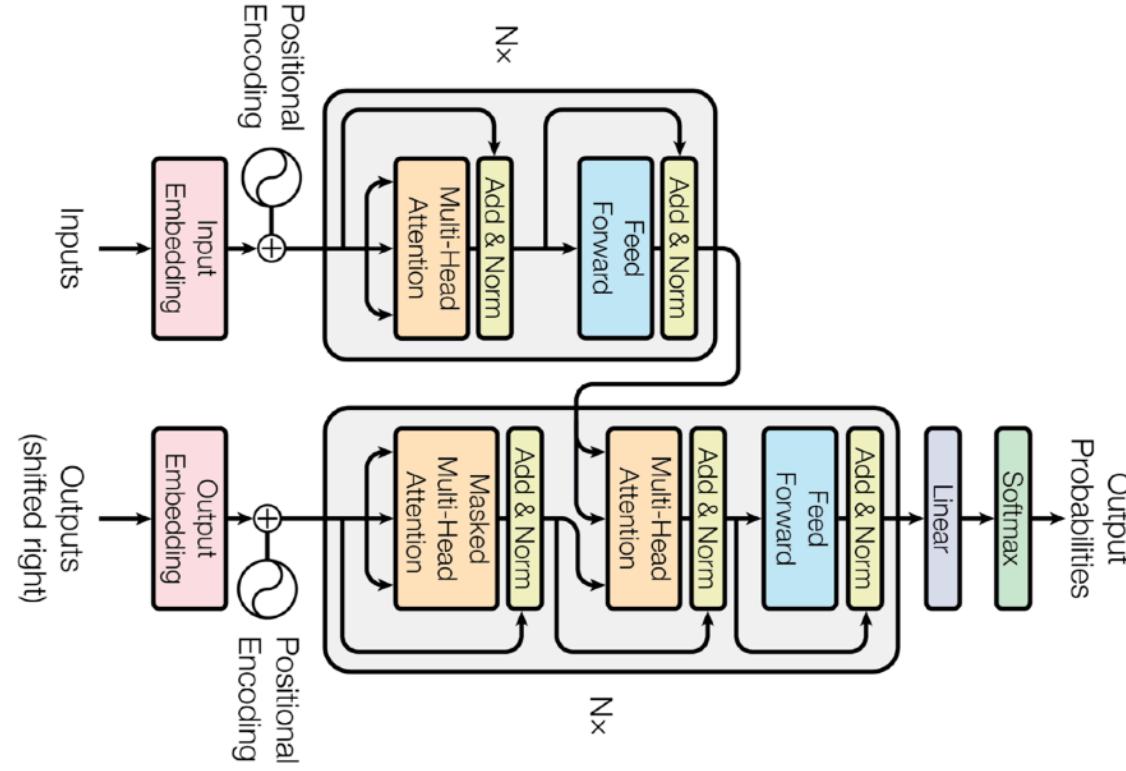


Permuted sequence

# Transformer

Attention is All You Need, NIPS, 2017

# Transformer



- Transformer is the first sequence transduction model based entirely on attention.



# Transformer

- From a bird's-eye view, this is what the Transformer does for machine translation tasks.



Jay Alammar

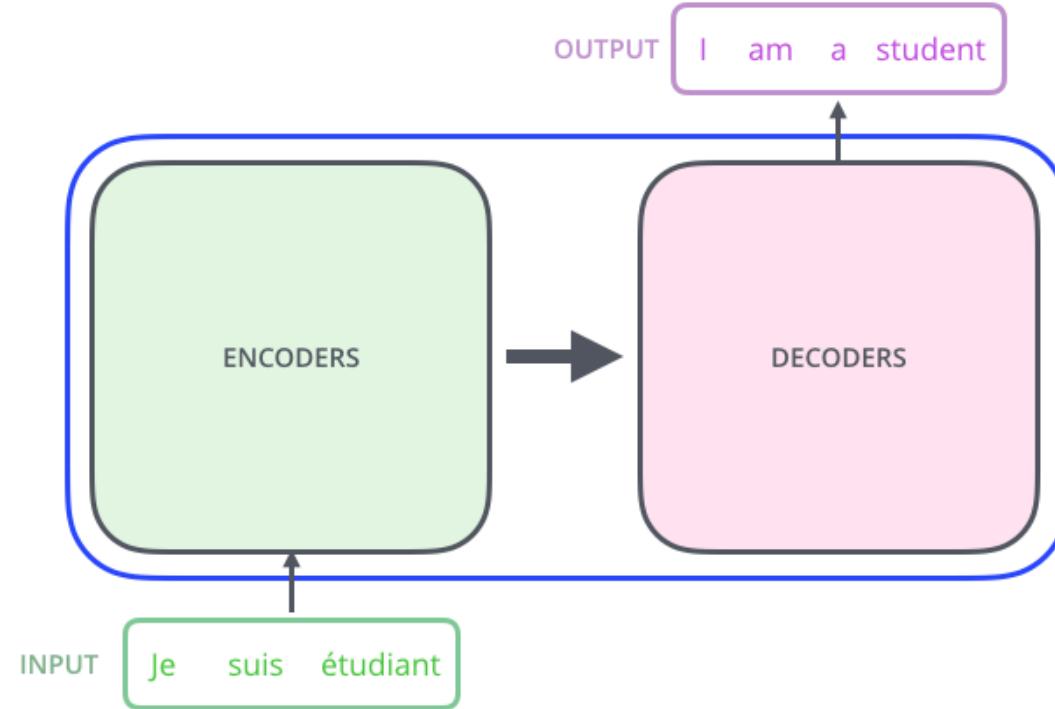
Visualizing machine learning one concept at a time.

@JayAlammar on Twitter. [YouTube Channel](#)

<http://jalammar.github.io/illustrated-transformer/>

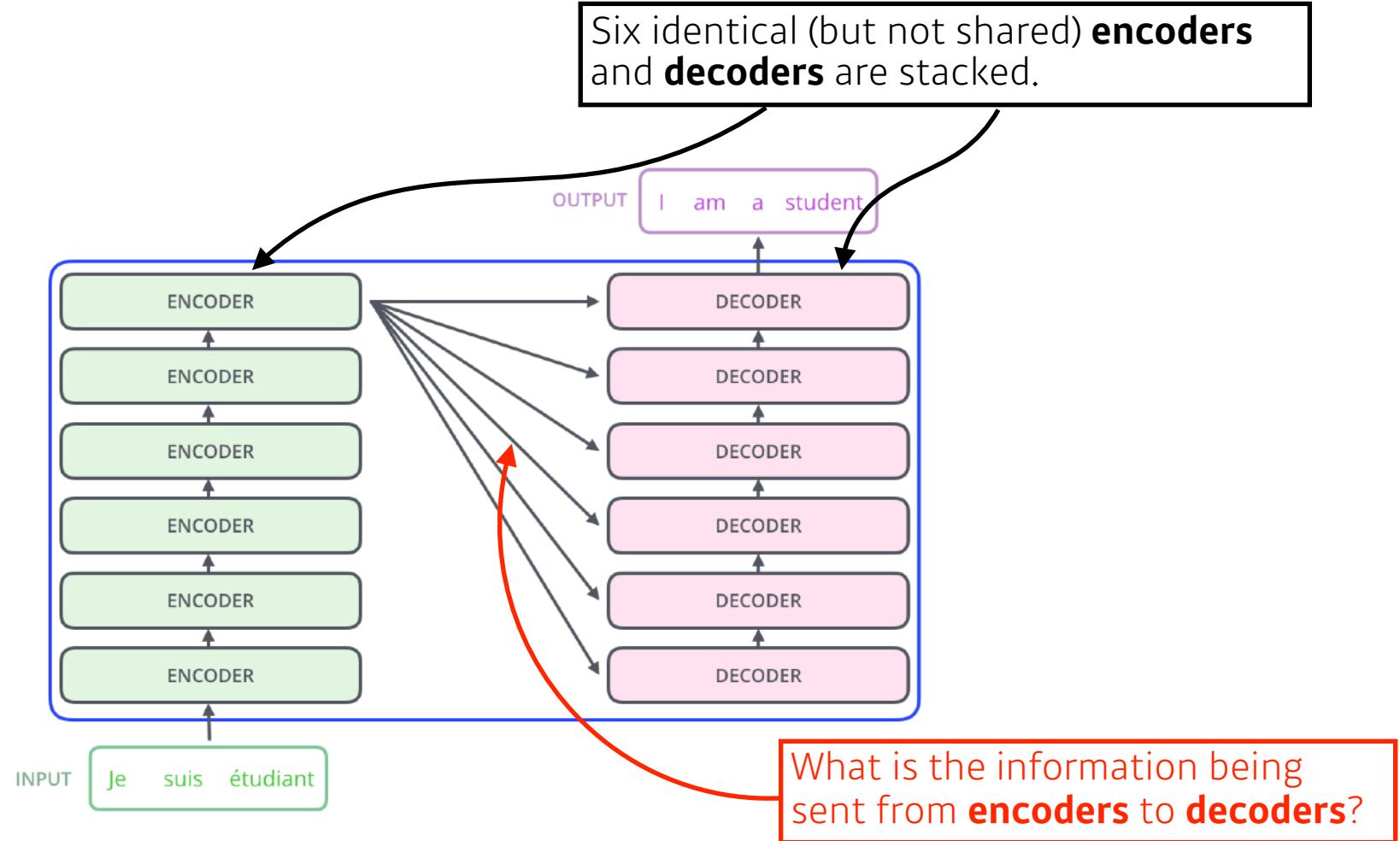
# Transformer

---

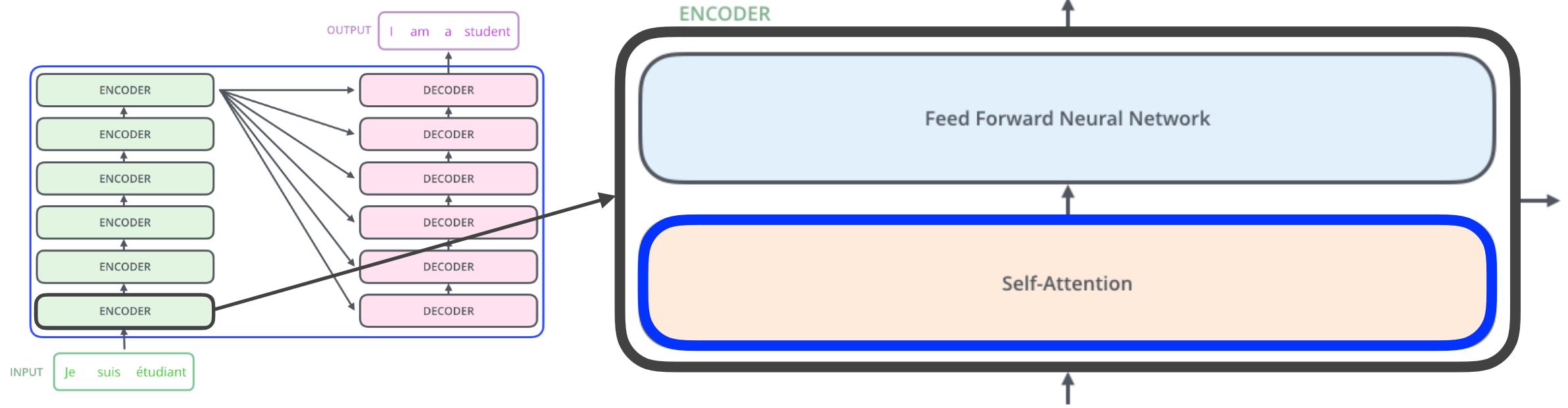


- If we glide down a little bit, this is what the Transformer does.

# Transformer



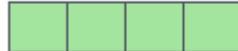
# Transformer



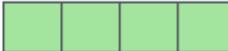
- The **Self-Attention** in both encoder and decoder is the cornerstone of Transformer.

# Transformer

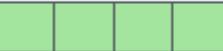
---

$x_1$  

Je

$x_2$  

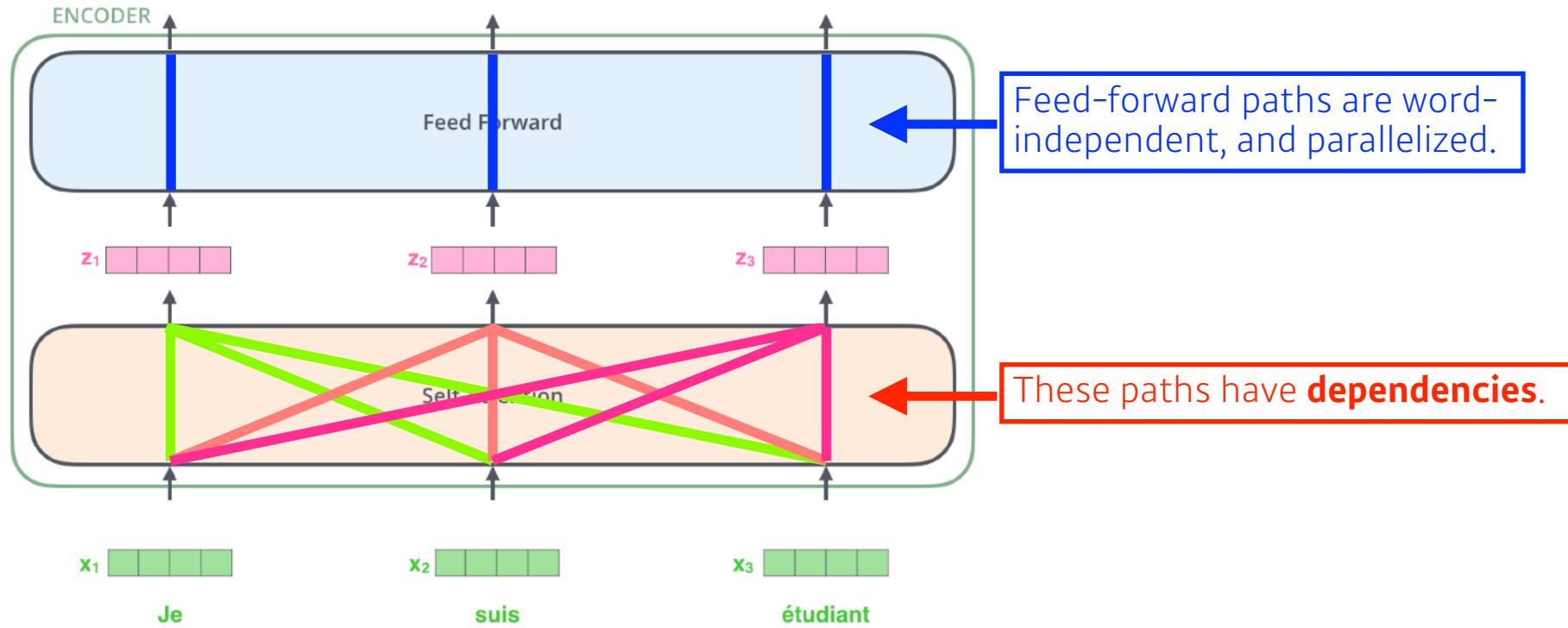
suis

$x_3$  

étudiant

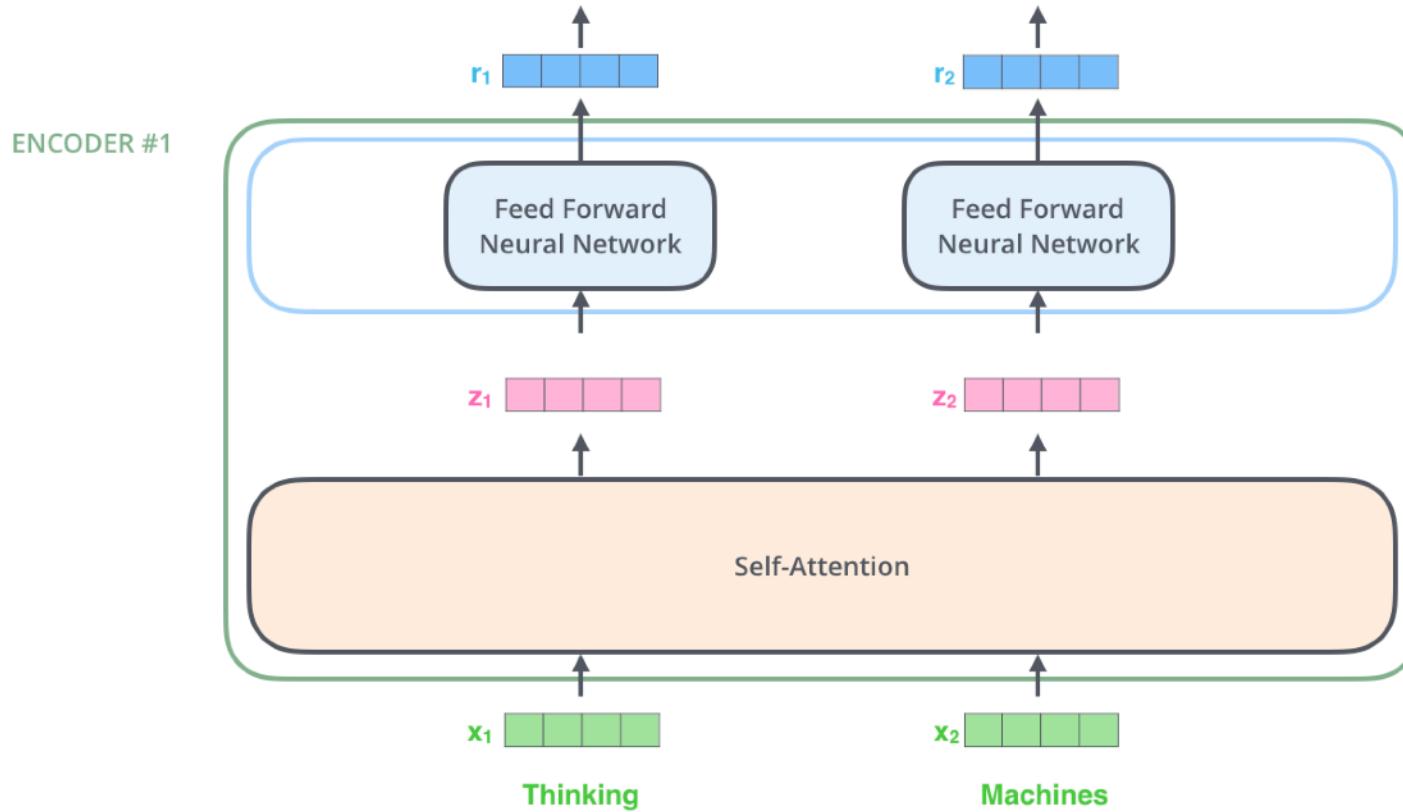
- ➊ First, we represent each word with some embedding vectors.

# Transformer



- Then, Transformer encodes each word to feature vectors with Self-Attention.

# Transformer



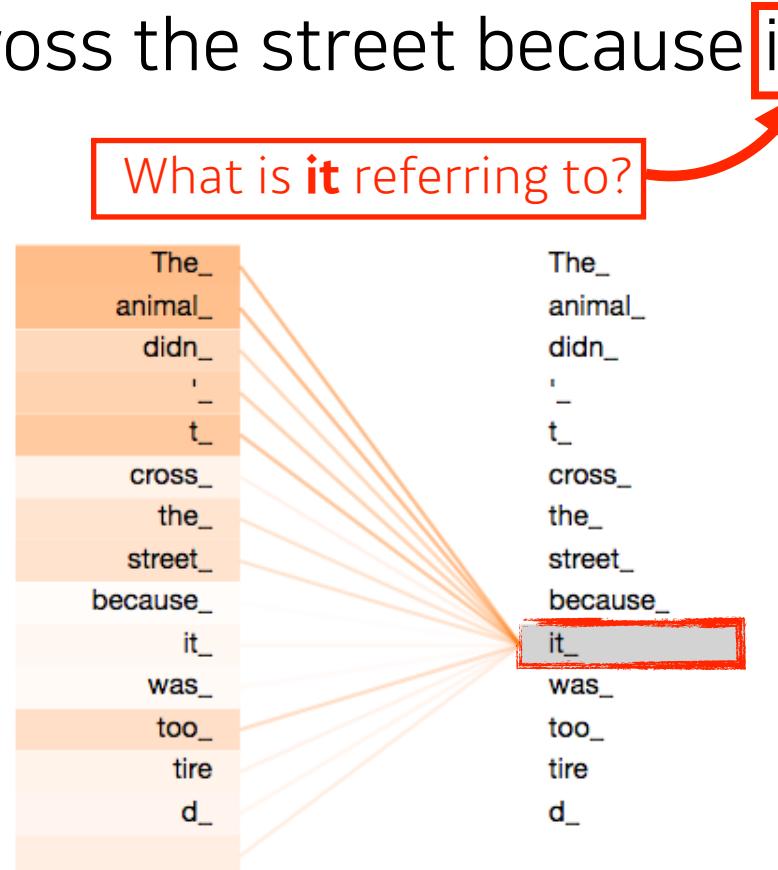
- Suppose we are encoding two words:
  - Thinking and Machines.



# Transformer

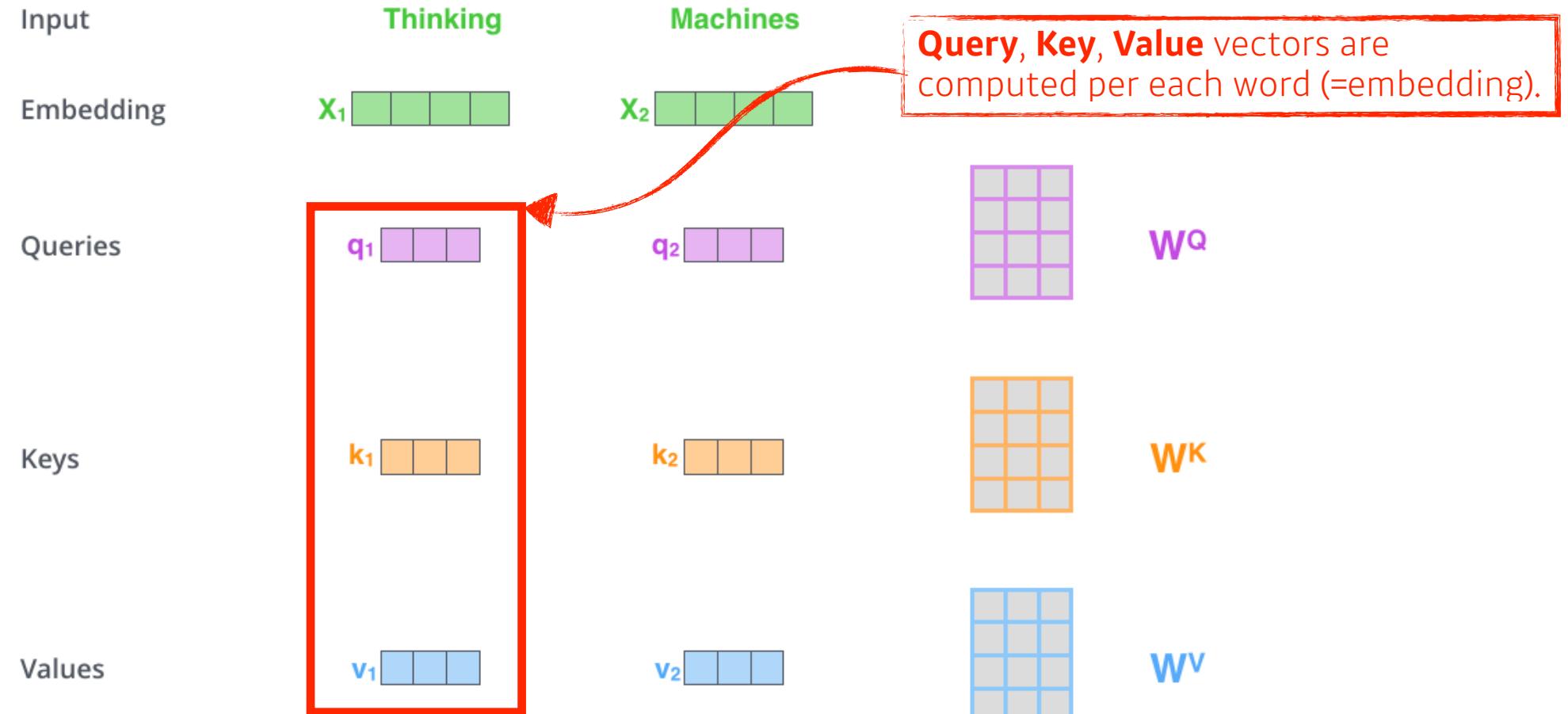
- Self-Attention at a high level

- The animal didn't cross the street because **it** was too tired.

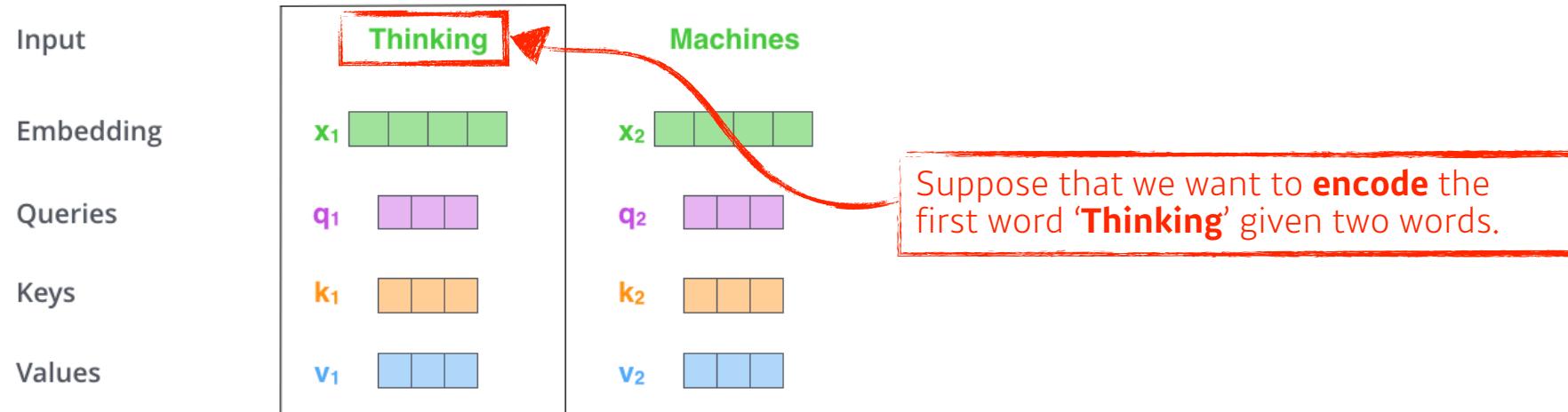


<http://jalammar.github.io/illustrated-transformer/>

# Transformer

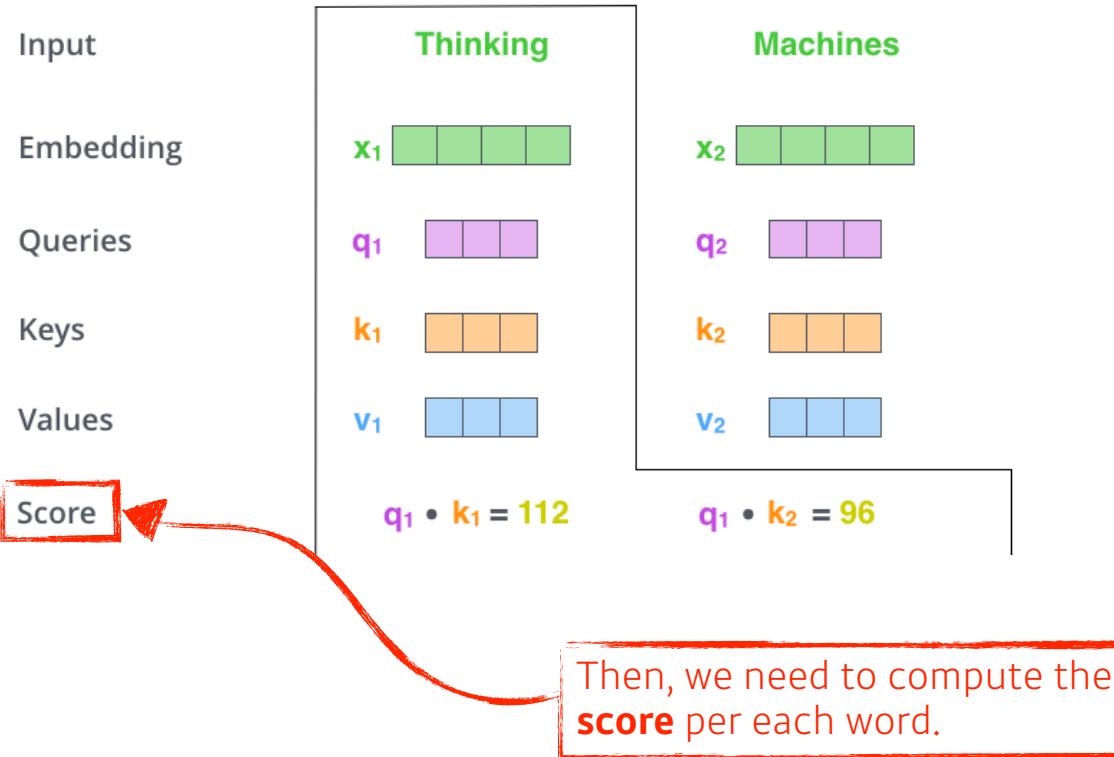


# Transformer



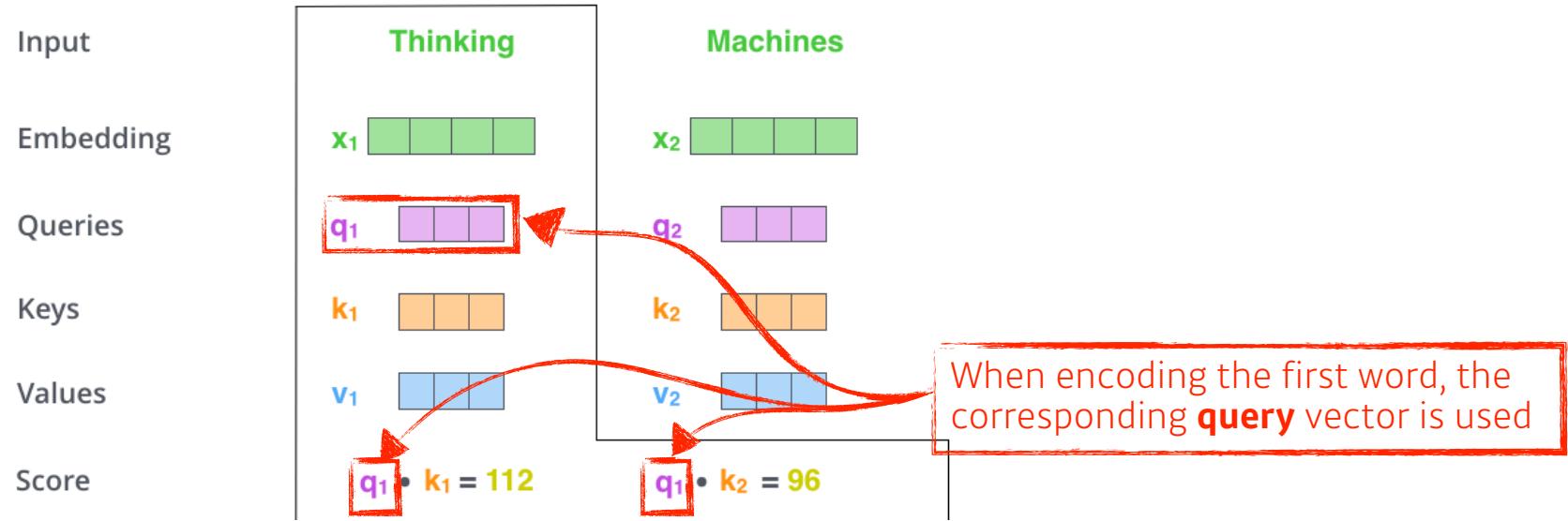
- Suppose we are encoding the first word: '**Thinking**' given '**Thinking**' and '**Machines**'.

# Transformer



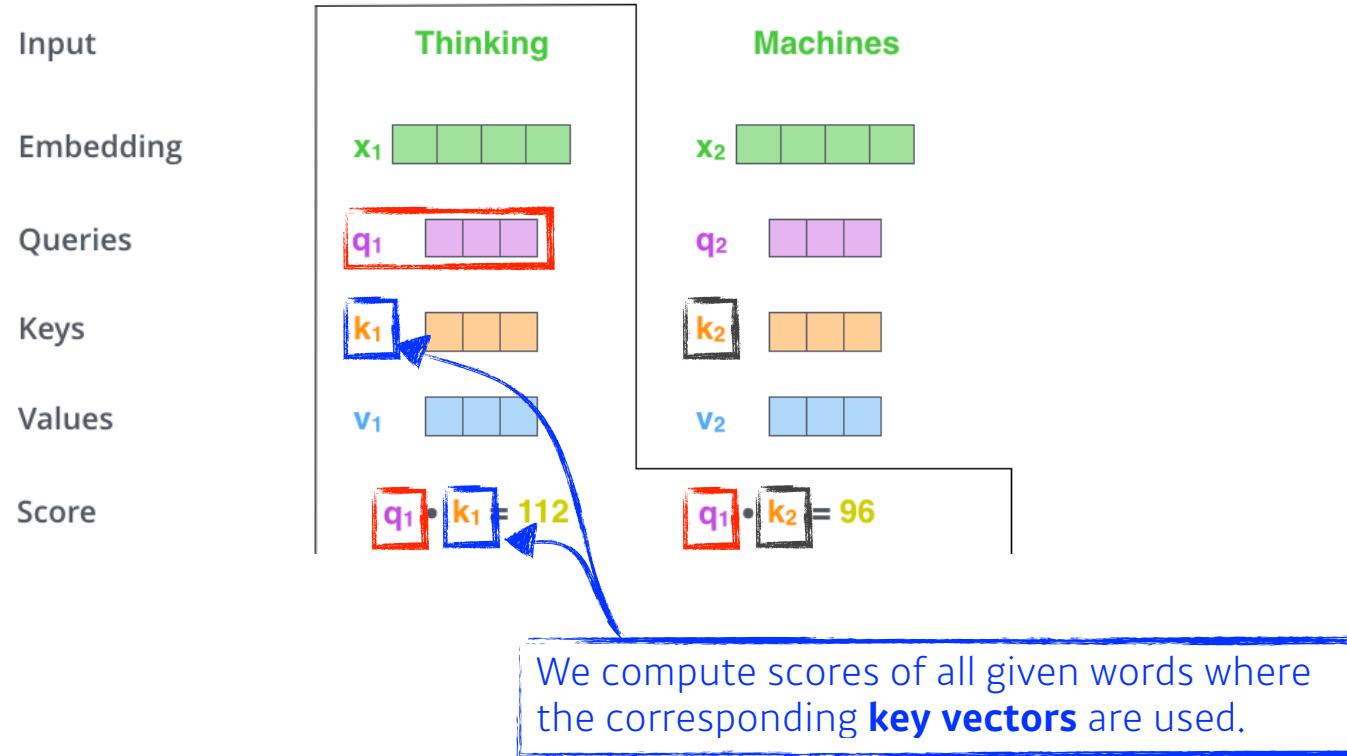
- Suppose we are encoding the first word: 'Thinking' given 'Thinking' and 'Machines'.

# Transformer



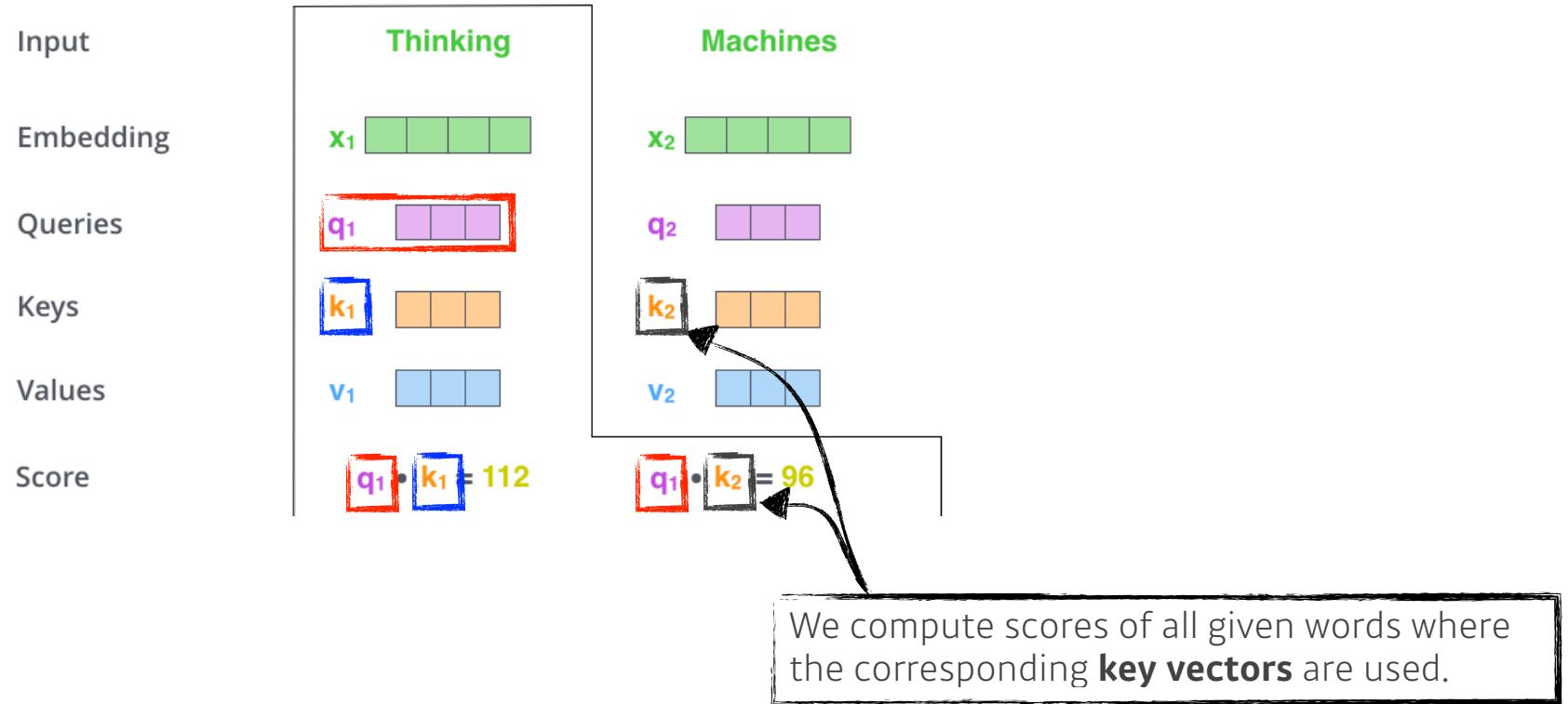
- Suppose we are encoding the first word: 'Thinking' given 'Thinking' and 'Machines'.

# Transformer



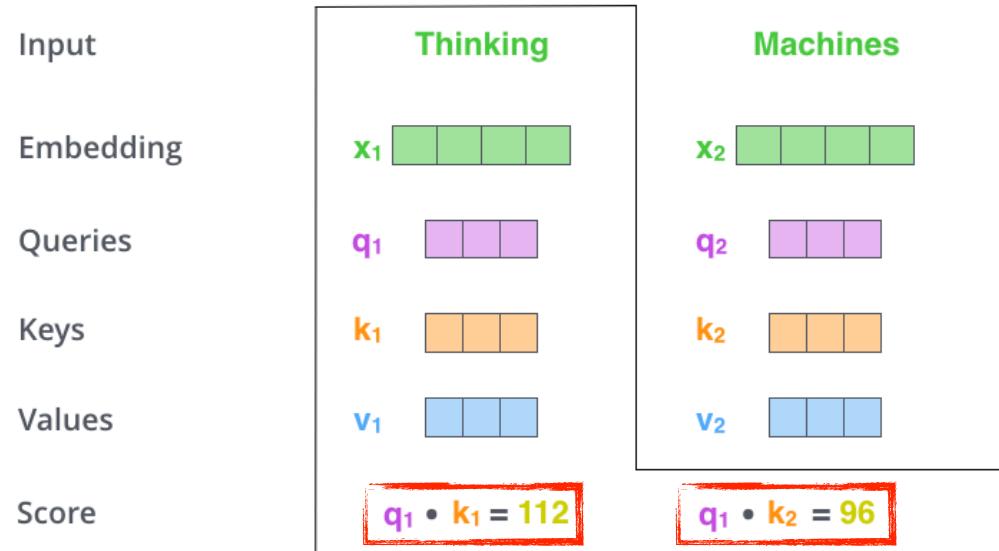
- Suppose we are encoding the first word: 'Thinking' given 'Thinking' and 'Machines'.

# Transformer



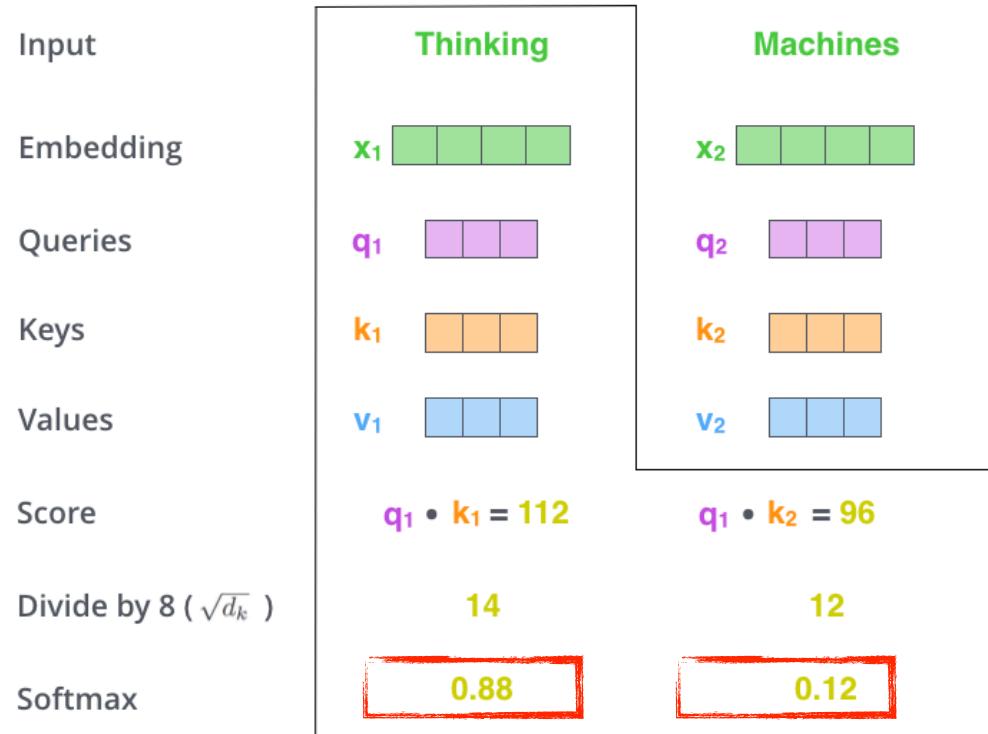
- Suppose we are encoding the first word: 'Thinking' given 'Thinking' and 'Machines'.

# Transformer



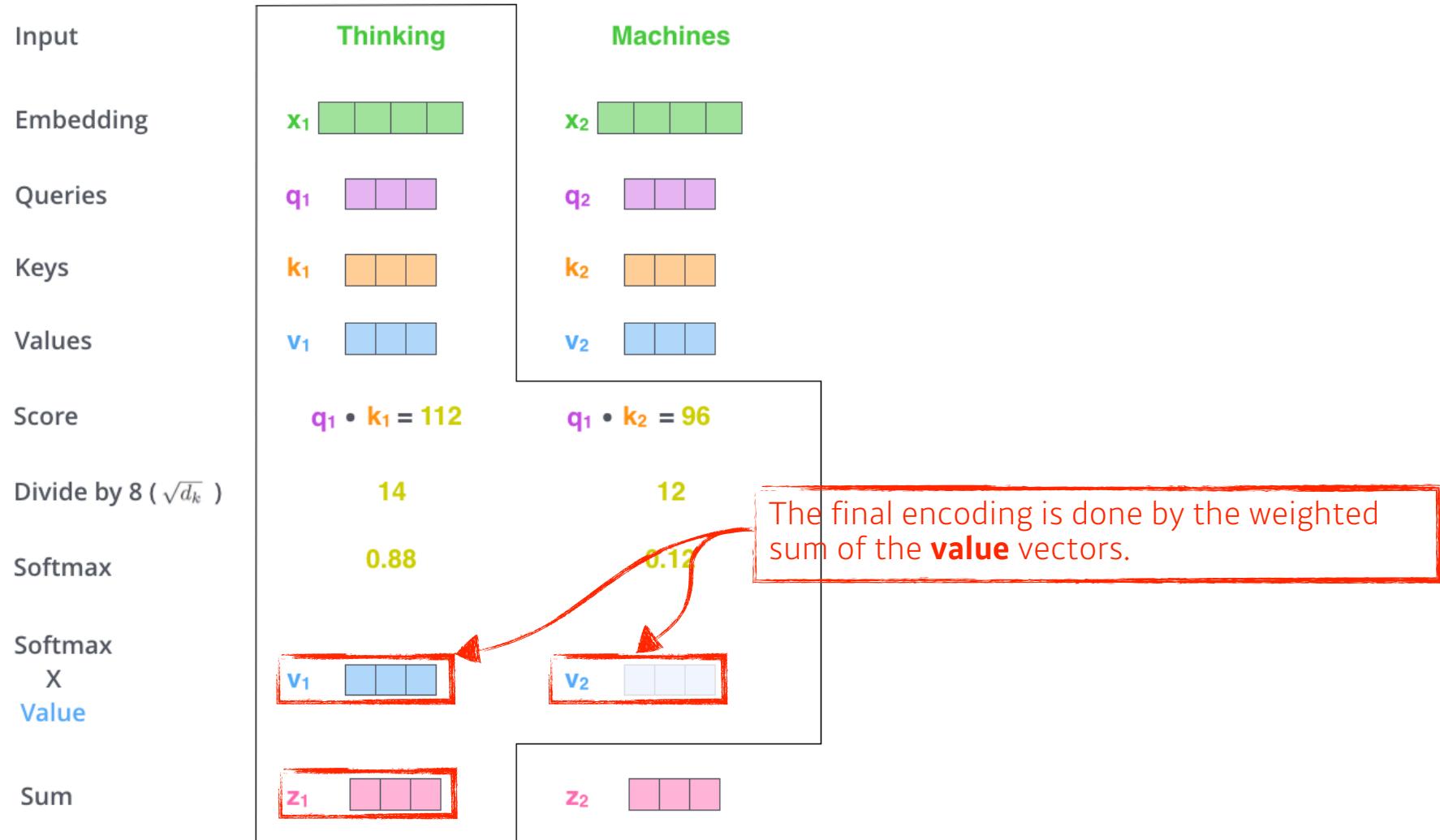
- We compute the **scores** of each word with respect to 'Thinking'.

# Transformer

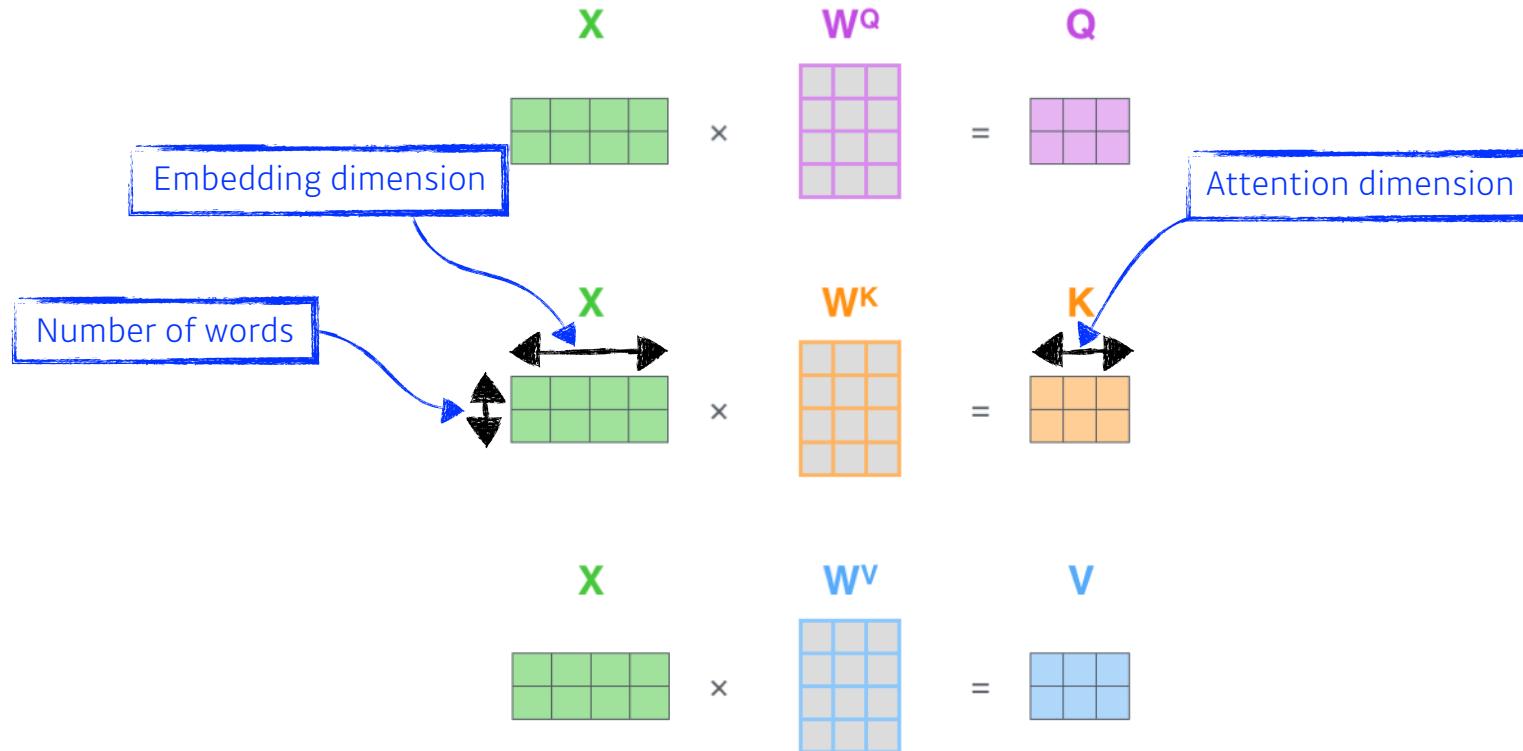


- Then, we compute the **attention weights** by scaling followed by softmax.

# Transformer



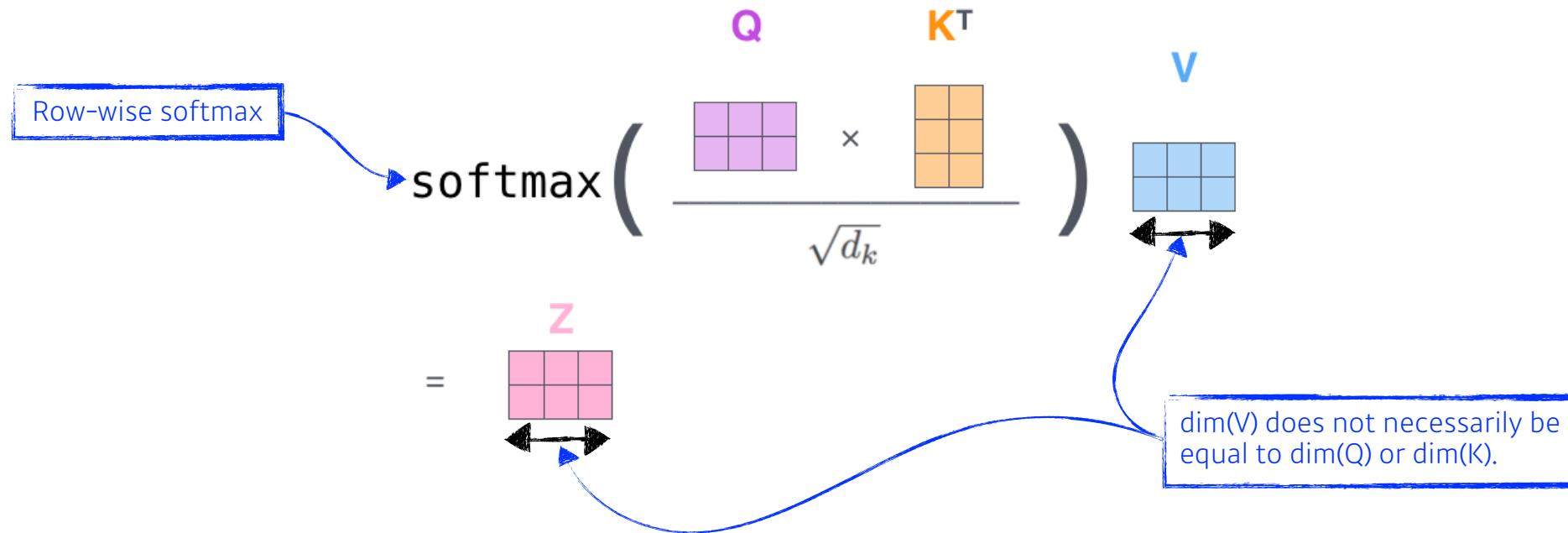
# Transformer



- Calculating Q, K, and V from X in a matrix form.

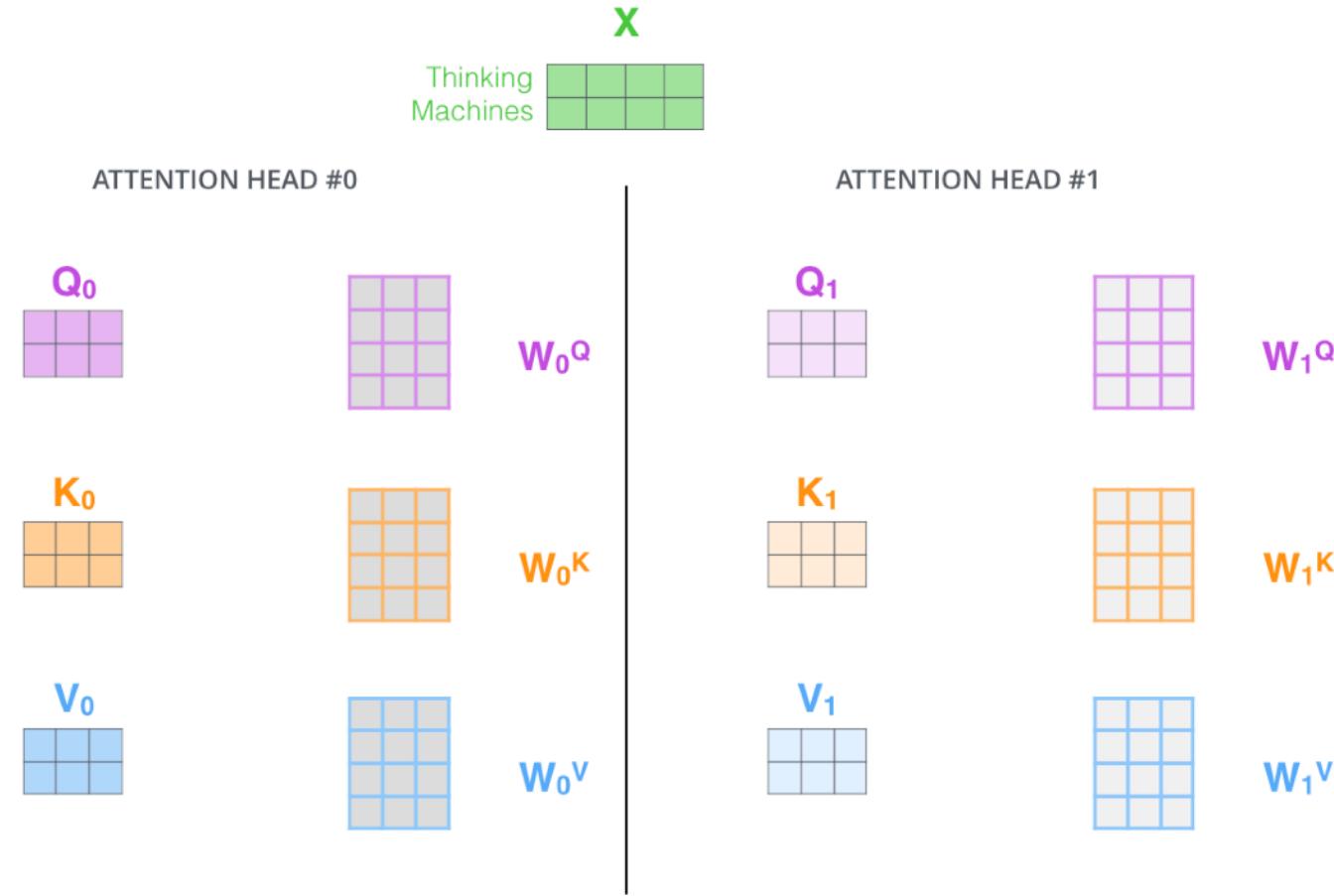
# Transformer

---



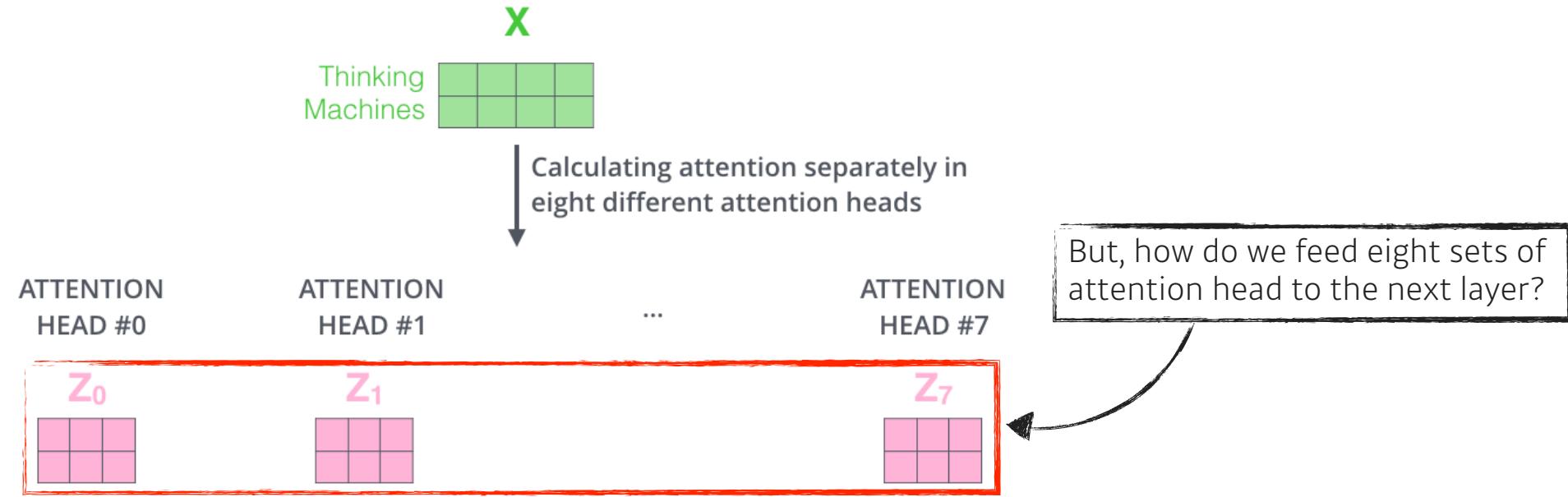
- Calculating  $Q$ ,  $K$ , and  $V$  from  $X$  in a matrix form.

# Transformer



- Multi-headed attention (MHA) allows Transformer to focus on different positions.

# Transformer



- If eight heads are used, we end up getting eight different sets of encoded vectors (**attention heads**).

# Transformer

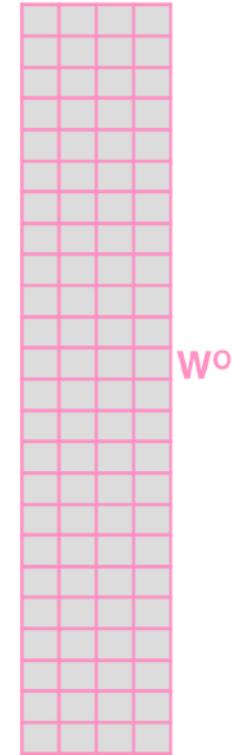
---

1) Concatenate all the attention heads



2) Multiply with a weight matrix  $W^o$  that was trained jointly with the model

$x$



3) The result would be the  $Z$  matrix that captures information from all the attention heads. We can send this forward to the FFNN

$$= \begin{matrix} Z \\ \begin{matrix} \text{---} & \text{---} & \text{---} & \text{---} \end{matrix} \end{matrix}$$

- We simply pass them through additional (learnable) linear map.

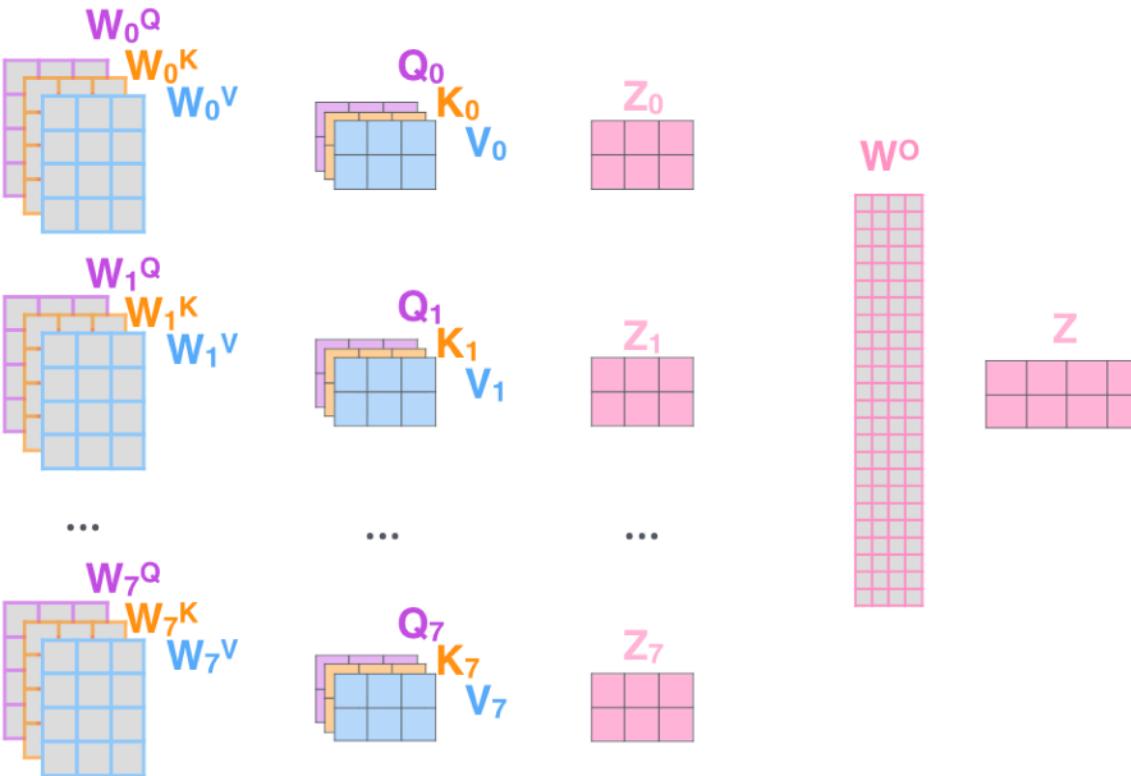
# Transformer

- 1) This is our input sentence\*  $X$
- 2) We embed each word\*  $R$
- 3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices  $W_0^Q, W_0^K, W_0^V$
- 4) Calculate attention using the resulting  $Q/K/V$  matrices
- 5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer

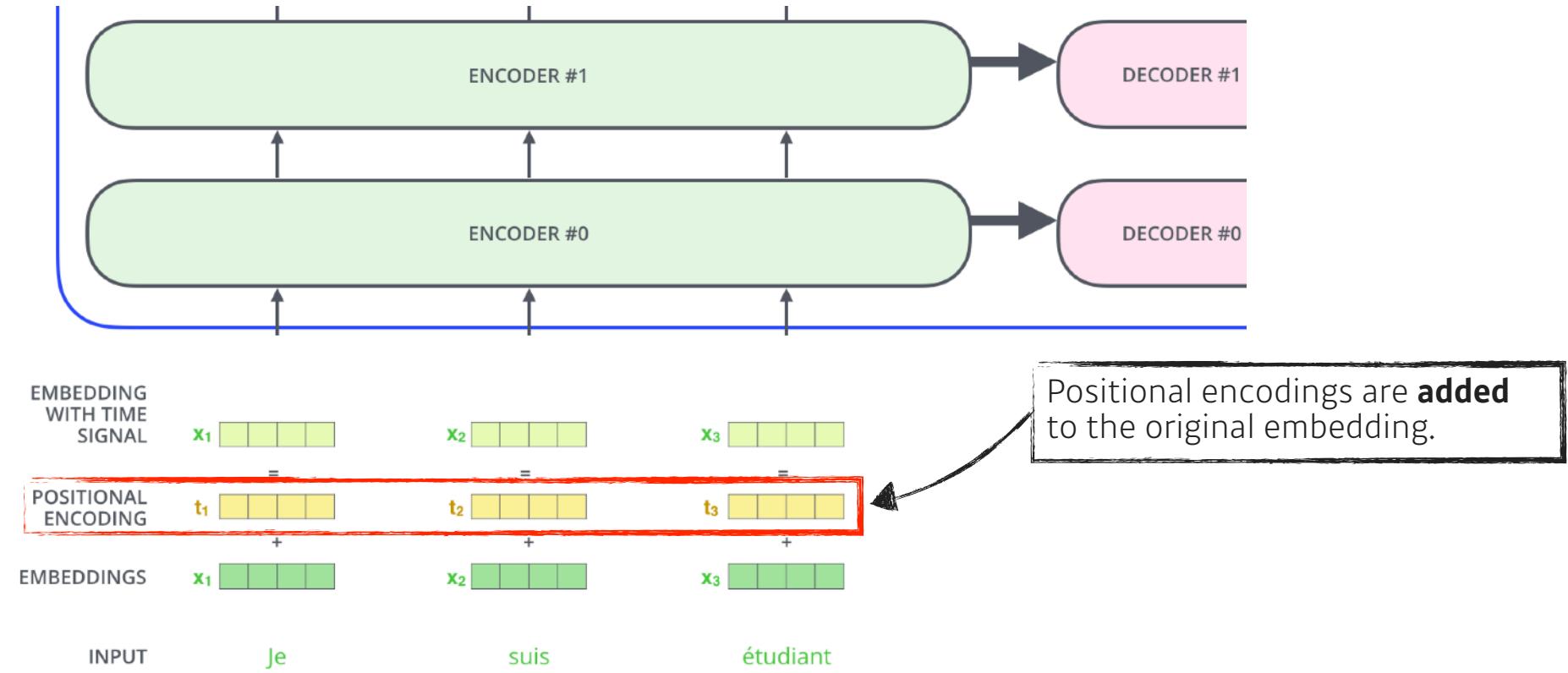
Thinking Machines  
 $X$

\* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

$R$



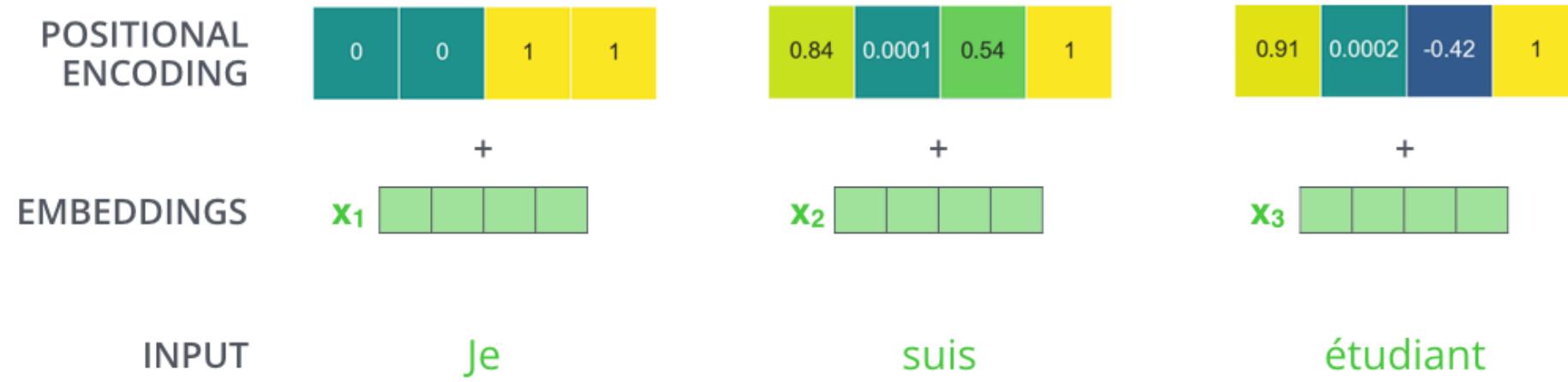
# Transformer



- Why do we need positional encoding?

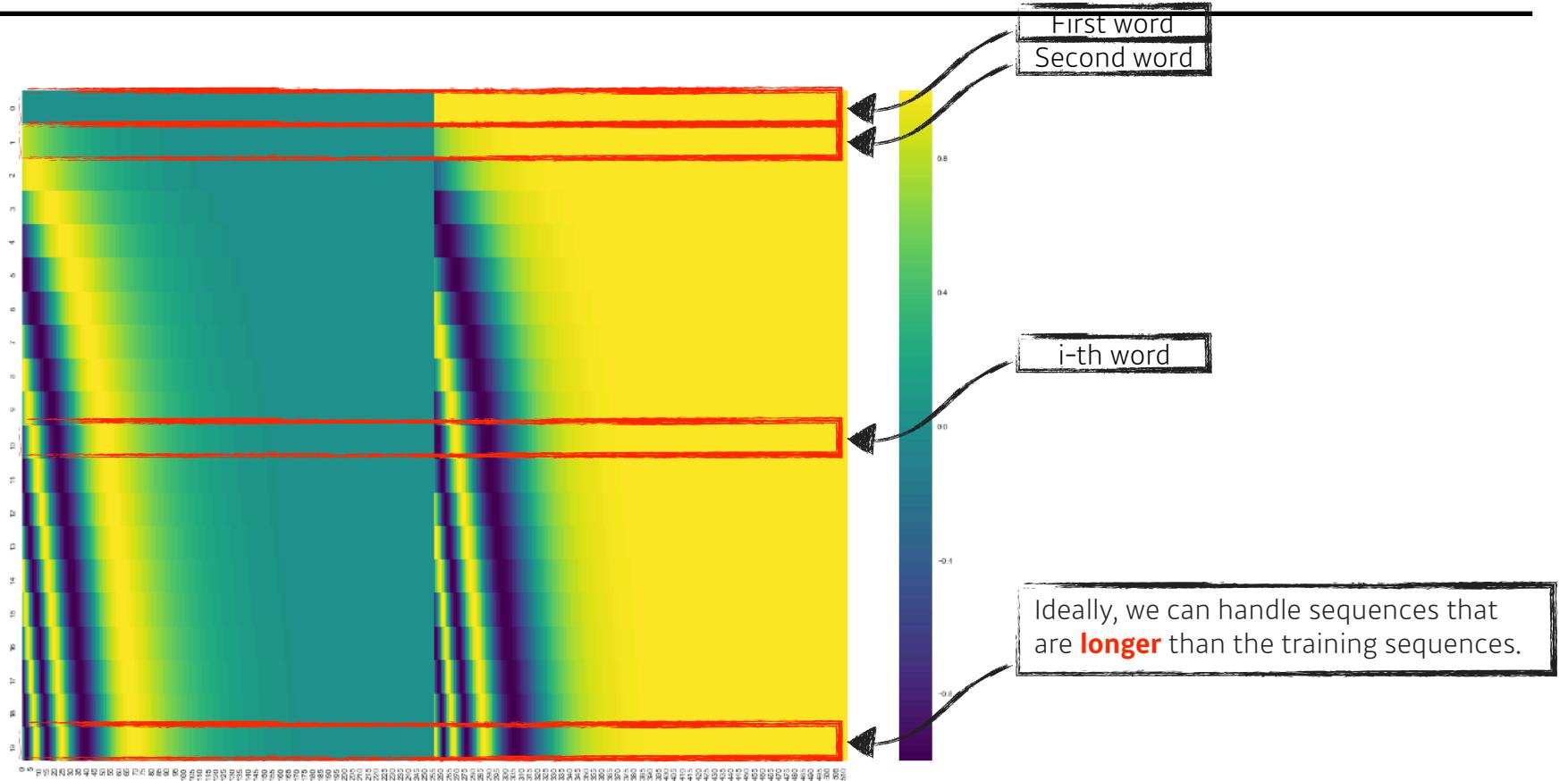
# Transformer

---



- This is the case for 4-dimensional encoding.

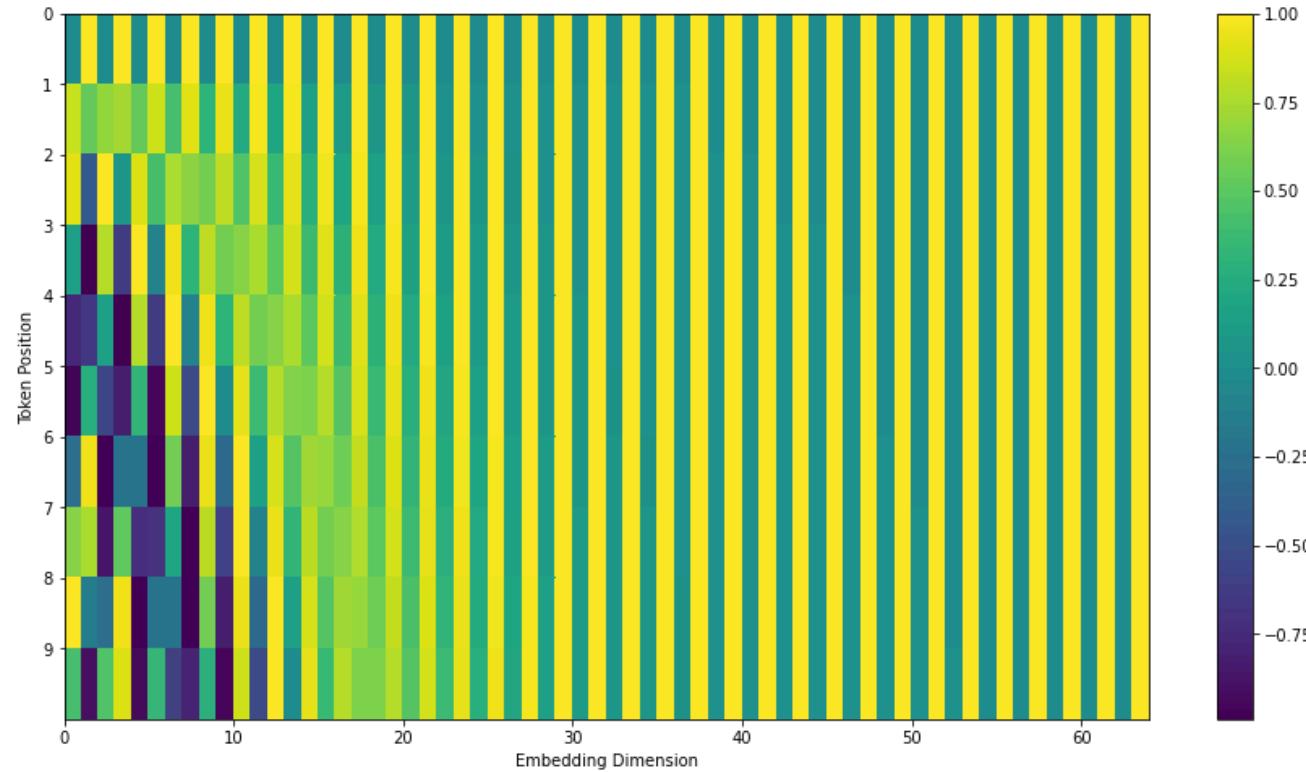
# Transformer



- This is the case for 512-dimensional encoding.

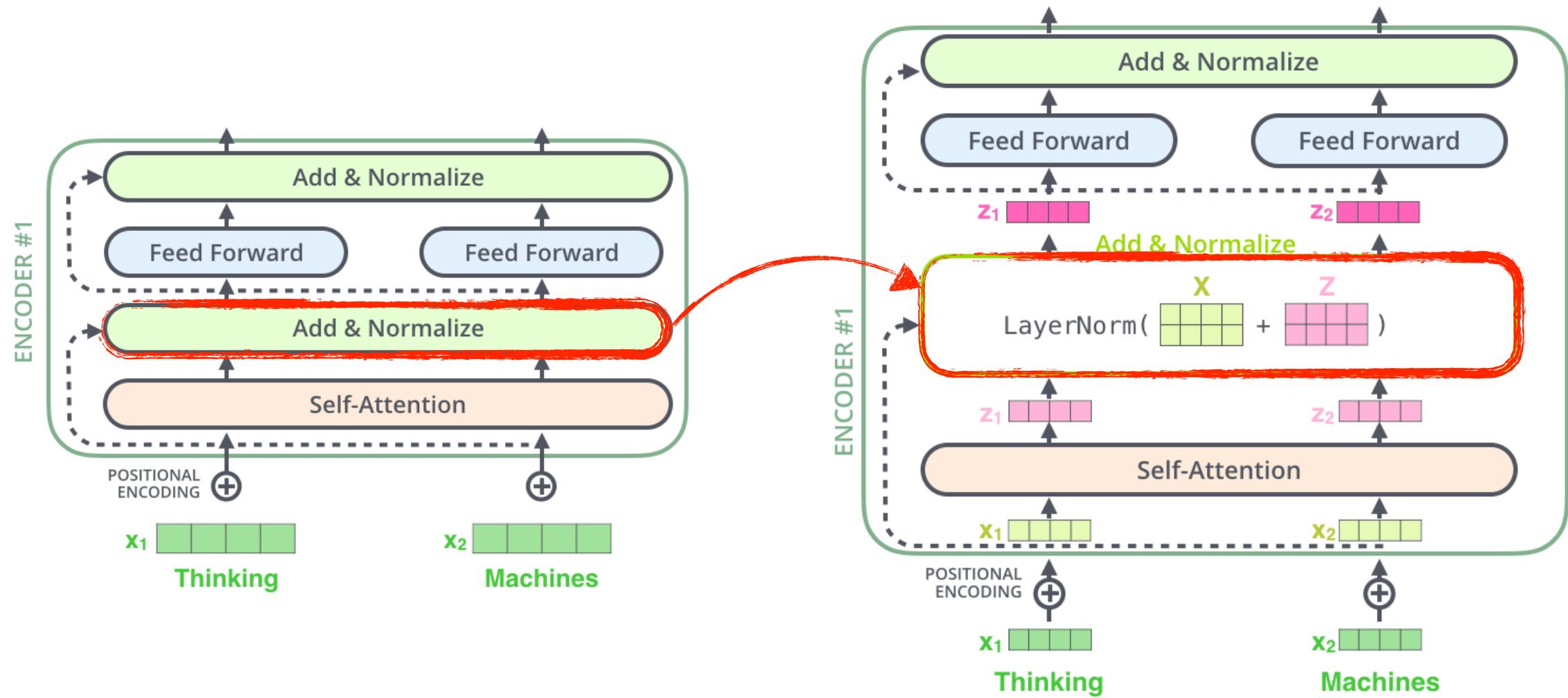
# Transformer

---



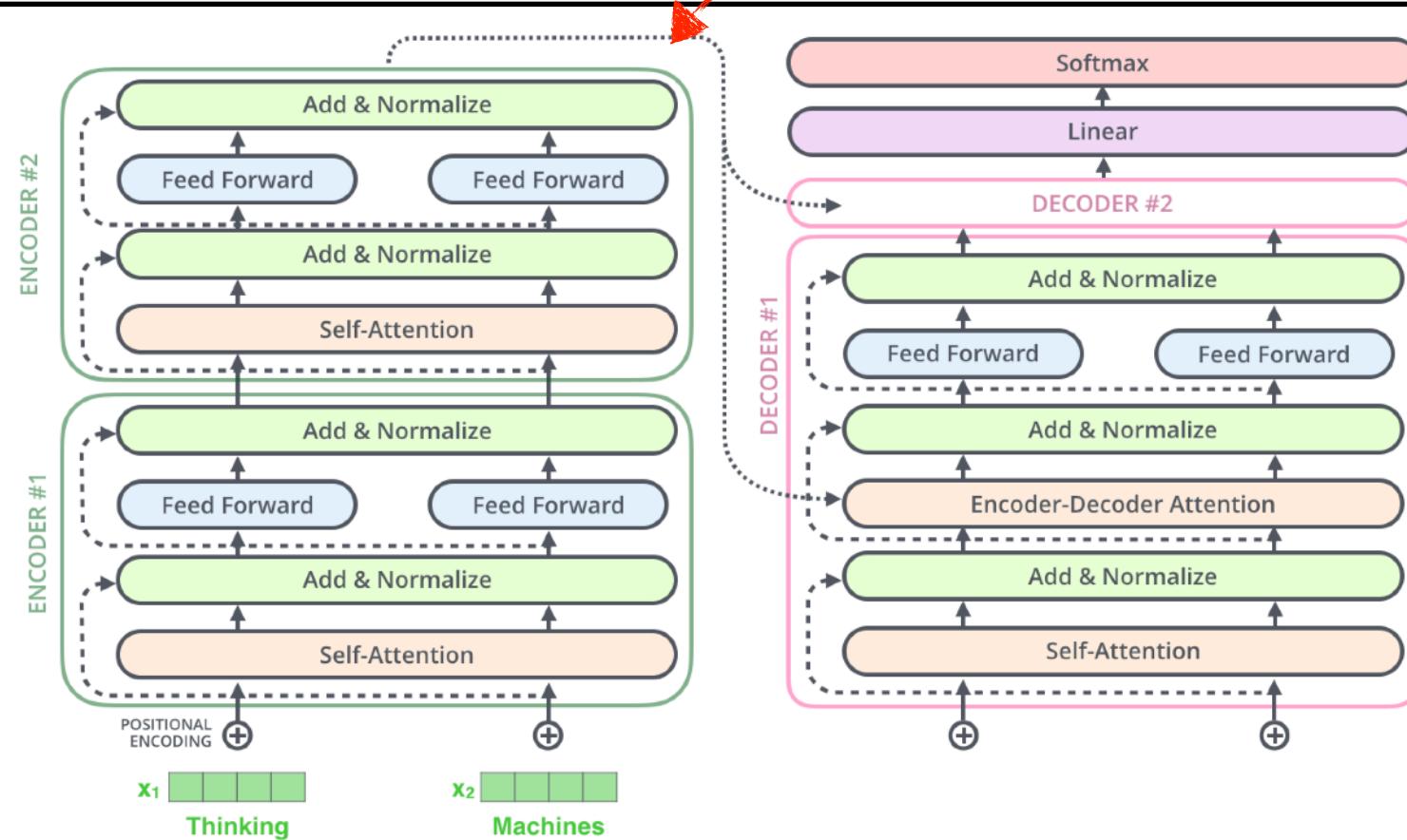
- Recent (July, 2020) update on positional encoding.

# Transformer



# Transformer

What is the information being sent from **encoders** to **decoders**?

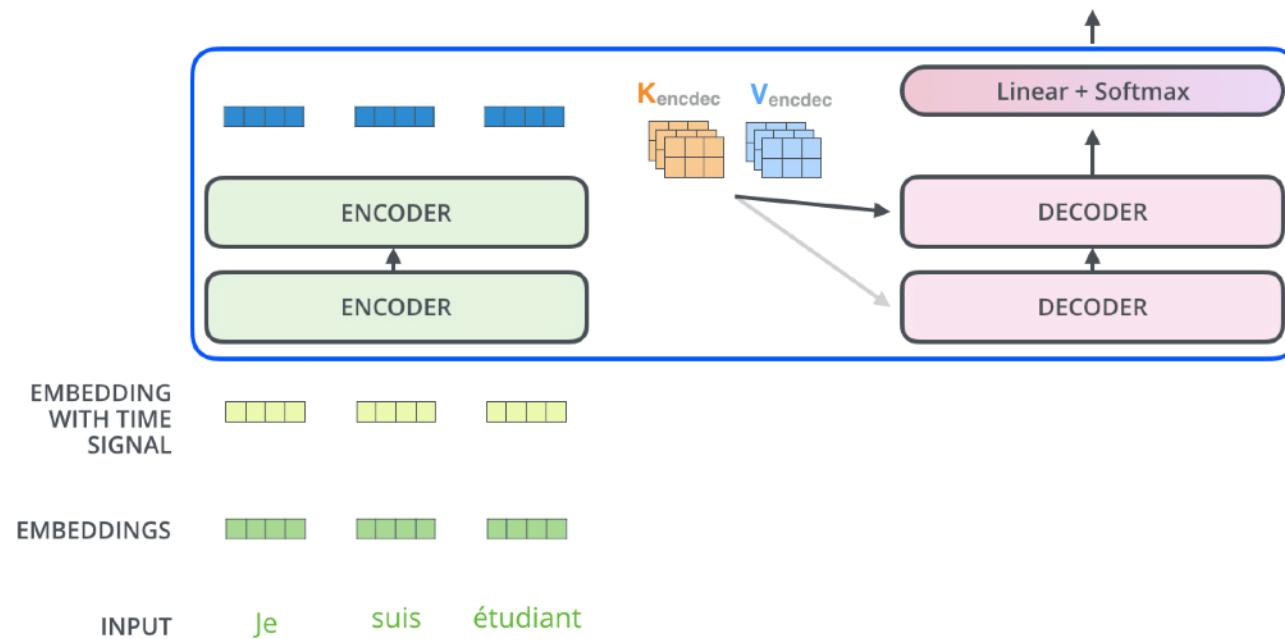


- Now, let's take a look at the decoder side.

# Transformer

Decoding time step: 1 2 3 4 5 6

OUTPUT |



K and V are transferred from the encoder (i.e., input words).

A detailed view of the multi-head attention mechanism. It shows the calculation of the context vector  $Z$  as follows:

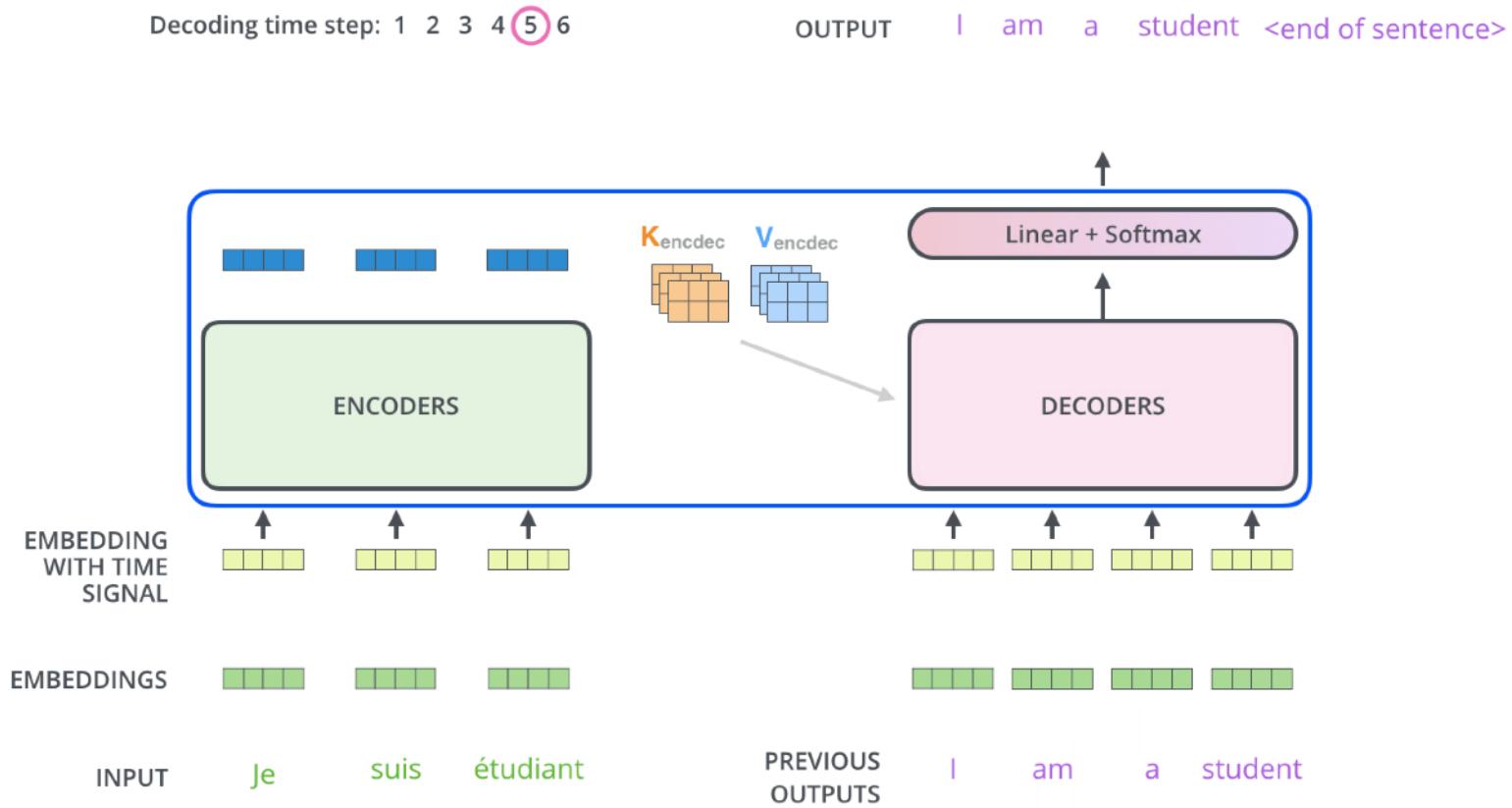
$$Z = \text{softmax} \left( \frac{Q \times K^T}{\sqrt{d_k}} \right) V$$

The diagram illustrates the components:  $Q$  (query matrix),  $K^T$  (key matrix transpose), and  $V$  (value matrix). The query matrix  $Q$  is multiplied by the transpose of the key matrix  $K^T$ , and the result is scaled by  $\sqrt{d_k}$  before being passed through a softmax function to produce the context vector  $Z$ .

- Transformer transfers **key (K)** and **value (V)** of the topmost encoder to the decoder.

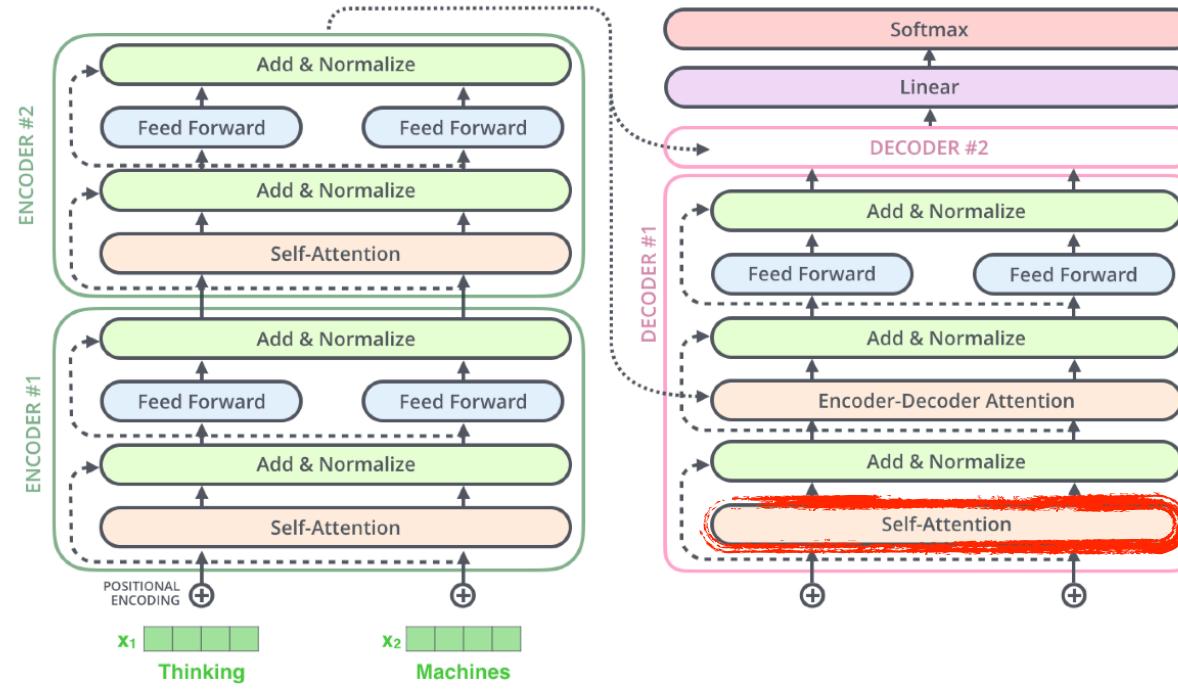


# Transformer



- The output sequence is generated in an autoregressive manner.

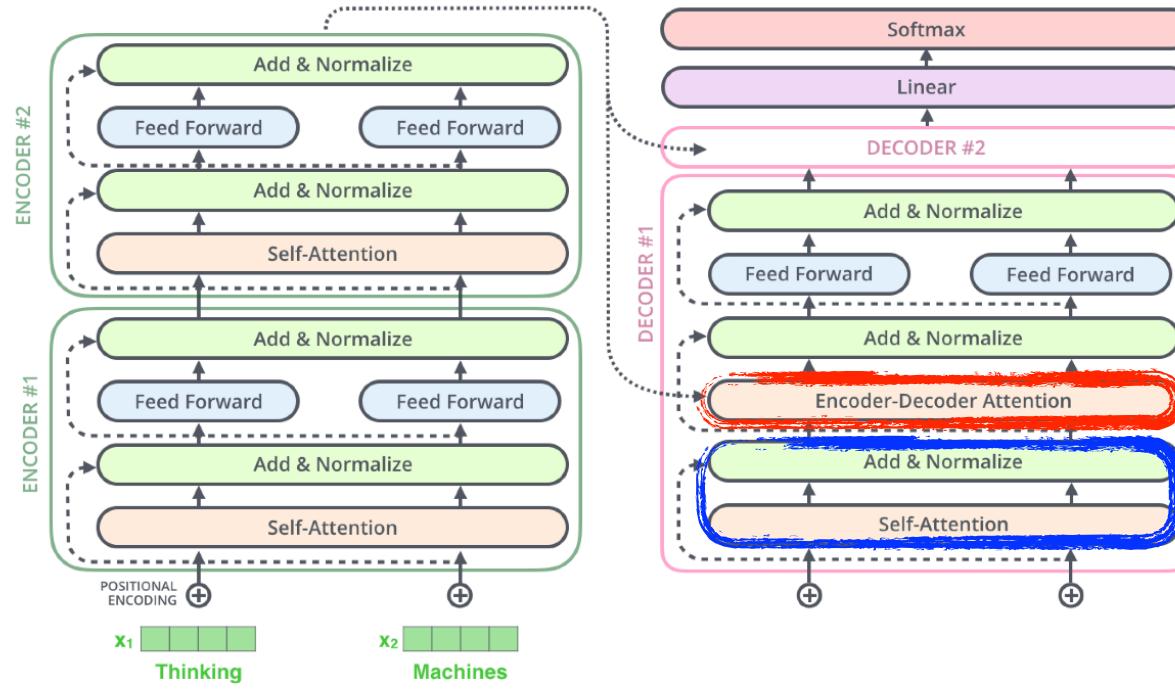
# Transformer



$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right)V = Z$$

- In the decoder, the **self-attention layer** is only allowed to attend to earlier positions in the output sequence which is done by masking future positions before the softmax step.

# Transformer



$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right)V = Z$$

- The “Encoder-Decoder Attention” layer works just like multi-headed self-attention, except it creates its **Queries** matrix from **the layer below it**, and takes the **Keys** and **Values** from the encoder stack.

# Transformer

Which word in our vocabulary  
is associated with this index?

am

Get the index of the cell  
with the highest value  
(**argmax**)

5

**log\_probs**



Softmax

**logits**



Linear

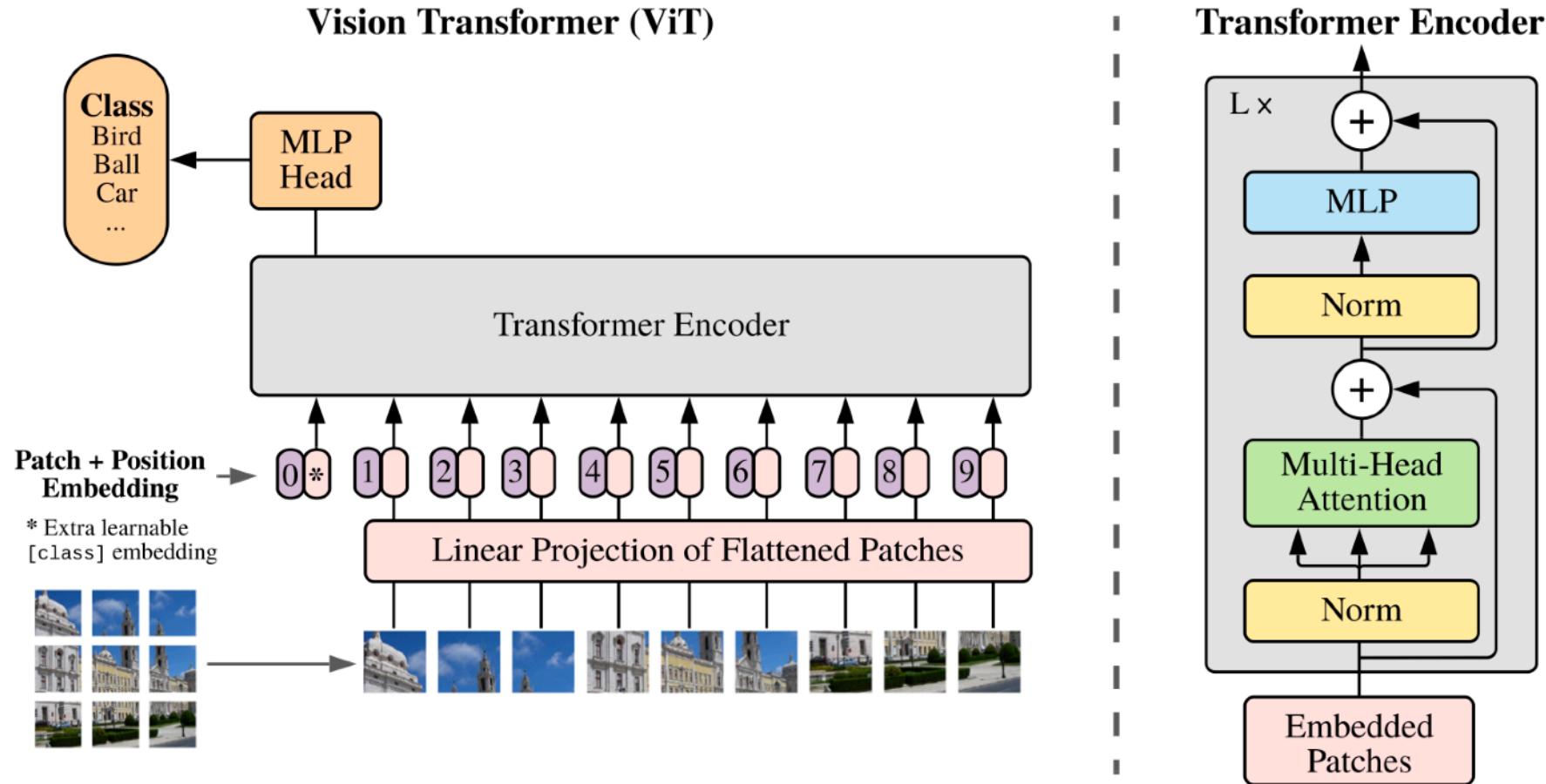
Decoder stack output



- The final layer converts the stack of decoder outputs to the distribution over words.

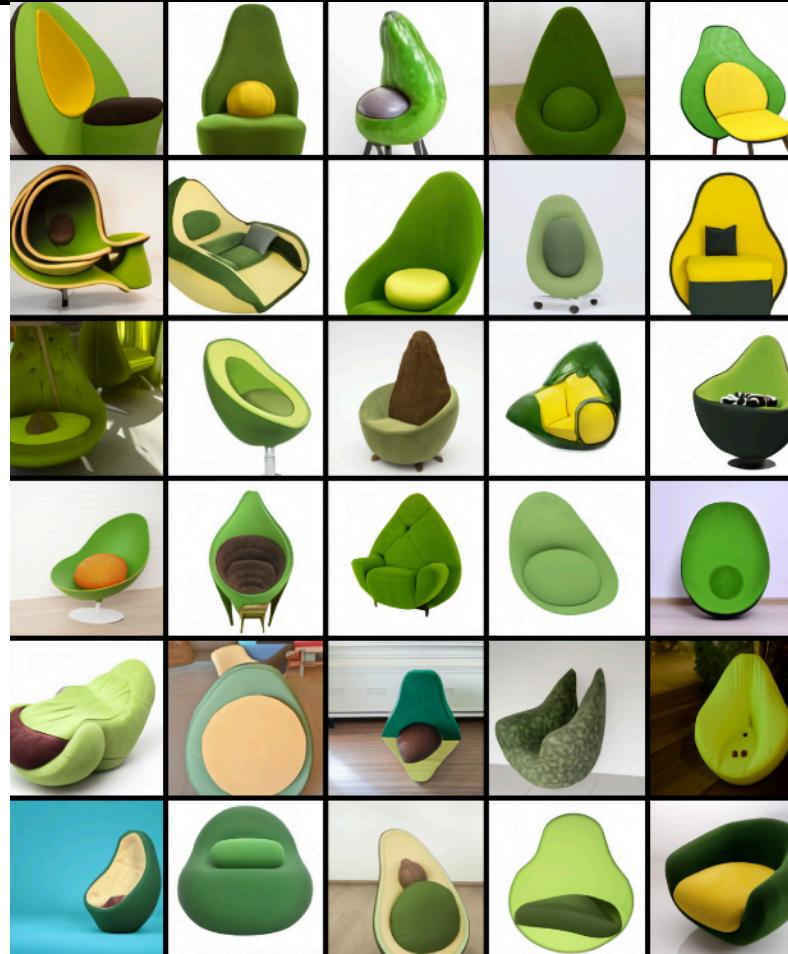


# Vision Transformer



# DALL-E

---



An armchair in the shape of an avocado.

<https://openai.com/blog/dall-e/>

# Thank you for listening

---