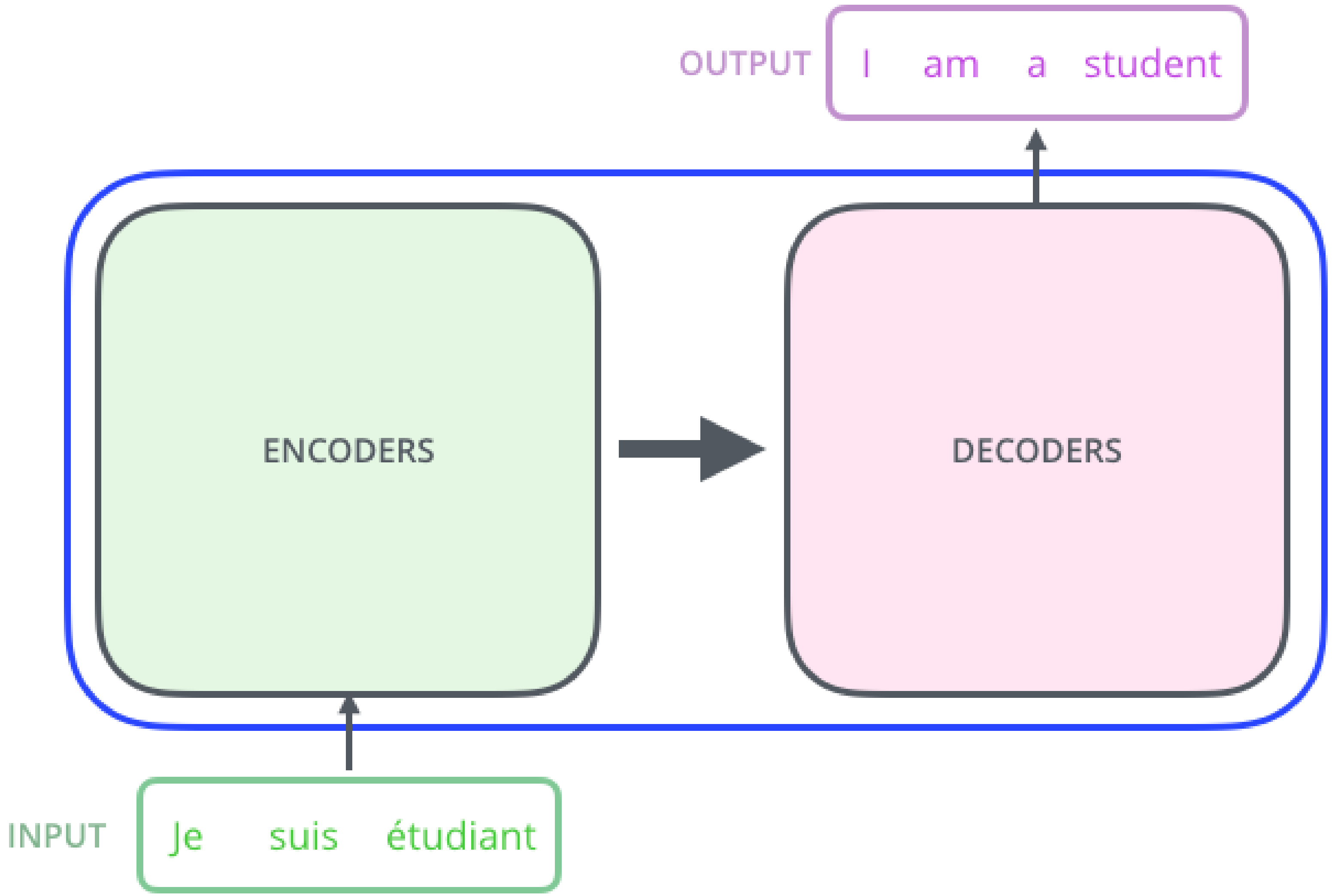
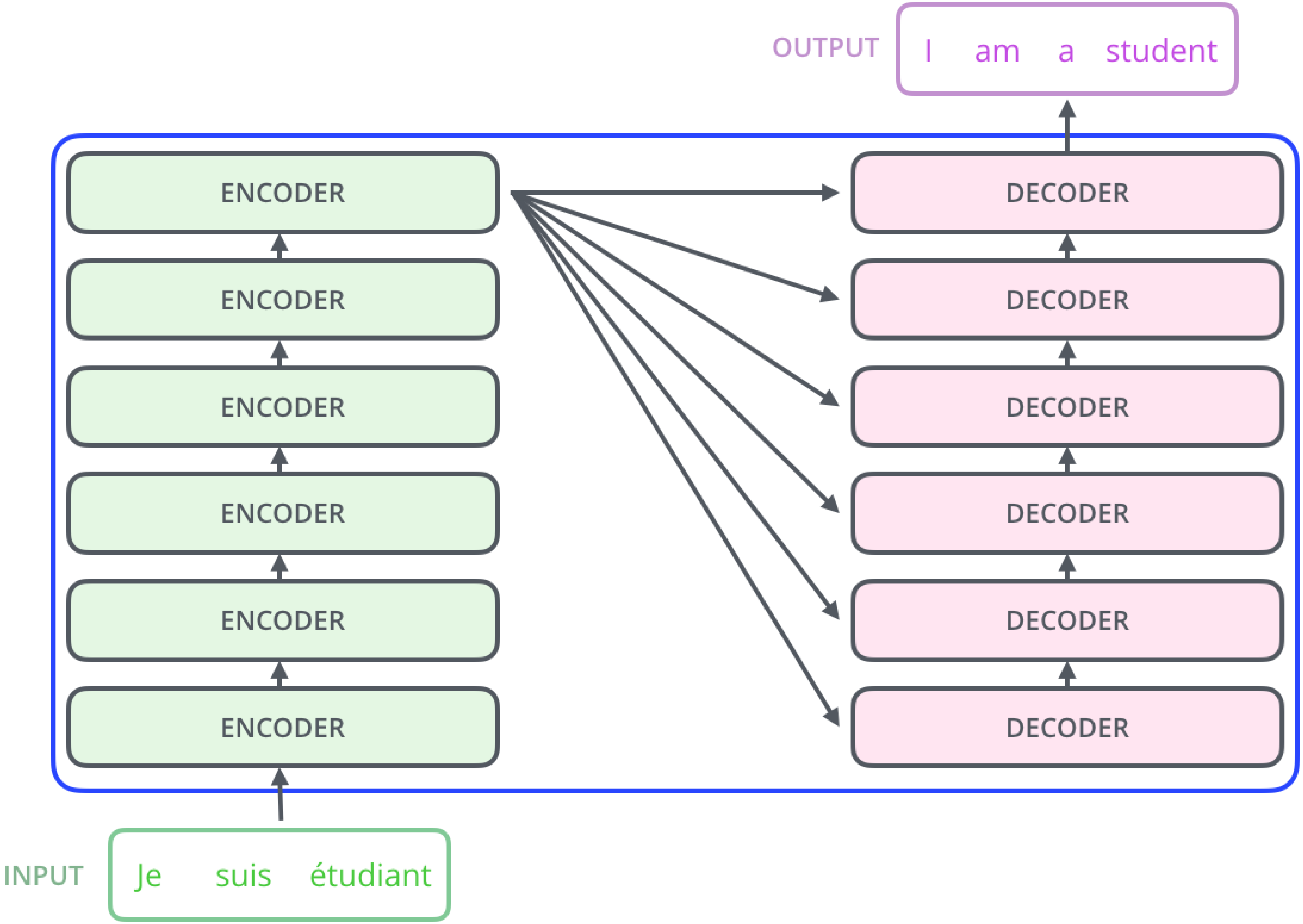
Transformer
(Attention is All You Need)

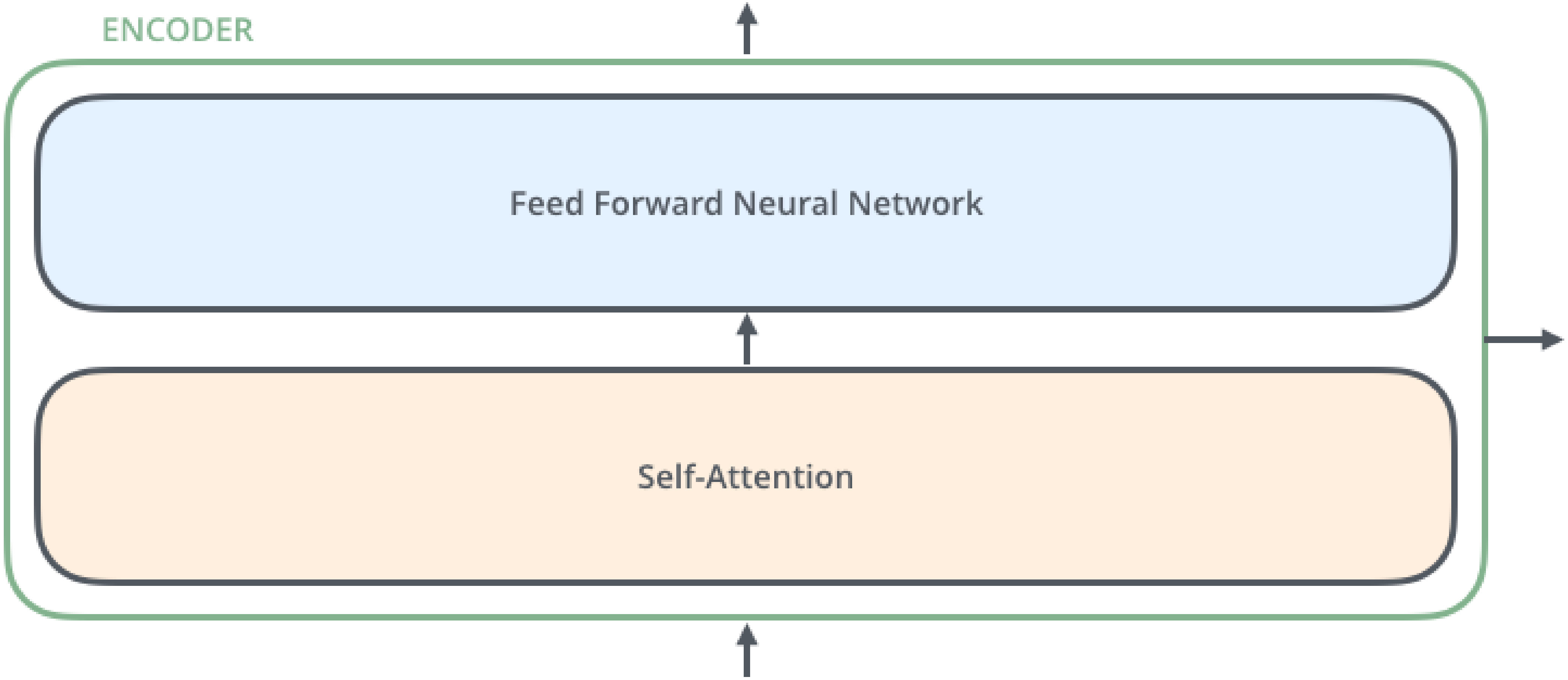
Transformer (A High-Level Look)

Transformer (**A High-Level Look**)

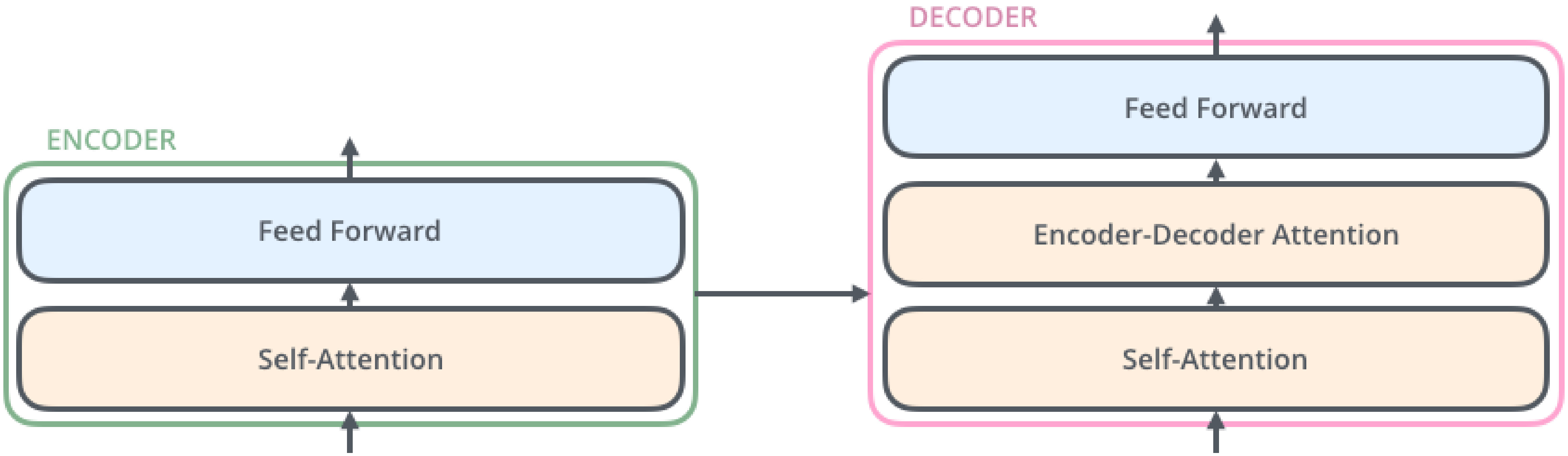



OUTPUT
I am a student
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
INPUT
Je suis étudiant

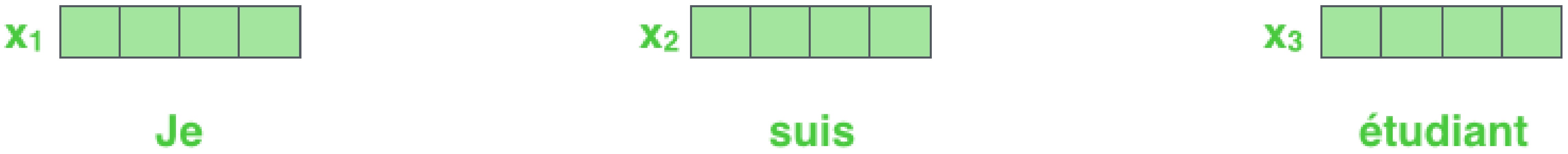
The encoders are all identical in structure. Each one is broken down into two sub-layers:

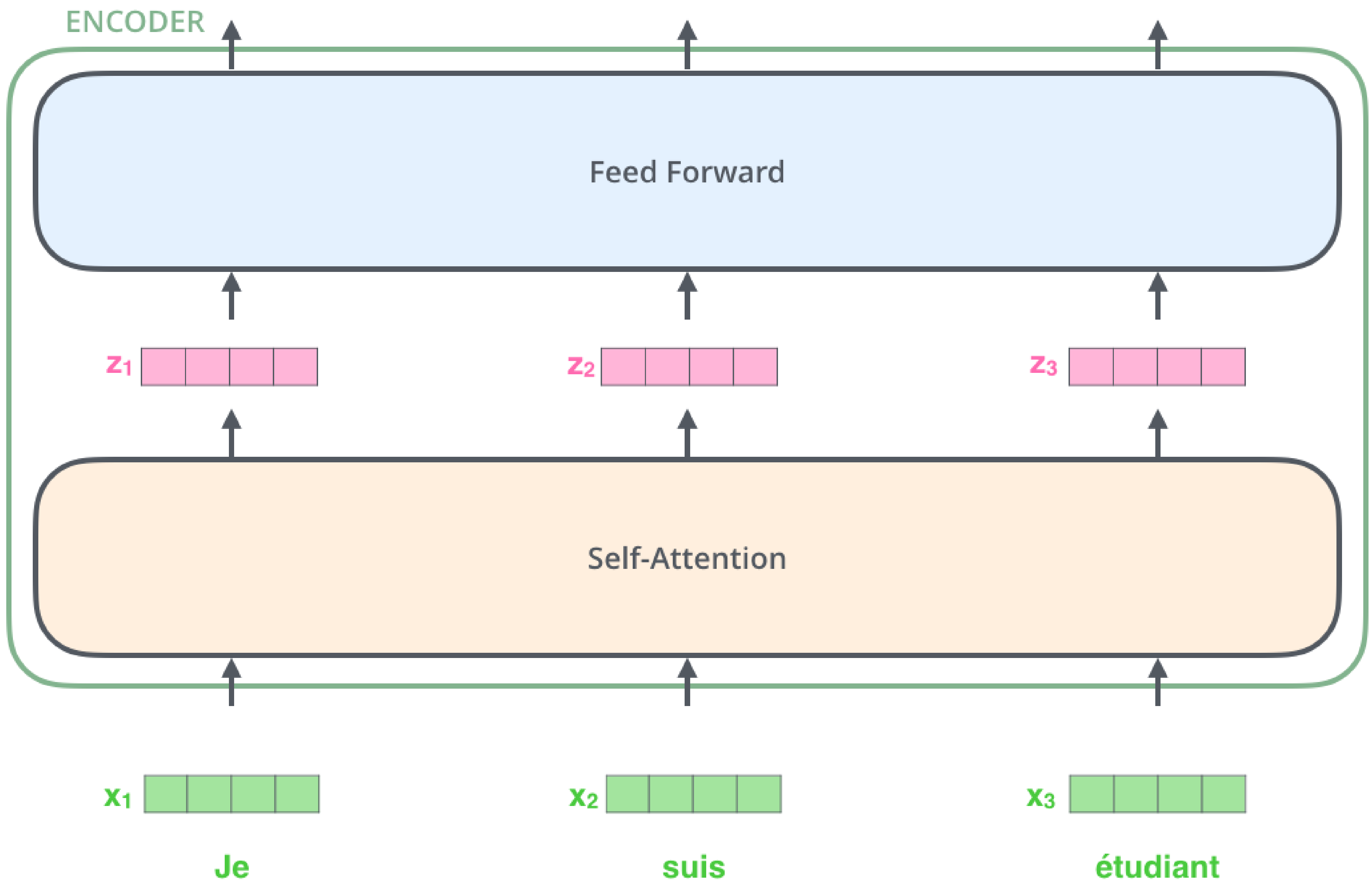


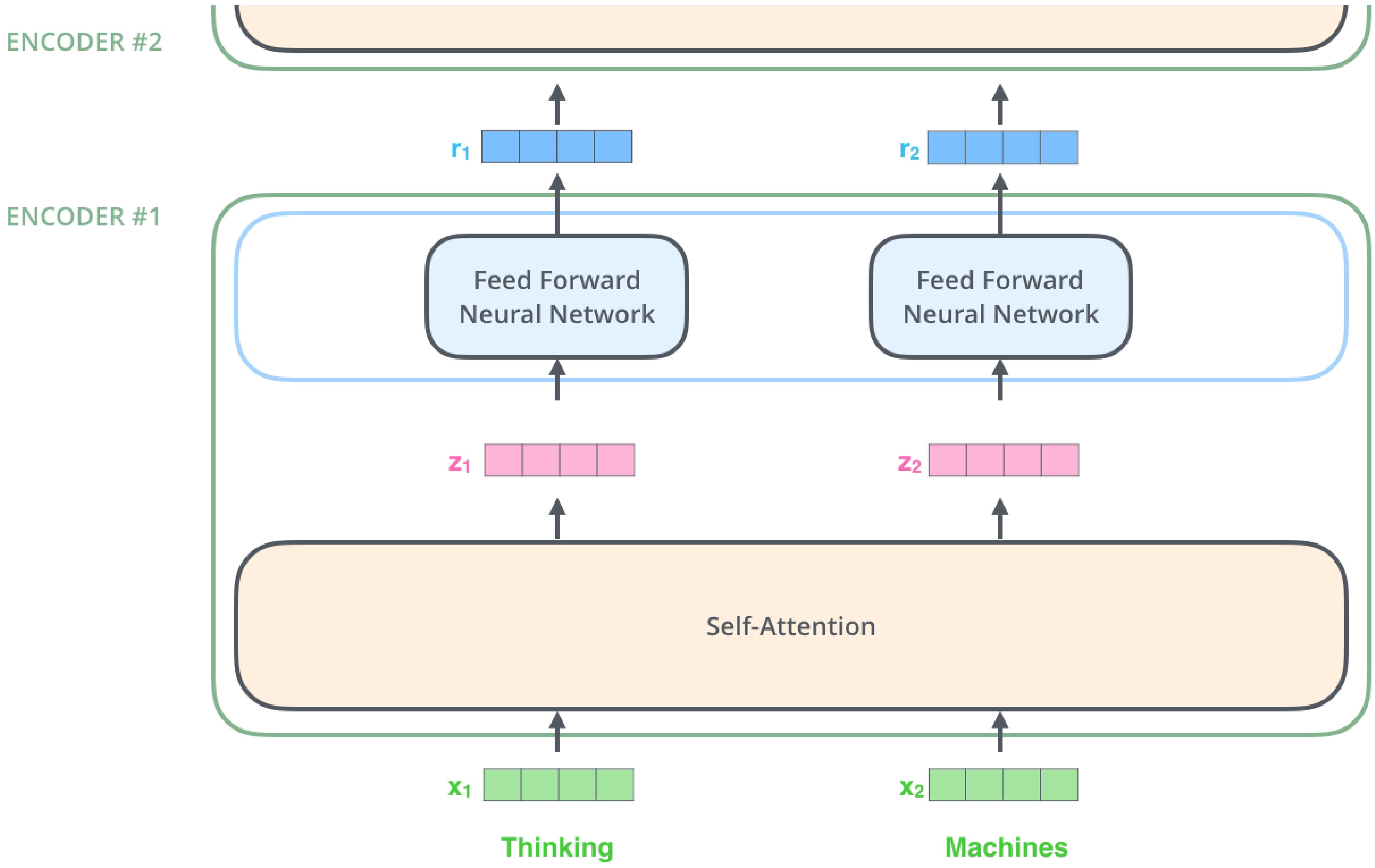
The encoders are all identical in structure. Each one is broken down into two sub-layers:



Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.

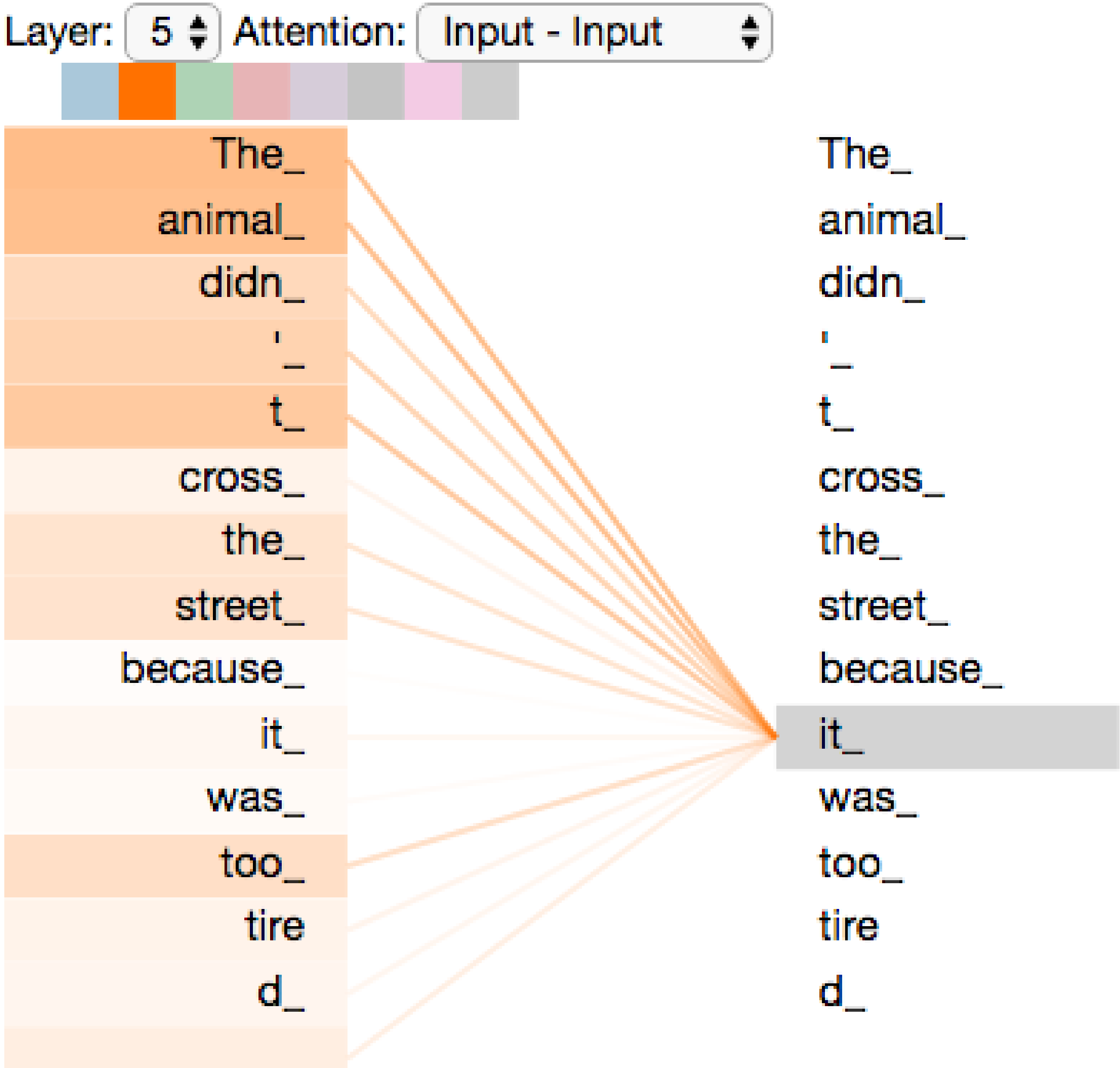


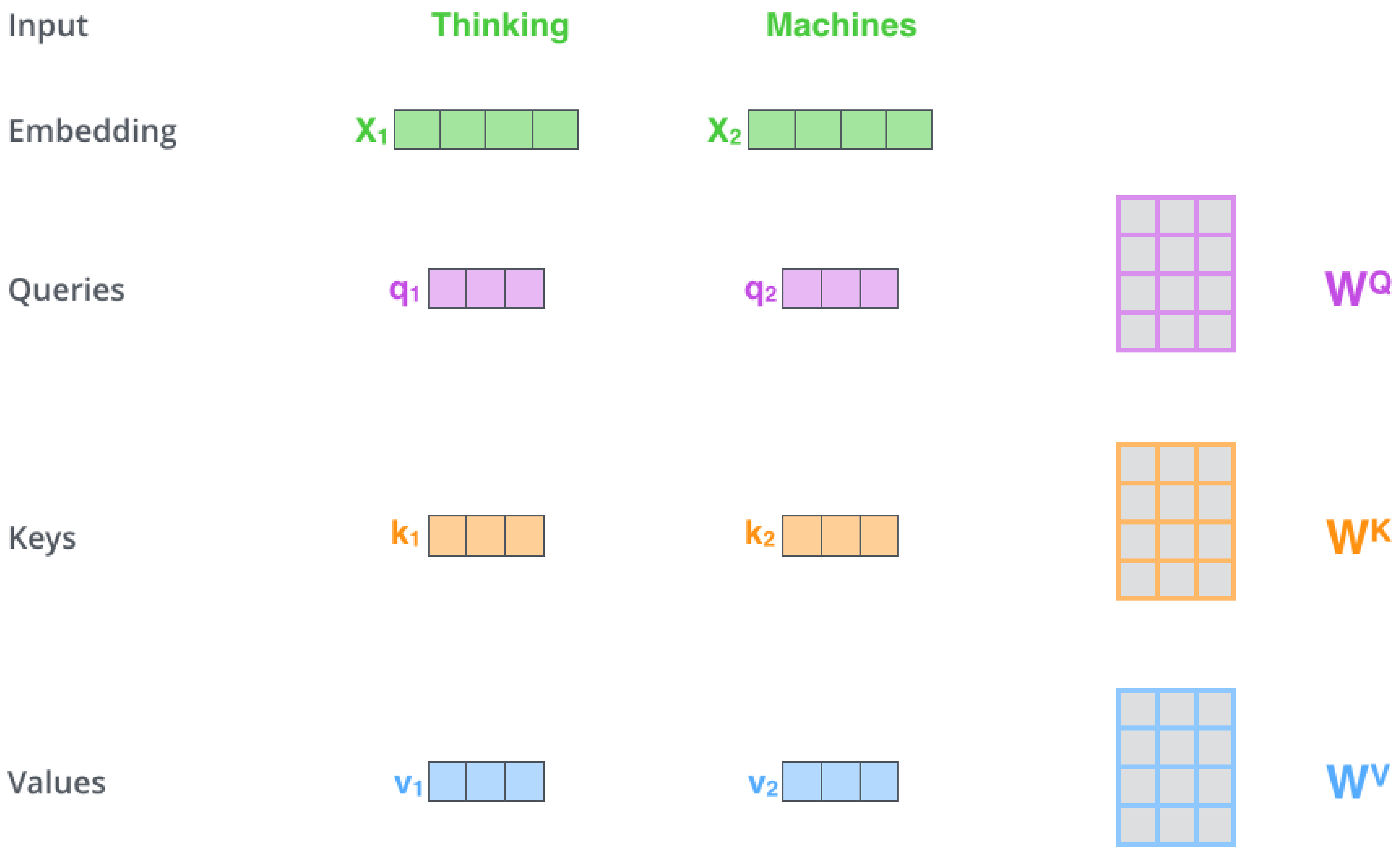

ENCODER
Feed Forward
z1
z2
z3
Self-Attention
x1
x2
x3
Je
suis
étudiant

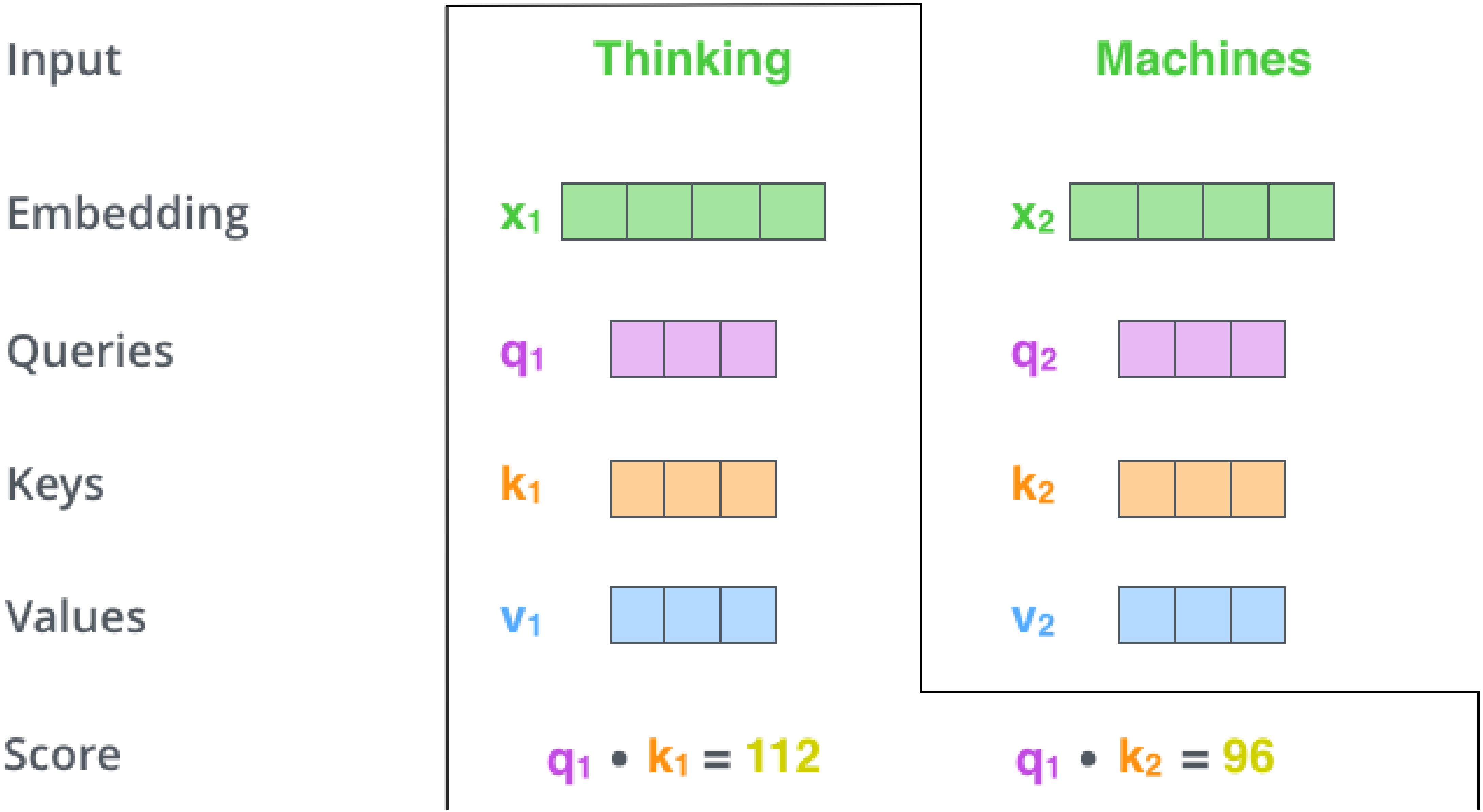


ENCODER #2
r1
r2
ENCODER #1
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

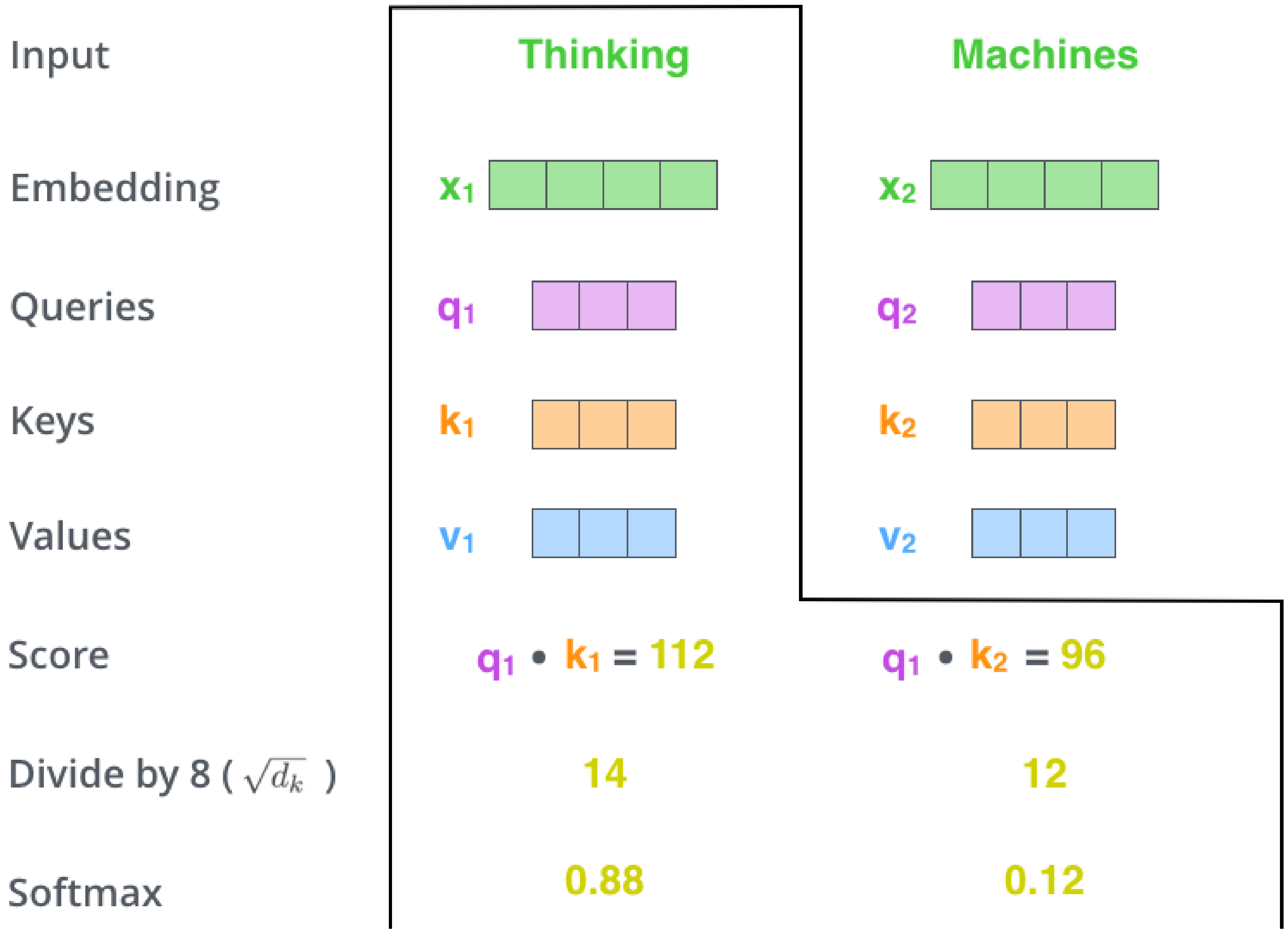
The animal didn't cross the street because it was too tired”

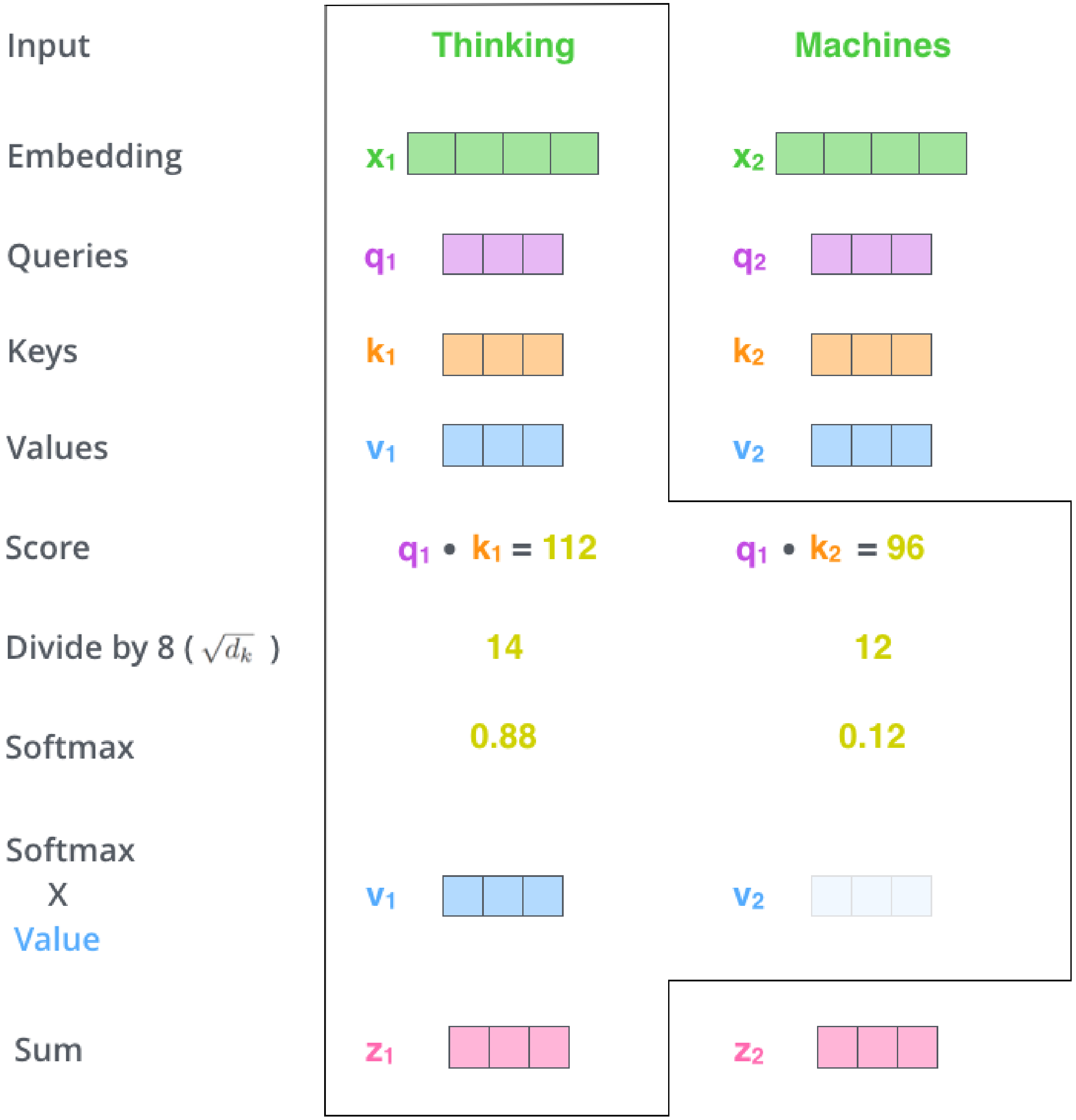



Input
Thinking
Machines
Embedding
X1
X2
Queries
q1
q2
WQ
Keys
k1
k2
WK
Values
v1
v2
WV

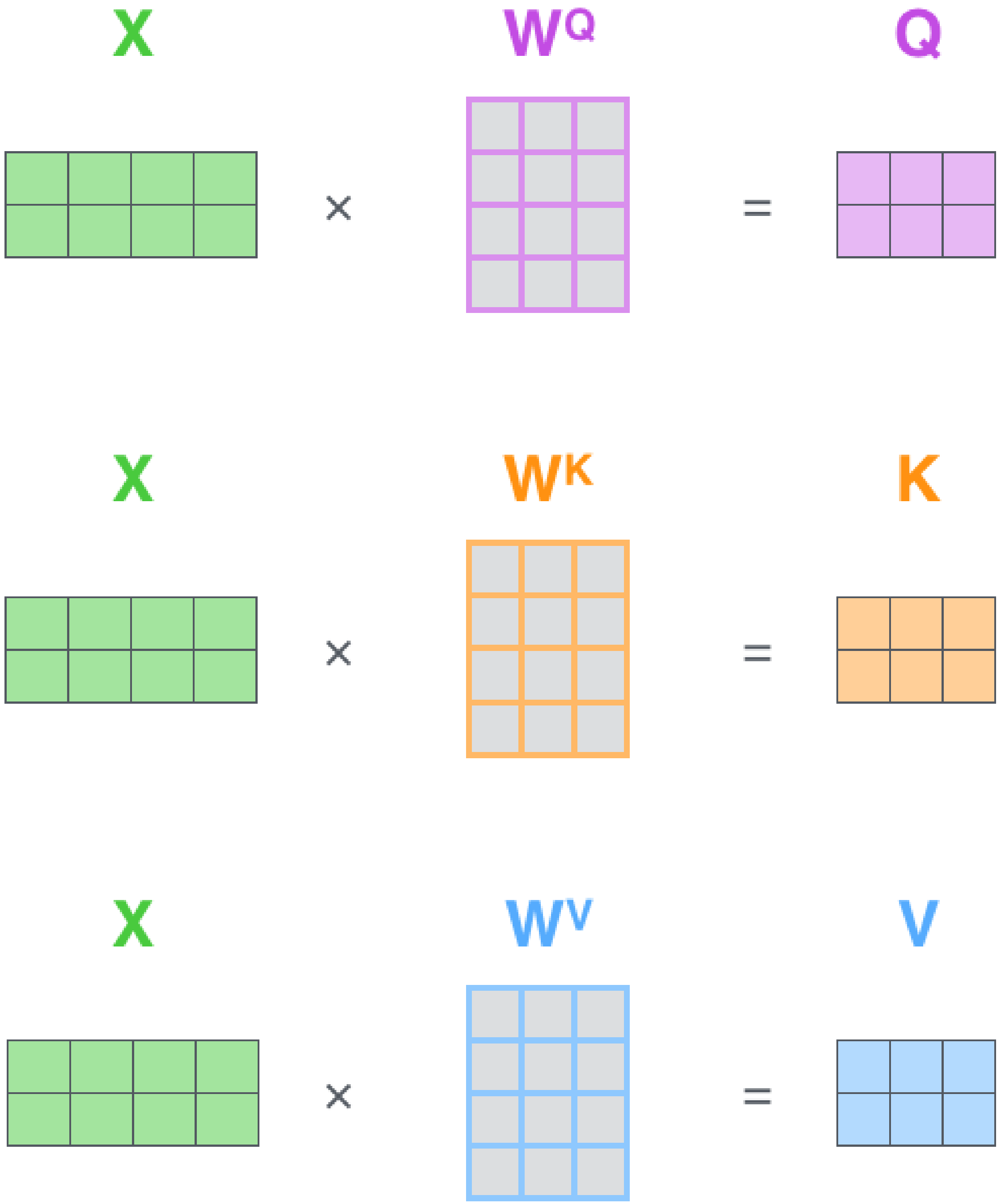


Input
Embedding
Queries
Keys
Values
Score
Thinking
Machines
x1
x2
q1
q2
k1
k2
v1
v2
q1 • k1 = 112
q1 • k2 = 96

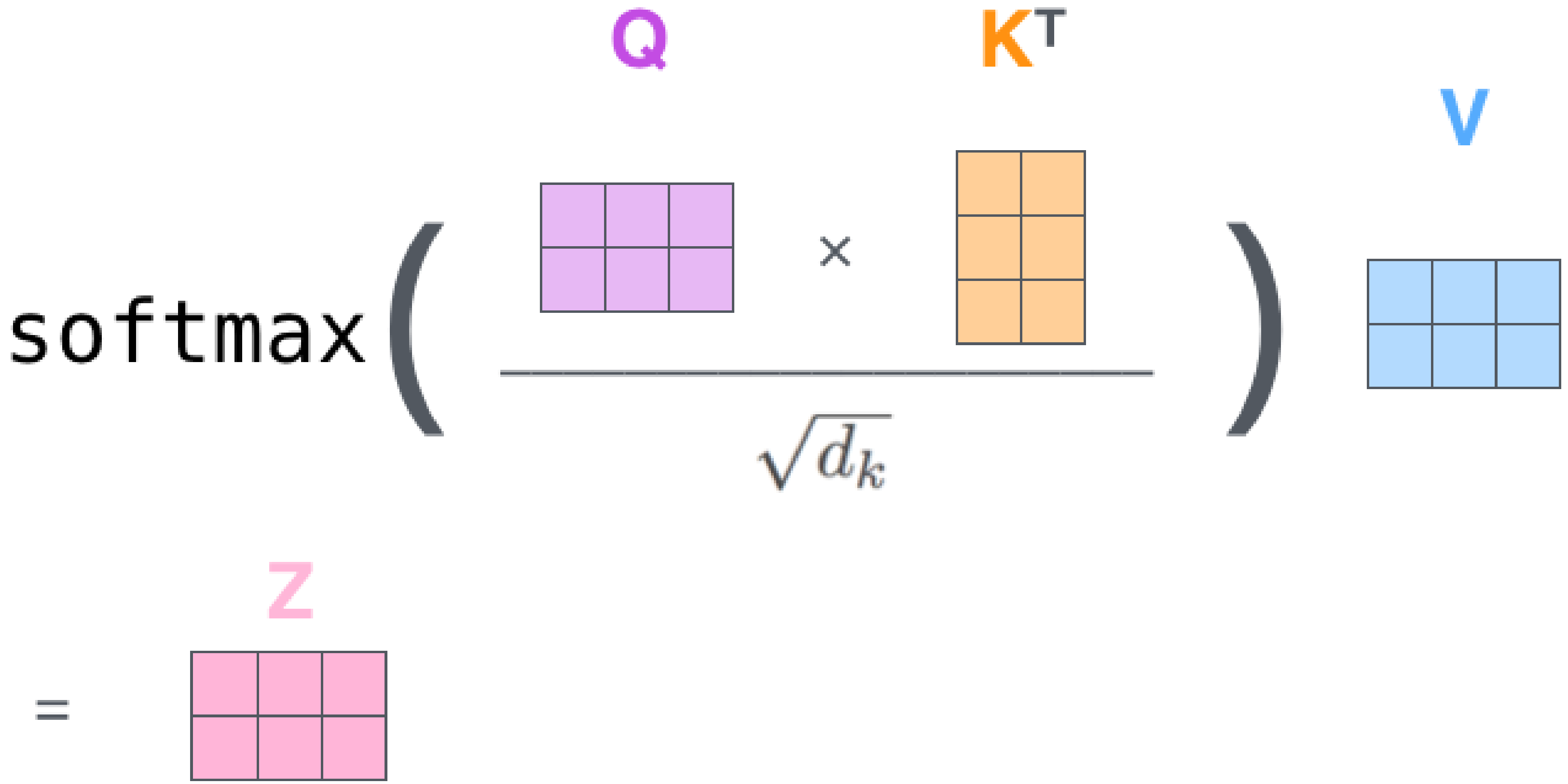

Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 ($\sqrt{d_k}$)
14
12
Softmax
0.88
0.12



Input
Embedding
Queries
Keys
Values
Score
Divide by 8 ($\sqrt{d_k}$)
Softmax
Softmax
X
Value
Sum
Thinking
Machines
x_1
x_2
q_1
q_2
k_1
k_2
v_1
v_2
$q_1 \bullet k_1 = 112$
$q_1 \bullet k_2 = 96$
14
12
0.88
0.12
v_1
v_2
z_1
z_2


X
W^Q
Q
X
W^K
K
X
W^V
V

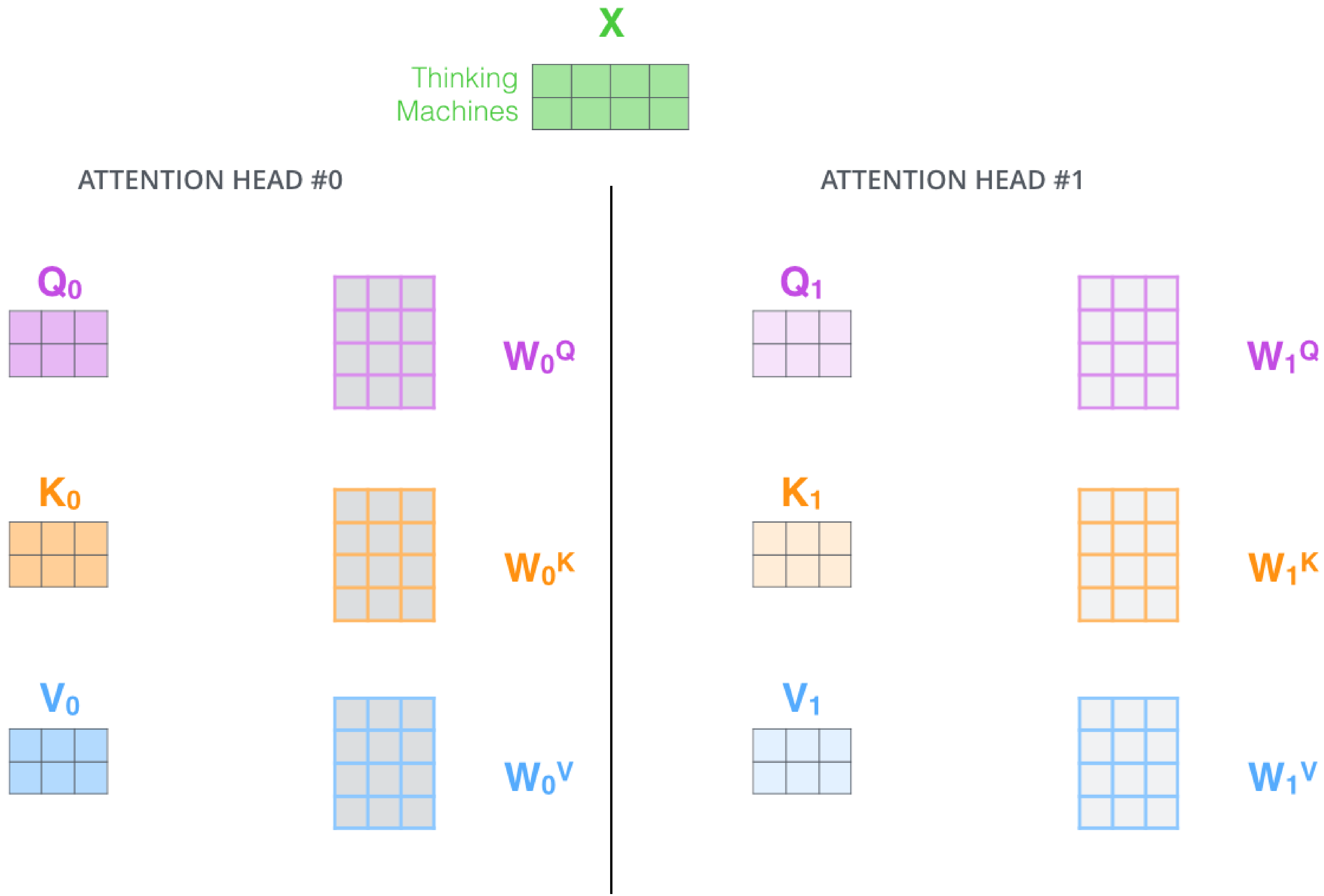
The self-attention calculation in matrix form

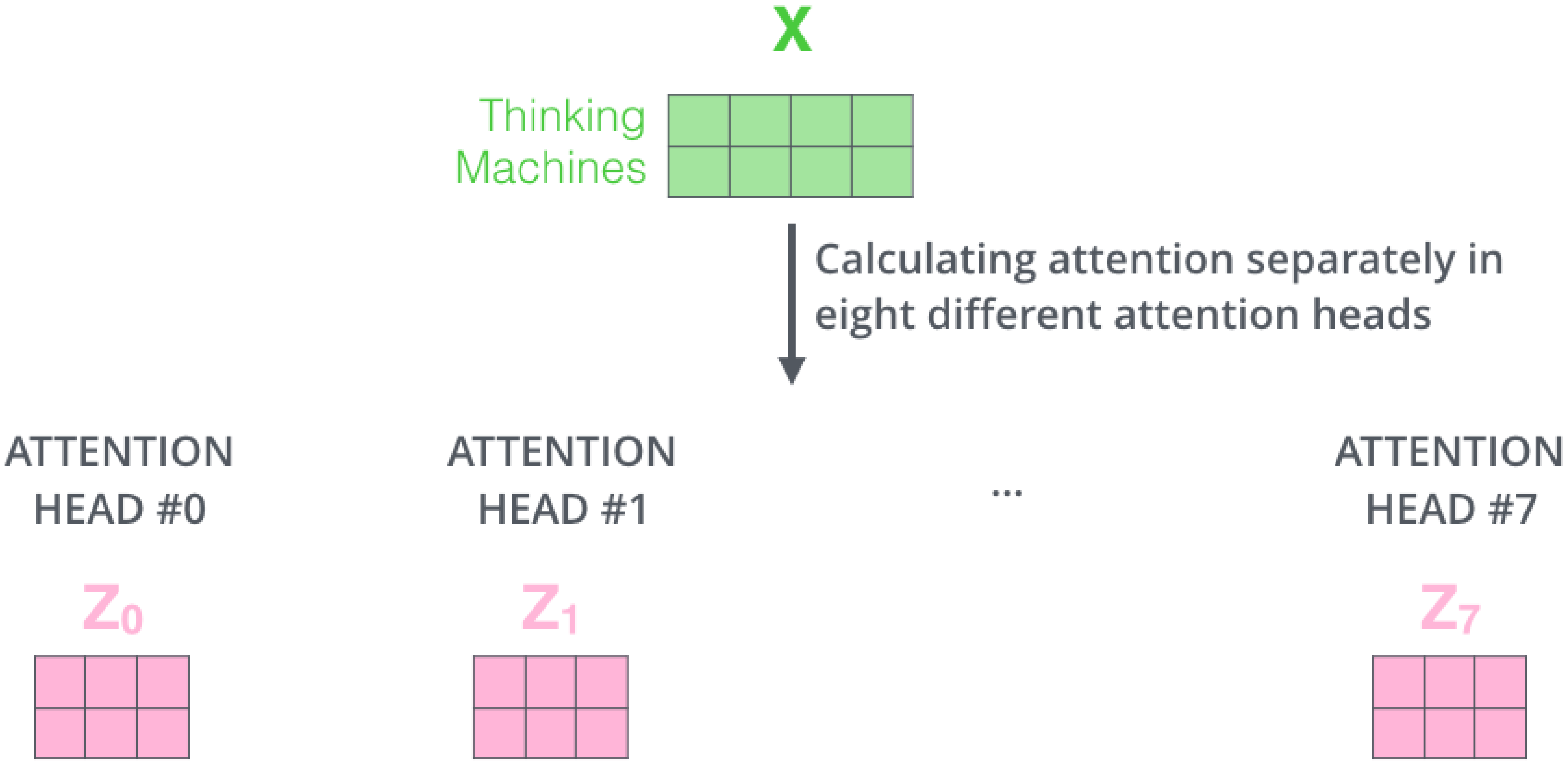


The Beast With Many Heads
Multi Head Attention

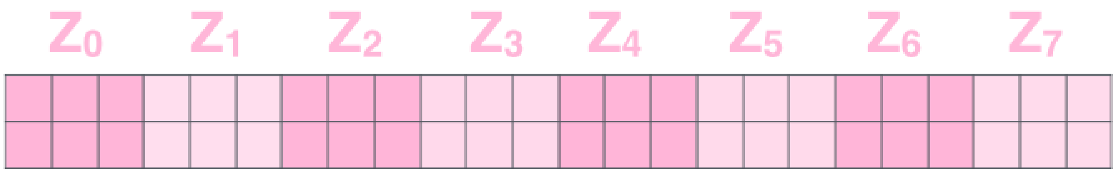
- It expands the model’s ability to focus on different positions

- It gives the attention layer multiple “representation subspaces”


X
Thinking
Machines
ATTENTION HEAD #0
ATTENTION HEAD #1
Q0
W0Q
K0
W0K
V0
W0V
Q1
W1Q
K1
W1K
V1
W1V

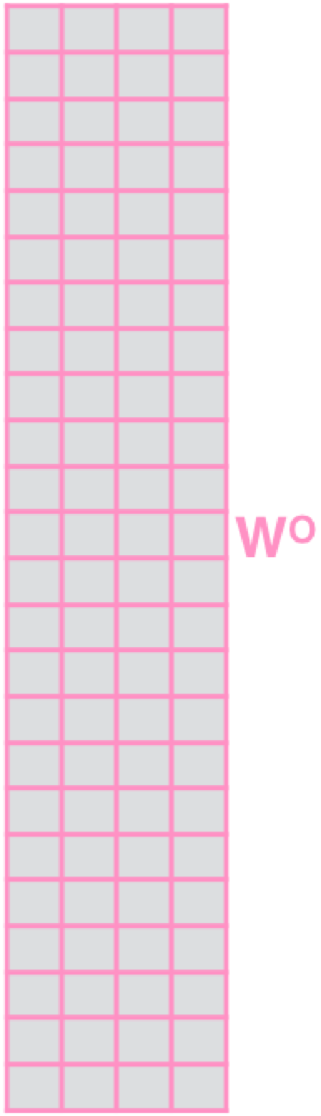

X
Thinking
Machines
Calculating attention separately in eight different attention heads
ATTENTION HEAD #0
ATTENTION HEAD #1
...
ATTENTION HEAD #7
Z0
Z1
Z7

1) Concatenate all the attention heads



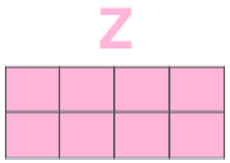
2) Multiply with a weight matrix W^O that was trained jointly with the model

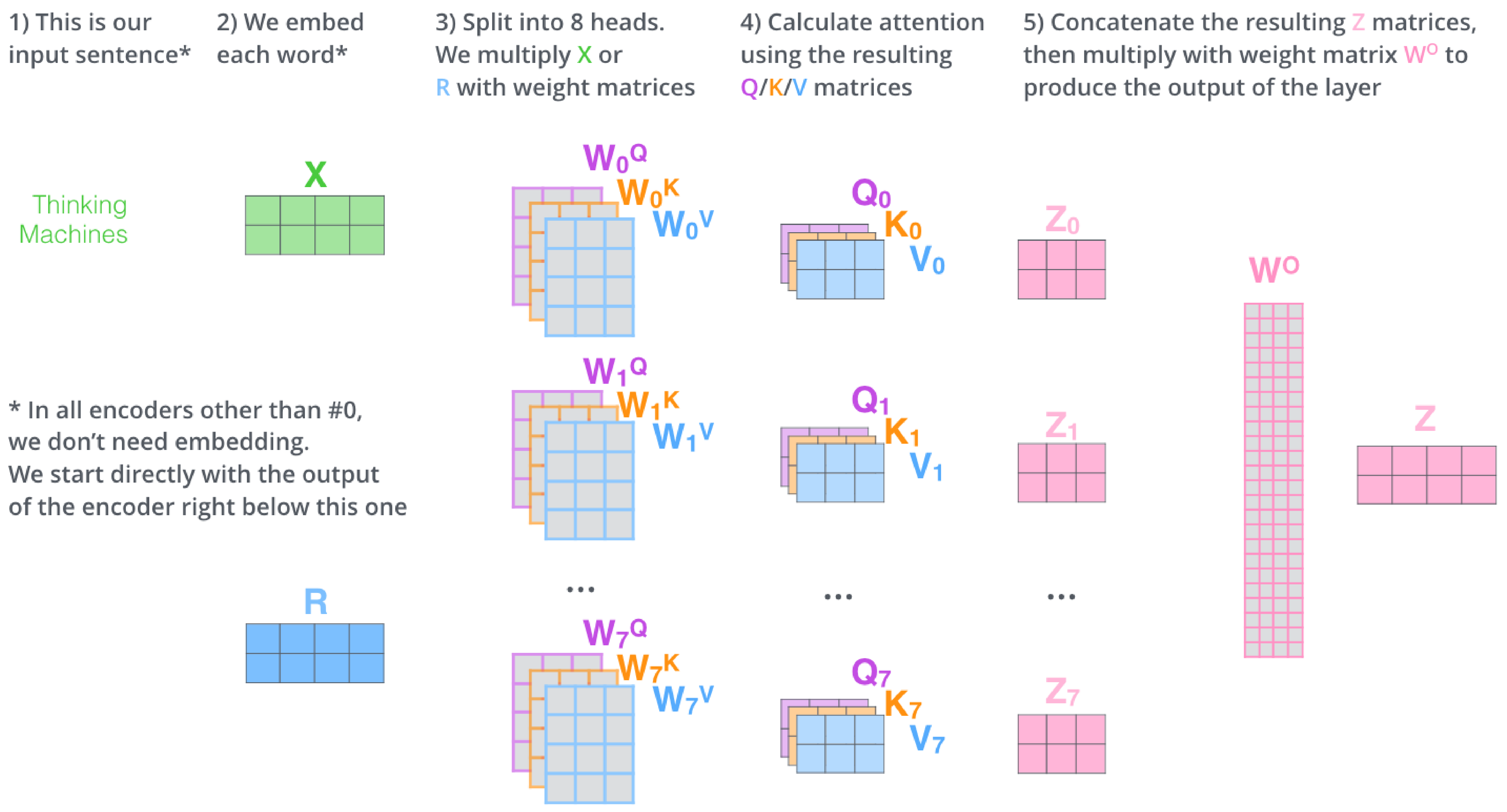
X



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

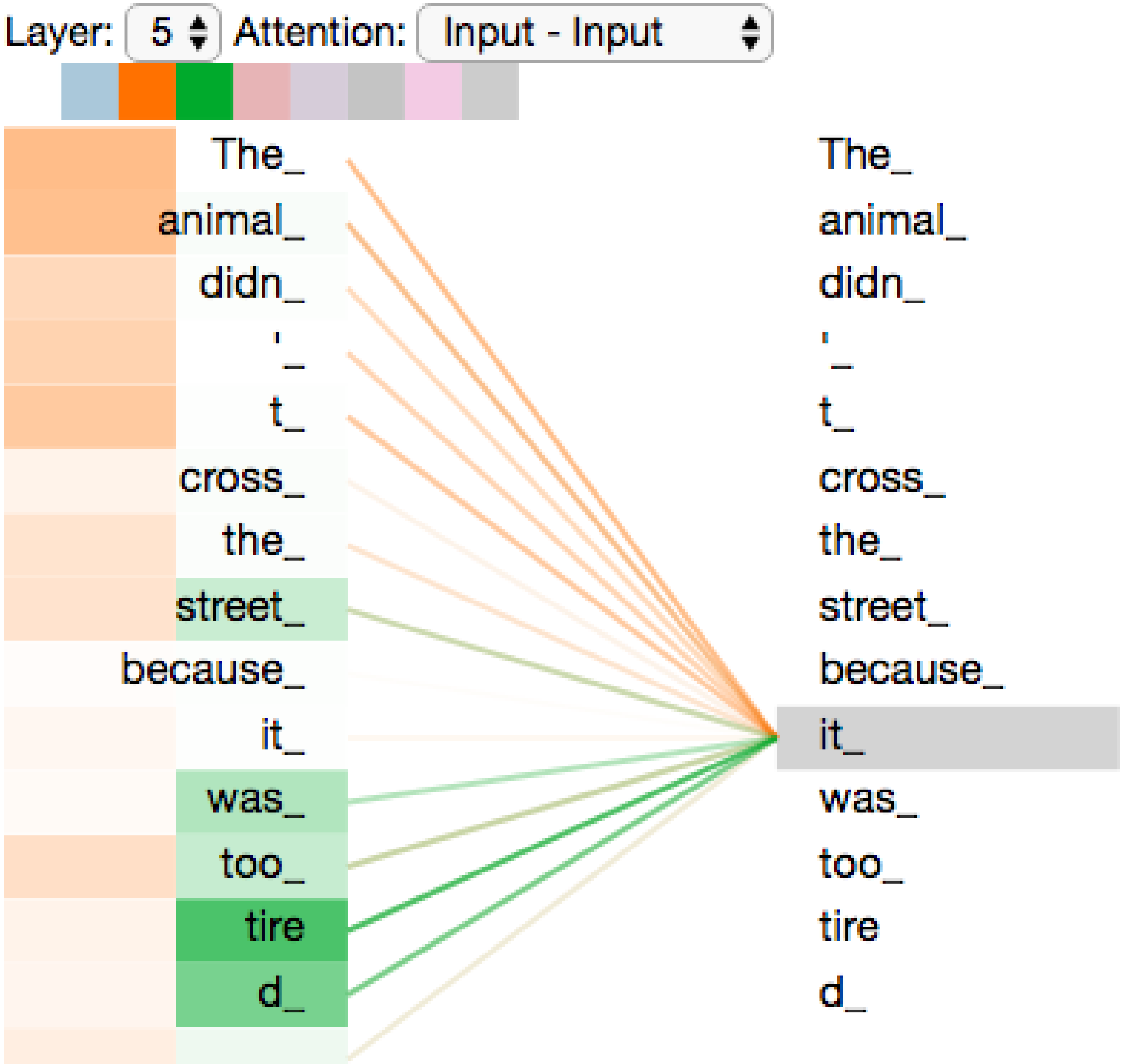
=



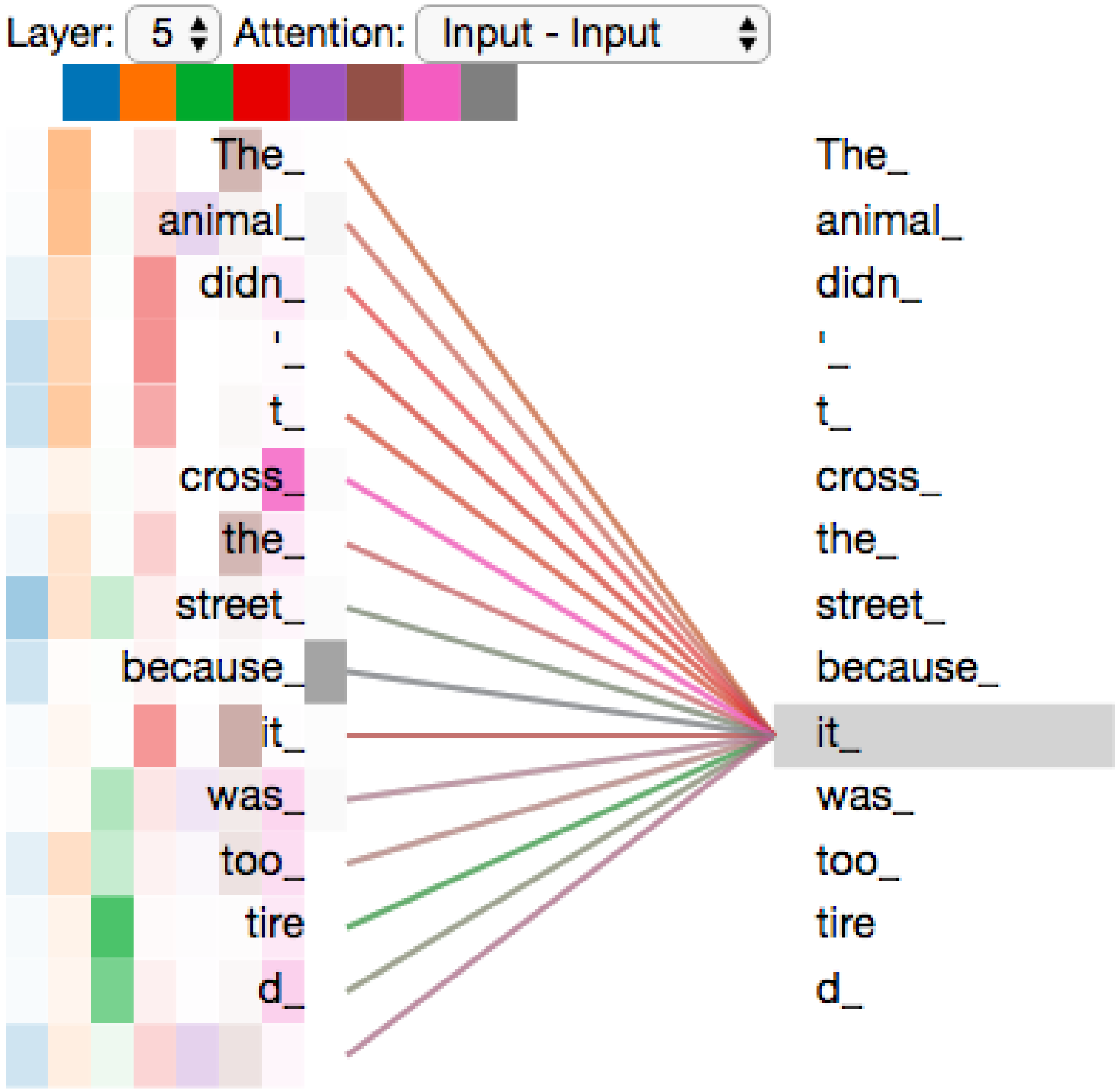


1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0
Z1
Z7
WO
Z

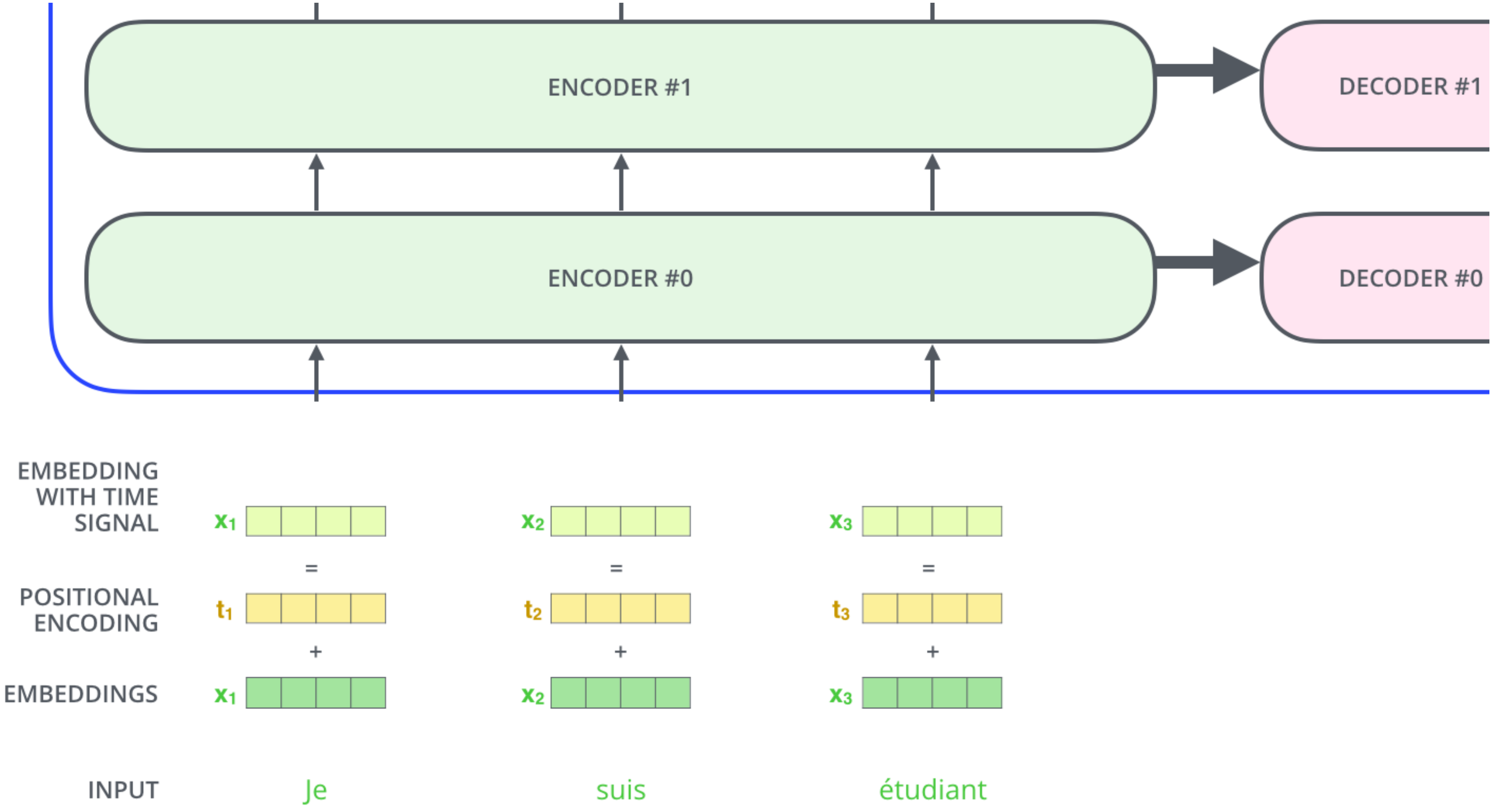
Multi Head Attention



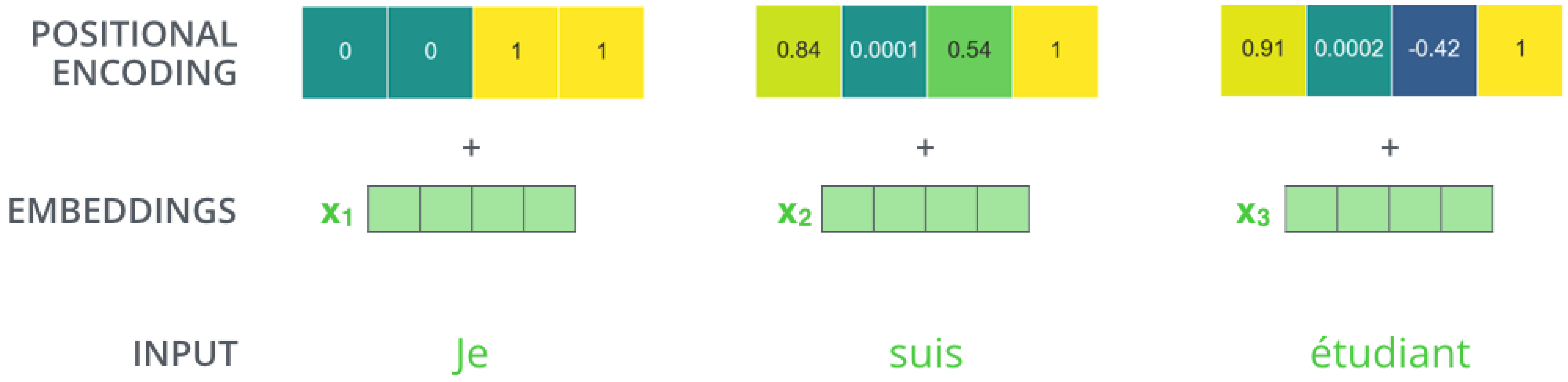
Multi Head Attention



Representing The Order of The Sequence Using Positional Encoding



Representing The Order of The Sequence Using Positional Encoding



Positional Embedding

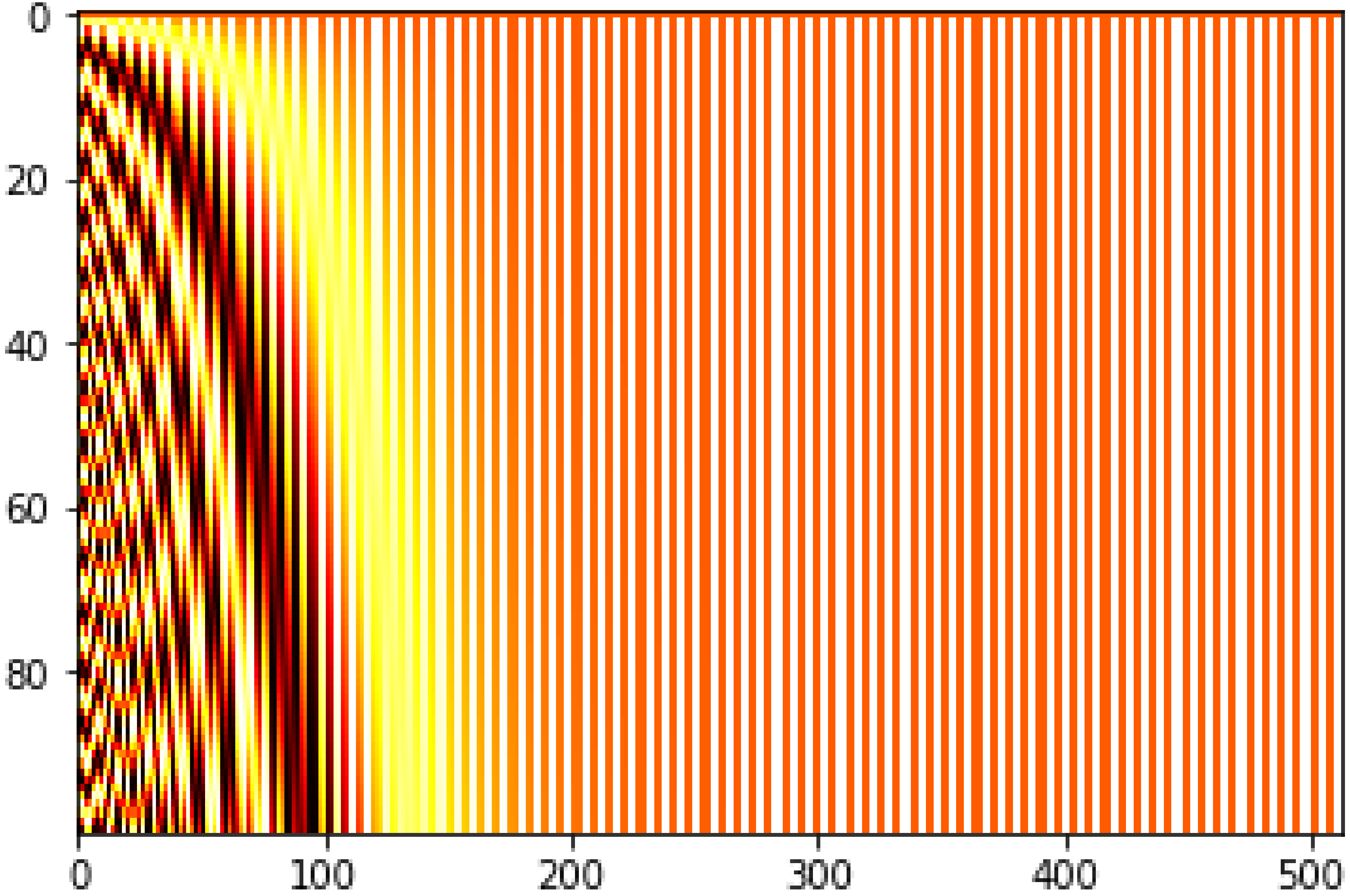
$$e'_w = e_w + \left[sin\left(\frac{pos}{10000^0}\right), cos\left(\frac{pos}{10000^0}\right), sin\left(\frac{pos}{10000^{2/4}}\right), cos\left(\frac{pos}{10000^{2/4}}\right)\right]$$
$$= e_w + \left[sin\,(pos)\,, cos\,(pos)\,, sin\left(\frac{pos}{100}\right), cos\left(\frac{pos}{100}\right)\right]$$

$$\mathrm{PE}(pos, 2i) = sin\left(\frac{pos}{10000^{2i/d_{\mathrm{model}}}}\right),$$

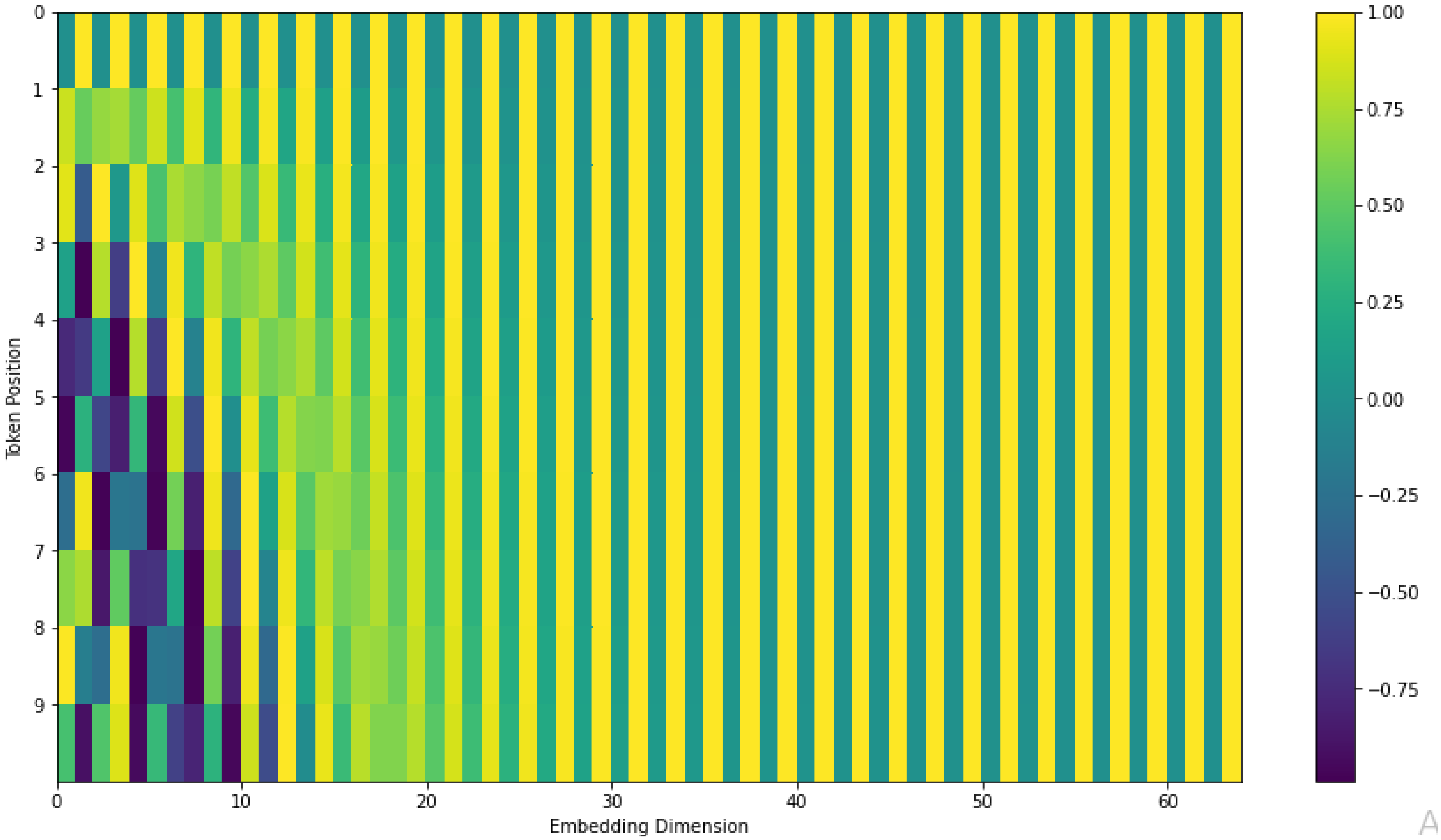
$$\mathrm{PE}(pos, 2i+1) = cos\left(\frac{pos}{10000^{2i/d_{\mathrm{model}}}}\right).$$

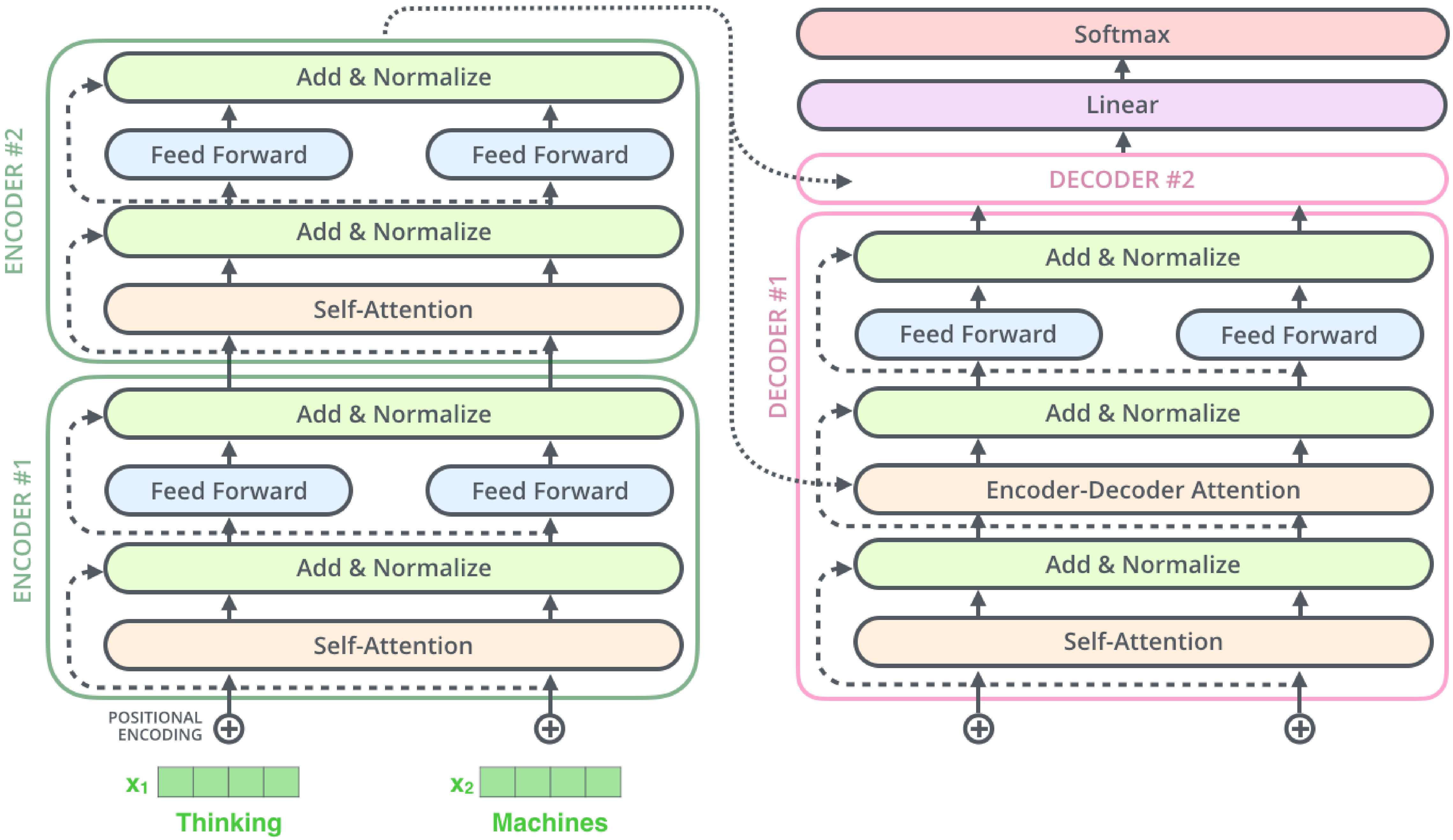
pos = position of a word in window e.g 1,2,3...
i = 512 vector size
d = dimension..idk

Positional Embedding



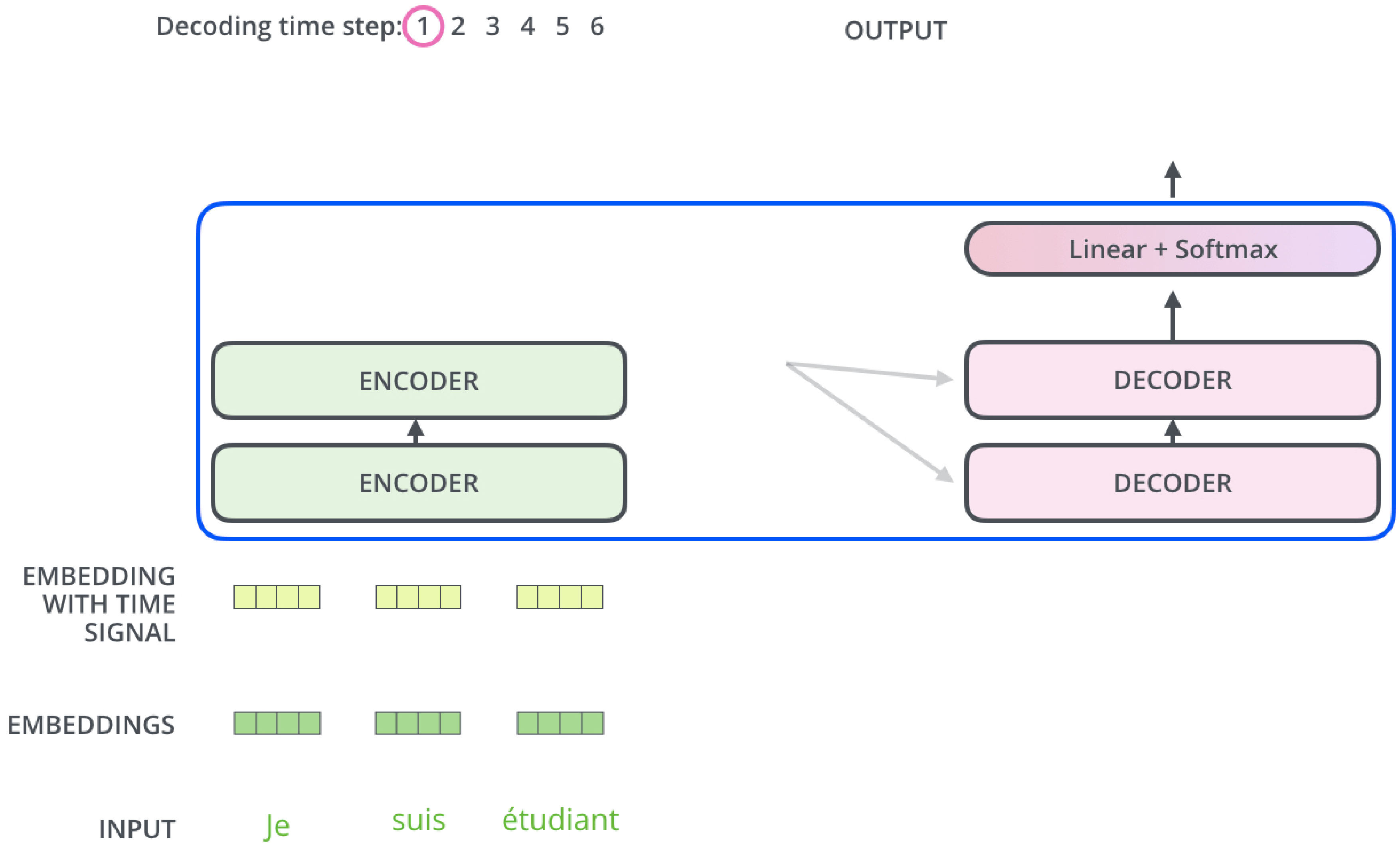
Positional Embedding





Softmax
Linear
DECODER #2
ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x_1
Thinking
x_2
Machines

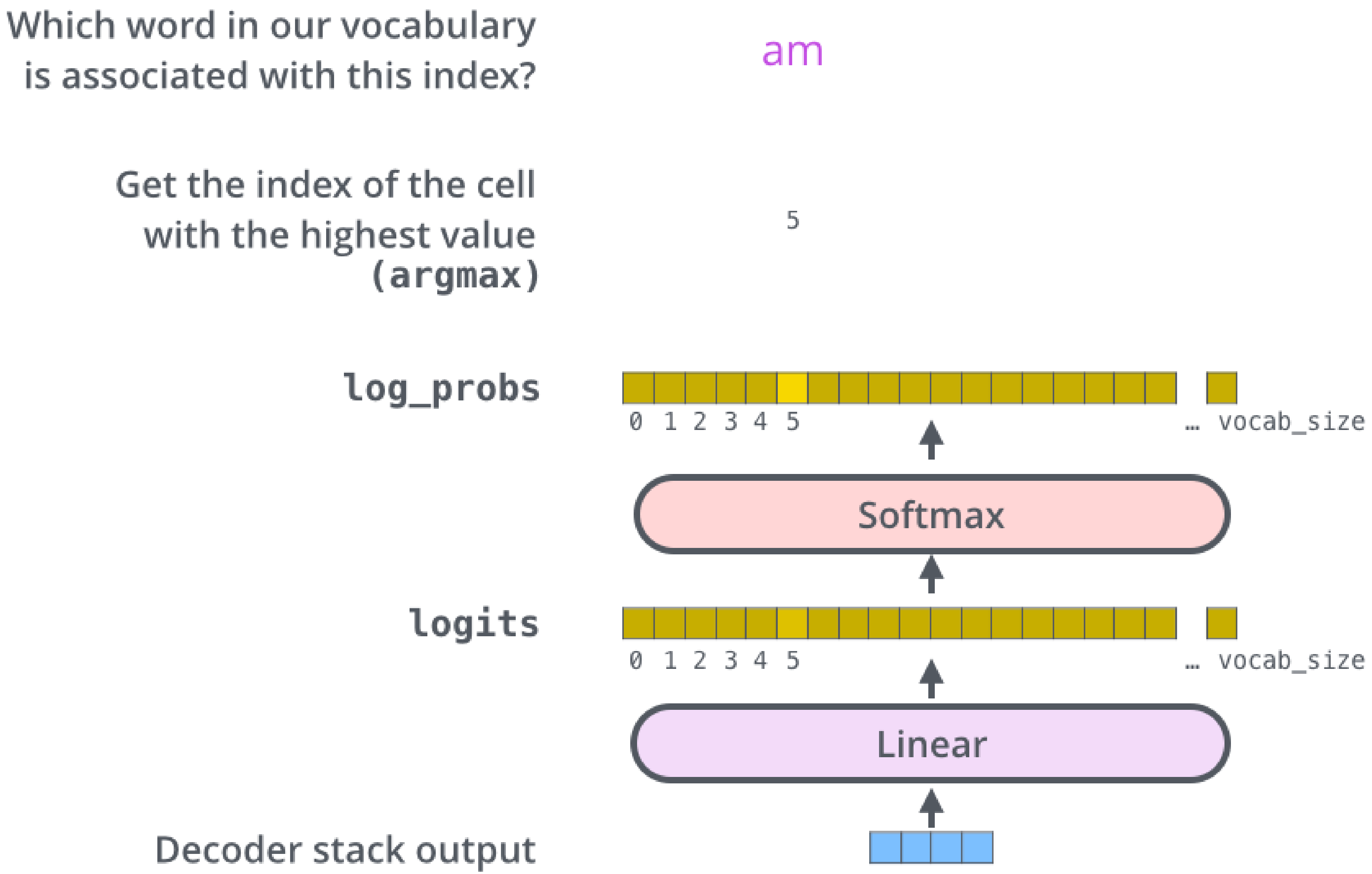
The Decoder Side



Attention in Decoder

- The self attention layers in the decoder operate in a slightly different way than the one in the encoder
- In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation
- The "Encoder-Decoder Attention" layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack

The Final Linear and Softmax Layer



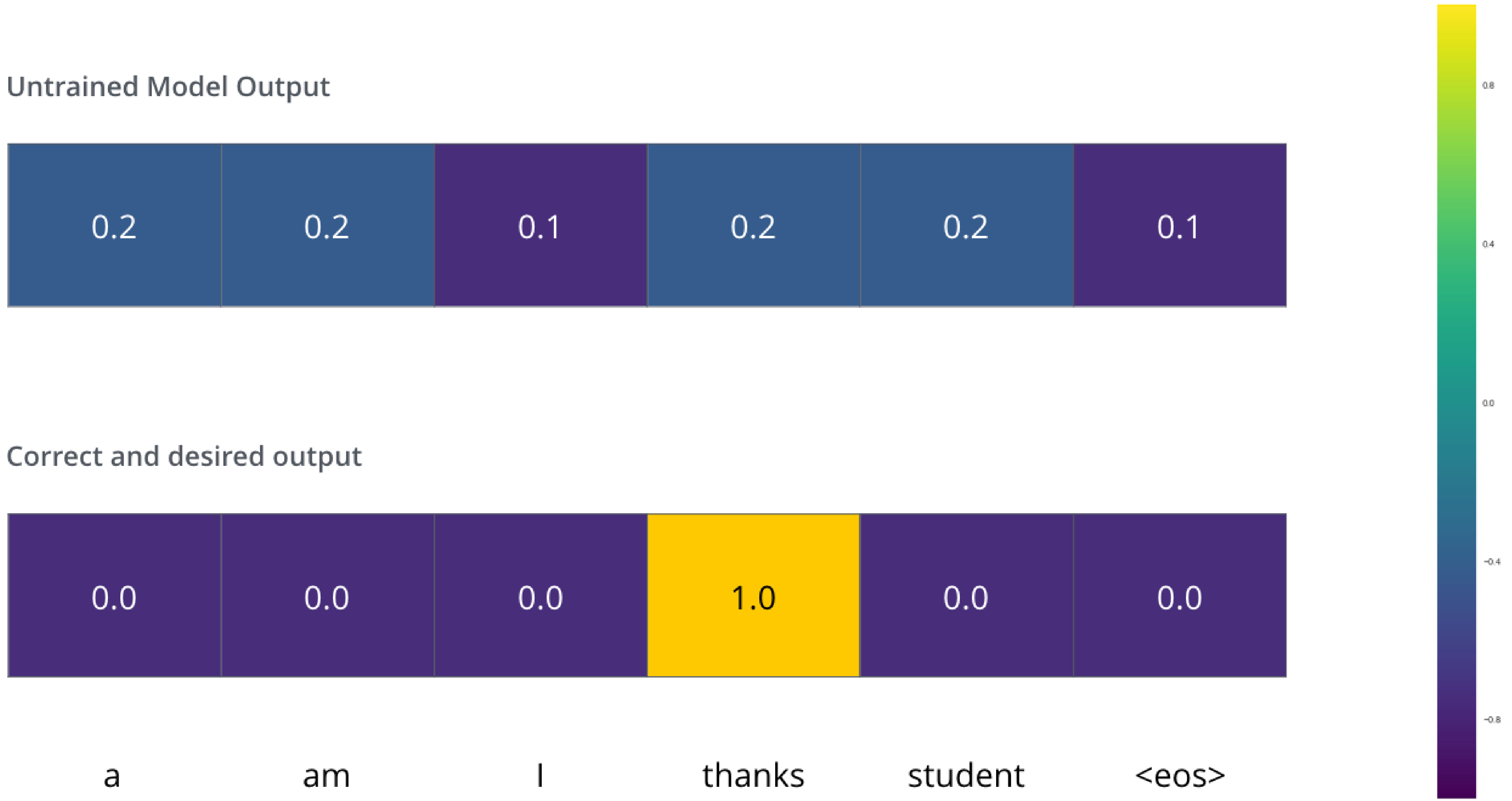
Recap Of Training

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

One-hot encoding of the word “am”

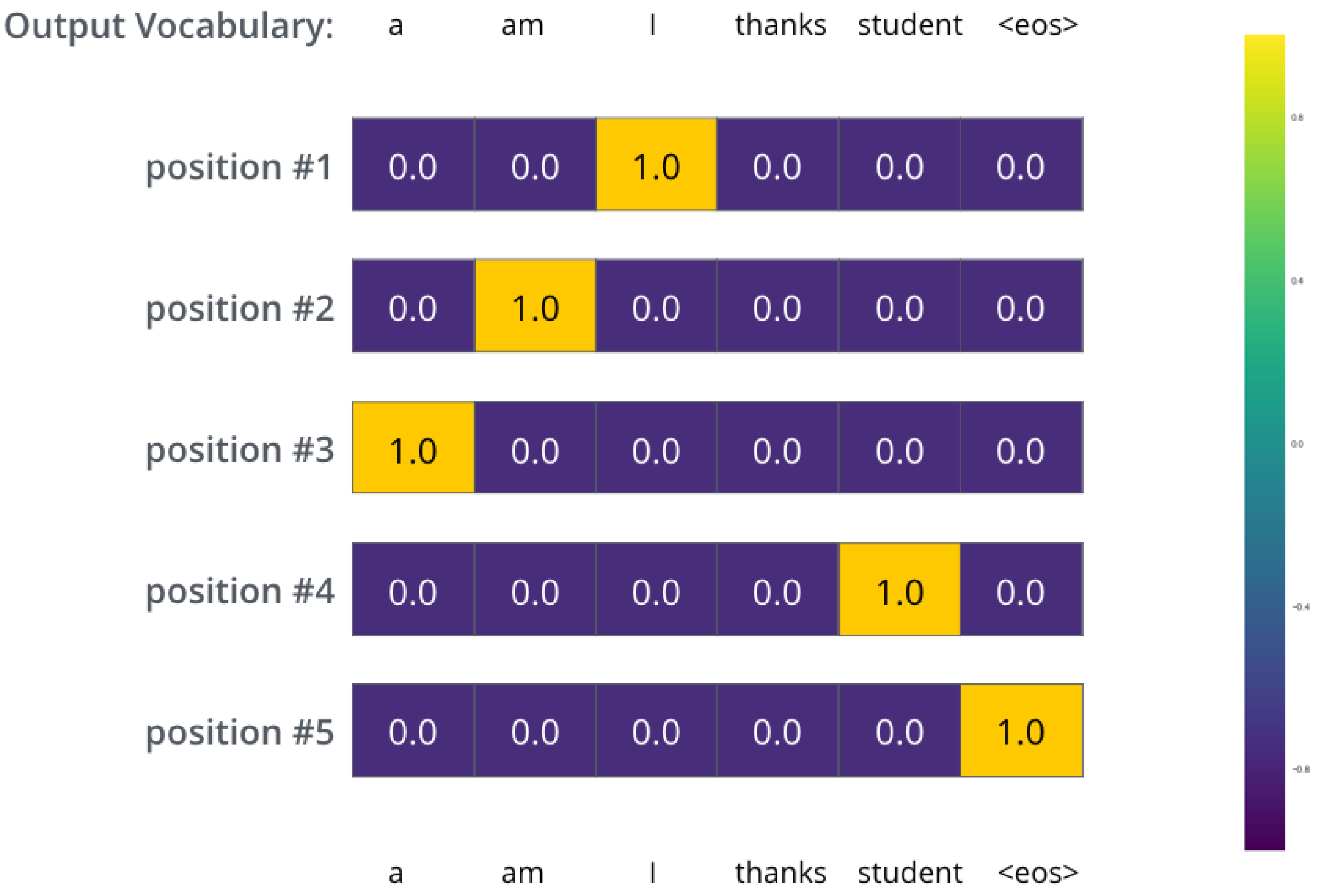
0.0	1.0	0.0	0.0	0.0	0.0


Untrained Model Output
0.2
0.2
0.1
0.2
0.2
0.1
Correct and desired output
0.0
0.0
0.0
1.0
0.0
0.0
a
am
I
thanks
student
<eos>
0.8
0.4
0.0
−0.4
−0.8

The Loss Function

- Each probability distribution is represented by a vector of width vocab_size (6 in our toy example, but more realistically a number like 3,000 or 10,000)
- The first probability distribution has the highest probability at the cell associated with the word “i”
- The second probability distribution has the highest probability at the cell associated with the word “am”
- And so on, until the fifth output distribution indicates

Target Model Outputs



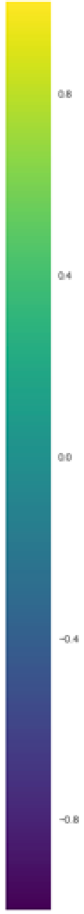
Trained Model Outputs

Beam Search

Output Vocabulary:

	a	am	I	thanks	student	<eos>
position #1	0.01	0.02	0.93	0.01	0.03	0.01
position #2	0.01	0.8	0.1	0.05	0.01	0.03
position #3	0.99	0.001	0.001	0.001	0.002	0.001
position #4	0.001	0.002	0.001	0.02	0.94	0.01
position #5	0.01	0.01	0.001	0.001	0.001	0.98

a am I thanks student <eos>



References

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

- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. **Attention is all you need**. In Proceedings of *the 31st International Conference on Neural Information Processing Systems* (NIPS'17),