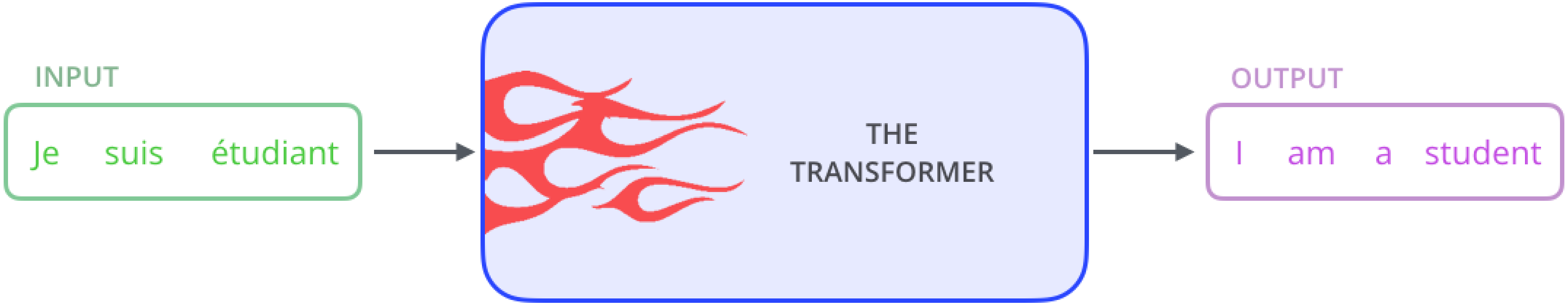
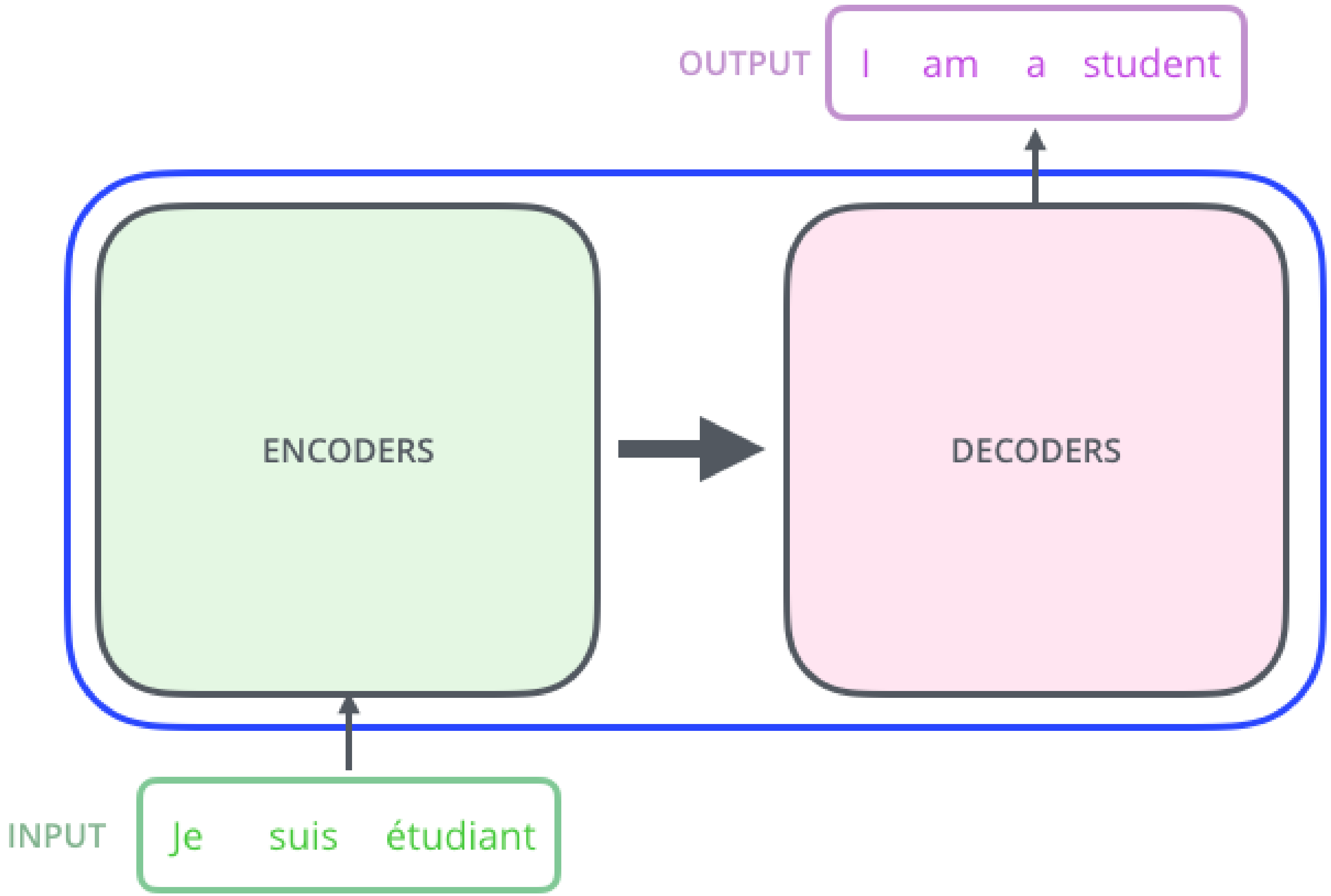
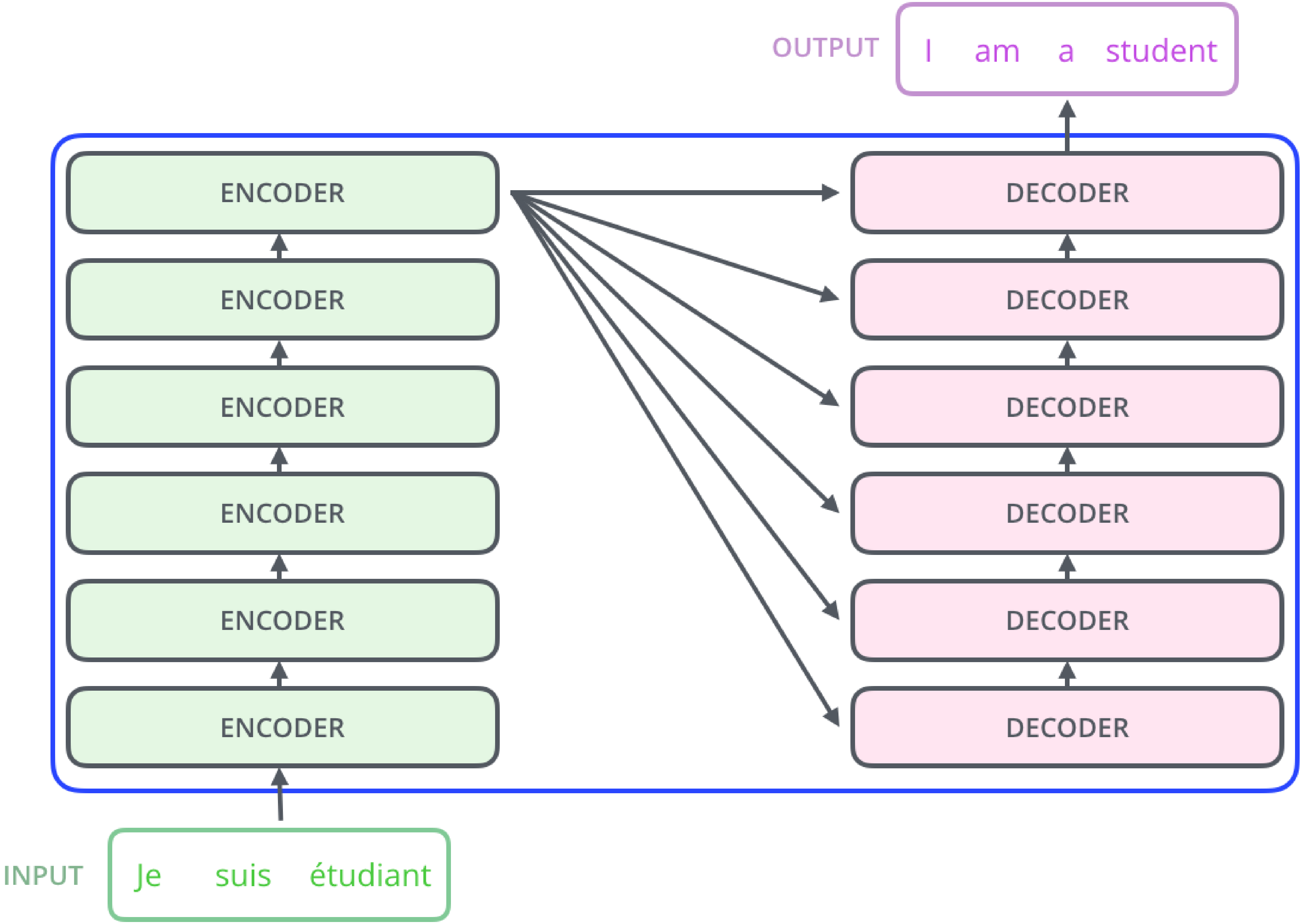
Transformer
(Attention is All You Need)

Transformer (A High-Level Look)

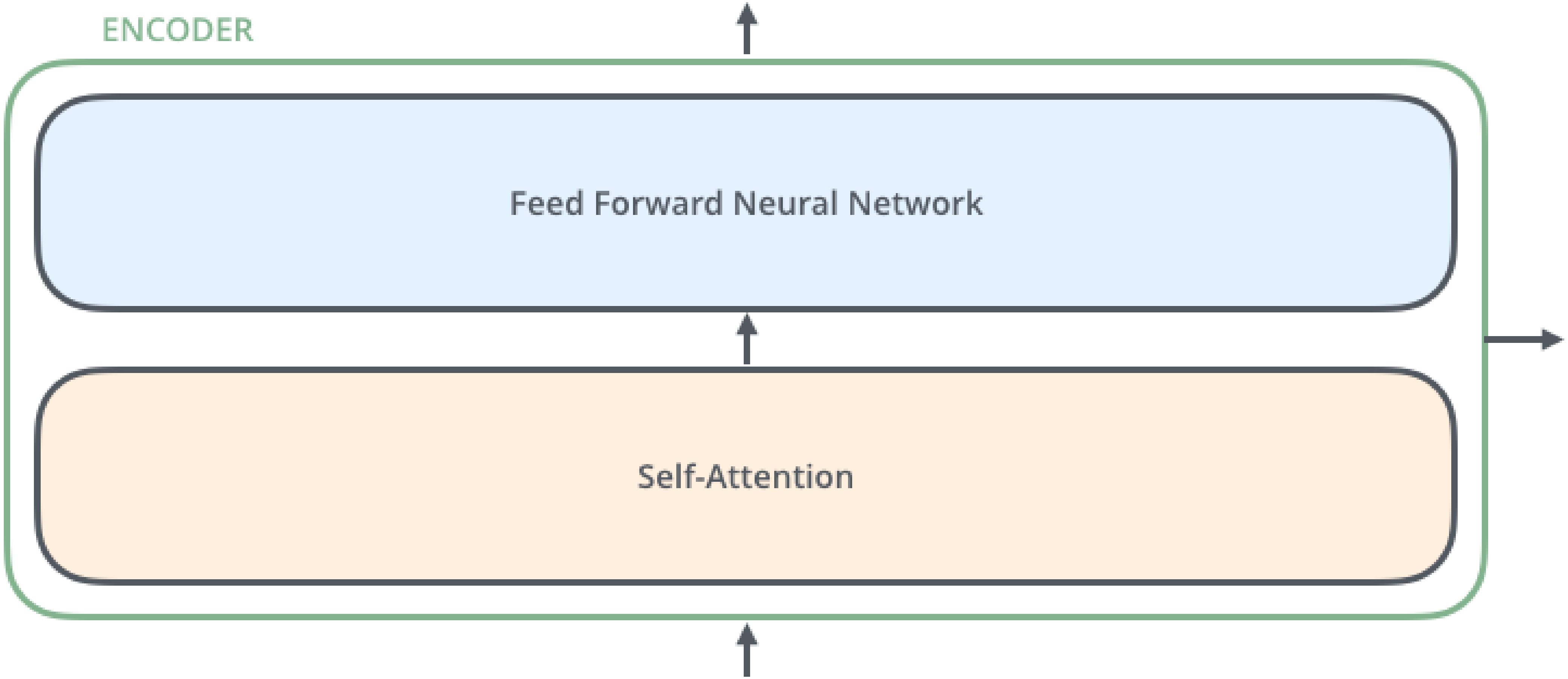


Transformer (**A High-Level Look**)

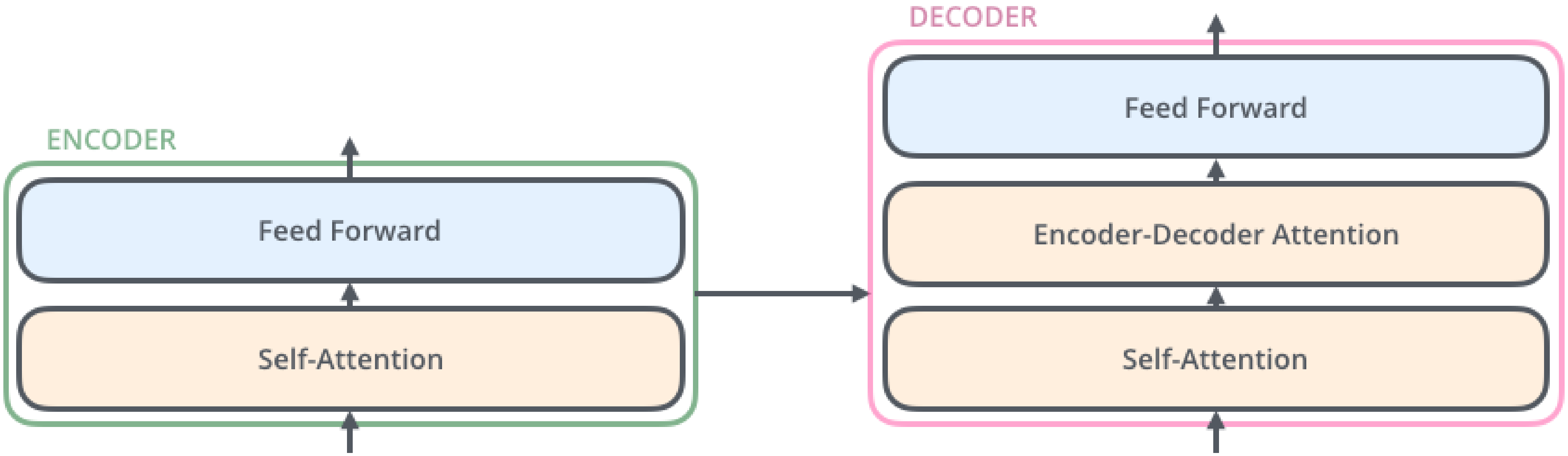



OUTPUT
I am a student
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
INPUT
Je suis étudiant

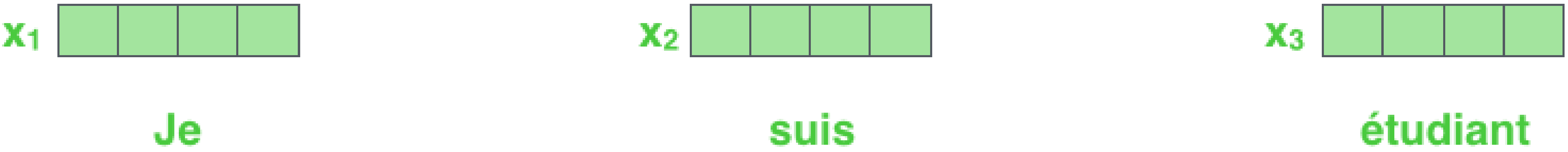
The encoders are all identical in structure. Each one is broken down into two sub-layers:

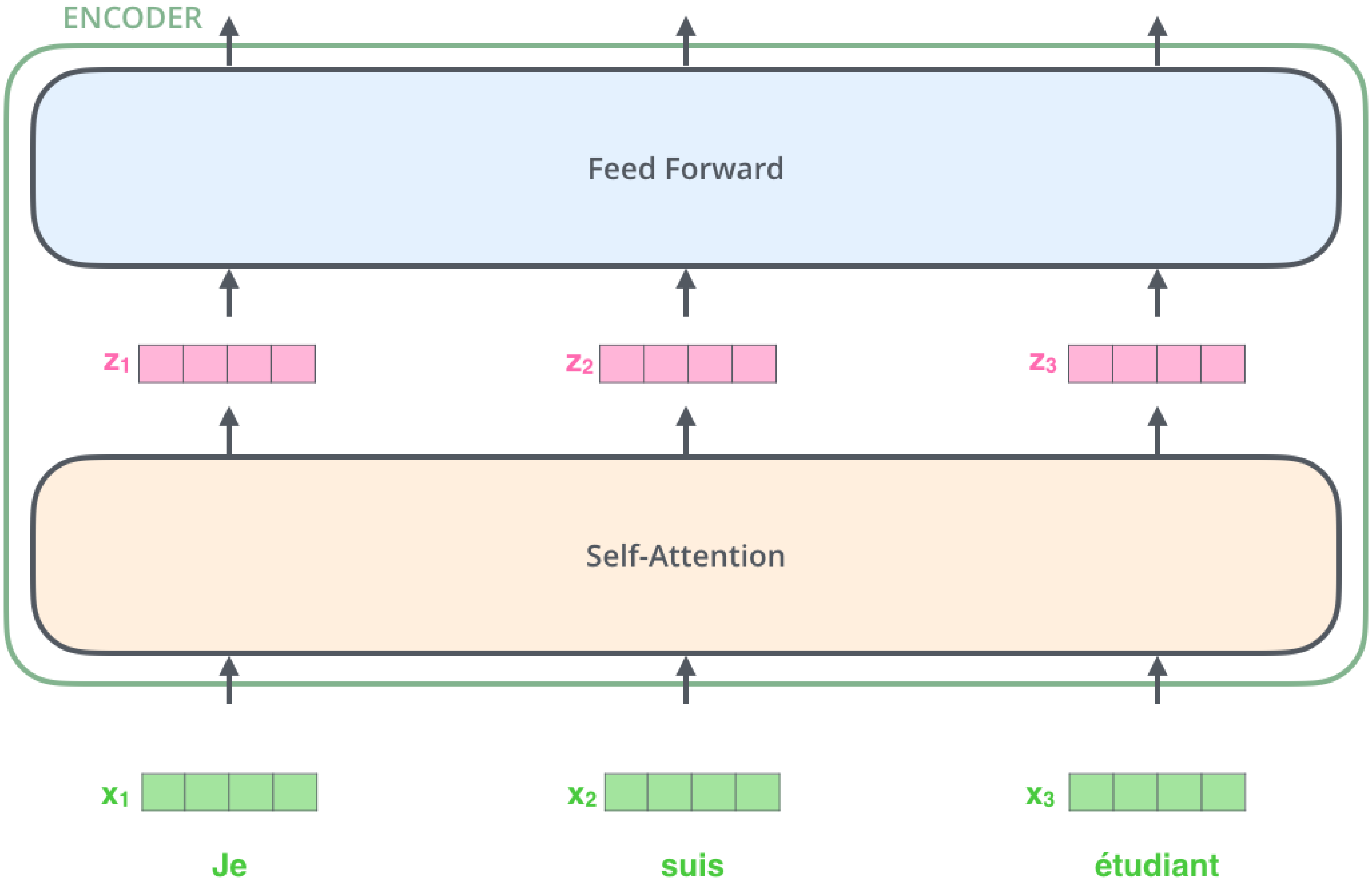


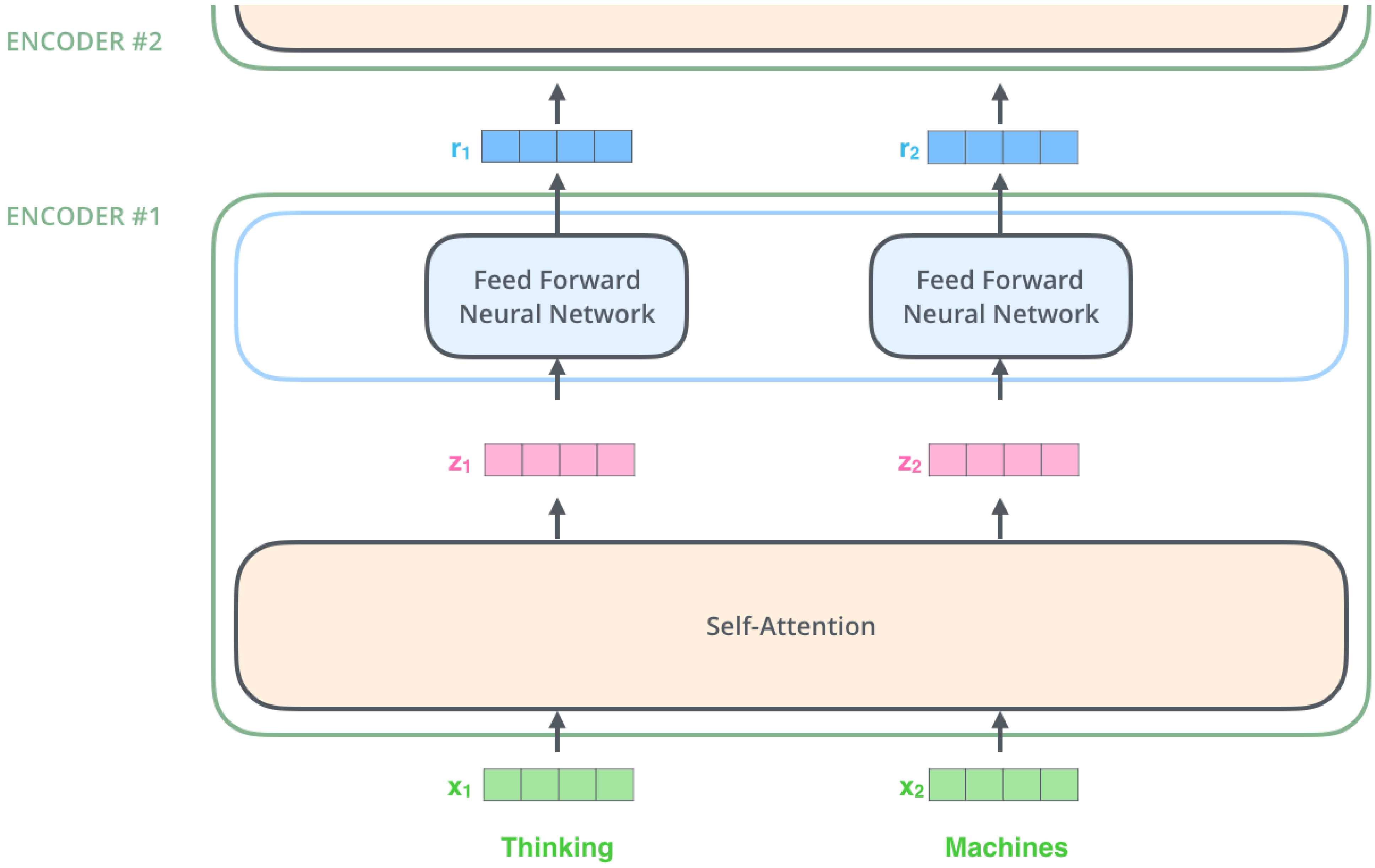
The encoders are all identical in structure. Each one is broken down into two sub-layers:



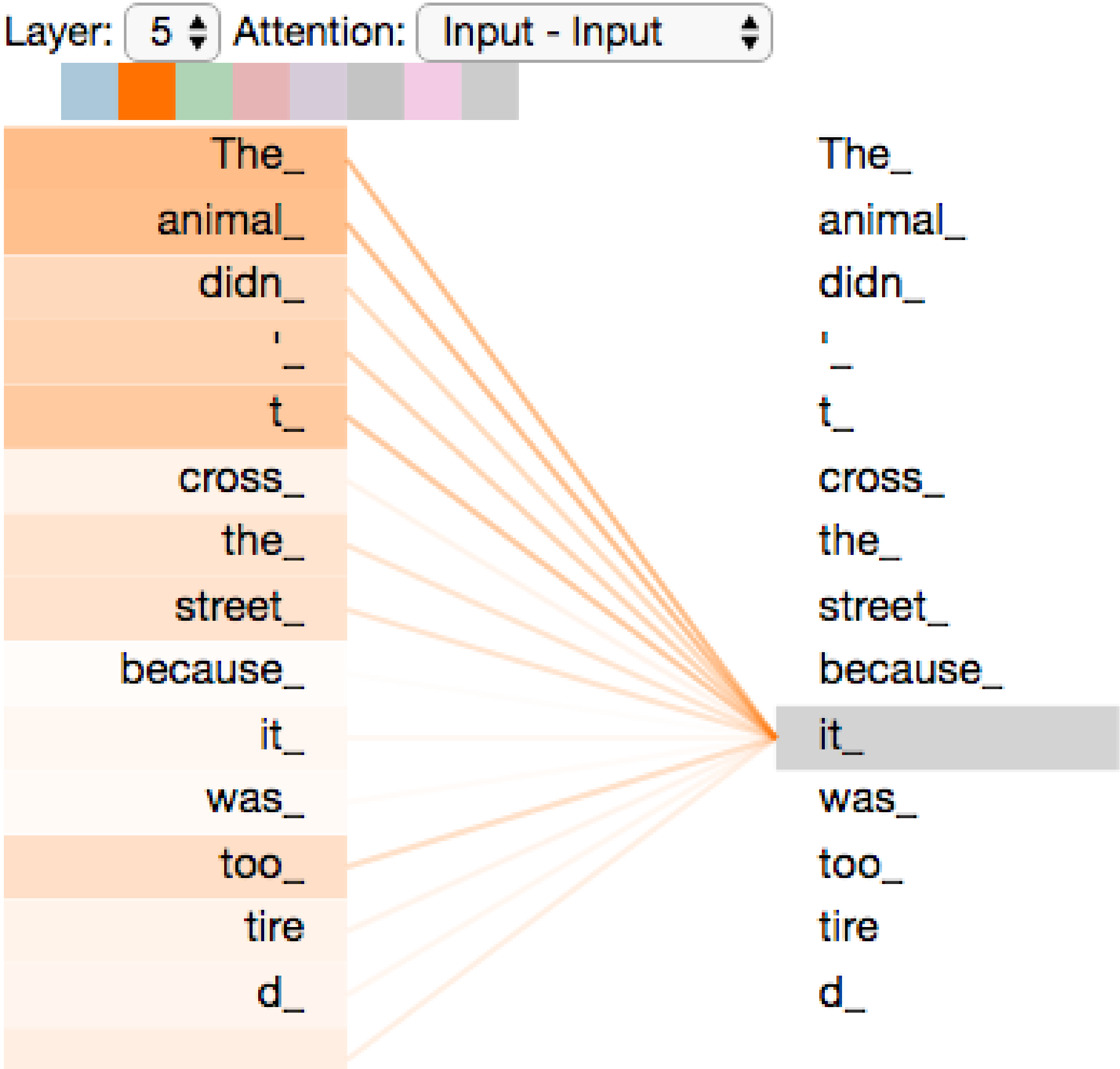
Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.



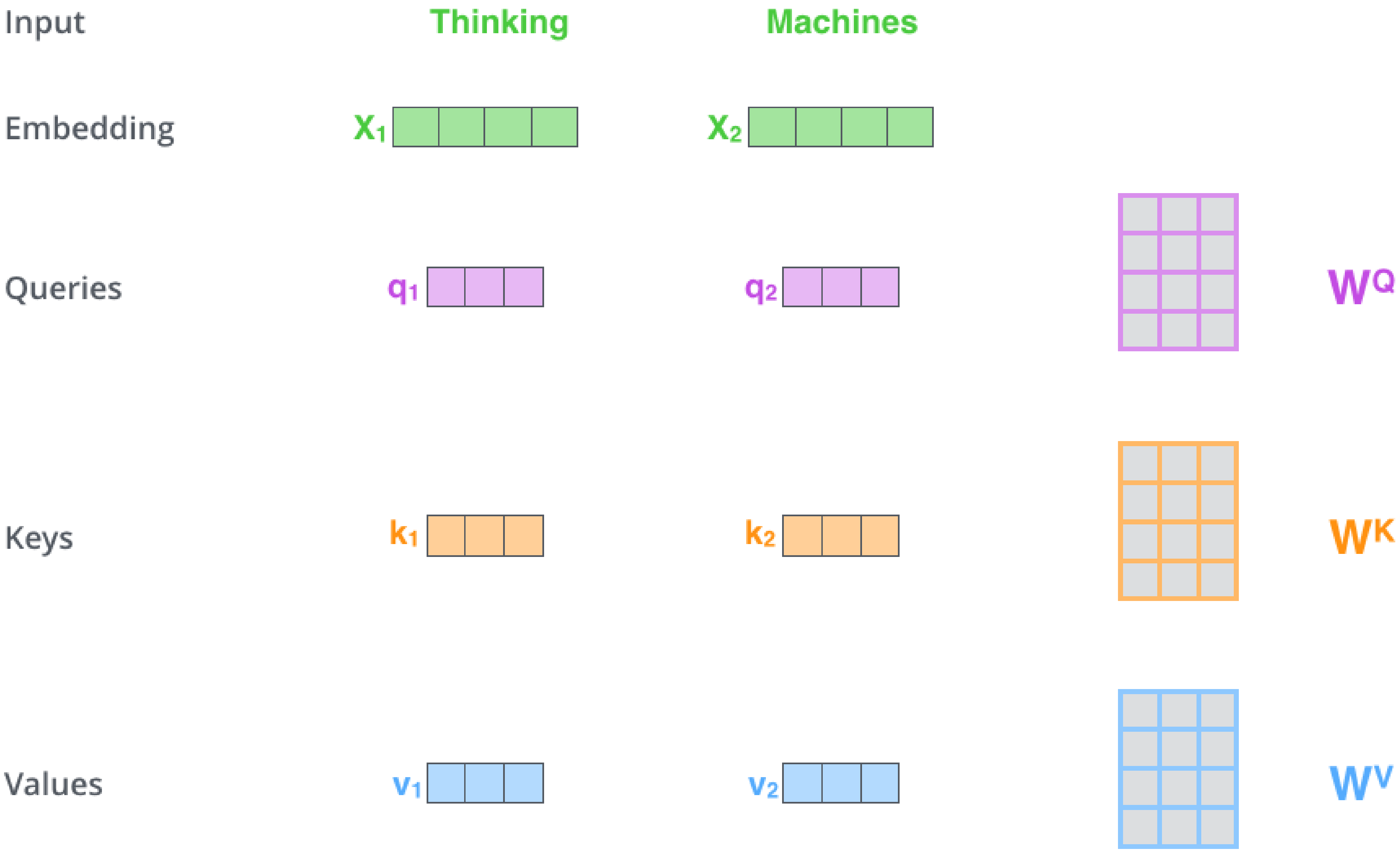

ENCODER
Feed Forward
z_1
z_2
z_3
Self-Attention
x_1
x_2
x_3
Je
suis
étudiant


ENCODER #2
ENCODER #1
r_1
r_2
Feed Forward Neural Network
Feed Forward Neural Network
z_1
z_2
Self-Attention
x_1
x_2
Thinking
Machines

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



Layer: 5 Attention: Input - Input



Input

Thinking

Machines

Embedding

X_1

X_2

Queries

q_1

q_2

W^Q

Keys

k_1

k_2

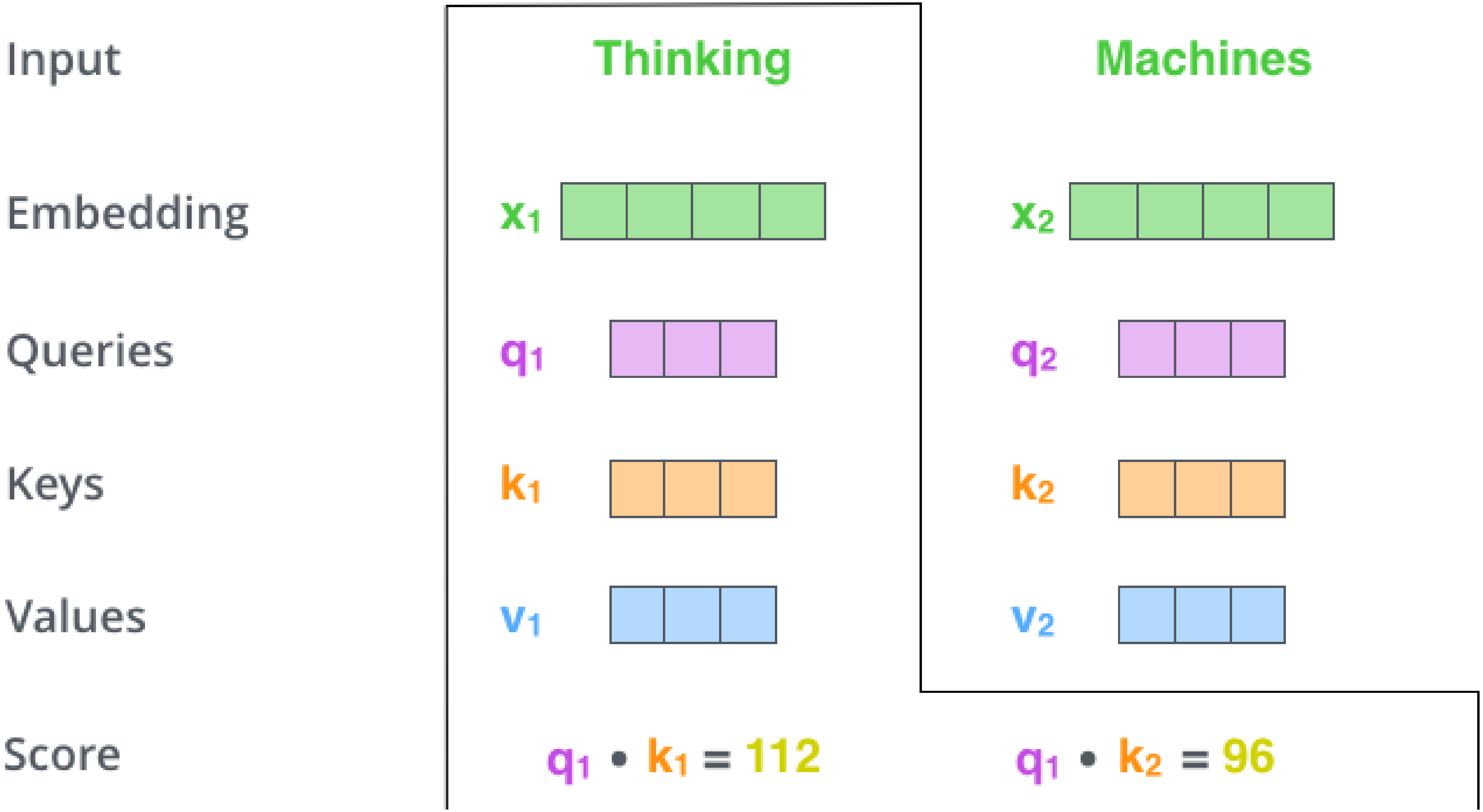
W^K

Values

v_1

v_2

W^V



Input Thinking Machines

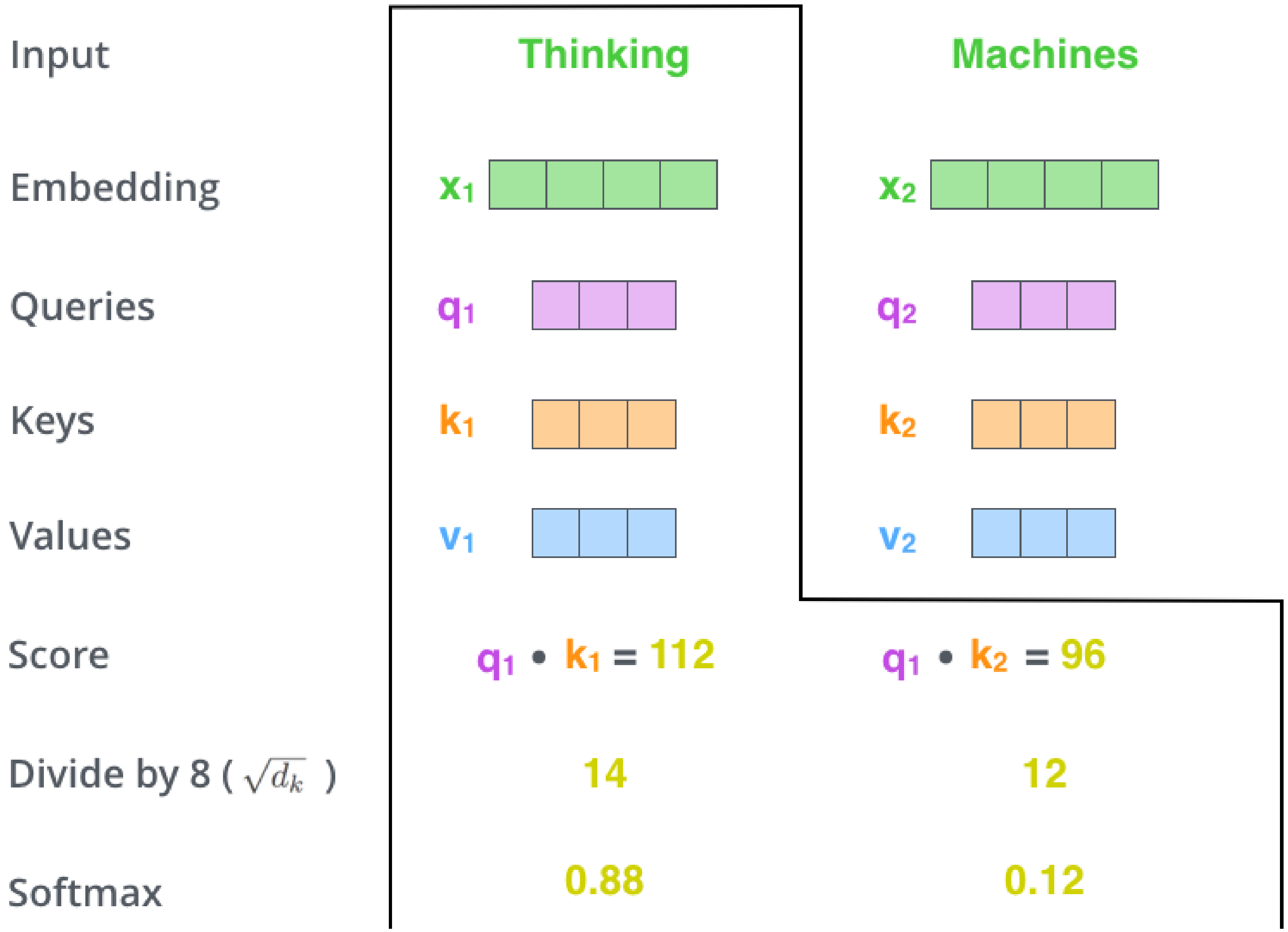
Embedding x_1 x_2

Queries q_1 q_2

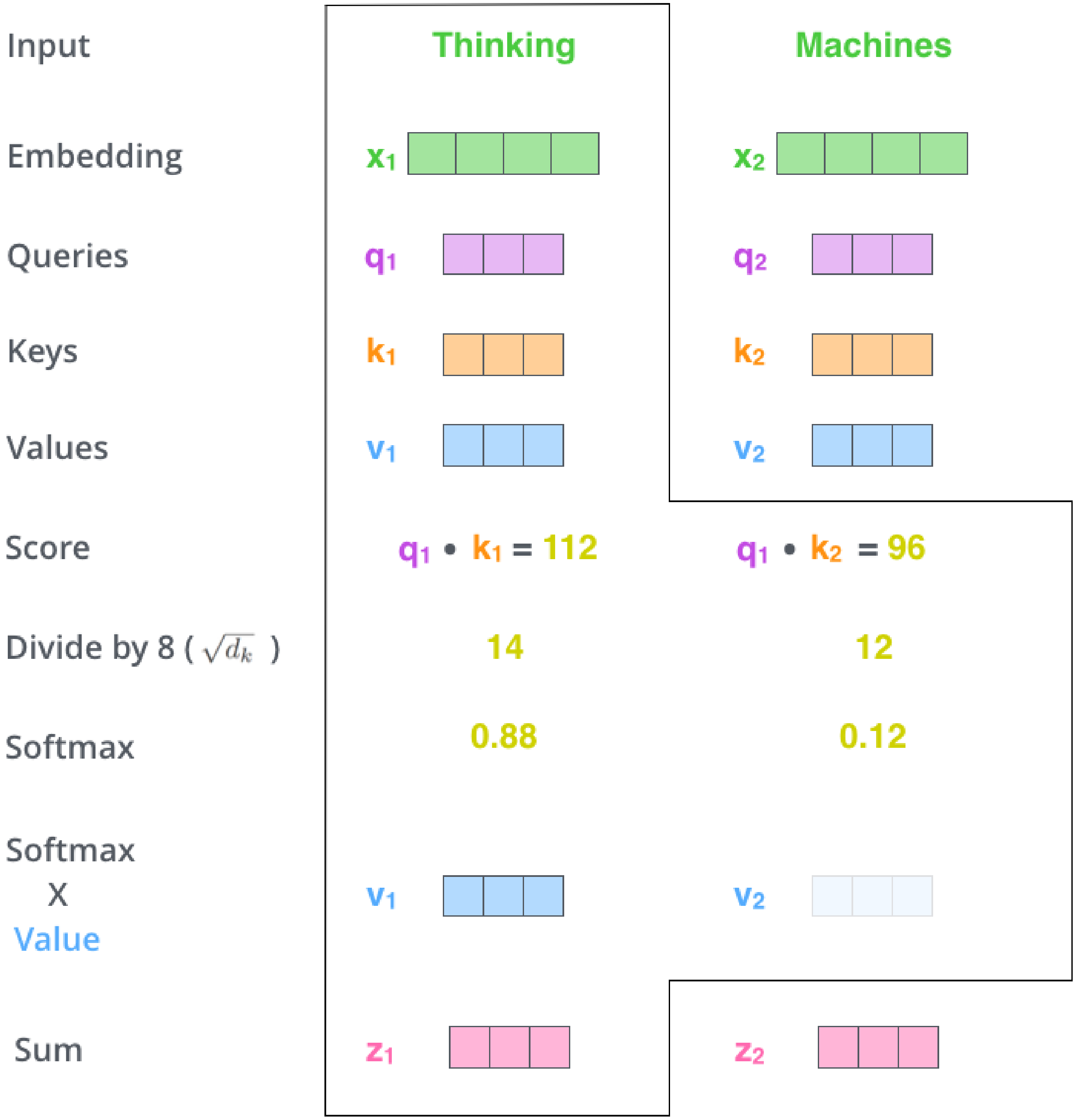
Keys k_1 k_2

Values v_1 v_2

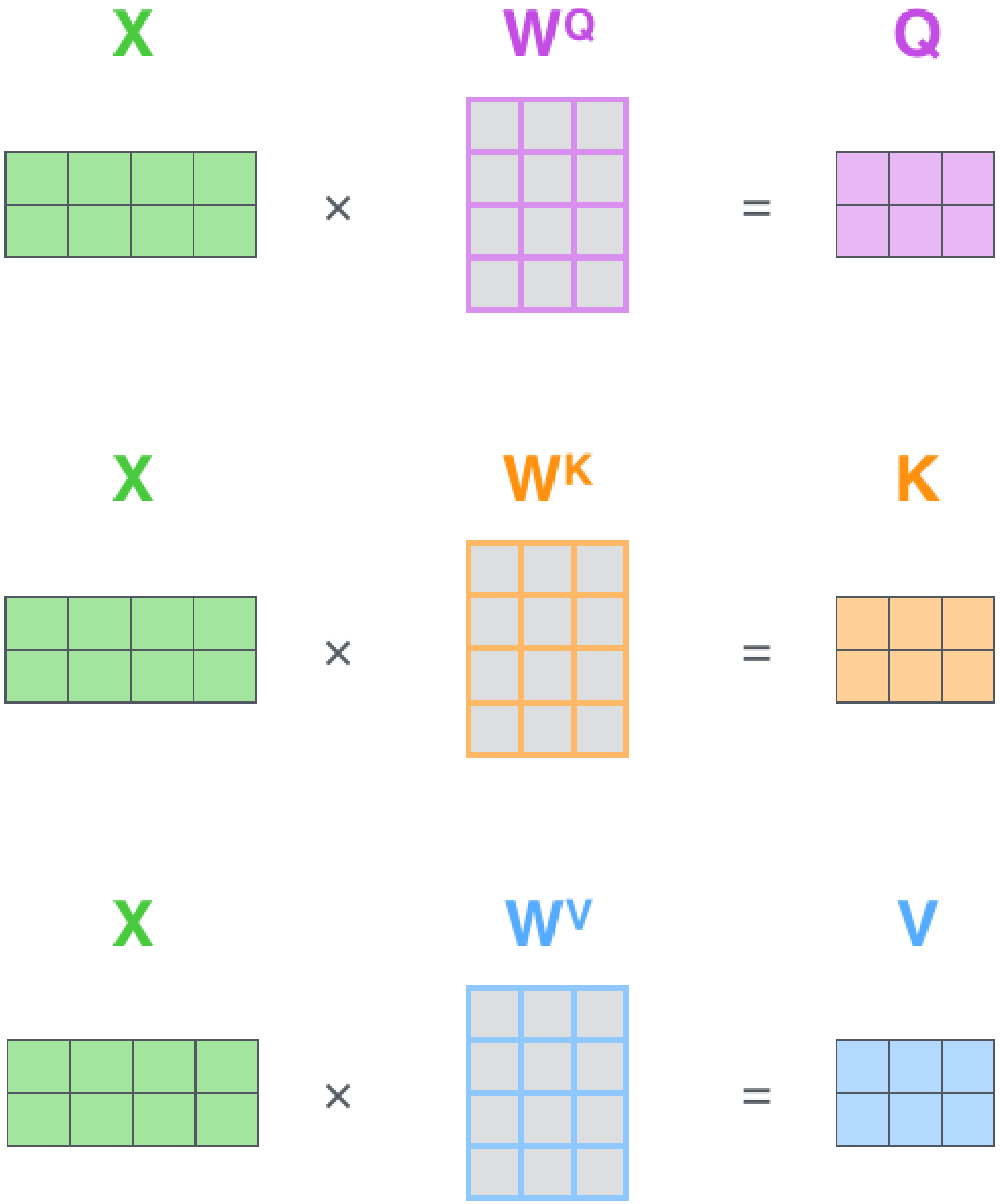
Score $q_1 \cdot k_1 = 112$ $q_1 \cdot k_2 = 96$



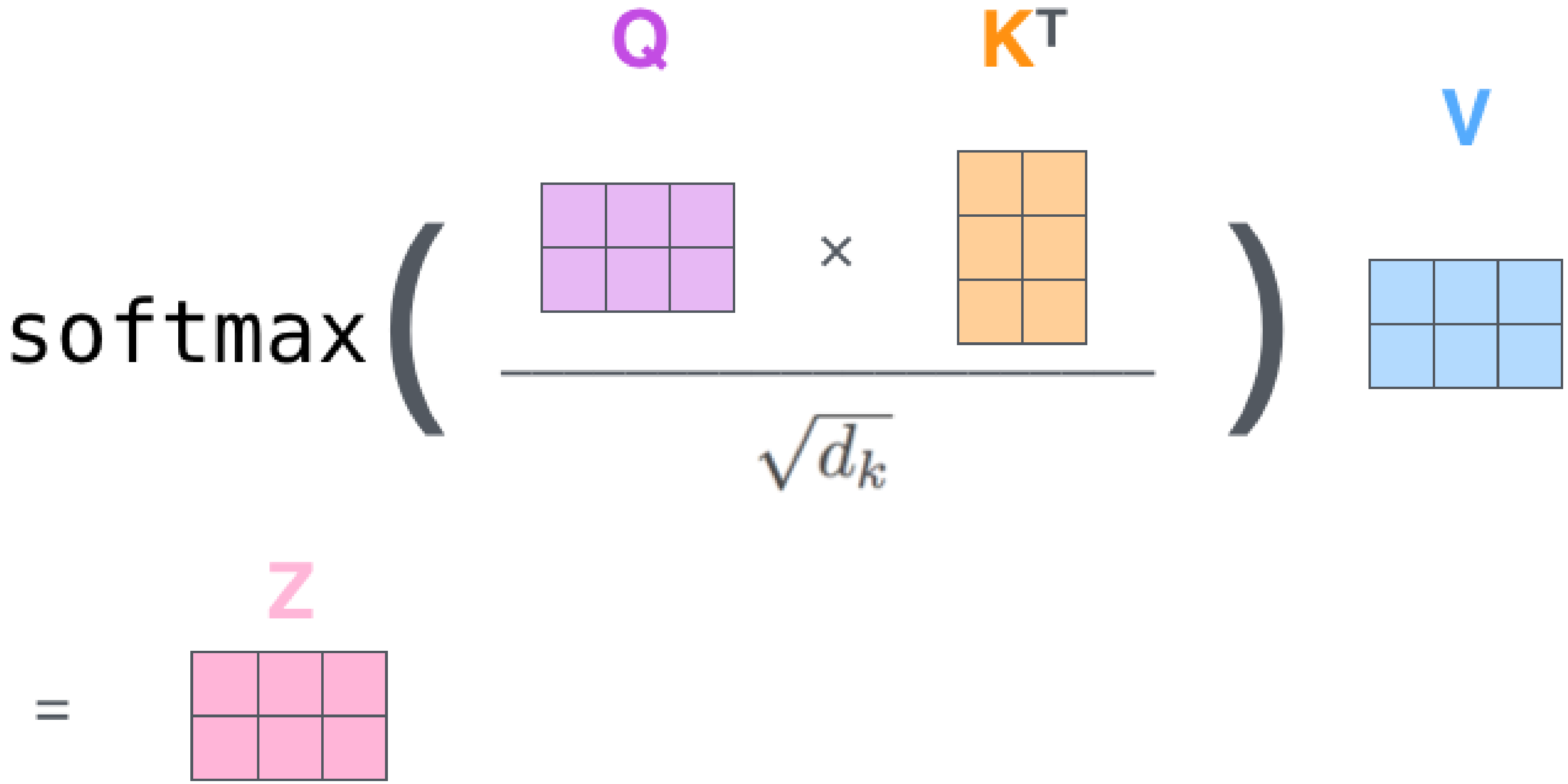
Input	Thinking	Machines
Embedding	x_1	x_2
Queries	q_1	q_2
Keys	k_1	k_2
Values	v_1	v_2
Score	$q_1 \bullet k_1 = 112$	$q_1 \bullet k_2 = 96$
Divide by 8 ($\sqrt{d_k}$)	14	12
Softmax	0.88	0.12



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


X
×
W^Q
=
Q
X
×
W^K
=
K
X
×
W^V
=
V

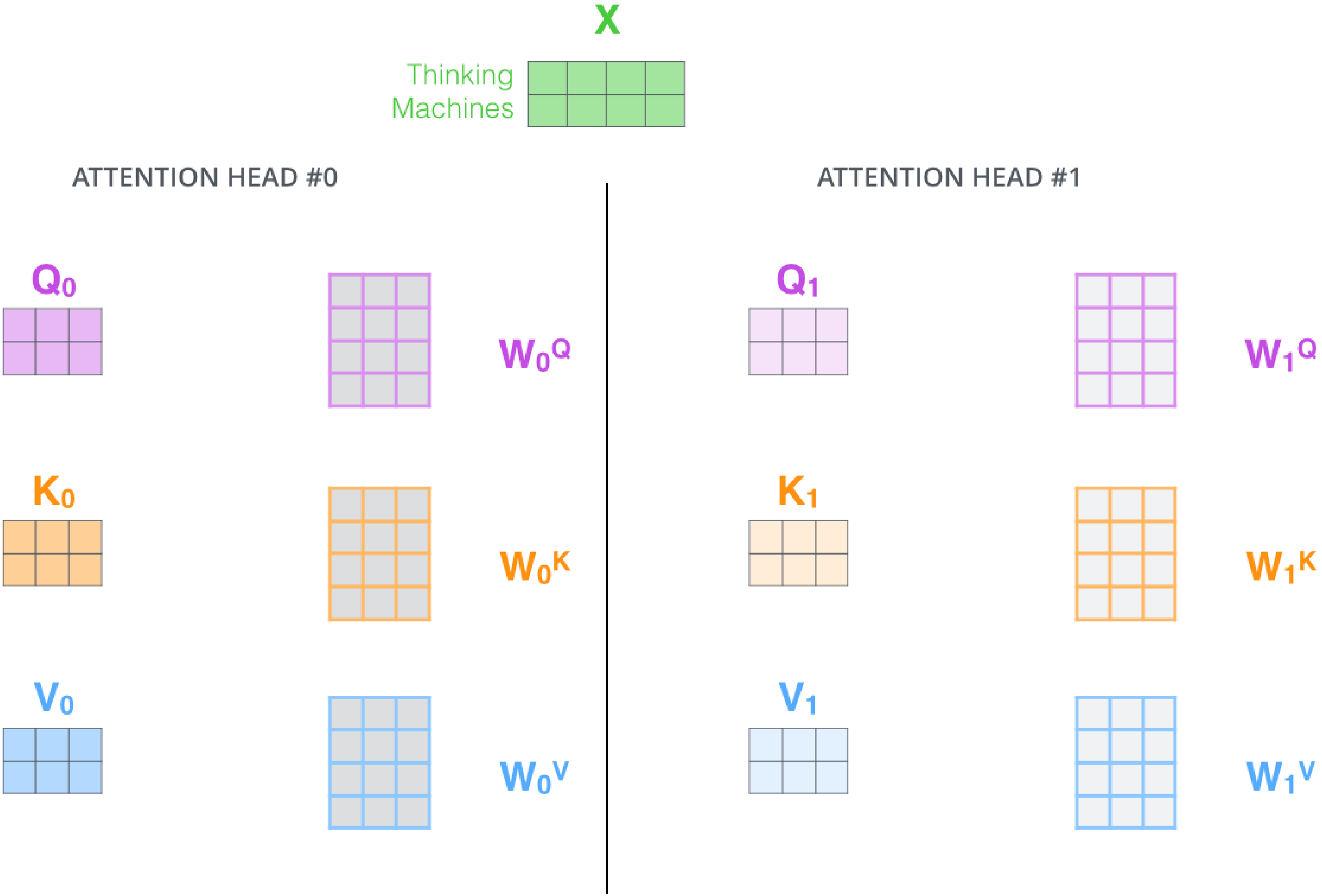
The self-attention calculation in matrix form

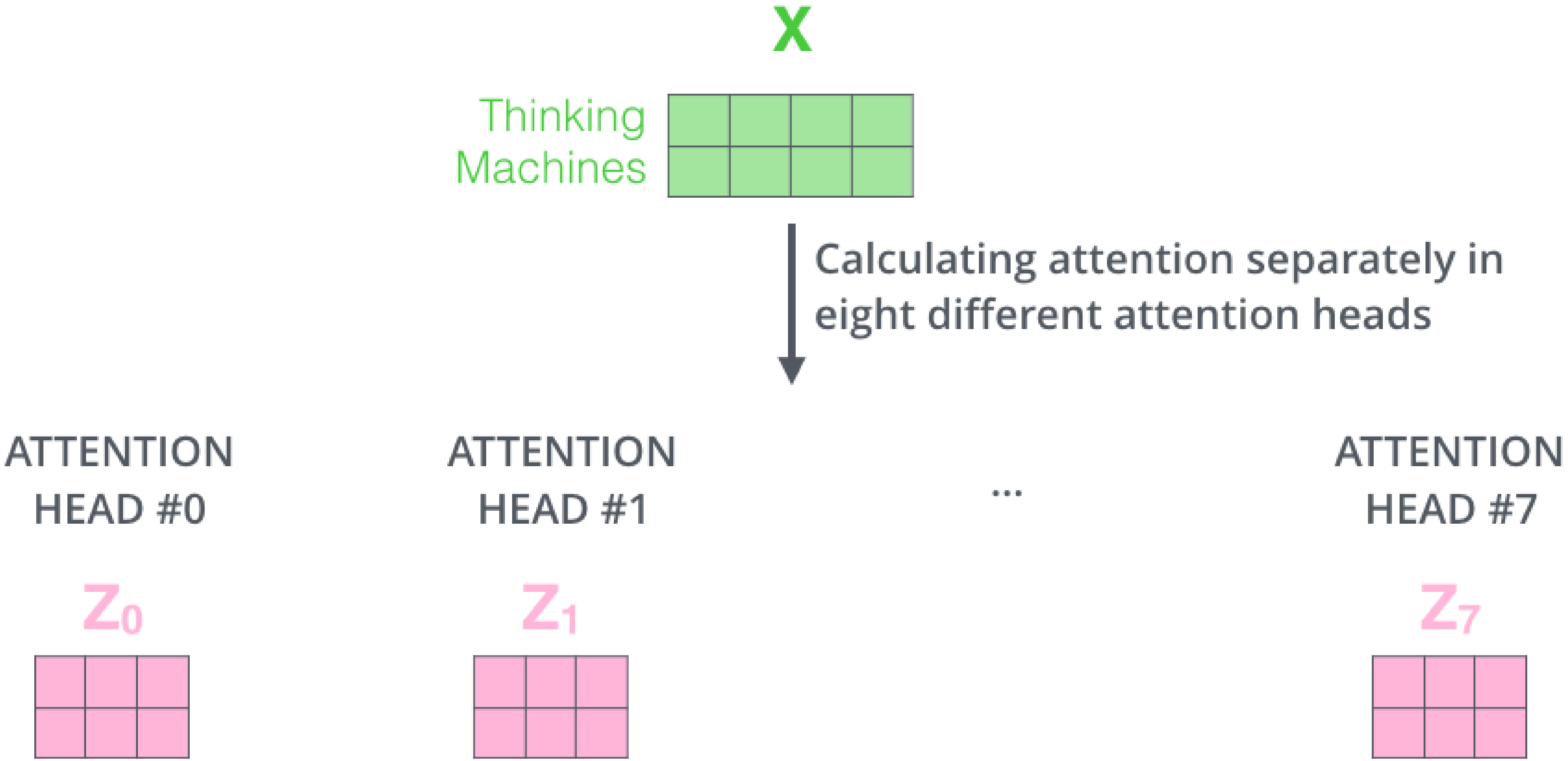


The Beast With Many Heads
Multi Head Attention

- 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
Q_0
W_0^Q
K_0
W_0^K
V_0
W_0^V
Q_1
W_1^Q
K_1
W_1^K
V_1
W_1^V



X

Thinking
Machines

Calculating attention separately in eight different attention heads

ATTENTION HEAD #0

ATTENTION HEAD #1

...

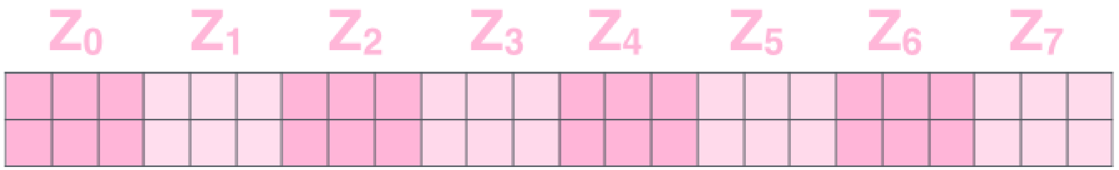
ATTENTION HEAD #7

Z_0

Z_1

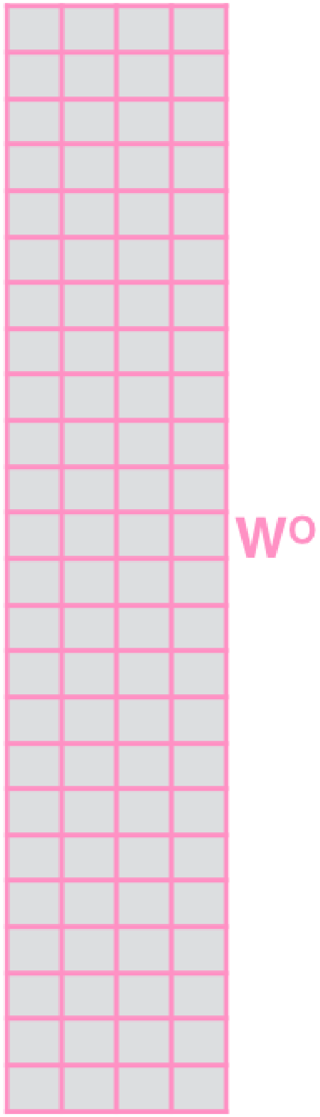
Z_7

1) Concatenate all the attention heads



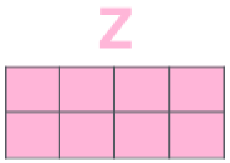
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

=



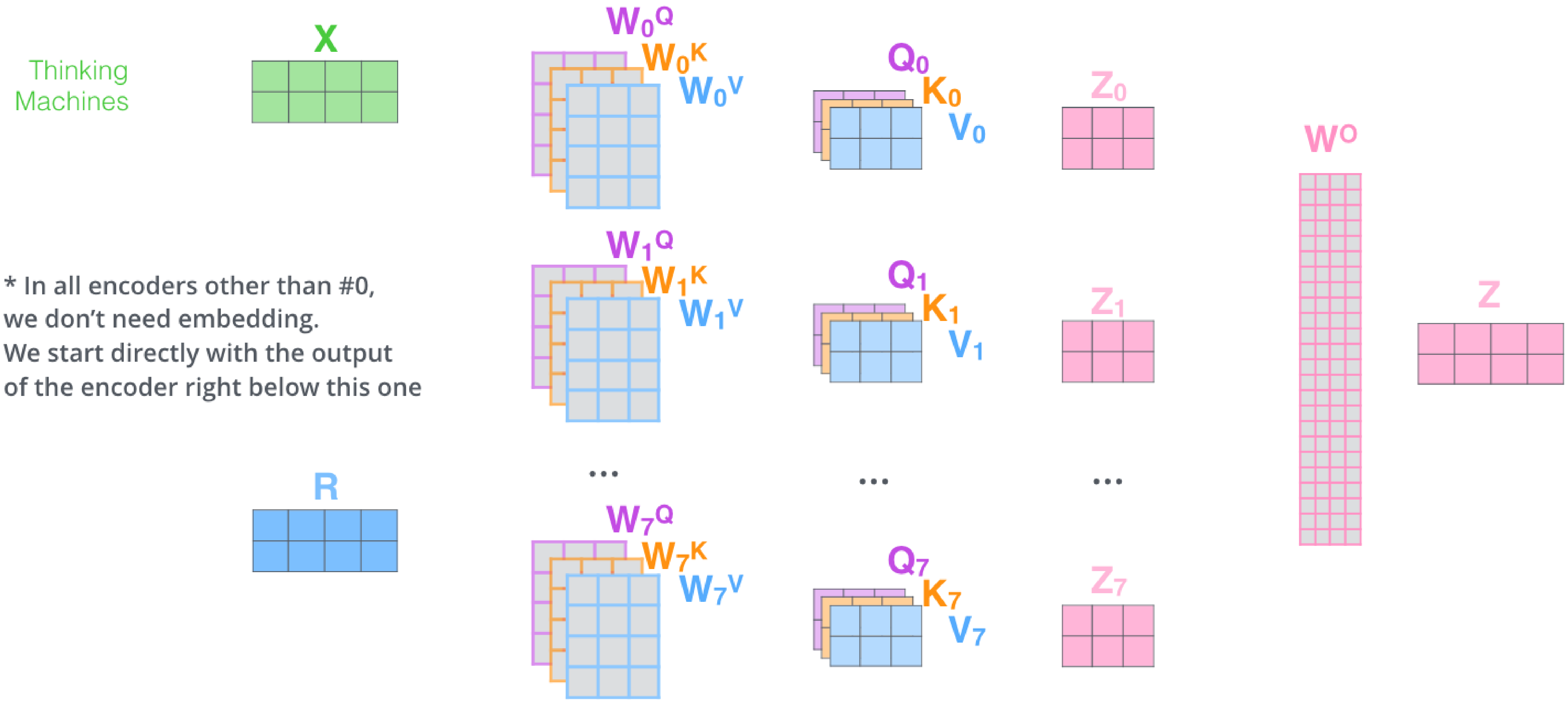
1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

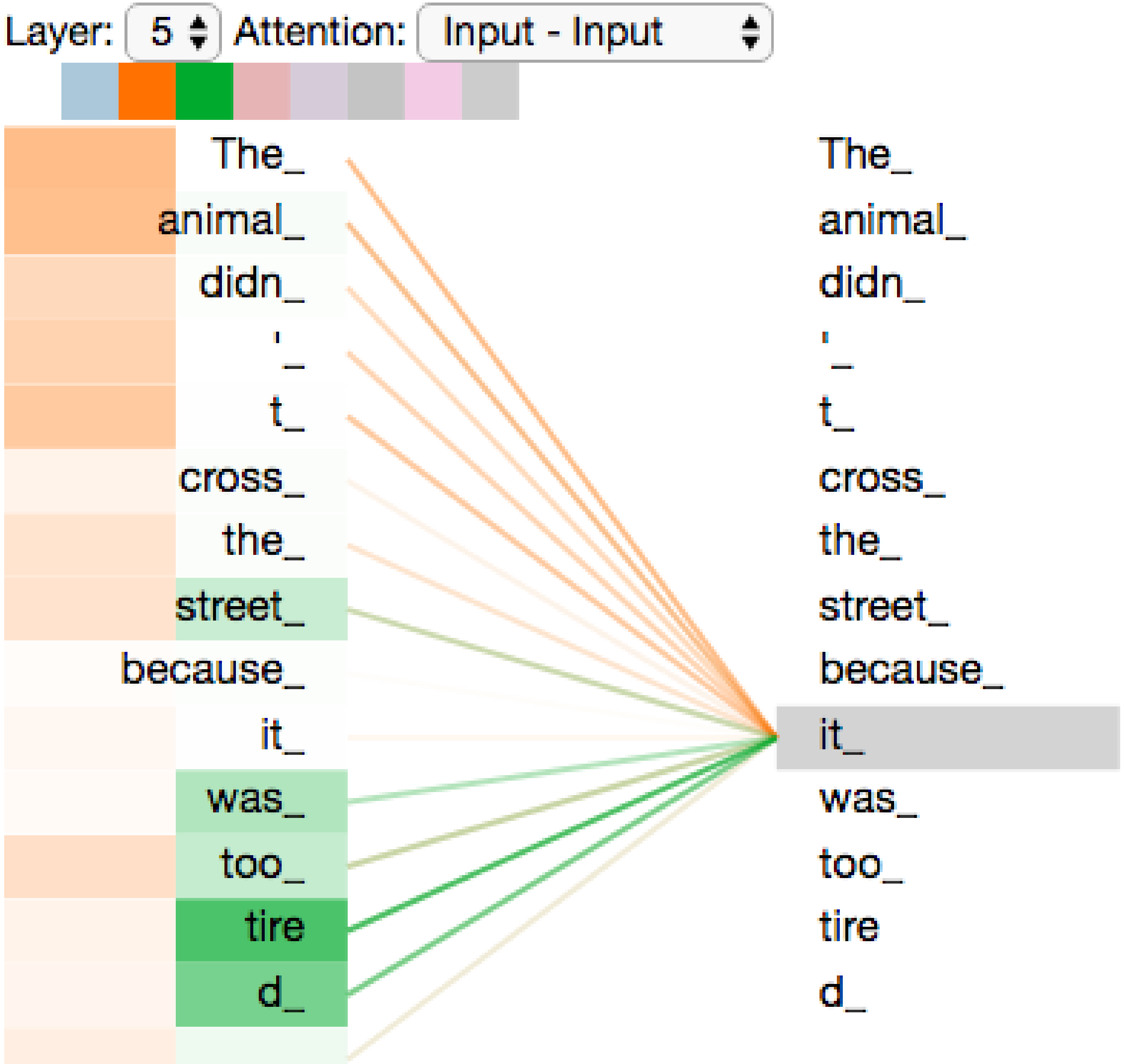
4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



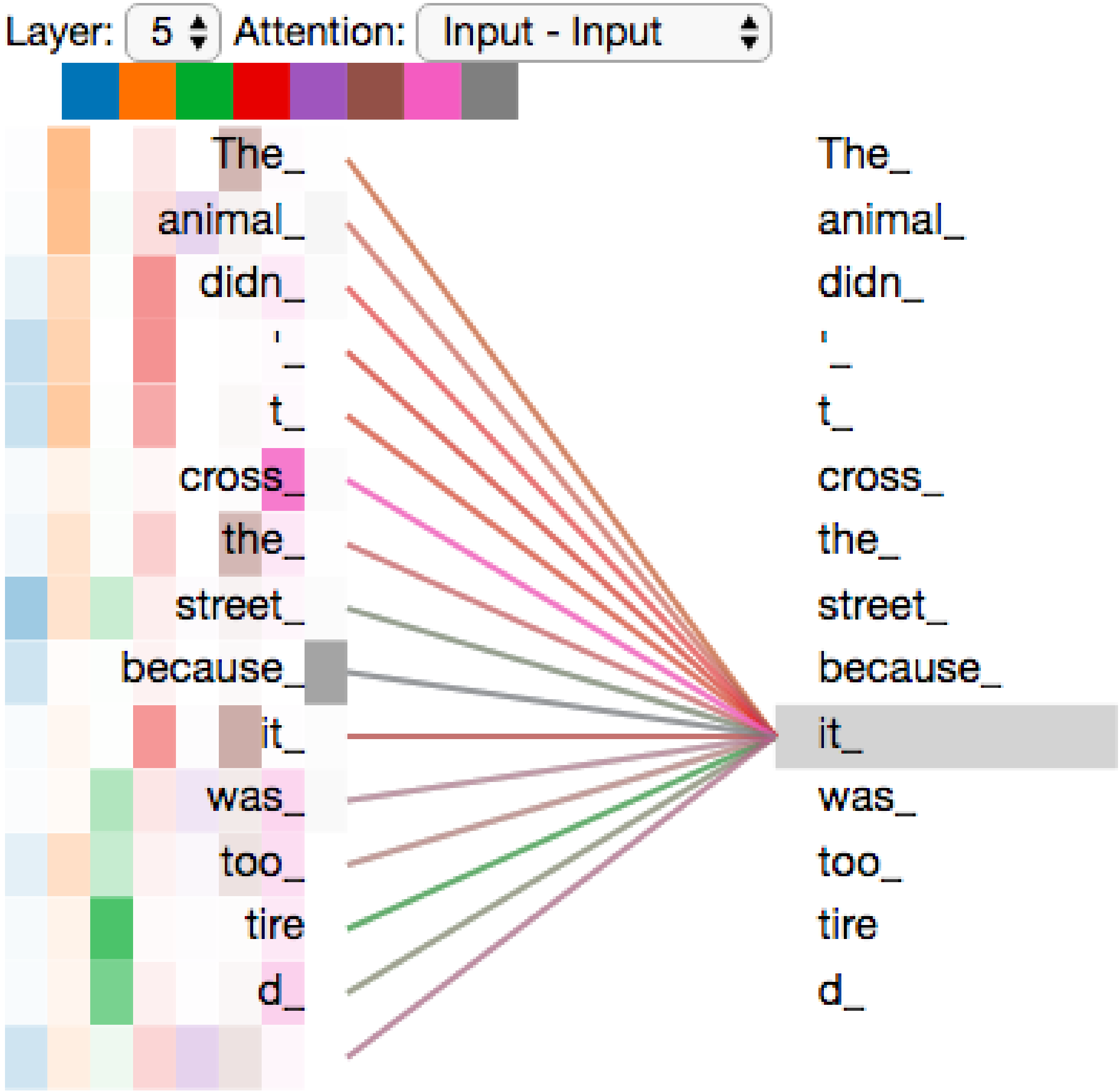
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Multi Head Attention



Layer: 5 Attention: Input - Input

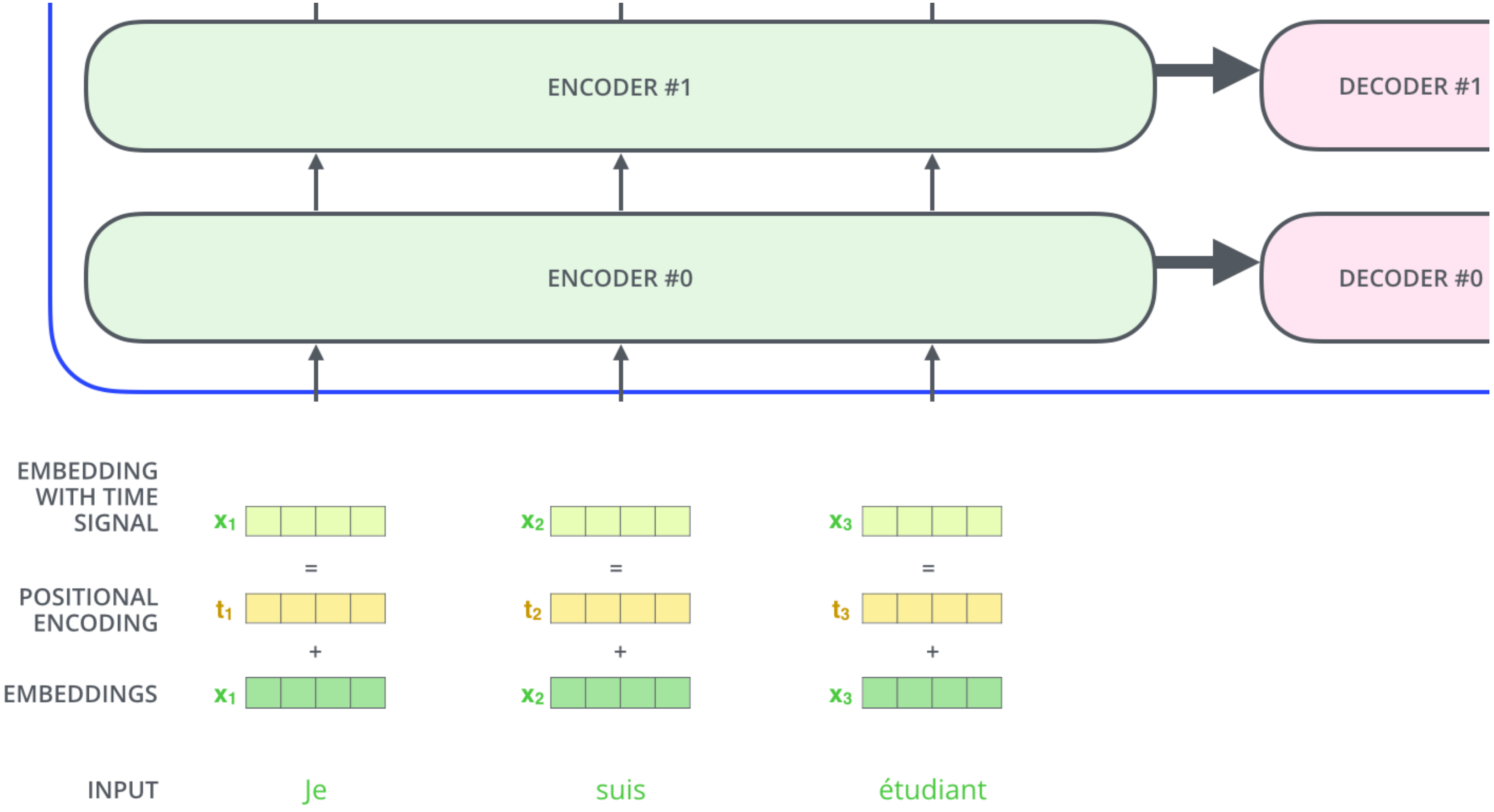
Multi Head Attention



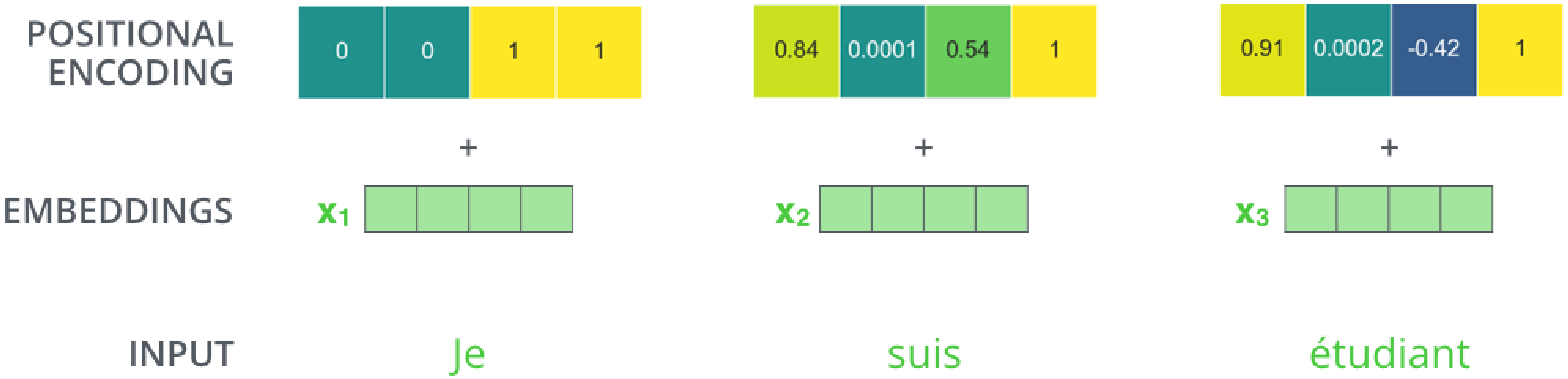
Layer: 5

Attention: Input - Input

Representing The Order of The Sequence Using Positional Encoding



Representing The Order of The Sequence Using Positional Encoding



POSITIONAL ENCODING

0 0 1 1

0.84 0.0001 0.54 1

0.91 0.0002 -0.42 1

+ \+ \+

EMBEDDINGS

x_1 x_2 x_3

INPUT

Je suis étudiant

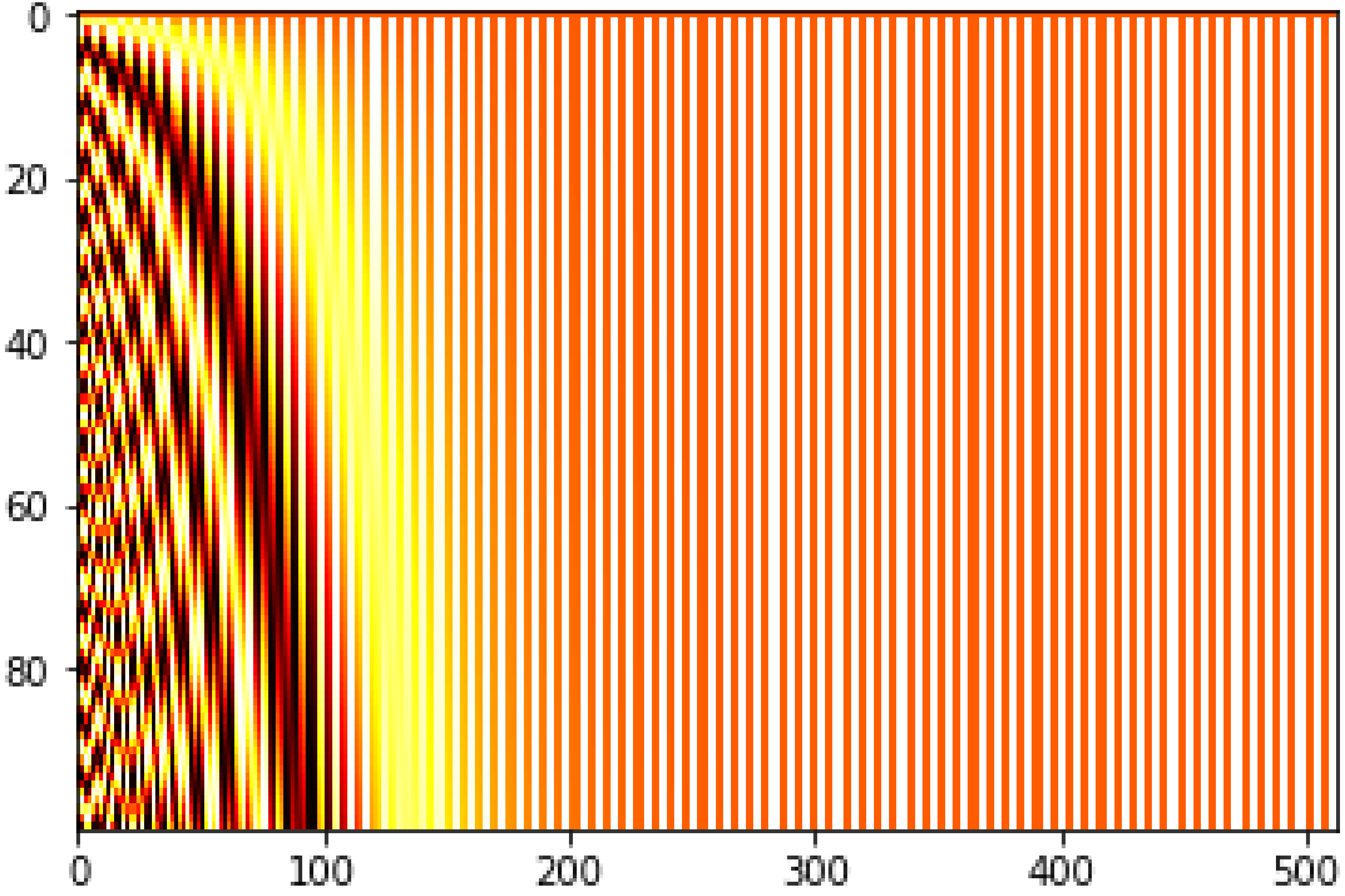
Positional Embedding

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

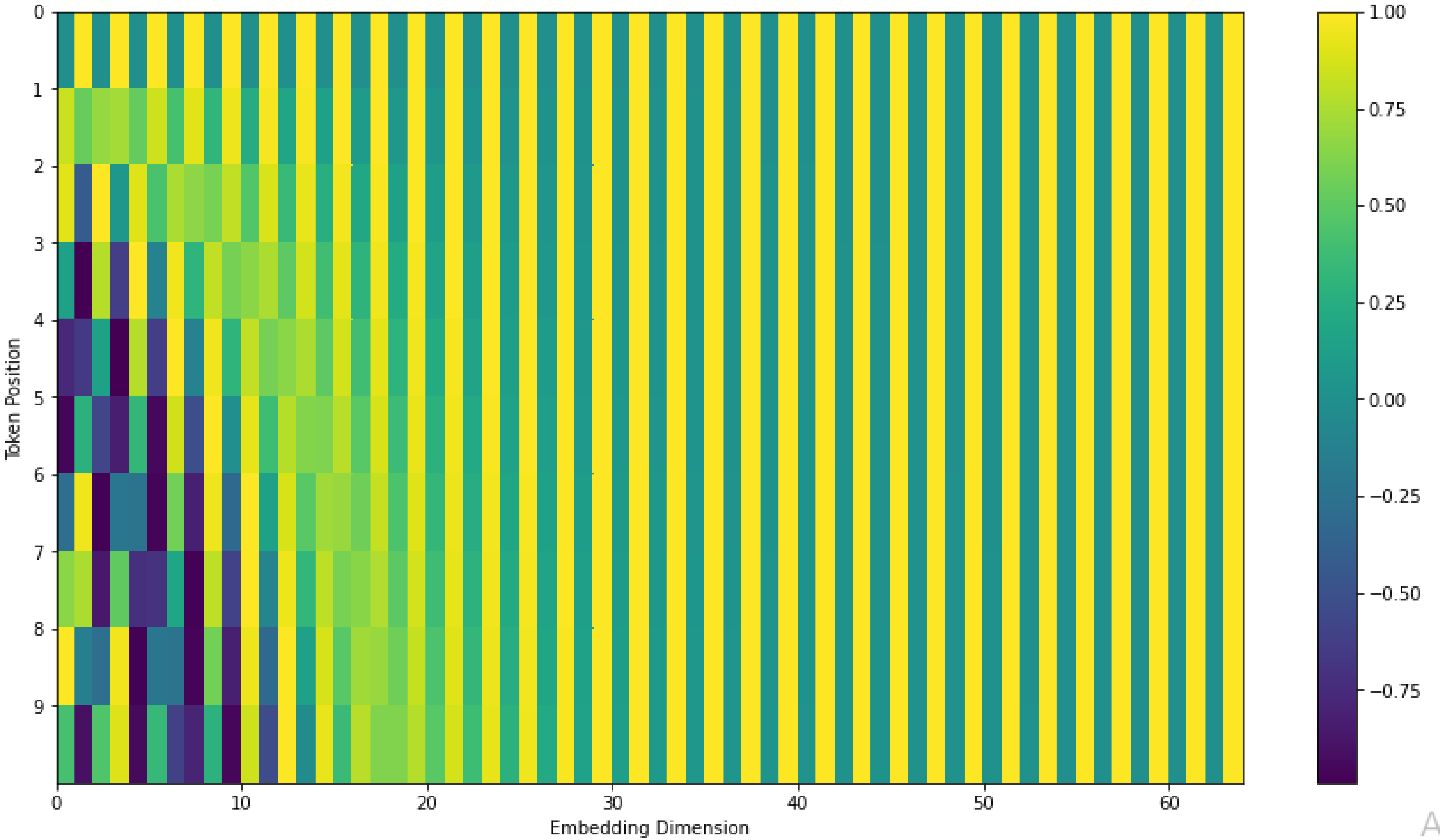
$$
\mathrm{PE}(pos, 2i) = sin\left(\frac{pos}{10000^{2i/d_{\mathrm{model}}}}\right),
$$

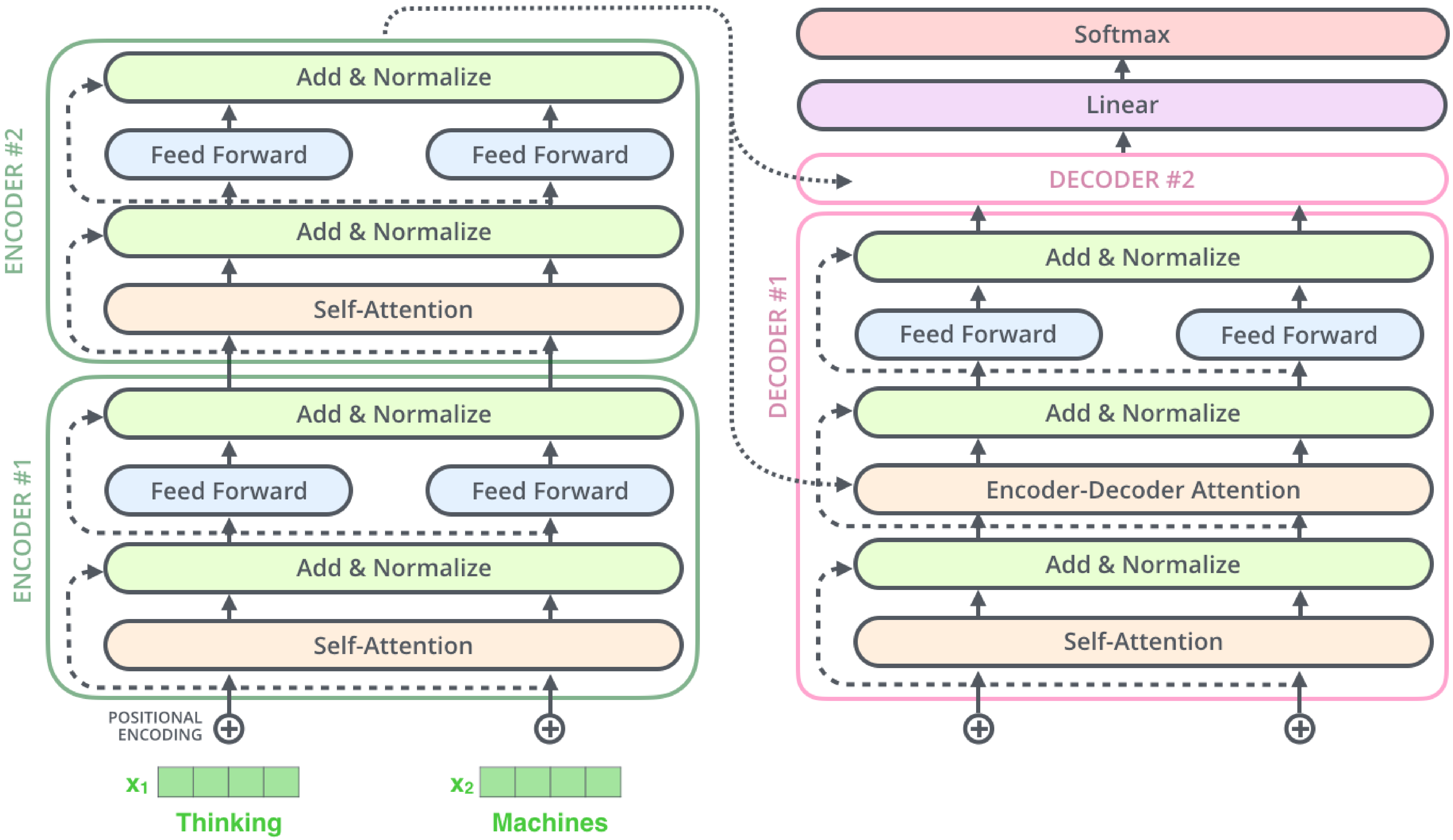
$$
\mathrm{PE}(pos, 2i+1) = cos\left(\frac{pos}{10000^{2i/d_{\mathrm{model}}}}\right).
$$

Positional Embedding

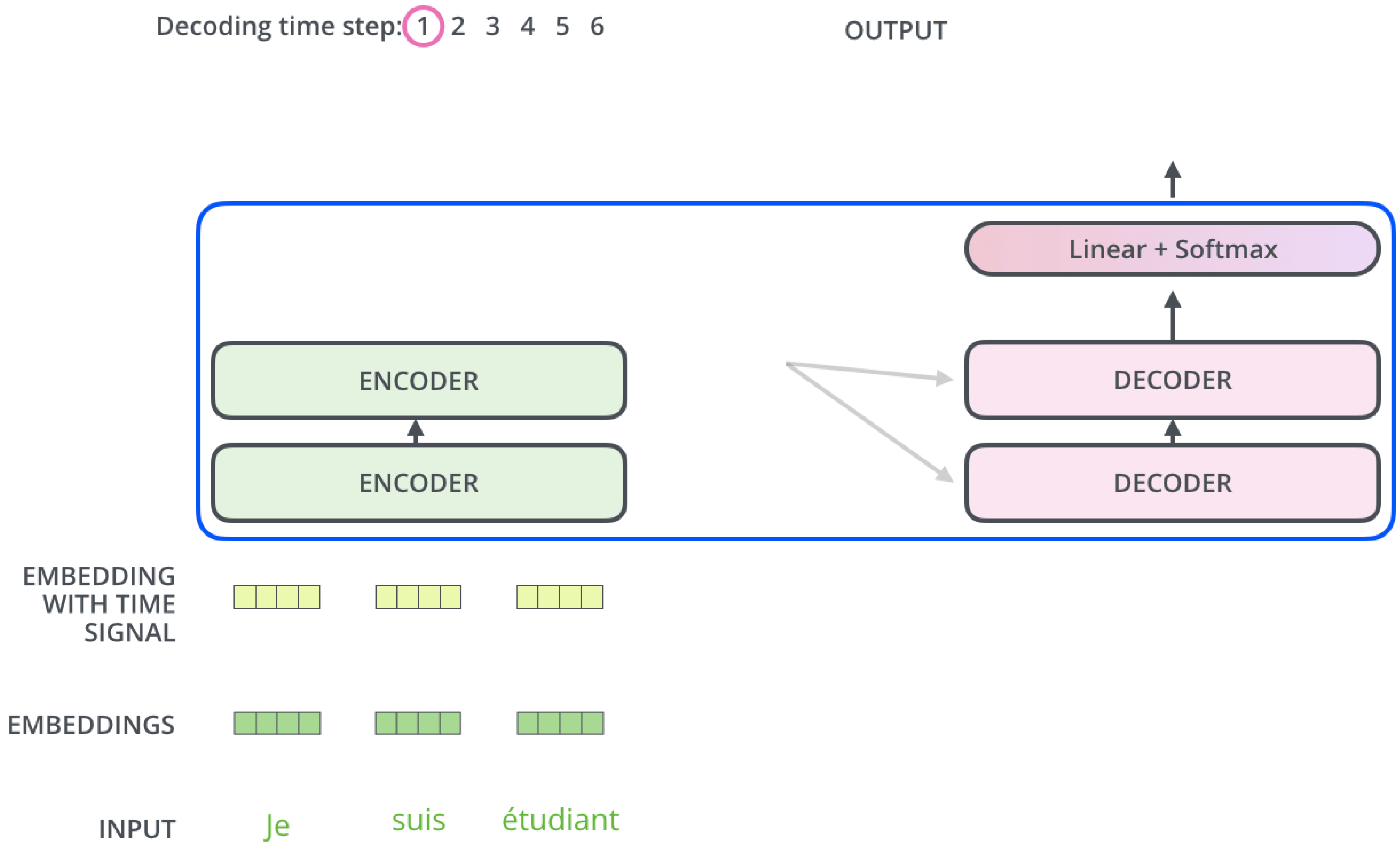


Positional Embedding




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

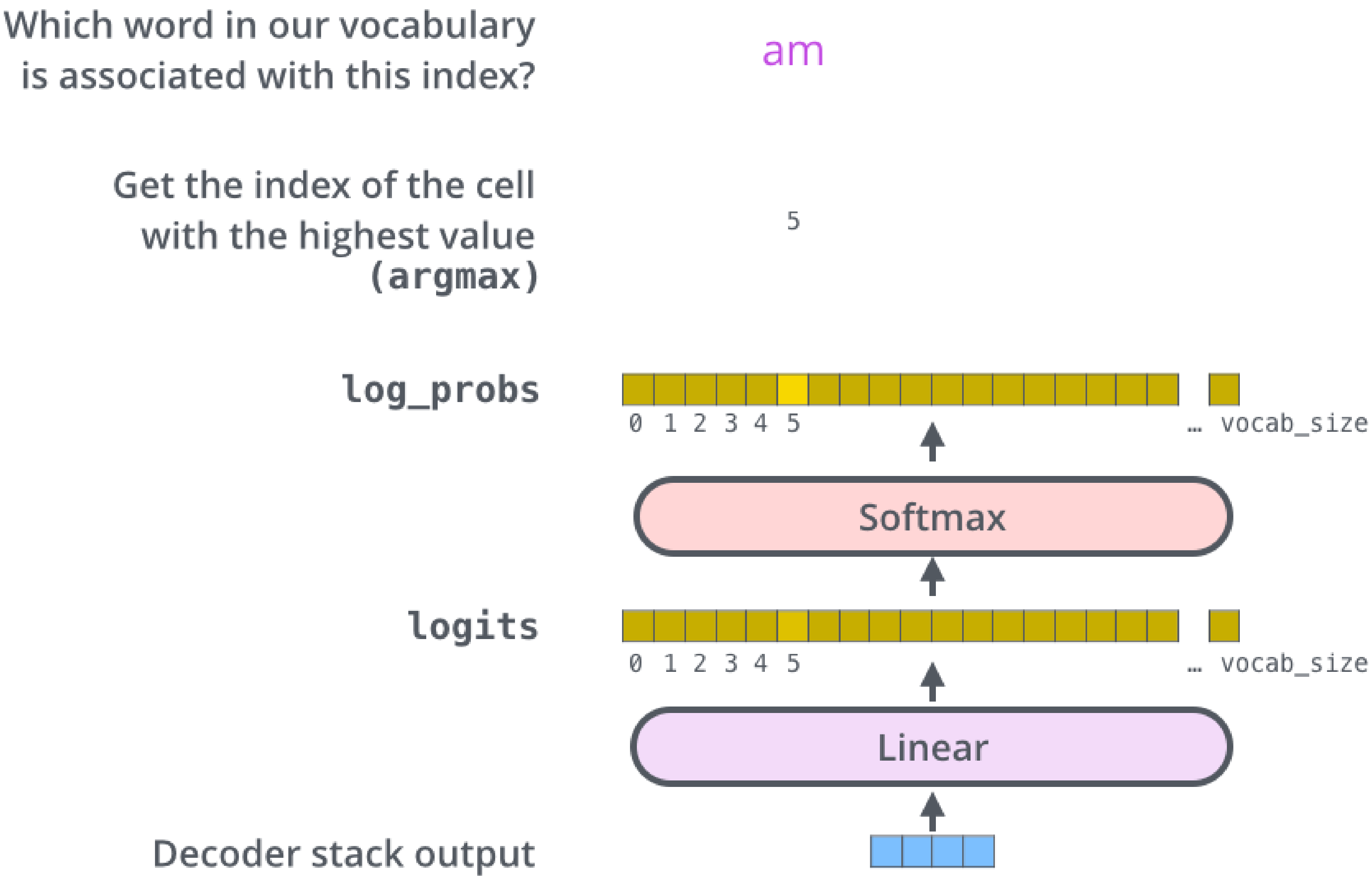
The Decoder Side



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



Which word in our vocabulary is associated with this index?

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

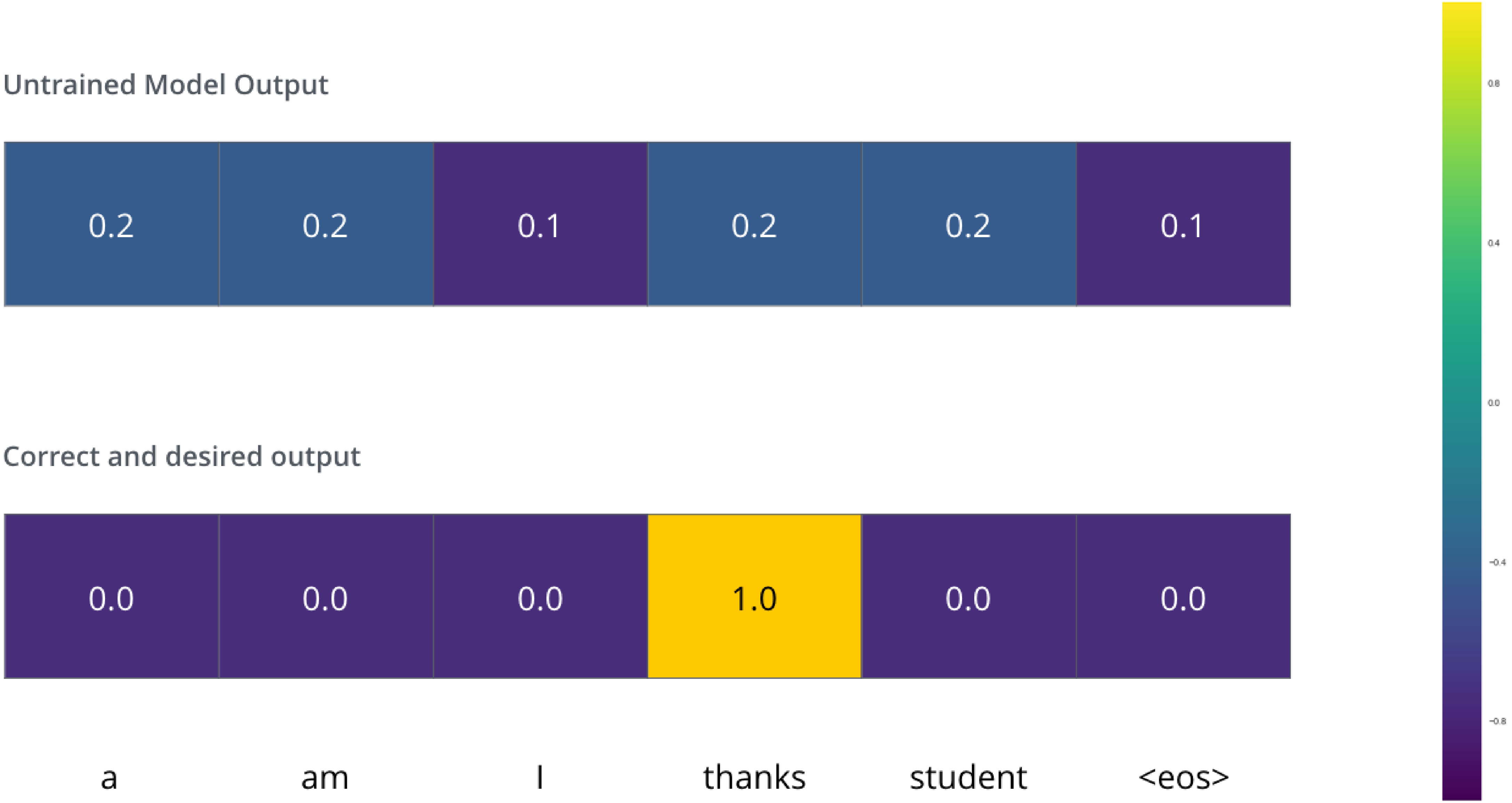
Recap Of Training

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

a	am	I	thanks	student	<eos>
0.2	0.2	0.1	0.2	0.2	0.1

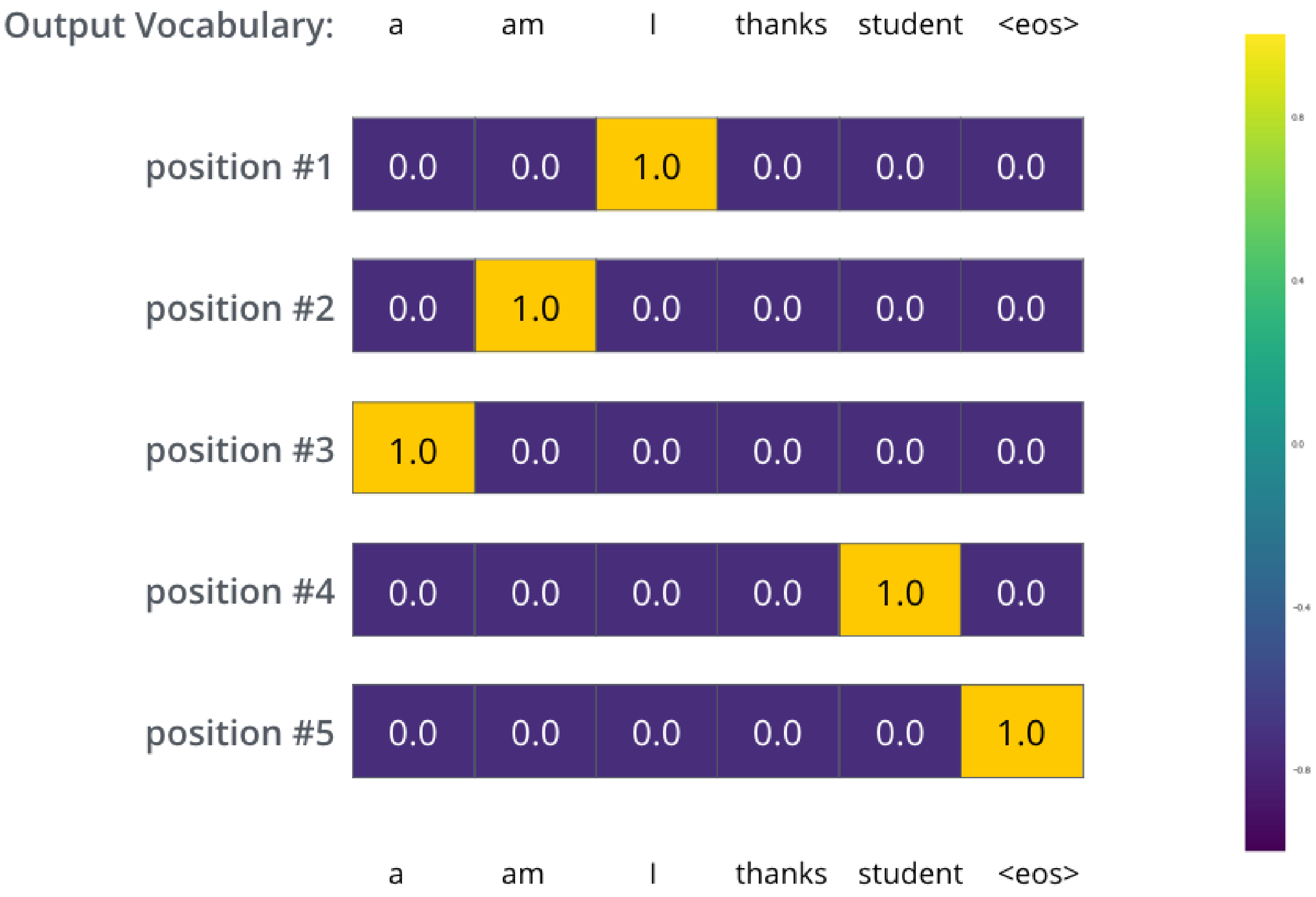
Correct and desired output

0.0	0.0	0.0	1.0	0.0	0.0

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

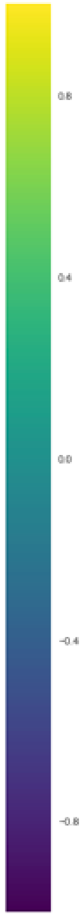


Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.0	0.0	1.0	0.0	0.0	0.0
position #2	0.0	1.0	0.0	0.0	0.0	0.0
position #3	1.0	0.0	0.0	0.0	0.0	0.0
position #4	0.0	0.0	0.0	0.0	1.0	0.0
position #5	0.0	0.0	0.0	0.0	0.0	1.0
	a	am	I	thanks	student	<eos>

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),