Attention Is All You Need

By: Subash Sah

Why Transformers?

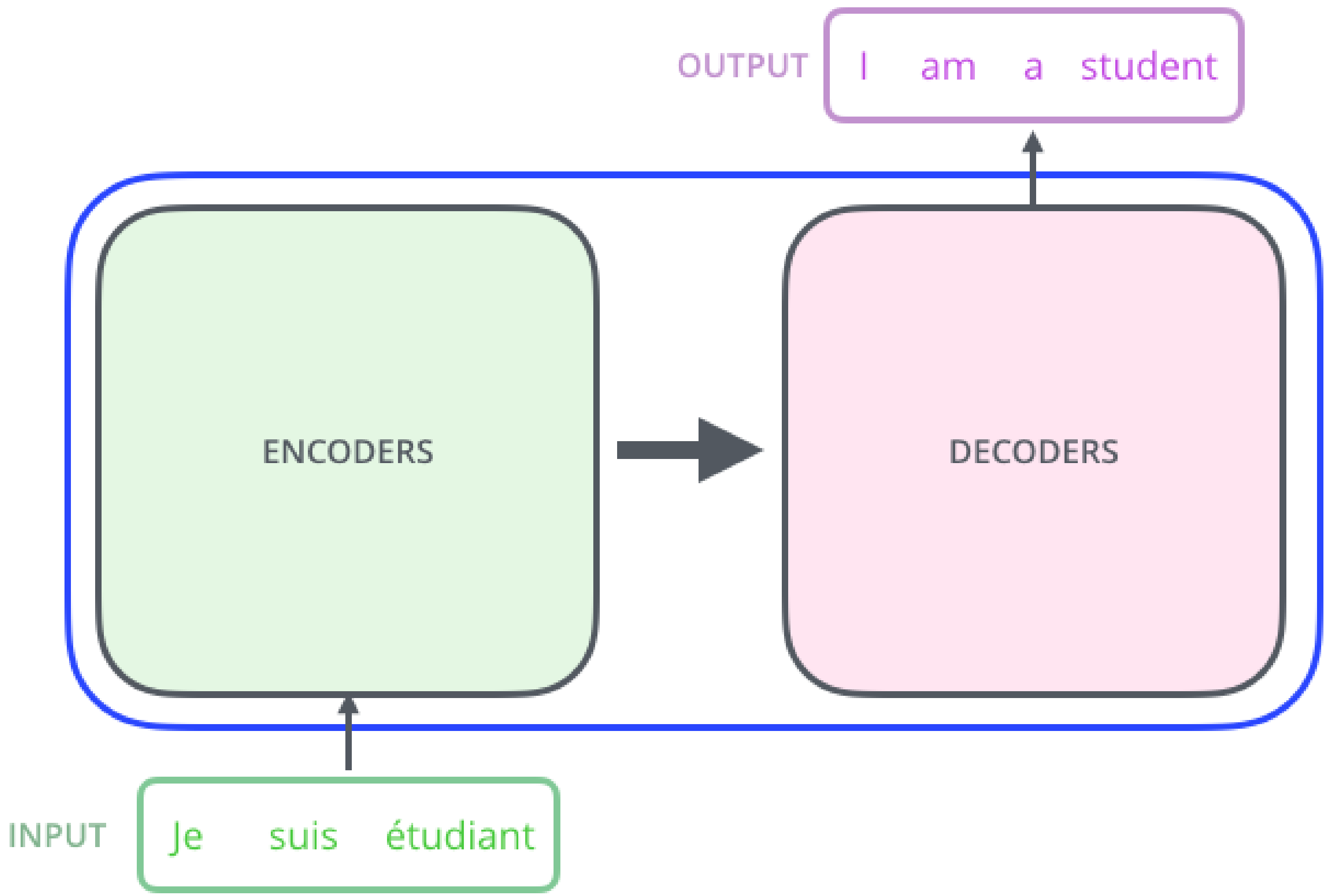
- RNNs/LSTMs are slow and sequential
- Hard to capture long-range dependencies
- Transformers allow full parallel processing
- Much better performance in translation and NLP

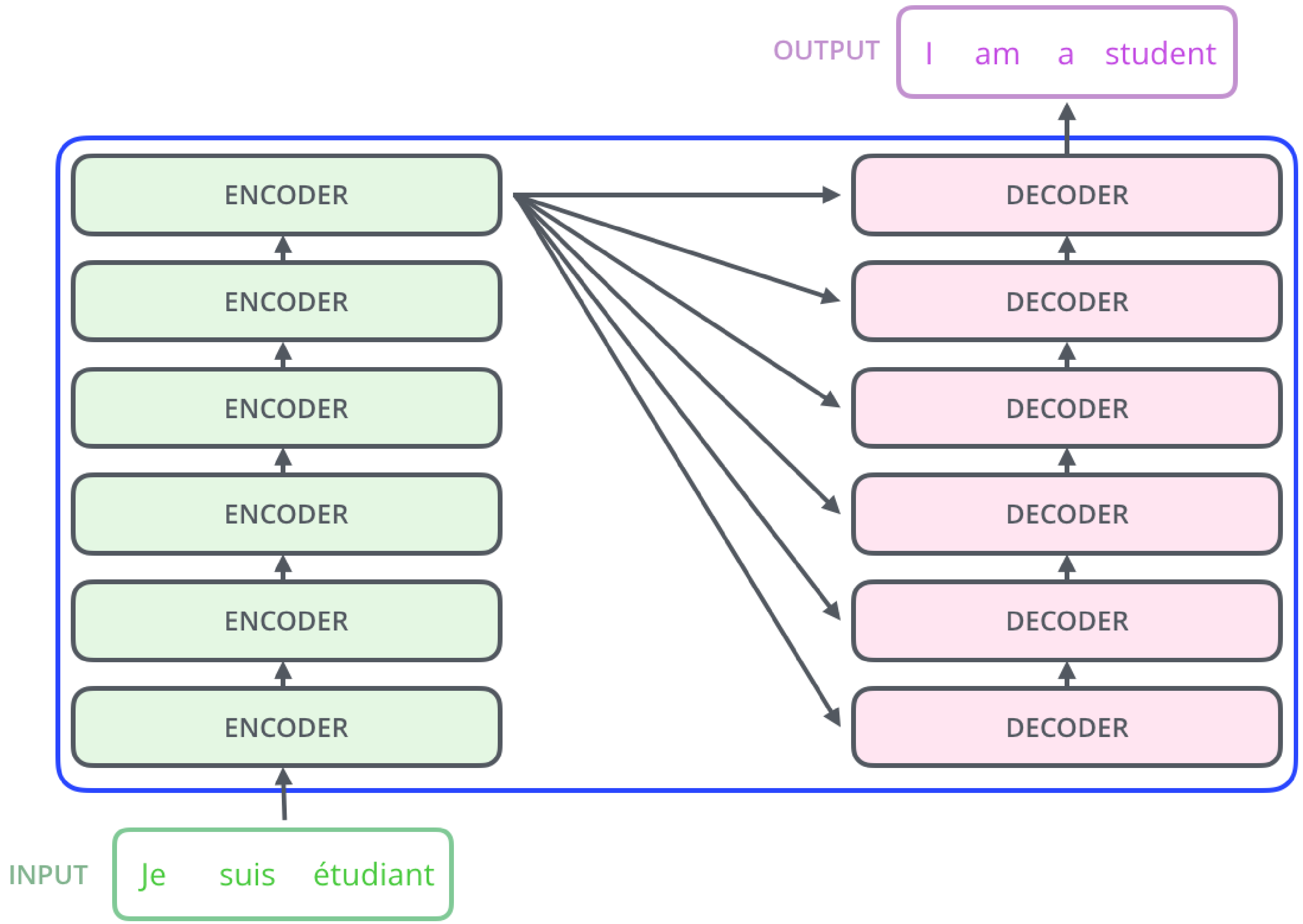
What Makes Transformers Different

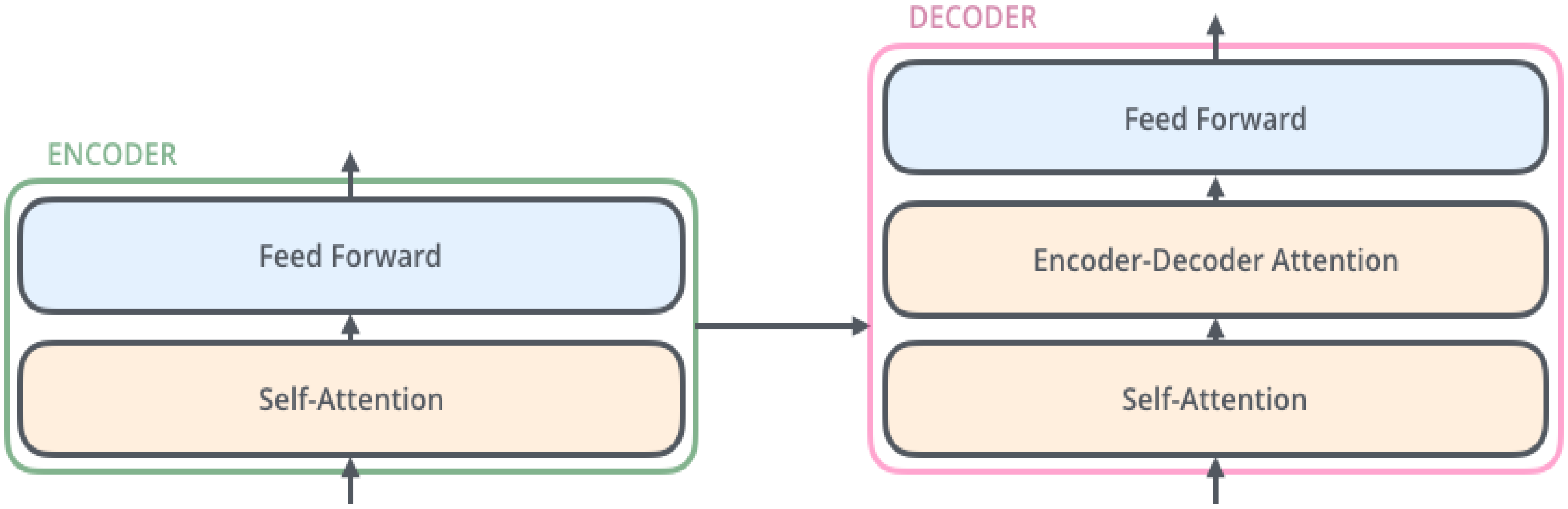
- No recurrence
- Use self-attention instead
- Handle whole sequences at once
- Scale well with data and model size
- Foundation for modern models like BERT and GPT

Architecture of Transformer

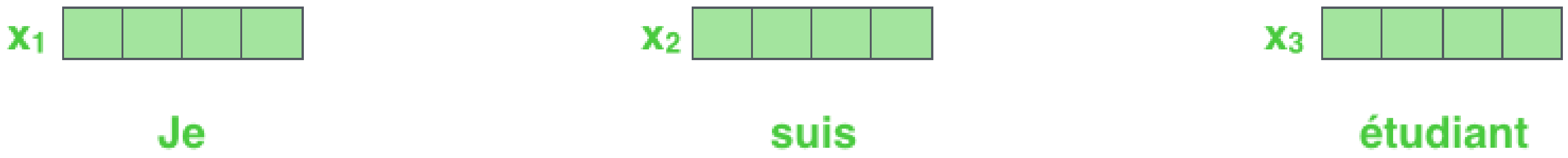
A High-Level Look


OUTPUT
I am a student
ENCODERS
DECODERS
INPUT
Je suis étudiant

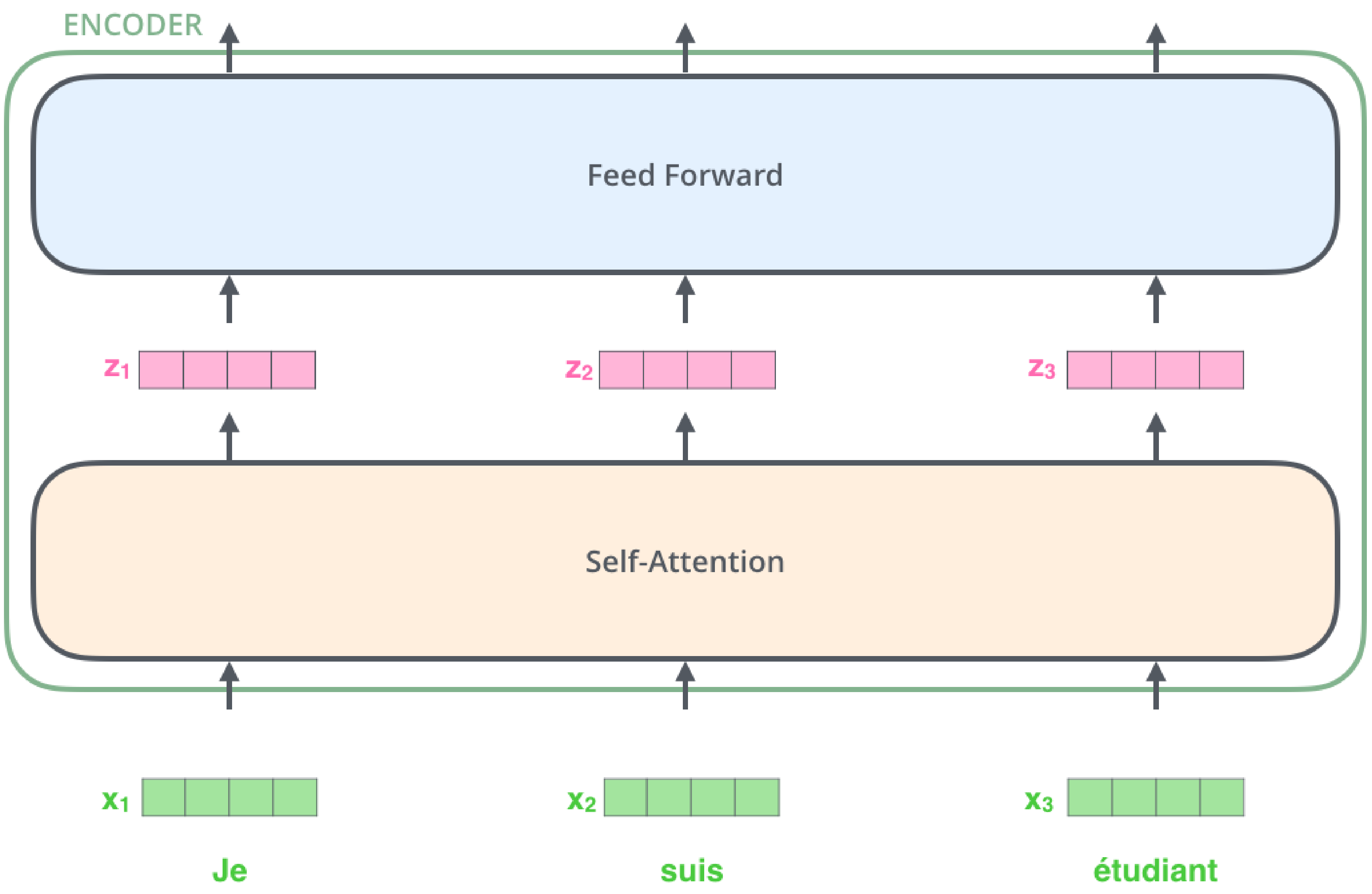

OUTPUT
I am a student
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
INPUT
Je suis étudiant

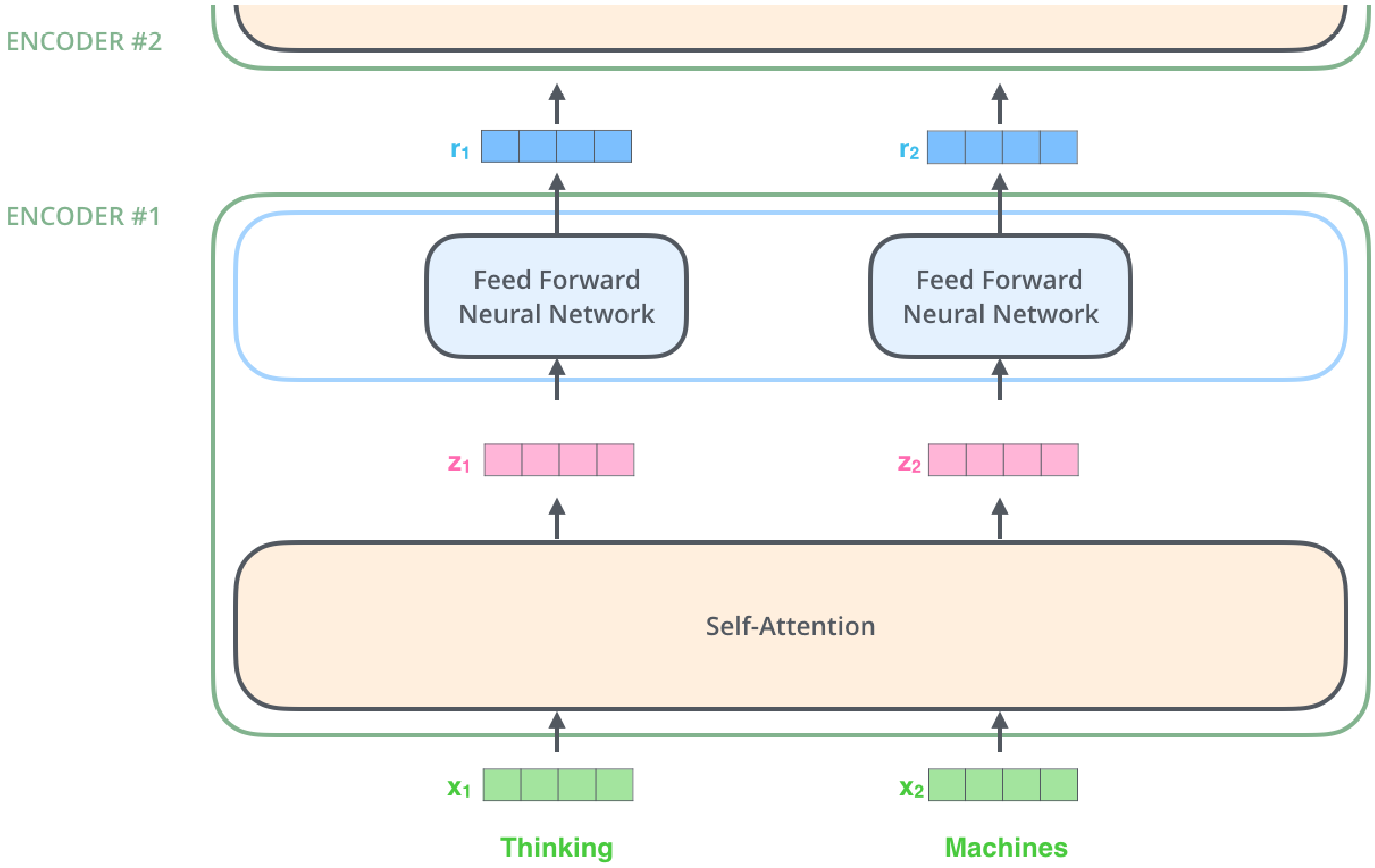

ENCODER
Feed Forward
Self-Attention
DECODER
Feed Forward
Encoder-Decoder Attention
Self-Attention

Bringing Tensors/Vectors into the Picture



Vector 512 dimensions


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

Self-Attention at High Level

- Self-attention is a mechanism where each word in a sentence looks at every other word (including itself) to decide which ones are important for understanding its meaning.

Why we need it?

- When reading the sentence:

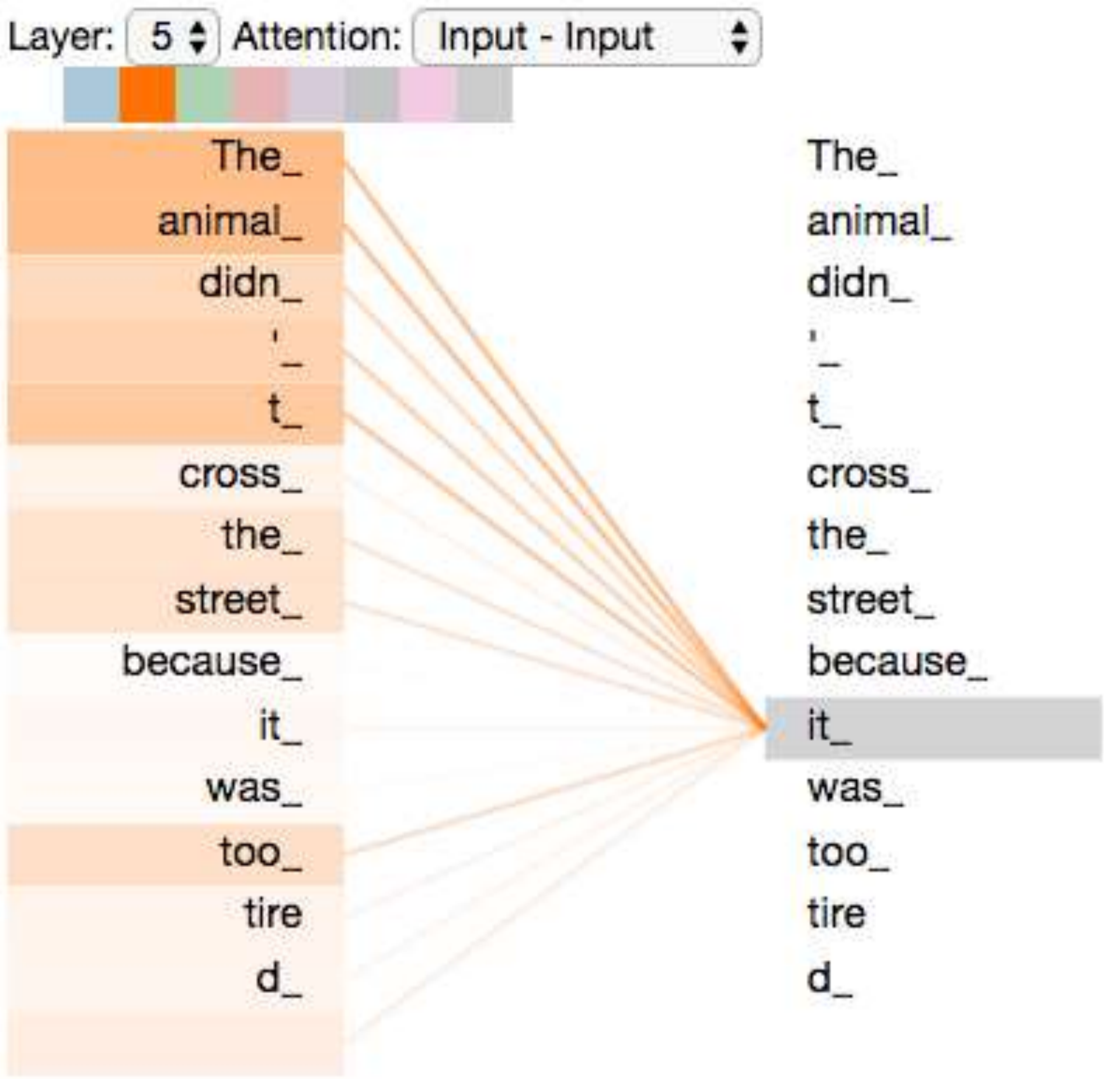
  “The animal didn’t cross the street because it was too tired.”

  To understand “it”, we must look at “animal”, not “street”.

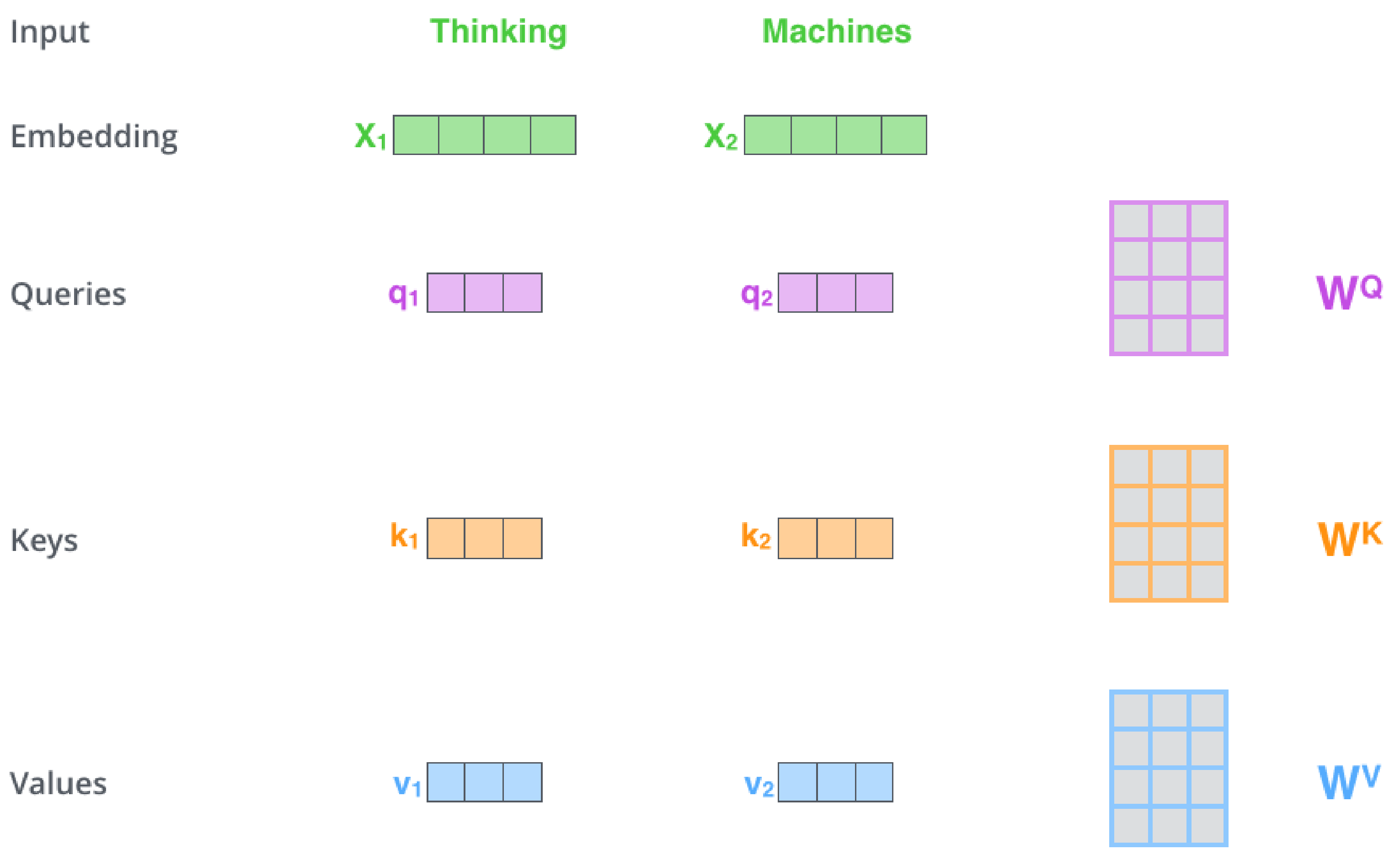
- Self-attention lets each word pull in information from other relevant words.

For each word, self-attention asks:

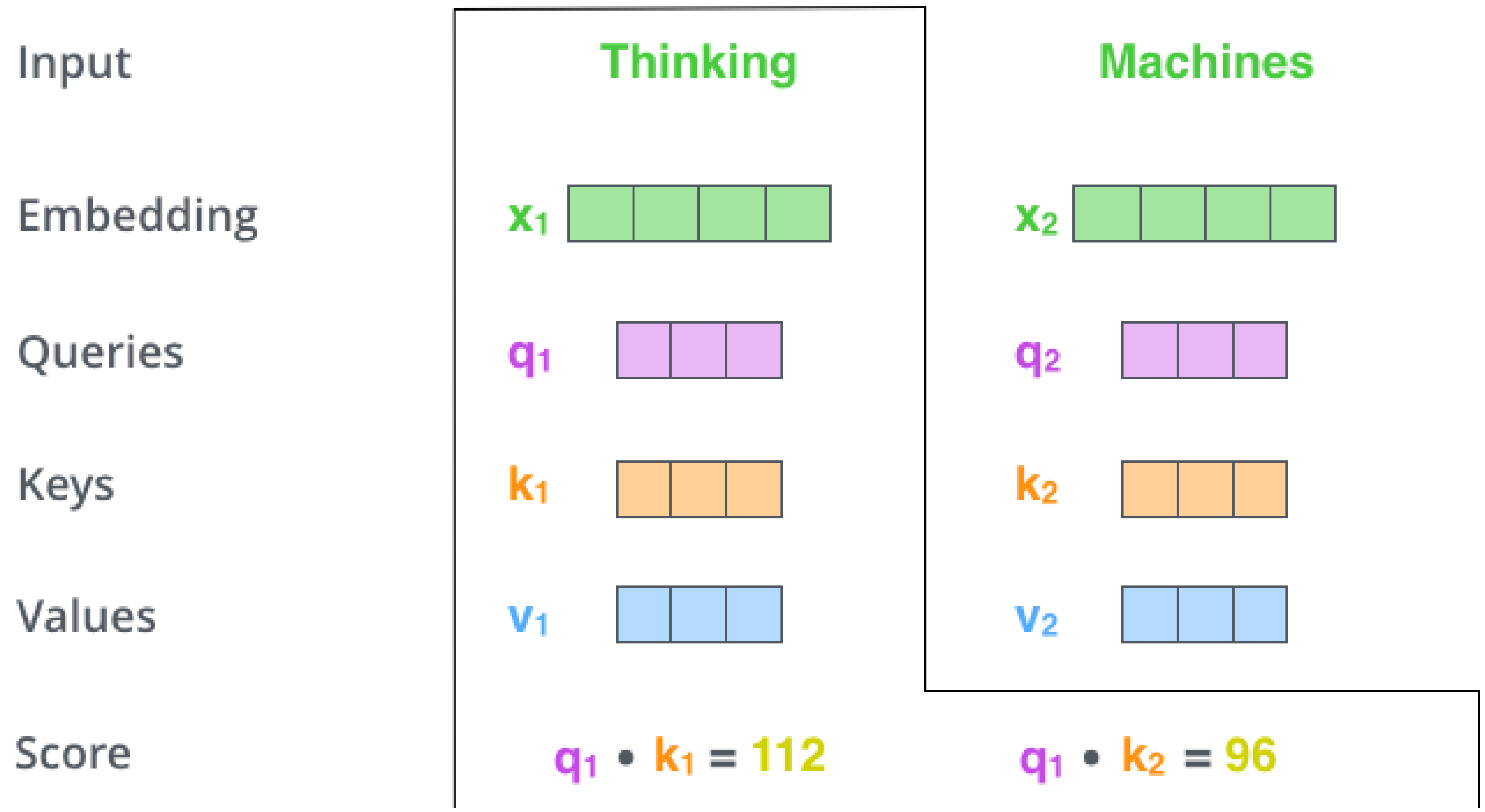
- “Which words in this sentence should I pay attention to?” and
- “How much should I pay attention to each one?”


Layer: 5 Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

Self-Attention in Detail



The first step in calculating self-attention is to create three vectors from each of the encoder’s input vectors.



The second step in calculating self-attention is to calculate a score.

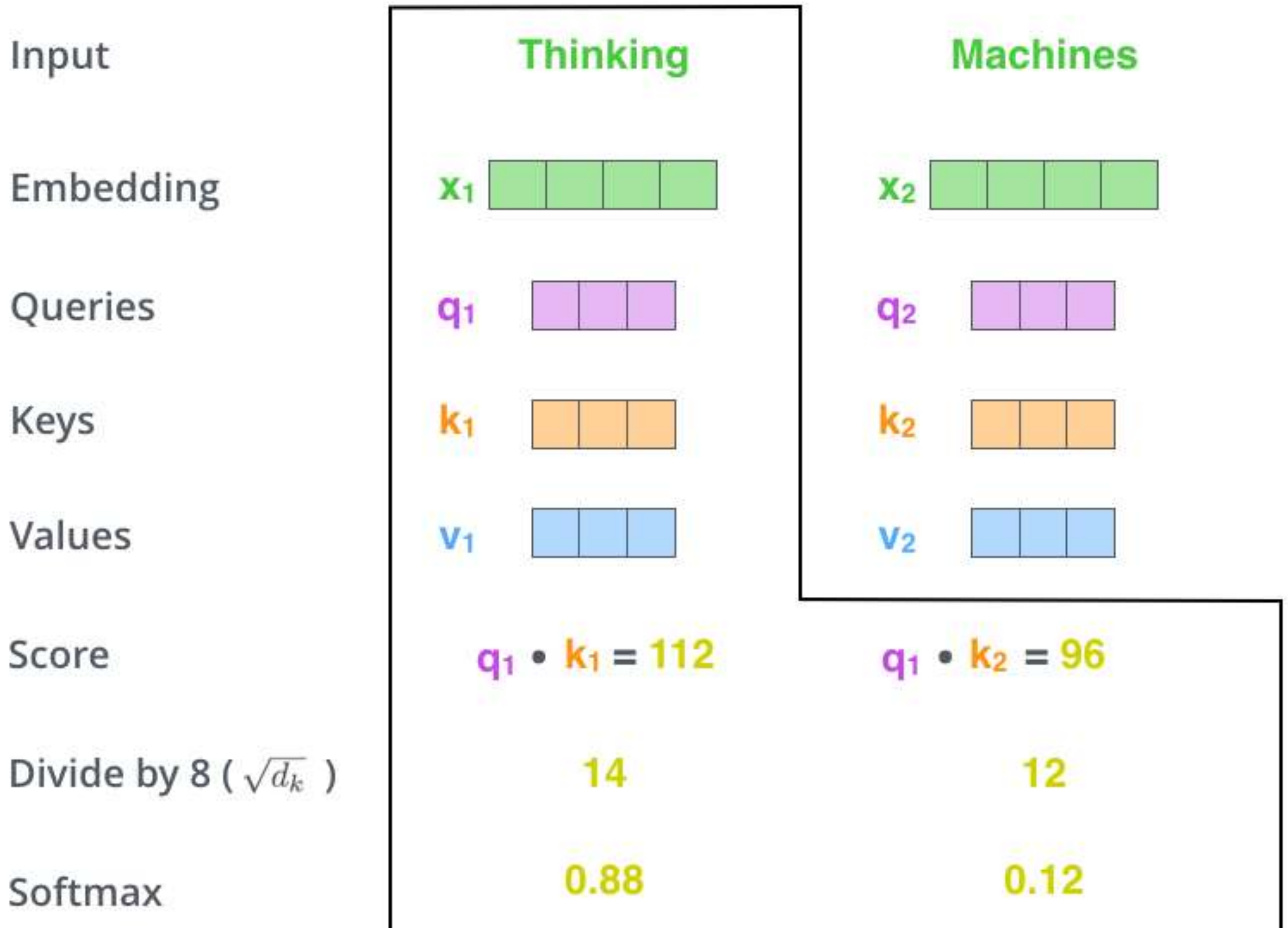
Interpretation of Score value

- High score → The second word is very relevant to the first
- Low score → The word is not important
- Near zero → The words are unrelated
- Negative (sometimes) → They should be ignored

So while computing attention for the query of “it”, we expect:

- q_it · k_animal → highest score
- q_it · k_street → lower
- q_it · k_it → moderate
- q_it · k_otherwords → lower

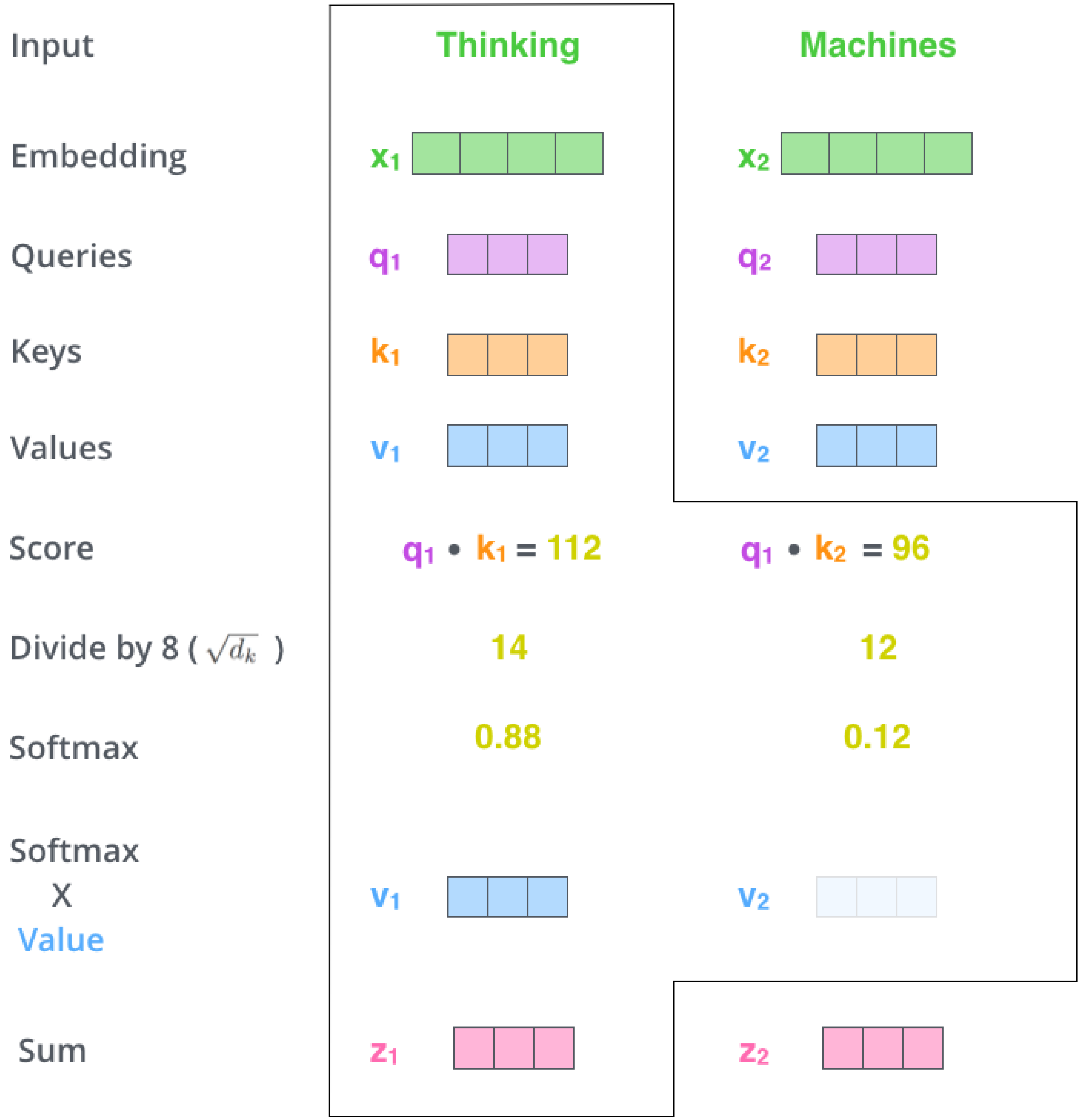
In one sentence: The attention score (q·k) is a learned measure of how much one word should pay attention to another.



Step 3 is to divide the score by 8(square root of dimension of key vectors) and step 4 is to use softmax operation.

- Why divide by 8?
  - Without dividing, the dot product becomes too large and can cause exploding gradient problems(training becomes unstable)

- Why softmax?
  - Softmax takes raw scores and turns them into a probability distribution.
  - That means:
    - all values become positive
    - they add up to 1
    - bigger scores become bigger weights
    - smaller scores become smaller weights
  - It answers the question: “How much attention should I give to each word?”



Fifth step is to multiply each value vector by softmax score. And 6th step is to sum up the weighted value vectors.

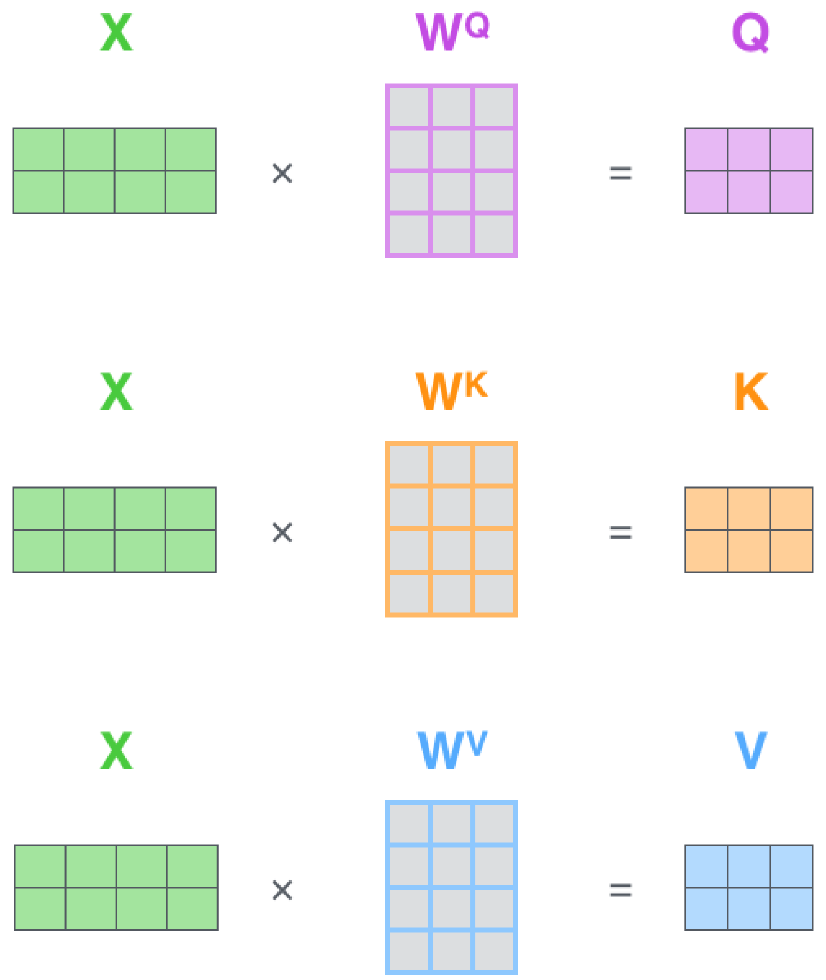
- Why multiply?

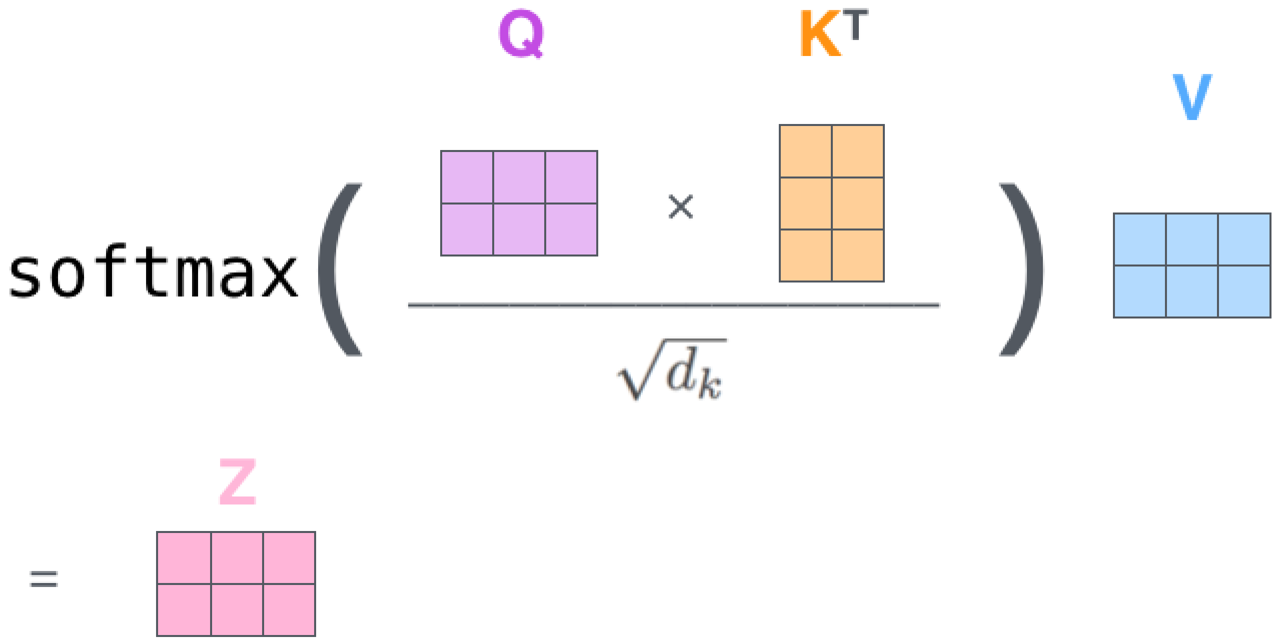
Because the Value vectors represent the actual content of each word.

  - High score → keep more of that word’s content
  - Low score → keep less of it
  - Zero score → ignore that word completely

- And then finally add all the value vectors to give Z which is the output of self attention.
- The Z vector contains:
  - The word itself
  - Plus relevant information from other words
  - Weighted by how important they are

Matrix Calculation of Self-Attention

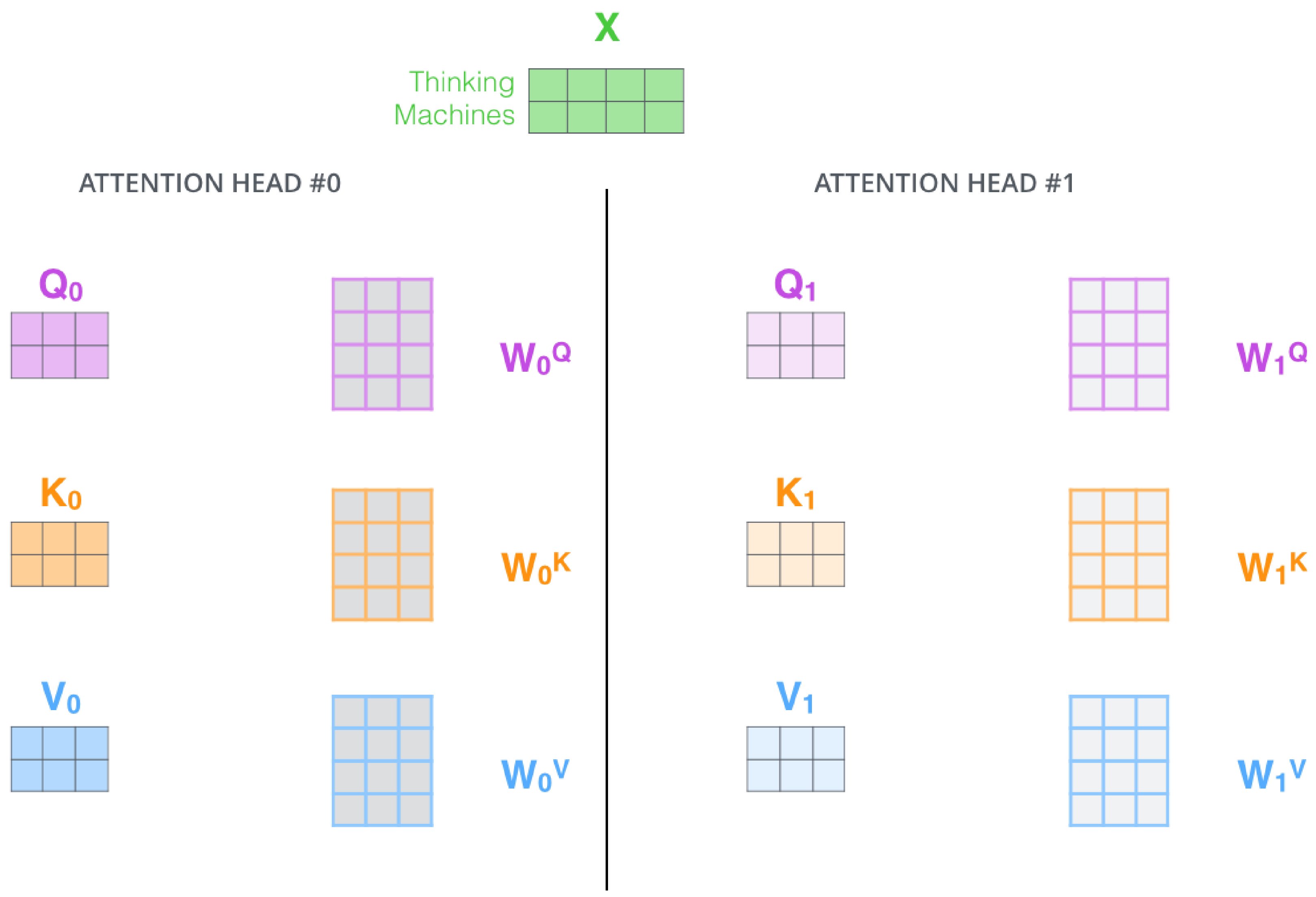


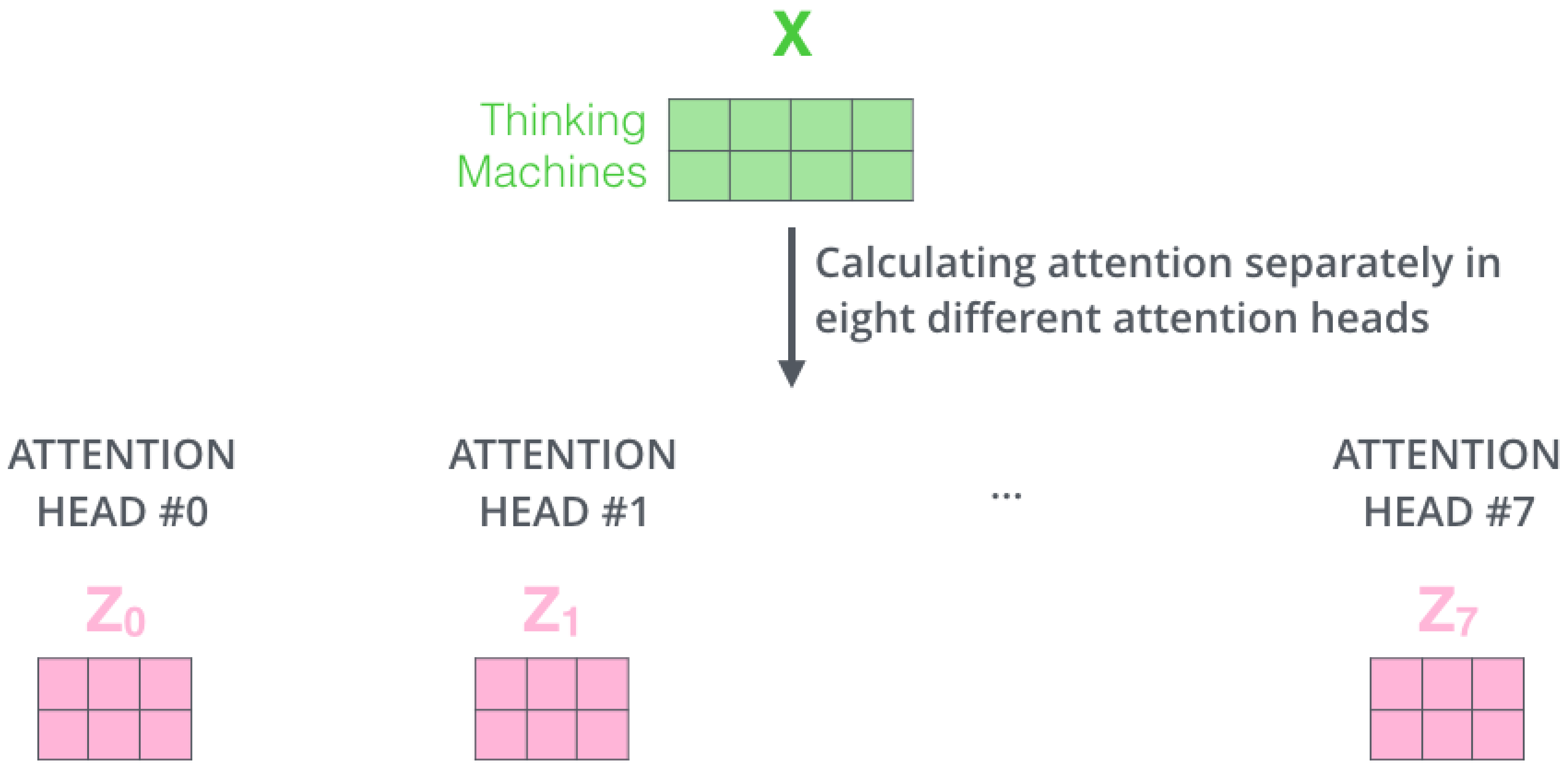


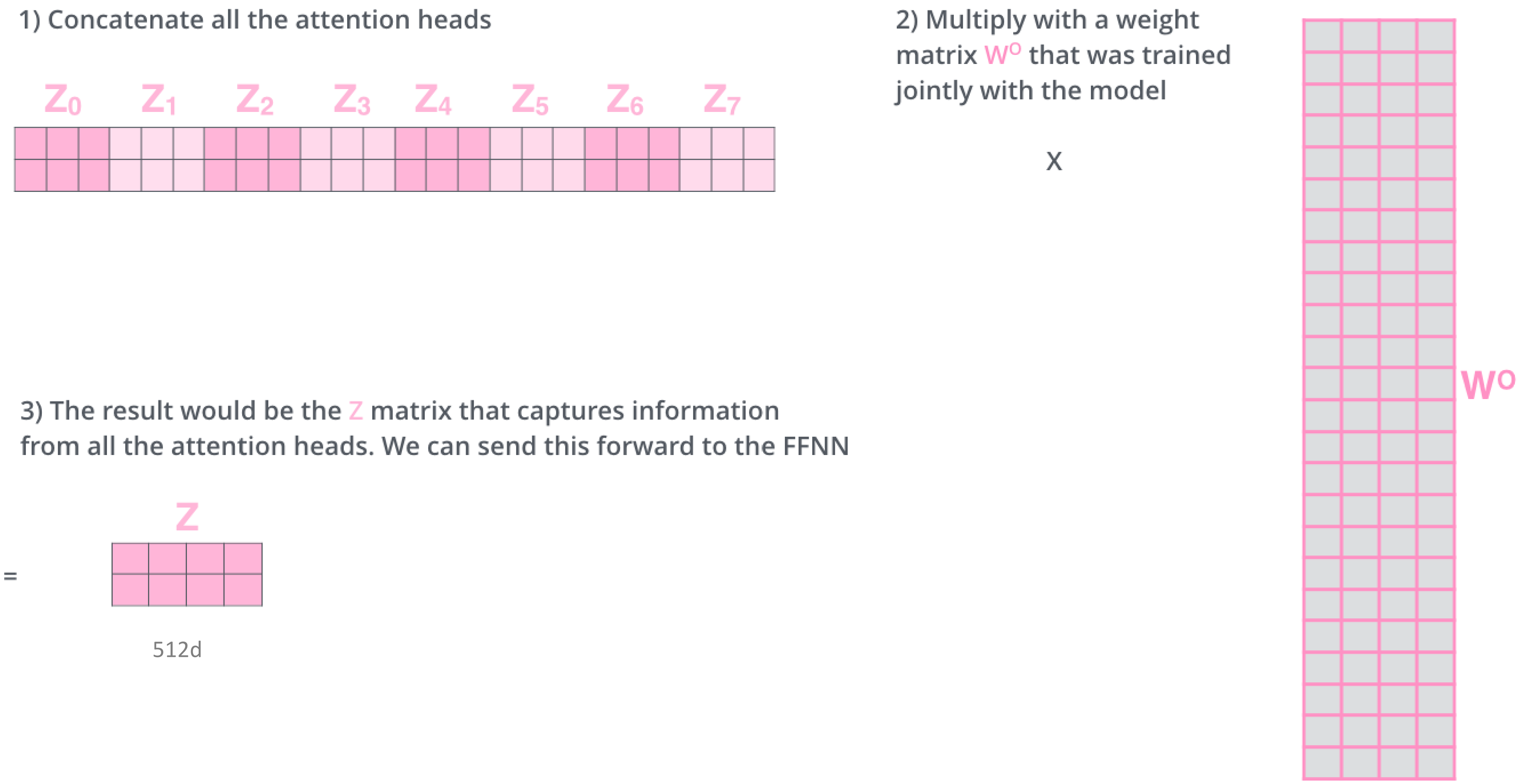
The self-attention calculation in matrix form.

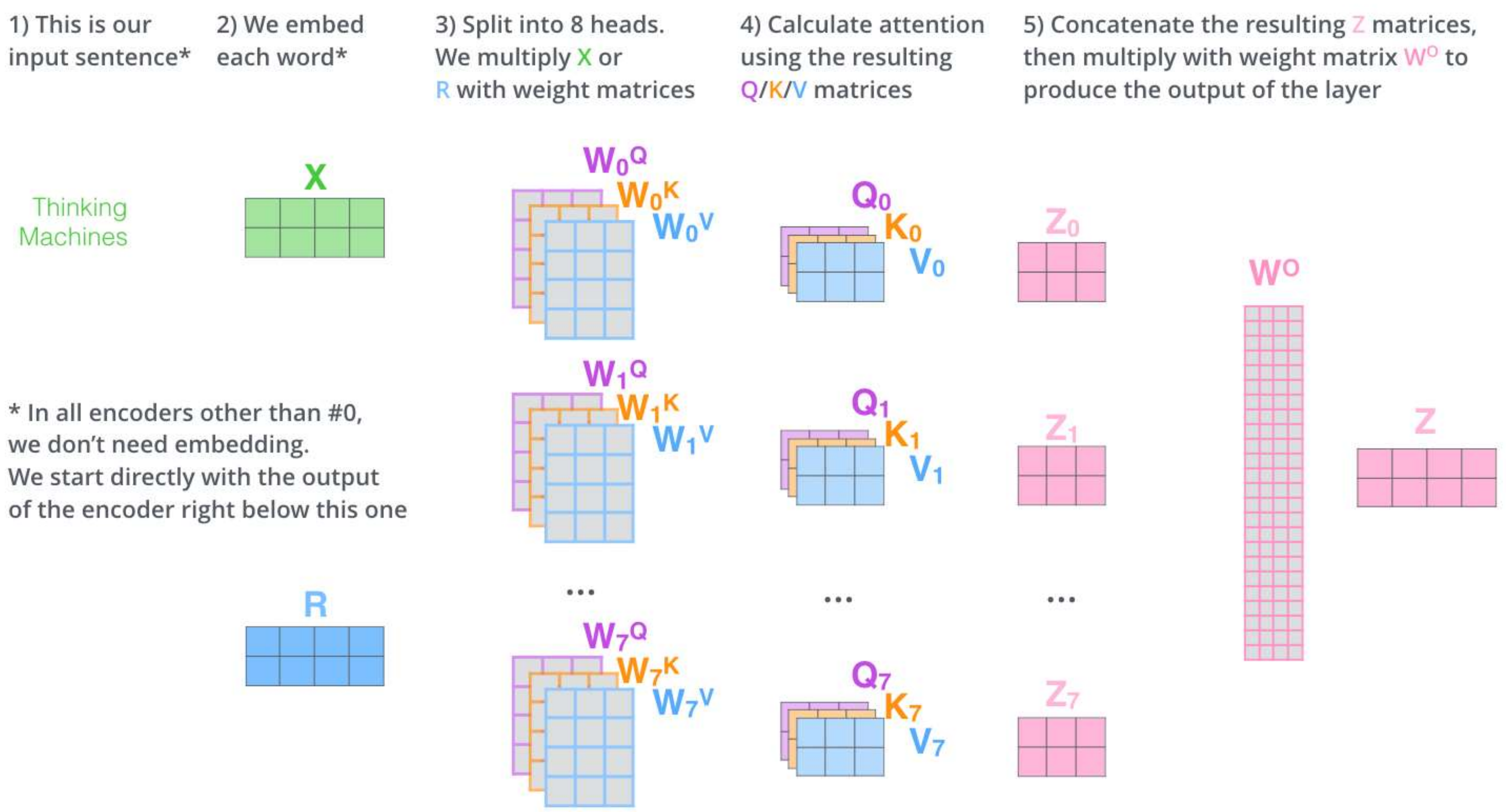
Multi-Head Attention

- Why multi-head attention?
  - One head cannot capture all the patterns at once.
  - Multiple heads allow the transformer to understand language from many angles at the same time.
- In the example sentence:
  - "The animal didn't cross the street because it was tired.", different kinds of relationships matter.
  - Head 1: "it" ←→ "animal" (pronoun resolution)
  - Head 2: grammatical structure
  - Head 3: adjective → noun and so on.

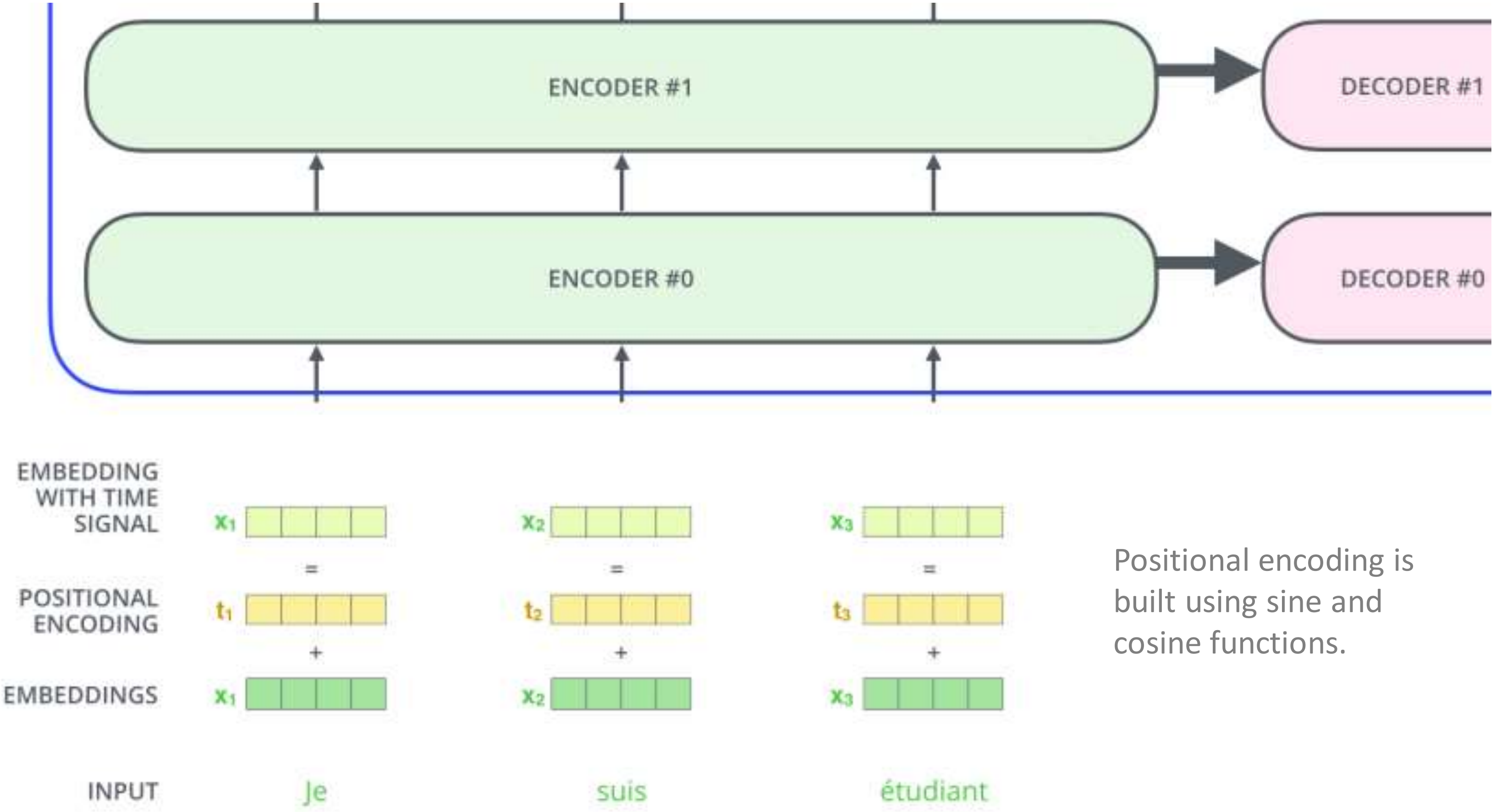

X
Thinking
Machines
ATTENTION HEAD #0
ATTENTION HEAD #1
Q_0
W_0^Q
K_0
W_0^K
V_0
W_0^V
Q_1
W_1^Q
K_1
W_1^K
V_1
W_1^V


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


1) Concatenate all the attention heads
Z0 Z1 Z2 Z3 Z4 Z5 Z6 Z7
2) Multiply with a weight matrix W^O that was trained jointly with the model
X
W^O
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
=
Z
512d


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
W_0^Q W_0^K W_0^V
W_1^Q W_1^K W_1^V
...
W_7^Q W_7^K W_7^V
Q_0 K_0 V_0
Q_1 K_1 V_1
...
Q_7 K_7 V_7
Z_0
Z_1
...
Z_7
W^O
Z

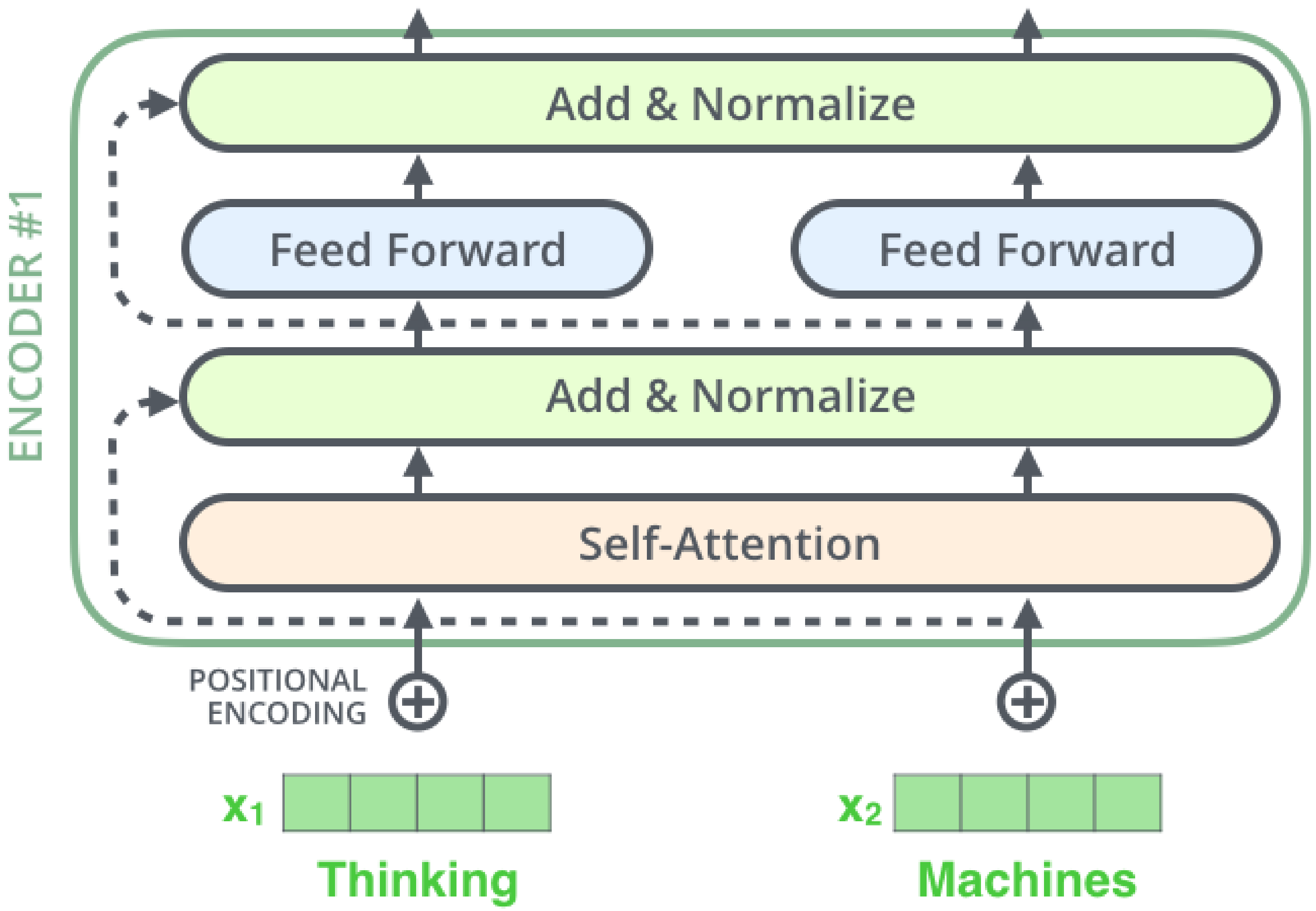
Representing The Order of The Sequence Using Positional Encoding

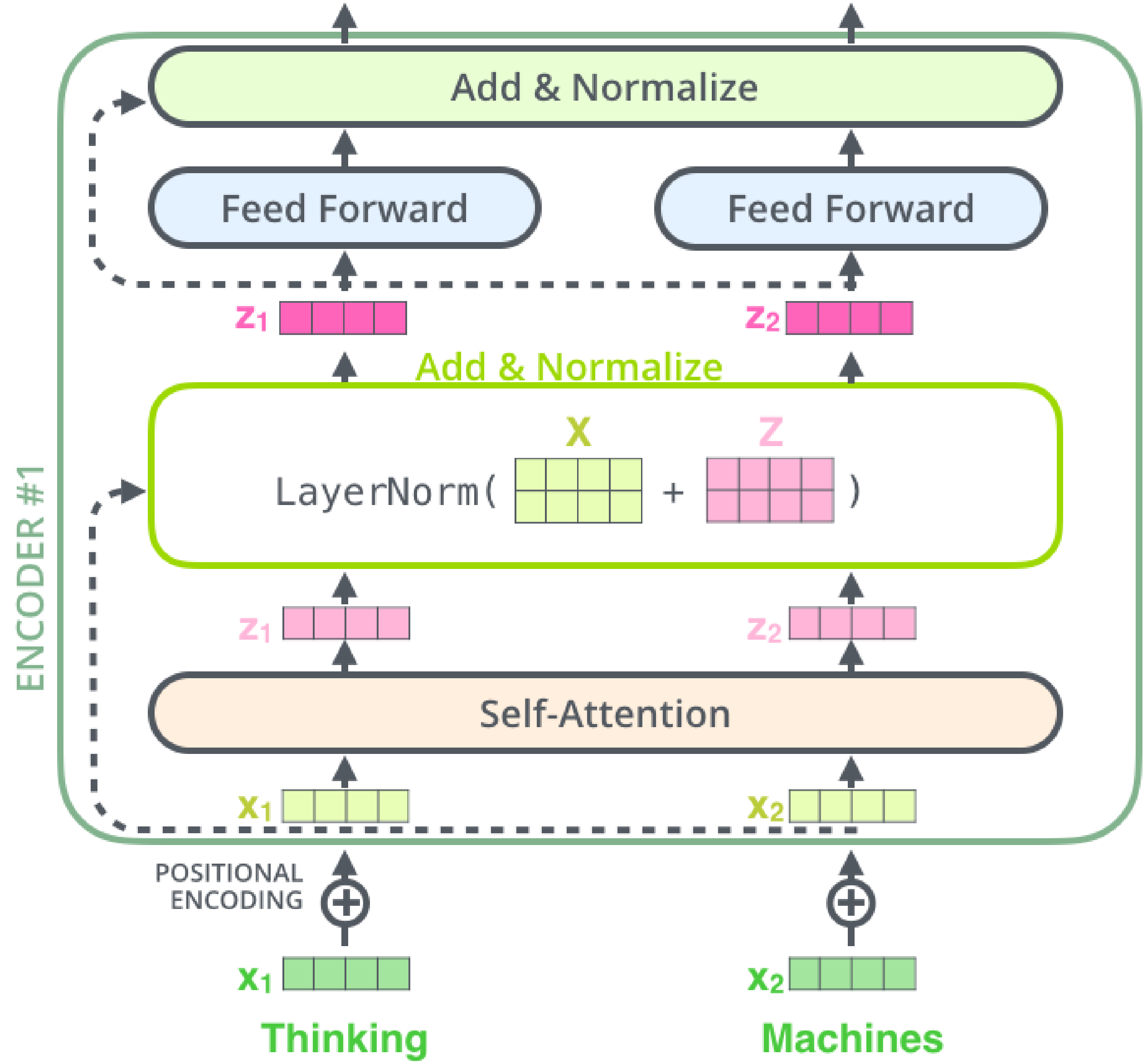


Positional encoding is built using sine and cosine functions.

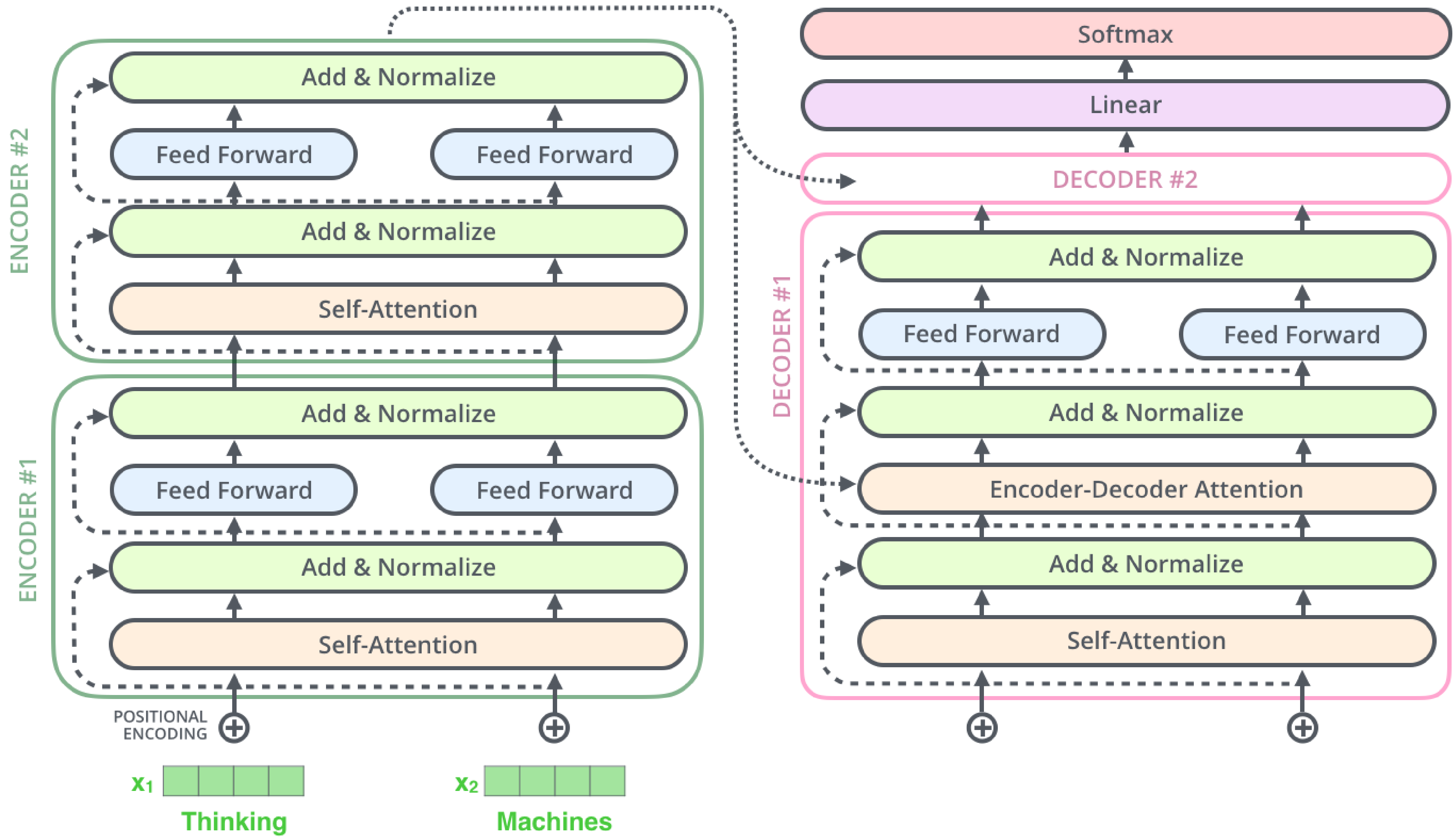
The Residuals

- It is a skip connection where the original input is added back to the output of:
  - Self-attention
  - Feed-forward network
- Why do we need residuals?
  - To avoid vanishing gradient problems.


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

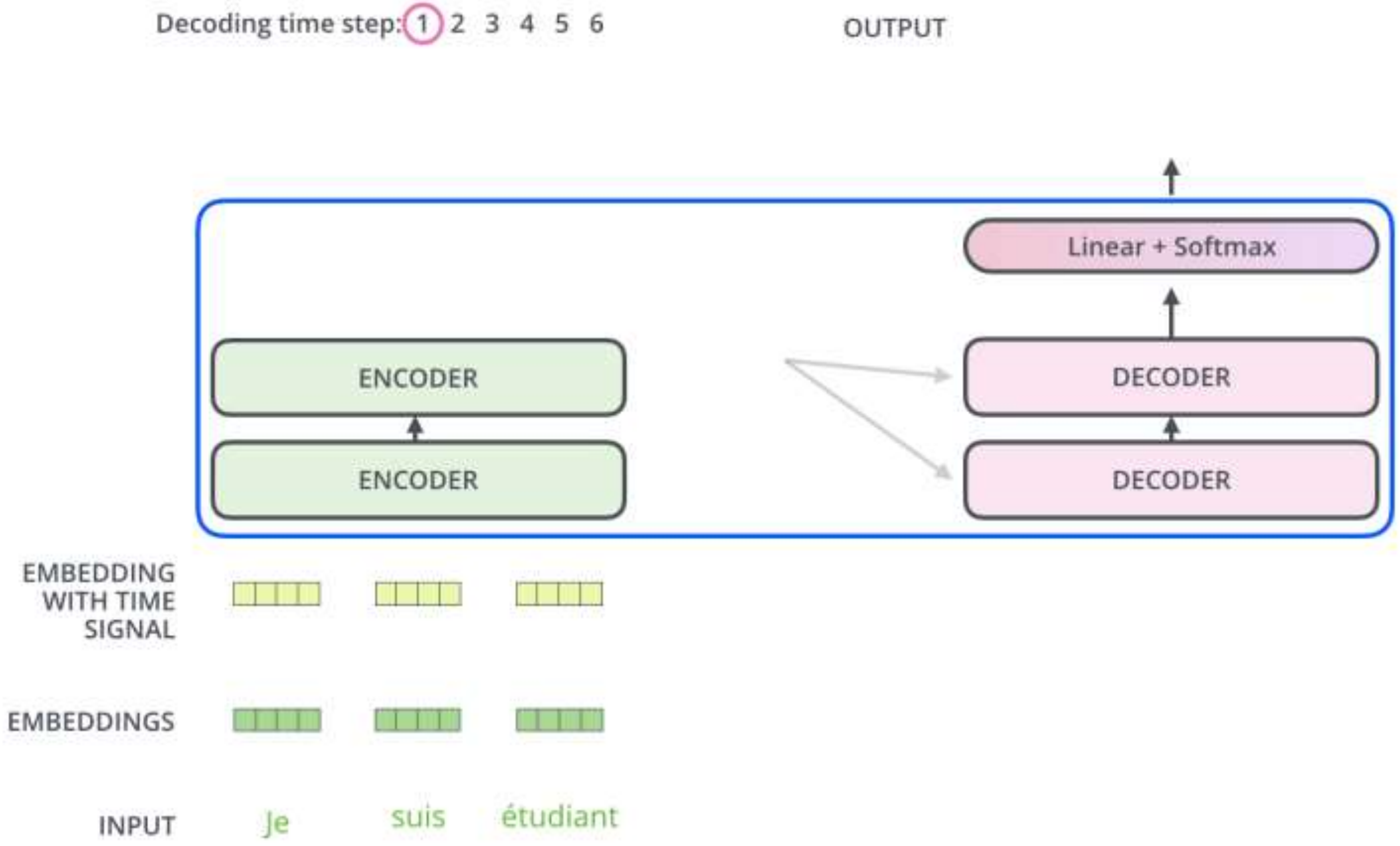


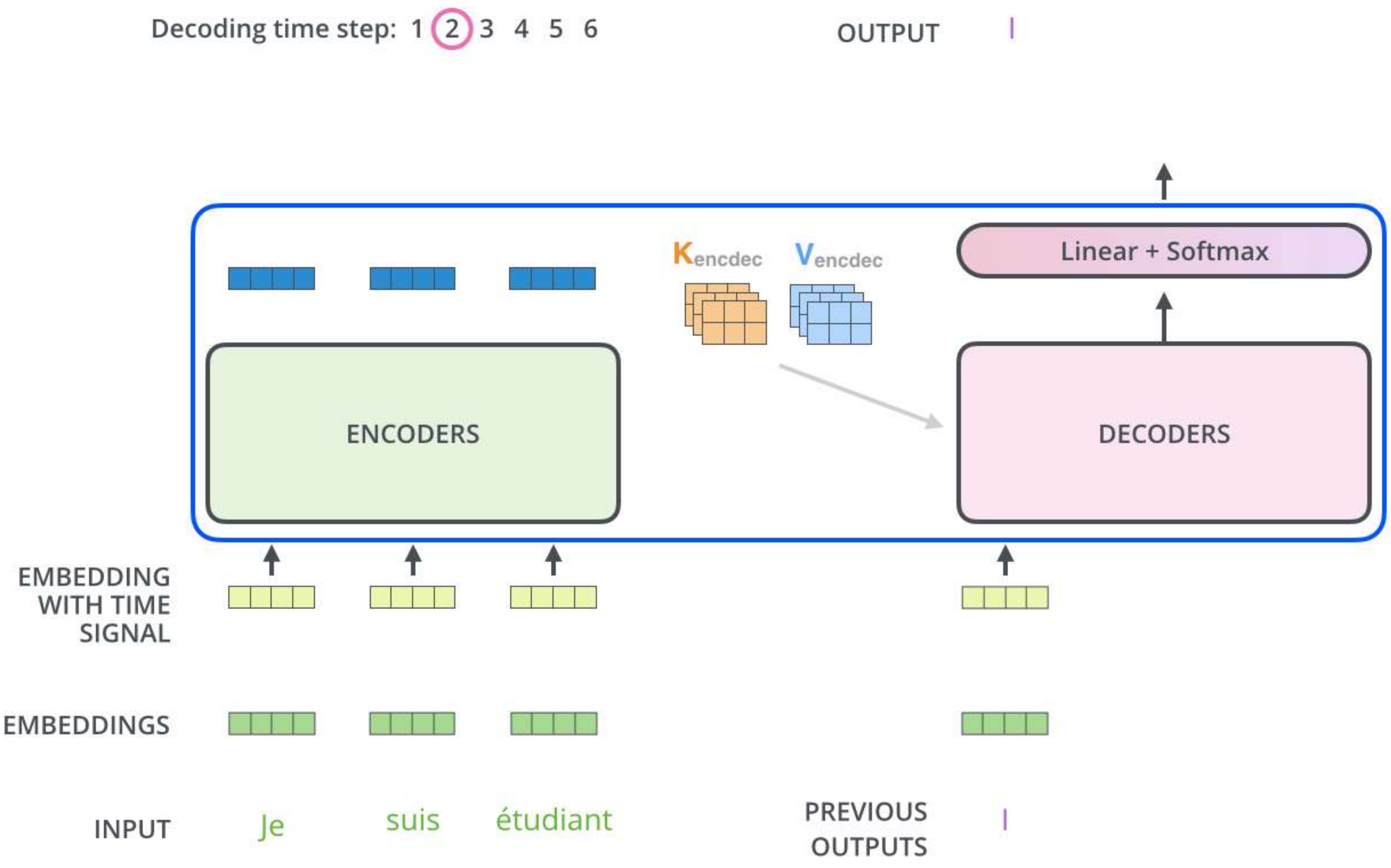
Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
z1
z2
Self-Attention
x1
x2
ENCODER #1
POSITIONAL
ENCODING
x1
x2
Thinking
Machines



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

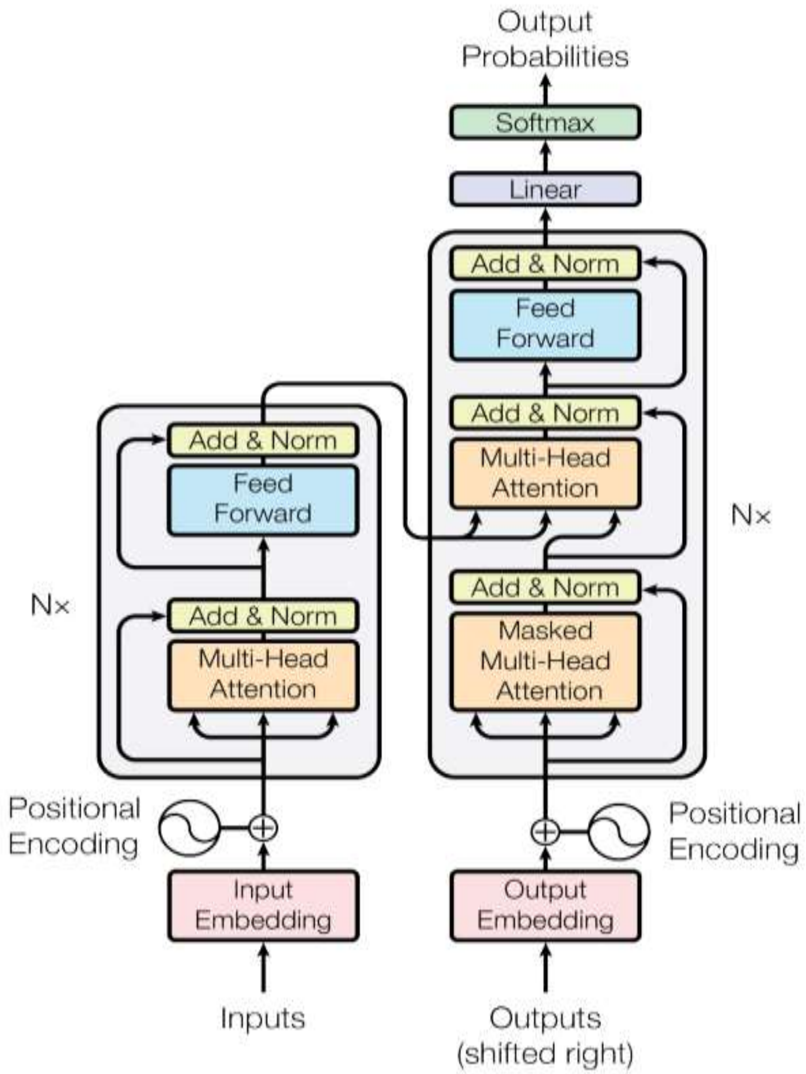
The Decoder Side





Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Kencdec
Vencdec
Linear + Softmax
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

The architecture from the paper



Reference

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

Thank You!