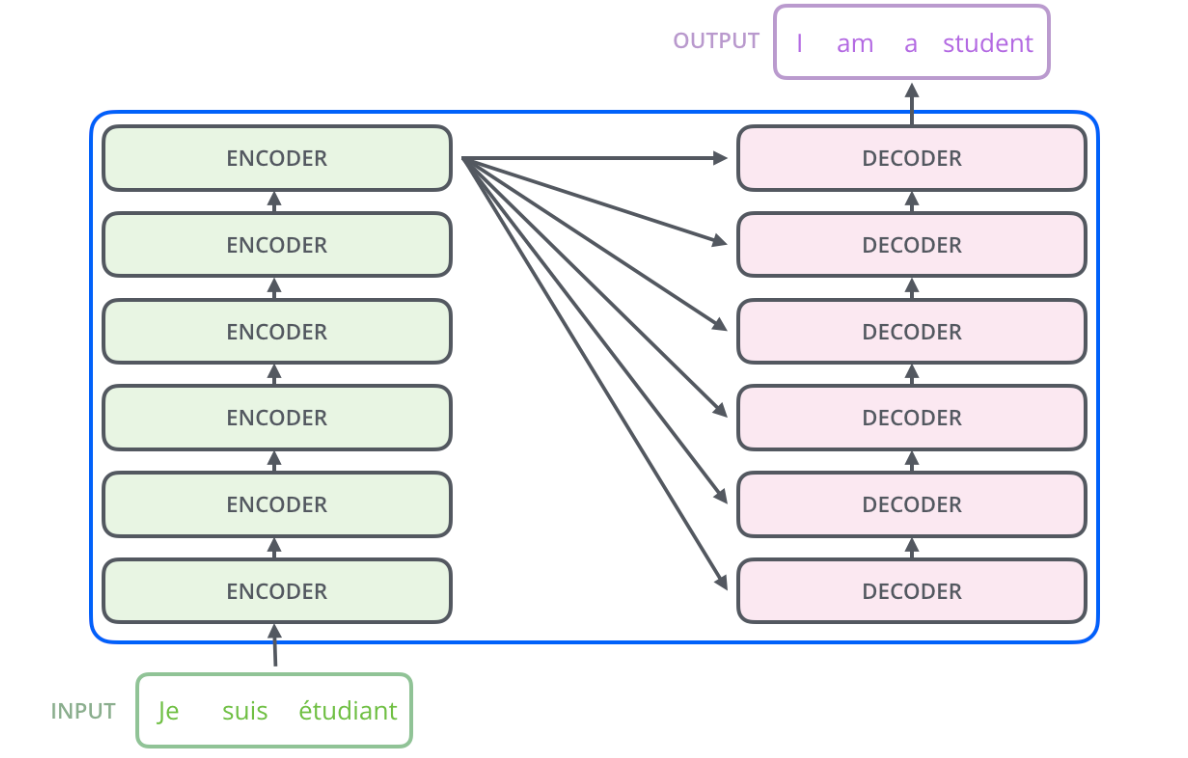
**Transformer Architecture for Long Language Modes (LLMs)**

Based upon the analysis and understanding of the research paper “Attention is all you need” by Vaswani et al. (2017)

Unlike traditional recurrent neural networks (RNNs) or long short-term memory (LSTM) networks, Transformers rely on self-attention mechanisms, allowing them to consider the entire input sequence simultaneously. Key components include:

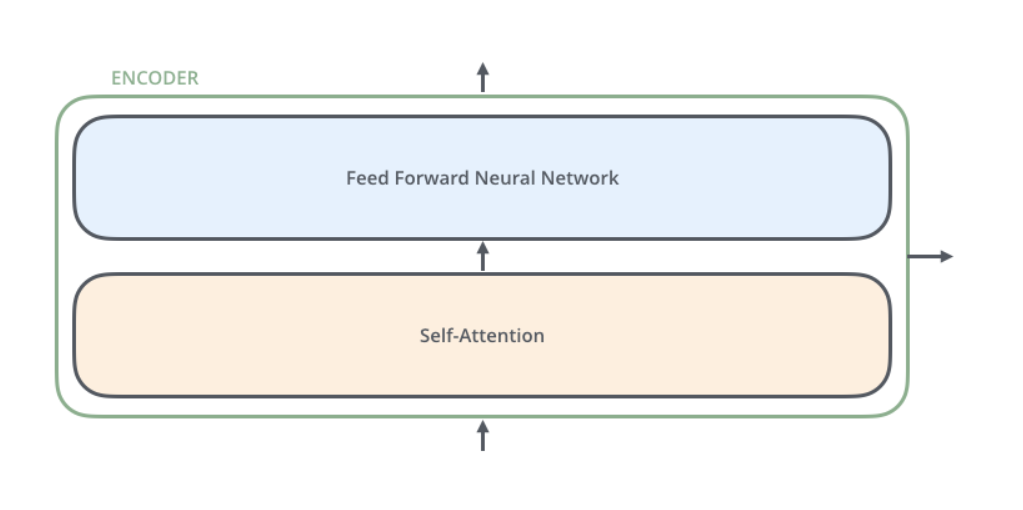
* **Self-Attention Mechanism**: Enables capturing contextual information by allowing each element in the input sequence to attend to all other elements simultaneously.
* **Positional Encoding**: Addresses the lack of inherent sequence order information in self-attention mechanisms by introducing positional information to the input embeddings.
* **Multi-Head Attention**: Multiple attention heads operate in parallel, each focusing on different aspects of the input. This enhances the model's ability to capture diverse relationships.
* **Feedforward Neural Networks:** Position-wise feedforward networks process information independently across positions, contributing to parallelization.
* **Layer Normalization and Residual Connections**: These elements enhance training stability and facilitate the training of deep models.

**Visual representation of transformer model:**

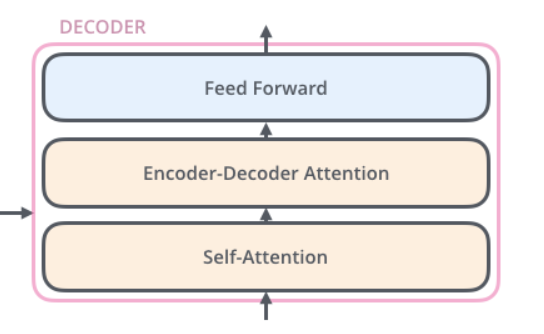


For every input in the transformer based model, the input goes through the encoder and decoder where it ultimately gets converted to the final required output. The encoding component is a stack of encoders. The decoding component is a stack of decoders of the same number.

The encoders are all identical in structure (yet they do not share weights). Each one is broken down into two sub-layers:



The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence

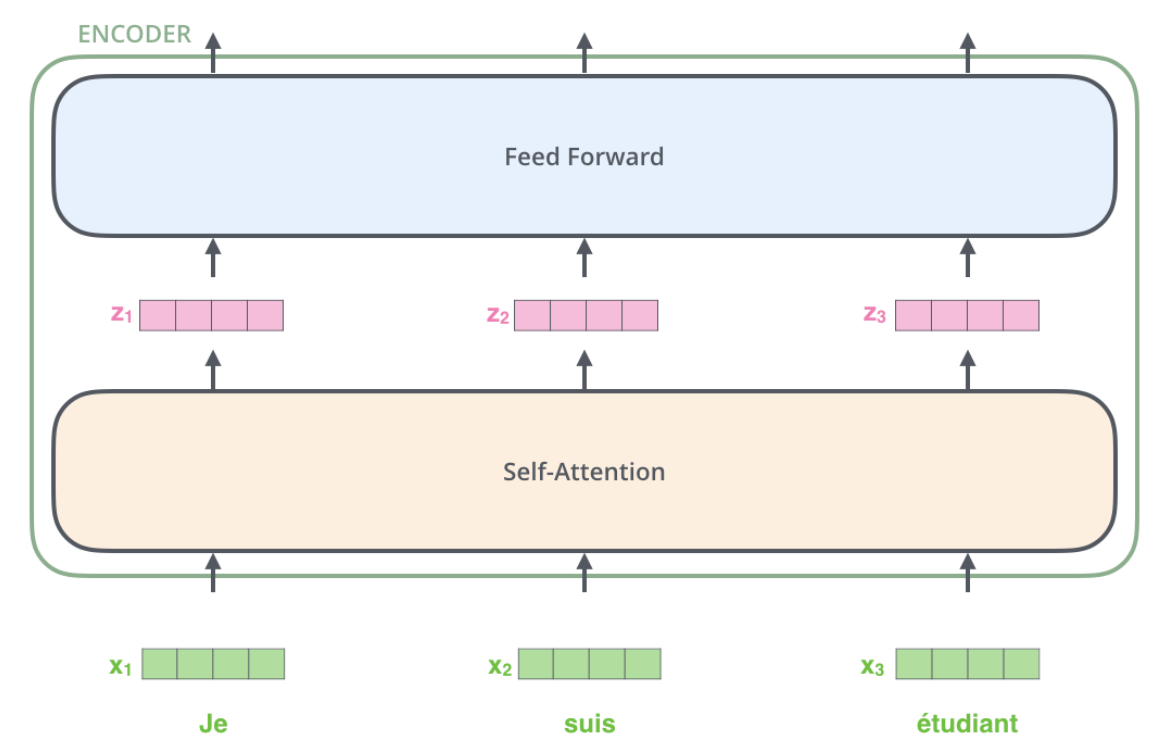


**Encoding the input:**

As is the case in NLP applications in general, we begin by turning each input word into a vector using an embedding algorithm.

Embedding occurs exclusively in the lowest encoder. The shared aspect among all encoders is that they are supplied with a collection of vectors, each with a size of 512. In the lowest encoder, these vectors represent word embeddings, while in subsequent encoders, they correspond to the output of the encoder directly beneath. The dimension of this vector collection is a customizable hyperparameter, typically set to the length of the longest sentence in our training dataset.

After embedding the words in our input sequence, each of them flows through each of the two layers of the encoder.



**Self of Attention Explained:**

In the sentence "The animal didn't cross the street because it was too tired," the word "it" refers to the "animal." During processing, self-attention enables the model to associate "it" with "animal" by examining other positions in the input sequence for contextual clues. Unlike RNNs, where hidden states maintain previous information, self-attention in Transformers integrates understanding from other relevant words into the current one being processed.

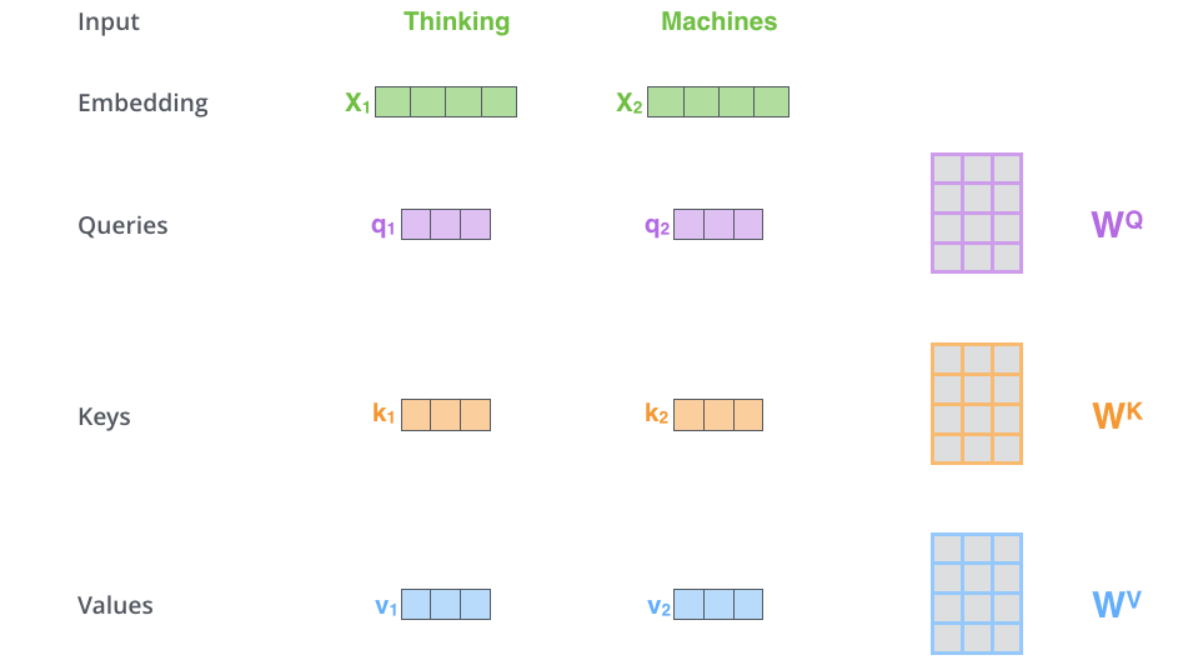
We create three vectors for each word in the encoder:

-Query

-Key

-Value

by multiplying the word's embedding with matrices trained during the training process.



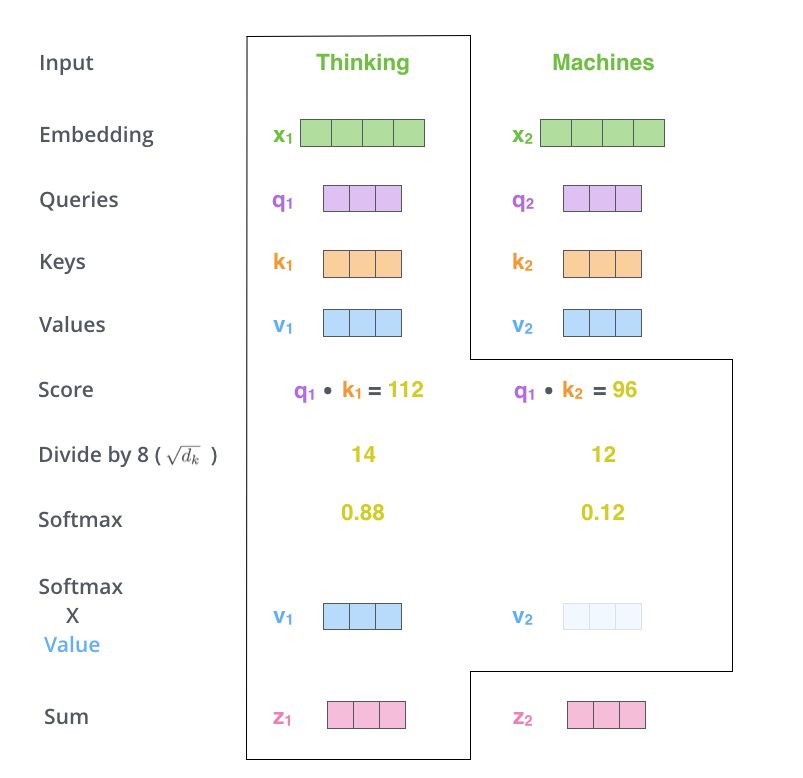
Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

In the second step of self-attention, scoring occurs. When processing the self-attention for the first word, like "Thinking," each word in the input sentence receives a score based on the dot product of its query vector (q1) with the key vector.

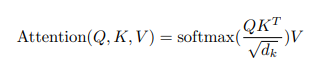
In the subsequent steps, we divide the scores by 8 (the square root of the key vector dimension in the paper, set to 64 for stable gradients). Following this, we apply a softmax operation to normalize the scores, ensuring they are all positive and collectively sum up to 1.

In the fifth step, we multiply each value vector by the softmax score to prepare for summation. This step aims to preserve the values of the focused words while minimizing the impact of irrelevant words, achieved by multiplying them with small numbers (e.g., 0.001).

Moving to the sixth step, we sum up the weighted value vectors, generating the output of the self-attention layer at this position (specifically, for the first word).



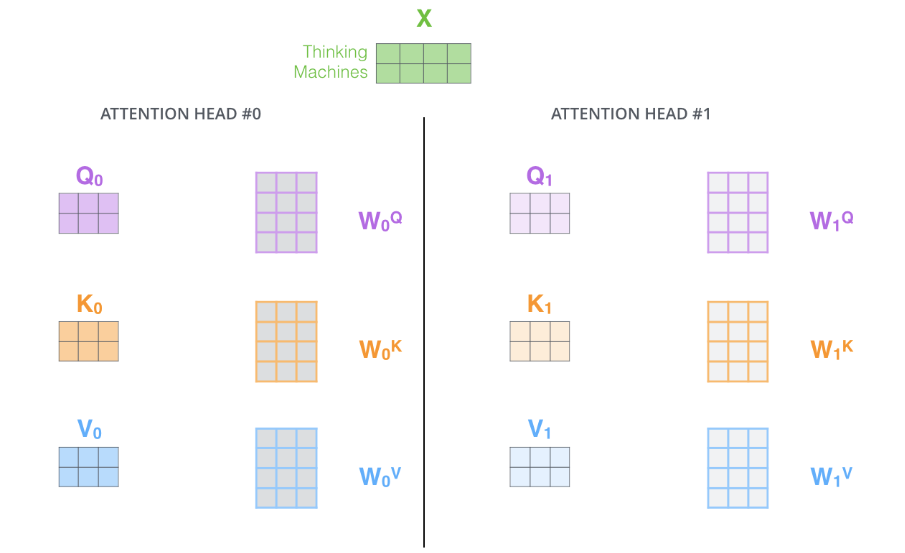
The entire process can be condensed into the following formula:



**Multi-Headed Attention:**

The paper improves the self-attention layer with "multi-headed" attention, offering two advantages:

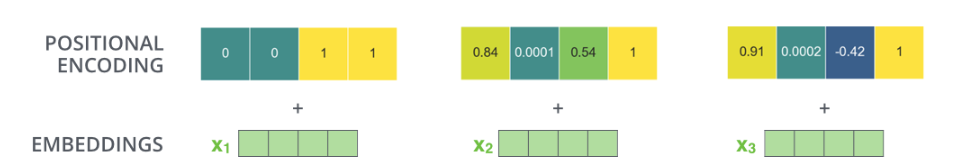
* **Broader Focus**: Multi-headed attention allows the model to concentrate on different positions, crucial for tasks like translation where understanding word references is vital.
* **Diverse Subspaces**: It introduces multiple sets of Query/Key/Value matrices (eight attention heads in the Transformer), each randomly initialized and later fine-tuned during training. These sets project input embeddings into distinct representation subspaces.

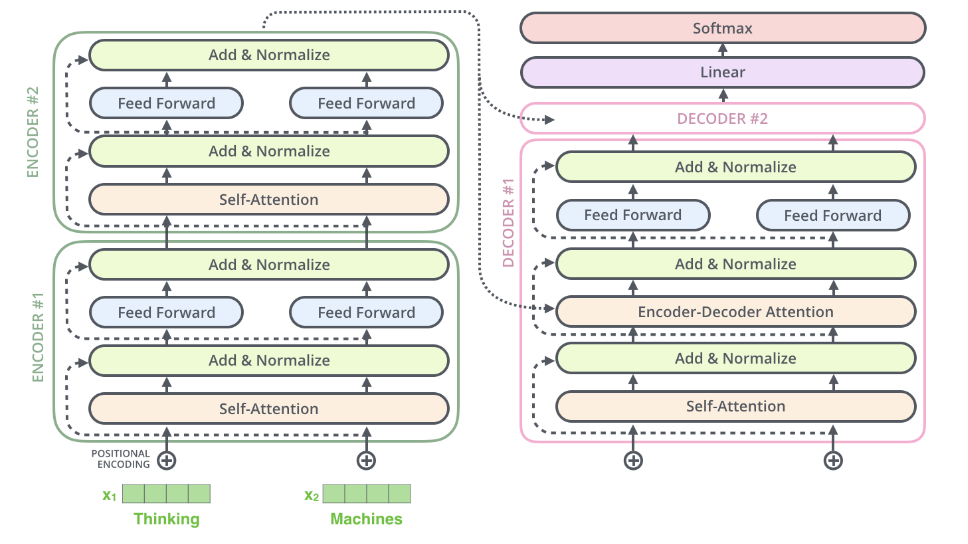


Executing the self-attention calculation outlined previously eight times with distinct weight matrices results in eight different Z matrices.

**Positional Encoding:**

To address the absence of word order consideration, the transformer incorporates position vectors into each input embedding. These learned vectors assist in understanding word positions and distances between them in the sequence, enhancing meaningful projections during Q/K/V vector creation and dot-product attention.





**The Decoder Side:**

After comprehending the encoder components, understanding the decoders involves recognizing their collaborative functionality.

* **Encoder Processing:** The input sequence undergoes processing by the encoder. The top encoder's output is transformed into attention vectors K and V, crucial for the "encoder-decoder attention" layer in each decoder. This allows the decoder to focus on relevant segments in the input sequence.
* **Decoding Phase:** Transitioning to the decoding phase, each step produces an element of the output sequence, such as an English translation sentence. Steps repeat until a special symbol signals completion.
* **Iterative Decoding:** Decoder outputs at each step become inputs for the subsequent step, akin to the encoder's iterative process. Decoder inputs are embedded, and positional encoding is added to indicate word positions.

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 multi-headed 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.