Department Of CSE

Byte Quest
Issue No 176

26 Sept 2025

Department Vision

To be acenter foracademicexcellence in the field of Computer Science and Engineering education to enable graduates to be ethical and competent professionals.

Department Mission

To enable students to develop logic and problem solving approach that will help build their careers in the innovative field of computing and provide creative solutions for the benefit of society.

Faculty Coordinators Dr. Bhargavi Peddireddy(Asc. Prof.)

Student Coordinators: (1602-23-733-118 Sneha) ,(1602-24-733-024) Mothilal Kumar

Transformers

The deep learning field has been experiencing a seismic shift, thanks to the emergence and rapid evolution of Transformer models.These groundbreaking architectures have not just redefined the standards in Natural Language Processing (NLP) but have broadened their horizons to revolutionize numerous facets of artificial intelligence.

What Are Transformers?

Transformers were first developed to solve the problem of sequence transduction, or neural machine translation, which means they are meant to solve any task that transforms an input sequence to an output sequence. This is why they are called "Transformers".

What Are Transformer Models?

A transformer model is a neural network that learns the context of sequential data and generates new data out of it.

To put it simply:

A transformer is a type of artificial intelligence model that learns to understand and generate human-like text by analyzing patterns in large amounts of text data.

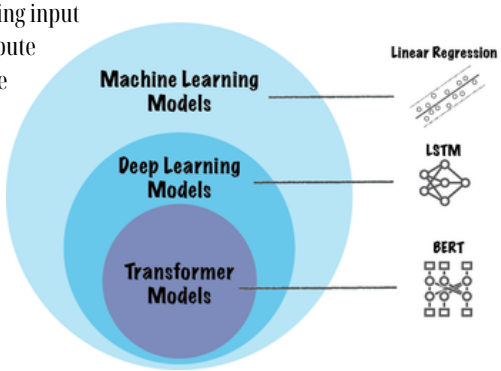
Transformers are a current state-of-the-art NLP model and are considered the evolution of the encoder-decoder architecture. However, while the encoder-decoder architecture relies mainly on Recurrent Neural Networks (RNNs) to extract sequential information, Transformers completely lack this recurrency.

So, how do they do it?

They are specifically designed to comprehend context and meaning by analyzing the relationship between different elements, and they rely almost entirely on a mathematical technique called attention to do so.

Transformer Architecture

Originally devised for sequence transduction or neural machine translation, transformers excel in converting input sequences into output sequences. It is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. The main core characteristic of the Transformers architecture is that they maintain the encoder-decoder model.



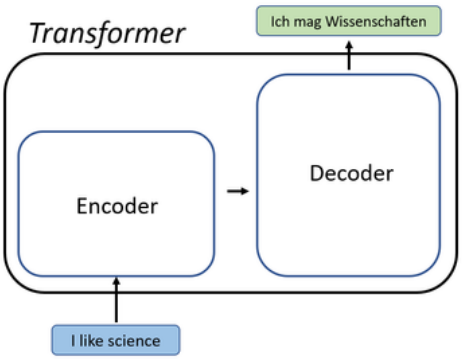
The Core Insight: Attention

The Transformer's central idea is the attention mechanism, especially self-attention.
Instead of processing words in order like RNNs, it asks:
"For this word, which other words should I care about, and how much?"
Example:
Sentence — "The cat that chased the dog was fast."
When looking at "was fast", the model assigns more weight (importance) to "cat" than "dog", because "cat" is the subject.
This weighting is learned from data.

The High-Level Structure

The Transformer model has two main parts:

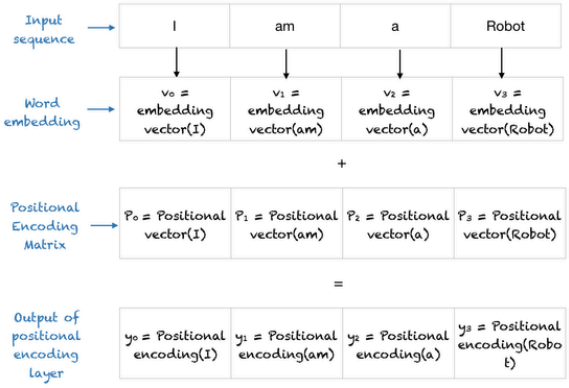
1. Encoder
   - Takes the input (e.g., a sentence in English) and produces a rich, contextual representation of each word.
   - This representation includes meaning + relationships between words.
2. Decoder
   - Takes the encoder's representation and generates output (e.g., a sentence in French).
   - At each step, it can look at both the encoder's output and the words it has generated so far.



In some cases (like BERT), we use only the encoder. In others (like GPT), we use only the decoder.

Input Embedding + Positional Encoding

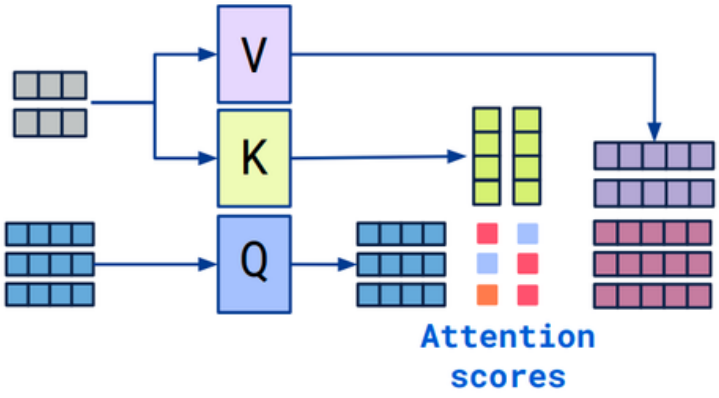
- Words are converted to vectors using an embedding matrix (like word2vec, but learned).
- Transformers don't have a sense of word order by default (unlike RNNs).
- To solve this, we add positional encoding.



Self-Attention (Core of Transformer)

For each token, we compute:
Query ()
Key ()
Value ()

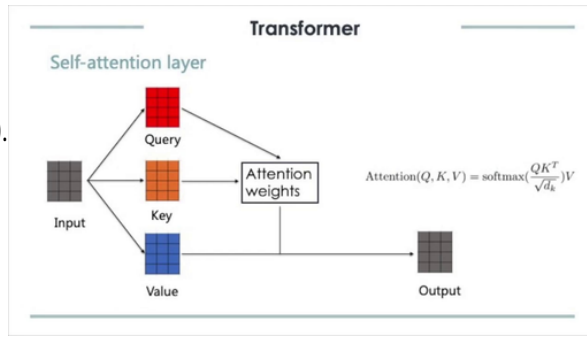
These are linear projections of the input embeddings:



$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

The attention score is:

$$Attention(Q, K, V) = softmax((QK^T)/sqrt(d_k))V$$



- QK^T : Measures similarity between words (who should attend to whom).
- Divide by sqrt(dk) : Prevents large values when dimension is big.
- Softmax: Turns scores into probabilities.
- Multiply by VVV: Collects information from relevant words.

This allows each word to "look at" others in the sentence.

Multi-Head Attention to Output Generation

After self-attention, the Transformer uses Multi-Head Attention, which means instead of computing a single attention distribution, the model runs several attention mechanisms in parallel, known as heads, and then concatenates their outputs. This allows the model to learn different types of relationships simultaneously (for example, one head might focus on nearby dependencies while another captures long-range ones). Mathematically, this is expressed as

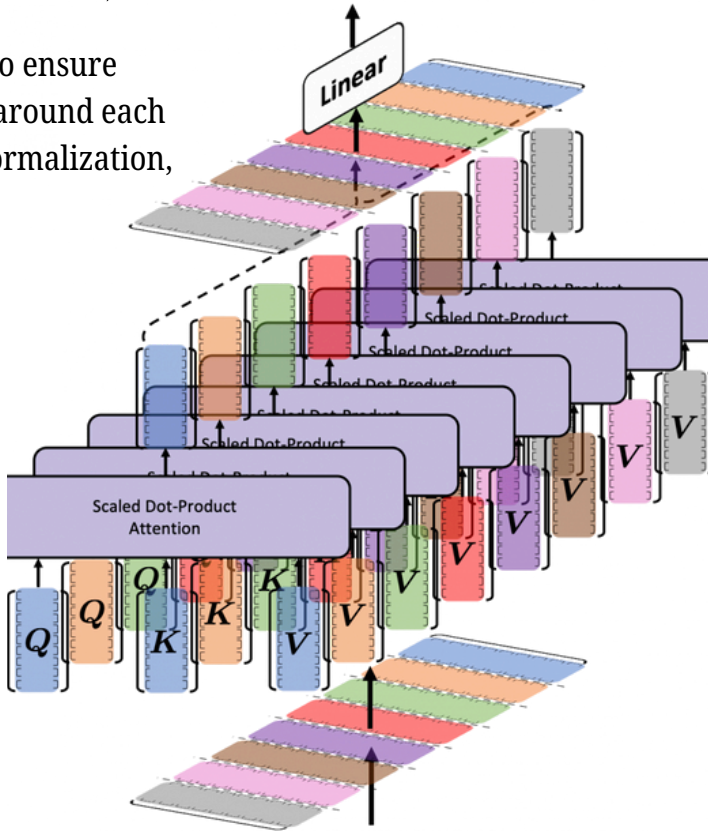
$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

where each head is an independent scaled dot-product attention. The result of multi-head attention for each token is then passed through a position-wise Feed-Forward Network (FFN), which is the same for all positions but applied independently to each toke This FFN consists of two linear layers with a non-linearity (usually ReLU) in between, and can be written as

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

which increases the representational power of the model. To ensure stable training, the Transformer uses residual connections around each sublayer (attention and feed-forward), followed by layer normalization, so that each block output looks like

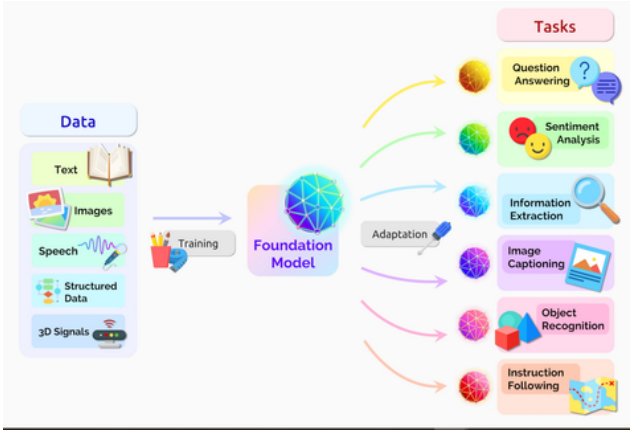
$$x + \text{SubLayer}(x)$$



By stacking multiple encoder layers like this, the Transformer builds very deep hierarchical representations of the input sequence. The decoder follows a similar design but includes two key differences: first, it uses Masked Multi-Head Attention, where the attention mechanism is modified so tokens cannot attend to future positions, ensuring the model generates outputs left-to-right; second, it introduces an Encoder-Decoder Attention block, where the queries come from the decoder and the keys and values come from the encoder's output, allowing the decoder to focus on relevant parts of the input sequence. Finally, the decoder output is passed through a linear layer followed by a softmax to produce probabilities over the vocabulary, enabling word-by-word generation.

Output Layer of the Transformer

The output layer is where the decoder's hidden states are converted into actual word predictions. Each hidden vector is passed through a linear transformation that maps it to the vocabulary size, producing raw scores called logits. A softmax function is then applied to turn these logits into probabilities over all possible words. During training, this probability distribution is compared to the true target word using a loss function (like cross-entropy). At inference, the model picks or samples the highest-probability word at each step and continues until an end-of-sequence token is reached, thus generating text.



🚀 Why Transformers Are So Powerful

- **Parallelization:** Unlike RNNs, process all tokens simultaneously.
- **Long-range dependencies:** Attention connects distant words easily.
- **Scalability:** Works with billions of parameters on GPUs/TPUs.
- **Versatility**: Works for text, speech, images, proteins, etc.