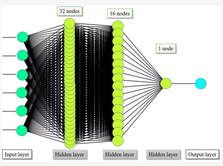
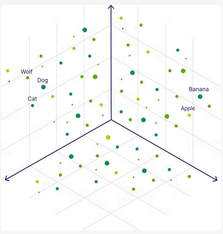
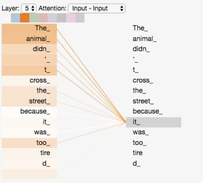
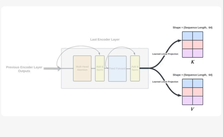
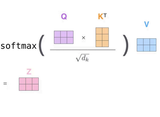
1. The Brain of AI

At the heart of modern AI, powering everything from Large Language Models (LLMs) to cutting-edge AI Agents and other Artificial Intelligence (AI) systems, lies a revolutionary architecture known as the **transformer**. Essentially the 'brain' of these systems, the transformer is responsible for enabling key capabilities: storing information (**Memory**), processing language (**Language**), understanding meaning (**Concepts**), reasoning (**Thinking**), formulating possibilities (**Hypotheses**), making choices (**Decisions**), and communicating those choices (**Speaking**), all while continuously **Learning** to improve. To achieve this, it operates using its own unique system of communication, where its alphabet consists of units called tokens. In this guide, we will embark on a comprehensive deep dive into the transformer's components and workings, providing you with a foundational understanding of this transformative technology.

 **by Jason Guillauto**

Transformer Architecture

Dive into the fundamental blocks of the transformer architecture in this section.

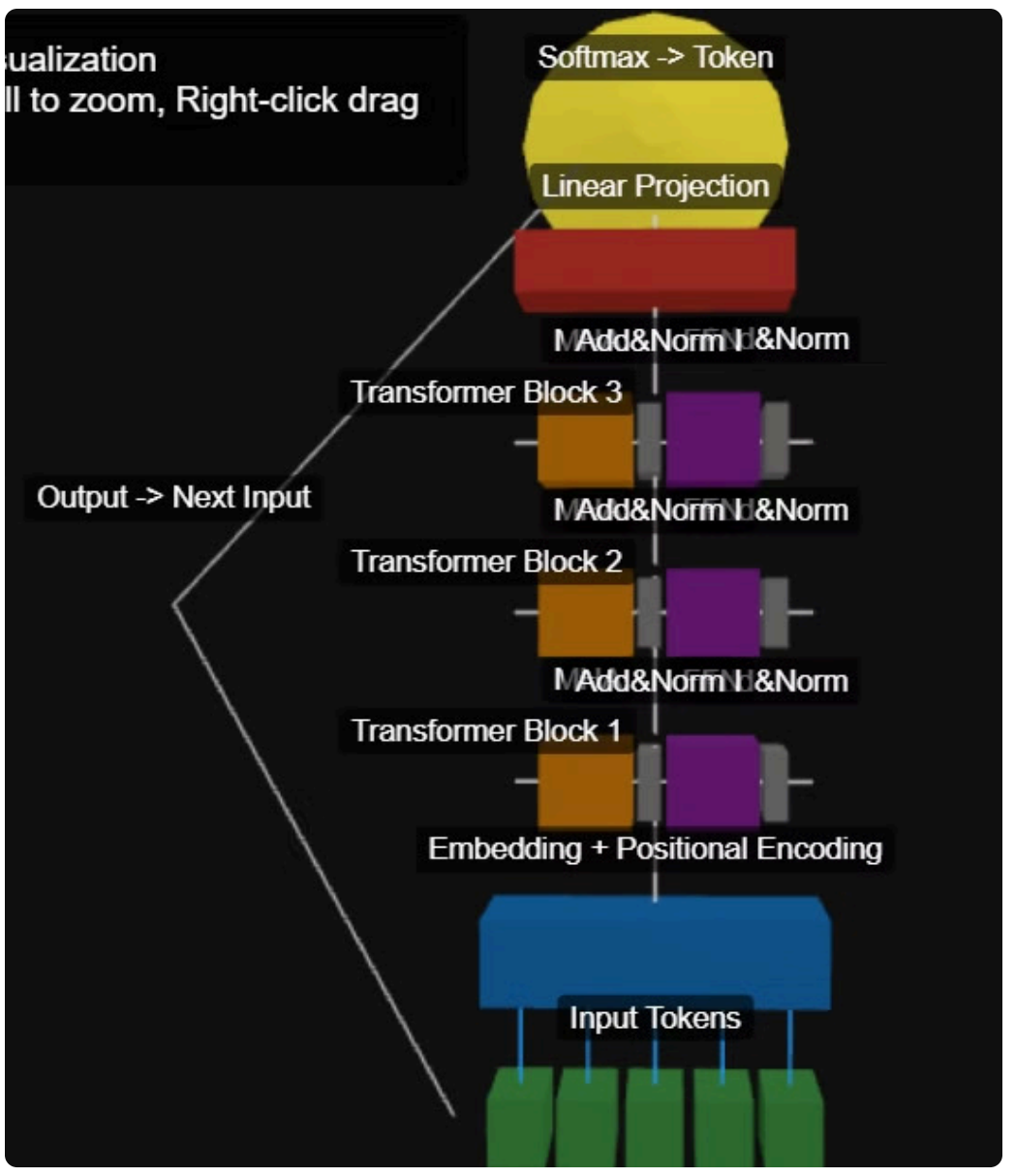
Functions/Components	Visual Representation	Description
Memory: Multi-Layer Perceptron (MLP) – a feedforward neural network component that processes information through multiple layers of neurons.		A Multi-Layer Perceptron (MLP) is a type of neural network that's really good at making decisions. Think of it like a detective team. Each detective (neuron) looks for clues (features) in the data. They pass their findings to other detectives, who combine the clues to form a bigger picture. Eventually, the team reaches a conclusion: 'The suspect is guilty,' or in AI's case, 'This image is a cat.'
Language: Processed as discrete **tokens** representing words and subwords		In a transformer, **language** is handled by breaking it down into discrete units called tokens. These tokens can be words, sub-words, characters,or even parts of images or other data. Large Language Models (LLMs) often work with vocabularies of tens of thousands of tokens, allowing them to represent a wide range of language.
Concepts: Encoded as high-dimensional numerical **vectors** in embedding space		The concepts in the transformer are encoded as high-dimensional numerical vectors in embedding space. These vectors carry semantic meaning and enable the model to understand relationships between different concepts. By representing concepts in this manner, the transformer can efficiently navigate and process complex information, enhancing its ability to perform a wide range of tasks accurately.
Thinking: Performed through layers of **transformer blocks** with **self-attention** mechanisms		**Transformer blocks** are the repeating units in the transformer architecture. They refine the model's understanding of language through self-attention, a technique that allows the model to focus on the relationships between words (or tokens) in a sentence, weighing each word's importance based on context. • **Multi-Head Attention:** At the bottom, this is where the model focuses on relationships between words in the input. The Q, K, and V represent Query, Key, and Value vectors, used to calculate attention. • **Add & Norm:** These are 'Add and Normalize' layers, used twice to stabilize the learning process. • **Feed Forward:** This is a neural network that further processes the information.
Self-Attention		**Self-attention** is a crucial technique in the transformer architecture that allows the model to understand the relationships between words (or tokens) in a sentence. It works by having each word 'pay attention' to all other words, weighing their importance to its own meaning, similar to how you might focus on certain words in a sentence to understand its overall context.
The transformer block equation		• **Input:** The original text or data that's fed into the transformer (e.g., "Je suis"). • **Embedding:** Converting the input words (or "tokens") into numbers that the model can understand. • **Queries:** Represent what each word is "asking" or "looking for" in the other words. • **Keys:** Represent what each word "offers" as information to other words. • **Values:** The actual information each word provides. • **Score:** A calculation (often a dot product) that measures how relevant each word is to the others. • **Divide by 8:** A scaling step (using the square root of the key dimension, likely 8 here) to stabilize the scores. • **Softmax:** Converting the scores into probabilities (numbers that add up to 1), showing how much "attention" each word should pay to others. • **Softmax x Value:** Weighting the "value" of each word based on the softmax probabilities. • **Sum:** Combining the weighted values to get the final representation of each word, incorporating context.
Hypotheses: Generated via **linear projection** layers that store the best embeddings in a hypotheses block		**Linear projection** is like stretching or rotating a picture. You're not changing the picture itself, but you're showing it in a different way that makes certain parts clearer. In AI, linear projection can be used to simplify complex data, like converting text into a format that the model can more easily process. It's similar to how a map projects the 3D Earth onto a 2D surface.
Decisions: Made through **softmax probability** distributions across possible tokens		Think of a group of people voting for their favorite color. Each color gets a 'score' based on how many votes it received. Softmax is like a system that takes those scores and turns them into percentages. So, if red gets a lot of votes, it gets a high percentage (like 70%), while blue gets fewer votes and a lower percentage (like 20%).
Speaking: Executed through an **autoregressive loop** that generates one token at a time		An **autoregressive loop** is a process where a model generates output step-by-step, using its own previous output as input for the next step. It's how transformers 'speak' one word at a time.
Learning: Achieved through **backpropagation** by calculating loss and adjusting weights		**Backpropagation** is how AI models, including transformers, learn from their mistakes. It's like a student getting feedback on a test: the model makes a guess, sees how wrong it was, and then tweaks itself to be more accurate next time.

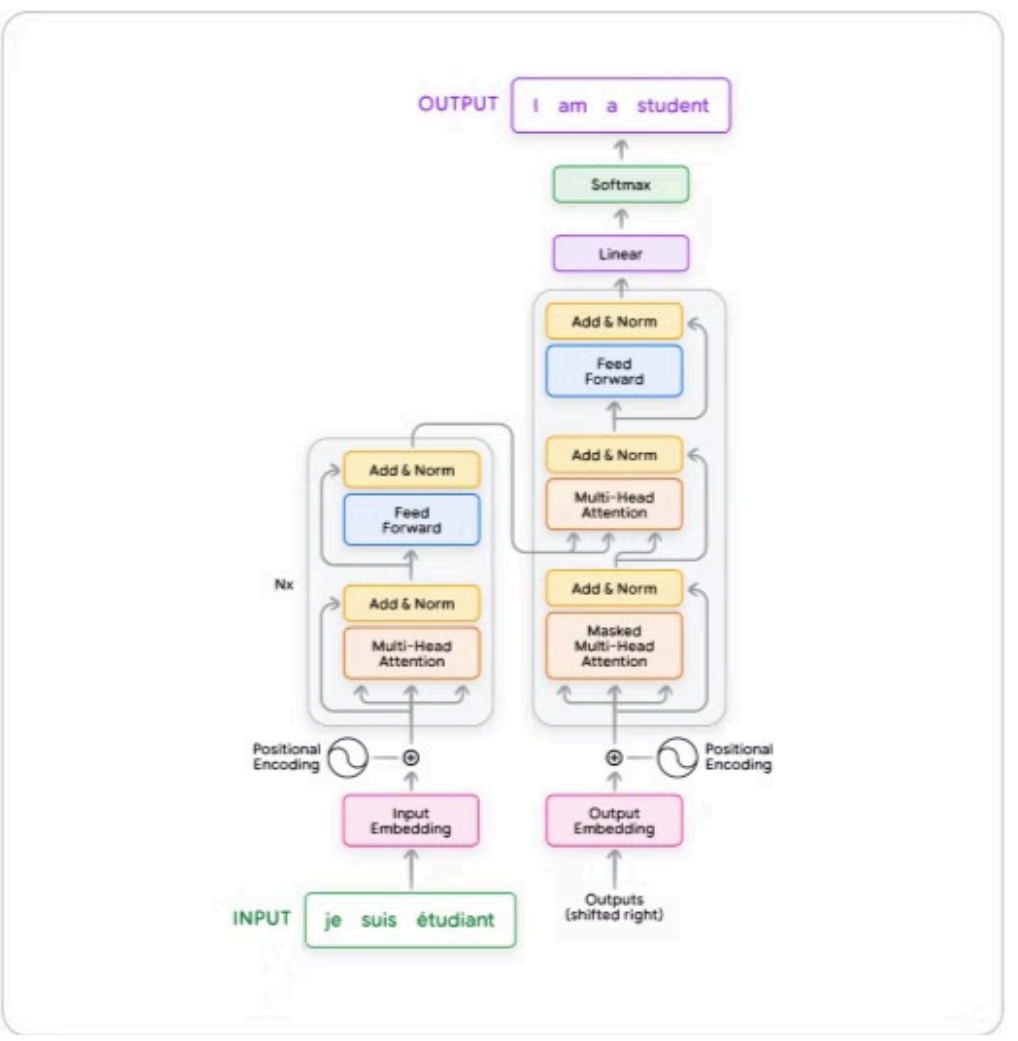
Glossary of Key Technical Terms

Reference guide for important concepts in transformer architecture

Term	Definition
Transformer	A neural network architecture based on self-attention mechanisms, designed to process sequential data without using recurrence or convolution.
Attention Mechanism	A technique that allows the model to focus on different parts of the input sequence when producing an output, mimicking human cognitive attention.
Self-Attention	A specific type of attention where the model relates different positions within a single sequence to compute a representation.
Embedding	A numerical representation of data (like words or tokens) in a continuous vector space, capturing semantic relationships between items.
Softmax	A mathematical function that converts a vector of numbers into a probability distribution, ensuring all values are between 0 and 1 and sum to 1.
Backpropagation	An algorithm for calculating gradients in neural networks, enabling the model to learn by adjusting weights based on prediction errors.
Linear Projection	A mathematical operation that transforms vectors from one space to another using matrix multiplication, enabling dimension changes.
Token	A basic unit of text in natural language processing, which could be a word, subword, or character depending on the tokenization method.
Autoregressive Loop	A process where the model generates output sequentially, with each new token dependent on previously generated tokens.
MLP	Multi-Layer Perceptron, a feedforward neural network component used in transformers to process information after attention mechanisms, consisting of linear layers with non-linear activation functions.
Query	In attention mechanisms, a representation of the current token used to determine its relationship with other tokens (keys) in the sequence, helping to calculate attention scores.
Vector	A mathematical structure consisting of an ordered array of numbers, used in AI to represent data in multi-dimensional space where each dimension encodes specific features or attributes.

Visual representation of a transformer





SIMPLE

1. **Inputs** are **tokenized:** The input text is broken down into tokens.
2. **Tokens** are **embedded** into **vectors:** Tokens are converted into numerical vector representations.
3. **Vectors** are processed through **transformer blocks** using **self-attention** mechanisms.
4. The processed **vectors** are transformed into scores for the next **token** in the **linear projection layer.**
5. **Tokens** are generated (output) by sampling from the probability distribution produced by the **softmax function**. The **softmax function** converts the scores from the **linear projection** layer into probabilities.
6. The **output token** is then fed back into the model as input for the next step (in decoder-only models). This describes the **autoregressive loop**.
7. The process is repeated until the desired output length or a termination condition is met. This clarifies the iterative nature.

COMPLETE

1. **Understanding Input (Encoder - Left Side):**
   - The process starts with the **Language** input "je suis étudiant", represented as **Tokens**.
   - These tokens are transformed into initial **Concepts** through Input Embedding and Positional Encoding, creating numerical **Vectors** capturing meaning and order.
   - The Encoder stack performs **Thinking**. Its Nx **Transformer Block** layers use Multi-Head Attention and Feed Forward networks to process these concepts, building a deep contextual understanding of "je suis étudiant".
   - Each transformer block contains **Add & Norm** components – critical stabilizing mechanisms that ensure efficient training. The "Add" implements residual connections, allowing information to flow directly through the network by adding the original input to the processed output. The "Norm" applies layer normalization, standardizing the outputs to control their scale, reducing training time and improving stability. Together, these components prevent vanishing/exploding gradients and maintain the network's ability to learn effectively across many layers.
2. **Generating Output (Decoder - Right Side / Autoregressive Loop):**
   - The **Speaking** begins via the autoregressive loop on the right side, generating "I am a student" one token at a time.
   - It uses Output Embedding and Positional Encoding to create **Concepts** from the output generated so far (starting with an initial token).
   - The Decoder stack then engages in **Thinking**. Its Nx **Transformer Block** layers use Masked Multi-Head Attention (to focus on previous output tokens) and standard Multi-Head Attention (to consult the encoded input **Concepts** from the Encoder), along with Feed Forward networks.
   - After **Thinking**, the final **Linear** layer generates **Hypotheses** – outputting raw scores (logits) for every possible *next* English token.
   - **Decisions** are made via **Softmax**, converting scores into probabilities to select the most likely next token (e.g., 'I', then 'am', 'a', 'student').
   - This selected token is the OUTPUT for that step and feeds back into the loop, continuing the **Speaking** process until the translation is complete.

The Complete Transformer Architecture

The transformer architecture forms the neural backbone of modern Large Language Models through its elegant design of parallel processing and attention mechanisms. At its core, transformers consist of dual components: an encoder that processes input by converting tokens into contextual representations through self-attention and feed-forward networks, and a decoder that generates output through an autoregressive loop. The critical innovation of transformers lies in their multi-head attention mechanism, which allows models to simultaneously consider different aspects of context while processing language. This architecture is enhanced by residual connections and layer normalization that stabilize training across many layers. Together, these components enable LLMs to perform sophisticated language understanding and generation tasks, processing information bidirectionally to capture nuanced meanings, relationships, and contexts that were impossible with previous architectures. This fundamental design has revolutionized natural language processing, allowing models to scale to billions of parameters while maintaining computational efficiency.