Understanding LLM Architecture

Aditya Chatterjee

What are Large Language Models?

What is a Language Model?
A language model is like a smart assistant that can read and understand text, respond to questions, write essays, and even complete sentences. LLMs are just huge versions of these assistants, trained on billions of examples from books, websites, and other text data.

Why Should We Care?
LLMs make it possible for machines to understand human language, leading to advancements in chatbots, translation tools, and search engines like Google.

The Core Idea Behind LLMs

How Do LLMs Work?

Imagine you're trying to guess the next word in a sentence based on what came before. LLMs do something similar by learning patterns in text. For example, if you say, "The sky is," the LLM will likely guess the next word is "blue" because it's seen that combination many times in its training.

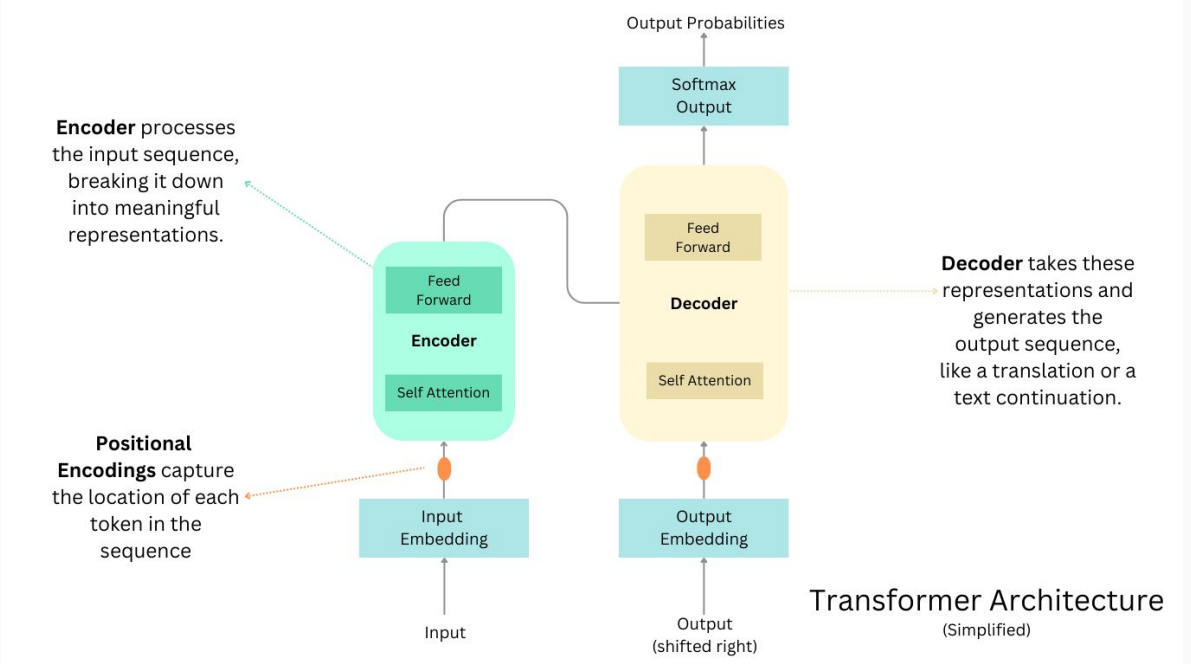
Learning from Big Data

LLMs don't memorize individual facts. Instead, they learn to recognize general patterns from huge amounts of text, just like a person gets better at understanding language by reading a lot of books.

Understanding Transformers

Why Transformers?
Before transformers, models used to read text one word at a time, which was slow and inefficient. The transformer model can read and understand all the words in a sentence at once, speeding up the process.

Attention Mechanism
This helps the model focus on important words. Think of it as a spotlight highlighting relevant parts of a sentence. For example, in the sentence “The dog chased the ball,” attention helps the model focus on "chased" when trying to understand the action.



1000x

The transformer architecture allowed models to process text up to 1,000 times faster than older methods, like RNNs (Recurrent Neural Networks), by handling all words in a sequence simultaneously instead of one at a time. This breakthrough made training huge language models like GPT and BERT possible, changing the field of natural language processing almost overnight.

What’s Inside a Transformer?

Encoders and Decoders

- Encoder: This part looks at the input text and processes it. Think of it like reading a book and turning it into a summary in your head.
- Decoder: This takes that summary and turns it back into readable text or answers. It’s like converting your mental notes into full sentences.

Multi-Head Attention

Instead of focusing on just one part of the text, the model can look at multiple parts at once to get a better understanding. Imagine reading a paragraph but being able to glance at different sentences to piece everything together.

Multi-Head Attention Mechanism

What is Multi-Head Attention?
Multi-head attention is a key part of the transformer's ability to process text effectively. Instead of focusing on just one part of the text at a time, the model breaks the input into multiple heads, allowing it to focus on different parts of the sentence simultaneously.

How Does it Work?
Let's say we have a sentence like "The cat sat on the mat." Each attention head might focus on a different word or relationship:

- One head might focus on "cat" and "sat" to understand the subject and verb relationship.
- Another head might focus on "the" and "mat" to understand the object.
- By combining information from multiple heads, the model gets a more complete picture.

Number Of Attention Heads

Why Multiple Heads?

More attention heads allow the model to focus on different parts of the sentence in parallel. Each head attends to different word relationships, helping the model understand the full context.

Impact of the Number of Heads

Fewer Heads (e.g., 4 heads): The model might struggle to capture all important relationships in complex sentences.
More Heads (e.g., 12 heads or 16 heads): The model can handle much more complexity, understanding deeper relationships between words. However, too many heads can also increase computational cost without providing proportional improvements.

Standard LLM Configurations:

- GPT-2 Small: 12 heads
- GPT-3: 96 heads
- BERT Base: 12 heads
- LLaMA: 16 heads

Why 12 or More Heads?

Models like GPT-2 and BERT Base use 12 attention heads because this number strikes a balance between performance and computational efficiency. It allows the model to attend to different parts of a sentence from different perspectives without slowing down training too much.

Attention Scores and Softmax

- **How Does Attention Work?**
  In each attention head, the model calculates how much attention each word in the sentence should get. This is done using a function called **scaled dot-product attention**, which compares the words and assigns a score to each pair.
- **Example**:
  For the sentence “The cat sat on the mat,” the word “cat” might get a high attention score relative to “sat,” while “on” might get a lower score. These scores help the model understand which words are important for generating the output.
- **Why Scale the Scores?**
  Without scaling, the attention scores can become too large and lead to unstable training. The scaling factor (usually the square root of the dimensionality of the input) ensures that the scores remain manageable.
- **Softmax Function**:
  The scores are then passed through a **softmax** function, which turns them into probabilities that sum to 1. This way, the model can decide how much weight to give to each word in the final output.

Feedforward Networks

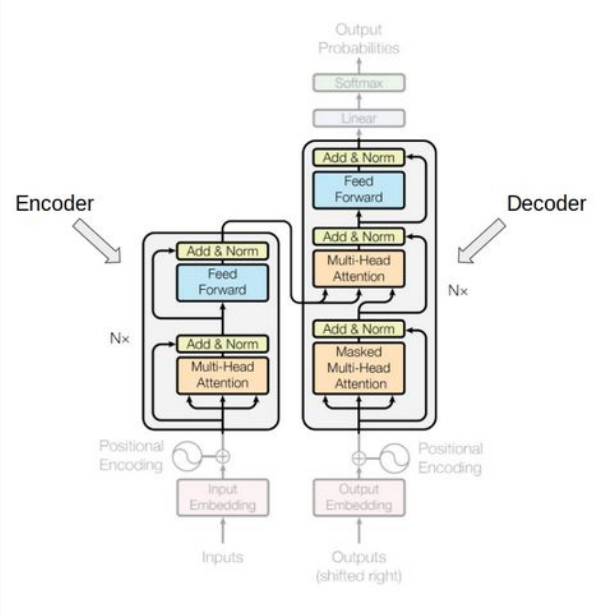
What Happens After Attention?

After the attention mechanism, each word (or token) in the input is passed through a fully connected **feedforward network**. This network helps the model transform the attention-weighted input into more meaningful representations.

Key Parameters:

Hidden Layer Size: In typical LLMs, the hidden layer is much larger than the input size. For example, in BERT, the hidden layer is often **4 times** the size of the input.

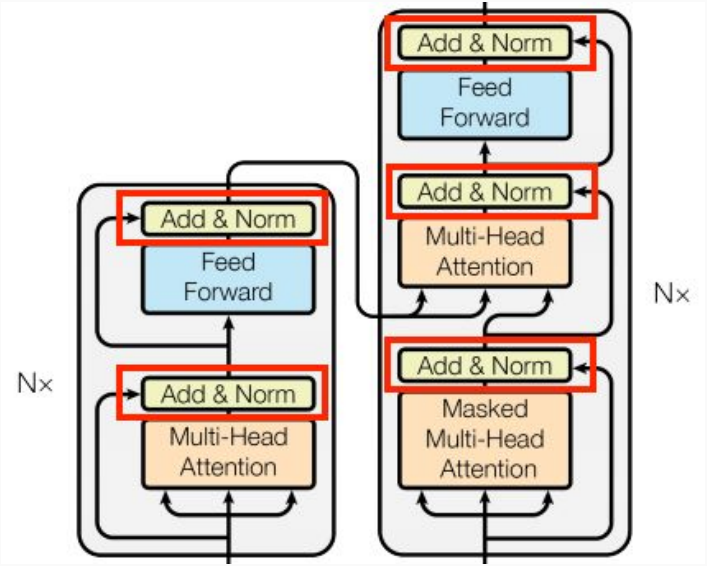
Non-linearity: These feedforward networks use an activation function like **ReLU** (Rectified Linear Unit) to introduce non-linearity, which helps the model learn more complex relationships.



Layer Normalization and Residual Connections

Why Use Layer Normalization?
During training, the model's internal weights can fluctuate wildly, which makes it hard to learn efficiently. **Layer normalization** helps stabilize these weights, making training faster and more reliable.

Residual Connections:
Think of residual connections as shortcuts. They allow the model to "skip" layers if needed, ensuring that even deep networks (like those with 48+ layers) don't forget the original input. This prevents the **vanishing gradient problem**, where important information is lost as it moves through the layers.



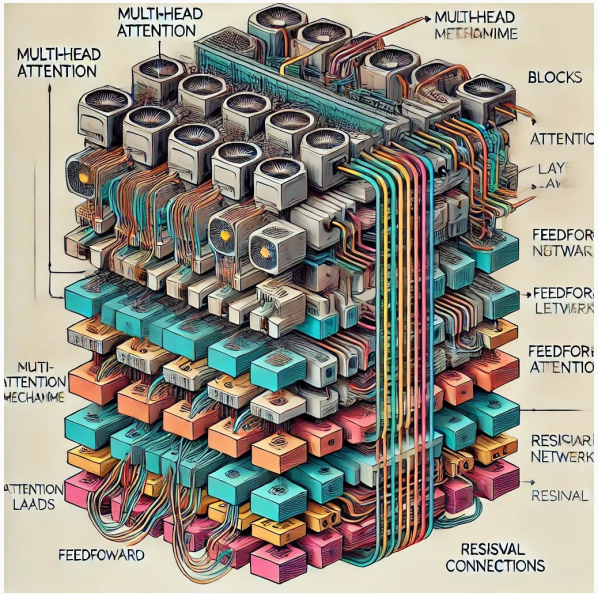
Stacking Layers for More Power

Why Stack Layers?
Each transformer block consists of attention heads and feedforward layers. These blocks are stacked on top of each other to make the model deeper and more powerful. More layers allow the model to capture more complex patterns in the text.

How Many Layers?

- **BERT Base**: 12 layers (transformer blocks)
- **GPT-3**: 96 layers
- **LLaMA-65B**: 80 layers

Impact of Depth:
More layers = more learning power, but also more computation. Deep models like GPT-3 with 96 layers can understand extremely complex language patterns but require immense computational resources.



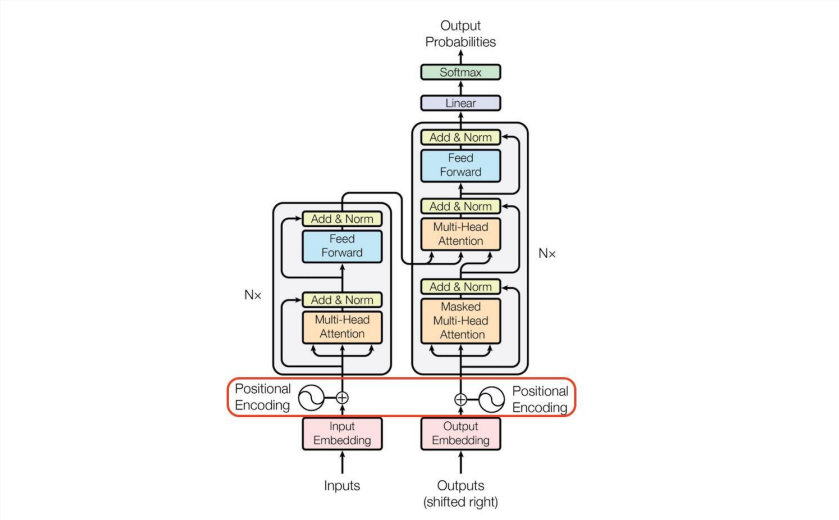
Note that this is AI generated

Positional Encoding - knowing where the words are

Why Positional Encoding?
Unlike older models, transformers don't process text one word at a time. This is more efficient, but it creates a problem: how does the model know the order of the words? For example, "The cat sat on the mat" has a different meaning from "On the mat sat the cat."

Positional Encoding solves this by adding a unique number to each word based on its position in the sentence. This helps the model understand the order of words.

How Does It Work?
Each word gets a unique vector that encodes its position in the sentence. These vectors are added to the word embeddings, so the model knows which words come first, second, etc.



Putting It All Together

A Transformer is like a Factory
Each layer is a machine that takes the input, processes it, and passes it to the next machine (layer). Attention heads are like workers, each focusing on different parts of the task, while the feedforward network refines the work before passing it along.

Optimizing Efficiency:
The number of layers, heads, and size of each hidden layer are key design choices that determine how fast and well the model can process language.

Attention Is All You Need!