📘 Understanding How LLMs Work: The Case of ChatGPT(24-07-25)

1️⃣What is an LLM?

LLM stands for **Large Language Model**—a type of deep learning model trained on massive amounts of text data to understand, generate, and manipulate human language. LLMs are built on transformer-based architectures and are designed to perform a wide range of Natural Language Processing (NLP) tasks such as:

- Text generation
- Summarization
- Translation
- Sentiment analysis
- Question answering

2️⃣Example: How ChatGPT Works

ChatGPT is a powerful example of an LLM. It stands for:

Chat + **G**enerative **P**re-trained **T**ransformer.

It performs text-based conversations with users using deep learning and natural language understanding. ChatGPT has been trained on diverse internet text but does not know facts about events occurring after its knowledge cutoff.

3️⃣Foundation: The Transformer Architecture

The core of most LLMs—including ChatGPT—is the **Transformer** architecture, introduced in the landmark 2017 paper "Attention is All You Need" by Vaswani et al.

⚙️ Why Transformers?

Unlike earlier models (like RNNs and LSTMs), Transformers allow for:

- Parallel processing of input data
- Better context understanding via attention mechanisms
- More efficient training on large datasets

🔄 Workflow: How a Transformer-based LLM Processes Input

Below is the step-by-step process ChatGPT or similar LLMs follow to generate responses:

🪜 Step 1: Tokenization (Encoding)

When a user inputs text, the first step is to **tokenize** it—splitting the text into smaller units (called **tokens**), which can be words, subwords, or characters.

For example:
`"ChatGPT is amazing"` → `["Chat", "G", "PT", "is", "amazing"]`

Different models use different tokenizers (e.g., Byte Pair Encoding or WordPiece). ChatGPT uses a tokenizer based on OpenAI's Tiktoken system.

🧠 Step 2: Vector Embedding

Each token is converted into a numerical vector through **embedding**—a learned representation in multi-dimensional space that captures semantic meaning.

> 📌 **Embeddings** help the model understand similarity between tokens. For example, "king" and "queen" may have vectors close to each other.

🔗 Learn more: OpenAI Embedding Models

📍 Step 3: Positional Encoding

Since Transformers do not inherently understand the order of tokens (unlike RNNs), **positional encoding** is added to the embeddings to indicate each token's position in the sequence.

These encodings use mathematical functions like sine and cosine to create distinguishable patterns for each position.

🧲 Step 4: Multi-Head Self-Attention

This is the heart of the Transformer. The **self-attention mechanism** allows the model to weigh the importance of each word in relation to every other word in the input.

> 🔍 For example, in the sentence "The cat chased the mouse," the model learns that "cat" is closely related to "chased" and "mouse".

🧠 **Multi-Head Attention**
Instead of computing attention once, **multi-head attention** runs multiple parallel attention operations (called "heads") to capture different types of relationships in the sentence. This improves the model's understanding of:

- Syntax
- Semantics
- Long-range dependencies

🔄 Step 5: Add & Norm

After the attention mechanism, the output is added to the original input (residual connection) and then passed through **layer normalization**.

This helps:

- Prevent vanishing gradients
- Stabilize and accelerate training

🏗️ Step 6: Feed Forward Neural Network

The normalized data is passed through a fully connected neural network (feed-forward layers) applied independently to each token. It helps in transforming the features learned by attention.

🔄 Step 7: Add & Norm Again

Another residual connection and layer normalization follow to ensure consistent gradients and stable training.

🧱 Step 8: (In Decoder) Masked Multi-Head Attention

In models like ChatGPT that generate text one word at a time (auto-regressively), the decoder uses **masked attention** to prevent the model from seeing future tokens.

> This ensures that predictions are made one token at a time, based only on what has already been generated.

📐 Step 9: Linear + Softmax Layer

The output from the final Transformer block is passed through a **linear layer** that maps the output to vocabulary size. Then a **softmax function** is applied to predict the probability of each possible next token.

📤 Step 10: Output Generation

The token with the highest probability is selected and appended to the output sequence. The process repeats until a stopping condition is met (e.g., reaching maximum length or end-of-sequence token).

🔄 Summary of LLM Workflow

mathematica
CopyEdit

```
User Input
   ↓
Tokenization
   ↓
Embedding + Positional Encoding
   ↓
Multi-Head Attention
   ↓
Add & Norm
   ↓
Feed Forward Network
   ↓
Add & Norm
   ↓
Linear → Softmax
   ↓
Next Token Prediction
   ↓
Repeat until complete response
```

🧪 Training of LLMs

ChatGPT and similar LLMs undergo two major phases:

1. Pretraining

- Trained on massive amounts of publicly available text data (books, websites, articles)
- Objective: Predict the next word/token in a sentence
- Example: "The cat sat on the ___" → "mat"

2. Fine-tuning (for ChatGPT)

- Human feedback is used to guide and shape the model's behavior
- Reinforcement Learning with Human Feedback (RLHF) is commonly used
- The model learns how to provide more helpful, safe, and factual responses

⚠️ Challenges in LLMs

- **Hallucination**: Model may generate confident but incorrect answers
- **Bias**: Reflects biases present in the training data
- **Compute Intensive**: Requires large-scale hardware and infrastructure
- **Interpretability**: Difficult to explain why the model made a specific prediction

✅ Use Cases of LLMs

- AI chatbots (like ChatGPT)
- Code generation (e.g., GitHub Copilot)
- Legal document summarization
- Customer service automation
- Creative writing and content ideation

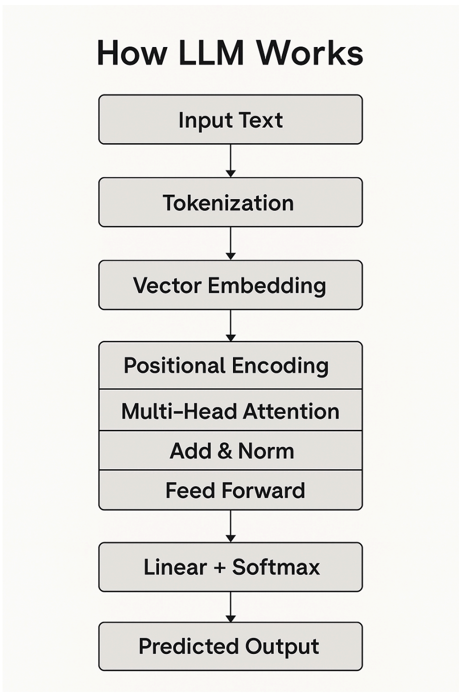
🔚 Conclusion

Large Language Models like ChatGPT are revolutionizing how humans interact with machines through natural language. Built upon the transformer architecture, they combine deep statistical analysis, massive-scale data, and complex neural computation to understand and generate human-like text.

Understanding the internal mechanics—from tokenization to attention to output prediction—helps us appreciate the sophistication behind simple conversational interfaces like ChatGPT.

📚 Recommended Resources

- 🔗 OpenAI Embeddings Guide
- 🔗 Illustrated Transformer (Jay Alammar)
- 🔗 HuggingFace Transformers Docs
- 🔗 Attention is All You Need Paper


How LLM Works
Input Text
Tokenization
Vector Embedding
Positional Encoding
Multi-Head Attention
Add & Norm
Feed Forward
Linear + Softmax
Predicted Output