**🧠 LLM Architecture: A Deep Dive**

**🔧 1. Foundational Model: Transformer**

At the heart of nearly every LLM is the **Transformer architecture** (Vaswani et al., 2017).

**🔄 Key Mechanism: Self-Attention**

Instead of reading sequences one token at a time (like RNNs), Transformers use **self-attention** to process the entire input in parallel and let each token "attend to" others.

**⚙️ Components:**

| **Component** | **Function** |
| --- | --- |
| **Embedding layer** | Converts tokens into vectors |
| **Positional encoding** | Adds word order info |
| **Attention blocks** | Compute relationships between all tokens |
| **Feedforward layers** | Apply non-linear transformations |
| **Layer norm + residuals** | Stabilize and preserve gradient flow |
| **Output head** | Maps final hidden state to vocabulary logits |

**🧠 2. Decoder-Only Transformers**

LLMs like GPT use **decoder-only architectures**, meaning:

* Input is passed left-to-right (autoregressive)
* Output token t is generated using only tokens before it

Models like **GPT-3/4**, **LLaMA**, and **Mistral** use this setup.

Contrast this with:

* **Encoder-only** (like BERT): good for classification, not generation
* **Encoder-decoder** (like T5): good for translation or summarization

**📏 3. Scaling Laws and Capacity**

LLMs are built big — really big.

**🔢 Scaling Dimensions:**

| **Factor** | **Examples** |
| --- | --- |
| **Parameters** | GPT-2: 1.5B → GPT-3: 175B → GPT-4: ~>1T (estimate) |
| **Training data** | Common Crawl, books, code, etc. (hundreds of billions of tokens) |
| **Compute** | FLOPs scale with model size and sequence length |

Kaplan et al. (2020) showed that loss scales predictably with model, data, and compute size.

**🏗️ 4. Training Pipeline**

**🛠️ Pretraining (unsupervised)**

* Objective: **next token prediction**
* Data: massive corpus of web, books, code, etc.
* Loss: cross-entropy between predicted and actual tokens

**🎯 Fine-tuning (optional)**

* Smaller supervised datasets (e.g., legal QA, medical summaries)

**👍 RLHF (Reinforcement Learning from Human Feedback)**

* Human raters rank outputs
* Reward model trained to mimic human preferences
* Final model trained via **Proximal Policy Optimization (PPO)**

**⚡ 5. Attention Mechanics (Under the Hood)**

**Self-Attention Equation:**

For input tokens converted to vectors:

plaintext

CopyEdit

Attention(Q, K, V) = softmax(QKᵀ / √d\_k) · V

* Q = Queries
* K = Keys
* V = Values
* d\_k = dimension size

Each token’s output is a **weighted sum** of all other tokens, where weights depend on similarity (dot product).

**Multi-Head Attention:**

* Multiple attention "heads" let the model focus on different relationships.
* Outputs are concatenated and linearly transformed.

**📐 6. Architectural Enhancements in Modern LLMs**

| **Technique** | **Purpose** |
| --- | --- |
| **Rotary positional embeddings (RoPE)** | Better extrapolation to long contexts |
| **Mixture of Experts (MoE)** | Route parts of the model, reducing compute per token |
| **FlashAttention** | More memory-efficient attention |
| **ALiBi / Linear Attention** | Reduce quadratic cost of attention |
| **LoRA / QLoRA** | Lightweight fine-tuning using low-rank adapters |
| **SwiGLU / GEGLU** | Enhanced activation functions for FFNs |

**🧩 7. Tokenizer & Vocabulary**

* LLMs use **Byte Pair Encoding (BPE)** or **SentencePiece** to break text into subword tokens.
* Common vocab size: ~32,000–100,000 tokens.
* Tokenization affects **compression**, **speed**, and **capability**.

**🛠️ 8. Inference and Deployment**

**Generation Strategies:**

| **Method** | **Description** |
| --- | --- |
| **Greedy** | Always pick highest-probability token |
| **Beam search** | Explore multiple sequences at once |
| **Top-k / Top-p sampling** | Introduce randomness for creativity |
| **Temperature** | Controls entropy; lower = more focused |

**Optimizations:**

* **KV Caching**: reuse attention keys/values across tokens
* **Quantization**: reduce memory by storing weights in 4/8-bit formats
* **Distributed inference**: use model parallelism or tensor parallelism for serving large models

**🧠 Summary Diagram**

Here’s how the components connect:

plaintext

CopyEdit

[Input Text]

↓

[Tokenizer → Tokens]

↓

[Embedding + Positional Encoding]

↓

[Transformer Blocks x N]

↓

[Final Hidden State]

↓

[LM Head → Vocabulary Distribution]

↓

[Output Token]