Transformers — The Foundation of LLMs

1. Why Transformers?

Before Transformers, we had:

- **RNNs / LSTMs** → sequential, slow, hard for long text.
- **CNNs** → good for local patterns, but not great for global context.
- Transformers solved this by introducing **Self-Attention**, letting models see *all words at once* and figure out which ones matter most.

2. Transformer Architecture (High-Level):

Two main blocks:

1. **Encoder** (like BERT) → understands context.
2. **Decoder** (like GPT) → generates outputs.
3. **Full Transformer** = Encoder + Decoder (used in machine translation, T5, etc.).

3. Core Component — Attention:

Imagine sentence:
"The dog chased the ball because it was rolling fast."

Question: What does "it" refer to?

- Attention helps model learn that "it" → "ball", not "dog".

4. Self-Attention Equation:

Attention(Q,K,V)=softmax(QKTdk)VAttention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) VAttention(Q,K,V)=softmax(dkQKT)V

- **Q (Query)** = what we're looking for.
- **K (Key)** = what each word offers.
- **V (Value)** = representation of each word.
- Dot product QK^T = similarity between words.
- Softmax → weights (importance).
- Multiply with V = weighted sum = contextualized word.

#This lets words "pay attention" to the right other words.

4. Multi-Head Attention:

- One attention head = one perspective.
- Multiple heads = multiple perspectives (syntax, semantics, long-range, etc.).

- Outputs are concatenated → richer understanding.

5. Positional Encoding:

Transformers don't process text sequentially → need **positional encoding** to know word order.

6. Encoder vs Decoder:

- **Encoder (BERT)**:
  - Input: sequence of words.
  - Output: contextual embeddings.
  - Use-case: classification, Q&A, embeddings.
- **Decoder (GPT)**:
  - Input: past tokens.
  - Output: next token.
  - Use-case: text generation, chatbots.
- **Encoder-Decoder (T5, BART)**:
  - Input: sequence (encoder).
  - Output: sequence (decoder).
  - Use-case: translation, summarization.

7. Why Transformers Scaled into LLMs:

- Parallelizable (unlike RNNs).
- Trained on **huge datasets** (trillions of tokens).
- Scaling laws: more data + more parameters → better performance.
- Foundation for **GPT, BERT, T5, LLaMA, Falcon, etc.**

8. Code Example — Tiny Transformer with Hugging Face:

Here's a small demo for **text classification with DistilBERT** (lightweight BERT variant):

```
from transformers import pipeline

# Sentiment analysis pipeline
classifier = pipeline("sentiment-analysis")

print(classifier("The movie was absolutely wonderful, I loved it!"))
print(classifier("This product is terrible and I want a refund."))
```

Output:

[{'label': 'POSITIVE', 'score': 0.999}]

[{'label': 'NEGATIVE', 'score': 0.998}]

#Expected Output of Transformers:

- **Encoders (BERT)** → good at understanding context.
- **Decoders (GPT)** → good at generating text.
- **Full (T5, BART)** → good at translation, summarization, seq2seq tasks.

What Changes from Small Transformers → LLMs

- **Scale**:
  - BERT = 110M params.
  - GPT-3 = 175B params.
  - LLaMA 2–70B = standard open models now.
- **Training Data**:
  - LLMs are trained on trillions of tokens (web, books, code).
- **Emergent Abilities**:
  - Few-shot learning (just examples in prompt).
  - Chain-of-thought reasoning.
  - Zero-shot generalization.
- **Applications**:
  - Chatbots, assistants, code completion, summarization, search augmentation.

#Families of LLMs:

- **Encoders** (BERT, RoBERTa, DistilBERT) → used for embeddings, classification.
- **Decoders** (GPT, LLaMA, Falcon) → used for generation.
- **Encoder-Decoder hybrids** (T5, BART, FLAN) → translation, summarization.

#**How You Use LLMs in Practice:**

1. **Inference only** (like you did with Stable Diffusion prompts).
2. **Fine-tuning / instruction-tuning** (make model follow your style/task).
3. **RAG (Retrieval-Augmented Generation)** (inject external data into LLM context).
   - Example: connect GPT with a company’s private knowledge base → chatbot that answers from docs.