📌 Tokenization

Tokenization is the process of **breaking text into smaller units** called **tokens**. These tokens can be **words, subwords, or characters**, depending on the approach used.

✒️ Example

- **Word Tokenization** - ["I", "love", "AI", "!"]
- **Subword Tokenization** - ["I", "love", "A", "I", "!"] (Uses Byte Pair Encoding)
- **Character Tokenization** - ["I", " ", "l", "o", "v", "e", " ", "A", "I", "!"]
- **Sentence Tokenization** - ["I love AI!"] (Splitting paragraphs into sentences)

✒️ AI Model Tokenization Techniques

1. GPT (GPT-3, GPT-4, GPT-4 Turbo) - Byte-Pair Encoding (BPE) with GPT-2 Tokenizer
2. BERT - WordPiece Tokenization
3. RoBERTa - Byte-Pair Encoding (BPE) with SentencePiece
4. T5 - SentencePiece with Unigram Language Model
5. LLaMA - SentencePiece with Byte-Pair Encoding (BPE)
6. Gemini - SentencePiece with BPE
7. Claude - Custom BPE Tokenizer
8. Whisper - Multilingual Byte-Level BPE

✒️ Why do we take the dot product of Query Matrix and Key Matrix to get the similarity?

In the **self-attention mechanism** of **Transformer models**, we compute the dot product of the **Query (Q) and Key (K)** matrices to measure **how much attention one word should pay to another**. This helps in capturing relationships between words.

- **Higher dot product** → More similar words → More attention assigned.
- **Lower dot product** → Less relevant words → Less attention assigned.
- The dot product is **fast and scalable**, making it ideal for large-scale attention mechanisms.

- It allows **matrix multiplication** across all words at once (**parallel processing**).
- The dot product gives a raw score, which is then **scaled and passed through Softmax to convert it into probabilities**.
- This ensures that **attention weights sum to 1** and can be interpreted meaningfully.

📌 Encoder vs Decoder

Feature	Encoder	Decoder
Purpose	Converts input into a contextual representation	Converts the encoded representation into the final output
Input	Raw data (e.g., text, image, or audio)	Encoded representation from the encoder
Output	A fixed-size vector or contextual embeddings	A generated sequence (text, image, etc.)
Used In	Feature extraction, understanding input context	Generating predictions, translating output
Example in NLP	Processes a sentence in English	Generates its translation in French
Example in Autoencoders	Compresses an image into a latent vector	Reconstructs the image from the latent vector
Architecture in Transformers	Self-attention + Feedforward layers	Self-attention + Encoder-Decoder attention + Feedforward layers
Information Flow	Processes full input at once	Generates output **sequentially** (one step at a time)
Self-Attention?	Yes (focuses on relationships within the input)	Yes (but also attends to encoder output)

📌 Masking Values

Masking is a technique used in Transformers (especially in **Decoders**) to prevent models from looking at future tokens while generating output. It ensures that **each token only attends to previous tokens**, maintaining the **auto-regressive** nature of text generation.

📌 How Masking Works?

In **self-attention**, the attention scores are computed using:

$$A = \frac{QK^T}{\sqrt{d_k}}$$

After computing these scores, **a mask is applied** to ensure that a token at position t **cannot see tokens at future positions** $t+1, t+2$, **etc.**.

Example: Attention Matrix Before Masking

For a 5-word sentence, the **attention scores** might look like:

$$\begin{bmatrix} ✓ & ✓ & ✓ & ✓ & ✓ \\ ✓ & ✓ & ✓ & ✓ & ✓ \\ ✓ & ✓ & ✓ & ✓ & ✓ \\ ✓ & ✓ & ✓ & ✓ & ✓ \\ ✓ & ✓ & ✓ & ✓ & ✓ \end{bmatrix}$$

- **Problem:** Every word sees all other words (including future words).

Example: Attention Matrix After Masking

After applying the mask, **future tokens are blocked (set to -∞ before softmax, resulting in 0 attention weights)**:

$$\begin{bmatrix} ✓ & 0 & 0 & 0 & 0 \\ ✓ & ✓ & 0 & 0 & 0 \\ ✓ & ✓ & ✓ & 0 & 0 \\ ✓ & ✓ & ✓ & ✓ & 0 \\ ✓ & ✓ & ✓ & ✓ & ✓ \end{bmatrix}$$

- **Now, each token can only see previous and current tokens, maintaining the autoregressive nature.**

📌 Parameters in Different AI Models

Model	Architecture	Parameters	Layers	Training Data Size	Context Window Size	Notable Features
GPT-3	Transformer Decoder	175 billion	96	570 GB (CommonCrawl), plus WebText2, Books, and Wikipedia	2,048 tokens	Excels in text generation and completion tasks.
BERT	Transformer Encoder	110 million (Base)	12	16 GB (BooksCorpus and English Wikipedia)	512 tokens	Bidirectional context understanding; strong in NER and QA tasks.
		340 million (Large)	24			
Llama 3	Transformer Decoder	405 billion	126	15 TB (various sources)	128,000 tokens	Advanced capabilities with extended context understanding.
Claude 2.1	Transformer	Not publicly disclosed	Not publicly disclosed	Not publicly disclosed	Up to 200,000 tokens	Enhanced long-context processing.
GPT-4 Turbo	Transformer Decoder	Not publicly disclosed	Not publicly disclosed	Not publicly disclosed	4,096 tokens	Optimized for faster inference and cost-effectiveness.
RoBERTa	Transformer Encoder	125 million (Base)	12	160 GB (diverse internet data)	512 tokens	Robust performance across various NLP tasks.
		355 million (Large)	24			
LLaMA	Transformer Decoder	7 billion to 65 billion	Varies	1.4 TB (various sources)	Varies	Open-source model with competitive performance.
GPT-Neo	Transformer Decoder	2.7 billion	32	825 GB (The Pile dataset)	2,048 tokens	Open-source alternative to GPT-3.
T5	Transformer Encoder-Decoder	220 million (Base)	12	745 GB (C4 dataset)	512 tokens	Unified text-to-text framework; versatile across NLP tasks.
		11 billion (Large)	24			