Tokenization In LLM

Tokenization is the initial phase of interacting with LLMs. It involves breaking the input text into smaller pieces known as tokens.

Considered the following sentence,

"The child's coloring the book."

If the tokenization splits the text after every white space characters. The result will be:

["The", "child's", "coloring", "the", "box"]

In this approach, the punctuation remains attached to the words like "child's" and "book".

Alternatively, tokenization can be done by seperating text based on both white spaces and punctuation the output would be:

["The", "child", "'", "s", "coloring", "book", "."]

The tokenization process is model dependent. The models are released as a pair of pre-trained tokenizers as associated model weights.

Main key points to remember is :

- Tokenization is the first step when interacting with a large language Model (LLM).
- It breaks down text into smaller units called tokens — those can be whole words, sub-words or even characters.
- Example: "unhappiness" might be split into tokens — "un" + "happy" + "ness.".

Why is Tokenization Important ?

Computers don't understant raw text. They need numbers to work with.

- Tokenization allows text to be mapped to numerical IDs, creating a dictionary of all unique tokens.
- Once text is tokenized, it can be transformed into embeddings (vectors of numbers) that capture meaning and relationship.

In the Context of LLMs

- Every Input sentence is scanned and broken into tokens.
- Each token is indexed (assigned an ID).
- This forms the stucture input that passes into embeddings — transformer layers final output.

Methods of Tokenization

- Word-level : Each word = one token.
  (Problem : large vocabulary, stuggles with unknown word)
- Character-level : Each character = one token
- Subword-level (used in modern LLMs).
  Breaks words into frequently occurring units using algorithms like
  → Byte Pair Encoding (BPE)
  → WordPiece
  → Sentence Piece / Unigram LM

Numerical Mapping

- After tokenization, each token is assigned a unique integer ID from the models vocabulary.

Example:

"happy" ⟶ 5024

"ness" ⟶ 841

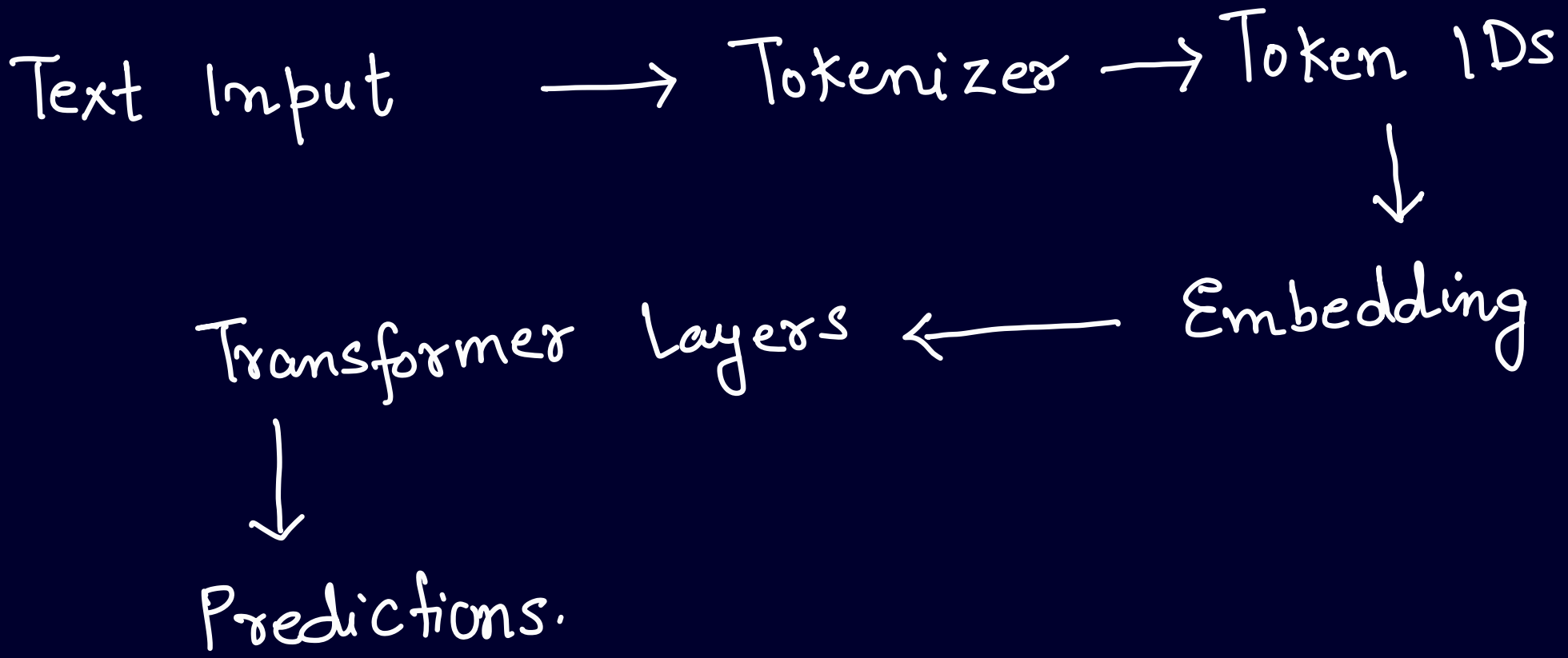
Impact on Context Size

- Transformers process tokens in fixed length windows (eg 2k, 8k, 128k tokens)
- Longer tokenized sequences require more memory and compute.

Why subword Tokenization Works well ?

- Efficiently handles rare woods by breaking them down.
- Keeps the vocabulary size manageable (~ 30K - 50K tokens)

Pipeline


Text Input
Tokenizer
Token IDs
Embedding
Transformer Layers
Predictions.