Lecture 7: Code an LLM Tokenizer from Scratch in Python

1. Introduction to Tokenization

Tokenization is the first step in the data pre-processing pipeline for building Large Language Models (LLMs).

- **The Goal:** Prepare raw text input for training. LLMs cannot process raw text directly; they require numerical input.
- **The Workflow:** The process consists of three distinct steps:
    1. **Splitting:** Breaking the input text into individual words or sub-words (tokens).
    2. **Token IDs:** Converting these text tokens into unique integer numbers (Token IDs).
    3. **Vector Embeddings:** Converting Token IDs into vector representations (covered in a later lecture).

2. Step 1: Splitting Text (Creating Tokens)

The lecture demonstrates how to split a text dataset (using Edith Wharton's short story *"The Verdict"* as a sample) using Python.

- **Tools Used:** The Python `re` (Regular Expression) library is used to split text based on specific patterns.
- **Evolution of the Split Logic:**
    - **Attempt 1 (Whitespace):** Initially, text is split only on whitespace (`\s`). This is flawed because punctuation (like commas) remains attached to words (e.g., "Hello," becomes one token).
    - **Attempt 2 (Punctuation):** The splitter is refined to separate characters like commas and periods. This ensures "Hello" and "," are separate tokens.
    - **Attempt 3 (Complex Punctuation):** The final robust splitter handles a wide variety of characters including colons, semicolons, question marks, dashes, and quotation marks.
- **Handling Whitespace:**
    - The lecture code removes whitespace from the token list to save memory and compute requirements.
    - *Note:* In some contexts (like training an LLM for Python code), keeping whitespace is essential because indentation matters, but for general text, it is often removed.

3. Step 2: Converting Tokens to IDs (Vocabulary Creation)

Once the text is split, the model needs a way to convert these strings into numbers.

- **Vocabulary Definition:** A list of all **unique** tokens found in the training data, sorted alphabetically.
- **Mapping:** Each unique token is assigned a unique integer (Token ID).

    - *Example:* If "Brown" is first alphabetically, it gets ID `0`. "Dog" might get ID `1`.
- **The Tokenizer Class:** Dr. Dander implements a Python class with two essential methods:
    1. **Encode:** Takes raw text, splits it into tokens, and converts them to Token IDs using the vocabulary.
    2. **Decode:** Takes Token IDs, converts them back into text strings, and joins them (handling spacing around punctuation) to reconstruct the original message.

4. The Limitation: Out-of-Vocabulary (OOV) Errors

A major issue arises when the tokenizer encounters a word that was not present in the training dataset.

- **The Scenario:** If the model was trained on *"The Verdict"* and you try to encode the word "Hello" (which isn't in the book), the simple tokenizer will crash because "Hello" has no assigned Token ID.
- **The Implication:** This highlights the need for massive, diverse training datasets to maximize vocabulary coverage.

5. The Solution: Special Context Tokens

To handle OOV errors and structure multiple documents, "Special Tokens" are added to the vocabulary.

- **The `<|unk|>` Token:**
    - Stands for "Unknown."
    - If the tokenizer encounters a word not in the vocabulary, it replaces it with the `<|unk|>` token rather than crashing.
- **The `<|endoftext|>` Token:**
    - Used to separate unrelated data sources.
    - If training on multiple books or articles, this token signals to the LLM where one document ends and a completely unrelated one begins.
- **Other Common Special Tokens:**
    - `BOS` (Beginning of Sequence) and `EOS` (End of Sequence).
    - `PAD` (Padding): Used to make batch sizes uniform.

6. GPT and Byte Pair Encoding (BPE)

The lecture concludes by noting that modern models like GPT-2 and GPT-3 do **not** actually use the `<|unk|>` token.

- **Why?** They use a more advanced tokenization scheme called **Byte Pair Encoding (BPE)**.
- **How it works:** Instead of mapping whole words, BPE breaks words down into sub-words or even individual characters.

- **Result:** Because it can always break a word down into characters, it never encounters a truly "unknown" token, eliminating the need for `<|unk|>`. This is the subject of the next lecture.