Name: Patel Fenil Dipakbhai
Student ID: 9001279
Final project

Tokenizer Module — Description

1) What the tokenizer is and why it matters here

In our system, the tokenizer is the very first step that converts a user's raw prompt into a form the rest of the pipeline can understand. It splits text into tokens—mostly words and punctuation—and from these tokens we compute simple but informative linguistic features. Those features roll up into a single complexity score, which is then used by:

1. the regression model to predict Energy (kWh) and CO_2 (kg) for a given prompt and model settings, and
2. The Isolation Forest to flag anomalies (i.e., prompts that look unusually "expensive" for their complexity).

In short: No tokenization → no complexity score → no energy/CO_2 estimate or anomaly alert. Tokenization is what turns a string of text into numbers we can analyse, visualize, and optimize.

2) Role in the overall architecture

The tokenizer sits inside the NLP module and exposes a clean, simple interface to the rest of the app:

- The Streamlit UI collects the prompt and model knobs (Layers, FLOPs, Hours).
- The UI calls the complexity function (which internally tokenizes and computes features).
- The resulting complexity score is appended to the feature vector [layers, flops_tflops, hours, complexity].
- That vector is sent to:
  - the predictor (to get energy_kWh and co2_kg), and
  - the anomaly service (to get is_anomaly and a score).

This design keeps the tokenizer modular (one place to maintain) and fast (a few milliseconds per prompt), which is critical for a real-time dashboard.

3) How the tokenizer works

A tokenizer converts raw text into smaller units called tokens, which are easier for a model to process. First, it applies rules or learned patterns to split text into words, subwords, or characters. For example, “tokenization” might become “token” and “ization.” Modern tokenizers often use algorithms like Byte Pair Encoding (BPE) or WordPiece to handle rare words by breaking them into frequent subword units. This allows the model to work with a fixed vocabulary while still representing almost any word. The tokenizer also maps each token to a numerical ID, producing the sequence of numbers that the model uses for training or inference.

Tokenization → Complexity → Energy/CO_2 Predictions

“Our tokenizer converts the raw prompt into tokens, then computes simple features—token count, word variety (TTR), average sentence length, and readability—to produce one complexity score. That score, combined with model settings (Layers, FLOPs, Hours), feeds our regression to estimate Energy (kWh) and CO_2 (kg), and also feeds isolation forest to flag inefficient prompts. It’s fast, transparent, and central to our optimization loop: we show the predicted impact, offer a rewritten prompt, and then show the expected energy/CO_2 savings—all grounded in the tokenizer’s measurements. The code lives in src/nlp/parser.py and src/nlp/complexity_score.py.”