Automated Code Documentation Generator

Group 12

Jatin Singh (220347) Suryansh Sharma (220350)

Priyanshu Garg (220649) Sahil Merothia (220653)

Problem Statement

Documentation for software projects tends to be outdated, poor, or completely absent, particularly for legacy or open source code. Creating documentation manually takes a lot of time, is inconsistent, and can be prone to human error.

Our goal is to create an NLP-based tool that will automatically generate external documentation for Python codebases, even if there are no docstrings or comments, using a pipeline of core NLP techniques.

Project Objectives

- Parse and extract code structure (functions, classes, arguments)
- Use classical NLP techniques (Tokenization, Lemmatization, TF-IDF, BoW, POS tagging).
- Infer function/class purpose utilizing NLP patterns/templates.
- Produce human-readable documentation in markdown format.
- Keep the solution explainable, reproducible, and dependence free on LLMs.

Step-by-Step Methodology

Step 1: Data Collection

Input: Raw Python code files (.py)
Source: User uploads or loads files from a directory

Step 2: Code Parsing

Python's AST (Abstract Syntax Tree) to extract:
- Function and class definitions
- Arguments
- Docstrings (if any)
- Line numbers for mapping

Using tokenize module to extract inline comments (# ...)

Step 3: NLP Processing Modules

Tokenization & Lemmatization:
- Split function names (getUserData → ['get', 'user', 'data'])
- Lemmatize tokens for consistency

Keyword Extraction:
- Use TF-IDF and BoW to extract important words from comments/docstrings

Sentence Summarization:
- Use TextRank or TF-IDF sentence scoring to summarize comments

POS Tagging & Template Generation:
- Use POS patterns to understand the role of tokens (verbs/nouns)
- Generate description templates like: “This function [verb] [noun] using [arguments].”

Action Inference from Code:
- Walk AST to extract logic (e.g., append, return, math operations)
- Infer likely action: “Performs a calculation,” “Returns a list,” etc.

Step 4: Documentation Generation

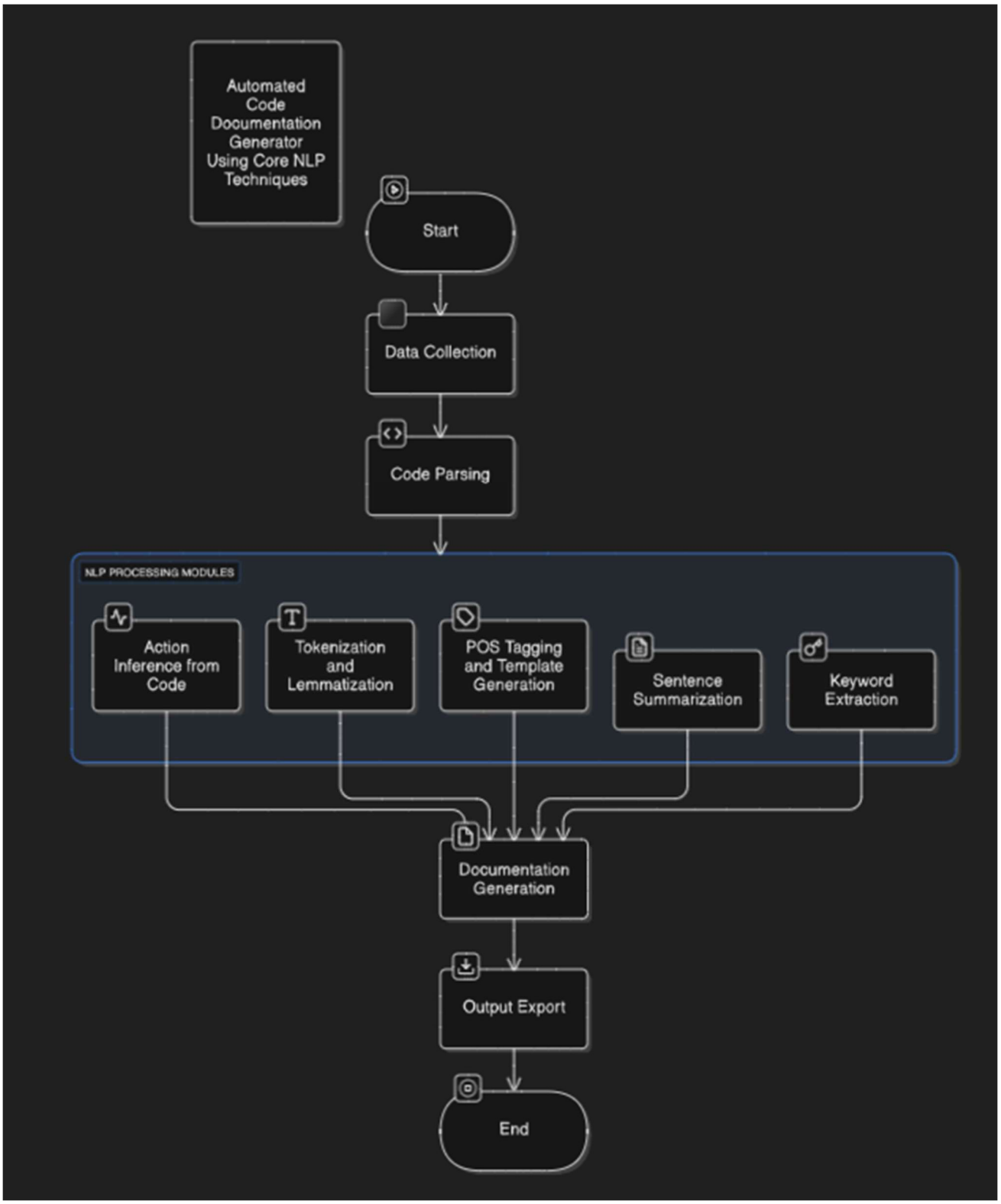
Using Jinja2 templates for Markdown formatting

Output documentation structure includes:
- Title
- File-level overview
- Class & method sections
- Function sections
- Arguments, comments, inferred descriptions

Step 5: Output Export

Documentation saved in .md, .html, or .pdf
Output directory: data/output_docs/

System Diagram (High-Level Workflow)



Architecture Diagram (Technical Design)

```
Main Script (main.py)
    │
    ├──> Code Parser (AST/tokenize)
    │      └──> code_parser.py
    │
    ├──> NLP Modules
    │     ├── tokenizer.py
    │     ├── summarizer.py
    │     ├── keyword_extractor.py
    │     └── pos_tagger.py
    │
    ├──> Doc Writer (Markdown Engine)
    │      └──> doc_writer.py + templates/
    │
    └──> Output: /data/output_docs/
```

Tools and Libraries Used

Module	Tool/Library
Parsing	ast, tokenize
NLP	nltk, sklearn
Template Engine	Jinja2
Output Format	Markdown, HTML, PDF

Summary

This approach entails a comprehensive, scalable NLP-powered method for automatic documentation generation with Python code. By applying NLP processing techniques such as TF-IDF, tokenization, and summarization to static code parsing, the approach generates informative documentation, and it does so without the use of contemporary LLMs. Thus, an automatic documentation generator based on this approach is useful in education and production environments.