SMARTSDLC Documentation

Overview

SMARTSDLC is an AI-powered software development lifecycle assistant built using Python, Gradio, PyTorch, and the Transformers library. It provides two main functionalities: Requirements Analysis and Code Generation. The application allows users to upload a PDF document or input text requirements, which are then analyzed to extract functional, non-functional, and technical requirements. Additionally, it can generate code in various programming languages based on user-specified requirements.

This documentation provides an overview of the code structure, functionality, setup instructions, and usage guidelines.

Table of Contents

Features

Requirements Analysis: Extracts and organizes software requirements from a PDF file or text input into:

Functional Requirements
Non-Functional Requirements
Technical Specifications

Code Generation: Generates code in multiple programming languages (Python, JavaScript, Java, C++, C#, PHP, Go, Rust) based on user-provided requirements.

User Interface: Built with Gradio, providing an intuitive web-based interface with two tabs:

Code Analysis: For uploading PDFs or entering requirements text.

Code Generation: For specifying code requirements and selecting a programming language.

AI Model: Powered by the ibm-granite/granite-3.2-2b-instruct model for natural language processing and code generation.

Dependencies

The following Python libraries are required to run the application:

gradio: For creating the web-based user interface.
torch: For handling the AI model and tensor computations.
transformers: For loading and using the pre-trained language model.
PyPDF2: For extracting text from PDF files.
io: For handling file streams.

These dependencies can be installed using the following commands:
pip install gradio torch transformers
pip install transformers torch gradio PyPDF2 -q

Code Structure
The code is structured as follows:

Imports:

Imports necessary libraries (gradio, torch, transformers, PyPDF2, io).

Model Setup:

Loads the ibm-granite/granite-3.2-2b-instruct model and tokenizer.
Configures the model to use GPU (if available) with torch.float16 precision or CPU with torch.float32.
Sets the padding token to the end-of-sequence token if not defined.

Core Functions:

generate_response(prompt, max_length): Generates a response using the AI model for a given prompt.
extract_text_from_pdf(pdf_file): Extracts text from an uploaded PDF file.
requirement_analysis(pdf_file, prompt_text): Analyzes requirements from a PDF or text input and organizes them.
code_generation(prompt, language): Generates code in the specified programming language based on user requirements.

Gradio Interface:

Creates a web interface with two tabs: "Code Analysis" and "Code Generation".
The interface includes file upload, text input, dropdowns, buttons, and output text boxes for user interaction.

Functionality
Requirements Analysis

Input: Users can upload a PDF file containing software requirements or input requirements as text.
Processing:
If a PDF is uploaded, the extract_text_from_pdf function extracts text using PyPDF2.
The extracted text or user-provided text is passed to the requirement_analysis function.
A prompt is constructed to instruct the AI model to organize requirements into:

Functional Requirements: Features and functionalities the software must provide.
Non-Functional Requirements: Performance, usability, and other quality attributes.
Technical Specifications: Technical constraints or requirements.

Output: The AI model generates a structured response with the categorized requirements.

Code Generation

Input: Users provide a description of the desired code and select a programming language from a dropdown menu.
Processing:
The code_generation function constructs a prompt specifying the programming language and requirements.
The AI model generates code based on the prompt.

Output: The generated code is displayed in a text box.

Setup Instructions

Install Dependencies:Run the following commands in your Python environment:
pip install gradio torch transformers
pip install transformers torch gradio PyPDF2 -q

Run the Code:

Ensure you have a compatible Python environment (Python 3.7 or higher recommended).
Copy the provided code into a .py file or a Jupyter Notebook (e.g., SMARTSDLC.ipynb).
Execute the script or notebook. The Gradio interface will launch a web server.

Access the Interface:

If running locally, open the provided URL (e.g., http://127.0.0.1:7860) in a web browser.
If running in a shared environment (e.g., Google Colab), use the provided public URL.

Hardware Requirements:

GPU (Optional): For faster model inference, a CUDA-compatible GPU is recommended.
CPU: The application works on CPU but may be slower for large inputs.
RAM: At least 8GB of RAM is recommended for smooth operation.

Usage

Requirements Analysis:

Navigate to the "Code Analysis" tab.
Upload a PDF file containing software requirements or enter requirements in the text box.
Click the "Analyze" button to view the categorized requirements in the output box.

Code Generation:

Navigate to the "Code Generation" tab.
Enter the code requirements in the text box (e.g., "Create a function to calculate the factorial of a number").
Select the desired programming language from the dropdown (e.g., Python, JavaScript).
Click the "Generate Code" button to view the generated code.

Example Inputs:

Requirements Analysis:
PDF: Upload a document with requirements like "The system must allow user login with email and password."
Text: "The software should support user authentication, have a response time under 2 seconds, and use a MySQL database."

Code Generation:
Prompt: "Write a function to sort an array of integers in ascending order."
Language: Python
Expected Output: A Python function implementing a sorting algorithm.

Limitations

PDF Extraction: The PyPDF2 library may struggle with poorly formatted PDFs or scanned documents without OCR.
Model Accuracy: The ibm-granite/granite-3.2-2b-instruct model may occasionally produce incomplete or incorrect outputs, especially for complex requirements or code.
Language Support: Code generation is limited to the languages listed in the dropdown. Support for other languages may require modifications.
Performance: On CPU, processing large PDFs or generating complex code may be slow.
Error Handling: Limited error handling for invalid PDFs or malformed inputs.

Future Improvements

Enhanced PDF Processing: Integrate OCR or advanced PDF parsing libraries for better text extraction.
Model Fine-Tuning: Fine-tune the AI model for specific software engineering tasks to improve accuracy.
Additional Languages: Expand the list of supported programming languages.
Code Validation: Add functionality to validate and test generated code.
Export Options: Allow users to download analysis results or generated code as files.
Improved Error Handling: Implement robust error handling for invalid inputs or processing errors.