SmartSDLC – AI-Enhanced Software Development Lifecycle

Project Documentation

1. Introduction

- Project Title: AI Code Analysis & Generator Using IBM Granite
- Team Member: VARSHINI G
- Team Member: SWETHA A
- Team Member: ABINAYA J
- Team Member: DIVYA R

2. Project Overview

- Purpose:

The purpose of this application is to leverage the power of the IBM Granite large language model (LLM) to assist software developers and analysts in the early stages of the Software Development Lifecycle (SDLC). It provides two core AI-powered functionalities: automated requirement analysis and intelligent code generation. By processing natural language descriptions or technical PDF documents, the tool extracts and categorizes key software requirements. It further accelerates development by generating syntactically correct code snippets in various programming languages based on simple requirement prompts. This application aims to boost productivity, reduce manual effort, and serve as an intelligent assistant for developers.

- Features:

1. Requirement Analysis: Extracts and categorizes requirements from the modules.
2. Multi-Language Code Generation: Generates code in popular programming languages.
3. AI-Powered Insights: Leverages IBM Granite LLM for intelligent processing.

4. User-Friendly Interface: Simple, tab-based web interface built with Gradio.
5. Flexible Input Options: Accepts both direct text input and PDF file uploads for analysis.

3. Architecture

- **Frontend (Gradio)**

The frontend is developed using Gradio, which provides a simple and interactive web-based user interface. It is organized into two main tabs: "Code Analysis" and "Code Generation." Users can input requirements, upload PDF documents, select a programming language, and view the AI-generated results instantly in structured text boxes.

- **Backend (Python)**

The backend is written in Python and contains the core application logic. It handles functions such as PDF text extraction (via PyPDF2), prompt construction tailored for SDLC tasks, and managing communication with the AI model. The backend processes user requests and returns the formatted outputs.

- **LLM Integration (IBM Granite LLM)**

The application integrates with the `ibm-granite/granite-3.2-2b-instruct` model from the Hugging Face Transformers library. This layer is responsible for all natural language understanding and generation tasks. The integration is optimized to run on both CPU and GPU (if available), ensuring flexibility across different hardware environments.

- **Deployment Layer**

The deployment is handled by Gradio's built-in web server. The application can be executed locally for development and testing. Using the `share=True` parameter, it can generate a public link for easy sharing and demonstration, making it highly accessible.

4. Setup Instructions

Prerequisites:

- Python 3.8 or later
- `pip` package manager
- An internet connection to download the model and dependencies
- (Recommended) A virtual environment (e.g., `venv` or `conda`)

Installation Process:

1. Clone or create your project directory.
2. Install dependencies: Create a `requirements.txt` file with the following content:

```txt
torch>=2.0.0
transformers>=4.30.0
gradio>=4.0.0
pypdf2>=3.0.0
```

Then run: `pip install -r requirements.txt`

3. Run the application: Execute the Python script:

```bash
python smartsdlc.py
```

4. Access the application: Open the local URL (e.g., `http://127.0.0.1:7860`) or the public Gradio link provided in the terminal in your web browser.

5. Code Structure & Explanation:

- `smartsdlc.py`: The main application file containing all logic.
- Model Loading: The IBM Granite model and tokenizer are loaded from Hugging Face at startup, configured to use GPU (CUDA) if available for faster inference, otherwise it falls back to CPU.
- Key Functions:
- `generate_response(prompt, max_length)`: Core function that sends a prompt to the LLM and returns the generated text. It handles tokenization, model inference, and response decoding.

- `extract_text_from_pdf(pdf_file)`: Uses PyPDF2 to read text content from an uploaded PDF file.
- `requirement_analysis(pdf_file, prompt_text)`: Constructs a prompt to analyze software requirements from either a PDF or text input. The LLM is instructed to organize findings into functional, non-functional, and technical specifications.
- `code_generation(prompt, language)`: Constructs a prompt to generate code in a user-selected language based on the provided requirement description.

6. User Interface (Gradio Blocks)

The interface is built with Gradio's `Blocks` API for flexibility.

- Tab 1: Code Analysis
- Inputs: A file upload component for PDFs and a textbox for direct requirement input.
- Action Button: "Analyze"
- Output: A textbox displaying the structured requirement analysis.
- Tab 2: Code Generation
- Inputs: A textbox for code requirements and a dropdown menu to select the programming language (Python, JavaScript, Java, etc.)
- Action Button: "Generate Code"
- Output: A textbox displaying the generated code snippet.

7. Usage Guide

1. Launch the application (`python smartsdlc.py`).
2. For Requirement Analysis:
    - Navigate to the "Code Analysis" tab.
    - Either upload a PDF document containing software requirements or type/paste them into the textbox.
    - Click the "Analyze" button. The analyzed and categorized requirements will appear in the output box.

3. For Code Generation:

- Navigate to the "Code Generation" tab.
- Describe the functionality you want the code to perform in the textbox.
- Select your desired programming language from the dropdown menu.
- Click the "Generate Code" button. The generated code will appear in the output box.

8. Known Issues & Limitations

- **PDF Extraction Accuracy**: The text extraction from PDFs relies on PyPDF2, which may not work accurately with scanned documents, images, or complex, non-standard layouts.
- **Model Limitations**: The LLM has a maximum context window. Very large PDFs may be truncated, leading to incomplete analysis.
- **Code Correctness**: Generated code should be treated as a starting point or suggestion. It may contain errors, bugs, or not follow best practices and must be thoroughly reviewed and tested by a developer.
- **First-Load Latency**: Initial startup time is slow as the large AI model (several GBs) is downloaded and loaded into memory.
- **Hardware Demands**: Running the model, especially without a GPU, can be resource-intensive (high CPU/RAM usage) and may slow down other applications on the machine.

9. Future Enhancements

- **Enhanced PDF Processing**: Integrate OCR (e.g., Tesseract) to handle scanned documents and more advanced PDF libraries (e.g., `pdfplumber`) for better text extraction.
- **Input Chunking**: Implement processing of large documents by breaking them into chunks and analyzing them sequentially to overcome context window limits.

- **Output Formatting**: Improve the structure of the output by using markdown, bullet points, or a dedicated UI component for better readability of requirements and code.
- **Authentication & History**: Add user authentication and a database to save analysis and generation history for future reference.
- **Code Explanation**: Add a feature to explain what the generated code does, line by line.
- **API Deployment**: Refactor the backend into a FastAPI or Flask server to separate it from the UI, allowing for programmatic access via API calls.
- **Additional SDLC Features**: Expand to cover other phases like test case generation, architecture suggestion, or documentation writing.

Screenshots:

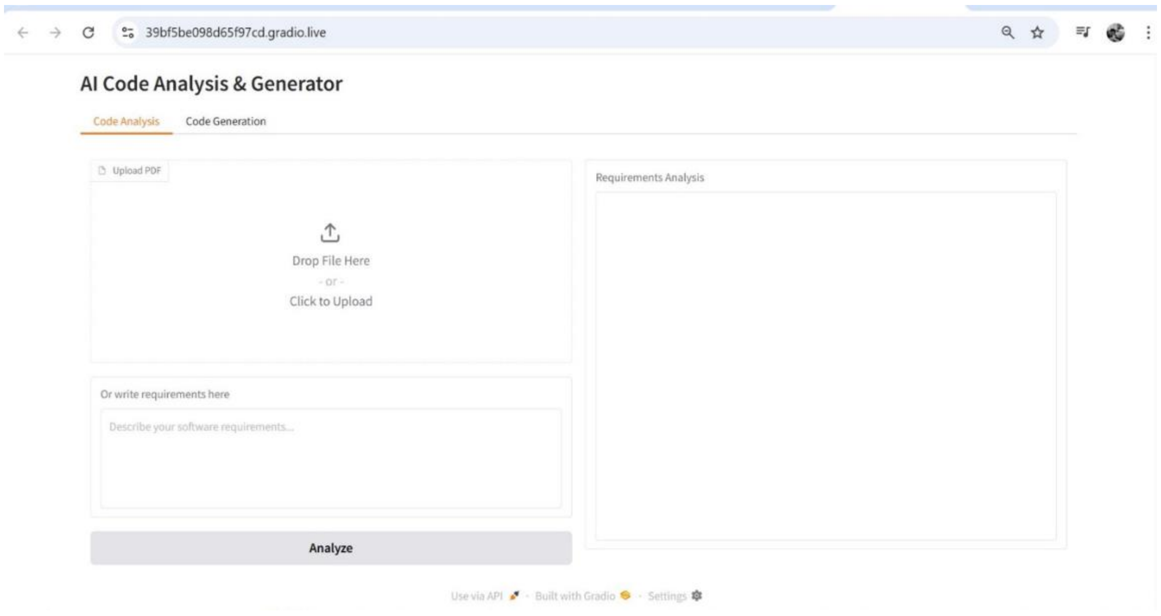
39bf5be098d65f97cd.gradio.live
AI Code Analysis & Generator
Code Analysis
Code Generation
Upload PDF
Drop File Here
- or -
Click to Upload
Requirements Analysis
Or write requirements here
Describe your software requirements...
Analyze
Use via API · Built with Gradio · Settings

Fig.1

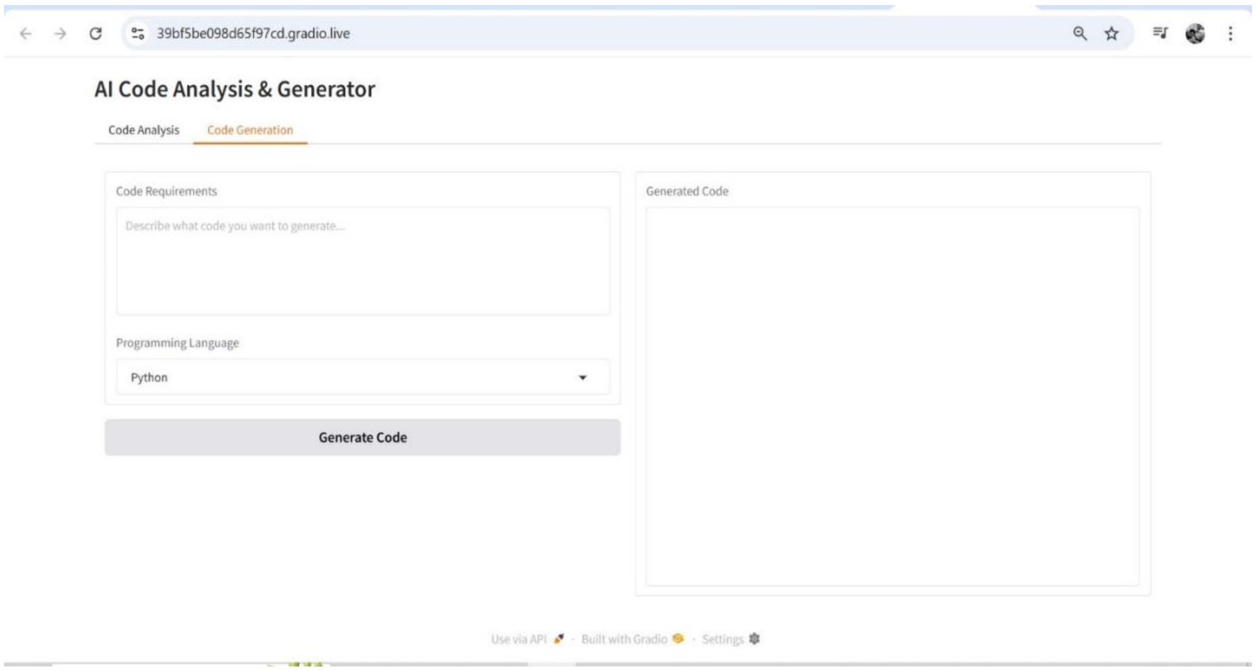
39bf5be098d65f97cd.gradio.live
AI Code Analysis & Generator
Code Analysis
Code Generation
Code Requirements
Describe what code you want to generate...
Programming Language
Python
Generate Code
Generated Code
Use via API · Built with Gradio · Settings

Fig .2