Smart-SDLC

Project Documentation

1. Introduction

Project title: Smart-SDLC AI-Enhanced Software Development Lifecycle Generative AI with IBM

Team Leader: Azhagumurugan V

Team member: Ajay S

Team member: Devaraj M

Team member: Dinesh K

2. Project Overview

Purpose:

The purpose of this application is to empower software teams to analyze requirements and generate code using large language models (LLMs) via a user-friendly Gradio interface. The tool extracts requirements from user-provided documents/text and produces organized software requirement lists or code snippets for various programming languages.

Features:

Conversational Interface: Natural language interaction for analyzing requirements and generating code.

Multimodal Input Support: Accepts both PDFs and direct text.

Requirement Analysis: Automatically organizes requirements into functional, non-functional, and technical specifications.

AI Code Generation: Generates code in Python, JavaScript, Java, C++, C#, PHP, Go, and Rust from user prompts.

Tabbed UI: Separates analysis and generation functionalities for workflow clarity.

3. Architecture

Frontend (Gradio)

Built using Gradio's Blocks API with an interactive UI, including file upload, tabbed interface, and text outputs.

Tabs:

Code Analysis: For extracting categorized requirements from text or PDF.

Code Generation: For generating language-specific code according to user input.

Backend (Transformers, PyTorch)

Loads the IBM Granite model (ibm-granite/granite-3.2-2b-instruct) using Hugging Face Transformers library.

PDF processing uses PyPDF2 to extract text from uploaded documents.

Backend routines manage prompt engineering for requirement structure and code generation.

Model inference is handled on GPU (if available) for performance.

4. Setup Instructions

Prerequisites:

Python 3.9 or later

pip and virtual environment tools

torch, transformers, gradio, PyPDF2

Installation Process:

Clone/download the project files

Install dependencies:

pip install torch gradio transformers PyPDF2

Download the IBM Granite model (automatically handled on first run)

Run the application script

Gradio will provide a local (and optionally, public) URL for access

5. Folder Structure

app.py or main script: Gradio app, UI, model loading routines

Additional utility modules may include:

granite_llm.py: Model prompt utilities (optional)

pdf_utils.py: PDF extraction helpers

6. Running the Application

Run the script (e.g. python app.py)

Access the Gradio UI at the provided address

Select "Code Analysis" to upload PDF/enter text and view organized requirements

Select "Code Generation" to provide requirement text and output AI-generated code

All interactions are real-time and processed locally or on enabled GPU.

7. API Documentation

Internal application logic includes:

requirement_analysis:

Inputs: PDF file and/or text

Output: Organized requirements (functional, non-functional, technical)

code_generation:

Inputs: Prompt text and language selection

Output: Language-specific generated code

User interactions occur via the UI, not as web API endpoints.

8. Authentication

The application runs open by default for demonstration.For secure use, Gradio's built-in authentication or network security measures are recommended.

9. User Interface

Design: Minimal and accessible, intended for both technical and non-technical users.

Key Elements:

Navigation tabs (Code Analysis, Code Generation)

File upload for PDF analysis

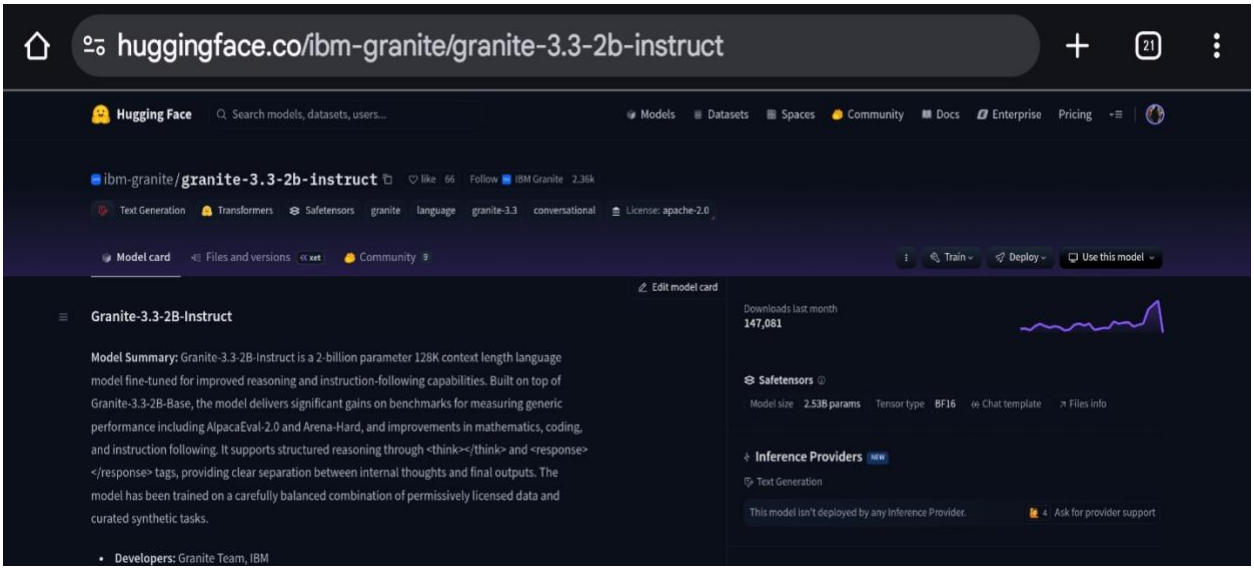
Text boxes for requirements/code prompts

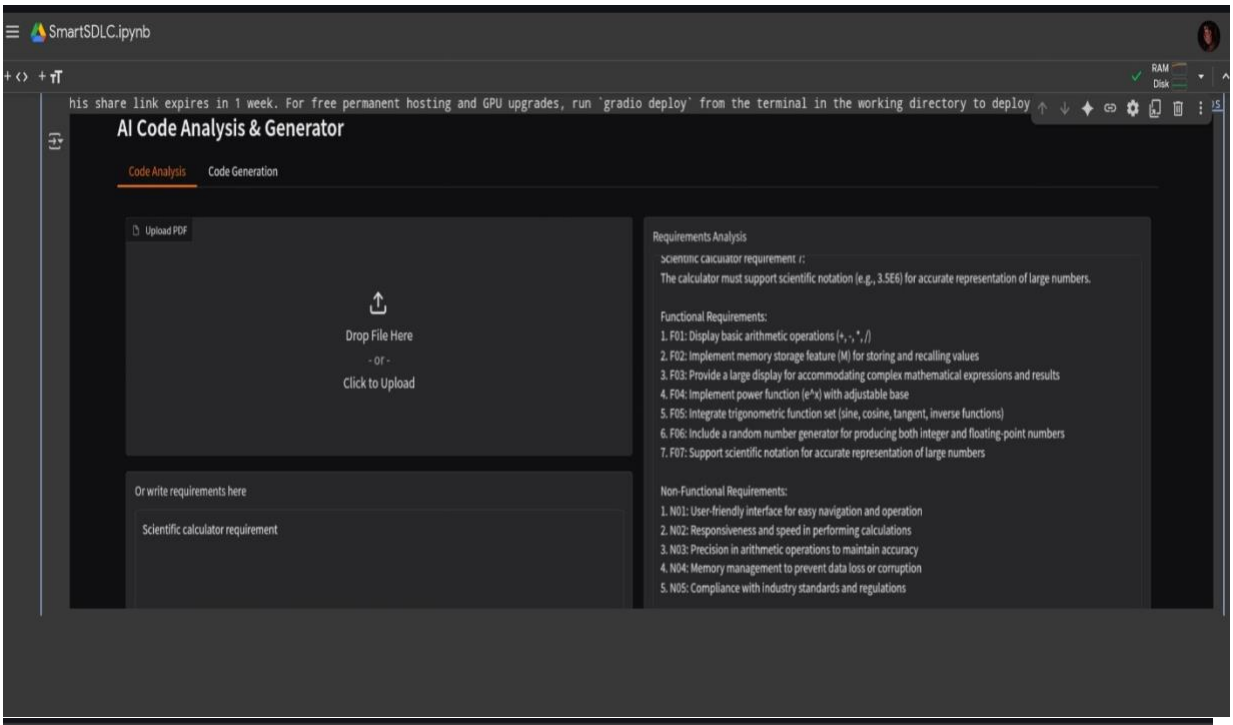
Output areas for results.

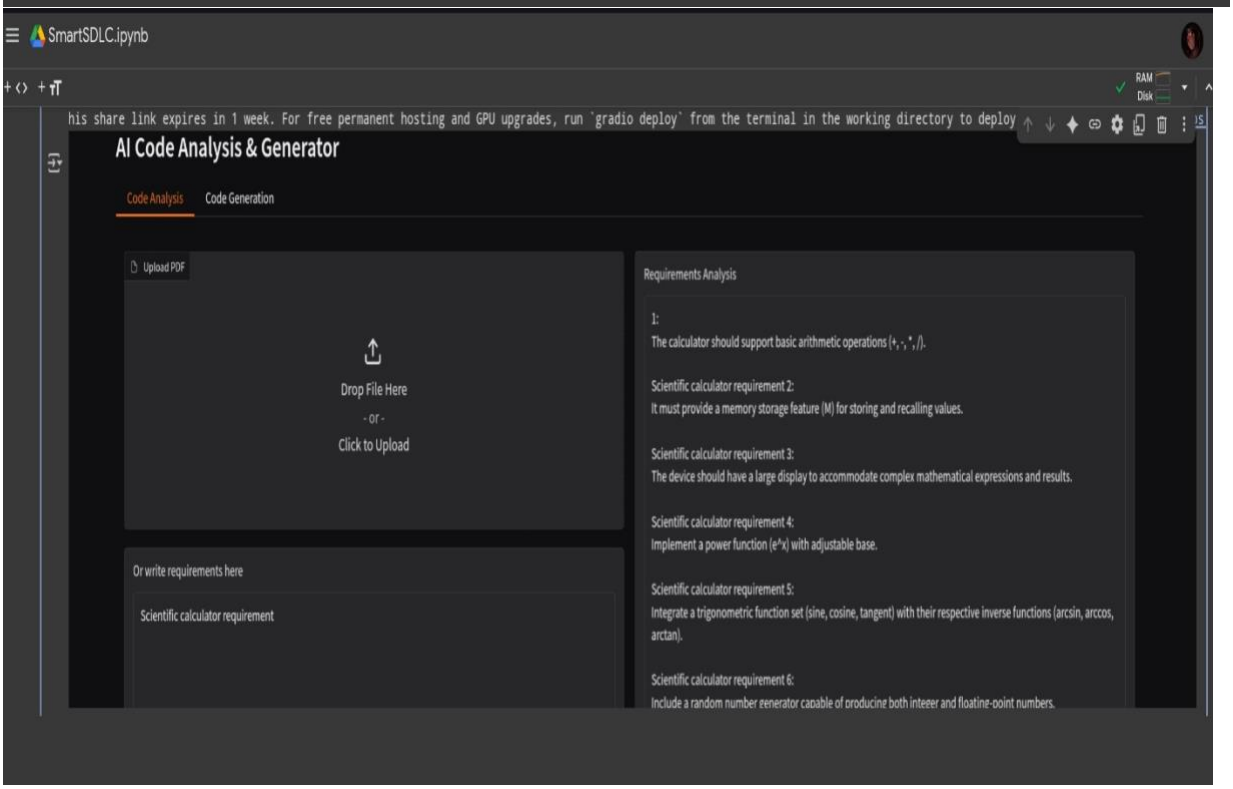
10. Testing

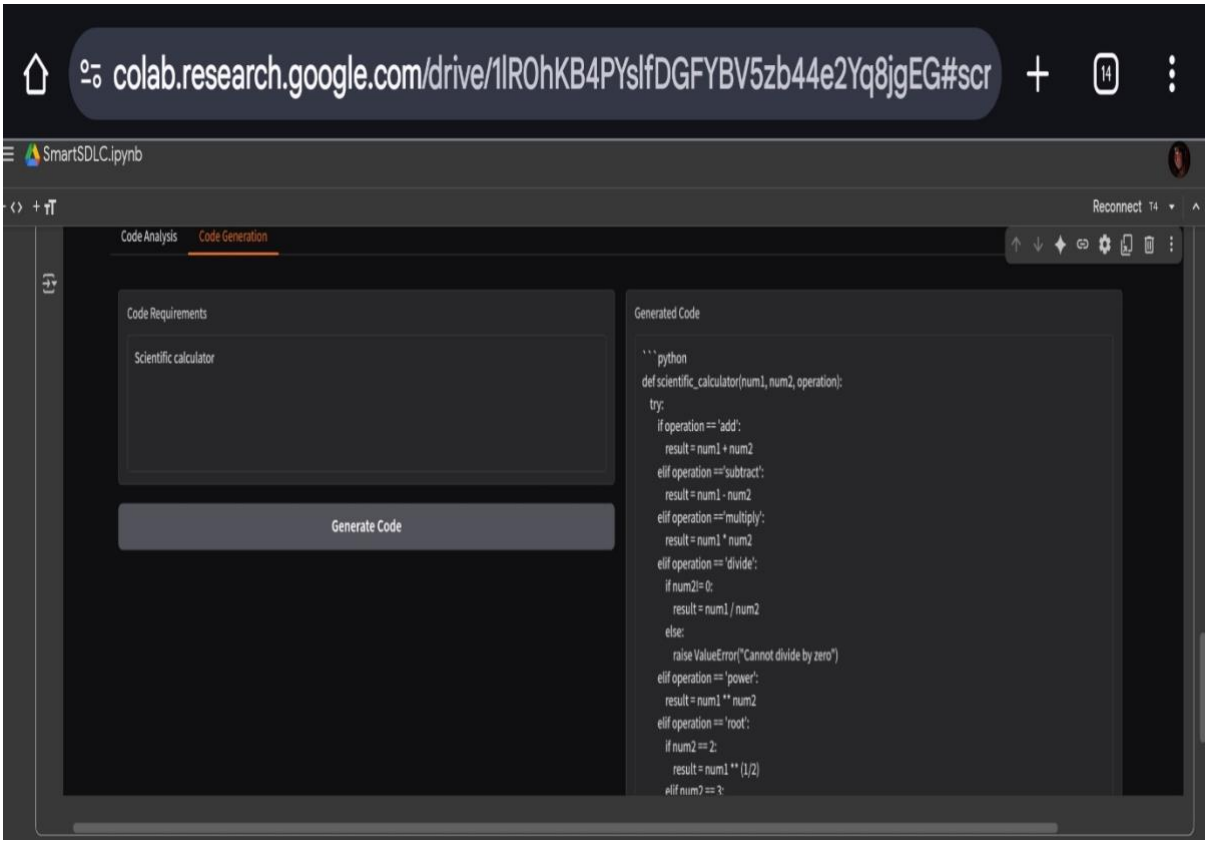
Unit testing for model prompt functions and utility code recommended Manual UI testing for uploads, responses, and output format Handle edge cases (malformed/large documents, empty inputs).

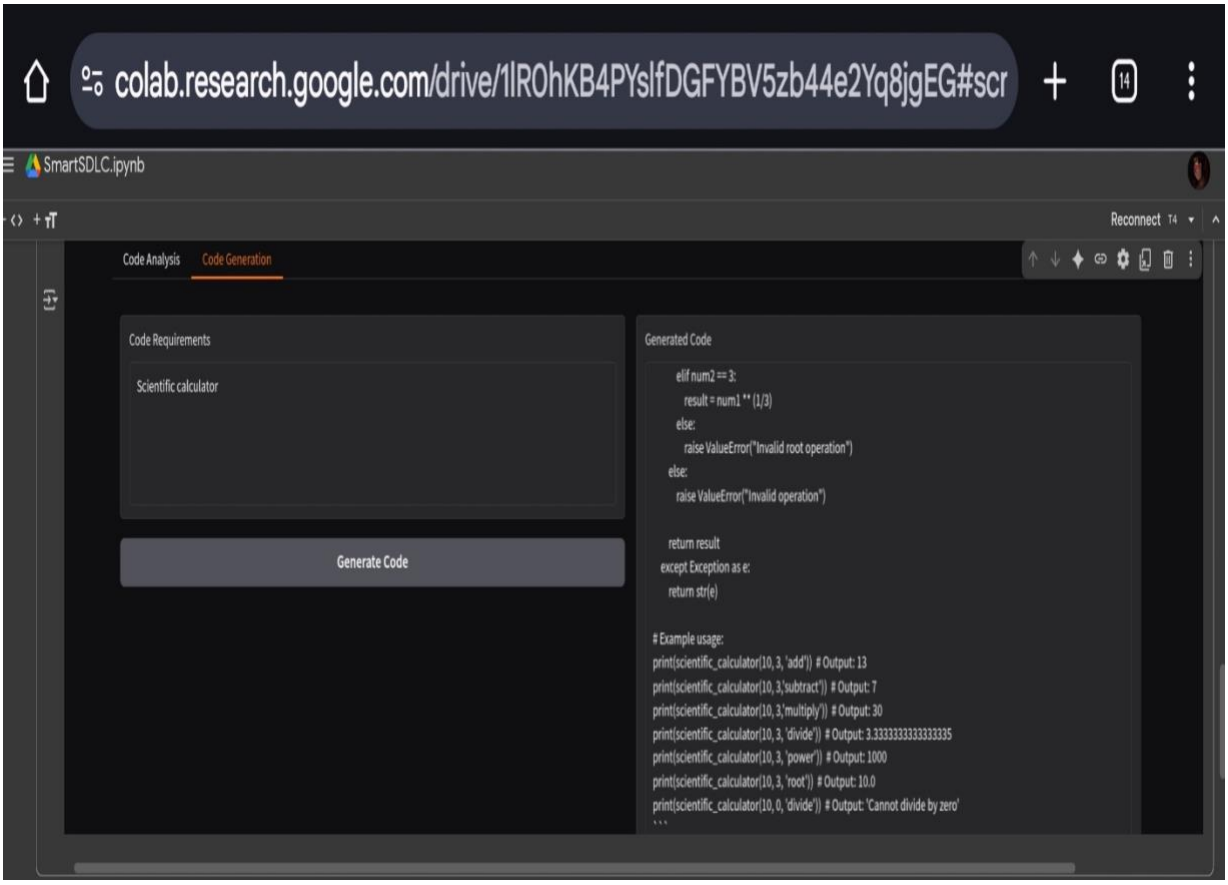
Unit testing for model prompt functions and utility code recommended Manual UI testing for uploads, responses, and output format Handle edge cases (malformed/large documents, empty inputs).




SmartSDLC.ipynb
his share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy
AI Code Analysis & Generator
Code Analysis
Code Generation
Upload PDF
Drop File Here
- or -
Click to Upload
Or write requirements here
Scientific calculator requirement
Requirements Analysis
Scientific calculator requirement r:
The calculator must support scientific notation (e.g., 3.5E6) for accurate representation of large numbers.
Functional Requirements:
1. F01: Display basic arithmetic operations (+, -, *, /)
2. F02: Implement memory storage feature (M) for storing and recalling values
3. F03: Provide a large display for accommodating complex mathematical expressions and results
4. F04: Implement power function (e^x) with adjustable base
5. F05: Integrate trigonometric function set (sine, cosine, tangent, inverse functions)
6. F06: Include a random number generator for producing both integer and floating-point numbers
7. F07: Support scientific notation for accurate representation of large numbers
Non-Functional Requirements:
1. N01: User-friendly interface for easy navigation and operation
2. N02: Responsiveness and speed in performing calculations
3. N03: Precision in arithmetic operations to maintain accuracy
4. N04: Memory management to prevent data loss or corruption
5. N05: Compliance with industry standards and regulations


SmartSDLC.ipynb
his share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy
AI Code Analysis & Generator
Code Analysis
Code Generation
Upload PDF
Drop File Here
- or -
Click to Upload
Or write requirements here
Scientific calculator requirement
Requirements Analysis
1:
The calculator should support basic arithmetic operations (+, -, *, /).
Scientific calculator requirement 2:
It must provide a memory storage feature (M) for storing and recalling values.
Scientific calculator requirement 3:
The device should have a large display to accommodate complex mathematical expressions and results.
Scientific calculator requirement 4:
Implement a power function (e^x) with adjustable base.
Scientific calculator requirement 5:
Integrate a trigonometric function set (sine, cosine, tangent) with their respective inverse functions (arcsin, arccos, arctan).
Scientific calculator requirement 6:
Include a random number generator capable of producing both integer and floating-point numbers.


colab.research.google.com/drive/1lR0hKB4PYslfDGFYBV5zb44e2Yq8jgEG#scr
SmartSDLC.ipynb
Reconnect T4
Code Analysis
Code Generation
Code Requirements
Scientific calculator
Generate Code
Generated Code
```python
def scientific_calculator(num1, num2, operation):
try:
if operation == 'add':
result = num1 + num2
elif operation =='subtract':
result = num1 - num2
elif operation =='multiply':
result = num1 * num2
elif operation == 'divide':
if num2!= 0:
result = num1 / num2
else:
raise ValueError("Cannot divide by zero")
elif operation == 'power':
result = num1 ** num2
elif operation == 'root':
if num2 == 2:
result = num1 ** (1/2)


colab.research.google.com/drive/1lR0hKB4PYslfDGFYBV5zb44e2Yq8jgEG#scr
SmartSDLC.ipynb
Reconnect T4
Code Analysis
Code Generation
Code Requirements
Scientific calculator
Generate Code
Generated Code
elif num2 == 3:
result = num1 ** (1/3)
else:
raise ValueError("Invalid root operation")
else:
raise ValueError("Invalid operation")
return result
except Exception as e:
return str(e)
# Example usage:
print(scientific_calculator(10, 3, 'add')) # Output: 13
print(scientific_calculator(10, 3,'subtract')) # Output: 7
print(scientific_calculator(10, 3,'multiply')) # Output: 30
print(scientific_calculator(10, 3, 'divide')) # Output: 3.3333333333333335
print(scientific_calculator(10, 3, 'power')) # Output: 1000
print(scientific_calculator(10, 3, 'root')) # Output: 10.0
print(scientific_calculator(10, 0, 'divide')) # Output: 'Cannot divide by zero'
```