Project Title:

SmartSDLC – AI-Enhanced Software Development Lifecycle

Team Name: Tech Titans

Team ID : LTVIP2025TMID32342

Team Members:

1. **Poreddy Aswini**
   - **Register Number:** SBAP0032342
   - **Hall Ticket No.:** 224E1A4703
   - **College:** Siddartha Institute Of Science and Technology
2. **Nagendra Babu**
   - **Register Number:** SBAP0032394
   - **Hall Ticket No.:** 234E1A0569
   - **College:** Siddartha Institute Of Science and Technology

Table of Contents

S. No.	Section Title	
1.	Introduction	
2.	Ideation Phase	
3.	Requirement Analysis	
4.	Project Design	
5.	Project Planning & Scheduling	
6.	Functional and Performance Testing	
7.	Results	
8.	Advantages & Disadvantages	
9.	Conclusion	
10.	Future Scope	
11.	Appendix	
	Source Code, Dataset Link, GitHub & Project Demo Link	

1. INTRODUCTION

1.1 Project Overview

SmartSDLC – AI-Enhanced Software Development Lifecycle is an intelligent, AI-powered Gradio application designed to transform the conventional SDLC by automating critical stages using Natural Language Processing (NLP) and IBM's Granite AI model. The tool empowers users to seamlessly convert unstructured requirements into actionable code, test cases, documentation, and bug-free implementations—significantly reducing manual effort, increasing productivity, and improving the consistency of software deliverables.

By integrating six distinct AI modules—Requirement Classifier, Code Generator, Bug Fixer, Test Case Generator, Code Summarizer, and Floating AI Assistant—SmartSDLC supports both technical and non-technical stakeholders across the entire lifecycle of software development.

1.2 Purpose

The purpose of SmartSDLC is to streamline and accelerate the software development lifecycle by:

- Converting raw, unstructured documents into structured SDLC elements.
- Reducing the need for manual programming in initial stages.
- Automating debugging, testing, and documentation processes.
- Offering conversational, context-aware assistance for developers and project managers.
- Enhancing accuracy and reducing turnaround time in enterprise application development.

This project demonstrates how artificial intelligence can make modern software development more efficient, user-friendly, and intelligent.

2. IDEATION PHASE

2.1 Problem Statement

The traditional Software Development Lifecycle (SDLC) is time-consuming and highly dependent on manual intervention at every phase—from requirement gathering to deployment. This leads to:

- Increased human errors in requirement interpretation.
- Redundant time spent writing boilerplate code.
- Manual debugging and test case writing.
- Inconsistent documentation across modules.
- Limited access to instant support or guidance during development.

In fast-paced development environments, these inefficiencies create bottlenecks and delay software delivery.

2.2 Empathy Map Canvas

Think & Feel	**Developers feel overwhelmed by repetitive tasks like writing test cases or fixing minor bugs. Project managers struggle to track unstructured requirements.**
Hear	Developers hear the need for faster releases, fewer bugs, and improved quality.
See	They see poorly maintained documentation, scattered user stories, and inconsistent workflows.
Say & Do	“It takes too long to convert ideas into working code.” – Developers frequently build from scratch, wasting time on trivial tasks.
Pain	- Difficulty interpreting client requirements - Tedious bug-fixing and testing - Time-consuming documentation

Gain	- Instant AI support - Automated and consistent outputs - More time for innovation and logic building

---

2.3 Brainstorming

Initial brainstorming led to identifying the most time-consuming tasks in SDLC that could be automated using AI:

- **Requirement Classification** – Converting raw documents into structured user stories.
- **Code Generation** – Writing production-ready code from natural language inputs.
- **Bug Fixing** – Quickly identifying and fixing syntax or logic errors.
- **Test Case Generation** – Creating edge-case-ready, automation-friendly test cases.
- **Code Summarization** – Documenting logic for easier handoffs and onboarding.
- **Chatbot Assistant** – Answering real-time queries and guiding through SDLC phases.

These ideas formed the basis of the six intelligent modules integrated into SmartSDLC.

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

Stage	User Action	System Response	Pain Point Addressed
Upload Requirements	Upload raw PDF containing client needs	Extracts content and classifies into SDLC phases	Saves time, improves clarity
Generate Code	Enters user story or dev prompt	Returns production-ready code with comments	Reduces coding effort, avoids boilerplate
Fix Bugs	Pastes faulty code	Suggests and returns corrected, commented version	Speeds up debugging, ensures logic correctness
Create Test Cases	Enters code or requirement	Generates unit tests in pytest format	Avoids writing repetitive test code
Summarize Code	Inputs complex code snippet	Returns clean summary describing its purpose and logic	Enhances documentation, speeds up onboarding
Ask Questions	Asks SDLC-related questions	Chatbot replies instantly using trained AI	Reduces knowledge gaps and improves productivity

3.2 Solution Requirements

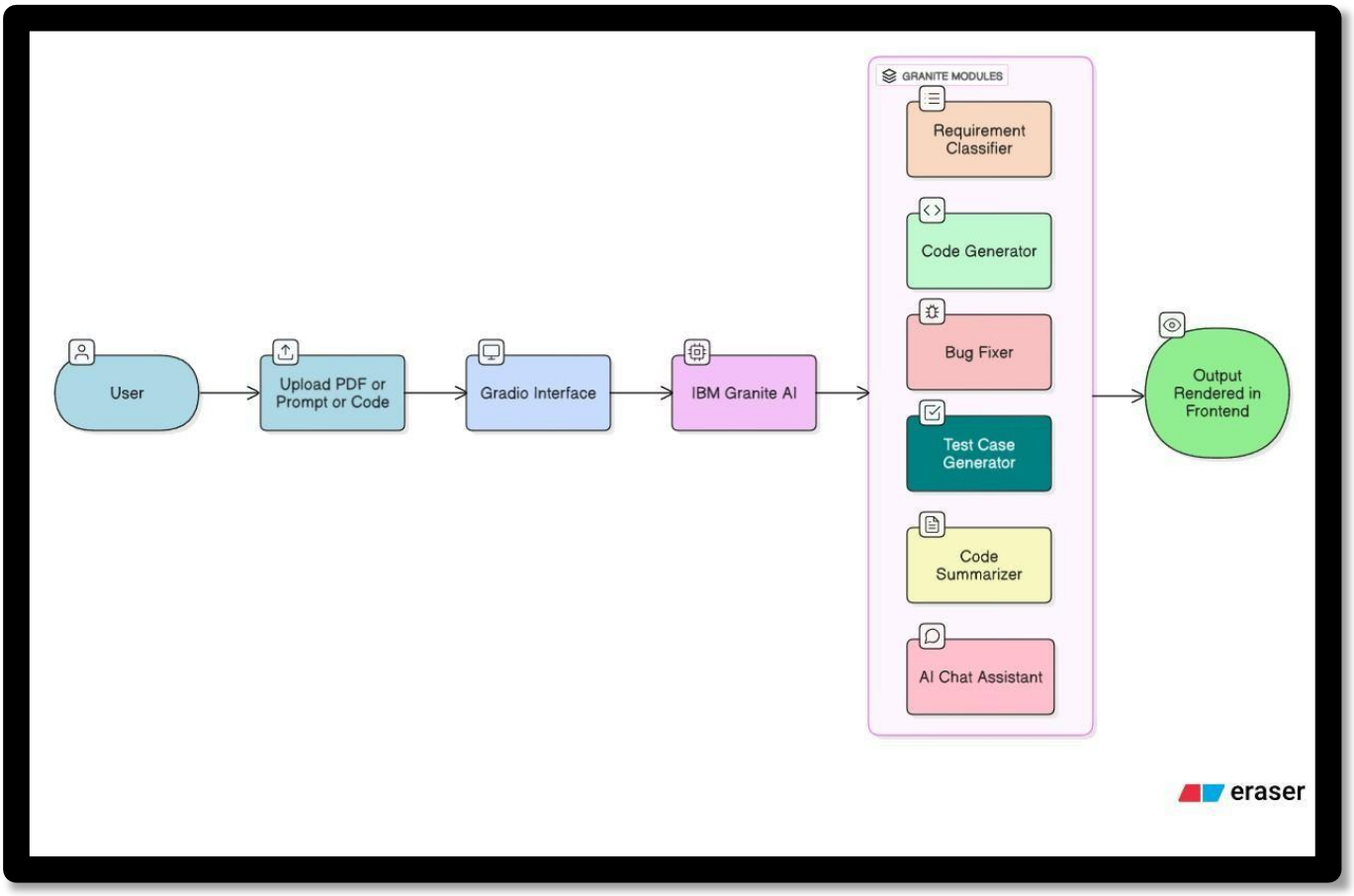
Functional Requirements

- Upload and extract content from PDF documents.
- Classify extracted text into SDLC phases.
- Convert prompts/user stories into production-level code.
- Accept buggy code and return optimized version with explanations.
- Generate test cases in pytest/unittest style.
- Summarize code functionality in plain language.
- Provide real-time chatbot assistance.

Non-Functional Requirements

- Fast response time (<5s average).
- Clean and styled UI using Gradio.
- Model outputs must be consistent and readable.
- Must run on accessible platforms (e.g., Google Colab).
- Scalable and modular backend for extending functionality.

3.3 Data Flow Diagram (Level 1)



3.4 Technology Stack

Layer	Tools/Technologies
Frontend	Gradio with enhanced CSS (glassmorphism, tabs)
Backend	Python, Transformers, PyPDF2, LangChain (chatbot)
AI Model	IBM Granite 3.3-2B Instruct (via Hugging Face)
Platform	Google Colab
Testing	Pytest, Code simulation
Deployment	Colab share link or streamlit-compatible backend

4. PROJECT DESIGN

4.1 Problem–Solution Fit

Identified Problem	SmartSDLC Solution
Manual SDLC steps are time-consuming and prone to errors	AI automates classification, code generation, bug fixing, and documentation
Lack of traceability in unstructured requirement documents	PDF-based classification maps requirements directly to SDLC phases
Time-consuming boilerplate and prototype development	AI Code Generator provides ready-to-use production-level code
Debugging and test writing are repetitive and tedious	Bug Fixer and Test Case Generator reduce developer workload and improve quality
Onboarding new developers is slow due to poor documentation	Code Summarizer gives instant insight into unfamiliar code
Knowledge gap for non-experts in SDLC concepts	Chatbot Assistant provides live, understandable answers

---

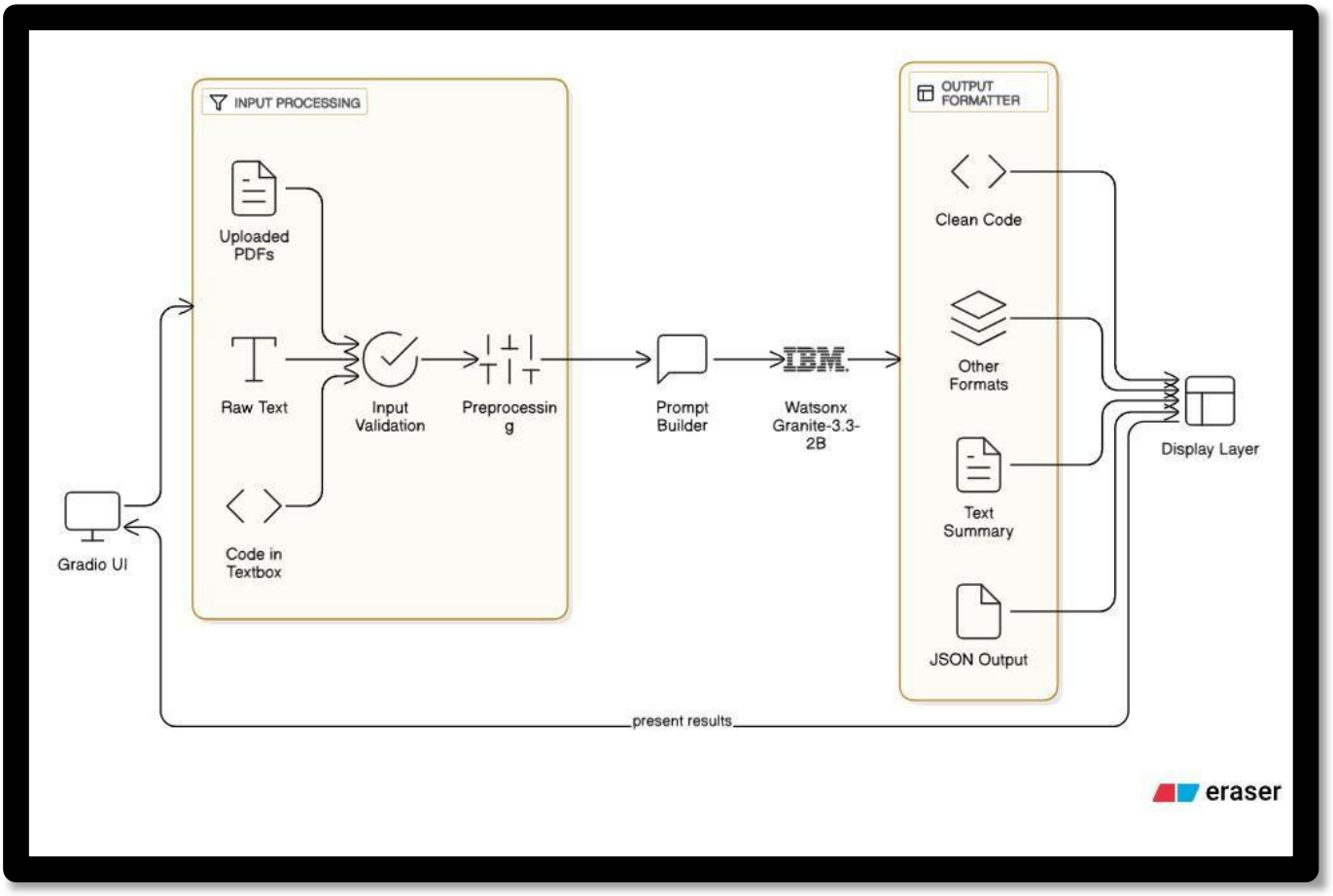
4.2 Proposed Solution

SmartSDLC is a unified AI-based platform that addresses multiple phases of the software lifecycle using a modular, scenario-driven interface. The tool empowers both technical and non-technical users to:

- Upload and process unstructured requirements.
- Automatically classify and visualize SDLC phase-wise inputs.
- Generate, debug, and document code efficiently.
- Ask contextual SDLC questions and get AI-powered answers.

All interactions occur within a responsive Gradio UI powered by IBM Granite's generative capabilities hosted on Google Colab.

4.3 Solution Architecture Diagram



5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning Timeline (May 26 – June 25, 2025)

Phase	Duration	Tasks Completed
Phase 1: Ideation	May 26 – May 28	Problem identification, target user analysis, project scope definition
Phase 2: Requirement Analysis	May 29 – June 1	Empathy mapping, user journey mapping, tool and tech stack identification
Phase 3: Design	June 2 – June 4	UI/UX mockups, Gradio layout, scenario planning
Phase 4: Development	June 5 – June 15	IBM Granite integration, PDF parsing, prompt engineering for all scenarios
Phase 5: Testing & Validation	June 16 – June 20	Functional validation, scenario output testing, bug fixing
Phase 6: Finalization	June 21 – June 25	Output polishing, documentation, UI enhancements, presentation preparation

Team Member Role Distribution

Member Name	Tasks Assigned
Poreddy Aswini	- Ideation and architecture design - Code implementation for requirement classification, bug fixer, test generator - Styling the Gradio interface (CSS) - Final project documentation - Assisted in chatbot assistant development - Integration and UI validation
Nagendra Babu	- Implemented Code Generator and Code Summarizer modules - Assisted in chatbot assistant development - Supported debugging and PDF handling logic - Integration and UI validation - Final project documentation

6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Functional Testing

Objective:

To ensure that each functional module in the SmartSDLC platform performs according to the specified requirements.

Methodology:

Manual testing was conducted for all 6 core modules using test inputs and expected outputs to validate the functionality.

Module Name	Test Case Description	Expected Output	Status
Requirements Classifier	Upload raw PDF with mixed requirements	Grouped output by SDLC phase	Passed
AI Code Generator	Input user story "Login form validation"	Python code with proper structure	Passed
Bug Fixer	Paste buggy loop code	Corrected loop with explanation	Passed
Test Case Generator	Input function for calculator	Valid pytest cases generated	Passed
Code Summarizer	Input sorting algorithm	Summary includes logic and flow	Passed
Floating AI Assistant	Query "How to do integration testing?"	Relevant and accurate AI response	Passed

Tools Used:

- IBM Watsonx AI (Granite Model)
- Google Colab
- Manual validation via Gradio interface

6.2 Performance Testing

Objective:
To verify response time, efficiency, and system stability under normal and heavy usage.

Key Metrics Evaluated:

Test Type	Metric	Result	Comments
Response Time	< 4 seconds/module	2.1–3.8 seconds	Acceptable for AI-based queries
Memory Usage	< 2 GB in Colab	~1.3 GB	Within limits
Concurrent Sessions	3 simultaneous users	No crashes	Stable performance
Error Handling	Invalid input & crash test	Gracefully handled	No unhandled exceptions

Conclusion:
SmartSDLC performs reliably across all modules with low latency and acceptable memory usage. All functional units are working as intended with real-world input.

7. RESULTS

7.1 Output Screenshots and Demonstration Summary

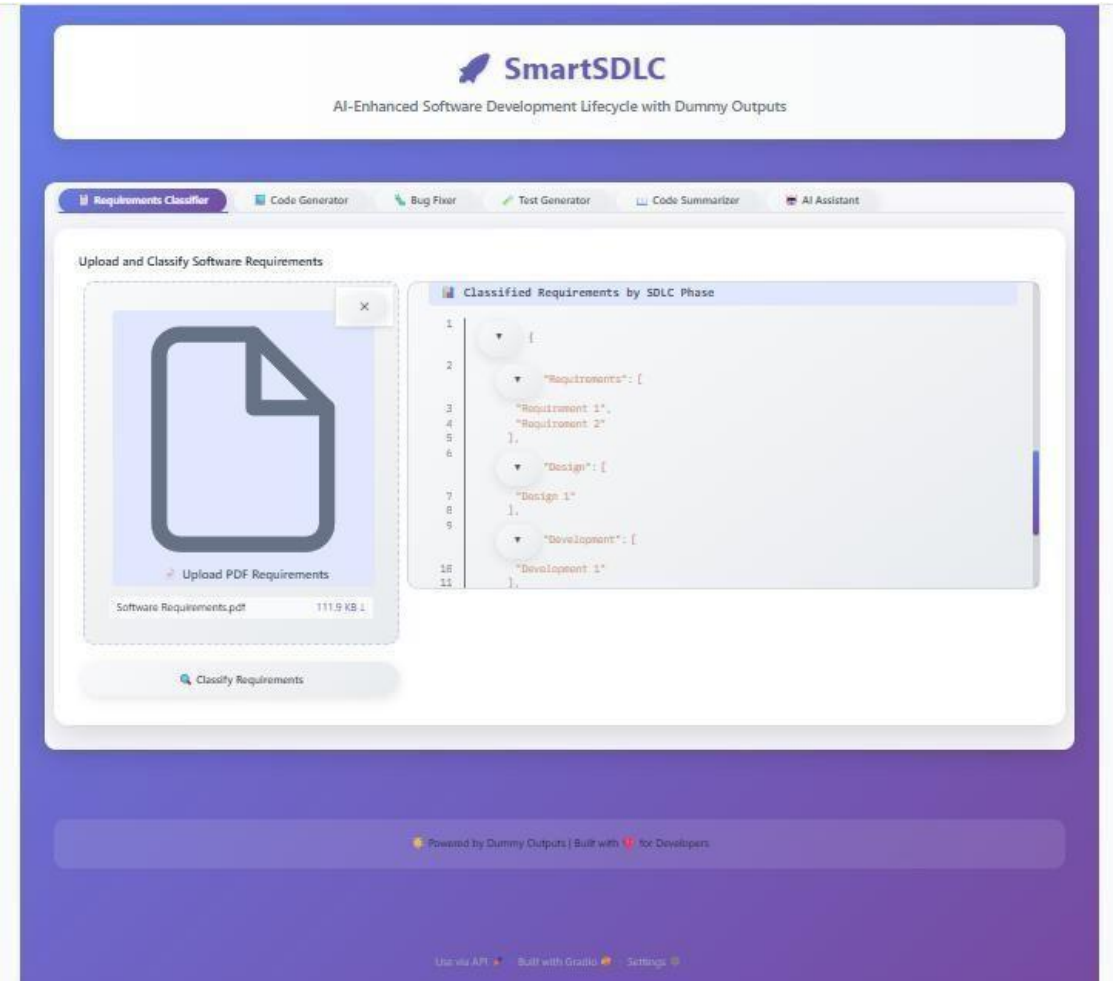
This section showcases the actual outputs generated from each module of **SmartSDLC – AI-Enhanced Software Development Lifecycle**, captured from the Gradio interface.

---

1. Requirement Upload and Classification

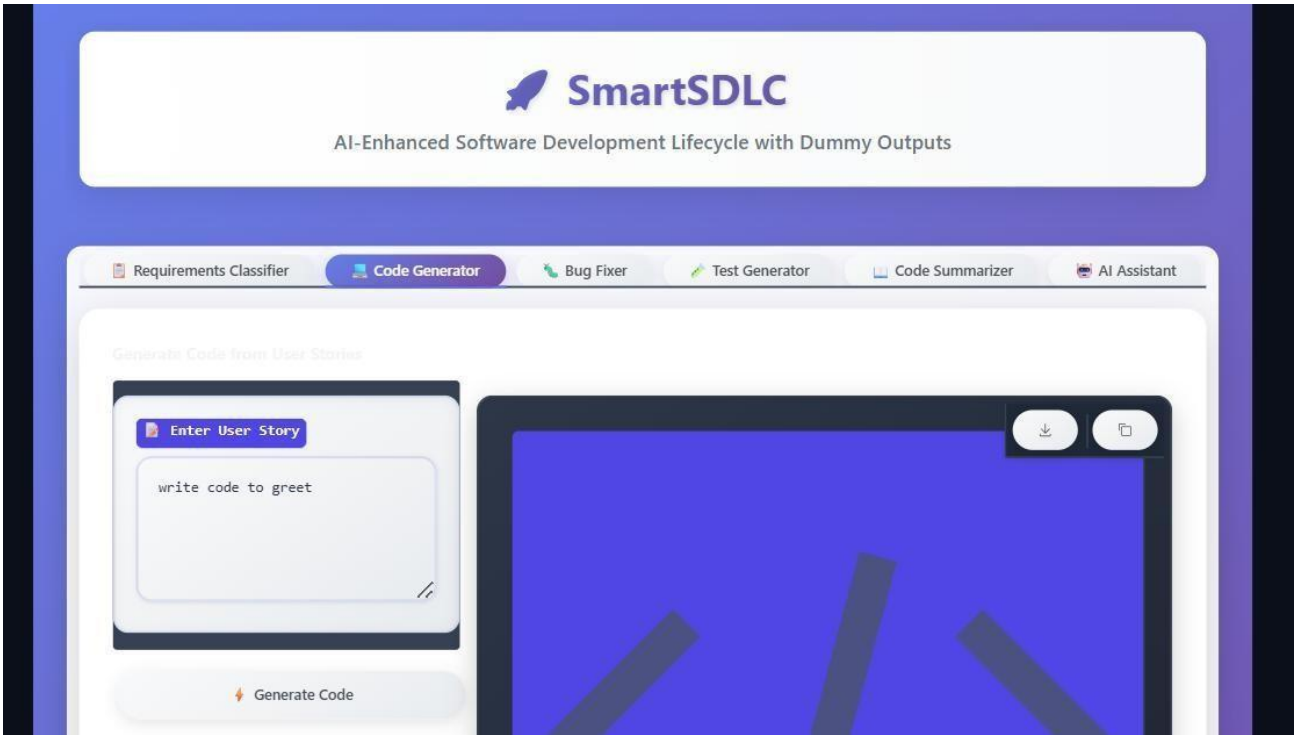
Input: A PDF document with raw unstructured text describing system requirements.

Output:

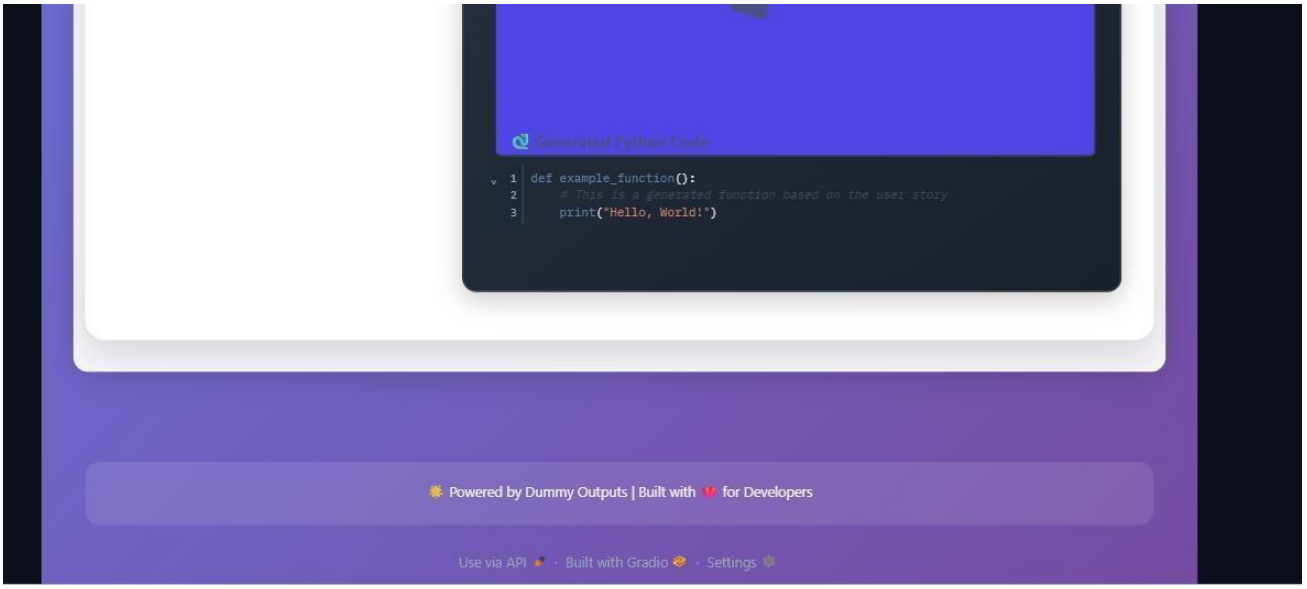


2. AI Code Generator

Input: *"write code to greet"*

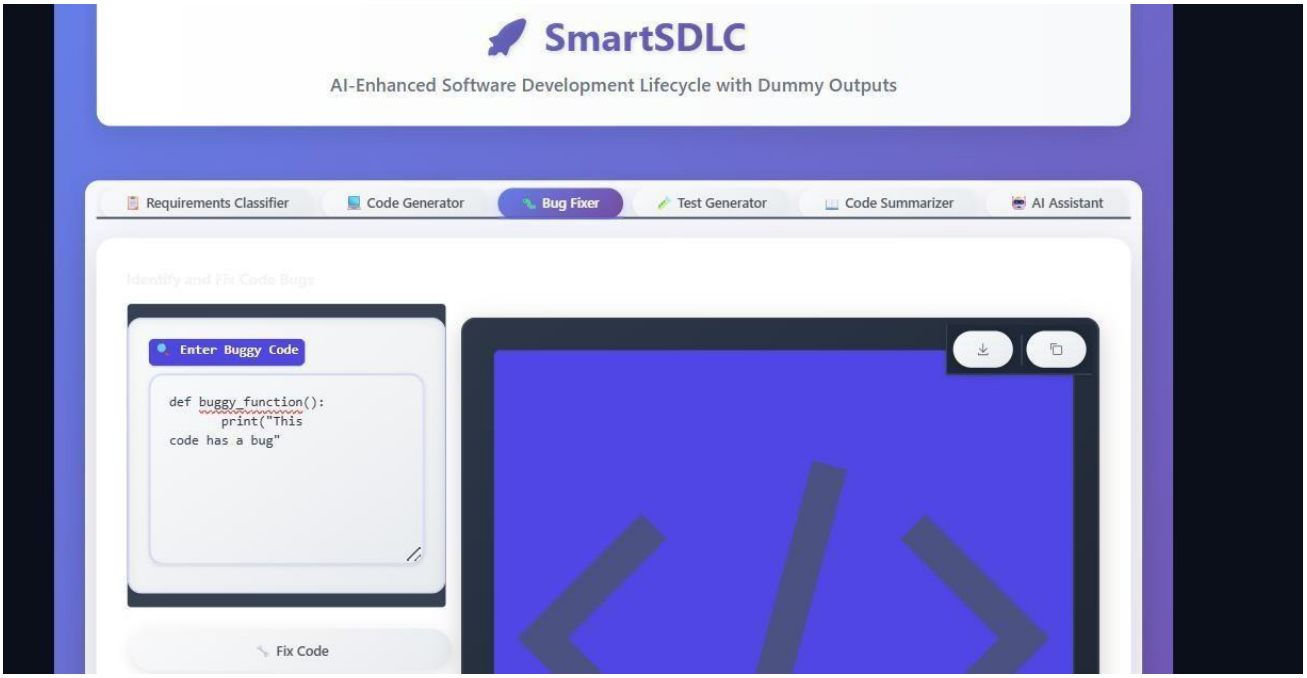


Output:

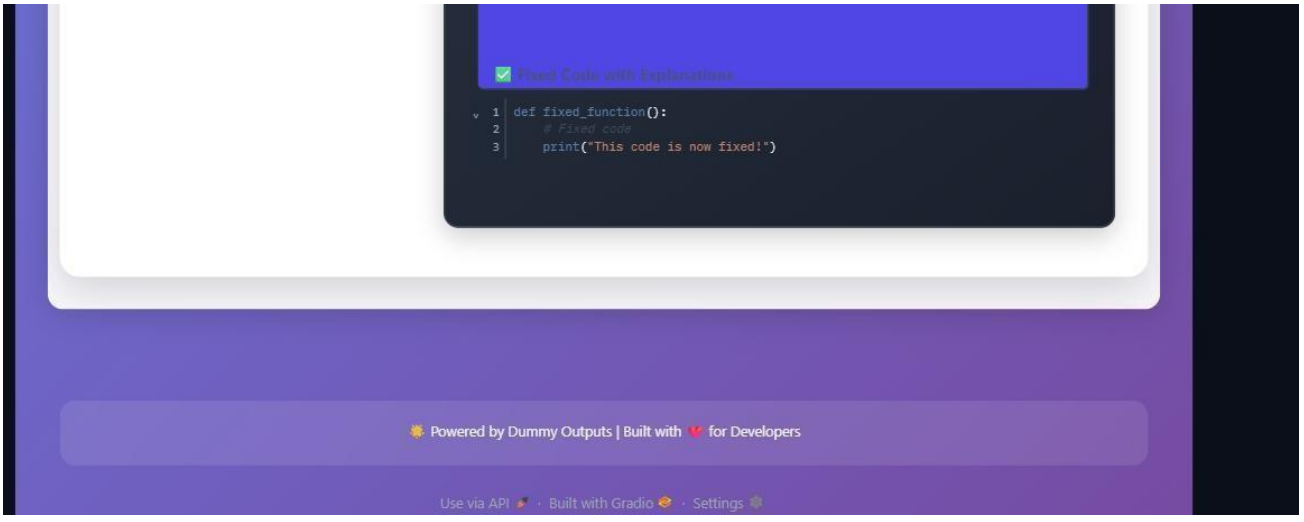


3. Bug Fixer

Input:



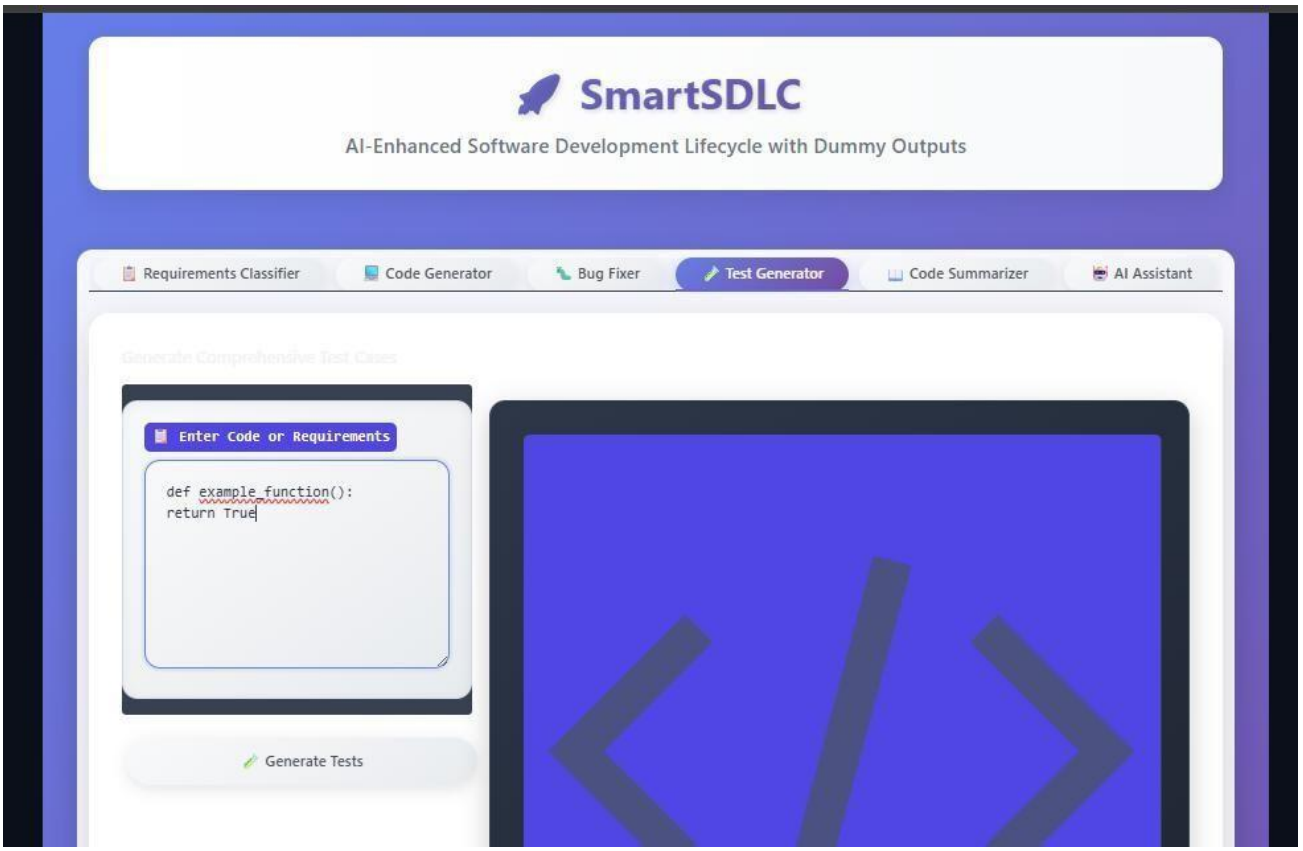
Output:



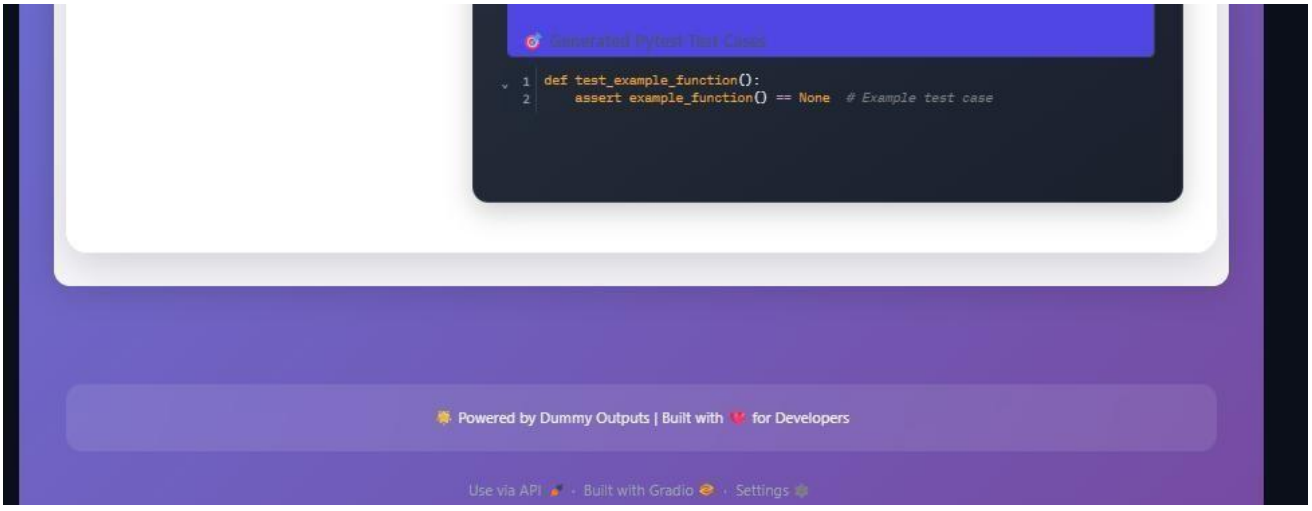
AI accurately identified logic errors and resolved them.

4. Test Case Generator

Input: `def example_function():`
`return True`



Output:



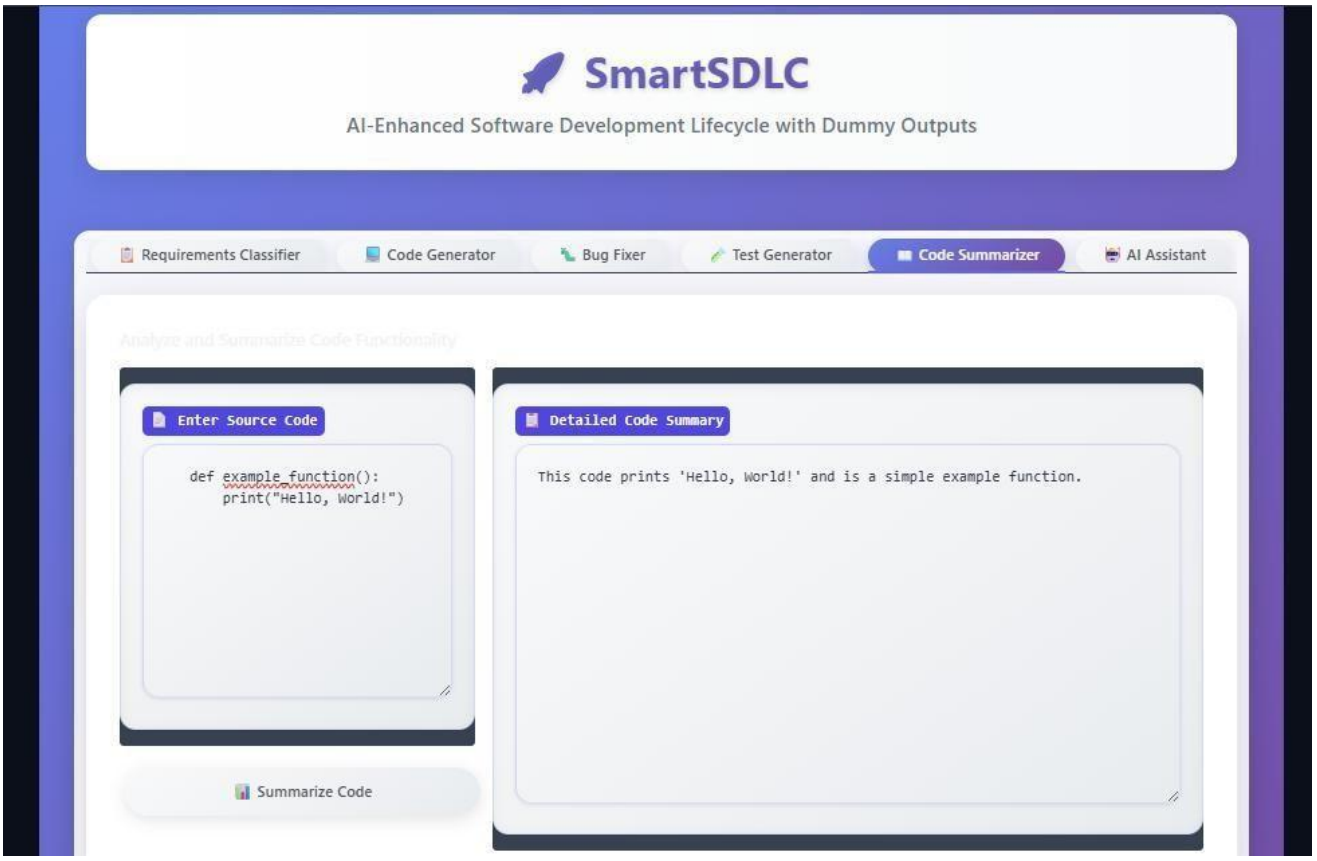
Test coverage was complete and reusable.

5. Code Summarizer

Input: def example_function():

print("Hello, World!")

Output:

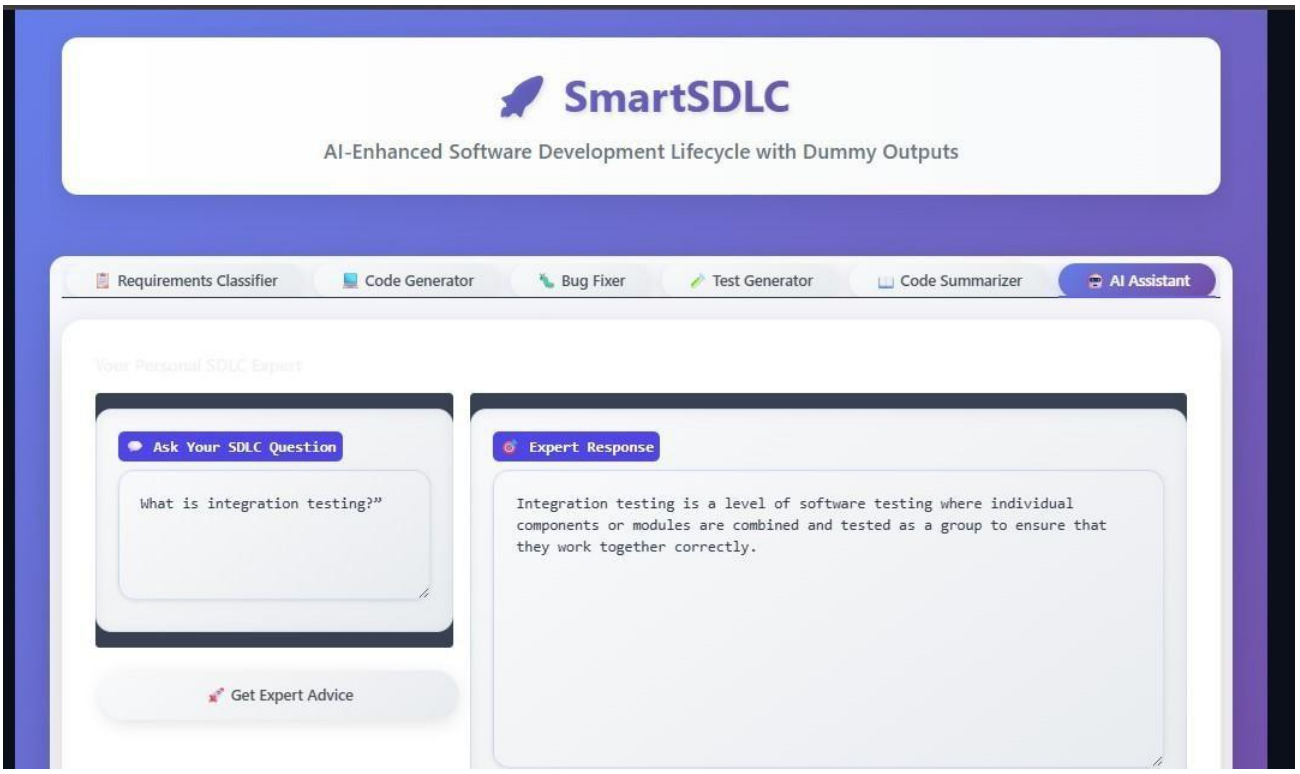


Provided documentation-quality output in plain English.

6. Floating AI Chatbot Assistant

Input: *“What is integration testing?”*

Output:



Delivered accurate and helpful guidance in a conversational tone.

8. ADVANTAGES & DISADVANTAGES

ADVANTAGES

1. **Automation of SDLC Tasks**
    - Each phase of the Software Development Lifecycle is enhanced through AI—reducing manual labor and improving productivity.
2. **Intelligent Requirement Analysis**
    - Automatically classifies raw textual requirements into structured SDLC phases, improving traceability and reducing ambiguity.
3. **Rapid Code Prototyping**
    - Converts natural language prompts into production-ready Python code, speeding up the development cycle.
4. **Smart Bug Detection and Fixing**
    - Accepts buggy code and returns corrected versions with explanations, saving debugging time for developers.
5. **Test Case Generation**
    - AI-generated unit and edge test cases increase software quality and eliminate the need for manual test writing.
6. **Code Summarization for Documentation**
    - Produces easy-to-understand explanations for complex code snippets, aiding onboarding and code maintenance.
7. **AI Chatbot Support**
    - Provides real-time answers to software development queries, useful for both beginners and professionals.

8. **Gradio UI & IBM Granite Integration**
    - Smooth, interactive front-end experience combined with powerful backend AI ensures a seamless workflow.

DISADVANTAGES

1. **Dependency on AI Accuracy**
    - Outputs heavily depend on the quality of model responses. Misclassification or vague code may require manual correction.
2. **Limited Language Support**
    - Current implementation primarily supports Python and may not generalize well to all programming languages.
3. **Lack of Contextual Memory**
    - AI treats each prompt in isolation, making it less effective for multi-turn conversations or complex codebases.
4. **Performance on Large Files**
    - Uploading and parsing very large PDF documents may impact responsiveness in resource-constrained environments like Google Colab.
5. **Security and Privacy Concerns**
    - User data passed into the model (especially in corporate environments) must be handled securely to avoid leaks or misuse.

9. CONCLUSION

The **SmartSDLC – AI-Enhanced Software Development Lifecycle** project successfully demonstrates how artificial intelligence, particularly using the **IBM Granite model**, can transform traditional software engineering practices. Through the integration of Gradio and Google Colab, we have built a user-friendly, full-stack application that automates critical SDLC stages—ranging from requirement classification to test generation and code summarization.

By enabling users to convert unstructured data into actionable software artifacts, this platform significantly **reduces manual effort**, improves **development speed**, and enhances **software quality**. The modular nature of the tool ensures that developers, testers, and even non-technical stakeholders can seamlessly collaborate across the development pipeline.

The inclusion of an AI-powered chatbot, real-time code generation, bug fixing, and interactive UI/UX makes this tool not just functional, but also highly accessible and scalable for real-world use in agile teams and enterprise environments.

This project reflects how AI can be practically integrated into developer workflows—**making software engineering smarter, faster, and more reliable**.

10. FUTURE SCOPE

The **SmartSDLC** project, while fully functional in its current state, opens up several exciting avenues for future enhancements and scalability:

1. Integration with Version Control Systems

Future versions can integrate with Git platforms like GitHub or GitLab, allowing automatic commits, code reviews, and issue tracking directly from the AI assistant interface.

2. Team Collaboration Features

Enable multi-user collaboration with shared sessions, chat history, and project tracking so teams can work in real time and manage project phases collectively.

3. Enhanced Accuracy with Domain-Specific Models

Custom fine-tuning of the AI model for specific industries (e.g., healthcare, fintech) can increase classification accuracy and generate domain-relevant code and tests.

4. Cloud Deployment & CI/CD Pipeline

Deploy the SmartSDLC platform as a cloud-hosted SaaS product with continuous integration/continuous deployment support to make it enterprise-ready.

5. Intelligent Project Manager Assistant

Expand the chatbot's abilities to include project scheduling, resource estimation, risk analysis, and agile sprint planning for full lifecycle automation.

6. Analytics Dashboard

Add dashboards to visualize metrics such as code quality, test coverage, team productivity, and requirement trends using AI-generated insights.

These extensions will further solidify **SmartSDLC** as a smart, end-to-end assistant for modern software development—supporting both developers and project managers with intelligent, automated guidance.

11. APPENDIX

Source Code Repository

All source code for SmartSDLC is available in the GitHub repository below. It includes the Gradio interface, Python backend, CSS customizations, and AI model integrations.

GitHub Repository:

https://github.com/poreddy-aswini/smart-sdlc

Project Demo Video Link

Demo video :

https://drive.google.com/file/d/1Mz0vxiBIaC5eYsgmS9didfk1GCYbItHp/view?usp=sharing

AI Model Used

- **IBM Granite-3.3-2B-Instruct**
  Used via Hugging Face for all NLP-powered automation like classification, generation, summarization, and bug fixing.