Smart SDLC – AI-Enhanced Software Development Lifecycle

Team ID : LTVIP2025TMID59978

Team Size : 4

Team Leader : Gopidhinne Reddy Chaitanya

Team member : Majji Madhuri

Team member : Iragam Reddy Nagalakshmi

Team member : Arveti Narendra

1. Project Overview:

SmartSDLC is an AI-powered, full-stack cloud application designed to transform the traditional Software Development Lifecycle (SDLC) by integrating intelligent automation into each core phase. Built on IBM Cloud and powered by the **IBM Granite foundation model**, SmartSDLC leverages advanced Natural Language Processing (NLP) and Generative AI to convert unstructured inputs—such as raw text requirements or code—into actionable development artifacts like structured user stories, production-ready code, test cases, bug fixes, and documentation. The IBM Granite model serves as the AI engine across the platform, enabling deep understanding and contextual generation of software assets with high accuracy. By automating requirement classification, code generation, debugging, testing, summarization, and real-time support through an integrated chatbot, SmartSDLC significantly enhances productivity, reduces manual effort, and ensures consistency across the SDLC, making it a powerful ecosystem for modern software teams.

→ **Purposes:**

- Automate crucial stages of the Software Development Lifecycle (SDLC) with artificial intelligence.
- Increase developer efficiency by automatically producing code, resolving errors, and building test cases.

Organize unstructured needs into user stories to improve planning and clarity.

Enhance software quality and decrease human mistake by using AI-assisted debugging and testing.

- To assist with documentation and onboarding, provide concise, accessible code descriptions.

Facilitate the involvement of non-technical users in the development process by utilizing natural language inputs.

The IBM Granite foundation paradigm is utilized to offer a cohesive and intelligent development environment.

- Cut down on repetitive and manual labor to save money and time on development.
- For enterprise-ready deployment, scalability, and dependability, choose IBM Cloud.

Integrating AI and NLP into contemporary software engineering methods may foster creativity.

→ **Features:**

- Upload PDF files containing raw, unstructured requirements
- AI-based classification of content into SDLC phases (Requirement, Design, Development, Testing, Deployment)
- Automatic generation of structured user stories from extracted requirements
- Natural language prompt-to-code generation using IBM Granite model
- AI-powered bug detection and correction for Python and JavaScript code
- Automated test case generation using frameworks like unittest and pytest
- Code summarization with human-readable explanations for documentation and onboarding
- Floating AI chatbot assistant for real-time SDLC guidance and help
- Syntax-highlighted, clean code display on the frontend
- Fully cloud-based deployment on IBM Cloud for scalability and availability
- Modular design — each feature can be used independently or as part of an integrated workflow
- Intelligent prompt routing and keyword detection using LangChain for chatbot responses.

2. Architecture:

➢ **Frontend: React-Based Architecture**

React.js is used to create the SmartSDLC frontend, which offers a responsive and modular interface. It manages all interactions as the user-facing layer, including: • Uploading necessary PDFs
Examining the SDLC stages categorized by AI

Providing a syntax-highlighted code viewer; displaying AI-generated code, test cases, issue patches, and summary

- Using a floating AI chatbot helper to provide immediate assistance **Key Features of**

  Frontend Architecture:

- Component-based structure for maintainability and reusability
- API integration with backend services using Axios or Fetch
- State management using Context API or Redux (based on scalability needs)
- Real-time rendering of outputs (e.g., code, test cases) with formatting
- Chatbot UI built with styled components, supporting dynamic interaction

➢ **Backend: Node.js + Express.js Architecture**

The backend is developed using **Node.js** with the **Express.js** framework. It acts as the bridge between the frontend and AI services (powered by IBM Watsonx and IBM Granite model).
Core Responsibilities:

- Handle file uploads (PDFs) and extract text using PyMuPDF (via Python child process or microservice)
- Route classification, code generation, bug fixing, and summarization requests to IBM Granite via secure API calls
- Manage authentication (if implemented), sessions, and chatbot prompt routing using **LangChain**
- Expose RESTful APIs for all frontend operations
- Log and handle user interactions for analytics and feedback **Backend Services**

  Include:

- /upload – for uploading and processing PDF files
- /classify – for SDLC phase classification
- /generate-code – for code generation
- /fix-code – for bug fixing
- /summarize – for code summarization
- /chatbot – for chatbot message processing

➢ **Database: MongoDB Integration**

The project uses **MongoDB** as a NoSQL document-based database to store and manage dynamic data across sessions.

Database Usage Includes:

- User profiles and roles (if login/auth is enabled)
- Uploaded requirement documents (metadata, classification results)
- Generated assets: code, test cases, summaries
- Chatbot query history for session continuity
- Audit logs for AI responses (optional for future analytics)

CODE:

```
# -*- coding: utf-8 -*-

"""smart.ipynb

Automatically generated by Colab. Original

file is located at
```

https://colab.research.google.com/drive/1BUmbwRYQxB7oMEGzDjs9daMeZZNPK95k

"""!pip install transformers torch gradio accelerate bitsandbytes PyPDF2
Imports import gradio as gr import torch from transformers import

AutoTokenizer, AutoModelForCausalLM, pipeline import PyPDF2 #

SmartSDLC-AI Core Class class SmartSDLC_AI:

def __init__(self):

self.model_name = "ibm-granite/granite-3.3-2b-instruct"

self.tokenizer = None self.model = None

self.pipeline = None self.load_model() def

load_model(self):

try:

print("🔄 Loading AI model...")

self.tokenizer = AutoTokenizer.from_pretrained(self.model_name,
trust_remote_code=True) self.model =

AutoModelForCausalLM.from_pretrained(

self.model_name, torch_dtype=torch.float16,

device_map="auto", trust_remote_code=True,

)

self.pipeline = pipeline(
"text-generation",

model=self.model,

tokenizer=self.tokenizer,

max_length=1024, temperature=0.7,

do_sample=True,

pad_token_id=self.tokenizer.eos_token_id

)

```
        print("☑ AI model loaded.")        except Exception as e:
print(f"❌ Error: {e}")            print("⚠ Falling back to DialoGPT-
medium...")            fallback_model = "microsoft/DialoGPT-medium"
self.tokenizer = AutoTokenizer.from_pretrained(fallback_model)
self.model = AutoModelForCausalLM.from_pretrained(fallback_model)
        self.pipeline = pipeline(                "text-
generation",                model=self.model,
tokenizer=self.tokenizer,
max_length=1024,                temperature=0.7,
do_sample=True,
pad_token_id=self.tokenizer.eos_token_id
        )
        print("☑ Fallback model loaded.")

    def analyze_requirements(self, text):
        prompt = f"You are a software requirement analysis assistant. Analyze the following
requirements and list key functionalities, ambiguities, and improvement
suggestions:\n\n{text}\n\nResponse:"        response =
self.pipeline(prompt)        result =
response[0]['generated_text'].split("Response:")[-1].strip()        return
result


    def generate_code(self, description):
        prompt = f"You are a software code generation assistant. Based on the following
description, generate Python code:\n\n{description}\n\nCode:"
response = self.pipeline(prompt)        result =
```

```
response[0]['generated_text'].split("Code:")[-1].strip()        return
result


# 📄 PDF Text Extraction Function def
extract_text_from_pdf(file_obj):
reader = PyPDF2.PdfReader(file_obj)
   text = ""     for page in
reader.pages:
      text += page.extract_text() or ""
return text


# 💻 Gradio Interface Builder def
create_gradio_interface():     with
gr.Blocks(title="SmartSDLC-AI")
as app:

      gr.HTML("<h1 style='text-align:center;'>🛠️ SmartSDLC-AI</h1><p style='text-
align:center;'>AI-powered Requirement Analysis & Code Generation Assistant</p>")


      with gr.Tabs():

         # 📝 Requirement Analysis Tab
with gr.Tab("📝 Requirement Analysis"):
with gr.Row():
               pdf_input = gr.File(label="Upload PDF Requirements")
text_input = gr.Textbox(label="Or Enter Requirements Prompt", lines=6)
```

```
analyze_btn = gr.Button("Analyze Requirements")                analysis_output =
gr.Textbox(label="Requirement Analysis Result", lines=12)


        # 💻 Code Generation Tab
with gr.Tab("💻 Code Generation"):
            code_desc_input = gr.Textbox(label="Describe the Functionality for Code
Generation", lines=6)                generate_code_btn = gr.Button("Generate Code")
code_output = gr.Code(label="Generated Python Code", language="python")


    # Requirement Analysis Function        def
handle_analysis(pdf_file, prompt_text):
        if pdf_file:
            text = extract_text_from_pdf(pdf_file)
elif prompt_text.strip():
            text = prompt_text
else:
            return "❗ Please upload a PDF or enter requirement text."
result = smart_sdlc_ai.analyze_requirements(text)            return
result


    analyze_btn.click(fn=handle_analysis, inputs=[pdf_input, text_input],
outputs=analysis_output)


    # Code Generation Function
def handle_code_generation(desc):
```

```
        if not desc.strip():

            return "❗ Please enter a description for code generation."

result = smart_sdlc_ai.generate_code(desc)          return result


    generate_code_btn.click(fn=handle_code_generation, inputs=code_desc_input,
outputs=code_output)


    gr.HTML("<p style='text-align:center; color:gray;'>🏵 Powered by IBM Granite AI |
SmartSDLC-AI for Modern Development</p>")


  return app


# 🚀 Run Application if

__name__ == "__main__":

print("🚀 SmartSDLC-AI

Initializing...")

smart_sdlc_ai =

SmartSDLC_AI()    iface =

create_gradio_interface()

print("🌐 Launching

SmartSDLC-AI with public

link...")

iface.launch(share=True)
```

4. Setup Instructions

♦ **Prerequisites**

Before starting the SmartSDLC project in Google Colab, ensure the following are available:

- **Google Account** to access and run Google Colab notebooks
- **IBM Cloud Account** with access to **Watsonx.ai** and the **Granite 3.2 Instruct model**
- IBM Cloud API Key with access rights to Watsonx foundation model
- Colab-compatible Python environment (Colab default: Python 3.10+)
- Required Python packages (listed below)

♦ Installation & Configuration Steps in Google Colab

You can run the full project pipeline inside a **Google Colab notebook**. Here's a step-by-step guide:

1. Set Up Required Python Packages

Install required packages in your Colab environment:

```
!pip install transformers torch gradio accelerate bitsandbytes PyPDF2
```

5. Folder Structure

Since **SmartSDLC** was developed entirely within a **Google Colab notebook**, the project does not follow a conventional file/folder structure (e.g., separate /client and /server directories). Instead, the entire system is organized into **modular notebook cells**, each representing logical components of the SDLC automation pipeline.

➢ Notebook-Based Architecture Overview

Instead of folders, the Colab notebook is divided into the following logical sections (cell groups):

1. PDF Upload & Requirement Extraction

- Uses PyMuPDF to read PDF content
- Extracts and preprocesses raw requirement text

2. IBM Granite Model Integration

- Authenticates with **Watsonx.ai** using the ibm-watson-machine-learning SDK
- Calls **Granite v3.2 Instruct** for various tasks:
  - o Classifying text into SDLC phases
  - o Generating code o Bug fixing o Test case generation o Summarizing code

3. Classification & User Story Generation

- Splits extracted text into sentences
- Prompts Granite to classify each sentence
- Optionally formats output into structured user stories

4. Code Processing Modules

• Separate cells for:
o Code generation from natural language o

Bug fixing in Python/JavaScript code o Test case creation o Code summarization

5. Chatbot Assistant (Optional)

- Implements a simple chatbot interface (text input/output)
- Routes user questions about SDLC to Granite with tailored prompts

6. Output Display and Export

- Displays results (code, test cases, summaries) directly in Colab
- Optionally saves outputs to Google Drive or downloads as files

6. Running the Application

The **SmartSDLC application is executed within a Google Colab notebook**, eliminating the need for local server setup. There are **no traditional frontend (npm start) or backend servers** involved.

❖ To run the application:

Step 1: Open the Google Colab notebook

Ensure you are logged into your Google account. Open the notebook from Google Drive or GitHub (if hosted).

Step 2: Run all cells in sequence

The notebook is divided into logical sections:

- Upload and extract PDF requirements
- Authenticate and connect to IBM Granite model

- Classify and process SDLC phases
- Generate code, fix bugs, write tests, summarize code
- Interact with the floating chatbot assistant (optional)

Step 3: View outputs directly in the notebook

- Extracted and classified requirements will be displayed in structured format
- AI-generated code, test cases, summaries, and bug fixes are printed below relevant cells
- No additional setup or local server is required

7. API Documentation:

SmartSDLC runs entirely within a **Google Colab notebook** and directly interacts with **IBM Watsonx's Granite v3.2 Instruct model** using the **IBM Watson Machine Learning Python SDK**. It does **not expose any public REST API endpoints**, but it internally follows a modular, function-based structure for various tasks.

8. Authentication:

♦ **Hugging Face API Key Authentication**

SmartSDLC uses **Hugging Face-hosted IBM Granite v3.2 Instruct model**, accessed through the **Hugging Face Inference API**. Authentication is handled via a **personal API key** provided by Hugging Face.

This key allows authorized access to large language models hosted on the Hugging Face platform without managing a complex user login system.

- **How It Works in the Project:**

You authenticate by passing the API key as a Bearer token in the HTTP request headers:

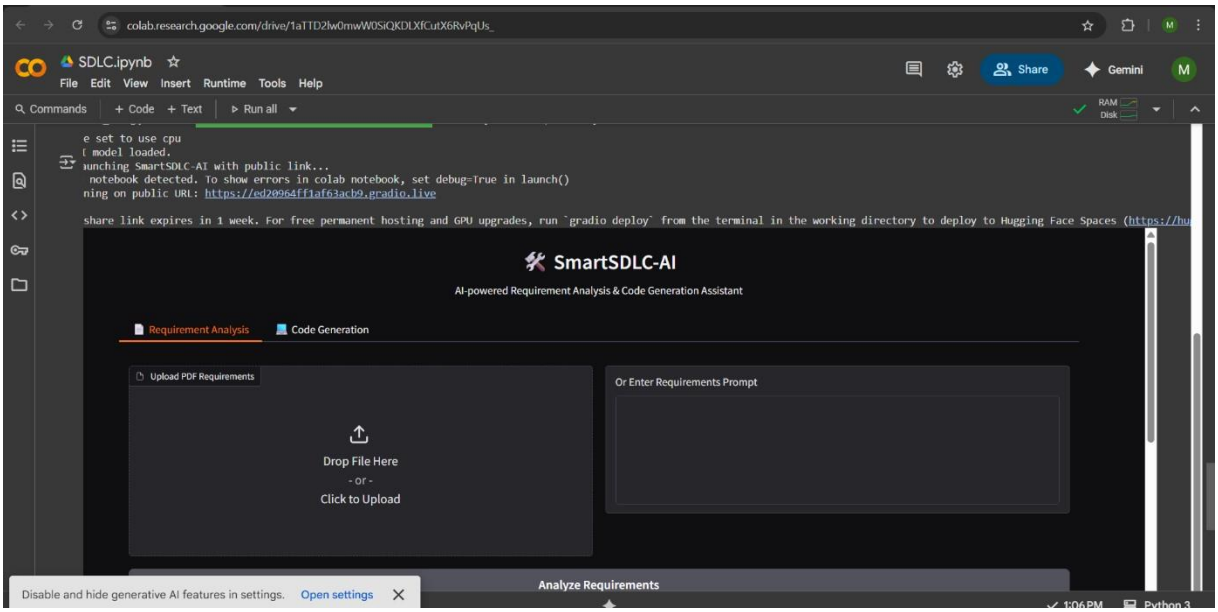
python
CopyEdit
import requests

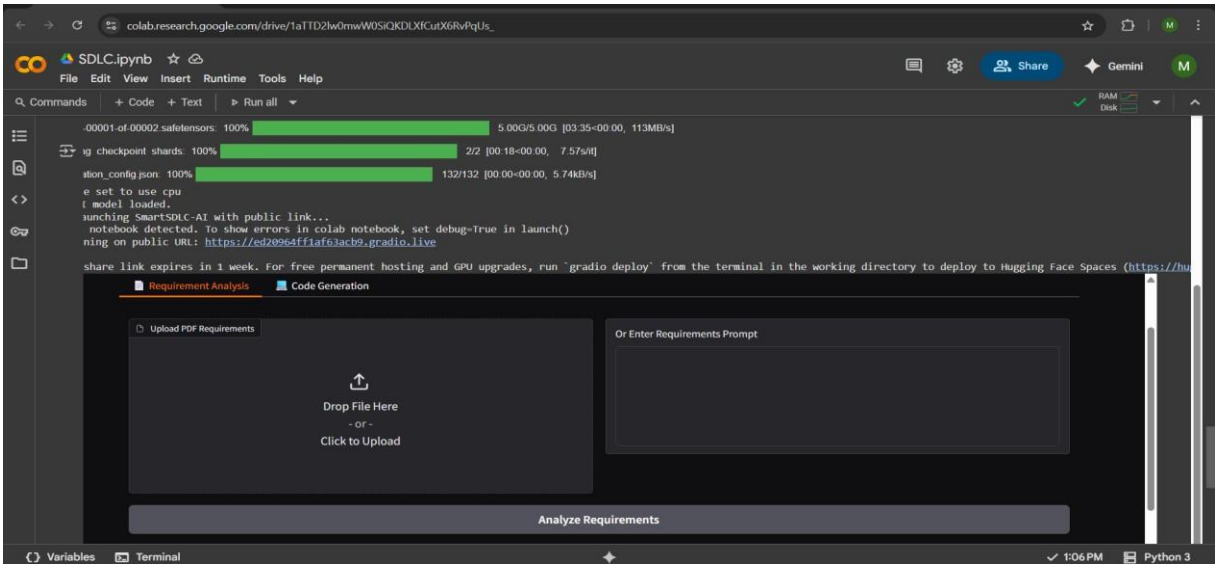
API_URL = "https://api-inference.huggingface.co/models/ibm/granite-3b-instruct" headers = {
"Authorization": f"Bearer YOUR_HUGGINGFACE_API_KEY"
}

response = requests.post(API_URL, headers=headers, json={"inputs": prompt})

9. User Interface:

SmartSDLC is designed to run in an interactive **Google Colab notebook**, where each module functions as a logically separated UI block. Users interact with the system by uploading files, entering text, and viewing AI-generated outputs directly within the notebook interface.





10. Testing:

➢ Testing Strategy

SmartSDLC-AI is tested through **manual testing and functional validation** using the Gradio interface. Each module is designed as a user-facing interactive block, allowing rapid iteration and debugging.

What Was Tested:

- **PDF Parsing**: Verified extraction of text from diverse PDF formats using PyPDF2.
- **Model Output Validation**:

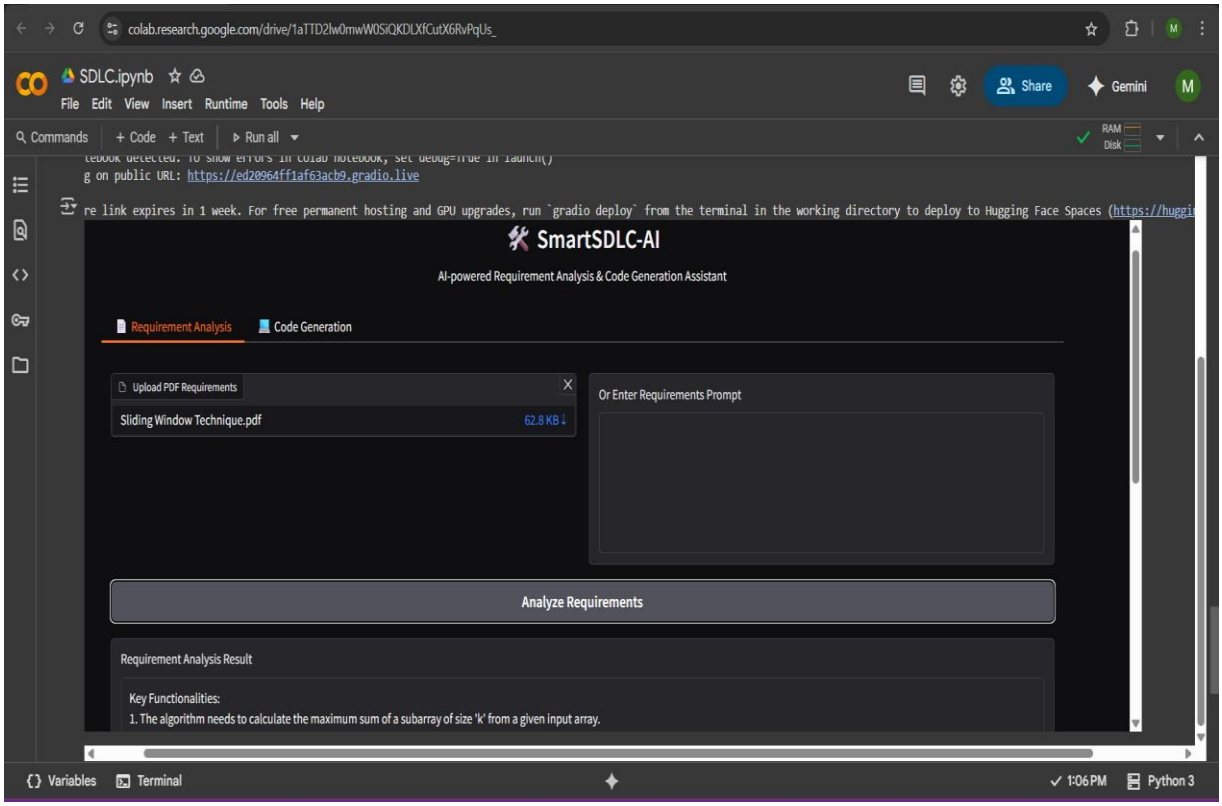
o Requirement analysis prompts yield structured, relevant functional outputs. o Code generation prompts produce working Python code for typical tasks.

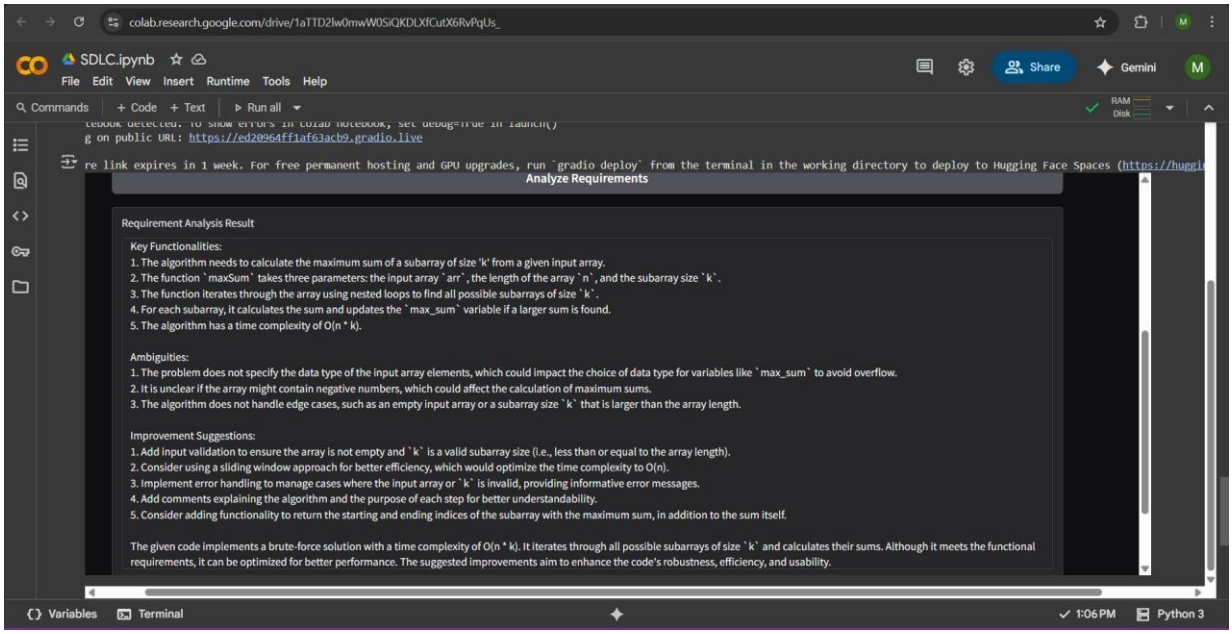
- **Fallback Model Handling**: Ensured system remains operational with a secondary model (DialoGPT-medium) when the IBM Granite model fails to load.

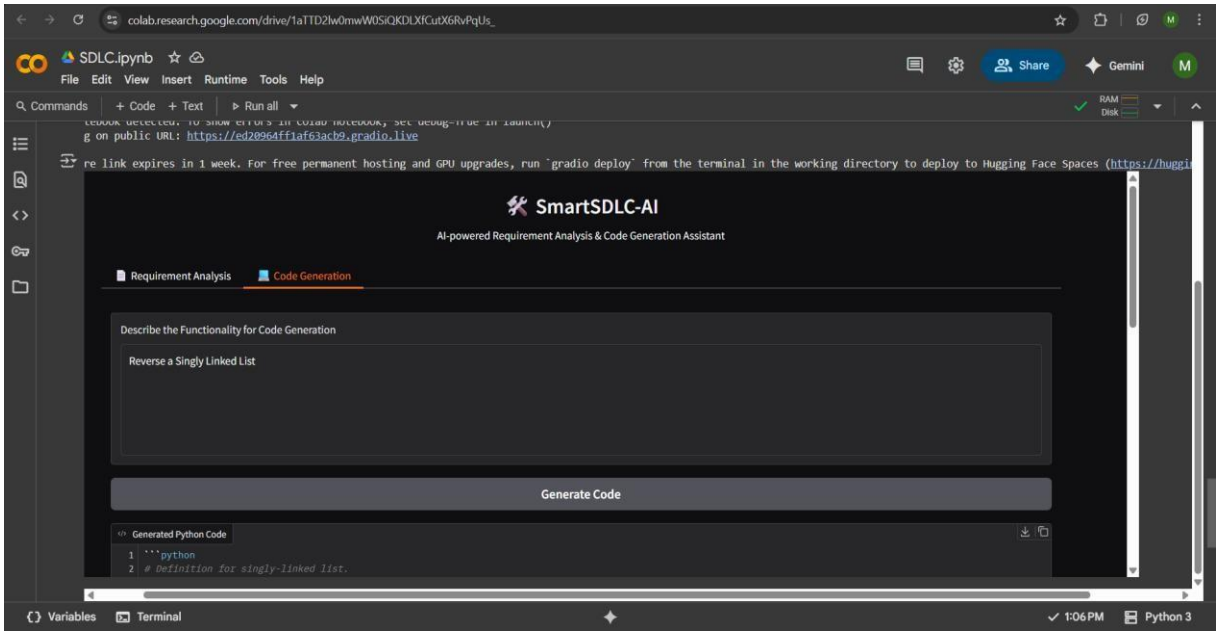
➢ Tools Used:

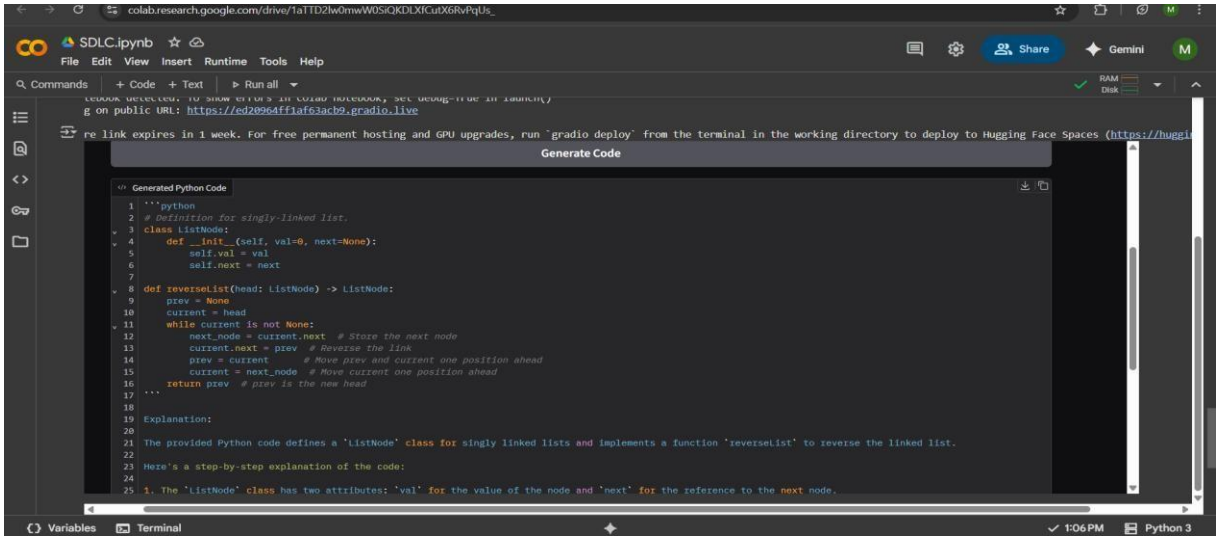
- Google Colab runtime
- Gradio's live UI for visual verification
- Print/log statements for model loading/debug fallback

11. Output:




colab.research.google.com/drive/1aTTD2lw0mwW0SiQKDLXfCutX6RvPqUs_
SDLC.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
g on public URL: https://ed20964ff1af63acb9.gradio.live
re link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggi
Analyze Requirements
Requirement Analysis Result
Key Functionalities:
1. The algorithm needs to calculate the maximum sum of a subarray of size 'k' from a given input array.
2. The function `maxSum` takes three parameters: the input array `arr`, the length of the array `n`, and the subarray size `k`.
3. The function iterates through the array using nested loops to find all possible subarrays of size `k`.
4. For each subarray, it calculates the sum and updates the `max_sum` variable if a larger sum is found.
5. The algorithm has a time complexity of O(n * k).
Ambiguities:
1. The problem does not specify the data type of the input array elements, which could impact the choice of data type for variables like `max_sum` to avoid overflow.
2. It is unclear if the array might contain negative numbers, which could affect the calculation of maximum sums.
3. The algorithm does not handle edge cases, such as an empty input array or a subarray size `k` that is larger than the array length.
Improvement Suggestions:
1. Add input validation to ensure the array is not empty and `k` is a valid subarray size (i.e., less than or equal to the array length).
2. Consider using a sliding window approach for better efficiency, which would optimize the time complexity to O(n).
3. Implement error handling to manage cases where the input array or `k` is invalid, providing informative error messages.
4. Add comments explaining the algorithm and the purpose of each step for better understandability.
5. Consider adding functionality to return the starting and ending indices of the subarray with the maximum sum, in addition to the sum itself.
The given code implements a brute-force solution with a time complexity of O(n * k). It iterates through all possible subarrays of size `k` and calculates their sums. Although it meets the functional requirements, it can be optimized for better performance. The suggested improvements aim to enhance the code's robustness, efficiency, and usability.
Variables
Terminal
1:06 PM
Python 3


colab.research.google.com/drive/1aTTD2lw0mwW0SiQKDLXfCutX6RvPqUs_
SDLC.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
g on public URL: https://ed20964ff1af63acb9.gradio.live
re link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggi
SmartSDLC-AI
AI-powered Requirement Analysis & Code Generation Assistant
Requirement Analysis
Code Generation
Describe the Functionality for Code Generation
Reverse a Singly Linked List
Generate Code
Generated Python Code
```python
# Definition for singly-linked list.
Variables
Terminal
1:06 PM
Python 3


colab.research.google.com/drive/1aTTD2lw0mwW0SiQKDLXfCutX6RvPqUs_
SDLC.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
g on public URL: https://ed20964ff1af63acb9.gradio.live
re link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggi
Generate Code
Generated Python Code
```python
# Definition for singly-linked list.
class ListNode:
def __init__(self, val=0, next=None):
self.val = val
self.next = next
def reverseList(head: ListNode) -> ListNode:
prev = None
current = head
while current is not None:
next_node = current.next  # Store the next node
current.next = prev  # Reverse the link
prev = current  # Move prev and current one position ahead
current = next_node  # Move current one position ahead
return prev  # prev is the new head
```
Explanation:
The provided Python code defines a 'ListNode' class for singly linked lists and implements a function 'reverseList' to reverse the linked list.
Here's a step-by-step explanation of the code:
1. The 'ListNode' class has two attributes: 'val' for the value of the node and 'next' for the reference to the next node.
Variables
Terminal
1:06 PM
Python 3

Demo Link:
```

12. Known Issues:

Here are some known bugs or limitations that may affect users or developers:

- **Model Load Time**:
  The IBM Granite 3.3-2B Instruct model is large and can take considerable time to load in Colab, especially on free-tier runtimes.
- **Memory Constraints in Google Colab**:
  Large models may cause memory overflows or slow performance on limited resources.
- **Fallback Model Simplicity**:
  The fallback model (DialoGPT-medium) is significantly less capable and may produce generic or unrelated outputs.
- **PDF Parsing Limitations**:
  Some PDFs, especially scanned images or non-standard encodings, may not be parsed accurately by PyPDF2.
- **Response Variability**:
  Generative AI outputs may vary between runs and may include irrelevant or partially complete code.
- **No Persistent Storage**:
  There's no backend database or file-saving mechanism, so all results are lost when the session ends unless manually saved.
- **No Authentication**:
  The app does not currently restrict access or protect the Hugging Face API key, which can be a security risk in shared environments.

13. Future Enhancements:

To improve usability, scalability, and feature richness, the following enhancements are recommended:

Functional Enhancements:

- **Add More SDLC Modules**:
  Incorporate additional tabs for:
  o Bug Fixing o Test Case Generation o Code Summarization o Deployment Suggestions

- **Custom Model Selector**:
  Let users choose between models (e.g., IBM Granite, GPT-4, Mistral) for different use cases.
- **Improve Prompt Engineering**:
  Use structured prompt templates and dropdowns to tailor AI responses more precisely.

Application & UI Improvements:

- **Persistent Storage**:
  Add integration with cloud storage or databases (e.g., Firebase or MongoDB) to save user inputs and outputs.
- **Export Functionality**:
  Allow users to download generated code and requirement summaries in .txt, .py, or .pdf format.
- **Code Execution Cell**:
  Let users run generated Python code inside the app using secure sandboxing (e.g., Code Interpreter or subprocess).
- **Enhanced Error Handling**:
  Provide clearer error messages, especially for model loading, PDF parsing, or API rate limits.

Security Enhancements:

- **API Key Protection**:
  Store the Hugging Face API key in environment variables or use OAuth-secured proxy to avoid exposure in notebooks.
- **User Authentication (Optional)**:
  Add login functionality if deployed publicly, especially when saving user data.

Deployment Options:

- **Dockerize the App**:
  Containerize with Docker for easier deployment on IBM Cloud or other cloud platforms.
- **Host on a Web Server**:
  Convert the app from Colab-based to a fully hosted app using Flask + Gradio, deployed on IBM Cloud or Hugging Face Spaces.

Conclusion:

The SmartSDLC platform incorporates AI-powered intelligence into every stage of the Software Development Lifecycle—from requirement analysis to code generation, testing, bug fixes, and documentation—it marks a substantial improvement in the automation of this process.
Both technical and nontechnical people can effectively engage with SDLC duties because to the platform's modular architecture and user-friendly interface. The value of AI when used carefully within a development framework is demonstrated by features like requirement classification from PDFs, AI-generated user stories, code generation from natural language, auto test case generation, intelligent bug repair, and integrated chat assistance.

In summary, the SmartSDLC not only increases accuracy and productivity but also lays the groundwork for upcoming improvements like cloud deployment, version control, team collaboration, and CI/CD integration. With intelligent automation at its heart, it is a step toward creating developer-friendly, intelligent ecosystems that support the demands of contemporary agile development.

GITHUB URL : https://github.com/chaitanya165/smartsdlc-ai-enhanced-software-development-lifecycle

DEMO LINK : https://drive.google.com/file/d/1RYcIjkWoejc1oFhcuAOIzWsMr-MsVq-o/view?usp=drivesdk