SMARTSDLC WITH IBM

Project Documentation

1. Introduction

• Project Title: SMARTSDLC with IBM

• Team Leader: KANISHKA V

• Team Member: PRIYADHARSHINI R

• Team Member: MANUSHA M

• Team Member: DHANALAKSHMI V

• Team Member: AARTHI R

2. Project Overview

• Purpose:

Smart SDLC automates software development steps by analyzing requirements and generating code, reducing time and effort in SDLC.

• Features:

Requirement analysis from text/PDF (functional, non-functional, technical).

Multi-language code generation (Python, Java, JS, etc.).

AI-powered analysis using IBM Granite.

Simple Gradio UI with tabs for analysis & code.

PDF text extraction and organized output.

Deployable with shareable link.

3. Architecture

1. UI (Gradio): Upload PDF / enter requirements, choose language.

2. Text Extraction: Extract text from PDFs (PyPDF2).

3. AI Module: IBM Granite analyzes requirements & generates code.

4. Code Generation: Converts requirements into multi-language code.

5. Output: Displays results in the interface.

4. Setup Instructions

Prerequisites:

- Python Programming Knowledge
- Gradio Framework
- IBM Granite Model Access (via Hugging Face)
- Google Colab with T4 GPU
- GitHub Account Steps:

1. Access the Naan Mudhalvan Smart Internz portal.
2. Choose an IBM Granite model from Hugging Face.
3. Run the application in Google Colab with required libraries.
4. Upload final project files to GitHub.
5. Folder Structure

- app/ – Backend logic and integration.
- ui/ – Gradio app interface files.
- citizen_ai.py – Main application file.
- model_loader.py – Handles IBM Granite model integration.
- dashboard.py – Visualization of citizen feedback.

6. Running the Application

1. Open Google Colab and load the project notebook.
2. Install dependencies and configure runtime with GPU.
3. Run the notebook cells to start the Gradio app.

4.Access the provided link to interact with Citizen AI.

7. API Documentation

Requirement_analysis(pdf, text) → Extracts functional, non-functional & technical requirements.

Code_generation(prompt, language) → Generates code in selected programming language.

Extract_text_from_pdf(pdf) → Extracts text from PDF.

Generate_response(prompt, max_length) → Returns AI-generated text response.

8.Authentication

Access: Currently, the API does not require authentication; anyone with access to the app can use it.

Future Enhancement: Can add API keys or token-based authentication for secure access.

9.User Interface

Gradio-based tabbed interface with two main tabs:

1. Code Analysis: Upload PDF or enter requirements to analyze functional, non-functional, and technical specifications.

2. Code Generator: Enter requirement text and select programming language to generate code.

Interactive and easy-to-use, outputs displayed directly in the interface.

10.Testing

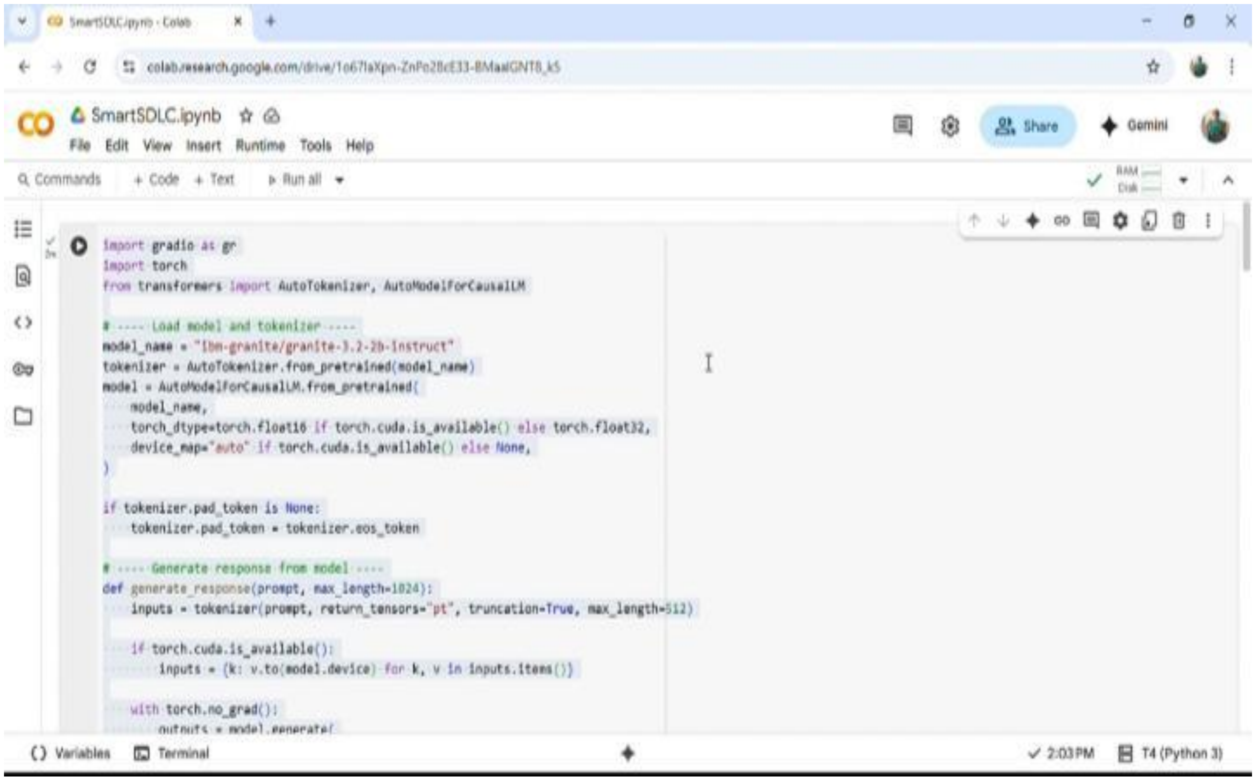
Testing included:

Unit Testing: Test individual functions like requirement_analysis, code_generation, and extract_text_from_pdf.

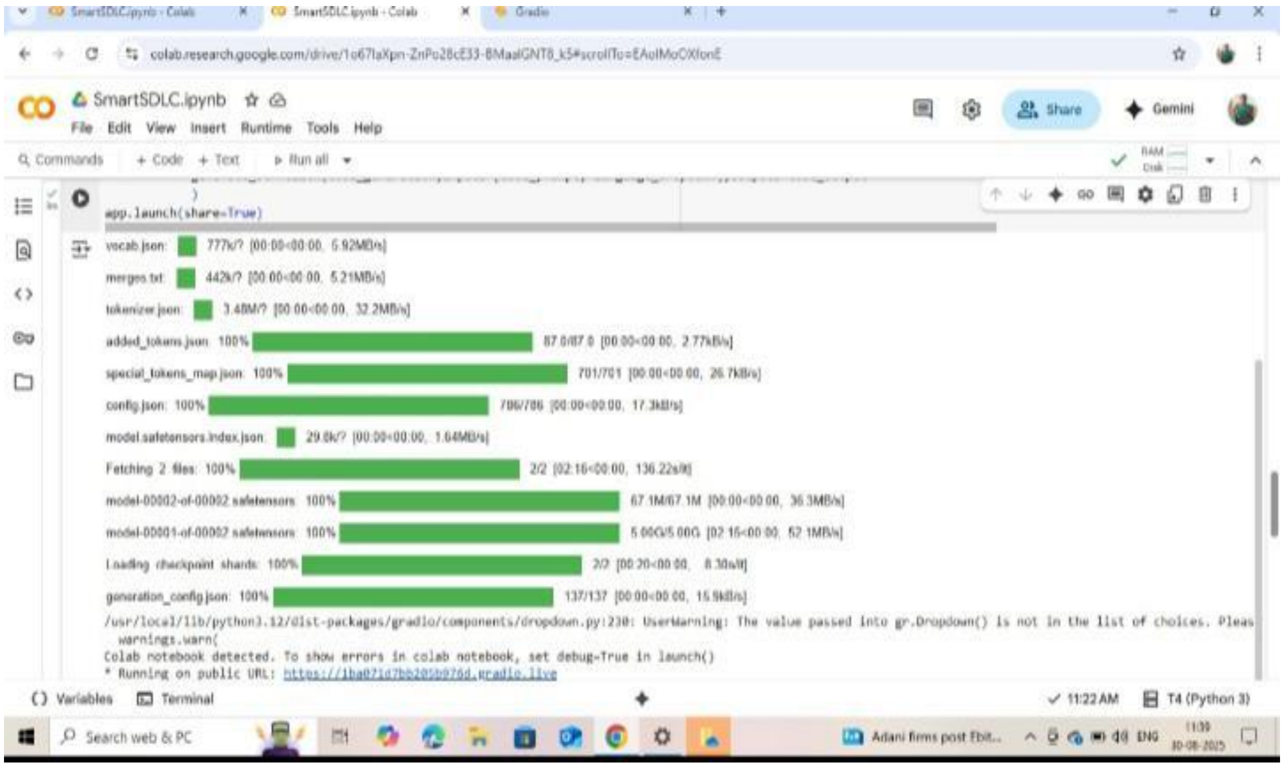
Integration Testing: Ensure PDF upload, AI analysis, and code generation work together correctly.

UI Testing: Verify Gradio interface buttons, inputs, and outputs function properly.

11.Screenshots

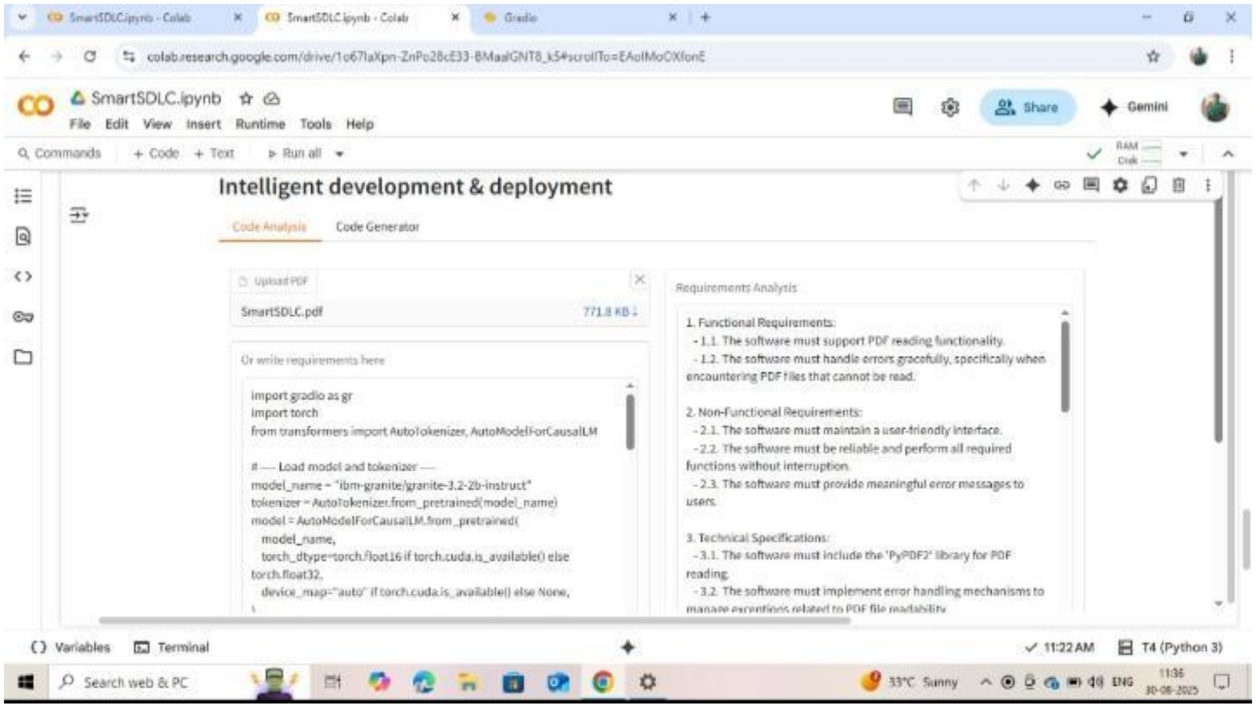

SmartSDLC.ipynb - Colab
colab.research.google.com/drive/1o67laXpn-ZnPo28cE33-BMaalGNT8_k5
SmartSDLC.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM
Disk
Variables
Terminal
2:03 PM
T4 (Python 3)

```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# ---- Load model and tokenizer ----
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None,
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# ---- Generate response from model ----
def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model.generate(
```



```
app.launch(share=True)
```

```
vocab.json:  777k/? [00:00<00:00,  5.92MB/s]
merges.txt:  442k/? [00:00<00:00,  5.21MB/s]
tokenizer.json:  3.48M/? [00:00<00:00,  32.2MB/s]
added_tokens.json: 100%  87.0/87.0 [00:00<00:00,  2.77kB/s]
special_tokens_map.json: 100%  701/701 [00:00<00:00,  26.7kB/s]
config.json: 100%  786/786 [00:00<00:00,  17.3kB/s]
model.safetensors.index.json:  29.8k/? [00:00<00:00,  1.64MB/s]
Fetching 2 files: 100%  2/2 [02:16<00:00, 136.22s/it]
model-00002-of-00002.safetensors: 100%  67.1M/67.1M [00:00<00:00,  36.3MB/s]
model-00001-of-00002.safetensors: 100%  5.00G/5.00G [02:15<00:00,  52.1MB/s]
Loading checkpoint shards: 100%  2/2 [00:20<00:00,  8.30s/it]
generation_config.json: 100%  137/137 [00:00<00:00,  15.9kB/s]
/usr/local/lib/python3.12/dist-packages/gradio/components/dropdown.py:230: UserWarning: The value passed into gr.Dropdown() is not in the list of choices. Pleas
  warnings.warn(
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://1ba071d7bb205b976d.gradio.live
```



Intelligent development & deployment

Code Analysis | Code Generator

Upload PDF: SmartSDLC.pdf 771.8 KB

Or write requirements here

```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# —— Load model and tokenizer ——
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
   model_name,
   torch_dtype=torch.float16 if torch.cuda.is_available() else
torch.float32,
   device_map="auto" if torch.cuda.is_available() else None,
)
```

Requirements Analysis

1. Functional Requirements:
  - 1.1. The software must support PDF reading functionality.
  - 1.2. The software must handle errors gracefully, specifically when encountering PDF files that cannot be read.

2. Non-Functional Requirements:
  - 2.1. The software must maintain a user-friendly interface.
  - 2.2. The software must be reliable and perform all required functions without interruption.
  - 2.3. The software must provide meaningful error messages to users.

3. Technical Specifications:
  - 3.1. The software must include the 'PyPDF2' library for PDF reading.
  - 3.2. The software must implement error handling mechanisms to manage exceptions related to PDF file readability

12.known issues

PDF extraction may fail for scanned or complex PDFs.

AI-generated code may require manual review for accuracy.

Some programming languages may produce incomplete or unoptimized code.

13.Future Enhancements

Add API key/token authentication for secure access.

Improve PDF parsing for scanned/complex documents.

Support more programming languages and frameworks.

Integrate automated testing for generated code.