Project Title 1. Introduction Project title: Smart SDLC _AI-Enhanced Software Development Lifecycle

Team Leader:Thamarai Ariyanatchi R (23scs29)

Team member: Ramyakrishnan P (23scs26)

Team member: Rohini K (23scs27)

Team member: Sangeetha S (23scs28)

Team member: Vimala S (23scs30)

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

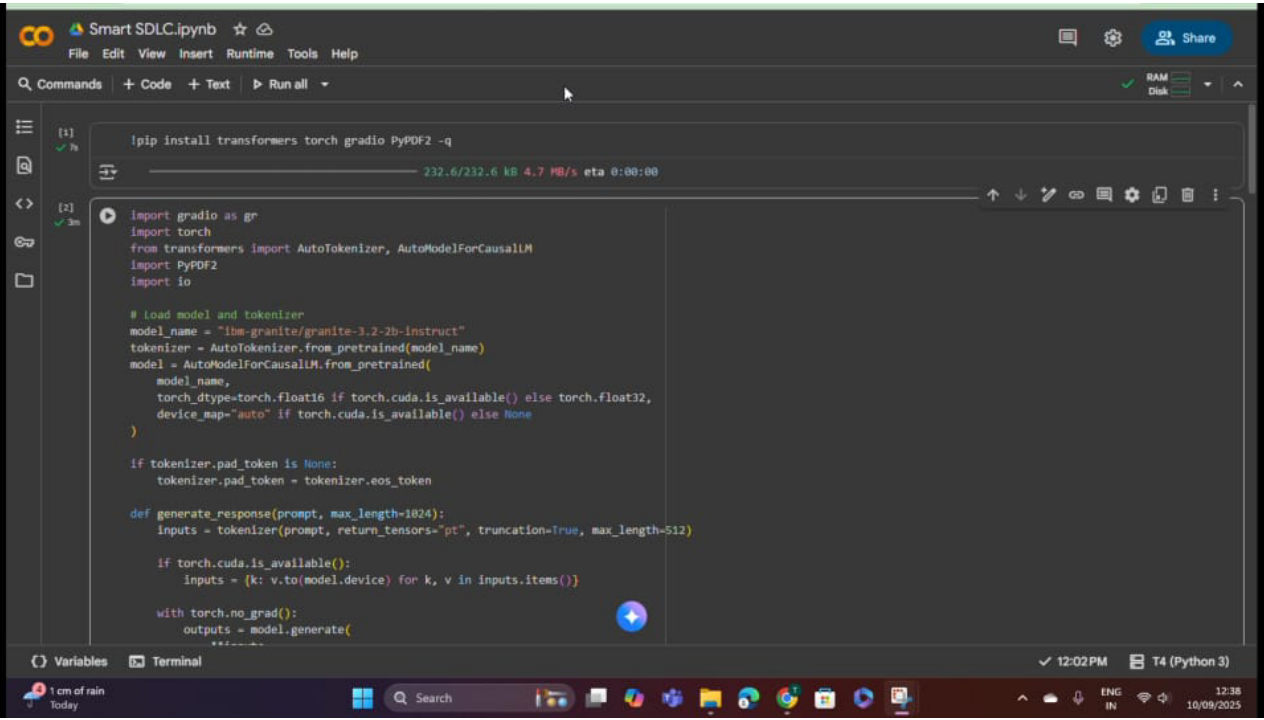
10. Testing

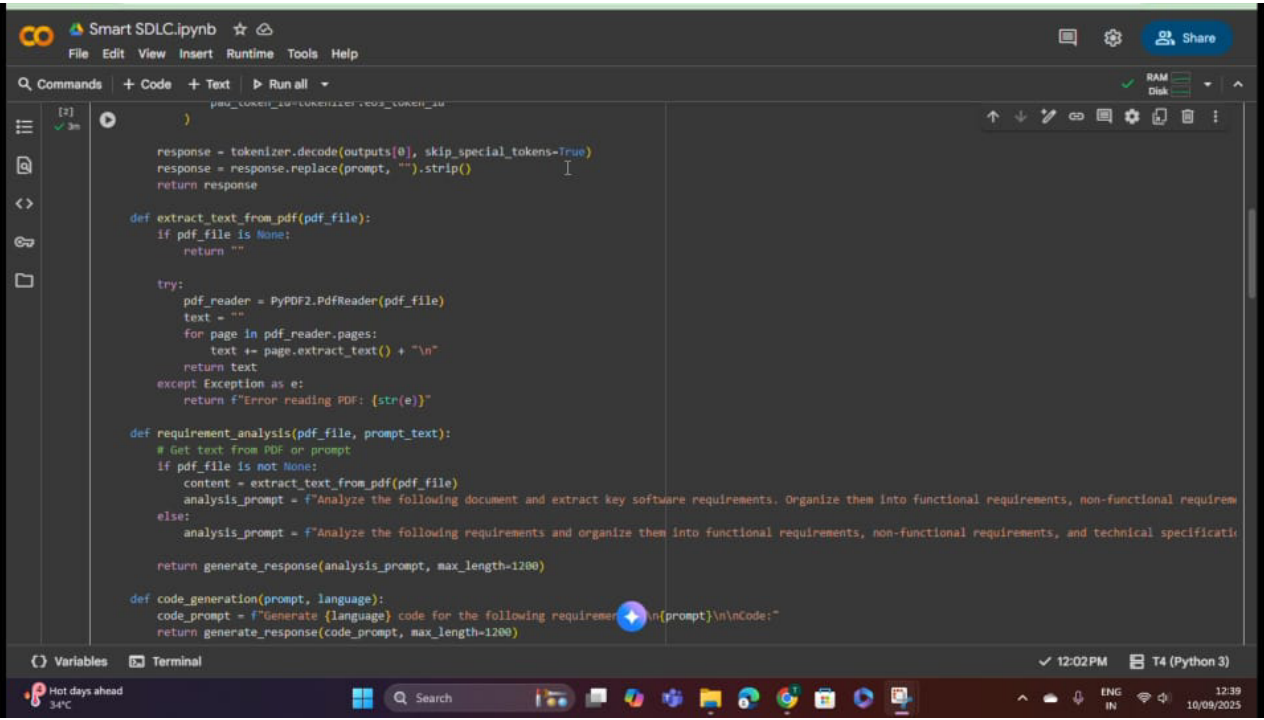
- Unit testing for model prompt functions and utility code recommended

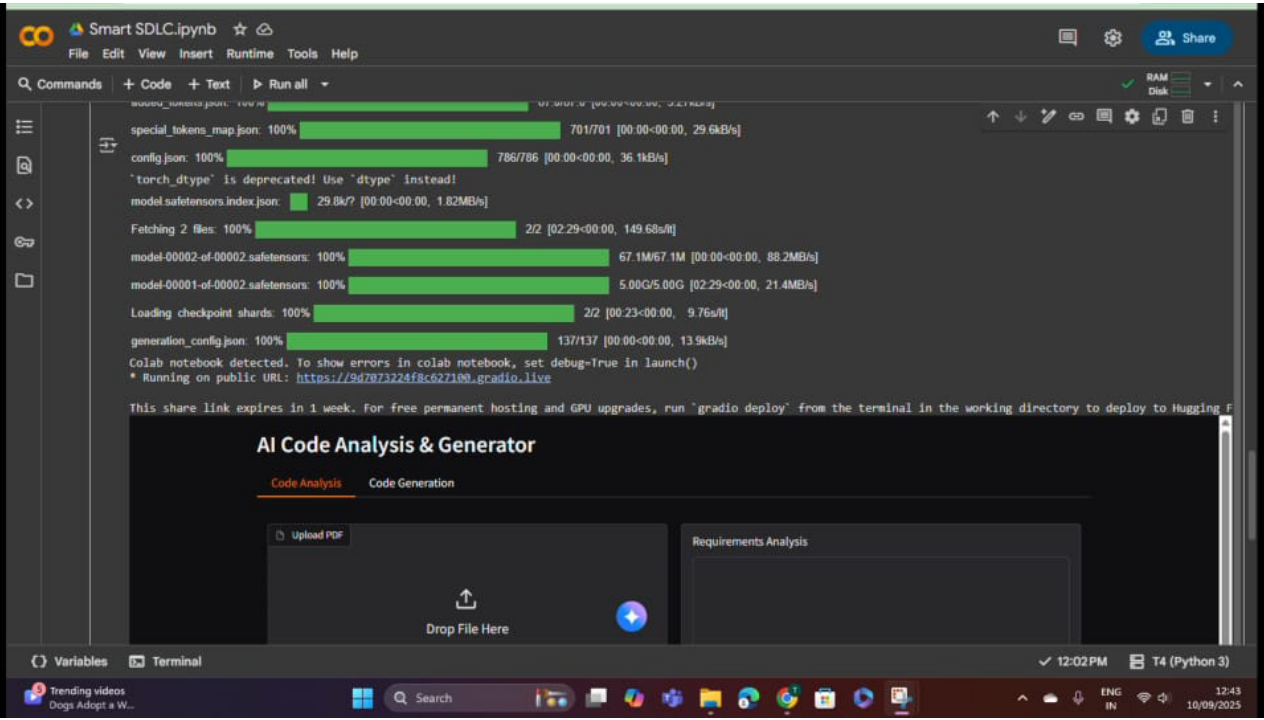
- Manual UI testing for uploads, responses, and output format

- Handle edge cases (malformed/large documents, empty inputs).

This structure closely follows your organizational template, providing technical teams with clear, concise, and process-aligned documentation for the Gradio LLM-based application.


Smart SDLC.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
[1]
!pip install transformers torch gradio PyPDF2 -q
232.6/232.6 kB 4.7 MB/s eta 0:00:00
[2]
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import PyPDF2
import io
# 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
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(
Variables
Terminal
12:02PM
T4 (Python 3)
1 cm of rain
Today
Search
ENG
IN
12:38
10/09/2025


Smart SDLC.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
[2]
3m
)
response = tokenizer.decode(outputs[0], skip_special_tokens=True)
response = response.replace(prompt, "").strip()
return response
def extract_text_from_pdf(pdf_file):
if pdf_file is None:
return ""
try:
pdf_reader = PyPDF2.PdfReader(pdf_file)
text = ""
for page in pdf_reader.pages:
text += page.extract_text() + "\n"
return text
except Exception as e:
return f"Error reading PDF: {str(e)}"
def requirement_analysis(pdf_file, prompt_text):
# Get text from PDF or prompt
if pdf_file is not None:
content = extract_text_from_pdf(pdf_file)
analysis_prompt = f"Analyze the following document and extract key software requirements. Organize them into functional requirements, non-functional requirem
else:
analysis_prompt = f"Analyze the following requirements and organize them into functional requirements, non-functional requirements, and technical specificati
return generate_response(analysis_prompt, max_length=1200)
def code_generation(prompt, language):
code_prompt = f"Generate {language} code for the following requirement
{prompt}\n\nCode:"
return generate_response(code_prompt, max_length=1200)
Variables
Terminal
12:02PM
T4 (Python 3)
Hot days ahead
34°C
Search
ENG
IN
12:39
10/09/2025


Smart SDLC.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
special_tokens_map.json: 100% 701/701 [00:00<00:00, 29.6kB/s]
config.json: 100% 786/786 [00:00<00:00, 36.1kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 1.82MB/s]
Fetching 2 files: 100% 2/2 [02:29<00:00, 149.68s/it]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:00<00:00, 88.2MB/s]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [02:29<00:00, 21.4MB/s]
Loading checkpoint shards: 100% 2/2 [00:23<00:00, 9.76s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 13.9kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://9d7073224f8c627100.gradio.live
This 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 to Hugging F
AI Code Analysis & Generator
Code Analysis
Code Generation
Upload PDF
Drop File Here
Requirements Analysis
Variables
Terminal
12:02PM
T4 (Python 3)
Trending videos
Dogs Adopt a W...
Search
ENG
IN
12:43
10/09/2025