# Smart SDLC  AI  enhanced software development life cycle

## Team Members

D.Akalya
A.Jeyasri
R.Gayathri
N.Devadharshini

## 1. INTRODUCTION

The Smart SDLC – AI Enhanced Software Development Lifecycle is an advanced framework that integrates Artificial Intelligence (AI) into each phase of the traditional software development process. This model builds upon existing methodologies like Waterfall, Agile, and DevOps by embedding AI-driven tools to automate tasks, optimize workflows, and enhance accuracy.

## 2. PROJECT OVERVIEW

The Smart SDLC is designed to revolutionize software development by providing intelligent assistance at every stage. It introduces AI-driven features such as conversation interfaces, policy summarization, resource forecasting, eco-friendly development tips, anomaly detection, KPI forecasting, multimodal input, and AI-powered dashboards.

Key Features:

• Conversational Interface – Enables natural communication between AI, developers, and clients.

• Policy Summarization – Ensures responsible AI usage and compliance.

• Resources Forecasting – Predicts time, cost, and resource allocation.

• Eco-Tip Generator – Suggests energy-efficient and sustainable development practices.

• Citizen Feedback Loop – Integrates continuous feedback from end-users.

• KPI Forecasting – Tracks and predicts project success metrics.

• Anomaly Detection – Detects irregularities and risks in the development process.

• Multimodal Input Support – Accepts text, voice, image, and sketch inputs.

• Streamlit to Gradio UI – Interactive, user-friendly, and AI-powered dashboards.

## 3. ARCHITECTURE

The architecture of Smart SDLC consists of two major components:

Front-end: AI-powered multimodal interface with dashboards, chatbots, and feedback forms.
Back-end: Cloud-based infrastructure integrated with AI/ML models, databases, and APIs.

Key Components:

• LLM Integration – Connects AI models to automate tasks like requirement analysis, coding, and testing.

• Vector Store – Retains project knowledge in embeddings for quick recall and reference.

• ML Modules – Specialized AI models for requirements, testing, forecasting, and anomaly detection.

## 4. SETUP INSTRUCTION

Prerequisites:

• Python 3.8+ installation

• Streamlit / Gradio for UI

• TensorFlow / PyTorch for ML models

• LangChain / Transformers for LLM integration

• Vector database (FAISS, Pinecone, Weaviate)

Installation Steps:

1. Install Python 3.8+

2. Install dependencies using 'pip install -r requirements.txt'

3. Configure vector database and API keys

4. Run backend server and frontend UI

## 5. AUTHENTICATION

Smart SDLC employs secure authentication mechanisms to prevent unauthorized access, safeguard sensitive project data, and ensure role-specific access. Security tokens and encrypted logins help build trust among stakeholders.

## 6. USER INTERFACE

The Smart SDLC user interface is designed to be intuitive and interactive. Dashboards provide real-time project tracking, while AI-powered chatbots allow seamless communication. The UI ensures collaboration, accessibility, and efficiency.

## 7. TESTING

Testing in Smart SDLC is AI-driven, automating the generation of test cases, monitoring system performance, and detecting defects. This results in higher quality assurance and faster debugging cycles.

## 8. KNOWN ISSUES

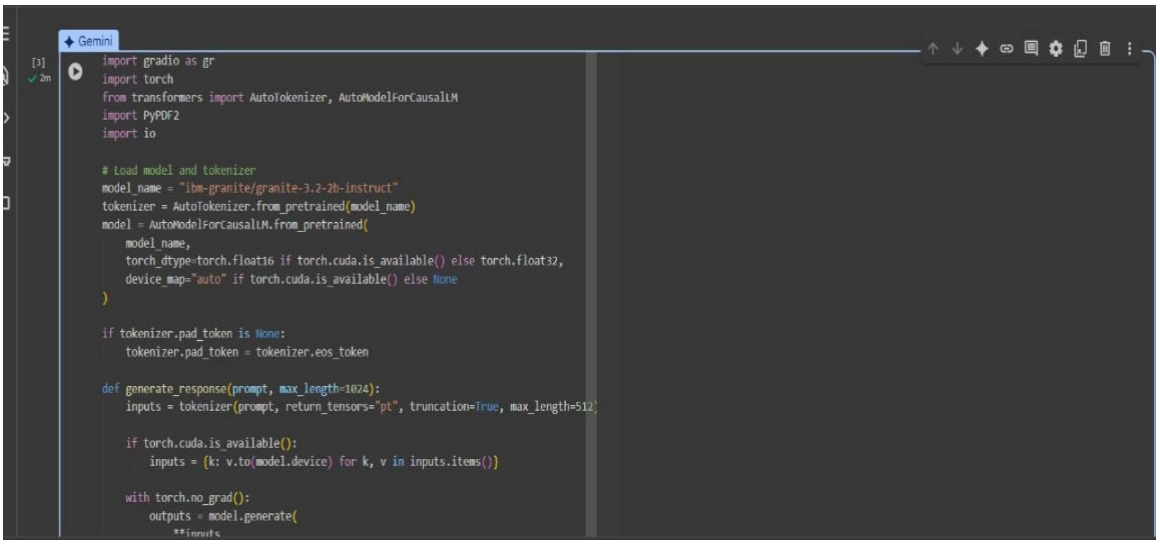
- Occasional delays in LLM response for large queries
- AI-generated code may require manual adjustments
- Multimodal input (voice, image) may face inconsistencies
- Vector store retrieval failures in rare cases
- System performance dependent on internet/cloud stability

## 9. FUTURE ENHANCEMENTS

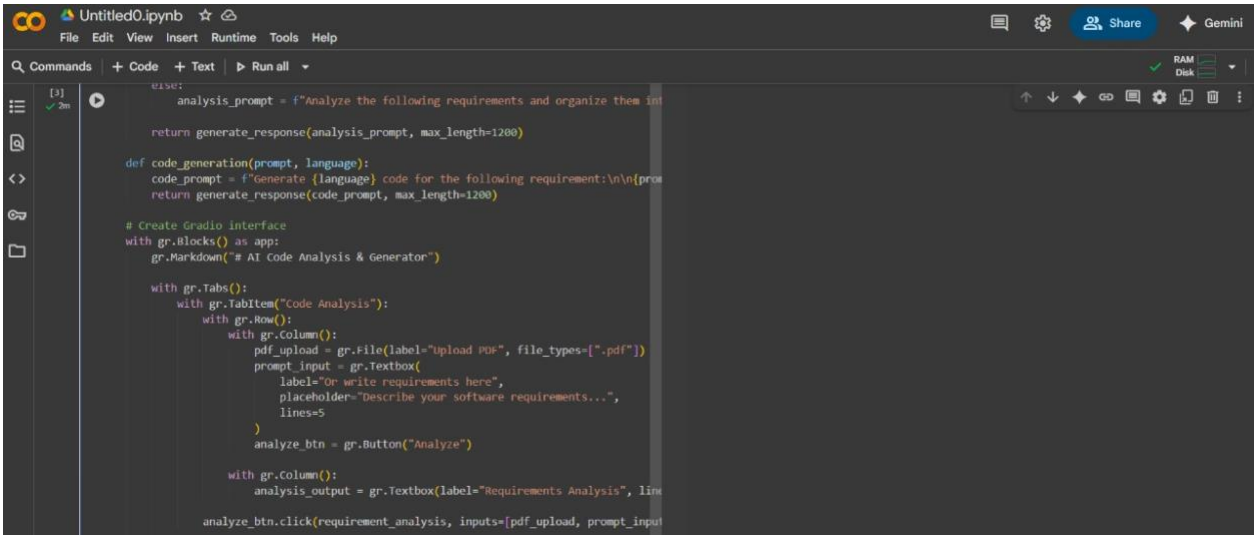
In the future, Smart SDLC can be enhanced with:

- Advanced LLMs for faster and more accurate results
- Real-time multi-user collaboration
- Enhanced multimodal input (voice, sketch, video)
- Predictive analytics for risk and cost estimation
- Automated deployment pipelines
- Improved security and scalability

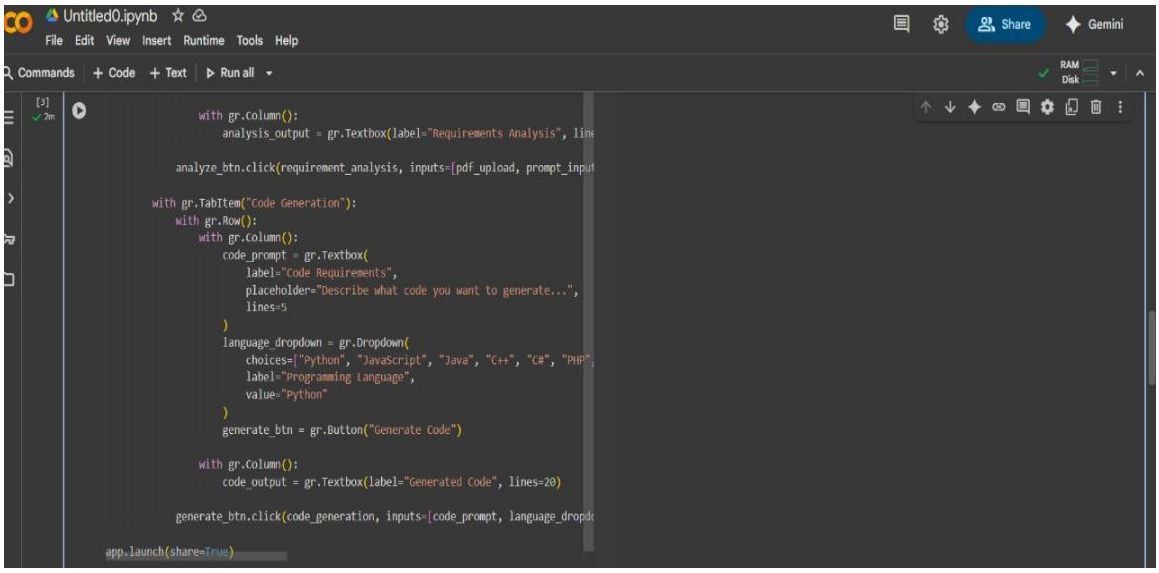
## 10.Project Screenshot

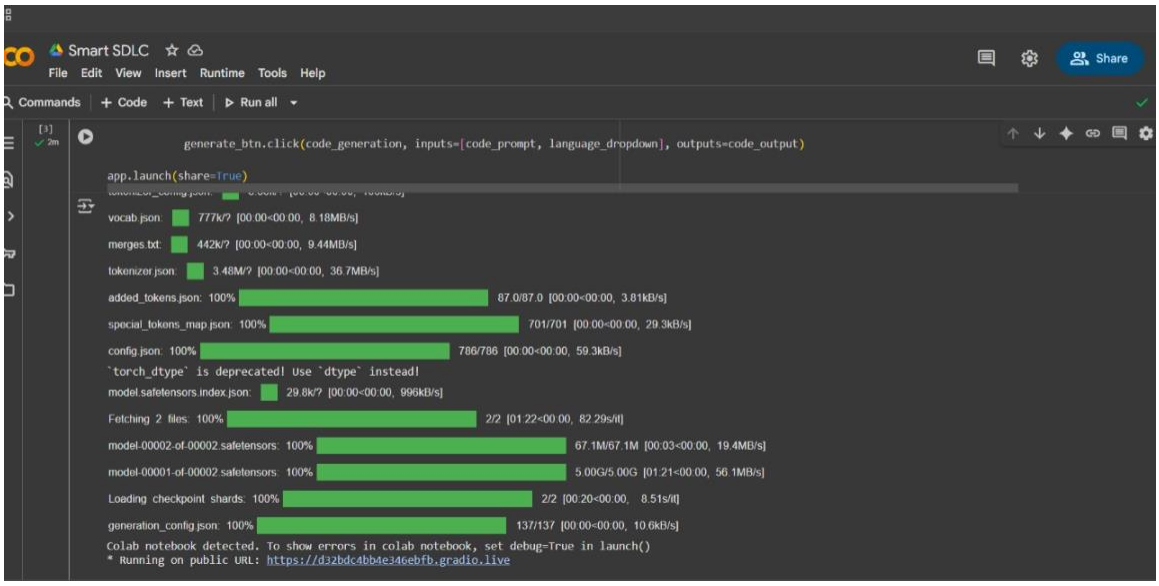


```
Gemini
[3]
✓ 2m
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import PyPDF2
import io

# 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(
            **inputs
```



```
Untitled0.ipynb
File Edit View Insert Runtime Tools Help
Share Gemini
Commands + Code + Text ▷ Run all
RAM Disk
[3]
✓ 2m
        else:
            analysis_prompt = f"Analyze the following requirements and organize them int

        return generate_response(analysis_prompt, max_length=1200)

    def code_generation(prompt, language):
        code_prompt = f"Generate {language} code for the following requirement:\n\n{pro
        return generate_response(code_prompt, max_length=1200)

    # Create Gradio interface
    with gr.Blocks() as app:
        gr.Markdown("# AI Code Analysis & Generator")

        with gr.Tabs():
            with gr.TabItem("Code Analysis"):
                with gr.Row():
                    with gr.Column():
                        pdf_upload = gr.File(label="Upload PDF", file_types=[".pdf"])
                        prompt_input = gr.Textbox(
                            label="Or write requirements here",
                            placeholder="Describe your software requirements...",
                            lines=5
                        )
                        analyze_btn = gr.Button("Analyze")

                    with gr.Column():
                        analysis_output = gr.Textbox(label="Requirements Analysis", lin

                    analyze_btn.click(requirement_analysis, inputs=[pdf_upload, prompt_input
```


Untitled0.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM
Disk
[3]
2m
with gr.Column():
analysis_output = gr.Textbox(label="Requirements Analysis", lin
analyze_btn.click(requirement_analysis, inputs=[pdf_upload, prompt_input
with gr.TabItem("Code Generation"):
with gr.Row():
with gr.Column():
code_prompt = gr.Textbox(
label="Code Requirements",
placeholder="Describe what code you want to generate...",
lines=5
)
language_dropdown = gr.Dropdown(
choices=["Python", "JavaScript", "Java", "C++", "C#", "PHP",
label="Programming Language",
value="Python"
)
generate_btn = gr.Button("Generate Code")
with gr.Column():
code_output = gr.Textbox(label="Generated Code", lines=20)
generate_btn.click(code_generation, inputs=[code_prompt, language_dropdo
app.launch(share=True)


Smart SDLC
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
[3]
2m
generate_btn.click(code_generation, inputs=[code_prompt, language_dropdown], outputs=code_output)
app.launch(share=True)
vocab.json: 777k/? [00:00<00:00, 8.18MB/s]
merges.txt: 442k/? [00:00<00:00, 9.44MB/s]
tokenizer.json: 3.48M/? [00:00<00:00, 36.7MB/s]
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 3.81kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 29.3kB/s]
config.json: 100% 786/786 [00:00<00:00, 59.3kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 996kB/s]
Fetching 2 files: 100% 2/2 [01:22<00:00, 82.29s/it]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:03<00:00, 19.4MB/s]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [01:21<00:00, 56.1MB/s]
Loading checkpoint shards: 100% 2/2 [00:20<00:00, 8.51s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 10.6kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://d32bdc4bb4e346ebfb.gradio.live

*Thank you*