SmartSDLC: AI-Enhanced Software Development Lifecycle

·Team Leader: HARIHARASUDAN.S
·Team member: GURUPRASANNA.S
·Team member: DENNIS LORDHU RAJ.A
·Team member: HARI.M

1. Introduction:

GreenSpark AI is an eco-focused AI-powered project that leverages IBM Granite models to enhance the Software Development Lifecycle (SDLC). It automates requirement gathering, code generation, testing, deployment, and documentation, while also providing sustainability-focused solutions such as carbon footprint estimation, policy summarization, eco-friendly lifestyle tips, and green technology ideas.

2. Project Overview:

The purpose of SmartSDLC (Smart Sustainable Development Life Cycle) is to create an AI-powered assistant that empowers individuals, communities, and policymakers to adopt more sustainable practices while simplifying complex decision-making processes. By integrating large language models with real-time document analysis and interactive features, SmartSDLC helps users generate eco-friendly lifestyle tips, summarize lengthy policy documents, estimate carbon footprints, and explore innovative green technologies. The system not only promotes awareness of environmental impact but also provides actionable insights, bridging the gap between technology, sustainability, and governance. Ultimately, SmartSDLC aims to foster a smarter and greener future by encouraging informed choices, sustainable development, and community engagement.

3. Architecture:

Frontend (Gradio):
The frontend is built with Gradio, offering a lightweight and interactive web-based interface. It provides a tab-based layout for eco tips, policy summarization, carbon footprint estimation, and green technology ideas. Users can upload documents, enter text, or describe activities, and instantly receive AI-generated outputs.
Backend (PyTorch + Transformers):
The backend leverages Hugging Face Transformers and PyTorch to handle natural language processing tasks. The IBM Granite model is integrated to generate human-like responses, summarize text, and provide sustainability recommendations.
Document Processing (PyPDF):
PDF files are processed using PyPDF, extracting raw text for further summarization and analysis. This enables quick policy insights without manual reading.
LLM Integration (IBM Granite 3.3-2B Instruct):
The IBM Granite LLM powers core AI functions such as summarization, eco-tip generation, innovation ideas, and carbon footprint estimation. Prompts are optimized to produce clear, structured, and actionable outputs.

4. Setup Instructions:

- · Python 3.9+ installed
- · Git installed → Download
- · Hugging Face account → Sign up
- · GitHub account → Sign up

5. Folder Structure:

```
📁 SmartSDLC
|
|— 📁 app/ # FastAPI backend logic
| |— 📁 api/ # Modular API routes
| | |— chat/ # Chat-related routes
| | |— feedback/ # Feedback routes
| | |— report/ # Report generation routes
| | |— vectorizer/ # Document vectorization routes
| |
| |— models/ # Database + Pydantic models
| |— integrations/ # External service integrations
|
|— 📁 ui/ # Streamlit frontend
| |— components/ # Card layouts, forms, widgets
| |— pages/ # Streamlit dashboard pages
|
|— smart_dashboard.py # Entry script for Streamlit dashboard
|— granite_llm.py # IBM Granite LLM integration
|— document_embedder.py # Document embeddings + Pinecone
|— kpi_file_forecaster.py # KPI forecasting (energy/water trends)
|— anomaly_file_checker.py # Anomaly detection in KPI data
|— report_generator.py # Sustainability report generator
|— app.py # Gradio Eco Assistant & Policy Analyzer
|
|— requirements.txt # Project dependencies
|— README.md # Documentation
|— .gitignore # Ignore unnecessary files
```

6. Running the Application:

- To start the project:
- Launch the FastAPI server to expose backend endpoints.
- Run the Streamlit dashboard to access the web interface.
- Navigate through pages via the sidebar.
- Upload documents or CSVs, interact with the chat assistant, and view outputs like Generate ECO Tips, Summarize Policy and Generate Green Tech Ideas.
- All interactions are real-time and use backend APIs to dynamically update the frontend.

7. Module Documentation:

- Eco Tips Generator: Provides actionable eco-friendly suggestions. - Policy Summarization: Extracts and summarizes environmental policies from PDFs or text. - Carbon Footprint Estimator: Estimates monthly CO■ emissions and reduction strategies. - Green Tech Ideas: Suggests innovative sustainable technologies.

8. Authentication:

Each API endpoint is tested and documented in Swagger UI for quick inspection and trial during development. Currently, the project runs in an open environment (no login required) for demonstration purposes.

For secure deployments, the following authentication methods can be integrated:

9. User Interface:

1. Main Header: Shows the app title "🌱 Eco Assistant & Policy Analyzer."

2. Tabbed Layout: Organizes features into four separate tabs.

3. Two-Column Design: Inputs are on the left, outputs on the right.

4. Eco Tips Generator Tab: Generates eco-friendly tips based on user keywords.

5. Policy Summarization Tab: Summarizes uploaded PDFs or pasted policy text.

6. Carbon Footprint Estimator Tab: Estimates user carbon footprint and suggests reductions.

10. Testing:

1. Model Loading Test: Verify the AI model and tokenizer load correctly without errors.

2. Eco Tips Functionality: Enter keywords and ensure eco-friendly tips are generated accurately.

3. Policy Summarization: Upload PDFs or paste text and confirm summaries display key points.

4. Carbon Footprint Estimation: Input daily/weekly activities and check if footprint and suggestions are reasonable.

5. Green Technology Ideas: Enter sector names and validate the generated innovative ideas.

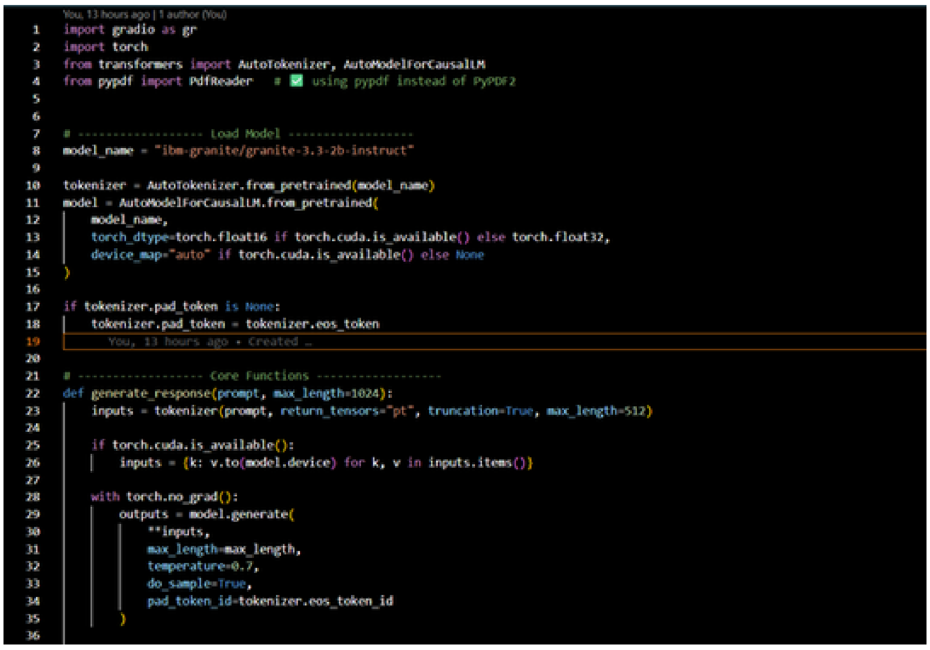
6. UI Interaction Test: Ensure all buttons trigger the correct functions and outputs appear in the corresponding textbox.

7. File Upload & Edge Cases: Test invalid PDFs, empty inputs, or very large text to confirm error handling works.

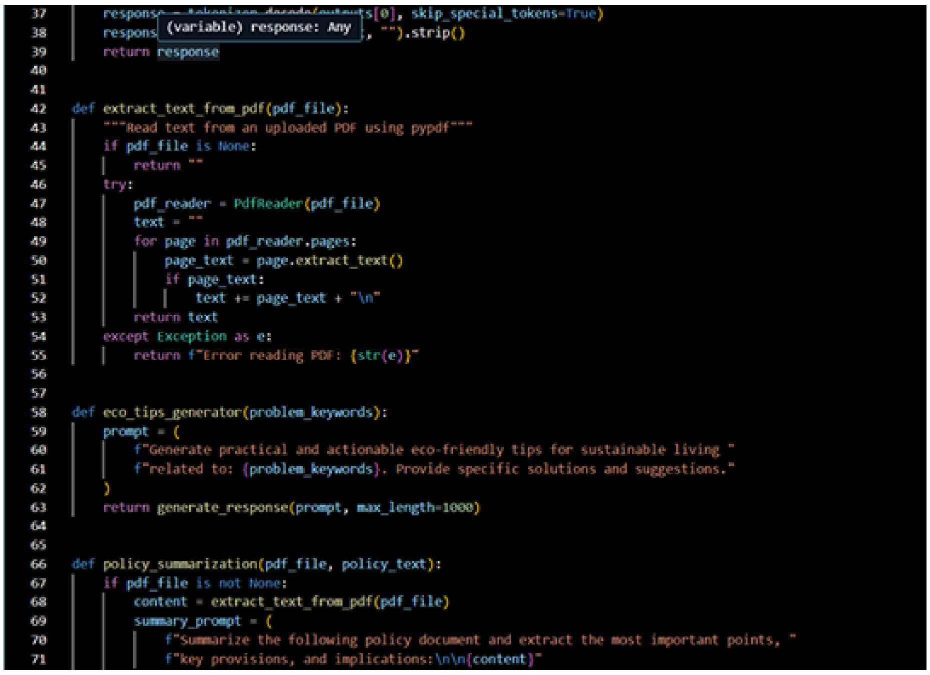
11. Screenshots & Outputs:

[Insert screenshots of Gradio interface, sample outputs, and code execution here]

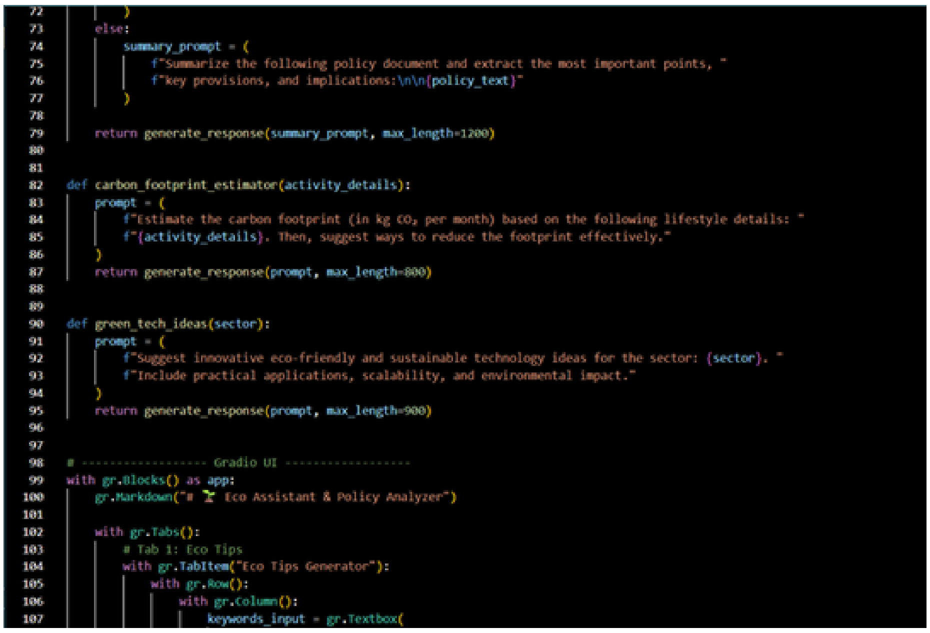
Program:



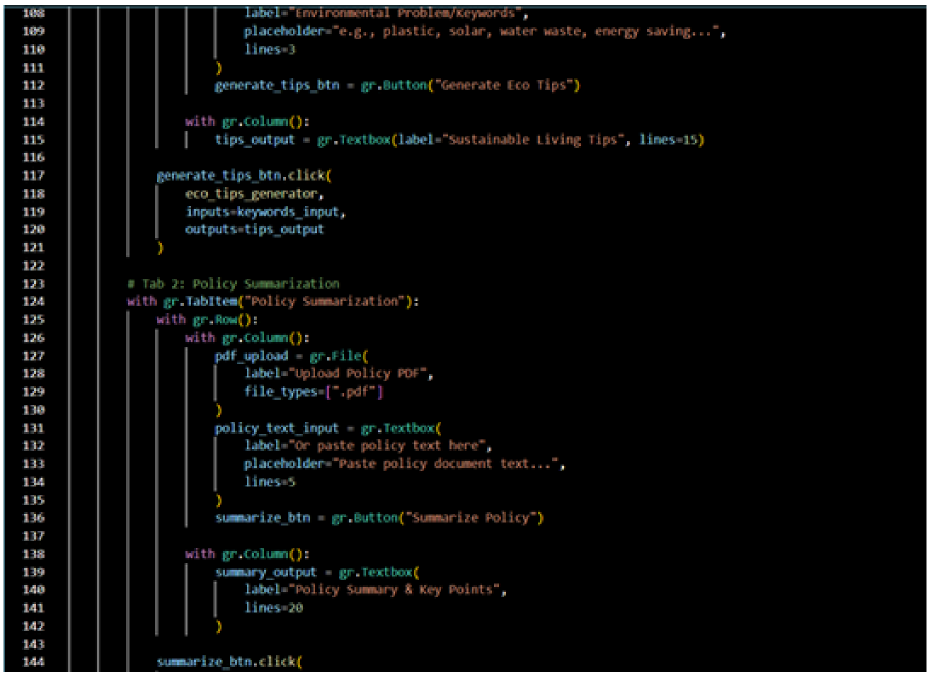
```python
You, 13 hours ago | 1 author (You)
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
from pypdf import PdfReader   # ✅ using pypdf instead of PyPDF2


# ------------------ Load Model ------------------
model_name = "ibm-granite/granite-3.3-2b-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
      You, 13 hours ago • Created …

# ------------------ Core Functions ------------------
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,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

```



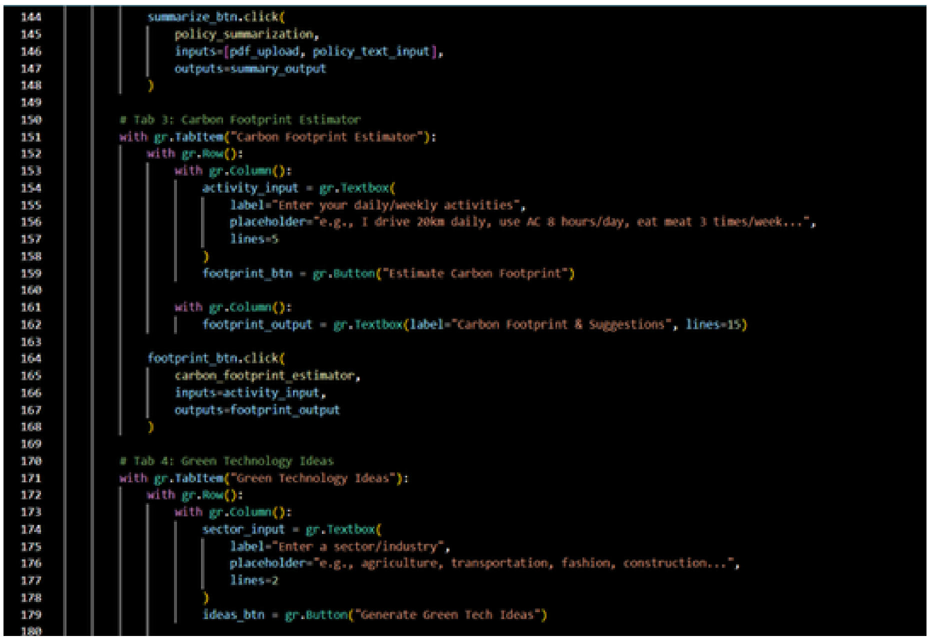
```python
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    respons (variable) response: Any , "").strip()
    return response


def extract_text_from_pdf(pdf_file):
    """Read text from an uploaded PDF using pypdf"""
    if pdf_file is None:
        return ""
    try:
        pdf_reader = PdfReader(pdf_file)
        text = ""
        for page in pdf_reader.pages:
            page_text = page.extract_text()
            if page_text:
                text += page_text + "\n"
        return text
    except Exception as e:
        return f"Error reading PDF: {str(e)}"


def eco_tips_generator(problem_keywords):
    prompt = (
        f"Generate practical and actionable eco-friendly tips for sustainable living "
        f"related to: {problem_keywords}. Provide specific solutions and suggestions."
    )
    return generate_response(prompt, max_length=1000)


def policy_summarization(pdf_file, policy_text):
    if pdf_file is not None:
        content = extract_text_from_pdf(pdf_file)
        summary_prompt = (
            f"Summarize the following policy document and extract the most important points, "
            f"key provisions, and implications:\n\n{content}"
```



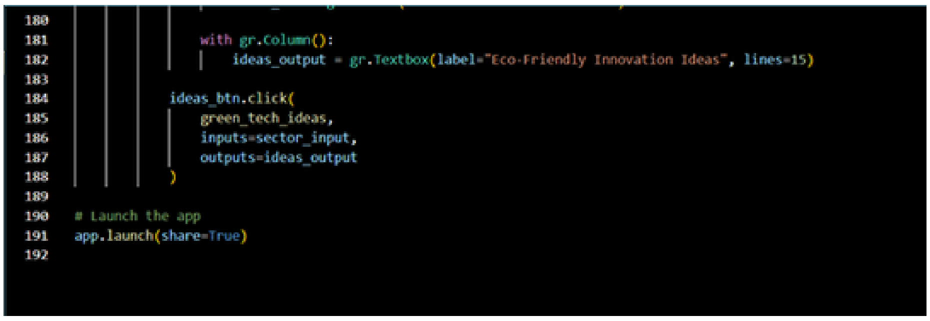
```python
        )
    else:
        summary_prompt = (
            f"Summarize the following policy document and extract the most important points, "
            f"key provisions, and implications:\n\n{policy_text}"
        )

    return generate_response(summary_prompt, max_length=1200)


def carbon_footprint_estimator(activity_details):
    prompt = (
        f"Estimate the carbon footprint (in kg CO₂ per month) based on the following lifestyle details: "
        f"{activity_details}. Then, suggest ways to reduce the footprint effectively."
    )
    return generate_response(prompt, max_length=800)


def green_tech_ideas(sector):
    prompt = (
        f"Suggest innovative eco-friendly and sustainable technology ideas for the sector: {sector}. "
        f"Include practical applications, scalability, and environmental impact."
    )
    return generate_response(prompt, max_length=900)


# ------------------ Gradio UI ------------------
with gr.Blocks() as app:
    gr.Markdown("# 🌱 Eco Assistant & Policy Analyzer")

    with gr.Tabs():
        # Tab 1: Eco Tips
        with gr.TabItem("Eco Tips Generator"):
            with gr.Row():
                with gr.Column():
                    keywords_input = gr.Textbox(
```



```python
                        label="Environmental Problem/Keywords",
                        placeholder="e.g., plastic, solar, water waste, energy saving...",
                        lines=3
                    )
                    generate_tips_btn = gr.Button("Generate Eco Tips")

                with gr.Column():
                    tips_output = gr.Textbox(label="Sustainable Living Tips", lines=15)

            generate_tips_btn.click(
                eco_tips_generator,
                inputs=keywords_input,
                outputs=tips_output
            )

        # Tab 2: Policy Summarization
        with gr.TabItem("Policy Summarization"):
            with gr.Row():
                with gr.Column():
                    pdf_upload = gr.File(
                        label="Upload Policy PDF",
                        file_types=[".pdf"]
                    )
                    policy_text_input = gr.Textbox(
                        label="Or paste policy text here",
                        placeholder="Paste policy document text...",
                        lines=5
                    )
                    summarize_btn = gr.Button("Summarize Policy")

                with gr.Column():
                    summary_output = gr.Textbox(
                        label="Policy Summary & Key Points",
                        lines=20
                    )

            summarize_btn.click(
```



```
            summarize_btn.click(
                policy_summarization,
                inputs=[pdf_upload, policy_text_input],
                outputs=summary_output
            )

        # Tab 3: Carbon Footprint Estimator
        with gr.TabItem("Carbon Footprint Estimator"):
            with gr.Row():
                with gr.Column():
                    activity_input = gr.Textbox(
                        label="Enter your daily/weekly activities",
                        placeholder="e.g., I drive 20km daily, use AC 8 hours/day, eat meat 3 times/week...",
                        lines=5
                    )
                    footprint_btn = gr.Button("Estimate Carbon Footprint")

                with gr.Column():
                    footprint_output = gr.Textbox(label="Carbon Footprint & Suggestions", lines=15)

            footprint_btn.click(
                carbon_footprint_estimator,
                inputs=activity_input,
                outputs=footprint_output
            )

        # Tab 4: Green Technology Ideas
        with gr.TabItem("Green Technology Ideas"):
            with gr.Row():
                with gr.Column():
                    sector_input = gr.Textbox(
                        label="Enter a sector/industry",
                        placeholder="e.g., agriculture, transportation, fashion, construction...",
                        lines=2
                    )
                    ideas_btn = gr.Button("Generate Green Tech Ideas")
```



```

                with gr.Column():
                    ideas_output = gr.Textbox(label="Eco-Friendly Innovation Ideas", lines=15)

            ideas_btn.click(
                green_tech_ideas,
                inputs=sector_input,
                outputs=ideas_output
            )

# Launch the app
app.launch(share=True)
```

Hugging Face Link:

https://huggingface.co/spaces/HARIHARASUDAN06/SmartSDLC

GitHub Link:

https://github.com/HARIHARASUDAN15/SmartSDLC.git