Sustainable Smart City Assistant
Project Documentation

1.Introduction

- **Project title : Sustainable Smart CIty Assistant**
- **Team Leader :Samsuriya K**
- **Team member :Tharunkumar S**
- **Team member :Paramesh G**
- **Team member :RonNicciy S**

2. Project overview :

• Purpose : The purpose of a Sustainable Smart City Assistant is to empower cities and their residents to thrive in a more eco-conscious and connected urban environment. By leveraging AI and real-time data, the assistant helps optimize essential resources like energy, water, and waste, while also guiding sustainable behaviors among citizens through personalized tips and services. For city officials, it serves as a decisionmaking partner—offering clear insights, forecasting tools, and summarizations of complex policies to support strategic planning. Ultimately, this assistant bridges technology, governance, and community engagement to foster greener cities that are more efficient, inclusive, and resilient.

• Features: Conversational Interface Key Point: Natural language interaction Functionality: Allows citizens and officials to ask questions, get updates, and receive guidance in plain language Policy Summarization Key Point: Simplified policy understanding Functionality: Converts lengthy government documents into concise, actionable summaries. Resource Forecasting Key Point: Predictive analytics Functionality: Estimates future energy, water, and waste usage using historical and real-time data. Eco-Tip Generator Key Point: Personalized sustainability advice Functionality: Recommends daily actions to reduce environmental impact based on user behavior. Citizen Feedback Loop Key Point: Community engagement Functionality: Collects and analyzes public input to inform city planning and service improvements. KPI Forecasting Key Point: Strategic planning support Functionality: Projects key performance indicators to help officials track progress and plan ahead. Anomaly Detection Key Point: Early warning system Functionality: Identifies unusual patterns in sensor or usage data to flag potential issues. Multimodal Input Support Key Point: Flexible data handling Functionality: Accepts text, PDFs, and CSVs for document analysis and forecasting. Streamlit or Gradio UI Key Point: User-friendly interface Functionality: Provides an intuitive dashboard for both citizens and city officials to interact with the assistant.

3. Architecture Frontend (Stream lit):

The frontend is built with Stream lit, offering an interactive web UI with multiple pages including dashboards, file uploads, chat interface, feedback forms, and report viewers. Navigation is handled through a sidebar using the stream lit-option-menu library. Each page

is modularized for scalability. Backend (Fast API): Fast API serves as the backend REST framework that powers API endpoints for document processing, chat interactions, eco tip generation, report creation, and vector embedding. It is optimized for asynchronous performance and easy Swagger integration. LLM Integration (IBM Watsonx Granite): Granite LLM models from IBM Watsonx are used for natural language understanding and generation. Prompts are carefully designed to generate summaries, sustainability tips, and reports. Vector Search (Pinecone): Uploaded policy documents are embedded using Sentence Transformers and stored in Pinecone. Semantic search is implemented using cosine similarity to allow users to search documents using natural language queries. ML Modules (Forecasting and Anomaly Detection): Lightweight ML models are used for forecasting and anomaly detection using Scikit-learn. Time-series data is parsed, modeled, and visualized using pandas and matplotlib.

4. Setup Instructions Prerequisites:

o Python 3.9 or later o pip and virtual environment tools o API keys for IBM Watsonx and Pinecone o Internet access to access cloud services Installation Process: o Clone the repository o Install dependencies from requirements.txt o Create a .env file and configure credentials o Run the backend server using Fast API o Launch the frontend via Stream lit o Upload data and interact with the modules

5. Folder Structure app/ –

Contains all Fast API backend logic including routers, models, and integration modules. app/api/ – Subdirectory for modular API routes like chat, feedback, report, and document vectorization. ui/ – Contains frontend components for Stream lit pages, card layouts, and form UIs. smart_dashboard.py – Entry script for launching the main Stream lit dashboard. granite_llm.py – Handles all communication with IBM Watsonx Granite model including summarization and chat. document_embedder.py – Converts documents to embeddings and stores in Pinecone. kpi_file_forecaster.py – Forecasts future energy/water trends using regression. anomaly_file_checker.py – Flags unusual values in uploaded KPI data. report_generator.py – Constructs AI-generated sustainability reports.

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 reports, summaries, and predictions.
➢ All interactions are real-time and use backend APIs to dynamically update the frontend.

Frontend (Stream lit):

The frontend is built with Stream lit, offering an interactive web UI with multiple pages including dashboards, file uploads, chat interface, feedback forms, and report viewers. Navigation is handled through a sidebar using the stream lit-option-menu library. Each page is modularized for scalability. Backend (Fast API): Fast API serves as the backend REST

framework that powers API endpoints for document processing, chat interactions, eco tip generation, report creation, and vector embedding. It is optimized for asynchronous performance and easy Swagger integration.

7. API Documentation Backend APIs available include:

POST /chat/ask – Accepts a user query and responds with an AI-generated message POST /upload-doc – Uploads and embeds documents in Pinecone GET /search-docs – Returns semantically similar policies to the input query GET /get-eco-tips – Provides sustainability tips for selected topics like energy, water, or waste POST /submit-feedback – Stores citizen feedback for later review or analytics Each endpoint is tested and documented in Swagger UI for quick inspection and trial during development.

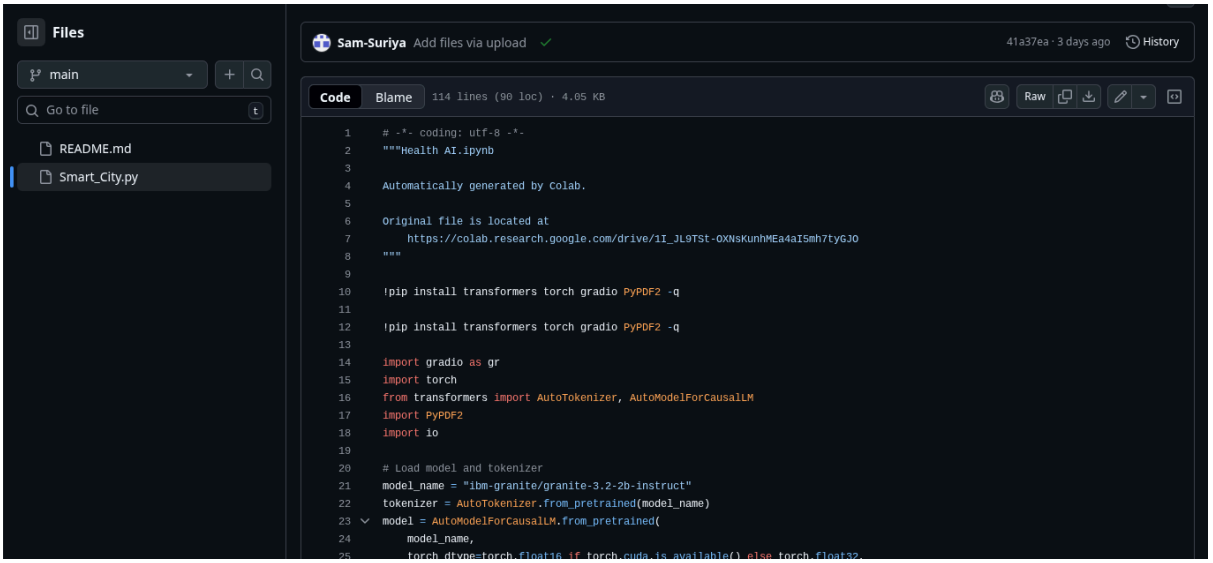
8. Authentication of each endpoint is tested and documented in Swagger UI for quick inspection and trial during development. This version of the project runs in an open environment for demonstration. However, secure deployments can integrate: • Token-based authentication (JWT or API keys) • OAuth2 with IBM Cloud credentials • Role-based access (admin, citizen, researcher) • Planned enhancements include user sessions and history tracking.

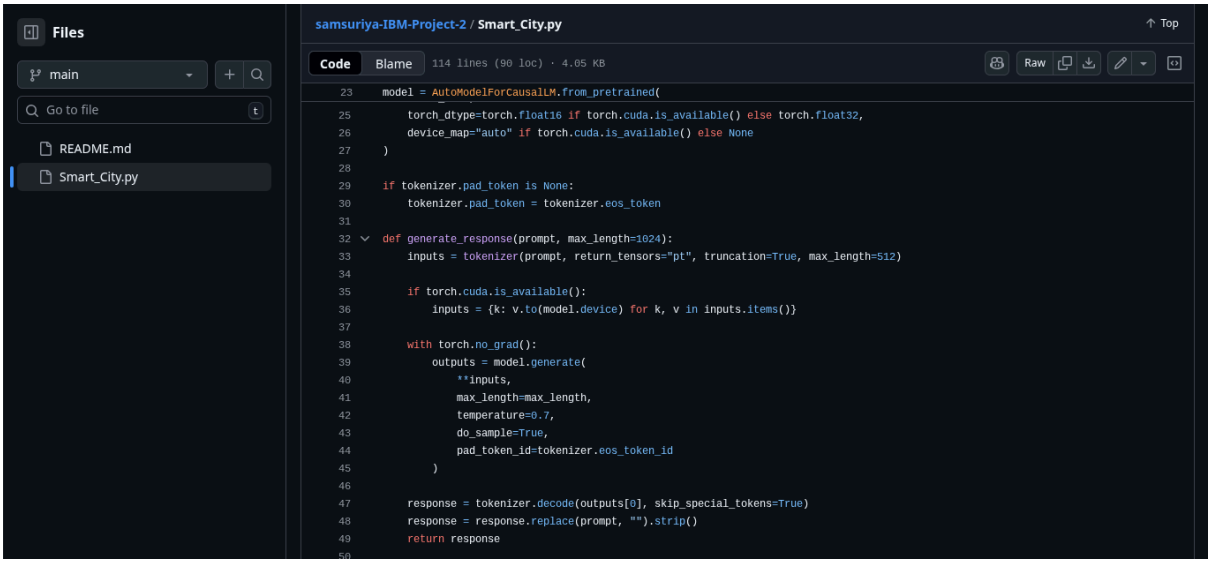
9. User Interface The interface is minimalist and functional, focusing on accessibility for nontechnical users. It includes: Sidebar with navigation KPI visualizations with summary cards Tabbed layouts for chat, eco tips, and forecasting Real-time form handling PDF report download capability The design prioritizes clarity, speed, and user guidance with help texts and intuitive flows.

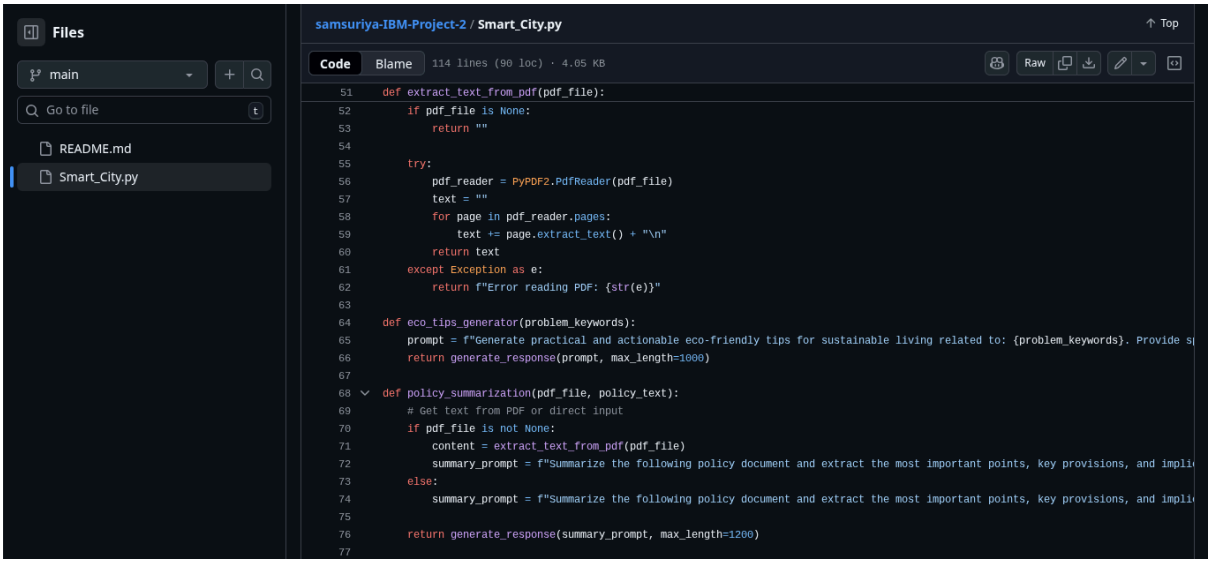
10. Testing Testing was done in multiple phases:

Unit Testing: For prompt engineering functions and utility scripts API Testing: Via Swagger UI, Postman, and test scripts Manual Testing: For file uploads, chat responses, and output consistency Edge Case Handling: Malformed inputs, large files, invalid API keys Each function was validated to ensure reliability in both offline and APIconnected modes.

11.screen shots :







Files

main

Go to file t

README.md

Smart_City.py

samsuriya-IBM-Project-2 / Smart_City.py

↑ Top

Code Blame 114 lines (90 loc) · 4.05 KB

Raw

```
68  def policy_summarization(pdf_file, policy_text):
76      return generate_response(summary_prompt, max_length=1200)
77
78  # Create Gradio interface
79  with gr.Blocks() as app:
80      gr.Markdown("# Eco Assistant & Policy Analyzer")
81
82      with gr.Tabs():
83          with gr.TabItem("Eco Tips Generator"):
84              with gr.Row():
85                  with gr.Column():
86                      keywords_input = gr.Textbox(
87                          label="Environmental Problem/Keywords",
88                          placeholder="e.g., plastic, solar, water waste, energy saving...",
89                          lines=3
90                      )
91                      generate_tips_btn = gr.Button("Generate Eco Tips")
92
93                  with gr.Column():
94                      tips_output = gr.Textbox(label="Sustainable Living Tips", lines=15)
95
96              generate_tips_btn.click(eco_tips_generator, inputs=keywords_input, outputs=tips_output)
97
98          with gr.TabItem("Policy Summarization"):
99              with gr.Row():
100                 with gr.Column():
101                     pdf_upload = gr.File(label="Upload Policy PDF", file_types=[".pdf"])
```

Files

main

Go to file t

README.md

Smart_City.py

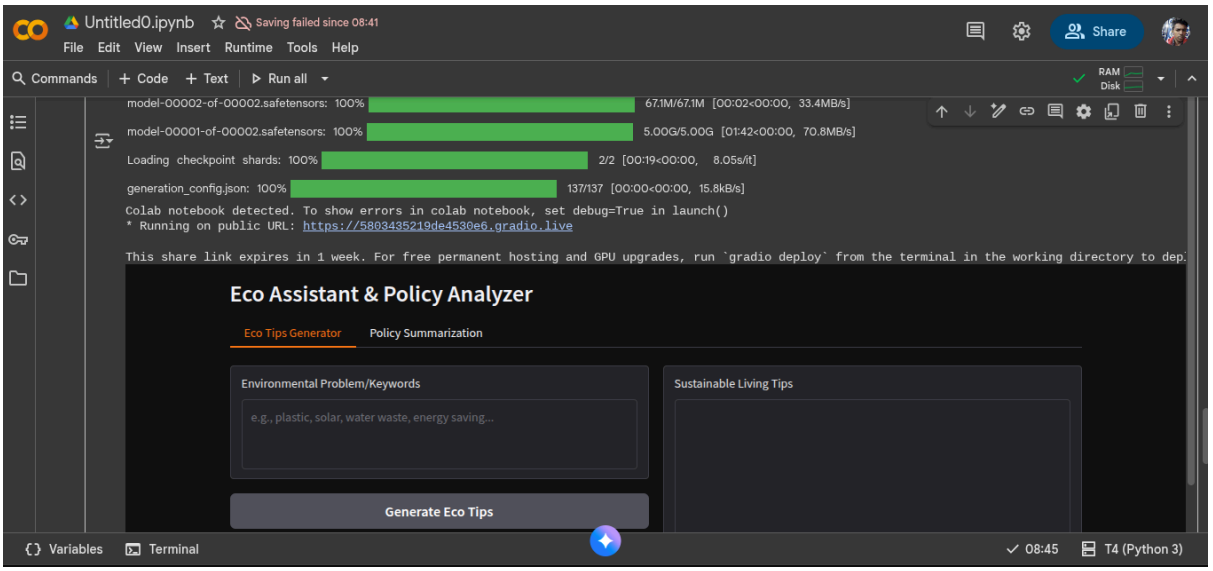
samsuriya-IBM-Project-2 / Smart_City.py

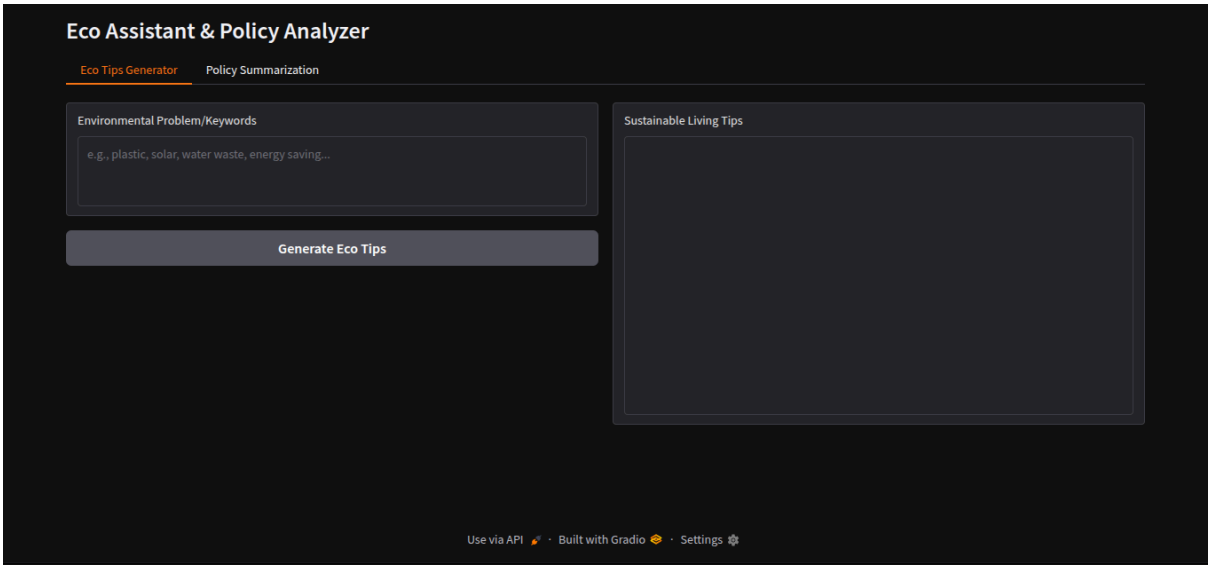
↑ Top

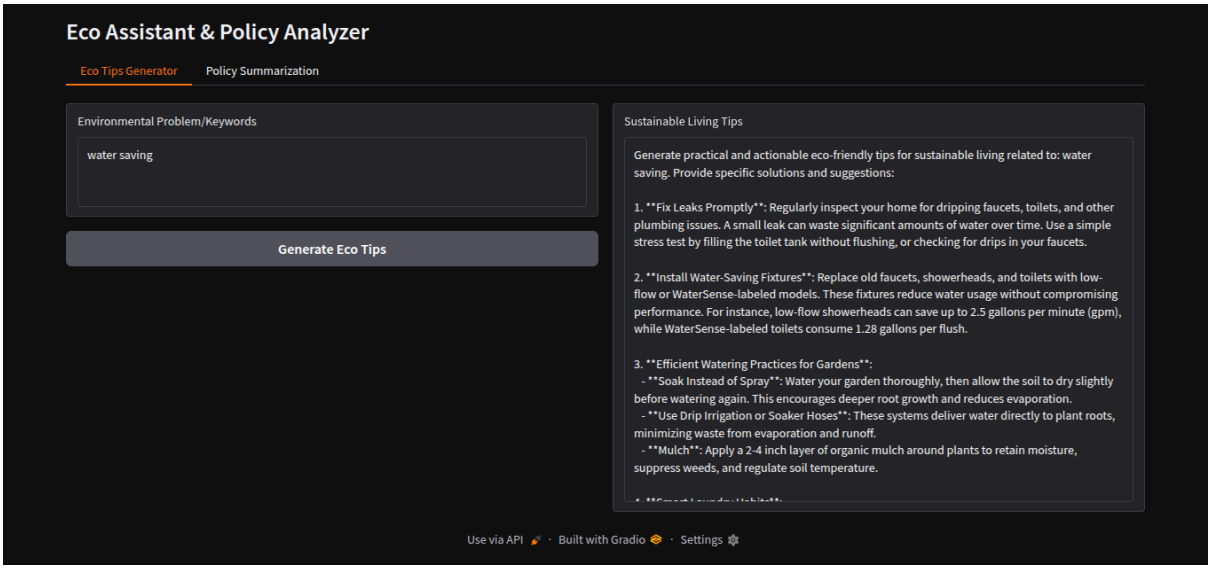
Code Blame 114 lines (90 loc) · 4.05 KB

Raw

```
68  def policy_summarization(pdf_file, policy_text):
92
93                  with gr.Column():
94                      tips_output = gr.Textbox(label="Sustainable Living Tips", lines=15)
95
96              generate_tips_btn.click(eco_tips_generator, inputs=keywords_input, outputs=tips_output)
97
98          with gr.TabItem("Policy Summarization"):
99              with gr.Row():
100                 with gr.Column():
101                     pdf_upload = gr.File(label="Upload Policy PDF", file_types=[".pdf"])
102                     policy_text_input = gr.Textbox(
103                         label="Or paste policy text here",
104                         placeholder="Paste policy document text...",
105                         lines=5
106                     )
107                     summarize_btn = gr.Button("Summarize Policy")
108
109                 with gr.Column():
110                     summary_output = gr.Textbox(label="Policy Summary & Key Points", lines=20)
111
112             summarize_btn.click(policy_summarization, inputs=[pdf_upload, policy_text_input], outputs=summary_output)
113
114 app.launch(share=True)
```







12. Known Issues:

Take more time to generate tips

13. Future enhancement:

Improve the speed of reply