Sustainable Smart City Assistant Project Documentation

1. Introduction

• Project Title: Sustainable Smart City Assistant Using IBM Granite LLM

• Team Members:

- Monica S

- Gayathiri Devi GH

- Nithiya Sri S

2. Project Overview

• **Purpose:**

The Sustainable Smart City Assistant is an AI-powered platform designed to empower cities and residents with eco-conscious tools and data-driven decision-making. It leverages IBM Watsonx Granite LLM and modern data pipelines to optimize energy, water, and waste usage while providing simplified policy summaries, citizen feedback loops, eco-advice, KPI forecasting, and anomaly detection. The platform bridges technology, governance, and citizen engagement to foster greener, inclusive, and resilient urban environments.

• **Features:**

Conversational Chat Assistant
Key Point: Natural language interaction
Functionality: Citizens and officials ask sustainability questions and receive AI-powered guidance.

Policy Summarization
Key Point: Simplified understanding
Functionality: Summarizes complex city policy documents into concise, actionable insights.

Resource Forecasting
Key Point: Predictive analytics
Functionality: Forecasts water, energy, and waste usage based on past data.

Eco-Tip Generator

Key Point: Sustainable lifestyle advice

Functionality: Recommends daily eco-friendly actions based on user input.

Citizen Feedback Reporting

Key Point: Real-time issue reporting

Functionality: Enables residents to log city issues instantly for government review.

KPI Forecasting & Anomaly Detection

Key Point: Strategic planning & early warnings

Functionality: Forecasts key indicators and detects irregularities in urban data.

Multimodal Input Support

Key Point: Flexible data handling

Functionality: Accepts text, PDFs, and CSVs for summarization, forecasting, and anomaly detection.

Streamlit Dashboard

Key Point: User-friendly UI

Functionality: Provides interactive dashboards for data visualization, reports, and eco insights.

Use Case Scenarios

Policy Search & Summarization: A municipal planner uploads a complex city policy document, and the assistant generates a simplified summary.

Citizen Feedback Reporting: A resident submits an issue such as a burst pipe, and the assistant logs it with category tagging for officials.

KPI Forecasting: A city administrator uploads last year's water usage data and receives AI-powered consumption forecasts for planning.

3. Architecture

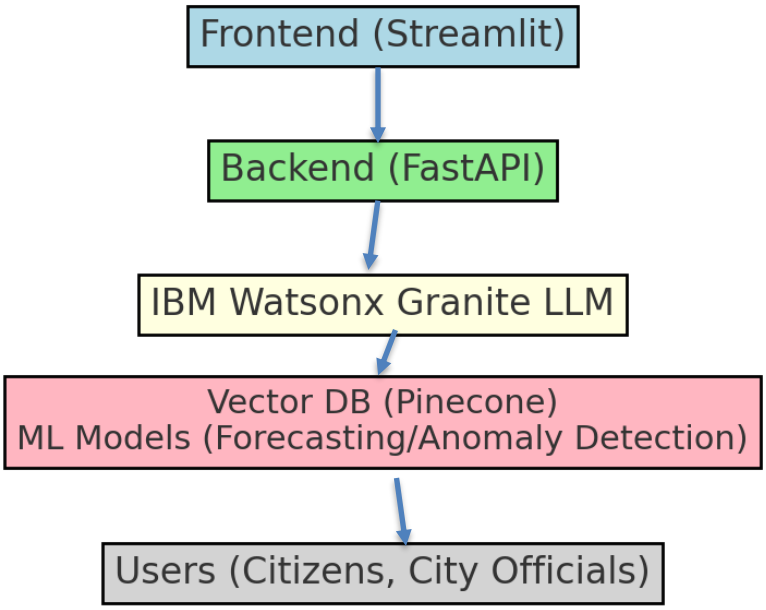
Frontend (Streamlit): Modular dashboard with chat, feedback, KPI views, policy search, eco tips, and anomaly detection.

Backend (FastAPI): RESTful APIs handling uploads, ML forecasting, anomaly detection, and LLM integration.

LLM Integration (IBM Watsonx Granite): Summarization, eco tips, sustainability reports, and conversational AI.

Vector Search (Pinecone): Semantic policy search with document embeddings.

ML Modules: Forecasting and anomaly detection using scikit-learn, pandas, and matplotlib.



4. Setup Instructions

Prerequisites:

- Python 3.9+
- FastAPI, Streamlit
- IBM Watsonx & Pinecone API keys
- scikit-learn, pandas, matplotlib
- Internet access

Installation Process:

- Clone the repository
- Install dependencies from requirements.txt
- Configure API credentials in .env
- Run FastAPI backend
- Launch Streamlit frontend
- Upload documents/data and explore modules

5. Folder Structure

app/ – FastAPI backend logic

app/api/ – Routers for chat, feedback, eco tips, policies, KPIs

ui/ – Streamlit frontend components

smart_dashboard.py – Entry script for dashboard

granite_llm.py – LLM service functions (summaries, eco tips, chat)

document_embedder.py – Converts documents to embeddings

kpi_file_forecaster.py – Forecasts urban KPIs

anomaly_file_checker.py – Flags anomalies in datasets

report_generator.py – Creates sustainability reports

6. Running the Application

➢ Start FastAPI backend
➢ Run Streamlit dashboard
➢ Navigate via sidebar
➢ Upload policies or KPI data
➢ Interact with chat assistant and eco tools
➢ View forecasts, anomalies, and sustainability reports

7. API Documentation

POST /chat/ask – AI-generated responses

POST /upload-doc – Uploads and embeds documents

GET /search-docs – Semantic policy search

GET /get-eco-tips – Provides sustainability tips

POST /submit-feedback – Stores citizen feedback

8. Authentication

• Token-based authentication (JWT / API keys)
• OAuth2 with IBM Cloud
• Role-based access (admin, citizen, researcher)
• Planned: session management & history tracking

9. User Interface

• Sidebar navigation with themed icons
• KPI visualizations with summary cards
• Chat assistant with real-time AI responses
• Feedback forms with issue categorization

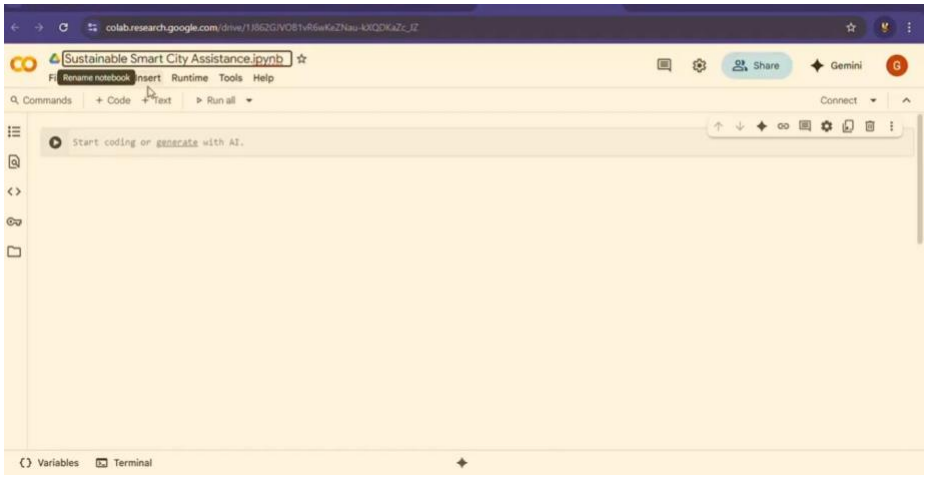
• Policy summarization and eco tips display
• Report generation and download

10. Testing

• Unit Testing: For backend services and ML functions
• API Testing: Swagger UI, Postman
• Manual Testing: For chat, policy search, forecasting
• Edge Case Handling: Large files, malformed inputs, invalid API keys

11. Screenshots with steps

Step 1: Project Setup in Google Colab

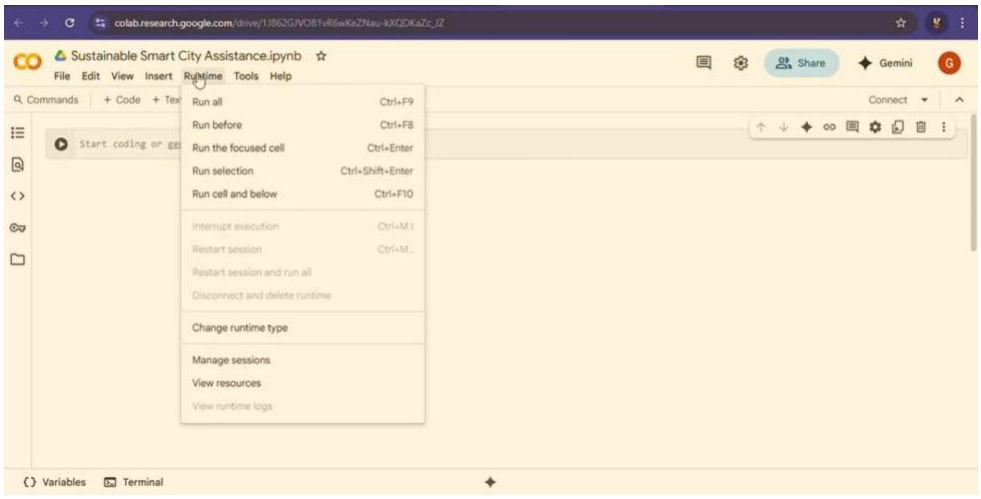


Objective: Begin coding in a cloud-based Python environment.

Action: Create a new notebook and name it appropriately (e.g., Sustainable Smart City Assistance).

Step 2: Runtime Configuration

Objective: Optimize performance using GPU acceleration.

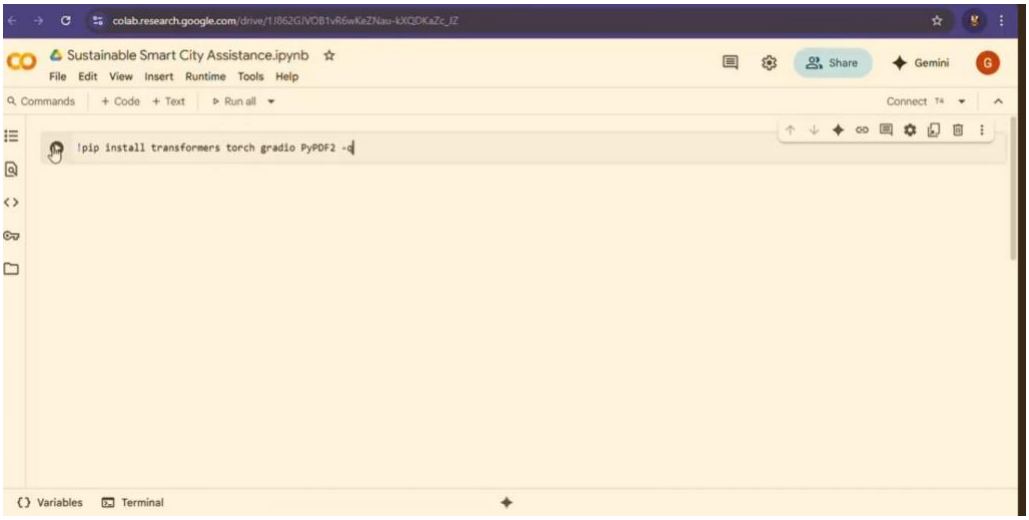


Runtime type set to Python 3 with T4 GPU.

Action: Go to Runtime > Change runtime type and select GPU for faster model execution.

Step 3: Install Required Libraries

Objective: Set up the environment with necessary packages.



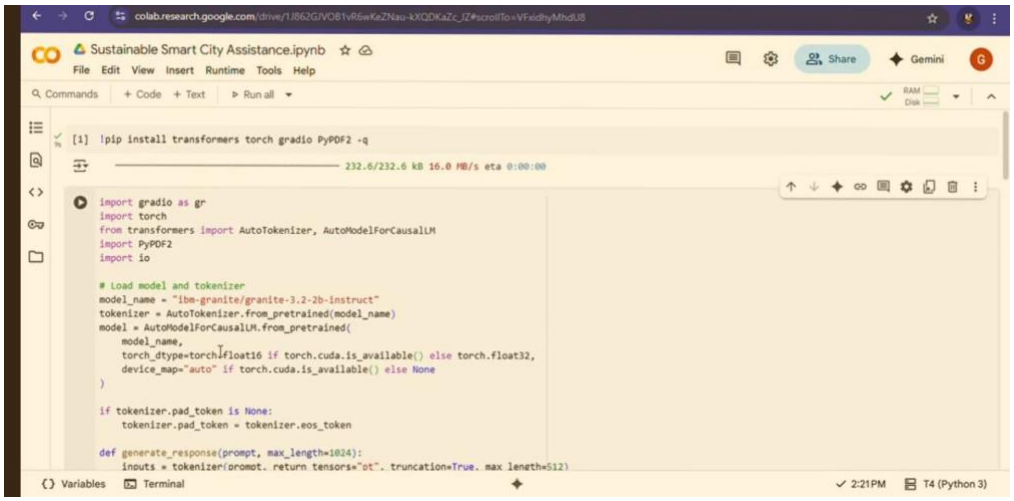
!pip install transformers torch gradio pypdf -q

Action: Run the command to install libraries for NLP, UI, and PDF handling.

Step 4: Build Gradio Interface

Objective: Create a user-friendly UI for eco tips and policy summarization.

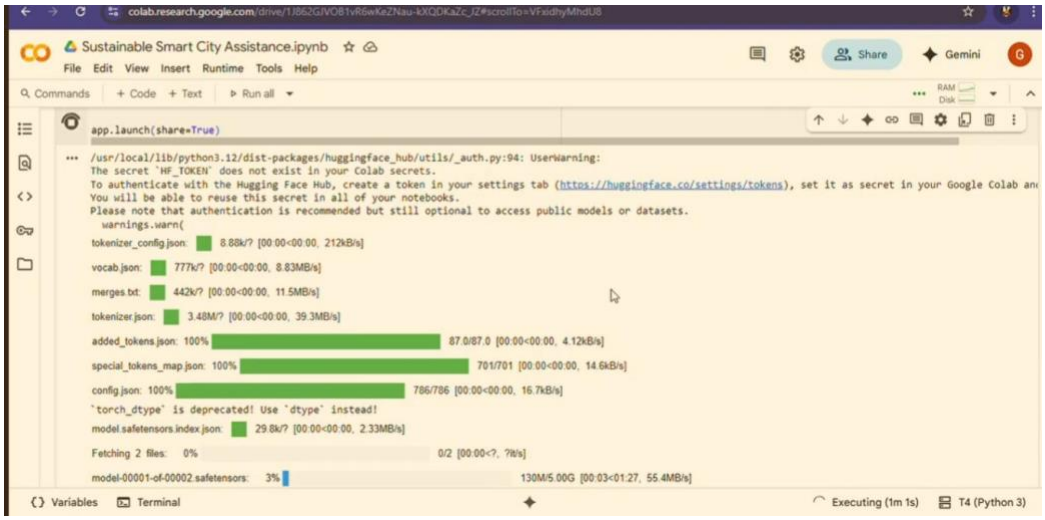
Gradio code with buttons for “Generate Eco Tags” and “Summarize Policy”.



Action: Use gr.Button, gr.Textbox, and gr.File to build interactive components.

Step 5: Load AI Model for Text Generation

Objective: Use Falcon-7B or BERT for generating responses and analyzing sentiment.



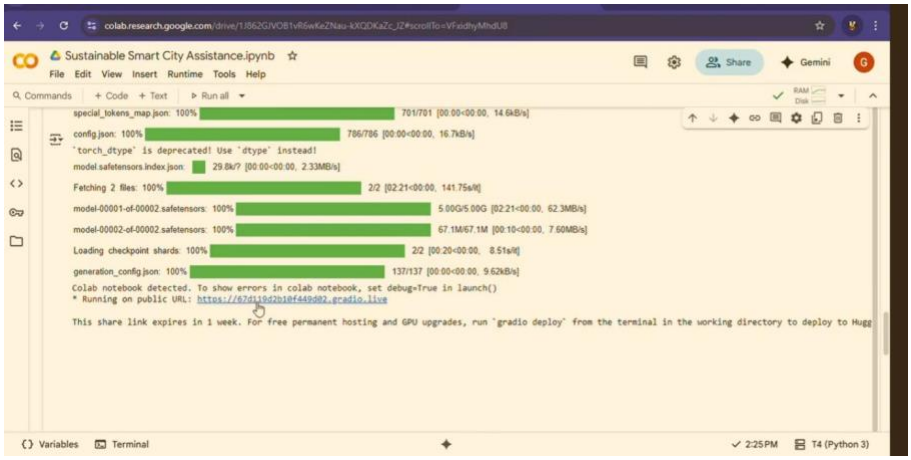
Model loading code with tokenizer and generate_response function.

Action: Load pre-trained models from Hugging Face and configure device settings.

Step 6: Web App Deployment

Objective: Launch the assistant as a public-facing tool.

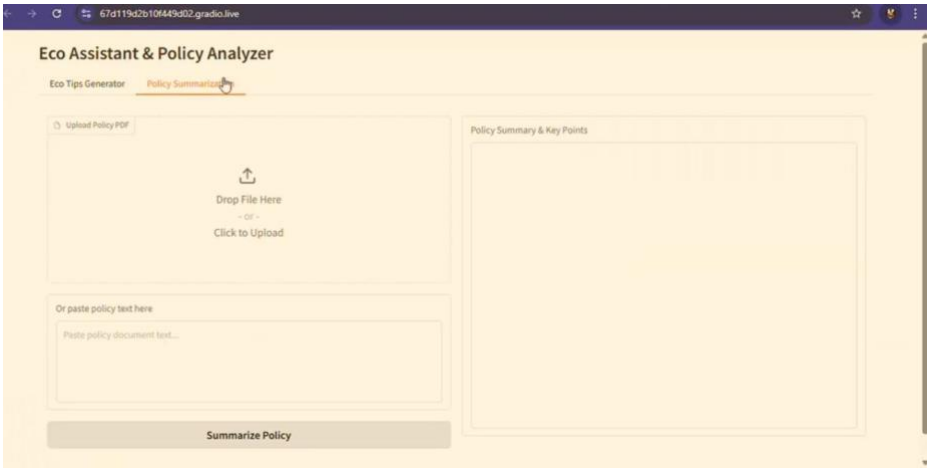
Gradio app titled “Eco Assistant & Policy Analyzer”.



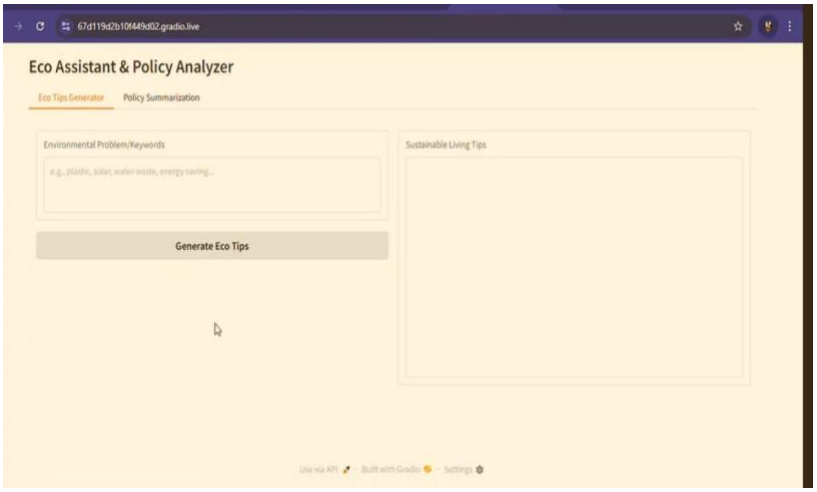
Action: Use app.launch(share=True) to deploy and share the app.

Step 7: Final Interface Overview

Objective: Showcase the complete UI for eco tips and policy analysis.



Web interface with input fields and summary output.



Action: Test the app with sample keywords and policy documents.

12. Known Issues

- Limited language support
- Requires stable cloud connectivity
- Dependent on API quota limits

13. Future Enhancements

- Multi-language support
- Integration with IoT and city sensors
- Advanced anomaly detection (deep learning)
- Mobile-friendly dashboard
- Doctoral/official verification of eco policies