Sustainable smart City Assistant using IBM Granite LLM Documentation

1.Introduction

Project Title : Sustainable smart City Assistant using IBM Granite LLM
Team member: LUBNA BATUL Z
Team member: NIRMALA R
Team member: PREETHI P
Team member: MANJUVARTHINI N

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 decision- making 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.

- Conversational Interface

  o Key Point: Natural language interaction

  o Functionality: Allows citizens and officials to ask questions, get updates, and receive guidance in plain language

- Policy Summarization

  o Key Point: Simplified policy understanding

  o 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

  oKey Point: User-friendly interface

  oFunctionality: Provides an intuitive dashboard for both citizens and city officials to interact with the assistant.

3. Architecture

- Frontend (Streamlit):

  The frontend is built with Streamlit, 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 streamlitoption-menu library. Each page is modularized for scalability.

- Backend (FastAPI):

  FastAPI 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.

4. Setup Instructions

Prerequisites:

- Python 3.9 or later
- pip and virtual environment tools
- API keys for IBM Watsonx and Pinecone
- Internet access to access cloud services

Installation Process:

- Clone the repository
- Install dependencies from requirements.txt
- Create a .env file and configure credentials
- Run the backend server using
- Fast API Launch the frontend via Stream lit 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 Fast API 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

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

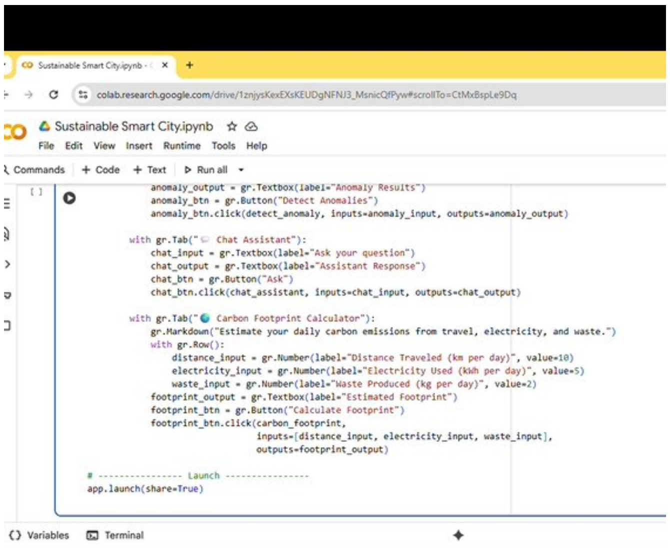
Testing was done in multiple phases:

- 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 API connected modes.

11. Screenshots

Code Running :


Sustainable Smart City.ipynb
colab.research.google.com/drive/1znjysKexEXsKEUDgNFNJ3_MsnicQfPyw#scrollTo=CtMxBspLe9Dq
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Variables Terminal

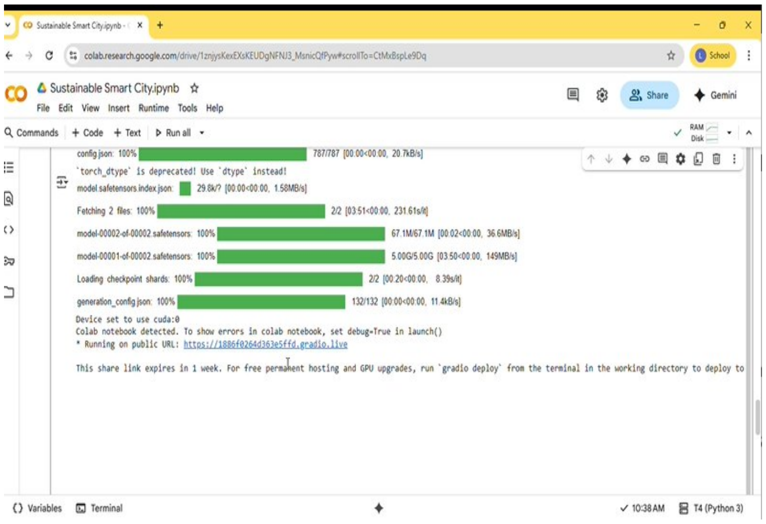
```
            anomaly_output = gr.Textbox(label="Anomaly Results")
            anomaly_btn = gr.Button("Detect Anomalies")
            anomaly_btn.click(detect_anomaly, inputs=anomaly_input, outputs=anomaly_output)

        with gr.Tab("💬 Chat Assistant"):
            chat_input = gr.Textbox(label="Ask your question")
            chat_output = gr.Textbox(label="Assistant Response")
            chat_btn = gr.Button("Ask")
            chat_btn.click(chat_assistant, inputs=chat_input, outputs=chat_output)

        with gr.Tab("🌍 Carbon Footprint Calculator"):
            gr.Markdown("Estimate your daily carbon emissions from travel, electricity, and waste.")
            with gr.Row():
                distance_input = gr.Number(label="Distance Traveled (km per day)", value=10)
                electricity_input = gr.Number(label="Electricity Used (kWh per day)", value=5)
                waste_input = gr.Number(label="Waste Produced (kg per day)", value=2)
            footprint_output = gr.Textbox(label="Estimated Footprint")
            footprint_btn = gr.Button("Calculate Footprint")
            footprint_btn.click(carbon_footprint,
                                inputs=[distance_input, electricity_input, waste_input],
                                outputs=footprint_output)

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

Output:


Sustainable Smart City.ipynb
colab.research.google.com/drive/1znjysKexEXsKEUDgNFNJ3_MsnicQfPyw#scrollTo=CtMxBspLe9Dq
School
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM
Disk
Variables Terminal
10:38 AM
T4 (Python 3)

```
config.json: 100% 787/787 [00:00<00:00, 20.7kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 1.58MB/s]
Fetching 2 files: 100% 2/2 [03:51<00:00, 231.61s/it]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:02<00:00, 36.6MB/s]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [03:50<00:00, 149MB/s]
Loading checkpoint shards: 100% 2/2 [00:20<00:00, 8.39s/it]
generation_config.json: 100% 132/132 [00:00<00:00, 11.4kB/s]
Device set to use cuda:0
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://1886f0264d363e5ffd.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
```

12.Known Issues

- Limited to demo-level security (no full authentication/authorization yet).
- Dependence on external APIs (IBM Watsonx, Pinecone) — project won't work without active internet & valid keys
- Forecasting models are basic; they may not handle very large or complex datasets accurately.
- UI is functional but not fully optimized for mobile devices.
- System may face delays when processing very large PDF or CSV files.

13. Future Enhancements

- Implement full user authentication (JWT, OAuth2) and role-based access for secure deployment.
- Expand ML models with deep learning for better forecasting and anomaly detection.
- Add multilingual support so citizens can interact in regional languages.
- Improve UI with responsive design for both web and mobile apps.
- Enable integration with IoT/sensor networks for real-time smart city data.
- Build advanced analytics dashboards with custom visualizations for city officials.
- Allow offline/edge deployment to reduce dependency on cloud services.