# Sustainable Smart City Assistant Using IBM Granite LLM

# 1.Intoduction

Project title: Sustainable smart city assistant IBM granite LLM

|  |  |  |
| --- | --- | --- |
| **MEMBERS** | **NAME** | **NM\_ ID** |
| Team Leader | G. Divya | 46C0891F2DCBE6E9F8D41F5960EEB079 |
| Team Member | S. Sweetha | 035CF33001982EB8D134 18302F32393B |
| Team Member | S. Saranya | A40D0E9E86AF879A0014C04050ED549A |
| Team Member | S. Monisha | 3E492141CA8E53AA611461B89040126F |

# 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.

# • Features

1. **Policy Search & Summarization**

Urban Planner Support:A city planner uploads a 200-page renewable energy policy. The assistant uses IBM Granite LLM to summarize the key sections (objectives, deadlines, incentives) into a 2-page executive summary for quick decision-making.

Citizen Engagement: A community member searches, “What does the new waste management policy say about plastic usage?” The assistant extracts and explains the relevant clause in simple, non-technical language.

Comparative Analysis: A policymaker asks, “How does our city’s smart mobility policy compare with Singapore’s?” The assistant analyzes and provides a comparative summary, highlighting similarities and gaps.

1. **Citizen Feedback Reporting**

Real-Time Issue Reporting:A commuter notices a malfunctioning traffic light. Through the assistant’s mobile dashboard, they file a report tagged as “Traffic & Safety.” The system automatically routes it to the city traffic department.

Sentiment Analysis:Thousands of citizens submit opinions about a new metro project. The assistant uses Watson NLP to summarize overall sentiment (positive, negative, neutral), giving city leaders a quick snapshot of public opinion.

Feedback Prioritization: If multiple users report “garbage not collected” in the same area, the assistant clusters the reports into a priority task and alerts the sanitation department.

# KPI Forecasting & Analytics

Water Consumption Forecasting: A city admin uploads 5 years of water consumption data. The assistant forecasts next year’s demand, helping plan future water supply infrastructure.

Energy Optimization: The assistant predicts electricity demand peaks during summer and suggests solar energy storage strategies to prevent outages.

Public Transport Insights: The system forecasts bus ridership growth based on past usage and special events, allowing the city to optimize fleet deployment.

Anomaly Detection: If one neighborhood suddenly shows a spike in water consumption, the assistant flags it as a possible leak or misuse case.

# 4. Chat Assistant for Citizens & Administrators

Citizen Queries: A resident asks, “What is the garbage collection schedule in my area?” The assistant provides an instant answer, pulling information from municipal records.

Policy Guidance: An administrator queries, “Summarize all housing policies related to low-income families.” The assistant delivers a ready-to-use summary.

Sustainability Advice:A citizen asks, “How can I reduce my household water usage?” The assistant generates eco-friendly tips tailored to city sustainability goals.

Interactive KPI Insights: A policymaker types, “Show me the trend of carbon emissions over the last 5 years.” The assistant retrieves the KPI dashboard and overlays predictive insights.

# 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:

* 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.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.

# 6. 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.

7. 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.8. Authentication

8. User Interface

The interface is minimalist and functional, focusing on accessibility for non technical 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.

9. Testing

Testing was done in multiple phases;

Unit Testing: For prompt engineering functions and utility scripts

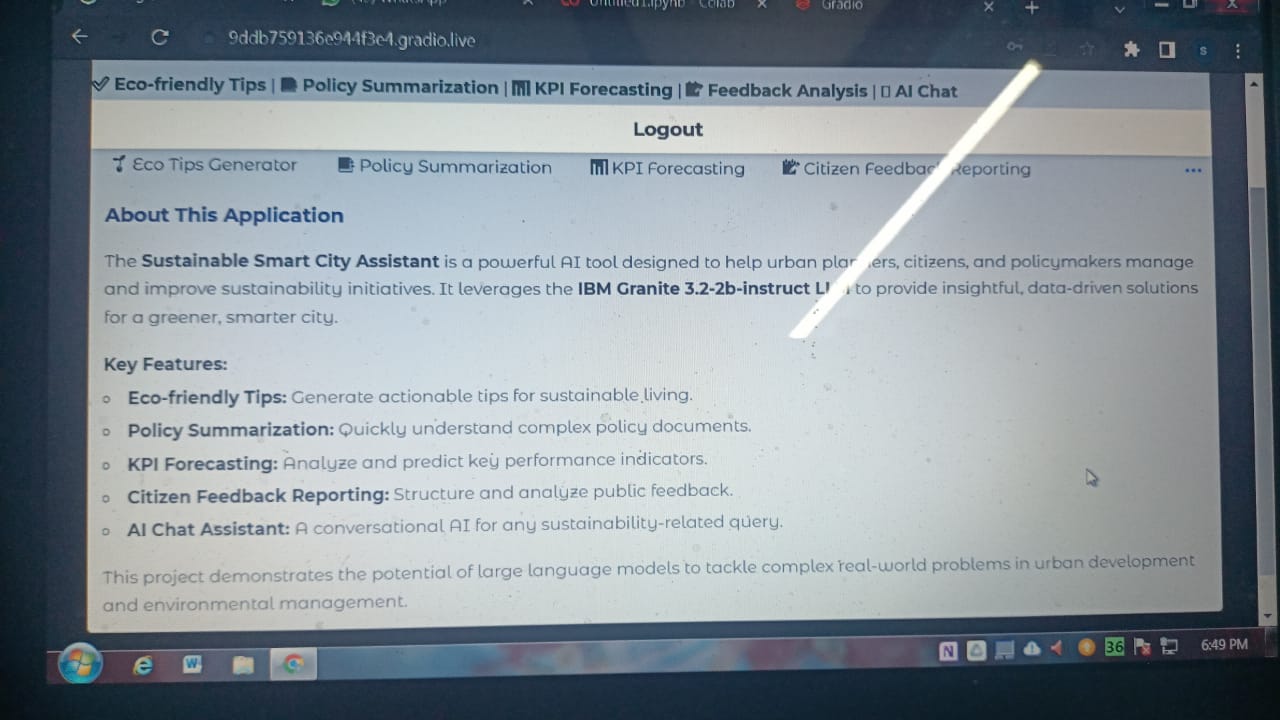
API Testing: Via Swagger UI, Postman, and test scripts

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.

10.screen shots

****