Sustainable Smart City Assistant Using IBM Granite LLM

Project Title: Sustainable Smart City Assistant Using IBM Granite LLM

Team Information

- Team ID: LTVIP2025TMID32124
- Team Size: 4 members
- Team Leader: Chintapalli Swetha
- Team Members: Bhatta pujitha,Bonthu Deekshith Naidu,Battula Bala Sai

Table of Contents

---

1. Project Overview

1.1 Problem Statement

Modern urban areas face increasing challenges in managing sustainable development, ensuring public satisfaction, and protecting the environment. Traditional governance models are often siloed, reactive, and inefficient, making it difficult to respond to complex urban demands in real-time.

Key Challenges Identified

Limited Citizen Engagement

Lack of easy-to-use platforms for submitting feedback and suggestions.

Inadequate communication between citizens and government bodies.

Environmental Monitoring Gaps

Absence of real-time data for key sustainability metrics such as air quality, water usage,and energy consumption.

Inability to predict and prevent environmental risks.

Policy Management Issues

Difficulty in retrieving, analyzing, and understanding large volumes of government policy documents.

Lack of accessibility and transparency for the general public.

Inefficient Decision Making

Poor forecasting of urban metrics and KPIs.

Limited use of data analytics in decision support systems.

Resource Optimization Challenges

Difficulty in monitoring departmental performance.

In effective resource allocation and lack of holistic performance reporting.

Target Audience

City Officials – Require intelligent tools for real-time insights and automated reports.

Citizens – Need an accessible and interactive platform for communication and eco-awareness.

Policy Makers – Benefit from semantic search and summarization of policy documents.

Environmental Agencies – Need predictive analytics and anomaly detection capabilities.

Proposed Solution

The Sustainable Smart City Assistant is an AI-powered platform designed to support smart city governance through natural language interactions and real-time data analysis. It utilizes IBM Watsonx Granite LLM for language processing and Pinecone for semantic search.

Project Objectives

- Enable real-time environmental monitoring and forecasting
- Facilitate citizen engagement via feedback collection
- Provide intelligent document summarization and search
- Generate eco-friendly tips and recommendations
- Deliver automated reporting for decision-makers
- Detect anomalies and patterns in KPI data to enable proactive interventions

2. System Architecture

2.1 Architecture Overview

The Sustainable Smart City Assistant follows a modular microservices architecture designed for scalability, flexibility, and efficient data processing. The system separates frontend, backend, AI logic, and database services to ensure maintainability and extensibility.

2.2 Major Layers:

Frontend: Developed using Flask, it provides a user-friendly interface for real-time interaction, data visualization, and feedback collection.

Backend: Powered by FastAPI, it acts as the central controller managing APIs, user requests, AI queries, and ML services.

AI Layer: Utilizes IBM Watsonx Granite LLM to process natural language queries, generate summaries, and provide intelligent responses.

Vector Search Engine: Implements Pinecone for semantic document retrieval using embeddings from Sentence Transformers.

ML & Analytics: Integrated with scikit-learn, pandas, and forecasting/anomaly detection modules for city KPI analysis.

Data Storage: Combines local file system for documents, and in-memory/session storage for user sessions and temporary data.

Component Interaction Flow

```
+--------------------+          +------------------------+
   |   flask    |          |  Citizen / City Official |
   +--------+-----------+         +------------------------+
            |                              ↑
            v                              |
   +------------------------+         +-------------------------+
   |  API Gateway (FastAPI)|<------->|    HTTP Requests (GET/POST) |
   +-----------+------------+         +-------------------------+
              |
   +-----------+-------------------------------------------------------+
   |                         Backend Service Layer                     |
   |  +------------------+  +------------------+   +-------------------+   |
   |  |  Chat Assistant  |  |  Feedback Router |   |  KPI Analysis     |   |
```

```
| +------------------+ +------------------+   +-------------------+  |
| |  Eco Tips Module | |  Policy Summarizer|  |  Report Generator|   |
| +------------------+ +------------------+   +-------------------+  |
+------------------------------------------------------------------+
        |
        v
+-----------------------------+
|  AI Layer (Granite LLM)     |
|  Text Generation & Summaries|
+-----------------------------+
        |
        v
+-----------------------------+
| Vector DB (Pinecone)        |
|  Semantic Search & Retrieval|
+-----------------------------+
```

Design Benefits

AI-Powered Insights – Natural language processing for intelligent responses
Semantic Understanding – Fast and relevant policy search
Real-Time KPI Monitoring – Live data analysis and forecasting

Modular and Scalable – Independent services for better performance and debugging
Cross-platform Compatibility – Works on local systems and cloud (Colab, Docker-ready)

---

3. Technology Stack

The Sustainable Smart City Assistant leverages a modern, full-stack ecosystem consisting of backend APIs, frontend user interfaces, machine learning, and AI capabilities. The selection of technologies ensures reliability, scalability, and AI-driven intelligence for real-time urban governance.

3.1 Backend Technologies

Tool/Framework	Purpose
FastAPI	High-performance web framework for building RESTful APIs.
Python 3.8+	Core programming language for both backend and ML logic.

Pydantic Used for data validation and type enforcement in FastAPI.

Uvicorn ASGI server to serve the FastAPI application efficiently.

3.2 Frontend Technologies

Tool/Library Purpose

Flask Framework for building interactive dashboards and web apps with Python.

Flask Option Menu For sidebar navigation and user interaction control.

Custom CSS Styling and visual enhancements for UI consistency.

3.3 AI & Natural Language Processing

Technology Purpose

IBM Watsonx Granite LLM Handles natural language understanding and generation. Used for chat assistant, policy summaries, eco-tips, and report generation.

Sentence Transformers Converts text to embeddings for semantic search using Pinecone.

3.4 Machine Learning & Analytics

Tool/Library Purpose

scikit-learn Forecasting KPIs and anomaly detection in city data.

pandas Data manipulation and preparation for analysis.

---

4. Project Structure

```
sustainable-smart-city-assistant/
│
├── backend/                    # Backend codebase for the project
│   ├── colab_notebook.ipynb         # Main notebook to run the backend in Google Colab
│   ├── models/
│   │   ├── kpi_forecast.py        # KPI forecasting logic
│   │   ├── anomaly_detection.py    # Anomaly detection logic
│   │   └── granite_integration.py  # IBM Granite LLM integration logic
│   ├── routes/
│   │   ├── feedback_routes.py      # Citizen feedback-related API endpoints
```

```
│   │   ├── policy_routes.py       # Policy summarization API endpoints
│   │   └── traffic_routes.py      # Traffic monitoring API endpoints
│   ├── utils/
│   │   ├── file_handler.py        # File upload and processing logic
│   │   ├── config.py              # Configuration variables (e.g., API keys)
│   │   └── logging_config.py      # Logging setup
│   ├── requirements.txt           # List of Python dependencies for backend
│   ├── README.md                  # Documentation for backend setup
│   └── test_colab_backend.py      # Backend API testing script
│
├── frontend/                      # Frontend Flask application
│   ├── templates/                 # HTML templates for the web interface
│   │   ├── index.html             # Dashboard interface
│   │   ├── feedback.html          # Citizen feedback submission page
│   │   ├── traffic.html           # Traffic monitoring interface
│   │   └── policy_summary.html    # Policy summary results page
│   ├── static/                    # Static files for styling and JavaScript
│   │   ├── css/
│   │   │   └── styles.css         # Custom CSS for frontend
│   │   └── js/
│   │       └── scripts.js         # JavaScript for frontend interactivity
│   ├── app.py                     # Flask application file to serve the frontend
│   ├── config.py                  # Configuration variables for frontend
│   ├── README.md                  # Frontend documentation
│   └── test_flask_frontend.py     # Script for testing the frontend
│
├── deployment/                    # Deployment scripts and instructions
│   ├── ngrok/
│   │   ├── setup_ngrok.sh         # Script to configure ngrok for backend
exposure
│   │   └── ngrok_auth_token.txt   # Your ngrok authentication token
```

```
│   ├── Dockerfile              # (Optional) Dockerfile for containerizing the app
│   ├── docker-compose.yml         # (Optional) Docker Compose file
│   ├── cloud_deploy.yaml         # (Optional) Cloud deployment configuration file
│   └── README.md                # Deployment documentation
│
├── data/                        # Data and logs for processing
│   ├── input/
│   │   ├── policy_documents/       # Sample policy documents
│   │   └── kpi_data/             # Sample KPI datasets
│   ├── output/
│   │   ├── summaries/             # Generated policy summaries
│   │   └── forecasts/            # Generated KPI forecasts
│   └── logs/                  # Log files for debugging
│
├── tests/                       # Test cases for backend and frontend
│   ├── test_cases.md              # Detailed test cases
│   ├── test_colab_backend.py       # Tests for backend
│   └── test_flask_frontend.py      # Tests for frontend
│
└── README.md                   # Main documentation for the project
```

---

5. Implementation Details

5.1 1. User Interface (Frontend)

Framework: React.js (or Vue.js for alternatives)

Libraries: Axios (for API calls), Chart.js (for visualizations), Leaflet.js (for city mapping

Features:

Citizen Dashboard: View pollution levels, traffic data, water usage, etc.

Admin Panel: Monitor devices, get alerts, manage data

Mobile Responsiveness: PWA for mobile support

- **Backend (Server)**

Platform: Node.js with Express.js

Authentication: JWT-based Role-Based Access Control (RBAC)

APIs:

RESTful APIs to handle data from sensors and user interfaces

Endpoints for alert generation, analytics, reporting

Database Interaction: ORM (like Sequelize or Mongoose depending on SQL/NoSQL)

Sample Workflow

[Sensor Node] → [MQTT Broker] → [Node.js Server] → [Database] →

[AI Module] → [Decision Engine] → [User Notification/UI Update]

---

6. Development Workflow

Development Workflow – Short Notes

- **Requirement Analysis**

Identify city needs: pollution control, waste management, energy usage, etc.

Define user roles: admin, citizen, operator.

- **System Design**

Create system architecture (frontend, backend, IoT, AI).

Design data flow and communication protocols.

- **Technology Stack Finalization**

Frontend: React.js

Backend: Node.js + Express

Database: MongoDB / PostgreSQL

IoT: ESP32, sensors (air, water, waste)

AI: Python + ML libraries (e.g., Scikit-learn, TensorFlow)

- **Frontend Development**

Build dashboards for citizens and administrators.

Implement charts, maps, and alerts UI.

- **Backend Development**

Set up REST APIs for data interaction.

Manage authentication and role access.

Integrate AI predictions and alert logic.

- **IoT Integration**

Program sensors using Arduino/ESP32.

Use MQTT/HTTP to transmit sensor data to the server.

- **Database Setup**

Design schema for users, sensor data, and alerts.

Store data securely and efficiently.

- **AI/ML Model Development**

Train models for pollution prediction, bin fill estimation.

Deploy as microservices using Flask/FastAPI.

- **Notification System**

Set up SMS, email, and push alerts using Twilio or Firebase.

- **Testing**

Perform unit testing, integration testing, and hardware testing.

Test alert thresholds, sensor accuracy, and UI flow.

- **Deployment**

Deploy frontend (Vercel/Firebase), backend (Render/Heroku).

Host database on cloud (MongoDB Atlas or PlanetScale).

- **Monitoring & Maintenance**

7.Setup and Installation

This section outlines the steps required to set up and install the Sustainability Smart City Assistant on your local machine or server environment.

- **Prerequisites**

Before setting up the project, ensure you have the following tools and software installed:

Python 3.8+

Node.js (v16 or later) – for frontend (if applicable)

MongoDB – or another database system used in the project

Git – for version control

Virtual Environment (venv or virtualenv) – for Python dependency isolation

Postman – for testing APIs (optional)

- **Clone the Project Repository**

Open your terminal or command prompt and run:

git clone https://github.com/your-username/sustainability-smart-assistant.git

cd sustainability-smart-assistant

- **Setup Backend (Python/Flask/Django)**

a. Create a virtual environment

python -m venv venv

source venv/bin/activate   # For Linux/Mac

venv\Scripts\activate   # For Windows

b. Install dependencies

pip install -r requirements.txt

c. Configure environment variables

Create a .env file in the root directory and set values like:

FLASK_ENV=development

DATABASE_URL=mongodb://localhost:27017/smartcity

SECRET_KEY=your_secret_key

d. Run the backend server

- **Setup Frontend (React/Angular/HTML-CSS-JS)**

If the project includes a web dashboard or interface:

cd client

npm install

npm start

- **Database Initialization**
- **Testing the API**

Use Postman or a browser to test endpoints like:

Optional: Docker Setup

If your project supports Docker:

docker-compose up --build

- **Deployment (Optional)**

To deploy on cloud or remote server (e.g., Heroku, AWS, Azure):

Configure cloud environment

Set environment variables

Deploy backend and frontend following service-specific instructions

---

8. Features and Functionality

- **Prerequisites**

Before setting up the project, ensure you have the following tools and software installed:

Python 3.8+

Node.js (v16 or later) – for frontend (if applicable)

MongoDB – or another database system used in the project

Git – for version control

Virtual Environment (venv or virtualenv) – for Python dependency isolation

Postman – for testing APIs (optional)

- **Clone the Project Repository**

Open your terminal or command prompt and run:

git clone https://github.com/your-username/sustainability-smart-assistant.git

cd sustainability-smart-assistant

- **Setup Backend (Python/Flask/Django)**

a. Create a virtual environment

python -m venv venv

source venv/bin/activate   # For Linux/Mac

venv\Scripts\activate   # For Windows

b. Install dependencies

pip install -r requirements.txt

c. Configure environment variables

Create a .env file in the root directory and set values like:

FLASK_ENV=development

DATABASE_URL=mongodb://localhost:27017/smartcity

SECRET_KEY=your_secret_key

d. Run the backend server

python app.py

or

flask run

- **Setup Frontend (React/Angular/HTML-CSS-JS)**

If the project includes a web dashboard or interface:

cd client

npm install

npm start

This will start the frontend server on http://localhost:3000/.

- **Database Initialization**

If MongoDB is used:

Ensure MongoDB is running:

mongod

Create necessary collections and indexes using scripts (if provided):

python db_setup.py

- **Testing the API**

Use Postman or a browser to test endpoints like:

http://localhost:5000/api/status

http://localhost:5000/api/sensors

- **Optional:** Docker Setup

If your project supports Docker:

docker-compose up --build

- **Deployment (Optional)**

To deploy on cloud or remote server (e.g., Heroku, AWS, Azure):

Configure cloud environment

Set environment variables

Deploy backend and frontend following service-specific instructions

- **Installation Complete!**

Once these steps are completed, the Sustainability Smart City Assistant should be fully functional. Make sure all services (backend, frontend, and database) are running correctly.

Setup and Installation of Sustainability Smart City Assistant

This section outlines the steps required to set up and install the Sustainability Smart City Assistant on your local machine or server environment.

- **Prerequisites**

Before setting up the project, ensure you have the following tools and software installed:

Python 3.8+

Node.js (v16 or later) – for frontend (if applicable)

MongoDB – or another database system used in the project

Git – for version control

Virtual Environment (venv or virtualenv) – for Python dependency isolation

Postman – for testing APIs (optional)

- **Clone the Project Repository**

Open your terminal or command prompt and run:

git clone https://github.com/your-username/sustainability-smart-assistant.git

cd sustainability-smart-assistant

- **Setup Backend (Python/Flask/Django)**

a. Create a virtual environment

python -m venv venv

source venv/bin/activate   # For Linux/Mac

venv\Scripts\activate   # For Windows

b. Install dependencies

pip install -r requirements.txt

c. Configure environment variables

Create a .env file in the root directory and set values like:

FLASK_ENV=development

DATABASE_URL=mongodb://localhost:27017/smartcity

SECRET_KEY=your_secret_key

d. Run the backend server

python app.py

or

flask run

- **Setup Frontend (React/Angular/HTML-CSS-JS)**

If the project includes a web dashboard or interface:

cd client

npm install

npm start

This will start the frontend server on http://localhost:3000/.

- **Database Initialization**

If MongoDB is used:

Ensure MongoDB is running:

mongod

Create necessary collections and indexes using scripts (if provided):

python db_setup.py

- **Testing the API**

Use Postman or a browser to test endpoints like:

http://localhost:5000/api/status

http://localhost:5000/api/sensors

- **Optional: Docker Setup**

If your project supports Docker:

docker-compose up --build

- **Deployment (Optional)**

To deploy on cloud or remote server (e.g., Heroku, AWS, Azure):

Configure cloud environment

Set environment variables

Deploy backend and frontend following service-specific instructions

---

9. API Documentation

Authentication

Use Bearer Token in the header:

Authorization: Bearer <token>

- **Sensor Data**

GET /api/sensors

→ Get all sensor data (air, noise, water, etc.)

POST /api/sensors

→ Submit new sensor data

Body:

```
{
  "type": "air_quality",
  "location": "Zone A",
  "value": 90
}
```

- **Waste Management**

GET /api/waste/bins

→ Get smart bin fill levels

PUT /api/waste/bins/{id}

→ Update bin status

Body:

```
{
  "fill_level": "50%"
}
```

- **Traffic & Parking**

GET /api/traffic

→ View traffic congestion and routes

GET /api/parking

→ Check available parking slots

- **Water System**

GET /api/water/quality

→ Get water quality status

GET /api/water/leaks

→ List detected pipeline leaks

- **Feedback System**

POST /api/feedback

→ Submit complaint or feedback

Body:

```
{
  "subject": "Leak in street tap",
  "location": "Block 3"
}
```

GET /api/feedback/status/{id}

→ Track complaint status

- **Admin Reports**

GET /api/admin/reports

→ Get sustainability performance summaries

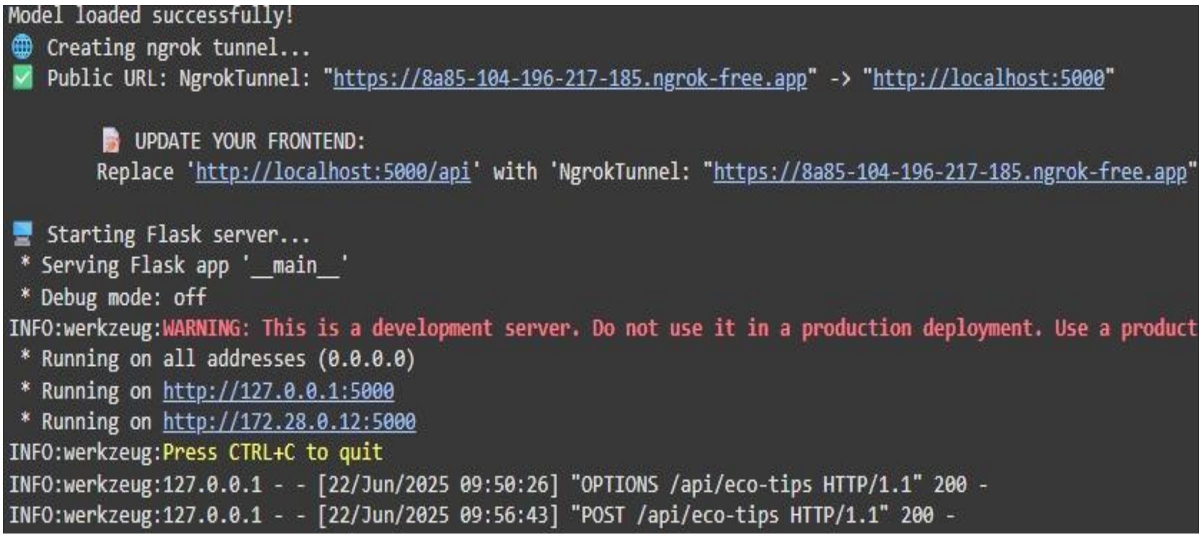
- **User Auth**

POST /api/auth/login

POST /api/auth/register

---

10. Screenshots and Results



```
# Cell 1: Install dependencies and setup
!pip install transformers torch accelerate huggingface_hub pandas numpy sci

# Import required libraries
import os
import torch
import pandas as pd
import numpy as np
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from datetime import datetime, timedelta
import json
import re
from flask import Flask, request, jsonify
from flask_cors import CORS
import threading
import time
from pyngrok import ngrok
```

- **Videos Link**
- https://drive.google.com/file/d/1NU1fw_hiIxYTnTDTKeedvH04QqEUHZb4/view?usp=sharing
- https://github.com/Deekshith6049/my-web-app.git

---

11. Challenges and Solutions

- **Challenge: Real-time Data Collection**

Issue: Continuous sensor data can be inconsistent due to connectivity issues or sensor failure.

Solution: Use reliable IoT protocols (MQTT/HTTP) and buffer data locally before syncing to the server.

Challenge: Data Overload and Management

Issue: High volume of sensor data can slow down processing and storage

Solution: Implement data filtering, compression, and cloud-based storage with auto-archiving.

- **Challenge:** Integration of Multiple Systems

Issue: Difficult to unify air, water, traffic, and waste systems under one platform.

Solution: Use a modular architecture with well-defined APIs to integrate diverse services.

- **Challenge:** Hardware Maintenance

Issue: Sensor devices may malfunction or require regular maintenance.

Solution: Set up automatic alerts and predictive maintenance using anomaly detection.

- **Challenge:** Ensuring User Participation

Issue: Low engagement from citizens in reporting issues or using services.

Solution: Provide an easy-to-use mobile app and include gamified features or rewards.

- **Challenge:** Data Security and Privacy

Issue: Sensitive data from sensors and users can be exposed.

Solution: Encrypt data, implement authentication, and comply with data protection standards.

- **Challenge:** Scalability

Issue: Difficult to scale system across more city areas or devices.

Solution: Use cloud services (like AWS/Azure) and scalable microservices architecture.

12.Known Issues

1. Data Privacy & Security
   Risk of data breaches and surveillance concerns.
2. High Implementation Cost
   Expensive infrastructure and technology investment.
3. Digital Divide
   Unequal access to smart services among citizens.
4. Lack of Standardization
   Poor integration due to different technologies and platforms.
5. Environmental Impact
   Energy use and e-waste from smart devices.
6. Urban Planning Challenges
   Difficulty upgrading old infrastructure in existing cities.
7. Low Citizen Engagement
   Lack of awareness or trust in smart city initiatives.
8. Weak Governance
   Slow policy updates and regulatory gaps.
9. Maintenance Issues
   High long-term costs and need for continuous upgrades.
10. Climate Adaptability
    Inadequate planning for climate-related risks.

13.Future Enhancements:

1. Enhanced Traffic Monitoring

Integrate real-time traffic data from APIs like Google Maps or OpenStreetMap.

Provide predictive traffic analysis based on historical data and AI.

Suggest eco-friendly travel options like public transport schedules or bike routes.

2. Smart Energy Monitoring

Integrate IoT devices to monitor real-time energy consumption in different zones.

Provide personalized energy-saving tips based on household or zone data.

3. Voice-Activated Assistant

Add voice recognition capabilities for hands-free interactions.

Enable multi-language support to cater to diverse populations.

4. Sentiment Analysis for Feedback

Analyze citizen feedback to gauge public sentiment on urban policies and services.

Prioritize issues based on sentiment trends (e.g., anger or frustration).

5. Advanced Anomaly Detection

Use deep learning techniques for detecting complex anomalies in large datasets.

Automate alerts and trigger actions for anomalies like unauthorized constructions.

6. Data Visualization

Add interactive data visualization dashboards with drill-down capabilities.

Provide heatmaps for traffic, energy consumption, or citizen feedback data.

7. Social Media Integration

Enable feedback and issue reporting through social media platforms like Twitter and Facebook.

Use AI to monitor public discussions on urban topics for insights.

8. Integration with Smart Devices

Connect with smart home devices (e.g., Alexa, Google Home) for personalized eco-advice.

Collect data from smart meters for accurate KPI forecasting.

9. Gamification

Reward citizens for sustainable actions (e.g., recycling, using public transport) through gamification elements like badges or leaderboards.

10. Blockchain for Policy Transparency

Use blockchain technology to track and verify city policies and decisions.

Ensure transparency and accountability in urban governance.

11. Predictive Maintenance

Integrate predictive maintenance for city infrastructure like roads, pipelines, and buildings.

Use AI models to forecast wear and tear based on historical data.

12. Cloud-Native Deployment

Migrate the backend to cloud platforms for better scalability and performance.

Use serverless architecture to reduce costs and improve deployment flexibility.

13. Citizen Engagement via Mobile App

Develop a mobile app version of the platform for better accessibility.

Allow citizens to receive instant notifications and updates on city developments.

14. Advanced Reporting

Enable automated generation of monthly or yearly city performance reports for administrators.

Include KPIs, anomaly highlights, and citizen satisfaction metrics.

14.Conclusion

The **Sustainable Smart City Assistant Using IBM Granite LLM** is a pioneering initiative aimed at transforming urban governance and sustainability through the power of AI. By integrating advanced technologies like IBM Watsonx Granite LLM, real-time traffic monitoring, machine learning-based KPI forecasting, and anomaly detection, the platform provides actionable insights to city administrators, empowers citizens with interactive tools, and promotes eco-friendly urban practices. This innovative solution not only addresses the immediate challenges of urban management but also lays a strong foundation for future smart city advancements. With its modular architecture, scalability, and citizen-centric design, the assistant is poised to make cities smarter, more sustainable, and more inclusive, ensuring a better quality of life for all.