SUSTAINABLE SMART CITY

Team ID : NM2025TMID01905

Team Members:

-Name1 V.Deepalakshmi (TL) NM-IDEF540A2C5E1E728B2C44091A98305387

-Name2 T.Thirumalai NM-ID:D5E7DBBDE4E4E972D8462448C50EE9C40E

-Name3 S.SaranKumar NM-ID:A034FA6EED091439746418F0A44E02CD

-Name4 R.Santhosh NM-ID:AC33E6BD4138F83BB0342EC7D633E548

1. INTRODUCTION

The Sustainable Smart City Assistant project aims to modernize the way cities manage resources, engage with citizens, and make data-driven governance decisions. By integrating IBM Watsonx's Granite LLM with advanced data pipelines and predictive analytics, the system provides real-time insights, feedback handling, and sustainability advice.

2. PROJECT OVERVIEW

2.1 Purpose:

The purpose of this project is to build an AI-powered assistant to support urban sustainability, governance, and citizen engagement. It empowers municipal planners with fast policy summarization, citizens with an easy reporting mechanism, and administrators with KPI forecasting tools.

2.2 Features:

-City Health Dashboard

-Citizen Feedback

-DocumentSummarization

-Eco-Advice

-Anomaly Detection

-Chat Assistant

3. ARCHITECTURE

3.1 System Components

3.1.1 Frontend: Streamlit Dashboard

The frontend is developed using Streamlit, chosen for its simplicity, interactivity, and ability to rapidly prototype dashboards. It provides a clean and responsive interface for both citizens and administrators.

Key features include:

- Citizen View: Feedback submission forms, eco-advice chatbot, and document upload for policy summarization.
- Administrator View: Access to KPI visualizations, anomaly alerts, and aggregated citizen reports.
- Interactivity: Graphs, filters, and search options for real-time data exploration.
- Accessibility: Mobile and desktop friendly, ensuring all users can easily interact.
- Security: Citizens only see their submissions, while admins have role-based access to dashboards.

The Streamlit UI acts as the first touchpoint for both citizens and planners, ensuring smooth engagement with minimal training required.

3.1.2 Backend: FastAPI

The backend is powered by FastAPI, a high-performance Python framework ideal

.

for asynchronous API calls. It acts as the central orchestrator of the system.

Responsibilities include:

•Request Handling: Manages citizen feedback, document uploads, and KPI datasets.

•AI Integration: Routes requests to IBM Granite LLM for summarization, eco-advice, and chatbot responses.

•Data Processing: Cleans, validates, and formats input data for further analysis.

•Security & Authentication: Implements JWT-based authentication, ensuring safe citizen– admin interactions.

•Scalability: Designed as modular microservices that can be scaled independently using Docker/Kubernetes.

•Error Handling & Logging: Provides transparency into system performance and quick debugging.

The FastAPI backend is the bridge between the frontend and the AI/ML services.

3.1.3 AI Models: IBM Granite LLM + Machine Learning

The AI layer is the intelligence backbone of the project, combining IBM Watsonx's Granite LLM with machine learning models.

•IBM Granite LLM:

oPerforms document summarization, citizen query answering, and chatbot interactions.

oSupports natural language understanding, contextual reasoning, and eco-friendly advice generation.

.

oCan be fine-tuned for urban governance datasets for domain-specific accuracy.

•Machine Learning Models:

oProphet (Forecasting): Used for predicting KPI trends such as water usage, energy demand, and waste generation.

oIsolation Forest (Anomaly Detection): Identifies unusual spikes in KPIs (e.g., sudden rise in electricity usage or water leaks).

oContinuous Learning: Models adapt as new KPI datasets are uploaded, improving accuracy over time.

Together, these models enable predictive, context-aware, and sustainable decision-making.

3.1.4 Database: PostgreSQL & MongoDB

The system uses a hybrid database approach to manage both structured and unstructured data:

•PostgreSQL:

oStores structured KPI data (e.g., water consumption, electricity usage). o

Supports relational queries for forecasting and performance monitoring.

oEnsures ACID compliance for consistency and security.

•MongoDB:

oStores unstructured feedback from citizens.

oHandles dynamic categories like “Water Issues”, “Waste Management”, or

“Streetlight Faults”.

oOffers flexibility for large-scale text data, suitable for NLP processing.

The combination of SQL and NoSQL ensures both reliability and flexibility.

3.2 Data Flow

The overall data workflow is as follows:

1.Citizen Interaction: A citizen submits feedback (e.g., water leakage), uploads a document (policy file), or queries the chatbot.

2.Backend Validation: FastAPI verifies the request (format, authentication) and logs metadata (timestamp, user ID).

3.AI Processing:

oIBM Granite LLM interprets natural language queries and generates summaries, chatbot responses, or eco-advice.

oML models (Prophet/Isolation Forest) analyze uploaded KPI data to generate forecasts or detect anomalies.

4.Database Storage: Processed feedback, query logs, and KPI datasets are stored in PostgreSQL and MongoDB.

5.Dashboard Visualization: Streamlit retrieves processed results and presents them to citizens or administrators via charts, tables, and alerts.

This end-to-end flow ensures seamless interaction, accurate AI responses, and actionable insights for city governance.

4. SETUP INSTRUCTIONS

.

4.1 Prerequisites

To run the Sustainable Smart City Assistant, the following tools and dependencies are required:

Python 3.10+

The core backend (FastAPI, ML models, IBM Granite integration) is written in Python.

Python 3.10 introduces structural pattern matching, improved type hinting, and better async support—features that are heavily used in this project.

Key libraries: FastAPI, Uvicorn, Pandas, Prophet, Scikit-learn, PyMuPDF.

Node.js (Optional, for extended frontend)

If the frontend dashboard is expanded beyond Streamlit (e.g., into React-based interactive panels), Node.js will be required.

Provides package management with npm or yarn for frontend dependencies.

Enables live reloading and efficient build processes.

Docker (Optional, for deployment)

Used for containerizing backend, frontend, and database services.

Ensures consistency across local, staging, and production environments.

Docker Compose can orchestrate multiple containers (FastAPI, Streamlit, PostgreSQL, MongoDB).

Recommended for production deployments on IBM Cloud.

PostgreSQL + MongoDB

PostgreSQL for KPI data (structured).

MongoDB for citizen feedback and logs (unstructured).

Both can be installed locally or provisioned as managed services in IBM Cloud.

.

4.2 Installation

Step 1: Clone Repository

Clone the latest version of the project from GitHub (or any version control system used by your team).

git clone https://github.com/<username>/sustainable-smart-city-

assistant.git cd sustainable-smart-city-assistant

Step 2: Create Virtual Environment

It is recommended to create a virtual environment to isolate project dependencies.

python -m venv venv source

venv/bin/activate # For Linux/

Mac venv\Scripts\activate #

For Windows

Step 3: Install Dependencies

Install Python dependencies for backend and ML services:

pip install -r requirements.txt

Dependencies include: fastapi (API

framework) uvicorn (ASGI server) prophet

(KPI forecasting) scikit-learn (anomaly

detection) pymongo (MongoDB connector)

psycopg2 (PostgreSQL connector) python-

dotenv (environment variable management)

Step 4: Configure Environment Variables

Create a .env file in the project root. This file stores sensitive configurations:

IBM_GRANITE_API_KEY=your_ibm_api_key

POSTGRES_URL=postgresql://user:password@localhost:5432/smartcity

MONGO_URL=mongodb://localhost:27017

SECRET_KEY=your_jwt_secret

Note: Do not commit .env files to version control.

Step 5: Setup Database

Run the database schema setup scripts:

psql -U postgres -d smartcity -f database/schema.sql

For MongoDB, no schema is required, but collections will be created dynamically upon first insert.

4.3 Running the Application

Run FastAPI Backend

Start the backend API using Uvicorn:

uvicorn main:app –reload

Runs on http://127.0.0.1:8000 by default.

Interactive API docs available at http://127.0.0.1:8000/docs.

Hot reloading enabled with –reload.

Run Streamlit Frontend

Start the citizen/admin

dashboard:

streamlit run dashboard.py

Runs on http://localhost:8501.

Provides document upload, feedback form, and visualization panels.

Run with Docker (Optional)

Build and start all services using Docker

Compose: docker-compose up –build This

launches:

FastAPI backend (port 8000)

Streamlit frontend (port 8501)

PostgreSQL (port 5432)

MongoDB (port 27017)

5. FOLDER STRUCTURE

The project is organized into a modular folder structure to maintain clarity, scalability, and ease of collaboration among developers. Each folder has a well-defined purpose.

5.1 backend

This directory contains the entire FastAPI backend codebase that powers the API layer, integrates with AI models, and manages business logic.

main.py

Acts as the entry point for the FastAPI application.

.

Initializes the server, loads configurations, and registers all

routes. Provides access to Swagger API Docs at /docs and

Redoc at /redoc.

routes/

Stores all API endpoint definitions.

Divided into submodules for different functionalities, e.g.:

feedback.py Handles citizen feedback submission and retrieval.
summarization.py

Manages document upload and IBM Granite LLM summarization. forecast.py

Implements KPI forecasting using Prophet.

auth.py Manages login, registration, and JWT authentication.

This separation ensures better maintainability and scalability.

models/

Defines database schemas and ORM models.

PostgreSQL models for structured data like KPIs.

MongoDB models for unstructured data like citizen feedback.

Helps enforce data validation and consistency.

utils/ (optional, recommended)

Contains reusable helper functions such as:

Logging utilities.

Database connection handlers.

Input validation functions.

API response formatting.

.

5.2 frontend

This directory contains the Streamlit-based dashboard for both citizens and administrators.

dashboard.py

Main entry point for the Streamlit application.

Provides UI components like:

Feedback submission form.

Document upload & summarization panel.

KPI charts and anomaly detection graphs.

Eco-advice chatbot integration.

Fetches data dynamically from the FastAPI backend via REST API calls.

assets/ (optional)

Stores static files such as images, icons, and style sheets used in the dashboard.

5.3 database

This folder manages database setup and initialization

scripts. schema.sql

SQL script for setting up PostgreSQL tables.

Includes table definitions for KPIs, user accounts, and feedback

logs. Supports migrations and ensures consistent schema

across environments. mongo_setup.js (optional)

A script for initializing MongoDB collections used for citizen feedback storage.

.

5.4 tests

Contains automated tests for ensuring the reliability of the system.

Unit Tests

Validate individual functions (e.g., summarization, forecasting).

Example: Testing Prophet model predictions against sample CSV input.

Integration Tests

Validate the interaction between multiple modules.

Example: Submitting feedback through API Storing in DB Retrieving

on dashboard. Test Frameworks pytest for backend unit tests.

unittest or pytest-streamlit for frontend testing.

5.5 docs

This folder stores project documentation and supporting materials.

API References Explains available API endpoints.

Setup Guides Step-by-step installation and deployment instructions.

Architecture Diagrams Illustrates system design and data flow.

Reports For submission in academic/industrial contexts.

5.6 requirements.txt

Lists all Python dependencies required for the backend.

Ensures consistent environments across development, testing, and deployment.

.

Can be installed using:

pip install -r requirements.txt

5.7 README.md

High-level overview of the project.

Provides purpose, setup steps, and a quick start guide.

Acts as the first entry point for new developers.

6. MODULE: DOCUMENT SUMMARIZATION

Example FastAPI endpoint:

```python
from fastapi import FastAPI, UploadFile
import fitz

app = FastAPI()

@app.post('/summarize') async def
summarize_document(file: UploadFile):     text =
"
   pdf = fitz.open(stream=await file.read(),
filetype='pdf')     for page in pdf:
       text += page.get_text()
summary =
granite_summarize(text)     return
{"summary": summary}
```

.

```

## 7. MODULE: CITIZEN FEEDBACK

Example FastAPI endpoint:

```python from pydantic import
BaseModel

class Feedback(BaseModel):
    category: str
description: str
location: str

@app.post('/feedback') async def
submit_feedback(feedback: Feedback):
    save_to_db(feedback.dict())
return {"status": "success"}
```

8. MODULE: KPI FORECASTING

```python import pandas
as pd from prophet
import
Prophet

def forecast_kpi(csv_path):     df =
pd.read_csv(csv_path)      df =
df.rename(columns={"date": "ds", "value":

.

```
"y"})     model = Prophet()     model.fit(df)
future =
model.make_future_dataframe(periods=365)
  forecast = model.predict(future)
    return forecast[['ds', 'yhat']]
```

9. API DOCUMENTATION

Authentication:

-POST /login – User login

-POST /register – User registration

Citizen APIs:

-POST /feedback – Submit issue

-POST /summarize – Upload policy document

-POST /forecast – Upload KPI data for forecasting

10. USER INTERFACE

10.1 Citizen Dashboard:

-Feedback submission form

-Policy summarization upload

-Eco-advice chatbot

10.2 Admin Dashboard:

-View aggregated feedback

-Monitor KPIs

-View anomalies and forecasts

11. SECURITY & AUTHENTICATION

-JWT for API security

-Role-based access (Citizen/Admin)

-TLS/SSL encryption

-Password hashing

12. TESTING

Testing is a critical part of the development process for the Sustainable Smart City Assistant. It ensures that all modules—backend, frontend, AI models, and databases—work correctly and reliably under different scenarios. The testing strategy is divided into three main levels:

12.1 Unit Testing (FastAPI Routes)

Unit testing focuses on verifying individual components of the backend system.

•Scope:

- FastAPI endpoints such as /feedback, /summarize, and /forecast.
- Utility functions like data validation, file parsing, and API response formatting.
- Database interaction modules to ensure queries are stored and retrieved correctly.

•Methodology:

.

opytest is used as the main testing framework.

oMock databases (SQLite or in-memory stores) replace production databases during tests.

oSample input data is fed into each route to check if outputs match expected results.

•Example:

oUploading a sample PDF to /summarize should return a non-empty summary.

oSubmitting citizen feedback should return a success status and store the entry in the database.

•Benefits:

oDetects bugs early in development.

oEnsures backend routes remain stable even after code changes.

12.2 Integration Testing (Backend + Frontend)

Integration testing validates that different system modules work together seamlessly.

•Scope:

oInteraction between FastAPI backend, IBM Granite LLM, ML models, and databases.

oCommunication between Streamlit frontend and FastAPI APIs.

oEnd-to-end data flow from citizen input backend processing AI output dashboard visualization.

•

•Methodology:

oTest environments simulate multiple concurrent users. o Tools like pytest-asyncio are used to test asynchronous API calls.

oStreamlit test clients or Selenium are used to simulate frontend actions.

•Example:

oCitizen submits water leakage feedback API stores entry in MongoDB Dashboard retrieves and displays it correctly.

oAdmin uploads KPI CSV Forecasting model runs Results appear as graphs on the dashboard.

•Benefits:

oDetects mismatches in data formats and communication protocols.

oEnsures overall system reliability before deployment.

12.3 User Acceptance Testing (UAT)

UAT validates that the system meets real-world expectations of both citizens and administrators.

•Scope:

oCitizens test chatbot responses, feedback submission, and eco-advice suggestions.

oAdministrators test dashboards for KPI forecasts, anomaly alerts, and policy summaries.

oAccessibility and multilingual capabilities are evaluated.

•

•Methodology:

oConduct pilot testing sessions with actual users (e.g., municipal staff, citizens). o

Collect feedback on usability, clarity of responses, and system speed.

oEvaluate responsiveness across different devices (desktop, tablet, mobile).

•Example:

oCitizens confirm whether summaries of policy documents are easy to understand.

oAdmins confirm whether forecasts are realistic and useful for planning.

•Benefits:

oEnsures the assistant is user-friendly and practical.

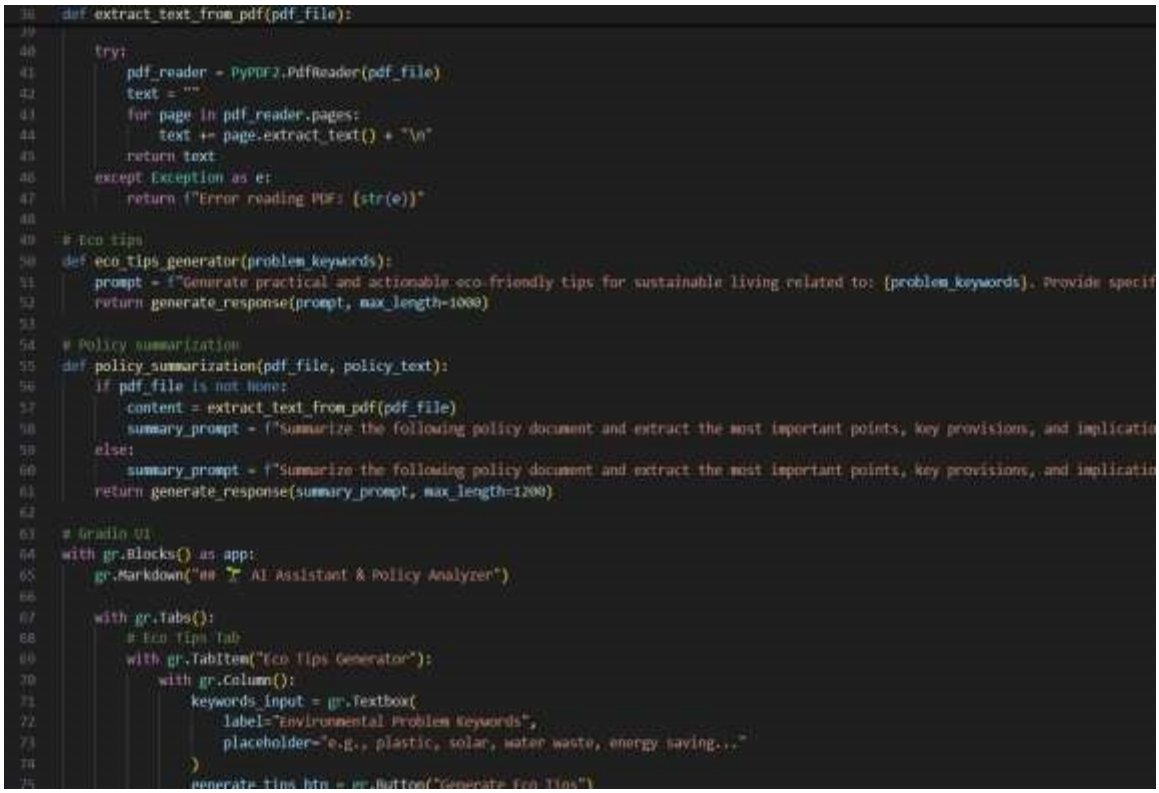
Identifies usability issues that may not appear in technical testing.

oBuilds trust among stakeholders before live deployment.

13. SCREENSHOTS

SOURCE CODE

•



```
def extract_text_from_pdf(pdf_file):

    try:
        pdf_reader = PyPDF2.PdfReader(pdf_file)
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text() + "\n"
        return text
    except Exception as e:
        return f"Error reading PDF: {str(e)}"

# Eco tips
def eco_tips_generator(problem_keywords):
    prompt = f"Generate practical and actionable eco-friendly tips for sustainable living related to: {problem_keywords}. Provide specif
    return generate_response(prompt, max_length=1000)

# Policy summarization
def policy_summarization(pdf_file, policy_text):
    if pdf_file is not None:
        content = extract_text_from_pdf(pdf_file)
        summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implicatio
    else:
        summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implicatio
    return generate_response(summary_prompt, max_length=1200)

# Gradio UI
with gr.Blocks() as app:
    gr.Markdown("## 🌍 AI Assistant & Policy Analyzer")

    with gr.Tabs():
        # Eco Tips Tab
        with gr.TabItem("Eco Tips Generator"):
            with gr.Column():
                keywords_input = gr.Textbox(
                    label="Environmental Problem Keywords",
                    placeholder="e.g., plastic, solar, water waste, energy saving..."
                )
                generate_tips_btn = gr.Button("Generate Eco Tips")
```

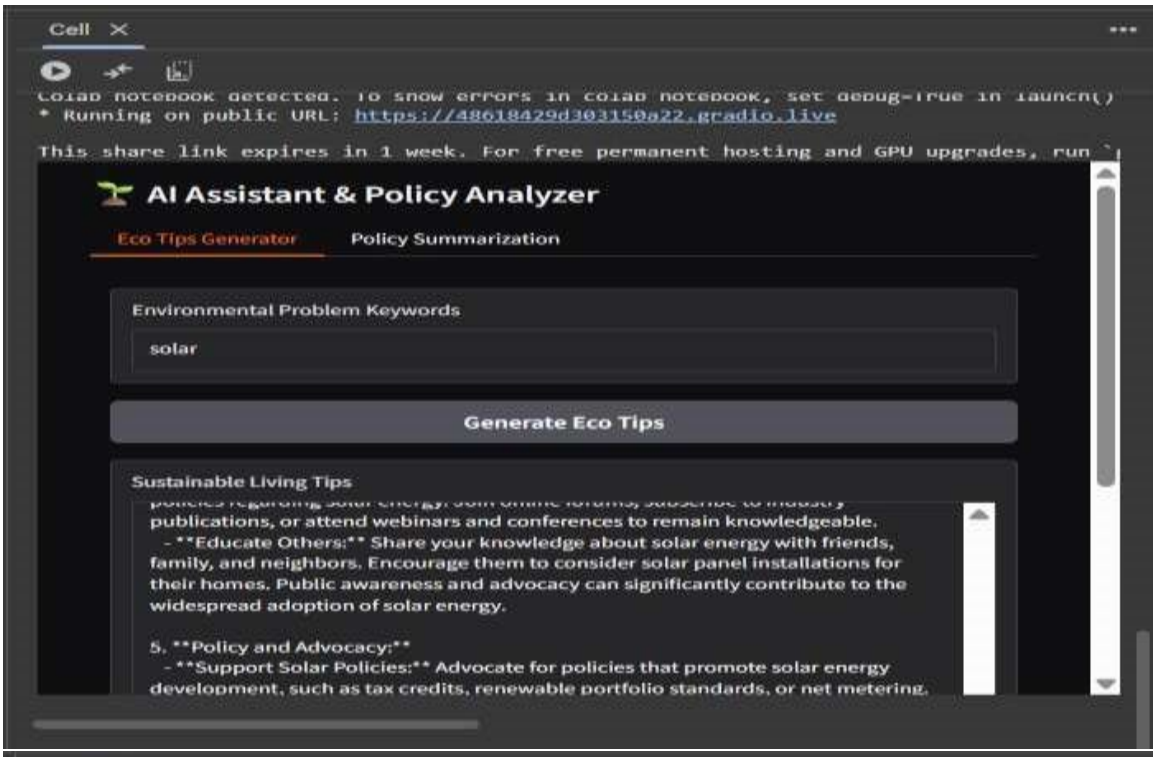
```python
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import PyPDF2

# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# Text generation
def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)
    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()
    return response

# Extract PDF text
def extract_text_from_pdf(pdf_file):
    if pdf_file is None:
        return ""
```

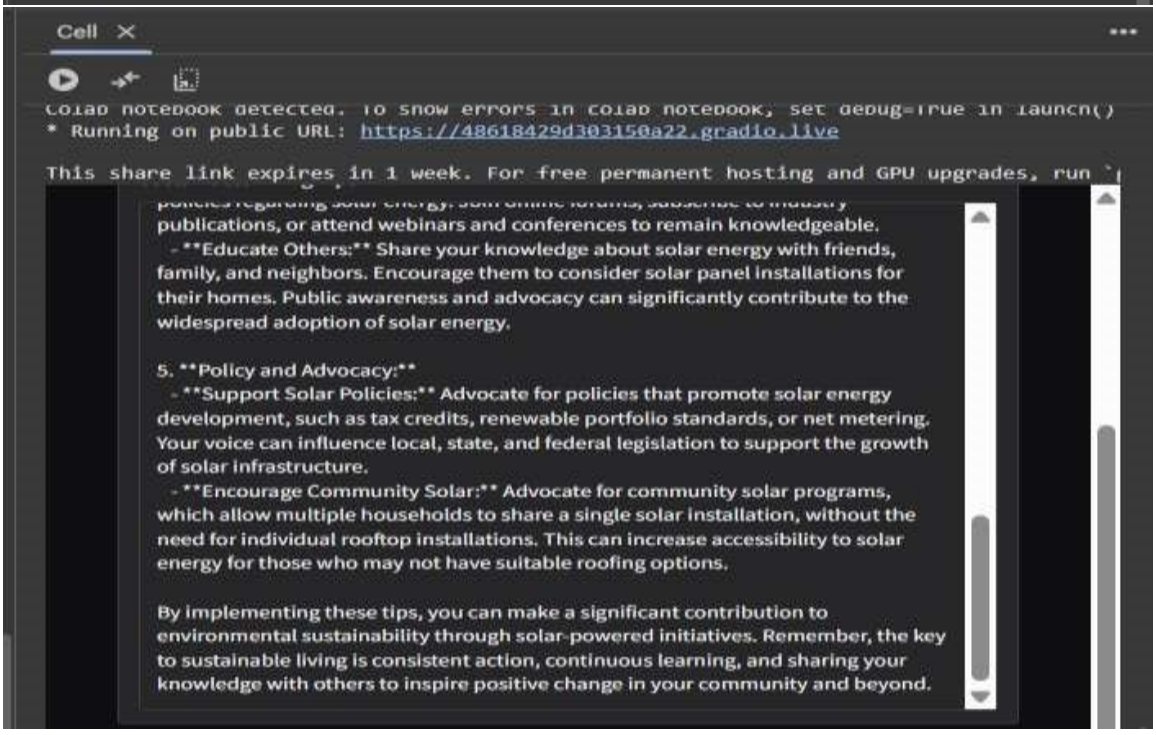
```python
# Gradio UI
with gr.Blocks() as app:
    gr.Markdown("## 🌱 AI Assistant & Policy Analyzer")

    with gr.Tabs():
        # Eco Tips Tab
        with gr.TabItem("Eco Tips Generator"):
            with gr.Column():
                keywords_input = gr.Textbox(
                    label="Environmental Problem Keywords",
                    placeholder="e.g., plastic, solar, water waste, energy saving..."
                )
                generate_tips_btn = gr.Button("Generate Eco Tips")
                tips_output = gr.Textbox(label="Sustainable Living Tips", lines=13)
                                                                    (variable) tips_output: Any
            generate_tips_btn.click(eco_tips_generator, inputs=keywords_input, outputs=tips_output)

        # Policy Summarization Tab
        with gr.TabItem("Policy Summarization"):
            with gr.Row():
                with gr.Column():
                    pdf_upload = gr.File(label="Upload Policy PDF", file_types=[".pdf"])
                    policy_text_input = gr.Textbox(
                        label="Or paste policy text here",
                        placeholder="Paste policy document text...",
                        lines=5
                    )
                    summarize_btn = gr.Button("Summarize Policy")

                with gr.Column():
                    summary_output = gr.Textbox(label="Policy Summary & Key Points", lines=20)

            summarize_btn.click(policy_summarization, inputs=[pdf_upload, policy_text_input], outputs=summary_output)

app.launch(share=True)
```

.

OUTPUT

.


Cell
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://48618429d303150a22.gradio.live
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run
AI Assistant & Policy Analyzer
Eco Tips Generator
Policy Summarization
Environmental Problem Keywords
solar
Generate Eco Tips
Sustainable Living Tips
publications, or attend webinars and conferences to remain knowledgeable.
- **Educate Others:** Share your knowledge about solar energy with friends, family, and neighbors. Encourage them to consider solar panel installations for their homes. Public awareness and advocacy can significantly contribute to the widespread adoption of solar energy.
5. **Policy and Advocacy:**
- **Support Solar Policies:** Advocate for policies that promote solar energy development, such as tax credits, renewable portfolio standards, or net metering.


Cell
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://48618429d303150a22.gradio.live
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run
publications, or attend webinars and conferences to remain knowledgeable.
- **Educate Others:** Share your knowledge about solar energy with friends, family, and neighbors. Encourage them to consider solar panel installations for their homes. Public awareness and advocacy can significantly contribute to the widespread adoption of solar energy.
5. **Policy and Advocacy:**
- **Support Solar Policies:** Advocate for policies that promote solar energy development, such as tax credits, renewable portfolio standards, or net metering. Your voice can influence local, state, and federal legislation to support the growth of solar infrastructure.
- **Encourage Community Solar:** Advocate for community solar programs, which allow multiple households to share a single solar installation, without the need for individual rooftop installations. This can increase accessibility to solar energy for those who may not have suitable roofing options.
By implementing these tips, you can make a significant contribution to environmental sustainability through solar-powered initiatives. Remember, the key to sustainable living is consistent action, continuous learning, and sharing your knowledge with others to inspire positive change in your community and beyond.

.

14. CONCLUSION

The Sustainable Smart City Assistant represents a forward-looking approach to modern urban governance. By combining IBM Watsonx's Granite LLM, machine learning models, and a modular data pipeline, the system creates a unified platform that addresses both citizen needs and administrative decision-making.

This assistant bridges the gap between citizens and city governance by offering:

•Transparency: Citizens can easily access summaries of complex policies, submit feedback, and view how their inputs influence governance.

•Data-Driven Insights: Administrators gain access to real-time dashboards, KPI forecasts, and anomaly alerts that support evidence-based decision-making.

•Sustainability: The eco-advice module promotes responsible resource usage, while Citizen Engagement: A user-friendly chatbot, feedback mechanisms, and multilingual capabilities foster inclusivity and trust.

The platform also future-proofs city operations by supporting IoT integration, predictive maintenance, and gamification features that actively involve citizens in sustainability initiatives. While challenges such as multilingual limitations, computational costs, and reliance on stable internet exist, these are acknowledged and addressed in the project roadmap.

In conclusion, the Sustainable Smart City Assistant is more than a digital tool—it is a strategic enabler for smart, inclusive, and sustainable urban development. By leveraging advanced AI and real-time analytics, it empowers cities to operate more efficiently, involve citizens in meaningful ways, and plan for a resilient, eco-

friendly future.

15. KNOWN ISSUES

While the Sustainable Smart City Assistant provides robust functionality, some limitations exist in the current implementation. These issues are recognized as areas for improvement in future versions.

13.1 Limited Multilingual Support

At present, the assistant supports English and only a few regional languages.

- Challenge: Many citizens prefer interacting in their native languages. Current models may struggle with regional dialects, idioms, or less widely spoken languages.
- Impact:
  - Citizens may face difficulty when submitting queries or feedback in unsupported languages.
  - Policy summaries and chatbot responses may not be easily accessible to nonEnglish speakers.
- Cause: IBM Granite LLM and ML models are primarily fine-tuned on English or widely used languages, with limited datasets for regional languages.

Mitigation: A future roadmap includes expanding multilingual datasets, integrating translation APIs, and fine-tuning Granite LLM on local dialects.

13.2 High Computational Cost for Large Query Volumes

The AI-powered modules, particularly IBM Granite LLM and forecasting models,

require significant computational resources.

- Challenge: Handling thousands of queries simultaneously can increase latency and operational costs.
- Impact:
  - Slower response times during peak citizen engagement.
  - Higher cloud resource usage leads to increased deployment costs for city administrations.
- Cause: Natural Language Processing and time-series forecasting are compute-intensive tasks, especially when processing large policy documents or big KPI datasets.
- Mitigation: Optimizations such as query caching, model distillation, and edge processing can reduce inference costs. Horizontal scaling with Kubernetes can also balance workloads.

13.3 Requires Stable Internet Connectivity

The platform relies on continuous cloud connectivity for smooth operation.

- Challenge: Network disruptions can interrupt communication with the backend, databases, and AI services.
- Impact:
  - Citizens may experience delays or failed responses in low-connectivity areas.
  - Real-time features like dashboards and anomaly alerts become unreliable without a stable connection.

Cause: Since the system integrates IBM Granite LLM and cloud-hosted ML models, queries cannot be processed offline.

•Mitigation: Future versions can include offline-first support, local caching for recent queries, and progressive synchronization when connectivity is restored.

16. FUTURE ENHANCEMENTS

The Sustainable Smart City Assistant has been designed with modularity and scalability in mind. While the current version provides robust functionalities, several enhancements can further improve its effectiveness, inclusivity, and citizen engagement.

14.1 IoT Sensor Integration

The integration of Internet of Things (IoT) sensors can provide real-time data collection from city infrastructure.

•Use Cases: Smart water meters to detect leaks, air-quality sensors to monitor pollution, and energy meters to track consumption.

•Benefits:

oEnables real-time anomaly detection (e.g., water pipe bursts, sudden power outages).

oImproves predictive forecasting by combining live sensor data with historical records. o Provides alerts and notifications directly to administrators when thresholds are exceeded.

•Technical Approach:

oIoT devices send data to cloud services via MQTT or Kafka.

.

oData pipelines push this information into PostgreSQL/MongoDB for further analysis.

.

14.2 Multilingual Expansion

Language inclusivity is essential for citizen participation in diverse urban populations.

•Objective: Extend IBM Granite LLM and the chatbot to handle multiple languages (English, Hindi, Tamil, Telugu, etc.).

•Features:

oAutomatic translation and sentiment analysis across languages. o Citizens can submit feedback or queries in their preferred language.

oPolicy summaries can be generated in regional dialects.

•Benefits:

oIncreases adoption among non-English-speaking citizens. o Enhances trust in governance through accessible communication.

oEnsures greater inclusivity for marginalized groups.

14.3 Predictive Maintenance

The assistant can be extended to proactively identify infrastructure issues before failures occur.

•Use Cases: Predicting streetlight failures, water pump malfunctions, or waste collection delays.

•Benefits:

•

oReduces operational costs by minimizing emergency repairs.

oImproves service reliability and citizen satisfaction.

oExtends the lifespan of infrastructure assets.

•Technical Approach:

oCombine IoT sensor data with ML algorithms (e.g., time-series anomaly detection).

oGenerate predictive maintenance schedules for city administrators.

oSend automated maintenance tickets to field workers.

14.4 Gamification for Citizens

Introducing gamification features can encourage citizens to actively participate in sustainability efforts.

•Features:

oReward points for reporting issues, providing eco-friendly suggestions, or reducing household energy consumption.

oLeaderboards to showcase top eco-friendly neighborhoods or citizens.

oBadges and certificates for sustainable actions.

•Benefits:

oIncreases citizen engagement and sense of responsibility.

- oPromotes eco-conscious behavior in everyday life.
- oBuilds a collaborative environment between citizens and administrators.