# SUSTAINABLE SMART CITY ASSISTANT

**1.Introduction**

Project Title: Sustainable Smart City Assistant

Team Members

**LEADER : Sued Asif Basha H**

Anwar Basha S

Midhun B Sebastian

Vishnu kanth

Sathish Kumar K

**2. Project Overview**

Purpose:

The Sustainable Smart City Assistant empowers cities and residents to thrive in an eco-conscious and connected urban environment by leveraging AI and real-time data for resource optimization and community engagement.

## Features:

Conversational Interface: Natural language interaction for user queries and updates.

Policy Summarization: Converts government documents into actionable summaries.

Resource Forecasting: Predictive analytics for energy, water, and waste usage.

Eco-Tip Generator: Personalized sustainability advice based on user behavior.

Citizen Feedback Loop: Collects and analyzes public input for city planning.

KPI Forecasting: Supports officials with strategic planning through projections.

Anomaly Detection: Flags unusual patterns for early warnings.

Multimodal Input Support: Accepts text, PDFs, and CSVs for analysis.

Streamlit or Gradio UI: User-friendly interactive dashboards.

**3. Architecture**

Frontend:

Built with Streamlit, featuring dashboards, file upload, chat interface, and reports.
Modular pages with sidebar navigation using streamlit-option-menu.

Backend:

Fast API REST framework powers document processing, chatbot, tip generation, and vector embeddings with high performance and Swagger support.

LLM Integration: IBM Watsonx Granite models enable natural language understanding and generation for summaries, chats, and reports.

Vector Search:

Pinecone is used to store and retrieve document embeddings for semantic search using cosine similarity.

ML Modules:

Scikit-learn for forecasting and anomaly detection, with data handled via pandas and matplotlib visualizations.

4. Setup Instructions

Prerequisites:

Python 3.9+

pip and virtual environment

API keys for IBM Watsonx and Pinecone

Internet access

Installation:

Clone the repo

Install dependencies from requirements.txt

Configure .env with credentials

Launch FastAPI backend

Run Streamlit frontend

Upload data and interact with the assistant

## 5. Folder Structure text

```
app/                        #Backend logic and APIs

app/api/                    #Modular API routes

ui/                         #Frontend Streamlit components

smart_dashboard.py          #Main dashboard entry point

granite_llm.py              #IBM Watsonx communication

document_embedder.py        #Document embedding logic

kpi_file_forecaster.py      #Forecasting models

anomaly_file_checker.py     #Anomaly detection modules

report_generator.py         #AI-generated report creation
```

## 6. Running the Application Start FastAPI server for backend APIs.

Run Streamlit dashboard for user interface.

Use sidebar navigation to access features.

Upload documents or CSVs, chat with AI, view reports.

All services update in real-time asynchronously.

## 7. API Documentation POST /chat/ask — AI chat query endpoint

POST /upload-doc — Document upload & embedding

GET /search-docs — Semantic search for policies

GET /get-eco-tips — Sustainability tips by category

POST /submit-feedback — Store citizen feedback

Swagger UI is available for API testing.

## 8. Authentication Open demo environment for now.

Planned enhancements: JWT, OAuth2, role-based access, sessions.

## 9. User Interface Sidebar navigation

KPI visualization cards

Tabs for chat, eco tips, forecasting

Real-time forms & updates

PDF report downloads

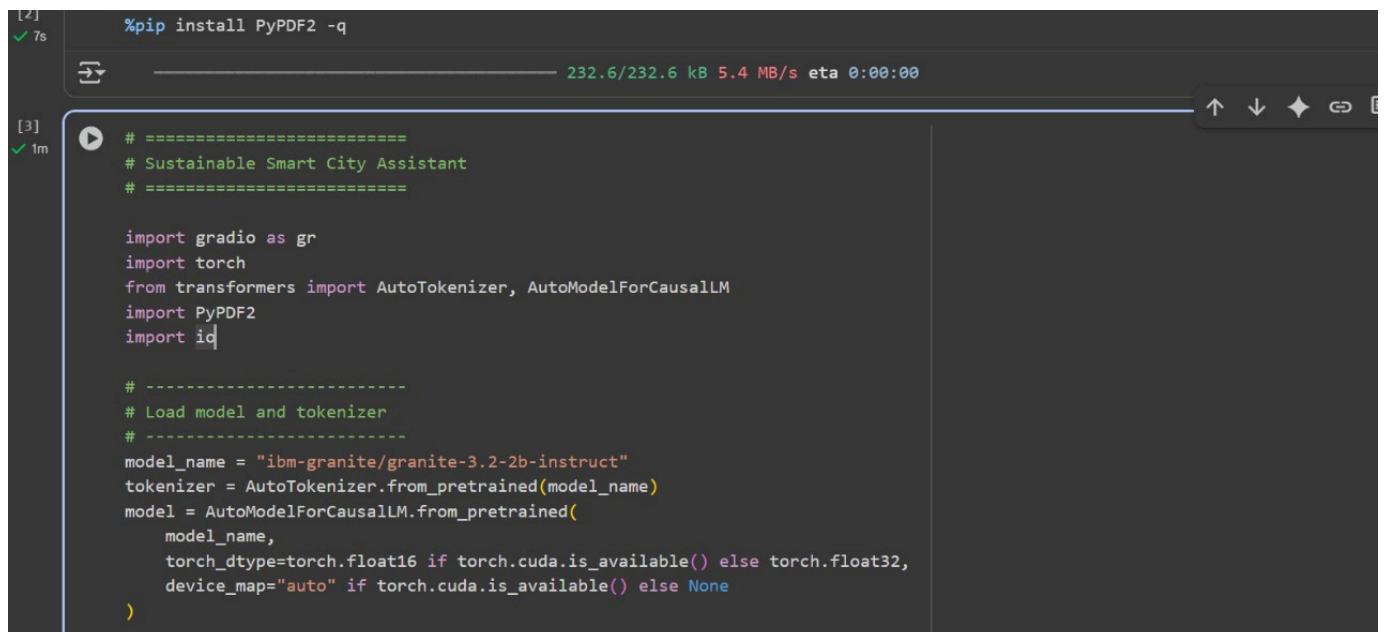Accessible design prioritizing clarity and ease of use

## 10. Testing Unit tests for core functions

API tests through Swagger/UI and Postman

Manual tests for file and chat interfaces

Edge case testing for invalid inputs

## 11.Screenshots

```
%pip install PyPDF2 -q

                                       232.6/232.6 kB 5.4 MB/s eta 0:00:00

# ==========================
# Sustainable Smart City Assistant
# ==========================

import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import PyPDF2
import io

# -------------------------
# 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
)
```

```python
    f tokenizer.pad_token is None:
        tokenizer.pad_token = tokenizer.eos_token

    # ---------------------------
    # Helper function to generate model responses
    # ---------------------------
    ef 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
```

```python
    # PDF text extraction
    # ---------------------------
    ef extract_text_from_pdf(pdf_file):
        if pdf_file is None:
            return ""
        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 Generator
    # ---------------------------
    ef eco_tips_generator(problem_keywords):
        prompt = f"Generate practical and actionable eco-friendly tips for sustainable living related to: {problem_keywords}. Provide specific solutic
        return generate_response(prompt, max_length=1000)

    # ---------------------------
    # Policy Summarization
    # ---------------------------
    ef policy_summarization(pdf_file, policy_text):
        if pdf_file is not None:
```

```python
            content = extract_text_from_pdf(pdf_file)
            summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{cc
        else:
            summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{pc

        return generate_response(summary_prompt, max_length=1200)

    # ---------------------------
    # Gradio Interface
    # ---------------------------
    ith gr.Blocks() as app:
        gr.Markdown("# 🌱 Sustainable Smart City Assistance\nEco Assistant & Policy Analyzer")

        with gr.Tab("Eco Living Tips"):
            with gr.Row():
                with gr.Column():
                    keywords_input = gr.Textbox(
                        label="Environmental Problem/Keywords",
                        placeholder="E.g., plastic, solar, water waste, energy saving...",
                        lines=3
                    )
                    generate_tips_btn = gr.Button("Generate Eco Tips")

                with gr.Column():
                    tips_output = gr.Textbox(label="Sustainable Living Tips", lines=15)
```

```python
        generate_tips_btn.click(eco_tips_generator, inputs=keywords_input, outputs=tips_output)

    with gr.Tab("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)

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

```
87.0/87.0 [00:00<00:00, 4.37kB/s]
```

les  ▶ Terminal                                          ✦                        ✓ 9:22 PM   🖳 T4 (Python 3

---

```
vocab.json:        777k/? [00:00<00:00, 7.66MB/s]
merges.txt:        442k/? [00:00<00:00, 11.2MB/s]
tokenizer.json:        3.48M/? [00:00<00:00, 40.1MB/s]
added_tokens.json: 100%        87.0/87.0 [00:00<00:00, 3.99kB/s]
special_tokens_map.json: 100%        701/701 [00:00<00:00, 22.6kB/s]
config.json: 100%        786/786 [00:00<00:00, 52.3kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json:        29.8k/? [00:00<00:00, 2.81MB/s]
Fetching 2 files: 100%        2/2 [01:21<00:00, 81.09s/it]
model-00001-of-00002.safetensors: 100%        5.00G/5.00G [01:20<00:00, 94.6MB/s]
model-00002-of-00002.safetensors: 100%        67.1M/67.1M [00:13<00:00, 4.74MB/s]
Loading checkpoint shards: 100%        2/2 [00:20<00:00,  8.67s/it]
generation_config.json: 100%        137/137 [00:00<00:00, 9.09kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
```

---

## 🌱 Sustainable Smart City Assistance

Eco Assistant & Policy Analyzer

**Eco Living Tips**    Policy Summarization

**Environmental Problem/Keywords**

> plastic

**Generate Eco Tips**

**Sustainable Living Tips**

1. **Reduce Plastic Consumption:**
   - **Reusable Shopping Bags:** Carry reusable cloth or jute bags for grocery shopping instead of using single-use plastic bags. Replace disposable plastic bags with reusable ones to minimize waste.
   - **Refillable Bottles:** Invest in a high-quality, BPA-free water bottle and refill it from the tap or a filtered source. This reduces the need for single-use plastic bottles.
   - **Reusable Containers:** Use glass, stainless steel, or silicone containers for storing food or leftovers in the kitchen. This eliminates the requirement for disposable plastic containers.
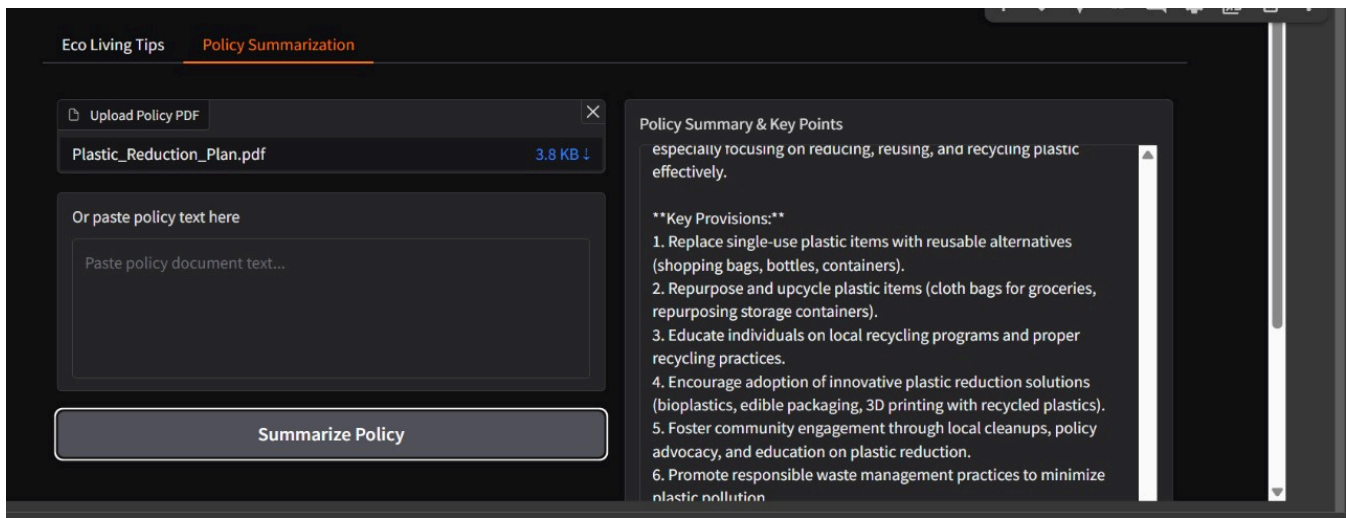
## 12. Known Issues Document Upload Limitations: Large or complex documents (especially very large

PDFs or CSV files with high dimensional data)
may experience slow processing times or timeouts during embedding and vectorization, affecting user experience and responsiveness.

AI Model Accuracy Variability: Occasional inaccuracies in policy summarization and eco-tip generation can occur due to nuances

in natural language understanding, ambiguities in source documents, or incomplete context. More extensive testing and prompt tuning may be needed for sensitive applications.

Real-Time Data Integration Constraints: Integration with live sensor data for forecasting and

anomaly detection may have delays or
incomplete updates due to network latency or temporary API failures, resulting in less timely insights.

User Interface Scalability: While the UI is designed to be modular, performance and

responsiveness may degrade with very
large datasets or numerous simultaneous users, especially in resource-constrained environments.

Authentication and Security: As the current version operates in an open demo mode, there is

no implemented user
authentication or role-based access control, posing potential security risks for deployment in production without additional safeguards.

Edge Case Handling: Though many input edge cases are handled, malformed or corrupted files and

unusual data formats
can still cause system errors or crashes if not preprocessed adequately before upload.

Limited Language Support: The conversational interface currently supports only

English, limiting accessibility for
non-English-speaking users.

## 13. Future Enhancements

Enhanced Authentication and Authorization: Implement robust security features including JWT-based token authentication, OAuth2 integration
with IBM Cloud, and detailed role-based access controls to ensure data protection and user privacy.

Multilingual Support:

Expand natural language understanding and generation capabilities to support multiple languages, increasing accessibility and usability across diverse user groups.

Improved AI Model Training:

Continuously train and fine-tune AI models with broader datasets and domain-specific knowledge to improve accuracy in policy summarization, eco-tip generation, and forecast reliability.

Real-Time Data Stream Integration:

Integrate with real-time IoT sensor streams and city data platforms to enhance the accuracy and timeliness of resource forecasting and anomaly detection modules.

User Session and History Tracking:

Add functionality to track individual user interactions, save chat histories, and personalize sustainability advice based on historical behavior and preferences.

Scalable Cloud Deployment:

Optimize backend and frontend components for scalable deployment on cloud platforms, enabling concurrent multi-user access with load balancing and high availability.

Advanced Visualization and Reporting:

Integrate richer data visualization tools, interactive charts, and customizable report generation to facilitate better decision-making for city officials and stakeholders.

Mobile Application Development:

Develop native or hybrid mobile applications to increase accessibility for citizens and officials on the go, enhancing engagement and real-time interaction.

Community Engagement Features:

Expand citizen feedback mechanisms with gamification, social sharing, and collaborative planning tools to foster greater community participation and awareness.

Accessibility Enhancements:

Improve UI design for compliance with accessibility standards (e.g., WCAG), including support for screen readers, keyboard navigation, and adjustable visual settings.