

# Sustainable Smart City Assistant Using IBM Granite LLM

## Project Documentation

### 1. Introduction

Project Title : **Sustainable Smart City Assistant Using IBM Granite LLM**  
Team member : **SANJITHA C**  
Team member : **SANJANA S**  
Team member : **SANGEETHAPRIYA G**  
Team member : **SAMUNDEESWARI M**

### 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:

##### Conversational Interface

Natural language interaction

Citizens and officials can ask questions, get updates, and receive guidance in plain language

##### Policy Summarization

Simplifies government documents into concise, actionable summaries

##### Resource Forecasting

Uses predictive analytics to estimate future energy, water, and waste usage

### **Eco-Tip Generator**

Provides daily sustainability advice based on user behavior

### **Citizen Feedback Loop**

Collects and analyzes public input to inform planning and improvements

### **KPI Forecasting**

Projects performance indicators for long-term city planning

### **Anomaly Detection**

Identifies unusual patterns in data to detect issues early

### **Multimodal Input Support**

Accepts text, PDFs, and CSVs for document analysis and forecasting

### **Streamlit or Gradio UI**

User-friendly dashboards for both citizens and officials

## **3. Architecture**

### **Frontend (Streamlit)**

Interactive web UI with dashboards, file uploads, chat interface, feedback forms, and report viewers.

Sidebar navigation using streamlit-option-menu.

Modularized pages for scalability.

## **Backend (FastAPI)**

REST framework powering document processing, chat, eco tips, report creation, and embeddings.

Optimized for performance and Swagger integration.

## **LLM Integration (IBM Watson Granite)**

Granite LLM used for natural language understanding and generation.

Prompts designed for summarization, eco tips, and reporting.

## **Vector Search (Pinecone)**

Policy documents embedded with Sentence Transformers.

Supports semantic search using cosine similarity.

## **ML Modules (Forecasting & Anomaly Detection)**

Lightweight ML models using scikit-learn.

Time-series modeling and visualization with pandas & matplotlib.

## **4. Setup Instructions**

### **Prerequisites:**

Python 3.9+

pip and virtual environment tools

API keys for IBM Watson and Pinecone

Internet access

### **Installation Process:**

1. Clone repository
2. Install dependencies from requirements.txt
3. Create .env file and configure credentials
4. Run backend server with FastAPI
5. Launch frontend with Streamlit
6. Upload data and interact with modules

## **5. Folder Structure**

app/ – FastAPI backend logic

app/api/ – Modular API routes

ui/ – Streamlit UI pages

smart\_dashboard.py – Main Streamlit dashboard

granite\_llm.py – Handles Watson Granite LLM

document\_embedder.py – Embedding + Pinecone storage

kpi\_file\_forecaster.py – Forecasting trends

anomaly\_file\_checker.py – Detects unusual data

report\_generator.py – AI-generated sustainability reports

## **6. Running the Application**

Launch FastAPI server

Run Streamlit dashboard

Navigate through sidebar pages

Upload documents or CSVs

Interact with chat assistant

View outputs: reports, summaries, predictions

All interactions are real-time with backend APIs.

## **7. API Documentation**

POST /chat/ask – Get AI-generated response

POST /upload-doc – Upload & embed documents

GET /search-docs – Semantic search of policies

GET /get-eco-tips – Sustainability tips

POST /submit-feedback – Store citizen feedback

Swagger UI used for API testing.

## **8. Authentication**

Demo runs open access

Future deployments may add:

Token-based authentication (JWT/API keys)

OAuth2 with IBM Cloud

Role-based access (admin/citizen/researcher)

User sessions & history tracking

## 9. User Interface

Minimalist, accessible design

Sidebar navigation

KPI visualizations & summary cards

Tabs for chat, eco tips, forecasting

Real-time form handling

PDF report download

## 10. Testing

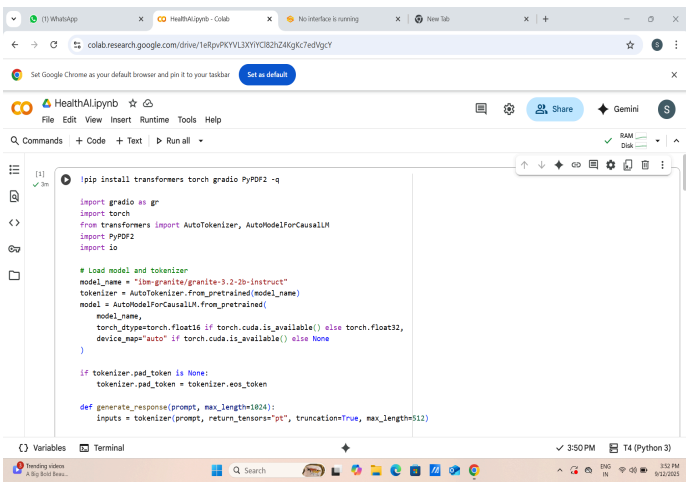
Unit Testing: For prompt engineering

API Testing: Swagger UI, Postman

Manual Testing: File uploads, chat, outputs

Edge Cases: Invalid inputs, large files, API errors

## 11. Program and Output Screenshots



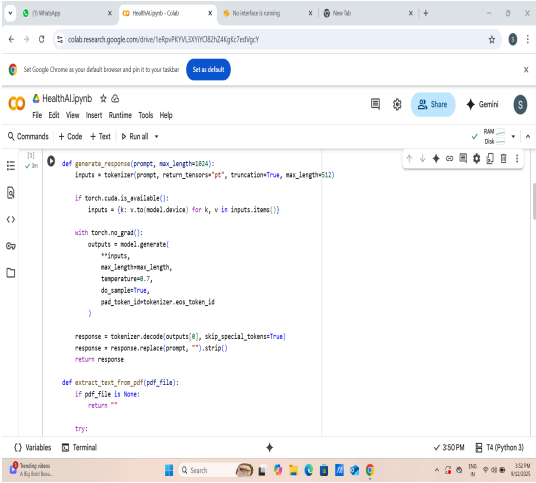
```
[1] ✓ 3s
!pip install transformers torch gradio PyPDF2 -q

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

# Load model and tokenizer
model_name = "llo-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

def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)
```



```
def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

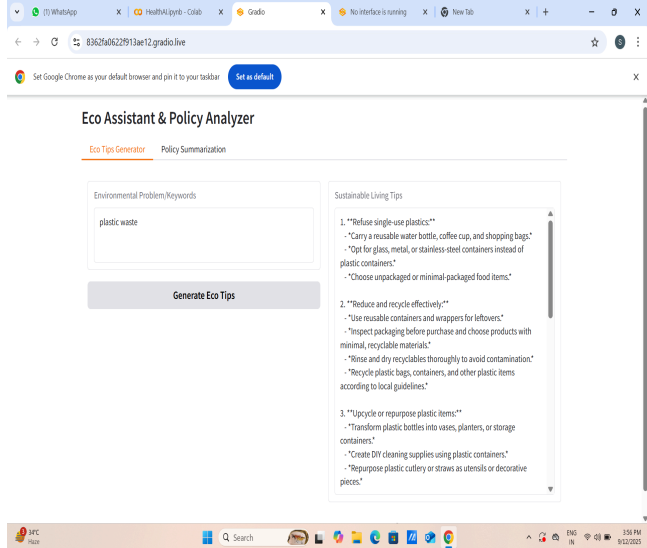
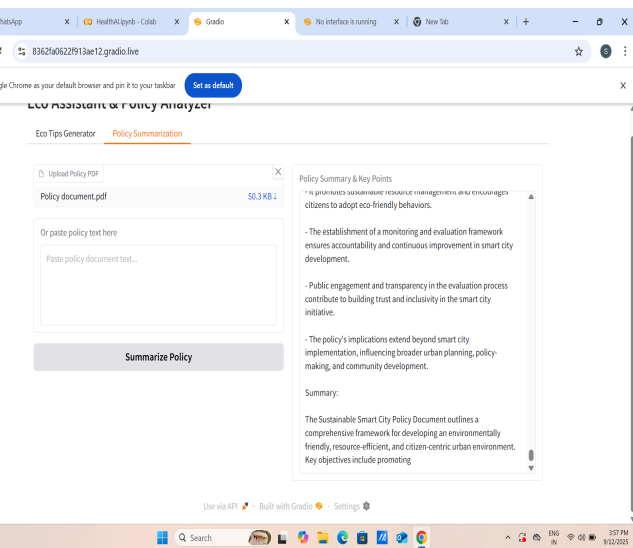
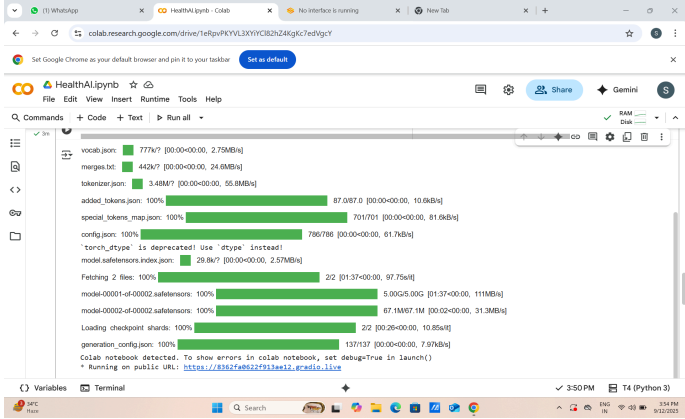
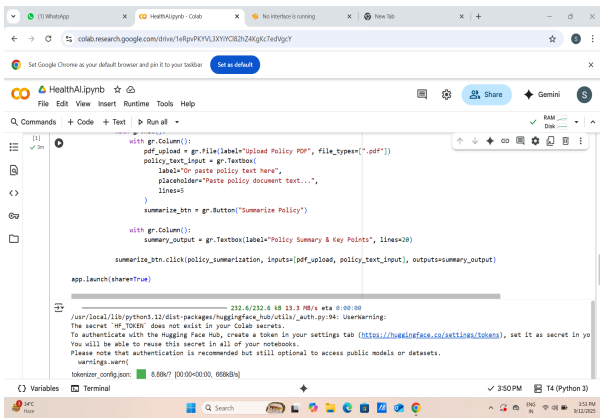
    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.5,
            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

def extract_text_from_pdf(pdf_file):
    if pdf_file is None:
        return ""

    try:
```



## 12. Future Enhancements

Multi-language support

Mobile app integration

Advanced AI forecasting

Integration with IoT sensor networks