

Sustainable Smart City Assistant Using IBM Granite LLM

Project Documentation

1.Introduction

- Project title : **Sustainable Smart City Assistant Using IBM Granite LLM**
- Team leader : CHELLAPRIYA S
- Team member : MOSHINA K
- Team member : HARISHA K K
- Team member : BHUVANA V

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

Key Point: Natural language interaction

Functionality: Allows citizens and officials to ask questions, get updates, and receive guidance in plain language

Policy Summarization

Key Point: Simplified policy understanding

Functionality: Converts lengthy government documents into concise, actionable summaries.

Resource Forecasting

Key Point: Predictive analytics

Functionality: Estimates future energy, water, and waste usage using historical and real-time data.

Eco-Tip Generator

Key Point: Personalized sustainability advice

Functionality: Recommends daily actions to reduce environmental impact based on user behavior.

Citizen Feedback Loop

Key Point: Community engagement

Functionality: Collects and analyzes public input to inform city planning and service improvements.

KPI Forecasting

Key Point: Strategic planning support

Functionality: Projects key performance indicators to help officials track progress and plan ahead.

Anomaly Detection

Key Point: Early warning system

Functionality: Identifies unusual patterns in sensor or usage data to flag potential issues.

Multimodal Input Support

Key Point: Flexible data handling

Functionality: Accepts text, PDFs, and CSVs for document analysis and forecasting.

Streamlit or Gradio UI

Key Point: User-friendly interface

Functionality: Provides an intuitive dashboard for both citizens and city officials to interact with the assistant.

3. Architecture

Frontend (Stream lit):

The frontend is built with Streamlit, 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 streamlit-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. Folder Structure

app/ – Contains all Fast API backend logic including routers, models, and integration modules.

app/api/ – Subdirectory for modular API routes like chat, feedback, report, and document vectorization.

ui/ – Contains frontend components for Stream lit pages, card layouts, and form UIs.

smart_dashboard.py – Entry script for launching the main Stream lit dashboard.

granite_llm.py – Handles all communication with IBM Watsonx Granite model including summarization and chat.

document_embedder.py – Converts documents to embeddings and stores in Pinecone.

kpi_file_forecaster.py – Forecasts future energy/water trends using regression.

anomaly_file_checker.py – Flags unusual values in uploaded KPI data.

report_generator.py – Constructs AI-generated sustainability reports.

6. 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.

7. 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. **8. 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

9. User Interface

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

10. Testing

Testing was done in multiple phases:

Unit Testing: For prompt engineering functions and utility scripts

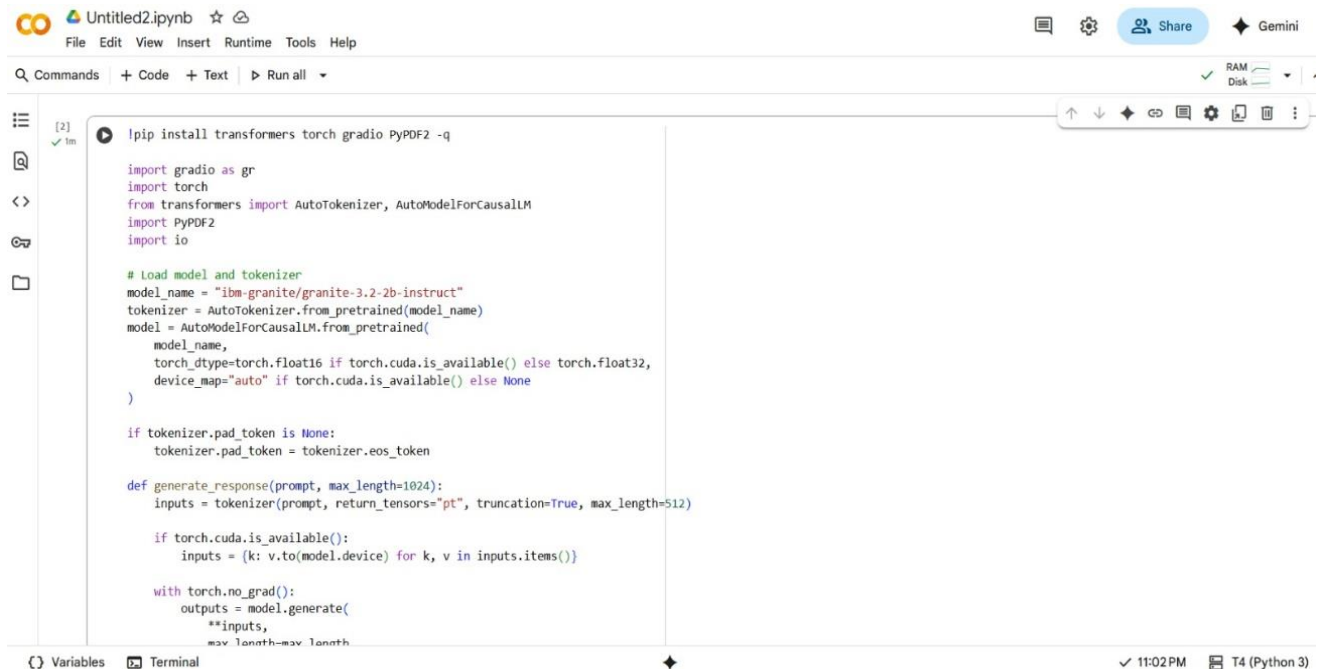
API Testing: Via Swagger UI, Postman, and test scripts

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.

11) Screenshots : Program code



The screenshot shows a Jupyter Notebook titled 'Untitled2.ipynb'. The code in the first cell is as follows:

```
[2] ✓ 1m
!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 = "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

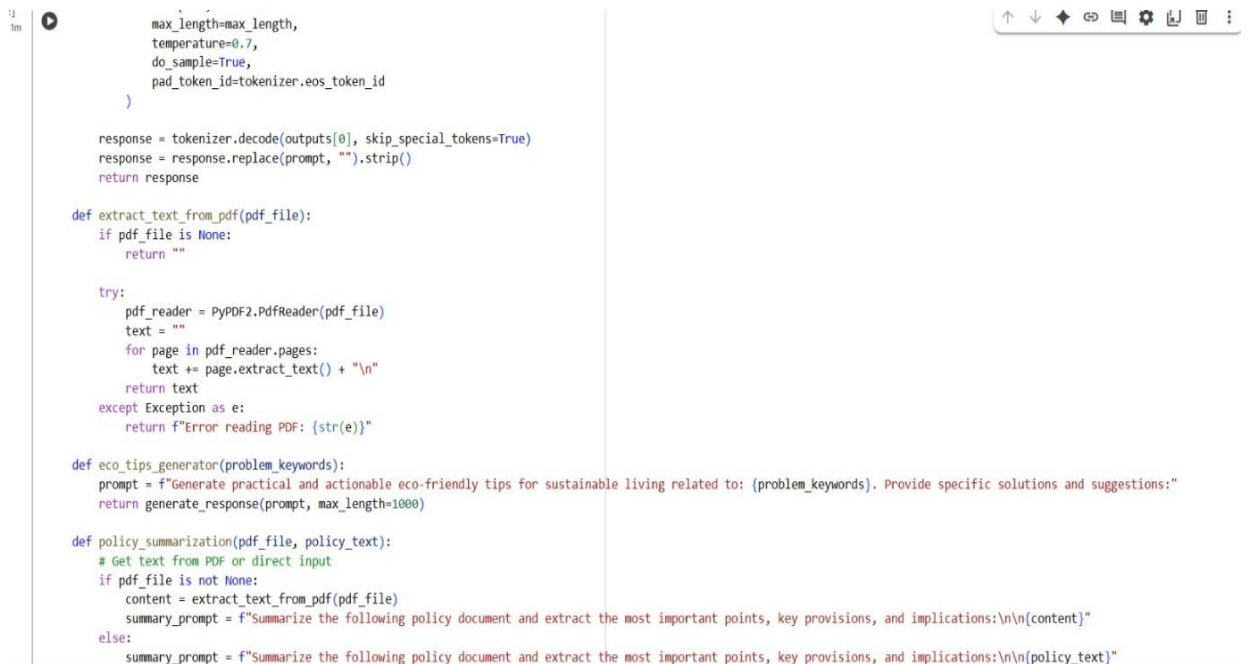
def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    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,
```

The interface includes a top bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help' menus. A left sidebar shows icons for file explorer, search, and other functions. The bottom status bar indicates '11:02 PM' and 'T4 (Python 3)'.

!



The screenshot shows the continuation of the Jupyter Notebook code. The code in the second cell is as follows:

```
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

def 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)}"

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

def policy_summarization(pdf_file, policy_text):
    # Get text from PDF or direct input
    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 implications:\n\n{content}"
    else:
        summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{policy_text}"
```

The interface shows the same top and bottom bars as the first screenshot. The left sidebar is also visible.

```

return generate_response(summary_prompt, max_length=1200)

# Create Gradio interface
with gr.Blocks() as app:
    gr.Markdown("# Eco Assistant & Policy Analyzer")

    with gr.Tabs():
        with gr.Tabitem("Eco Tips Generator"):
            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)

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

        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():

```

```

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

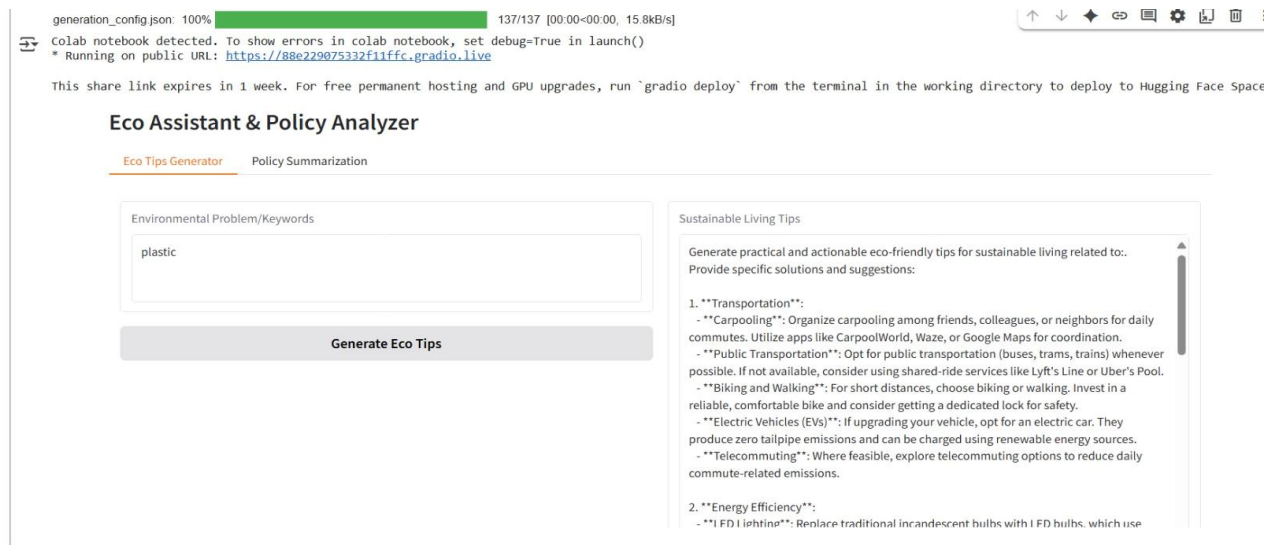
app.launch(share=True)

```

/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
 The secret 'HF_TOKEN' does not exist in your Colab secrets.
 To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>), set it as secret in your Google Colab and restart your session.
 You will be able to reuse this secret in all of your notebooks.
 Please note that authentication is recommended but still optional to access public models or datasets.
 warnings.warn(

tokenizer_config.json: 8.88k/? [00:00<00:00, 141kB/s]
 vocab.json: 777k/? [00:00<00:00, 8.02MB/s]
 merges.txt: 442k/? [00:00<00:00, 5.54MB/s]
 tokenizer.json: 3.48M/? [00:00<00:00, 11.2MB/s]
 added_tokens.json: 100% [87.0/87.0 [00:00<00:00, 1.81kB/s]
 special_tokens_map.json: 100% [701/701 [00:00<00:00, 20.5kB/s]
 config.json: 100% [786/786 [00:00<00:00, 18.4kB/s]
 "torch_dtype" is deprecated! Use "dtype" instead!
 model.safetensors.index.json: 29.8k/? [00:00<00:00, 2.35MB/s]
 Fetching 2 files: 100% [2/2 [01:15<00:00, 75.87s/it]
 model-00002-of-00002.safetensors: 100% [67.1M/67.1M [00:01<00:00, 70.2MB/s]
 model-00001-of-00002.safetensors: 100% [5.00G/5.00G [01:15<00:00, 49.0MB/s]
 Loading checkpoint shards: 100% [2/2 [00:20<00:00, 8.33s/it]
 generation_config.json: 100% [137/137 [00:00<00:00, 15.8kB/s]
 colab notebook detected. To show errors in colab notebook, set debug=True in launch()
 * Running on public URL: <https://88e229075332f11ffc.gradio.live>

OUTPUT:



12) Known Issues:

- The system currently depends call stable Internet connectivity which may affect performance in offline scenarios
- Some forecasting results may vary in accuracy due to limited historical data
- Integration with external city databases is still partial and may require manual updates

13)Future Enhancement:

- Add multilingual support to make dark assistant accessible to a wider range of citizens
- Integrate lo T devices and real time sensor networks for more accurate data collections
- Enhance the AI- models to provide more personalised eco friendly recommendations
- Implement mobile app support for better citizen in engagement on-the-go.