Sustainable smart City assistant using IBM granite LLM

1. Introduction

Project Title: Sustainable Smart City Assistant using IBM Granite LLM

Team Members:

Team Leader: Doppalapudi Navya

Team Member: Buddana Sri Rama Likhith

Team Member: Chimata Varshini

Team Member: Chinamuthevi Aparna

2. Project Overview

Purpose:

The goal of this project is to build a smart city digital assistant powered by IBM's Granite LLM that provides intelligent, sustainable, and data-driven solutions for urban management. It assists residents and city officials with sustainability insights, service queries, and actionable suggestions.

Features:

Natural language chat interface via IBM Granite LLM

Smart suggestions for energy/water conservation

Pollution and weather insights from APIs

Real-time question answering based on urban datasets

Simple web interface prototype (if applicable)

3. Architecture

Since this project was built in Google Colab, it simulates a MERN-like architecture:

Frontend:
A minimal interface is built using HTML/CSS within Colab or integrated using IPython.display tools (or optionally, Streamlit if extended externally). Input/output is handled using forms and embedded widgets.

Backend:
Python-based backend code runs within the Colab environment. It uses Flask (optional) or native Python functions to handle inputs, process data, and generate responses.

LLM Integration:
IBM Granite LLM is accessed via API from Colab using RESTful requests. All prompts and responses are managed within Python.

Database (Simulation):
Instead of MongoDB, data persistence is handled using Python dictionaries or saved as .json/.csv files in the session or mounted Google Drive.

4. Setup Instructions

Prerequisites:

-Google Account

-Access to Google Colab

-IBM Granite API credentials

-Python libraries: requests, json, IPython, pandas, etc.

Installation / Setup Steps:

1. Open the shared Colab link.

2. Mount Google Drive to load or save data (optional).

3. Enter IBM Granite LLM API key and endpoint in the specified cell.

4. Run the notebook cell by cell from top to bottom.

5. Folder Structure (Adapted for Colab)

Since this is a Colab notebook, there are no separate folders, but logical sections are organized as:

1. Setup & Imports

2. IBM Granite API Configuration

3. Input Form (Chat Interface)

4. Response Logic

5. Visualization / Output Section

6. (Optional) External API Integration

7. Save/Load Logs (Drive or CSV)

6. Running the Application

Open the notebook in Google Colab

Execute all cells in order

Input a query in the chat cell or form

Get a response from the IBM Granite LLM

7. API Documentation

API Used: IBM Granite LLM

Method: POST

Endpoint: https://<ibm-endpoint>

Request Example:

```
response = requests.post(
   url,
   headers={"Authorization": "Bearer <API_KEY>"},
   json={"input": "What are sustainable practices for water conservation?"}
```

)

Response Example:

```
{
  "output": "You can conserve water by using low-flow fixtures, fixing leaks, and harvesting rainwater."
}
```

---

8. Authentication

API access is authenticated using IBM-provided API keys.

The key is securely entered in a Colab input field and used in headers.

No user authentication system is implemented unless extended externally.

---

9. User Interface

Interface: Chat-based text interface created using input cells or interact() widgets

Features:

Text box for user queries

Response display area

Optional buttons for follow-up suggestions

10. Testing

Manual Testing:

Multiple queries tested via chat box

Edge cases and out-of-scope questions tested

Logging Responses:

Logged responses into a CSV file or printed inline for review

11. Screenshots or Demo

Code:

```
# ✅ Install Required Libraries
!pip install transformers accelerate gradio --quiet

# ✅ Imports
import gradio as gr
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

# ✅ Load IBM Granite Model
model_id = "ibm-granite/granite-3.3-2b-instruct"

try:
    tokenizer = AutoTokenizer.from_pretrained(model_id)
    model = AutoModelForCausalLM.from_pretrained(
        model_id,
        torch_dtype=torch.bfloat16,
        device_map="auto"
    )
    model.eval()
except Exception as e:
    print(f"Error loading model: {e}")
    def placeholder_inference(prompt, mode):
        return "⚠ Model loading failed. Please switch to GPU/High-RAM
runtime."
    inference_function = placeholder_inference
    model = tokenizer = None

# ✅ Inference Function
if model and tokenizer:
    def granite_inference(prompt, mode):
        try:
            if mode == "Eco-Query Assistant":
                formatted_prompt = (
                    f"Instruct: As a sustainable smart city assistant,
answer this eco-sustainability question:\n"
                    f"{prompt}\nOutput:"
                )
            elif mode == "Smart Complaint Resolver":
```

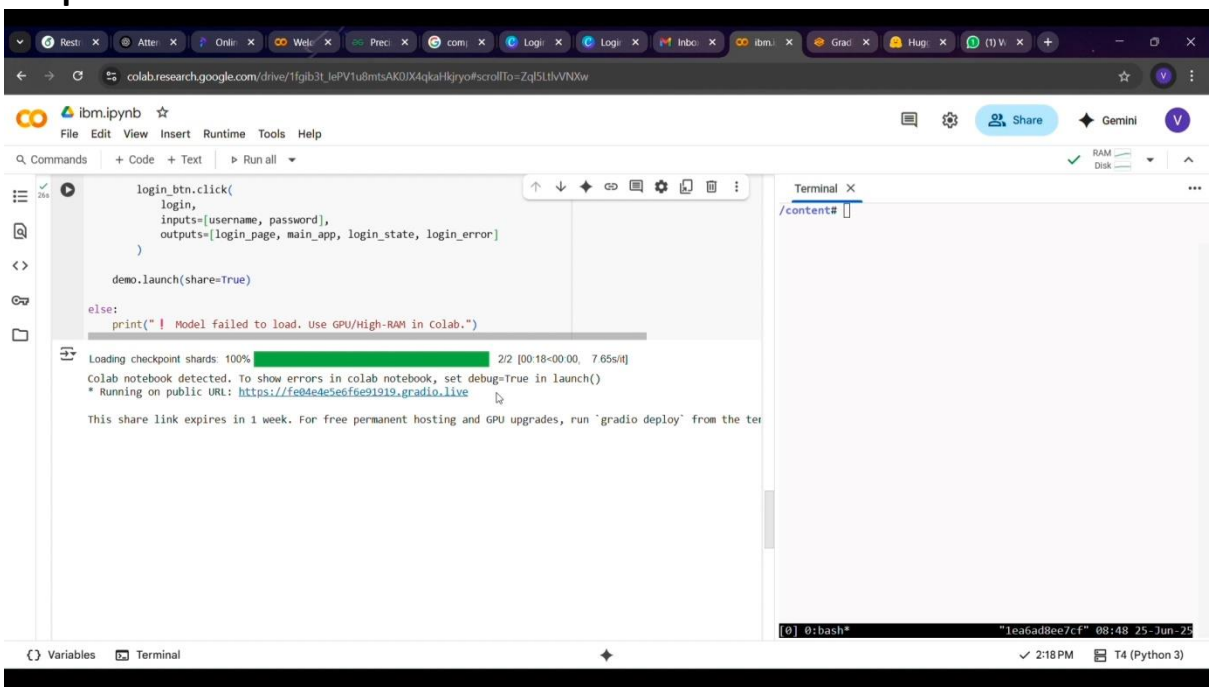
```
                formatted_prompt = (
                    f"Instruct: As a smart city complaint resolver,
analyze this civic issue. "
                    f"Classify it, suggest causes, and assign a
department.\n"
                    f"Issue: {prompt}\nOutput:"
                )
            else:
                formatted_prompt = f"Instruct: {prompt}\nOutput:"

            inputs = tokenizer(formatted_prompt,
return_tensors="pt").to(model.device)

            with torch.no_grad():
                outputs = model.generate(
                    inputs.input_ids,
                    max_new_tokens=800,
                    temperature=0.7,
                    top_p=0.9,
                    do_sample=True,
                    early_stopping=True
                )

            response =
tokenizer.decode(outputs[0][inputs.input_ids.shape[1]:],
skip_special_tokens=True)
            return response.strip()

        except Exception as e:
            return f"❌ Inference error: {e}"

    inference_function = granite_inference

# ✅ Gradio App with Login
if inference_function:
    with gr.Blocks(css="""
        .gradio-container {
            background: linear-gradient(-45deg, #ff6ec4, #7873f5,
#4ADEDE, #56FFA6);
            background-size: 400% 400%;
            animation: gradientFlow 15s ease infinite;
            padding: 20px;
            border-radius: 16px;
        }

        @keyframes gradientFlow {
            0% { background-position: 0% 50%; }
            50% { background-position: 100% 50%; }
```

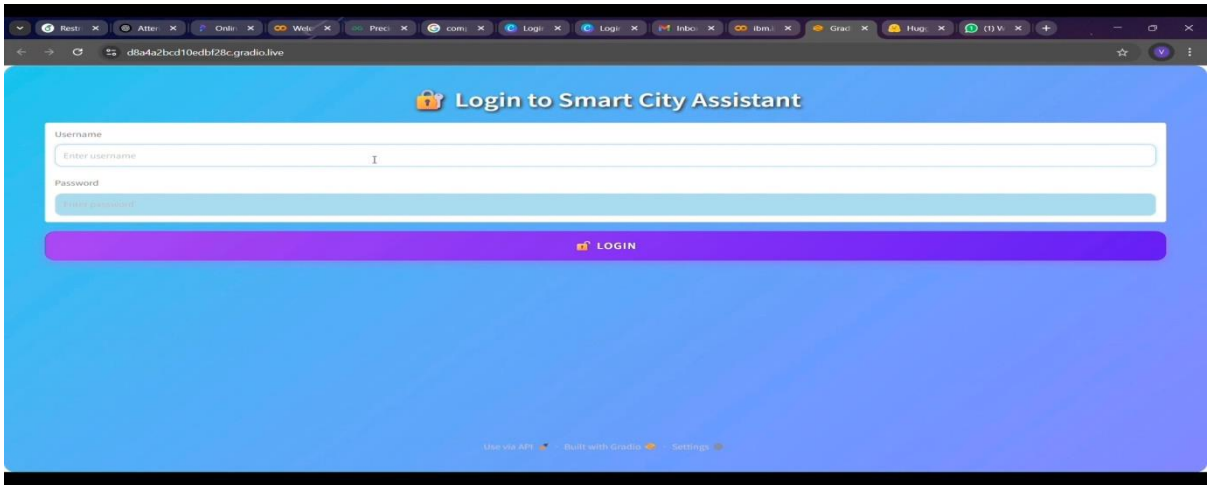
```
            100% { background-position: 0% 50%; }
        }

        .title {
            font-size: 36px;
            font-weight: bold;
            text-align: center;
            color: #ffffff;
            text-shadow: 2px 2px 5px #000;
        }

        .subtitle {
            font-size: 18px;
            text-align: center;
            color: #eeeeee;
            margin-bottom: 20px;
        }

        #fancy-button {
            background: linear-gradient(135deg, #8E2DE2, #4A00E0);
            color: white !important;
            font-weight: 700;
            font-size: 17px;
            padding: 14px 28px;
            border-radius: 14px !important;
            border: none;
            box-shadow: 0 5px 15px rgba(138, 43, 226, 0.4);
            transition: all 0.3s ease-in-out;
            text-transform: uppercase;
            letter-spacing: 1px;
        }

        #fancy-button:hover {
            background: linear-gradient(135deg, #4A00E0, #8E2DE2);
            transform: scale(1.05);
            box-shadow: 0 8px 20px rgba(72, 0, 255, 0.5);
        }

        textarea, input[type="text"], input[type="password"] {
            border-radius: 10px !important;
            background-color: #1e1e2f;
            color: white;
            border: 1px solid #444;
        }
    """) as demo:

        login_state = gr.State(False)
```

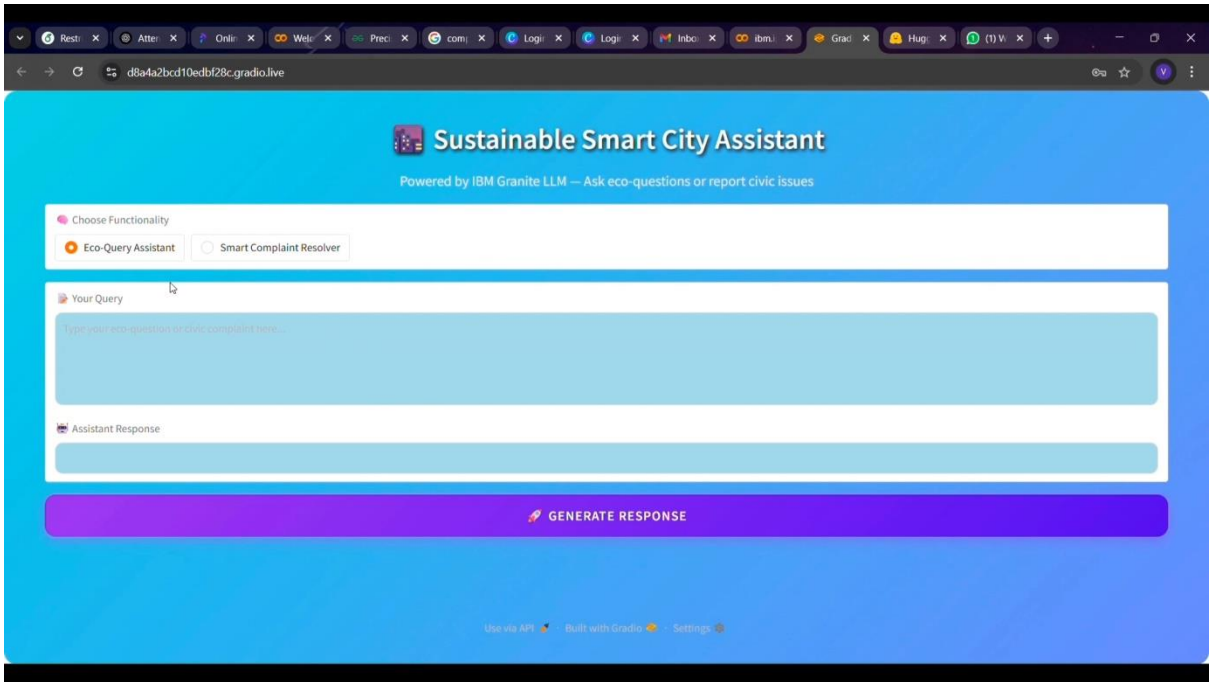
```
        def login(username, password):
            if username == "admin" and password == "1234":
                return gr.update(visible=False),
gr.update(visible=True), True, ""
            else:
                return None, None, False, "❌ Invalid credentials. Try
again."

        # Login Page
        with gr.Column(visible=True) as login_page:
            gr.Markdown("<div class='title'>🔐 Login to Smart City
Assistant</div>")
            username = gr.Textbox(label="Username", placeholder="Enter
username")
            password = gr.Textbox(label="Password", type="password",
placeholder="Enter password")
            login_btn = gr.Button("🔓 Login", elem_id="fancy-button")
            login_error = gr.Markdown("", visible=True)

        # Main App Page
        with gr.Column(visible=False) as main_app:
            gr.Markdown("<div class='title'>🌿 Sustainable Smart City
Assistant</div>")
            gr.Markdown("<div class='subtitle'>Powered by IBM Granite
LLM — Ask eco-questions or report civic issues</div>")

            with gr.Row():
                mode_choice = gr.Radio(
                    ["Eco-Query Assistant", "Smart Complaint
Resolver"],
                    label="🔄 Choose Functionality",
                    value="Eco-Query Assistant"
                )

            input_text = gr.Textbox(
                lines=5,
                placeholder="Type your eco-question or civic complaint
here...",
                label="📝 Your Query"
            )

            output_text = gr.Textbox(label="📤 Assistant Response",
interactive=False)

            submit_button = gr.Button("🚀 Generate Response",
elem_id="fancy-button")

            submit_button.click(
```

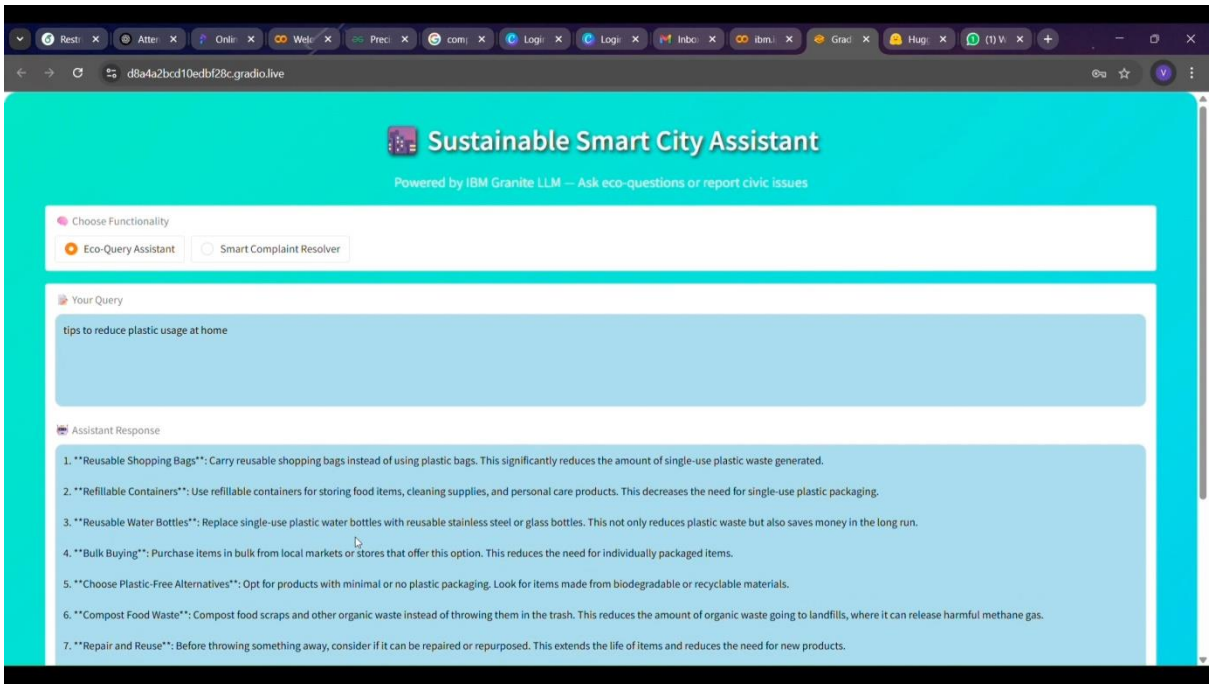
```
                fn=inference_function,
                inputs=[input_text, mode_choice],
                outputs=output_text
            )

        login_btn.click(
            login,
            inputs=[username, password],
            outputs=[login_page, main_app, login_state, login_error]
        )

    demo.launch(share=True)

else:
    print("! Model failed to load. Use GPU/High-RAM in Colab.")
```

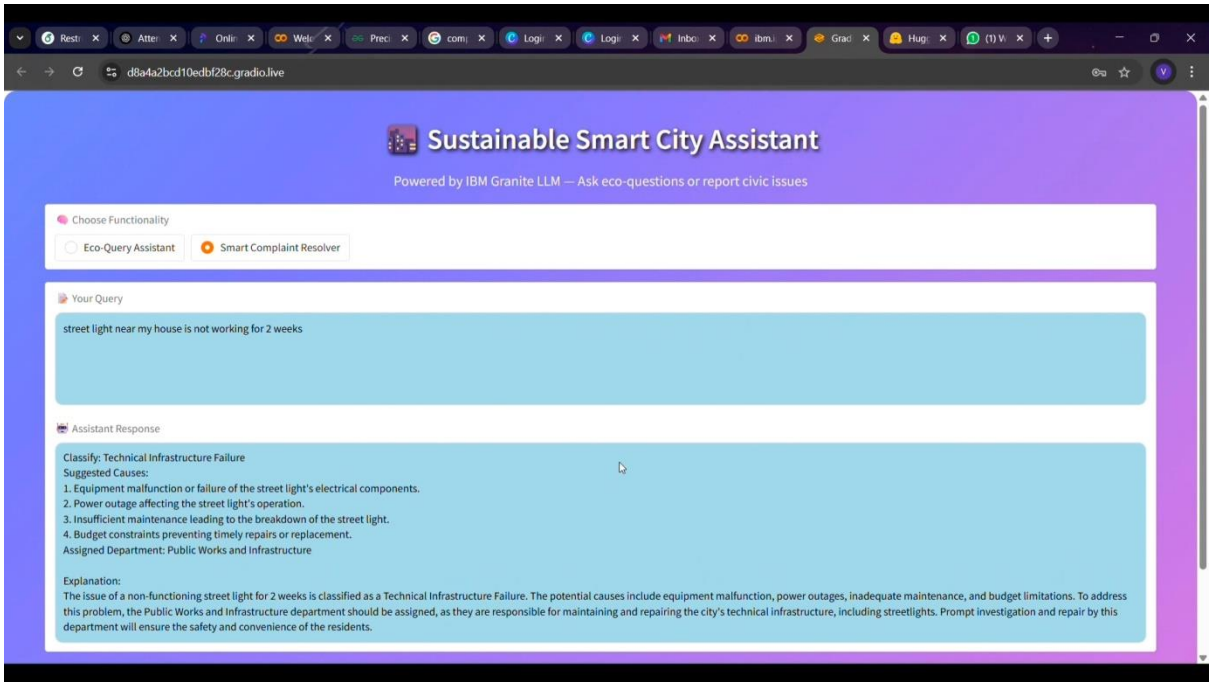
Output:






d8a4a2bcd10edbf28c.gradio.live
Sustainable Smart City Assistant
Powered by IBM Granite LLM — Ask eco-questions or report civic issues
Choose Functionality
Eco-Query Assistant
Smart Complaint Resolver
Your Query
Type your eco-question or civic complaint here...
Assistant Response
GENERATE RESPONSE
Use via API
Built with Gradio
Settings


d8a4a2bcd10edbf28c.gradio.live
Sustainable Smart City Assistant
Powered by IBM Granite LLM — Ask eco-questions or report civic issues
Choose Functionality
Eco-Query Assistant
Smart Complaint Resolver
Your Query
tips to reduce plastic usage at home
Assistant Response
1. **Reusable Shopping Bags**: Carry reusable shopping bags instead of using plastic bags. This significantly reduces the amount of single-use plastic waste generated.
2. **Refillable Containers**: Use refillable containers for storing food items, cleaning supplies, and personal care products. This decreases the need for single-use plastic packaging.
3. **Reusable Water Bottles**: Replace single-use plastic water bottles with reusable stainless steel or glass bottles. This not only reduces plastic waste but also saves money in the long run.
4. **Bulk Buying**: Purchase items in bulk from local markets or stores that offer this option. This reduces the need for individually packaged items.
5. **Choose Plastic-Free Alternatives**: Opt for products with minimal or no plastic packaging. Look for items made from biodegradable or recyclable materials.
6. **Compost Food Waste**: Compost food scraps and other organic waste instead of throwing them in the trash. This reduces the amount of organic waste going to landfills, where it can release harmful methane gas.
7. **Repair and Reuse**: Before throwing something away, consider if it can be repaired or repurposed. This extends the life of items and reduces the need for new products.



12. Known Issues

Limited persistence between sessions unless Google Drive is used

UI is minimal and not mobile responsive in Colab

API latency may vary based on query complexity

Requires re-authentication if notebook disconnects

13. Future Enhancements

Deploy a full web app (React frontend + Node backend) using the same logic

Store user interaction history in a MongoDB cluster

Integrate voice input/output for accessibility

Add real-time IoT data visualization for smart devices

Extend support for regional languages using IBM LLM fine-tuning