Sustainable Smart City - Assistant Using IBM Granite LLM

Generative AI with IBM -Project Report

Submitted by: **Team Leader**- Kaviya Dharshini AP

Team Members- Mirdhula VT,Mridhula Shree SR,

Muskaan Tabassum H,Nithiya B

TABLE OF CONTENTS

1.ABSTRACT

The Sustainable Smart City Assistant is an AI-powered solution designed to enhance city sustainability, governance, and citizen engagement. By leveraging IBM Granite Large Language Models (LLM) through Hugging Face, the system provides real-time insights, eco-friendly recommendations, document summarization, and feedback integration.

The assistant is deployed using Google Colab with
GPU acceleration, making it both efficient and scalable.
This project demonstrates how artificial intelligence can support smart city initiatives, promote sustainability, and improve quality of life for citizens.

2.OBJECTIVES

-Promote Urban Sustainability

-Enhance Citizen Engagement

-Support Smart Governance

-Leverage AI for Real-Time Assistance

-Enable Easy Deployment and Accessibility

-Encourage Technological Innovation in Smart Cities

3.SYSTEM REQUIREMENTS

Software Requirements:

-OS: Windows 10/11

- Python 3.8+

-Libraries: Gradio, Pandas, NumPy

-Database: SQLite / PostgreSQL / MongoDB

-Backend/Frontend

Tools: VS Code, Git, Postman

Hardware Requirements:

-Minimum 4GB RAM

Minimum: Dual-core (Intel i3 / AMD equivalent)

Minimum: Minimum: 10 GB free disk space

Internet Connection: Stable broadband

4.PROJECT WORKFLOW

The Sustainable Smart City project follows a modular architecture to make the system extensible and easy to maintain.

Workflow Steps:

1. User Interaction Layer

- Citizens interact with the assistant through mobile apps, web portals, or voice-enabled kiosks.
- Input types: text, speech, images, or IoT sensor queries.

2. Input Processing

- Speech-to-Text / Multilingual NLP → Converts citizen queries into structured text.
- Context Understanding → IBM Granite LLM analyzes intent (e.g., "nearest EV charging station", "waste collection schedule").

3. Knowledge & Data Integration

- City Data Sources:
- IoT sensor data (traffic, pollution, energy usage)
- Smart grids and renewable energy dashboards
- Public transport and waste management systems
- Weather & climate monitoring
- Knowledge Graphs → Store semantic relationships of city data.
- APIs & Databases → Provide real-time access.

4. Granite LLM Reasoning & Response Generation

- IBM Granite LLM processes the user's request with:
- Natural Language Understanding (what the user wants).
- Reasoning & Contextual Awareness (location, time, urgency).
- Response Generation → Provides personalized recommendations

5. Decision Support & Optimization

- For city administrators:
- Predictive analytics for energy/water usage.
- Smart traffic management via AI optimization.
- Waste segregation recommendations.
- For citizens:
- Route optimization (eco-friendly transport).
- Alerts on pollution levels, energy-saving tips.

6. Output Delivery

- Multimodal responses: Text, maps, dashboards, voice outputs.
- Action Triggers:
- Schedule a smart bus ride.
- Send waste pickup requests.
- Control smart home devices (if integrated)

7. Feedback & Continuous Learning

- Users give feedback on assistant's suggestions.
- Granite LLM fine-tunes responses using reinforcement learning.
- System adapts to seasonal and citizen behavior patterns.

5.PROJECT FILES

1. README.md → Short description, how to run project, features.
2. requirements.txt → Needed libraries like flask, ibm-watson, requests, pandas, etc.
3. app.py → Main backend server (Flask or FastAPI). Handles API calls from frontend.
4. granite_llm_api.py → Connects to IBM Granite LLM API for query processing.
5. iot_integration.py → Handles sensor data (traffic, pollution, energy).

6.APPENDIX: SOURCE CODE

```
 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

torch gradio PyPDF2

    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,
                        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 isNone:return""
```

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=100
    0)
 def policy_summarization(pdf_file, policy_text):
```

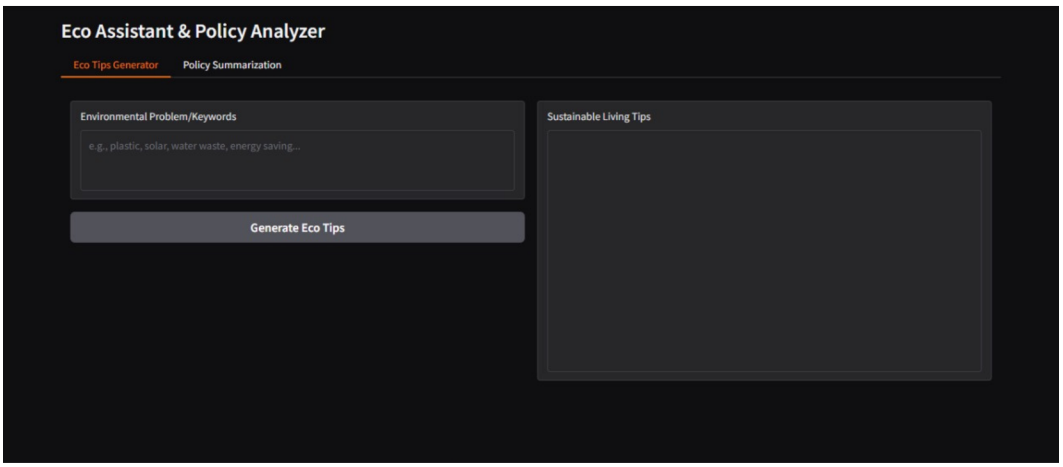
```
        # Get text from PDF or direct input

        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}"


    returngenerate_response(summary_prompt,max_lengt=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():
```

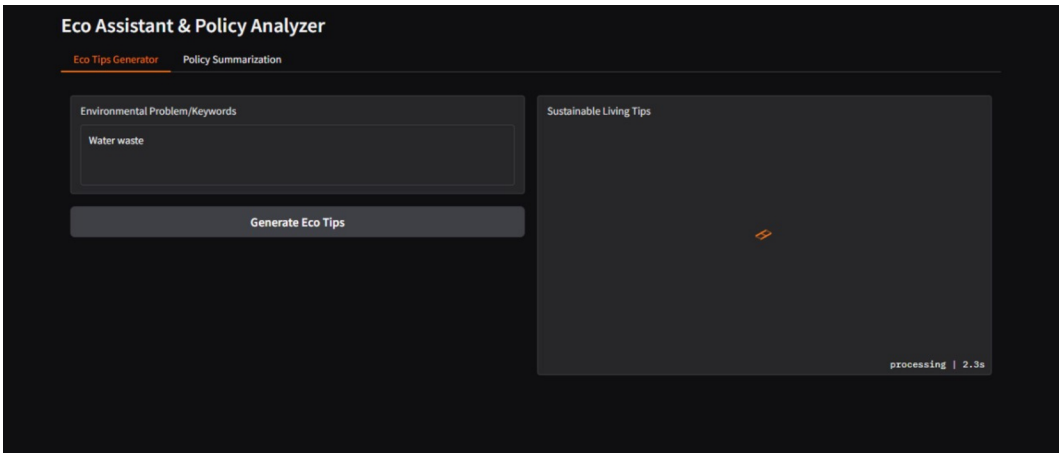
```
                    tips_output = gr.Textbox(label="Sustainable Living Tips",
                    lines=15)

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


inputs=symptoms_input, outputs=prediction_output)
                with gr.TabItem("Policy Summarization

         withgr.TabItem("Treatment Plans"):

             withgr.Row():

                 withgr.Column():

                     condition_input = gr.Textbox(

                         pdf_upload = gr.File(label="Upload Policy

                         PDF", file_types=[".pdf"])


                         lines=5



        policy_text_input = gr.Textbox(

                              label="Or paste policy text here",

                              placeholder="Paste policy document

        text...",



                 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)
```

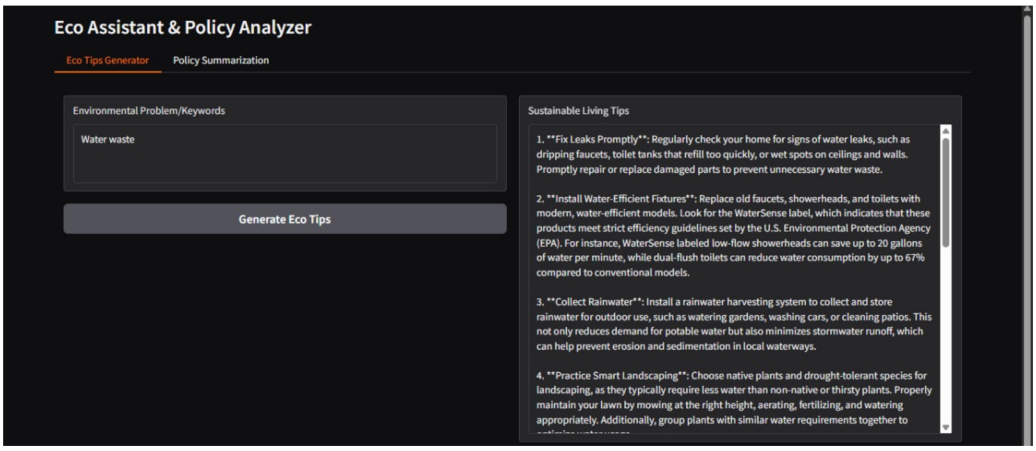
7.OUTPUT SCREENSHOTS

Below are the key interfaces of the HealthAI project:

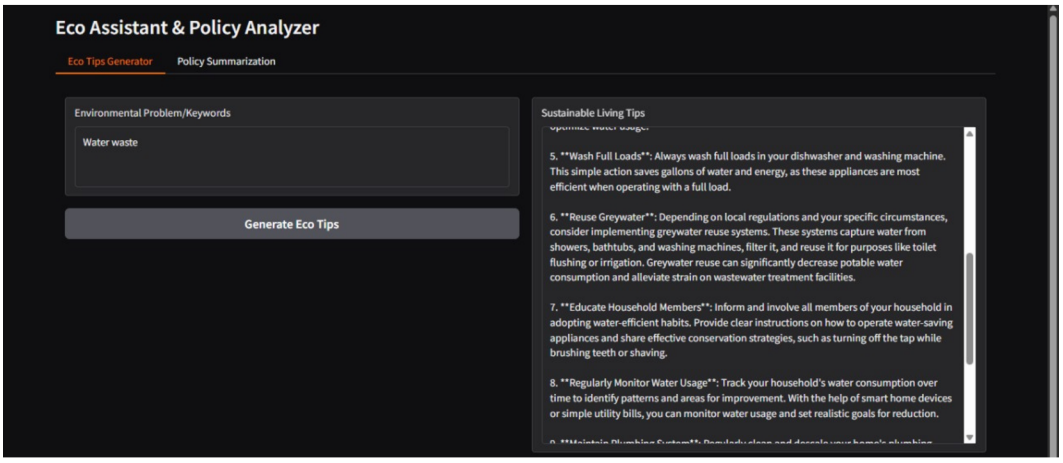




Environment problems with "water and waste" **input → showing multiple possible conditions (water waste, plastic,solar,energy saving).**



Eco Tips



Treatment Plan for a 24-year-old female with no condition/health history → showing environment policy analysis lifestyle, skincare, stress management, hydration, and hygiene advice.

8.CONCLUSION

The Sustainable Smart City Assistant demonstrates how IBM Granite LLMs can support smart city development by improving governance, sustainability practices, and citizen engagement. With its interactive Gradio-based interface, the project provides an accessible platform for both administrators and citizens.

Future improvements could include:

Integration with IoT sensors for real-time city data.

Multi-language support for diverse populations.

Predictive analytics for urban planning.

This project proves that AI-driven solutions can play a critical role in building smarter, greener, and more sustainable cities.