Sustainable Smart City Assistant Using IBM Granite LLM

Documentation

1. Introduction

Project Title: Sustainable Smart City Assistant Using IBM granite LLM

- Team Leader: KAMALESH M
- Team Member: PRATHIVRAJ R S
- Team Member: PRAVEEN RAJ R
- Team Member: SACHIN ANAND S

2. Project Overview

The Sustainable Smart City Assistant leverages **IBM's Granite Large Language Model (LLM)** to provide intelligent, real-time support for urban planning, resource management, citizen engagement, and sustainability initiatives in smart cities. This AI-driven assistant helps city administrators, residents, and businesses make data-informed decisions that align with sustainability goals and improve quality of life.

• Features:

- **Personalized Sustainability Advisor**
  Delivers tailored eco-friendly tips and actionable recommendations based on user behavior, location, and city sustainability goals.
-
- **Intelligent Policy Digest**
  Transforms lengthy regulatory and environmental documents into concise, easy-to-understand summaries highlighting critical points and community impact.
-
- **Seamless Document Processing**
  Supports direct uploading of PDFs and other document formats, automatically extracting and analyzing text for efficient policy review and insights.

- **User-Friendly Interactive Dashboard**
  Built with Gradio, the interface enables effortless conversations, document uploads, and visualization of sustainability data through a clean and accessible design.
- **Advanced AI Powered by IBM Granite via Hugging Face**
  Integrates IBM Granite LLM through Hugging Face Transformers for high-accuracy natural language understanding, supporting multilingual queries and contextual responses.

3. Architecture

Frontend (Gradio):

Provides a tabbed interface with input fields, buttons, and output textboxes for both eco tips and policy summaries.

Backend (PyTorch + Transformers):

Handles text processing, PDF parsing, and interaction with the IBM Granite model.

LLM Integration (IBM Granite):

Granite-3.2-2b-instruct model from Hugging Face provides language understanding and generation.

4. Setup Instructions

Prerequisites:

- Python 3.9 or later
- pip and virtual environment tools
- Internet access to download Hugging Face models

Installation Process:

- Install dependencies: pip install transformers torch gradio PyPDF2
- Run the script: python maja.txt
- The Gradio app will launch locally with a shareable link.

5. Folder Structure

maja.txt – Main application file containing model loading, functions, and Gradio interface.

functions/ – (Optional future extension) Separate utility functions.

uploads/ – (Optional) Store uploaded PDF files.

6. Running the Application

- Run python code
- Open the link in your browser provided by Gradio.
- Navigate to Eco Tips Generator tab for sustainability suggestions.
- Navigate to Policy Summarization tab to analyze PDFs or pasted text.

7. User Flows

Eco Tips Generator: Input keywords → AI generates actionable tips.

Policy Summarization: Upload PDF or paste text → AI produces key points.

8. Authentication

The current version runs in an open environment for demonstration. Future enhancements may include user authentication and access control.

9. User Interface

- Tabbed layout with two main sections (Eco Tips Generator, Policy Summarization).
- File upload support for policy analysis.
- Large output textboxes for readability.
- Shareable Gradio link for easy collaboration.

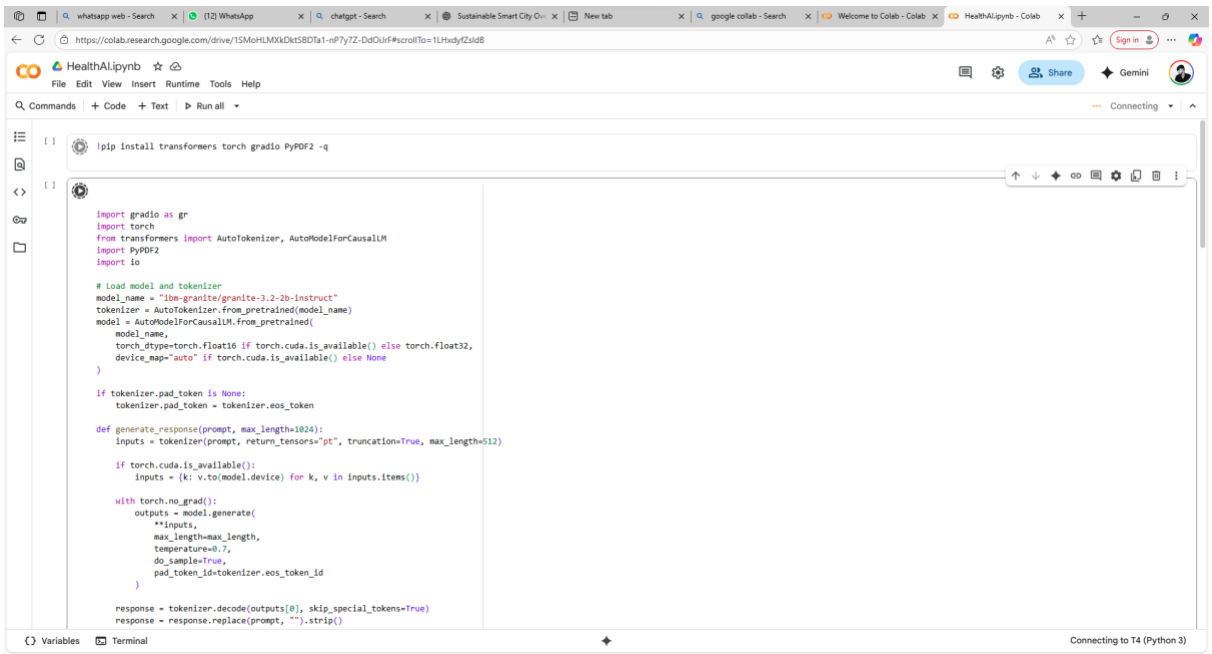
10. Testing

Unit Testing: Functions like eco_tips_generator and policy_summarization were tested individually.

Manual Testing: Gradio interface tested with sample inputs and PDFs.

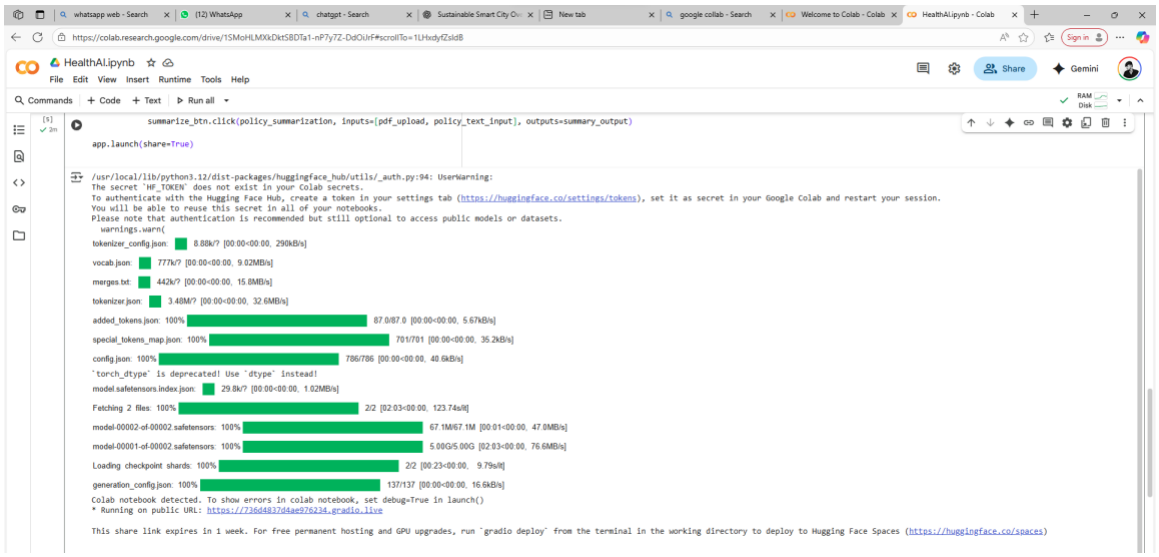
Edge Case Handling: Empty input, corrupted PDFs handled with error messages.

11. Screenshots



```
        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}"

        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...",
```

```
                        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():
                    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)
```



12. Known Issues

- Large PDFs may slow down summarization.
- Mobile app performance optimization pending.
- Requires stable internet connection for model usage.
- Limited offline functionality.

13. Future Enhancements

• Add multi-language support.

• Provide visual insights such as charts or graphs.

• Detailed policy recommendations beyond summarization.

• Extend API endpoints for integration with other applications.

14. GitHub Link:

Code link:

https://github.com/Prathivraj09/IBM-Project