Sustainable Smart City Assistant – Project Documentation

1. Introduction

Project Title: Sustainable Smart City Assistant Using IBM Granite LLM
Team ID: NM2025TMID02139
Team Leader: Firthose.R,
Team Members: Karthikeyan.A R, Sugan Prajan.P, Surendran.P

2. Project Overview ■

Purpose:
This project is a Sustainable Smart City Assistant web application built using Gradio and Hugging Face Transformers. It leverages the ibm-granite/granite-3.2-2b-instruct model to perform two primary functions: 1) Generating eco-friendly tips for sustainable living, and 2) Summarizing policy documents for urban planning and governance.

Users can upload a PDF document or enter environmental problems, and the assistant will provide specific solutions, policy insights, and sustainable practices. The application is designed to be accessible through an easy-to-use Gradio interface.

3. Architecture ■■

The application's architecture is a simple monolithic structure with clear separation of concerns, all running within a single Python script.

- **Frontend (UI):** Built using Gradio, provides a web-based, interactive dashboard with two tabs: "Eco Tips Generator" and "Policy Summarization".
- **Backend (Core Logic):** Powered by IBM Granite LLM (ibm-granite/granite-3.2-2b-instruct). Handles model loading, eco-tip generation, and policy summarization.
- **PDF Handling:** Implemented using PyPDF2 for text extraction from uploaded documents.
- **AI Interaction:** The generate_response function prepares inputs for the transformer model, executes inference, and decodes outputs into human-readable responses.

4. Setup Instructions ■■

1. Clone the repository (if applicable).
2. Create and activate a virtual environment.
3. Install dependencies:

```
pip install gradio torch transformers PyPDF2
```

4. Run the Python script to launch the application.

5. Folder Structure ■

```
/smart-city-assistant/
■ README.md
■ app.py # Main application script with all code
```

6. Running the Application ■■

1. Navigate to the project directory.
2. Run: python app.py
3. Access the interface at http://127.0.0.1:7860

7. API Documentation ■

The application uses Gradio as the interaction layer:
- **Eco Tips Generator:** Takes environmental keywords and returns actionable sustainable tips.
- **Policy Summarization:** Takes PDF/text input and generates a structured summary with key points.

8. User Interface ■

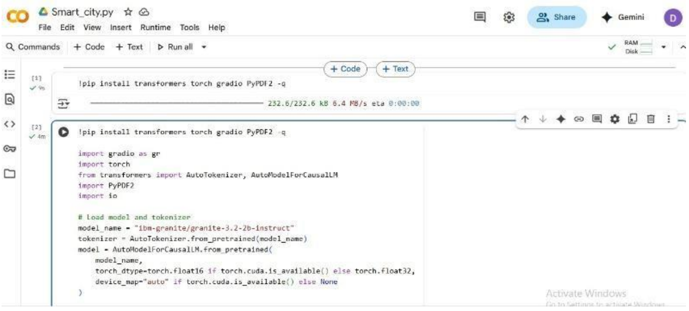
The UI is built with Gradio and has two tabs:
- Eco Tips Generator Tab: Input box for environmental problems and output box for generated solutions.
- Policy Summarization Tab: PDF upload/text input field and output box for summarized policy.

9. Testing ✔■

- Functional Testing: Ensure eco tips are generated correctly and policy summarization is accurate.
- Integration Testing: Verify UI buttons trigger correct backend functions.
- Performance Testing: Test response time for large PDFs and long prompts.
- Error Handling: Check invalid inputs such as non-PDF uploads or empty fields.

10. Screenshots ■

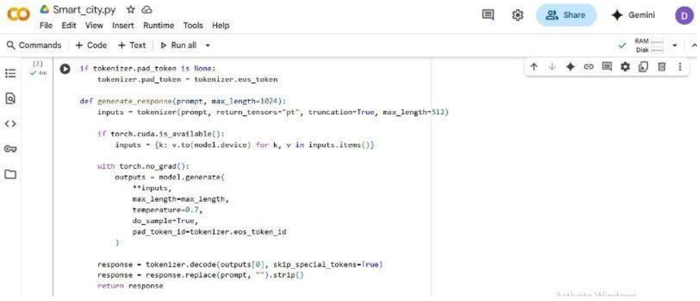
1. Input Screen (Eco Tips & Policy Summarization)



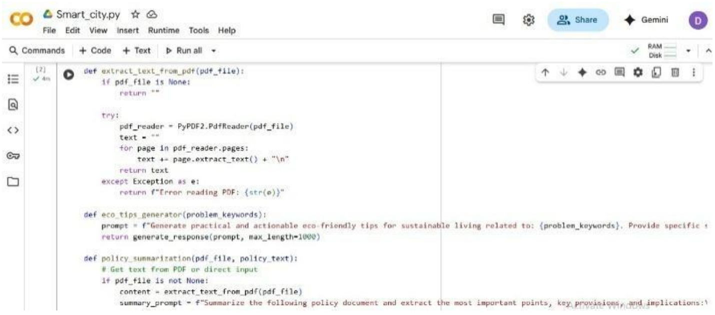
```
!pip install transformers torch gradio PyPDF2 -q
```

```
232.6/232.6 kB 6.4 MB/s eta 0:00:00
```

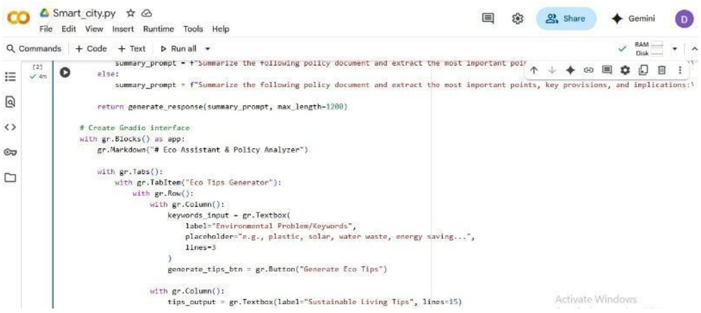
```
!pip install transformers torch gradio PyPDF2 -q

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
)
```



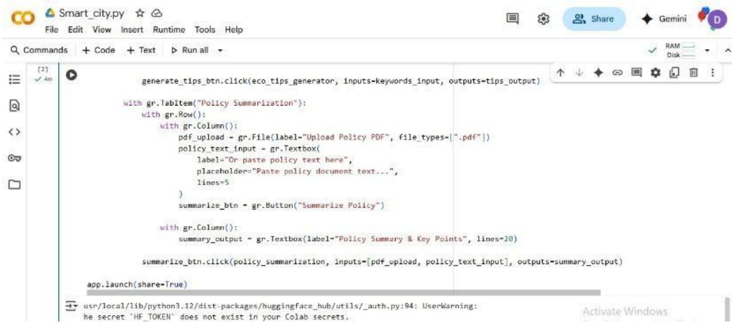
```
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

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,
            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
```


Smart_city.py
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share Gemini
RAM Disk
[2] ✓ 4m

```
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 
    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:\
```


Smart_city.py
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share Gemini
RAM Disk
[2] ✓ 4m
Activate Windows

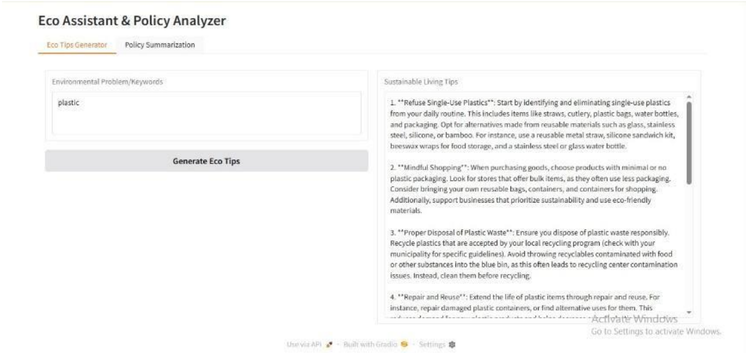
```
        summary_prompt = f"Summarize the following policy document and extract the most important poi
    else:
        summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\

    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)
```


Smart_city.py
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share Gemini
RAM Disk
[2] ✓ 4m
Activate Windows

```
            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)
```

```
usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
he secret `HF_TOKEN` does not exist in your Colab secrets.
```

2. Output Screen (Generated Tips & Summarized Policies)


Eco Assistant & Policy Analyzer
Eco Tips Generator
Policy Summarization
Generate Eco Tips
Use via API · Built with Gradio · Settings
Activate Windows
Go to Settings to activate Windows.

Environmental Problem/Keywords: plastic

Sustainable Living Tips

1. **Refuse Single-Use Plastics**: Start by identifying and eliminating single-use plastics from your daily routine. This includes items like straws, cutlery, plastic bags, water bottles, and packaging. Opt for alternatives made from reusable materials such as glass, stainless steel, silicone, or bamboo. For instance, use a reusable metal straw, silicone sandwich kit, beeswax wraps for food storage, and a stainless steel or glass water bottle.

2. **Mindful Shopping**: When purchasing goods, choose products with minimal or no plastic packaging. Look for stores that offer bulk items, as they often use less packaging. Consider bringing your own reusable bags, containers, and containers for shopping. Additionally, support businesses that prioritize sustainability and use eco-friendly materials.

3. **Proper Disposal of Plastic Waste**: Ensure you dispose of plastic waste responsibly. Recycle plastics that are accepted by your local recycling program (check with your municipality for specific guidelines). Avoid throwing recyclables contaminated with food or other substances into the blue bin, as this often leads to recycling center contamination issues. Instead, clean them before recycling.

4. **Repair and Reuse**: Extend the life of plastic items through repair and reuse. For instance, repair damaged plastic containers, or find alternative uses for them. This

11. Conclusion ■

This project demonstrates how AI-powered assistants can support sustainable city initiatives by providing eco-friendly solutions and simplifying complex policy documents. By integrating IBM Granite LLM with Gradio, the assistant offers an interactive, user-friendly platform for both citizens and policymakers. The solution lays the groundwork for future expansion with features like real-time data monitoring and integration with IoT-enabled smart city systems.