Project Title

Sustainable Smart City Assistant – Policy Analysis & Eco Assistant

---

1. Introduction

Artificial Intelligence (AI) is transforming industries, economies, and daily life. However, the rapid growth of AI technologies also raises concerns about their environmental, social, and ethical impacts. Sustainable AI refers to the design, development, and deployment of AI systems in ways that are environmentally friendly, socially responsible, and economically viable.

On the environmental side, sustainable AI emphasizes reducing the high energy consumption and carbon footprint of training large AI models by using renewable energy sources, efficient algorithms, and greener data centers.

On the social and ethical side, it ensures fairness, transparency, inclusivity, and accountability in AI systems so that they benefit society without reinforcing bias or inequality.

Sustainable AI also aligns with the broader goals of the United Nations Sustainable Development Goals (SDGs), helping to address challenges such as climate change, healthcare, education, and resource management.

In short, Sustainable AI seeks to balance technological progress with environmental preservation, ethical responsibility, and long-term societal well-being.

· Project Title: Sustainable Smart City Assistant – Policy Analysis & Eco Assistant

Team Leader : BARANIDHARAN J
Team member : ATHIVALAVAN A
Team member : AYYAPPAN R
Team member : BALAKUMAR K
Team member : BHUVANESHWARI C

---

2. Project Overview

Purpose

The Sustainable Smart City Assistant is designed to support cities and residents in building more sustainable, efficient, and connected urban environments. By leveraging AI, natural language processing, and predictive analytics, the assistant helps optimize resource usage, simplify policy understanding, and promote eco-friendly behaviors. It serves as a bridge between citizens, policymakers, and technology to foster greener, more resilient communities.

Features

Eco Tips Generator

· Key Point: Personalized sustainability advice

· Functionality: Generates actionable eco-friendly tips based on user-input keywords (e.g., plastic waste, energy saving, water conservation).

Policy Summarization

· Key Point: Simplified policy understanding

· Functionality: Extracts and summarizes key points from uploaded PDFs or pasted policy text, making complex documents accessible.

Carbon Footprint Estimator

· Key Point: Personalized environmental impact assessment

· Functionality: Estimates annual carbon footprint based on user inputs (travel, energy use, diet, etc.) and provides tailored advice for reduction.

Policy Comparison

· Key Point: Comparative policy analysis

· Functionality: Compares two policy documents (PDF or text) to highlight similarities, differences, and potential conflicts.

PDF Export

· Key Point: Shareable and printable outputs

· Functionality: Allows users to download generated tips, summaries, and reports as PDF documents.

---

3. Architecture

Frontend (Gradio)

· Built with Gradio for an intuitive, web-based UI with tabbed navigation.

· Supports file uploads, text input, and interactive outputs.

Backend (Python + Transformers)

· Powered by PyTorch and Hugging Face Transformers.

· Uses the IBM Granite-3.2B-Instruct model for text generation and summarization.

Document Processing

· PyPDF2 for PDF text extraction.

· ReportLab for PDF generation.

Model Integration

· Loads the pre-trained Granite-3.2B model via the Hugging Face transformers library.

· Implements prompt engineering for structured and context-aware responses.

---

4. Setup Instructions

Prerequisites

· Python 3.9+

· pip and virtualenv

· Internet access for model download

· GPU (recommended for faster inference)

Installation Process

1. Clone the repository (if applicable).

2. Install dependencies:

```bash
Pip install gradio torch transformers PyPDF2 reportlab
```

3. Run the application:

```bash
Python app.py
```

5. Access the web interface via the generated Gradio link.

---

6. Folder Structure (Simplified)

```
.
├── app.py                    # Main application script
├── requirements.txt          # Python dependencies
└── README.md                 # Project documentation
```

---

7. Running the Application

1. Execute the script to launch the Gradio interface.
2. Use the sidebar to navigate between features:

· Eco Tips Generator

· Policy Summarization

· Carbon Footprint Estimator

· Policy Comparison

3. Upload files or enter text as prompted.

4. View results and download PDF reports.

---

8. API Documentation (Internal)

The application uses internal function calls rather than REST APIs. Key functions include:

· generate_response(prompt): Generates AI responses using the Granite model.

· extract_text_from_pdf(pdf_file): Extracts text from uploaded PDFs.

· save_text_as_pdf(text, title): Converts text output to downloadable PDF.

---

9. Authentication

· No authentication is implemented in this demo version.

· Future versions may include user login, API keys, or role-based access.

---

10. User Interface

The Gradio UI includes:

· Tab-based navigation

· Text input fields and file uploaders

· Real-time response display

· PDF download buttons

· Clean, minimalist design for ease of use

---

11. Testing

· Unit Testing: Functions for PDF extraction, text generation, and PDF creation.

· Manual Testing: Validated with sample policies, user inputs, and edge cases.

· User Feedback: Informal testing with potential users for usability and clarity.

---

12. Screenshots

```python
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
```

```python
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 suggestion
    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...",
                        lines=3
```

```python
                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():
                    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)
```

Know process

· Large PDFs may take longer to process.

· Model inference may be slow on CPU-only systems.

· Limited context window may truncate very long documents.

---

13. Future Enhancements

· Integration with real-time city data APIs

· Multi-language support

· User accounts and history tracking

· Advanced visualization for carbon footprint data

· Deployment to cloud platforms (e.g., Hugging Face Spaces, AWS)
