

Project Title

Sustainable Smart City Assistant – Policy Analysis C 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 C Eco Assistant
 - Team Leader : BARANI V
 - Team Member: KISHORE KUMAR M
 - Team Member: KABILAN S
 - Team Member: HARIPRASATH A
-

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

```

1
2 import gradio as gr
3 import torch
4 from transformers import AutoTokenizer, AutoModelForCausalLM
5 import PyPDF2
6 import io
7
8 # Load model and tokenizer
9 model_name = "ibm-granite/granite-3.2-2b-instruct"
10 tokenizer = AutoTokenizer.from_pretrained(model_name)
11 model = AutoModelForCausalLM.from_pretrained(
12     model_name,
13     torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
14     device_map="auto" if torch.cuda.is_available() else None
15 )
16
17 if tokenizer.pad_token is None:
18     tokenizer.pad_token = tokenizer.eos_token
19
20 def generate_response(prompt, max_length=1024):
21     inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)
22
23     if torch.cuda.is_available():
24         inputs = {k: v.to(model.device) for k, v in inputs.items()}
25
26     with torch.no_grad():
27         outputs = model.generate(
28             **inputs,
29             max_length=max_length,
30             temperature=0.7,
31             do_sample=True,
32             pad_token_id=tokenizer.eos_token_id
33         )
34
35     response = tokenizer.decode(outputs[0], skip_special_tokens=True)
36     response = response.replace(prompt, "").strip()
37     return response

```

```

98
99 def extract_text_from_pdf(pdf_file):
100     if pdf_file is None:
101         return ""
102
103     try:
104         pdf_reader = PyPDF2.PdfReader(pdf_file)
105         text = ""
106         for page in pdf_reader.pages:
107             text += page.extract_text() + "\n"
108         return text
109     except Exception as e:
110         return f"Error reading PDF: {str(e)}"
111
112 def eco_tips_generator(problem_keywords):
113     prompt = f"Generate practical and actionable eco-friendly tips for sustainable living related to: {problem_keywords}. Provide specific solutions and suggestions."
114     return generate_response(prompt, max_length=1000)
115
116 def policy_summarization(pdf_file, policy_text):
117     # Get text from PDF or direct input
118     if pdf_file is not None:
119         content = extract_text_from_pdf(pdf_file)
120         summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{content}"
121     else:
122         summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{policy_text}"
123
124     return generate_response(summary_prompt, max_length=1200)
125
126 # Create Gradio Interface
127 with gr.Blocks() as app:
128     gr.Markdown("# Eco Assistant & Policy Analyzer")
129
130     with gr.Tabs():
131         with gr.TabItem("Eco Tips Generator"):
132             with gr.Row():
133                 with gr.Column():
134                     keywords_input = gr.Textbox(
135                         label="Environmental Problem/Keywords",
136                         placeholder="e.g., plastic, solar, water waste, energy saving...",
137                         lines=3
138                     )

```

```

139                 with gr.Column():
140                     keywords_input = gr.Textbox(
141                         label="Environmental Problem/Keywords",
142                         placeholder="e.g., plastic, solar, water waste, energy saving...",
143                         lines=3
144                     )
145                 generate_tips_btn = gr.Button("Generate Eco Tips")
146
147             with gr.Column():
148                 tips_output = gr.Textbox(label="Sustainable Living Tips", lines=15)
149
150             generate_tips_btn.click(eco_tips_generator, inputs=keywords_input, outputs=tips_output)
151
152         with gr.TabItem("Policy Summarization"):
153             with gr.Row():
154                 with gr.Column():
155                     pdf_upload = gr.File(label="Upload Policy PDF", file_types=[".pdf"])
156                     policy_text_input = gr.Textbox(
157                         label="Or paste policy text here",
158                         placeholder="Paste policy document text...",
159                         lines=5
160                     )
161                 summarize_btn = gr.Button("Summarize Policy")
162
163             with gr.Column():
164                 summary_output = gr.Textbox(label="Policy Summary & Key Points", lines=20)
165
166             summarize_btn.click(policy_summarization, inputs=[pdf_upload, policy_text_input], outputs=summary_output)
167
168 app.launch(share=True)
169
170

```

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)
