PROJECTTITLE

PROJECTDOCUMENTATION

1.Introduction

- Projecttitle : **HEALTH AI**
- TeamLeader : **LATHISH KUMAR B**
- Teammember : YUVARAJ K J
- Teammember : VIKRAM A
- Teammember : DHILIP K

---

2. Project Overview

Purpose

The purpose of Health AI is to help cities and their residents thrive in a more eco-conscious and connected urban environment. By leveraging AI and real-time data, the assistant aids in optimizing essential resources like energy, water, and waste while encouraging sustainable behaviors. For city officials, the assistant serves as a decision-making partner, providing insights, forecasting tools, and summarizations of complex policies to support strategic planning.

The assistant bridges the gap between technology, governance, and community engagement, aiming to create greener, more efficient, and resilient cities.

---

3. Problem Statement

Challenges

- Lack of practical eco-friendly guidance for citizens.
- Government/NGO policy documents are lengthy and complex, making them difficult for the general public to understand.

Need

- An intelligent assistant that generates actionable eco-tips.
- Automatic summarization of PDF policy files to make them easier to digest.

4. Objectives of the Project

The objectives of the project are:

1. To design a user-friendly assistant for sustainable living.
2. To use AI language models (IBM Granite) for generating eco-tips.
3. To integrate policy summarization from PDF or text files.
4. To provide a web-based solution using Gradio.

---

5. Literature Review

Artificial Intelligence in Sustainability

AI supports various sustainability initiatives, such as waste management, smart grids, and renewable energy forecasting. The technology helps reduce resource consumption and environmental impact by providing data-driven insights.

NLP in Policy Analysis

Natural Language Processing (NLP) simplifies complex legal and policy texts. It enables automatic extraction and summarization of key information, which is essential for effective policy analysis and decision-making.

Existing Solutions

- Chatbots exist, but they often lack domain-specific eco-advice.
- Existing summarization tools don't focus on sustainability policies, which limits their use for environmental applications.

---

6. Methodology

Tools & Technologies

- **Language**: Python
- **Framework**: Gradio
- **Libraries**: HuggingFace Inference API (Granite Model), PyPDF2

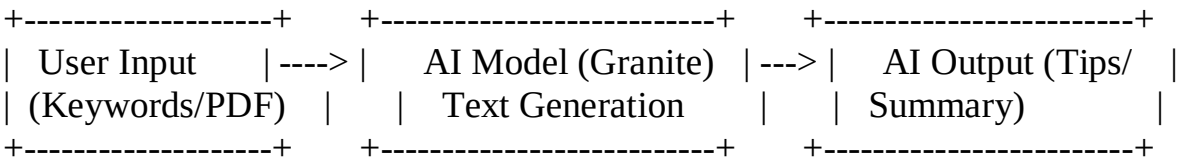
- **Hosting**: Local machine or HuggingFace Spaces

Workflow

1. **Input**: The user provides keywords or uploads a PDF.
2. **Preprocessing**: Extract text (if a PDF is uploaded).
3. **AI Model**: The IBM Granite model generates tips or summaries.
4. **Output**: Display results in a user-friendly UI.

---

7. System Architecture



---

8. Implementation

Extracting Text from PDF

- Used **PyPDF2** to read PDF documents.
- Extracted text is cleaned and passed to the Granite model for processing.

Eco-Tips Generator

- **Input**: Problem keywords (e.g., "plastic pollution").
- **AI Prompt**: "Give eco-friendly tips for: <keywords>."
- **Output**: Practical, short, actionable eco-tips.

Policy Summarization

- **Input**: PDF or manual policy text.
- **AI Prompt**: "Summarize the following policy document: <policy-text>."
- **Output**: Summary of key provisions, rules, and implications.

Gradio Interface

- Built two tabs for the user interface:
    1. **Eco-Tips Generator**

2. **Policy Summarization**

---

9. Sample Outputs

Example 1: Eco-Tips

Input: "plastic pollution"

Output:

- Use reusable cloth bags instead of plastics.
- Reduce bottled water usage by carrying refillable bottles.
- Promote biodegradable packaging materials.
- Conduct awareness drives in schools and communities.

Example 2: Policy Summarization

Input PDF: "RenewableEnergyPolicy2023"

Output:

- Encourages the installation of rooftop solar panels.
- Provides subsidies for EV charging stations.
- Mandates waste-to-energy plants in urban areas.
- Introduces stricter penalties for polluting industries.

---

10. Results & Discussion

- The assistant successfully provides **domain-relevant eco-tips**.
- The summaries are **short, clear, and actionable**, making them useful for a wide range of users, including students, policymakers, NGOs, and citizens.

---

11. Advantages

- **Simple**: Lightweight and mobile-friendly.
- **Real-World Application**: Works with actual PDF policy documents.

- **Expandable**: Can be extended to support multiple languages.
- **No GPU Required**: Utilizes the cloud-hosted Granite model via HuggingFace API.

---

12. Limitations

- Depends on HuggingFace API availability.
- Summarization may miss some minor details.
- Accuracy is dependent on the AI model's capabilities.

---

13. Future Enhancements

- Add **speech-to-text** for voice queries.
- Provide **data visualizations** for policy insights.
- Support **regional languages** (e.g., Tamil, Hindi).
- Enable a **policy comparison tool** for comparing multiple documents.
- Deploy the application on **HuggingFace Spaces** for global access.

---

14. Conclusion:

This project demonstrates how AI and NLP can be used to support sustainability efforts. The **Eco Assistant** encourages individuals to adopt eco-friendly habits, and the **Policy Analyzer** makes government regulations easier to understand. The project has significant social, educational, and environmental benefits.

Program Code:

```
import gradio as gr

import torch

from transformers import AutoTokenizer, AutoModelForCausalLM
```

```
# 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

def disease_prediction(symptoms):
    prompt = f"Based on the following symptoms, provide possible medical conditions and general medication suggestions. Always emphasize the importance of consulting a doctor for proper diagnosis.\n\nSymptoms: {symptoms}\n\nPossible conditions and recommendations:\n\n**IMPORTANT: This is for informational purposes only. Please consult a healthcare professional for proper diagnosis and treatment.**\n\nAnalysis:"
    return generate_response(prompt, max_length=1200)

def treatment_plan(condition, age, gender, medical_history):
    prompt = f"Generate personalized treatment suggestions for the following patient information. Include home remedies and general medication guidelines.\n\nMedical Condition: {condition}\nAge: {age}\nGender: {gender}\nMedical History: {medical_history}\n\nPersonalized treatment plan including home remedies and medication guidelines:\n\n**IMPORTANT:
```

```
This is for informational purposes only. Please consult a healthcare professional for proper treatment.**\n\nTreatment Plan:"

    return generate_response(prompt, max_length=1200)


# Create Gradio interface

with gr.Blocks() as app:

    gr.Markdown("# Medical AI Assistant")

    gr.Markdown("**Disclaimer: This is for informational purposes only. Always consult healthcare professionals for medical advice.**")


    with gr.Tabs():

        with gr.TabItem("Disease Prediction"):

            with gr.Row():

                with gr.Column():

                    symptoms_input = gr.Textbox(

                        label="Enter Symptoms",

                        placeholder="e.g., fever, headache, cough, fatigue...",

                        lines=4

                    )

                    predict_btn = gr.Button("Analyze Symptoms")


                with gr.Column():

                    prediction_output = gr.Textbox(label="Possible Conditions & Recommendations", lines=20)
```

```
            predict_btn.click(disease_prediction, inputs=symptoms_input,
outputs=prediction_output)


        with gr.TabItem("Treatment Plans"):
            with gr.Row():
                with gr.Column():
                    condition_input = gr.Textbox(
                        label="Medical Condition",
                        placeholder="e.g., diabetes, hypertension, migraine...",
                        lines=2
                    )
                    age_input = gr.Number(label="Age", value=30)
                    gender_input = gr.Dropdown(
                        choices=["Male", "Female", "Other"],
                        label="Gender",
                        value="Male"
                    )
                    history_input = gr.Textbox(
                        label="Medical History",
                        placeholder="Previous conditions, allergies, medications or
None",
                        lines=3
```

```
        )
        plan_btn = gr.Button("Generate Treatment Plan")

    with gr.Column():
        plan_output = gr.Textbox(label="Personalized Treatment Plan", lines=20)

    plan_btn.click(treatment_plan, inputs=[condition_input, age_input, gender_input, history_input], outputs=plan_output)

app.launch(share=True)
```

OUTPUT: