```
!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

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€}”
```

```
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...",
            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)
```