```
!pip install transformers torch gradio PYPDF2 -q
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 7.3 MB/s e
```

Double-click (or enter) to edit

```
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 torc
    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,

    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(e)}"

    def eco_tips_generator(problem_keywords):
        prompt = f"Generate practical and actionable eco-friendly tips f
        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 a
        else:
            summary_prompt = f"Summarize the following policy document a

        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 was
                            lines=3
                        )
                        generate_tips_btn = gr.Button("Generate Eco Tips

                    with gr.Column():
                        tips_output = gr.Textbox(label="Sustainable Livi

                generate_tips_btn.click(eco_tips_generator, inputs=keywo

            with gr.TabItem("Policy Summarization"):
                with gr.Row():
                    with gr.Column():
                        pdf_upload = gr.File(label="Upload Policy 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 Summar

            summarize_btn.click(policy_summarization, inputs=[pdf_uplo

app.launch(share=True)
```

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(e)}"
```

```
tokenizer.json:  3.48M/? [00:00<00:00, 46.7MB/s]
added_tokens.json:  100%                 87.0/87.0  [00:00<00:00,  2.11kB/s]
special_tokens_map.json:  100%           701/701  [00:00<00:00,  29.6kB/s]
config.json:  100%                       786/786  [00:00<00:00,  22.9kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json:  29.8k/?  [00:00<00:00,  1.20MB/s]
Fetching 2 files:  100%                  2/2  [01:13<00:00,  73.03s/it]
model-00001-of-                          5.00G/5.00G  [01:12<00:00,  77.2MB/s]
00002.safetensors:  100%
```

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)
```

```
00002.safetensors:  100%
Loading checkpoint shards:  100%         2/2  [00:21<00:00,  9.03s/it]
generation_config.json:  100%            137/137  [00:00<00:00,  9.79kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True
```

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 sav
                    lines=3
                )
                generate_tips_btn = gr.Button("Generate Eco Tips")

            with gr.Column():
                tips_output = gr.Textbox(label="Sustainable Living Tips", line

        generate_tips_btn.click(eco_tips_generator, inputs=keywords_input, out
```

```
    with gr.TabItem("Policy Summarization"):
        with gr.Row():
            with gr.Column():
                pdf_upload = gr.File(label="Upload Policy PDF", file_types=[".
                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

        summarize_btn.click(policy_summarization, inputs=[pdf_upload, policy_t
```

app.launch(share=True)