```
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)}"
def requirement_analysis(pdf_file, prompt_text):
        # Get text from PDF or prompt
        if pdf file is not None:
               content = extract_text_from_pdf(pdf_file)
analysis_prompt = f"Analyze the following document and extract key software requirements. Organize
them into functional requirements, non-functional requirements, and technical specifications:\n\n{content}"
        else:
               analysis_prompt = f"Analyze the following requirements and organize them into functional
requirements, non-functional requirements, and technical specifications:\n\n{prompt text}"
        return generate response (analysis prompt, max length=1200)
def code generation(prompt, language):
        \verb|code_prompt = f"Generate {language}| code for the following requirement: \\ \verb|n\n{prompt}| n\nCode: \\ \verb|code_prompt = f"Generate {language}| n\
        return generate_response(code_prompt, max length=1200)
# Create Gradio interface
with gr.Blocks() as app:
        gr.Markdown("# AI Code Analysis & Generator")
        with gr.Tabs():
               with gr.TabItem("Code Analysis"):
                       with gr.Row():
                                with gr.Column():
                                       pdf_upload = gr.File(label="Upload PDF", file_types=[".pdf"])
prompt_input = gr.Textbox(
                                                label="Or write requirements here",
```

```
placeholder="Describe your software requirements...",
                      lines=5
                  analyze btn = gr.Button("Analyze")
               with gr.Column():
                  analysis_output = gr.Textbox(label="Requirements Analysis", lines=20)
           analyze btn.click(requirement analysis, inputs=[pdf upload, prompt input],
outputs=analysis_output)
       with gr.TabItem("Code Generation"):
           with gr.Row():
               with gr.Column():
                  code_prompt = gr.Textbox(
    label="Code Requirements",
                      placeholder="Describe what code you want to generate...",
                      lines=5
                  label="Programming Language",
                      value="Python"
                  generate btn = gr.Button("Generate Code")
               with gr.Column():
                  code_output = gr.Textbox(label="Generated Code", lines=20)
           generate btn.click(code generation, inputs=[code prompt, language dropdown],
outputs=code_output)
app.launch(share=True)
```