```
!pip install torch torchaudio torchvision transformers gradio Pillow


import gradio as gr
from transformers import pipeline, AutoModelForQuestionAnswering, AutoTokenizer
import torch
from PIL import Image
import json

# ================================================================================
# INPUT PROCESSOR - Handles multimodal inputs and speech-to-text conversion
# ================================================================================
class InputProcessor:
    def __init__(self):
        # Whisper for speech-to-text conversion
        self.stt = pipeline("automatic-speech-recognition", model="openai/whisp

    def process_inputs(self, audio_file, text_input, image_file):
        """Process all input modalities and convert to standardized format"""
        processed_data = {
            'text_data': '',
            'image_data': None,
            'has_audio': False,
            'has_text': False,
            'has_image': False
        }

        # Process audio input - CONVERT AUDIO TO TEXT HERE
        if audio_file is not None:
            print(f"Processing audio file: {audio_file}")
            try:
                transcription = self.stt(audio_file)["text"]
                processed_data['text_data'] = transcription
                processed_data['has_audio'] = True
                print(f"Audio transcribed to: {transcription}")
            except Exception as e:
                print(f"Error transcribing audio: {e}")
                processed_data['text_data'] = f"Error processing audio: {str(e)

        # Process text input
        if text_input and text_input.strip():
            # Combine with audio transcription if both exist
            if processed_data['text_data']:
                processed_data['text_data'] += " " + text_input
                print(f"Combined audio + text: {processed_data['text_data']}")
            else:
                processed_data['text_data'] = text_input
            processed_data['has_text'] = True

        # Process image input
        if image_file is not None:
            processed_data['image_data'] = image_file
            processed_data['has_image'] = True
```

```
            print(f"Image processed: {image_file}")

        return processed_data


# ================================================================================
# GATING MECHANISM/ROUTER - Determines which expert models to engage
# ================================================================================
class GatingMechanism:
    def __init__(self):
        pass

    def route_to_experts(self, processed_data):
        """Determine which AI experts should be activated based on input data"'
        routing_plan = {
            'activate_clinical_llm': False,
            'activate_clinical_bert': False,
            'activate_llava_med': False,
            'activate_biomedclip': False,
            'data_for_experts': {}
        }

        # Route text data (including transcribed audio) to appropriate text-bas
        if processed_data['text_data']:
            routing_plan['activate_clinical_llm'] = True
            routing_plan['activate_clinical_bert'] = True
            routing_plan['data_for_experts']['text'] = processed_data['text_dat
            print(f"Routing text to Clinical LLM and ClinicalBERT: {processed_c

        # Route image data to vision experts
        if processed_data['has_image']:
            routing_plan['activate_llava_med'] = True
            routing_plan['activate_biomedclip'] = True
            routing_plan['data_for_experts']['image'] = processed_data['image_c
            print("Routing image to LLaVA-Med and BioMedCLIP")

        return routing_plan



# ================================================================================
# AI HEALTHCARE EXPERTS - Specialized models for different modalities
# ================================================================================
class AIHealthcareExperts:
    def __init__(self):
        # Initialize ClinicalBERT for QA
        try:
            self.tokenizer = AutoTokenizer.from_pretrained("emilyalsentzer/Bio_
            self.model_qa = AutoModelForQuestionAnswering.from_pretrained("emil
            self.clinical_bert = pipeline("question-answering", model=self.mode
        except Exception as e:
            print(f"Error loading ClinicalBERT: {e}")
            self.clinical_bert = None

    def clinical_llm_analysis(self, text_data):
```

```
    def clinical_llm_analysis(self, text_data):
        """Clinical LLM for general medical text analysis"""
        # Enhanced analysis with symptom extraction
        analysis = {
            'expert': 'Clinical_LLM',
            'findings': f"Clinical analysis of patient description: {text_data}
            'confidence': 0.85,
            'key_symptoms': self._extract_symptoms(text_data)
        }
        print(f"Clinical LLM analysis completed with confidence: {analysis['cor
        return analysis

    def _extract_symptoms(self, text):
        """Basic symptom extraction from text"""
        common_symptoms = ['pain', 'fever', 'cough', 'fatigue', 'headache', 'na
        found_symptoms = []
        text_lower = text.lower()
        for symptom in common_symptoms:
            if symptom in text_lower:
                found_symptoms.append(symptom)
        return found_symptoms if found_symptoms else ['General discomfort menti

    def clinical_bert_qa(self, text_data):
        """ClinicalBERT for symptom extraction and medical QA"""
        if self.clinical_bert is None:
            return {
                'expert': 'ClinicalBERT',
                'qa_results': [{'question': 'Model unavailable', 'answer': 'Cli
                'confidence': 0.0
            }

        # Generate relevant clinical questions
        questions = [
            "What symptoms is the patient experiencing?",
            "What is the patient's main complaint?",
            "Are there any concerning symptoms mentioned?",
            "What is the duration of symptoms?"
        ]

        results = []
        for question in questions:
            try:
                answer = self.clinical_bert(question=question, context=text_dat
                results.append({
                    'question': question,
                    'answer': answer['answer'],
                    'confidence': answer['score']
                })
            except Exception as e:
                results.append({
                    'question': question,
                    'answer': f'Unable to extract: {str(e)}',
                    'confidence': 0.0
                })
```

```
        avg_confidence = sum([r['confidence'] for r in results]) / len(results)
        print(f"ClinicalBERT QA completed with average confidence: {avg_confide

        return {
            'expert': 'ClinicalBERT',
            'qa_results': results,
            'confidence': avg_confidence
        }

    def llava_med_analysis(self, image_data):
        """LLaVA-Med for medical image analysis with natural language"""
        # Placeholder for LLaVA-Med - would use actual model
        analysis = {
            'expert': 'LLaVA_Med',
            'findings': 'Medical image analysis: Examining uploaded medical ima
            'confidence': 0.78,
            'detected_abnormalities': ['Image analysis placeholder - would dete
        }
        print(f"LLaVA-Med analysis completed with confidence: {analysis['confic
        return analysis

    def biomedclip_classification(self, image_data):
        """BioMedCLIP for medical image classification"""
        # Placeholder for BioMedCLIP - would use actual model
        classification = {
            'expert': 'BioMedCLIP',
            'primary_diagnosis': 'Medical condition classification pending',
            'confidence': 0.82,
            'differential_diagnoses': ['Condition A', 'Condition B', 'Normal'],
            'probabilities': [0.45, 0.35, 0.20]
        }
        print(f"BioMedCLIP classification completed with confidence: {classific
        return classification


# ================================================================================
# DIAGNOSTIC INTEGRATOR/SYNTHESIZER - Combines expert outputs
# ================================================================================
class DiagnosticIntegrator:
    def __init__(self):
        pass

    def synthesize_findings(self, expert_outputs):
        """Combine and synthesize findings from multiple AI experts"""
        # Extract symptoms from all expert outputs
        symptoms = []
        possible_conditions = []

        for output in expert_outputs:
            expert_name = output.get('expert', 'Unknown')

            # Extract symptoms from Clinical LLM
            if expert_name == 'Clinical_LLM':
                expert_symptoms = output.get('key_symptoms', [])
```

```
                symptoms.extend(expert_symptoms)

            # Extract symptoms from ClinicalBERT Q&A
            elif expert_name == 'ClinicalBERT':
                qa_results = output.get('qa_results', [])
                for qa in qa_results:
                    if qa['confidence'] > 0.3 and 'symptom' in qa['question'].l
                        answer = qa['answer'].lower()
                        # Extract common symptoms from answers
                        symptom_keywords = ['pain', 'fever', 'cough', 'fatigue'
                        for keyword in symptom_keywords:
                            if keyword in answer and keyword not in symptoms:
                                symptoms.append(keyword)

            # Extract conditions from image analysis
            elif expert_name in ['LLaVA_Med', 'BioMedCLIP']:
                diagnosis = output.get('primary_diagnosis', output.get('findinc
                if diagnosis and 'placeholder' not in diagnosis.lower():
                    possible_conditions.append(diagnosis)

        # Remove duplicates and clean up
        symptoms = list(set([s for s in symptoms if s and s != 'General discomf

        synthesis = {
            'symptoms': symptoms if symptoms else ['general discomfort'],
            'possible_conditions': possible_conditions,
            'medications': self._generate_medication_advice(symptoms),
            'home_care': self._generate_home_care_advice(symptoms),
            'when_to_see_doctor': self._generate_doctor_advice(symptoms)
        }

        print(f"Diagnostic integration completed: {len(symptoms)} symptoms ider
        return synthesis

    def _generate_home_care_advice(self, symptoms):
        """Generate simple home care advice based on symptoms"""
        advice = []

        if 'cough' in symptoms:
            advice.extend(['drink warm liquids', 'use honey for throat'])
        if 'fever' in symptoms:
            advice.extend(['rest and stay hydrated', 'use cool compress'])
        if 'pain' in symptoms:
            advice.extend(['apply heat or cold as needed', 'gentle stretching']
        if 'fatigue' in symptoms:
            advice.extend(['get plenty of sleep', 'eat nutritious meals'])

        # Default advice if no specific symptoms
        if not advice:
            advice = ['rest well', 'stay hydrated', 'eat healthy foods']

        return advice

    def _generate_doctor_advice(self, symptoms):
```

```
        """Generate advice on when to see doctor"""
        urgent_symptoms = ['chest pain', 'shortness of breath', 'severe pain']

        for symptom in symptoms:
            for urgent in urgent_symptoms:
                if urgent in symptom:
                    return ['see doctor immediately if symptoms worsen', 'call

        return ['see doctor if symptoms last more than a few days', 'contact dc

    def _generate_medication_advice(self, symptoms):
        """Generate medication suggestions based on symptoms"""
        medications = []

        # Over-the-counter medications based on symptoms
        if 'cough' in symptoms:
            medications.append('cough drops or cough syrup')
        if 'fever' in symptoms:
            medications.append('paracetamol or ibuprofen for fever')
        if 'pain' in symptoms or 'headache' in symptoms:
            medications.append('paracetamol or ibuprofen for pain')
        if 'nausea' in symptoms:
            medications.append('anti-nausea medication if needed')
        if 'fatigue' in symptoms:
            medications.append('multivitamins to support energy')

        # For respiratory symptoms
        if 'shortness of breath' in ' '.join(symptoms) or 'breathing' in ' '.jc
            medications.append('see doctor immediately - do not self-medicate')

        # For chest pain
        if 'chest pain' in ' '.join(symptoms):
            medications.append('seek immediate medical attention - do not self-

        # Default if no specific medications needed
        if not medications:
            medications = ['no specific medication needed - focus on rest and h

        return medications


# ================================================================================
# OUTPUT GENERATOR - Formats final diagnosis and recommendations
# ================================================================================
class OutputGenerator:
    def __init__(self):
        pass

    def generate_final_output(self, synthesis):
        """Generate simple, patient-friendly output"""
        output = {
            'symptoms': ', '.join(synthesis['symptoms']),
            'medications': ', '.join(synthesis['medications']),
            'home_care': ', '.join(synthesis['home_care']),
            'doctor_advice': ', '.join(synthesis['when_to_see_doctor'])
```

```
            'doctor_advice': ', '.join(synthesis['when_to_see_doctor']),
            'formatted_report': self._format_simple_report(synthesis)
        }
        return output

    def _format_simple_report(self, synthesis):
        """Format a simple, patient-friendly report"""
        report = "=== Your Health Summary ===\n\n"

        report += f"Symptoms you mentioned: {', '.join(synthesis['symptoms'])}\

        report += "Medications you can try:\n"
        for med in synthesis['medications']:
            report += f"• {med}\n"
        report += "\n"

        report += "What you can do at home:\n"
        for advice in synthesis['home_care']:
            report += f"• {advice}\n"

        report += "When to see a doctor:\n"
        for advice in synthesis['when_to_see_doctor']:
            report += f"• {advice}\n"

        if synthesis['possible_conditions']:
            report += f"\nPossible conditions to discuss with doctor: {', '.joi

        report += "\n⚠️ Important: This is AI advice only. Always talk to a rea

        return report


# ================================================================================
# MAIN ORCHESTRATION SYSTEM
# ================================================================================
class AIGPDoctorSystem:
    def __init__(self):
        self.input_processor = InputProcessor()
        self.gating_mechanism = GatingMechanism()
        self.ai_experts = AIHealthcareExperts()
        self.diagnostic_integrator = DiagnosticIntegrator()
        self.output_generator = OutputGenerator()

    def process_patient_case(self, audio_file, text_input, image_file):
        """Main processing pipeline following the architectural flow"""
        print("=== Starting AI GP Doctor Analysis ===")

        # Step 1: Input Processing (Audio → Text conversion happens here)
        processed_data = self.input_processor.process_inputs(audio_file, text_i

        # Step 2: Gating/Routing
        routing_plan = self.gating_mechanism.route_to_experts(processed_data)

        # Step 3: Expert Analysis (Text from audio + manual text goes to models
```

```
        # Step 3: Expert Analysis (Text from audio + manual text goes to models)
        expert_outputs = []

        if routing_plan['activate_clinical_llm']:
            output = self.ai_experts.clinical_llm_analysis(routing_plan['data_f
            expert_outputs.append(output)

        if routing_plan['activate_clinical_bert']:
            output = self.ai_experts.clinical_bert_qa(routing_plan['data_for_ex
            expert_outputs.append(output)

        if routing_plan['activate_llava_med'] and 'image' in routing_plan['data
            output = self.ai_experts.llava_med_analysis(routing_plan['data_for_
            expert_outputs.append(output)

        if routing_plan['activate_biomedclip'] and 'image' in routing_plan['dat
            output = self.ai_experts.biomedclip_classification(routing_plan['da
            expert_outputs.append(output)

        # Step 4: Diagnostic Integration
        synthesis = self.diagnostic_integrator.synthesize_findings(expert_outpu

        # Step 5: Output Generation
        final_output = self.output_generator.generate_final_output(synthesis)

        print("=== AI GP Doctor Analysis Complete ===")
        return final_output


# ================================================================================
# GRADIO INTERFACE - FIXED: Removed 'optional' parameters
# ================================================================================
def create_gradio_interface():
    ai_system = AIGPDoctorSystem()

    def process_interface(audio_file, text_input, image_file):
        """Interface function for Gradio"""
        try:
            result = ai_system.process_patient_case(audio_file, text_input, imag

            # Return simple, patient-friendly outputs
            return (
                result['formatted_report'],
                result['symptoms'],
                result['medications'],
                result['doctor_advice']
            )
        except Exception as e:
            error_msg = f"Error processing request: {str(e)}"
            return (error_msg, error_msg, error_msg, error_msg)

    # Create Gradio interface - REMOVED 'optional=True' parameters
    demo = gr.Interface(
        fn=process_interface,
        inputs=[
```

```
            gr.Audio(type="filepath", label="Voice/Audio Input (will be converte
            gr.Textbox(label="Textual Description", placeholder="Describe sympto
            gr.Image(type="filepath", label="Medical Images")
        ],
        outputs=[
            gr.Textbox(label="Health Summary", lines=12),
            gr.Textbox(label="Your Symptoms", lines=2),
            gr.Textbox(label="Recommended Medications", lines=3),
            gr.Textbox(label="Doctor Advice", lines=2)
        ],
        title="AI GP Doctor - Simple Medical Advice",
        description="Tell me your symptoms (by voice or text) and I'll give you
        examples=[
            [None, "I have chest pain and can't breathe well", None],
            [None, "I have a cough and feel tired", None],
        ]
    )

    return demo

# ================================================================================
# MAIN EXECUTION
# ================================================================================
if __name__ == "__main__":
    demo = create_gradio_interface()
    demo.launch()
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
  warnings.warn(
config.json: 100%                                                 1.98k/1.98k [00:00<00:00, 35.7kB/
                                                                  s]
model.safetensors: 100%                                           290M/290M [00:03<00:00, 124MB/
                                                                  s]
generation_config.json: 100%                                      3.81k/3.81k [00:00<00:00, 73.0kB/
                                                                  s]
tokenizer_config.json: 100%                                       283k/283k [00:00<00:00, 3.45MB/
                                                                  s]
vocab.json: 100%                                                  836k/836k [00:00<00:00, 6.52MB/
                                                                  s]
tokenizer.json: 100%                                              2.48M/2.48M [00:00<00:00, 7.61MB/
                                                                  s]
merges.txt: 100%                                                  494k/494k [00:00<00:00, 3.81MB/
                                                                  s]
```

```
-
normalizer.json: 100%  52.7k/52.7k [00:00<00:00, 909kB/s]
added_tokens.json: 100%  34.6k/34.6k [00:00<00:00, 378kB/s]
special_tokens_map.json: 100%  2.19k/2.19k [00:00<00:00, 42.4kB/s]
preprocessor_config.json: 100%  185k/185k [00:00<00:00, 2.88MB/s]
Device set to use cpu
config.json: 100%  385/385 [00:00<00:00, 10.9kB/s]
vocab.txt: 100%  213k/213k [00:00<00:00, 3.43MB/s]
pytorch_model.bin: 100%  436M/436M [00:02<00:00, 166MB/s]
model.safetensors: 100%  436M/436M [00:07<00:00, 48.2MB/s]
Some weights of BertForQuestionAnswering were not initialized from the mode
You should probably TRAIN this model on a down-stream task to be able to us
Device set to use cpu
It looks like you are running Gradio on a hosted a Jupyter notebook. For th

Colab notebook detected. To show errors in colab notebook, set debug=True i
* Running on public URL: https://e2dc7656b20d28ab52.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgra
```

# AI Health Helper - Simple Medical Advice

Tell me your symptoms (by voice or text) and I'll give you simple advice on what you can do.