```
# Install required packages
!pip install -q transformers accelerate bitsandbytes
!pip install -q sentence-transformers faiss-cpu
!pip install -q gradio PyPDF2 requests
!pip install -q peft torch

import os
import time
import torch
import requests
import PyPDF2
import numpy as np
from io import BytesIO
from typing import List, Dict, Tuple
import warnings
warnings.filterwarnings('ignore')

# Check GPU availability
print(f"GPU Available: {torch.cuda.is_available()}")
print(f"GPU Device: {torch.cuda.get_device_name(0) if torch.cuda.is_available() else 'None'}")
print(f"CUDA Version: {torch.version.cuda}")
```

```
GPU Available: True
GPU Device: Tesla T4
CUDA Version: 12.4
```

Authentication and Model Loading

```
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig
from huggingface_hub import login


HF_TOKEN = "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"
login(token=HF_TOKEN)

# Prepare 4-bit quantization config
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16
)

model_name = "Writer/camel-5b-hf"
print(f"Loading {model_name} with 4-bit…")

start_time = time.time()
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)

try:
    model = AutoModelForCausalLM.from_pretrained(
        model_name,
        quantization_config=bnb_config,
        device_map="auto",
        trust_remote_code=True,
        torch_dtype=torch.float16
    )
except RuntimeError as e:
    print(f"4-bit load failed ({e}). Retrying with 8-bit…")
```

```
        bnb_config.load_in_4bit = False
        bnb_config.load_in_8bit = True
        model = AutoModelForCausalLM.from_pretrained(
            model_name,
            quantization_config=bnb_config,
            device_map="auto",
            trust_remote_code=True,
            torch_dtype=torch.float16
        )

    # Ensure pad_token is set
    if tokenizer.pad_token is None:
        tokenizer.pad_token = tokenizer.eos_token

    loading_time = time.time() - start_time
    print(f"Model ready in {loading_time:.2f}s")
    print(f"GPU Mem: {torch.cuda.memory_allocated()/1024**3:.2f} GB")
```

```
Loading Writer/camel-5b-hf with 4-bit…
Loading checkpoint shards: 100%  3/3 [02:07<00:00, 35.26s/it]
Model ready in 130.28s
GPU Mem: 5.82 GB
```

Download and process PDFs

```
MEDICAL_PDFS = [
    {
        "url": "https://www.who.int/publications/i/item/9789240015902",
        "title": "WHO Guidelines for Clinical Management",
        "fallback_url": "https://apps.who.int/iris/rest/bitstreams/1278777/retrieve"
    },
    {
        "url": "https://www.cdc.gov/flu/pdf/professionals/antivirals/antiviral-summary-clinicia
        "title": "CDC Antiviral Guidelines",
        "fallback_url": "https://www.cdc.gov/flu/pdf/professionals/antivirals/antiviral-summary
    },
    {
        "url": "https://www.heart.org/-/media/files/health-topics/consumer-healthcare/what-is-c
        "title": "AHA Cardiovascular Disease Guide",
        "fallback_url": "https://www.heart.org/-/media/files/health-topics/consumer-healthcare/
    }
]

def download_pdf(url: str, title: str) -> bytes:
    """Download PDF from URL with error handling"""
    try:
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36'
        }
        response = requests.get(url, headers=headers, timeout=30)
        response.raise_for_status()
        return response.content
    except Exception as e:
        print(f"Failed to download {title}: {str(e)}")
        return None

def extract_text_from_pdf(pdf_content: bytes) -> str:
    """Extract text from PDF bytes"""
    try:
```

```
            pdf_file = BytesIO(pdf_content)
            reader = PyPDF2.PdfReader(pdf_file)
            text = ""
            for page in reader.pages:
                text += page.extract_text() + "\n"
            return text
        except Exception as e:
            print(f"PDF extraction failed: {str(e)}")
            return ""

    # Sample medical text as fallback
    FALLBACK_MEDICAL_TEXT = """
    MEDICAL KNOWLEDGE BASE - BASIC INFORMATION

    Cardiovascular Disease Overview:
    Cardiovascular disease (CVD) refers to conditions that involve narrowed or blocked blood vessel
    that can lead to heart attack, chest pain, or stroke. CVD is the leading cause of death globall

    Common symptoms include chest pain, shortness of breath, pain in neck, jaw, throat, upper abdom
    or back, and pain or numbness in legs or arms.

    Risk factors include high blood pressure, high cholesterol, diabetes, smoking, obesity,
    physical inactivity, and family history.

    Treatment approaches include lifestyle modifications, medications (statins, ACE inhibitors,
    beta-blockers), and surgical interventions when necessary.

    Diabetes Management:
    Type 2 diabetes is a chronic condition affecting blood sugar regulation. Management involves
    blood glucose monitoring, medication adherence, dietary modifications, and regular exercise.

    HbA1c targets are typically below 7% for most adults. Common medications include metformin,
    sulfonylureas, and insulin therapy when needed.

    Respiratory Health:
    Asthma is a chronic respiratory condition characterized by airway inflammation and bronchospasm
    Treatment includes bronchodilators, inhaled corticosteroids, and trigger avoidance.

    COPD (Chronic Obstructive Pulmonary Disease) management focuses on bronchodilators,
    pulmonary rehabilitation, and smoking cessation.

    Infectious Diseases:
    Antibiotic resistance is a growing concern. Proper antibiotic stewardship involves using
    appropriate antibiotics for confirmed bacterial infections and completing prescribed courses.

    Viral infections typically do not require antibiotics and are managed with supportive care.

    Preventive Care:
    Regular screenings include mammograms, colonoscopies, blood pressure checks, and cholesterol te
    Vaccinations remain crucial for disease prevention across all age groups.
    """

    # Download and process PDFs
    medical_texts = []
    successful_downloads = 0

    print("Collecting medical PDF sources...")

    for pdf_info in MEDICAL_PDFS:
        print(f"\n Attempting to download: {pdf_info['title']}")

        # Try main URL first
        pdf_content = download_pdf(pdf_info['url'], pdf_info['title'])
```

```
        # Try fallback if main fails
        if pdf_content is None and 'fallback_url' in pdf_info:
            print(f"Trying fallback URL...")
            pdf_content = download_pdf(pdf_info['fallback_url'], pdf_info['title'])

        if pdf_content:
            text = extract_text_from_pdf(pdf_content)
            if text and len(text.strip()) > 100:  # Minimum text threshold
                medical_texts.append({
                    'title': pdf_info['title'],
                    'content': text,
                    'source': pdf_info['url']
                })
                successful_downloads += 1
                print(f"Successfully processed: {pdf_info['title']} ({len(text)} characters)")
            else:
                print(f"Insufficient text extracted from {pdf_info['title']}")
        else:
            print(f"Failed to download {pdf_info['title']}")

    # Add fallback medical text if no PDFs were successfully downloaded
    if successful_downloads == 0:
        print("\n No PDFs downloaded successfully. Using fallback medical knowledge base.")
        medical_texts.append({
            'title': 'Fallback Medical Knowledge Base',
            'content': FALLBACK_MEDICAL_TEXT,
            'source': 'Built-in medical knowledge'
        })
        successful_downloads = 1

    print(f"\n Total medical sources collected: {len(medical_texts)}")
    total_chars = sum(len(text['content']) for text in medical_texts)
    print(f" Total text characters: {total_chars:,}")
```

```
 Collecting medical PDF sources...

  Attempting to download: WHO Guidelines for Clinical Management
 Failed to download WHO Guidelines for Clinical Management: 404 Client Error: not found for
 Trying fallback URL...
 Failed to download WHO Guidelines for Clinical Management: 403 Client Error: Forbidden for
 Failed to download WHO Guidelines for Clinical Management

  Attempting to download: CDC Antiviral Guidelines
 Failed to download CDC Antiviral Guidelines: 404 Client Error: Not Found for url: https://w
 Trying fallback URL...
 Failed to download CDC Antiviral Guidelines: 404 Client Error: Not Found for url: https://w
 Failed to download CDC Antiviral Guidelines

  Attempting to download: AHA Cardiovascular Disease Guide
 PDF extraction failed: EOF marker not found
 Insufficient text extracted from AHA Cardiovascular Disease Guide

  No PDFs downloaded successfully. Using fallback medical knowledge base.

  Total medical sources collected: 1
  Total text characters: 1,879
```

RAG System Setup with FAISS

```
import re
from transformers import AutoTokenizer as CE_Tok, AutoModelForSequenceClassification as CE_Mod
```

```
# 1. Regex sentence splitter
def chunk_text_semantic(text: str, max_tokens: int = 200):
    # Split on end-of-sentence punctuation
    sentences = re.split(r'(?<=[\.!?])\s+', text)
    chunks, current = [], []
    for s in sentences:
        candidate = " ".join(current + [s])
        if len(candidate.split()) <= max_tokens:
            current.append(s)
        else:
            if current:
                chunks.append(" ".join(current))
            current = [s]
    if current:
        chunks.append(" ".join(current))
    return chunks

# 2. Rebuilding semantic chunks & metadata
print("\n Processing medical texts into semantic chunks...")
all_chunks, chunk_metadata = [], []
for doc in medical_texts:
    sem_chunks = chunk_text_semantic(doc['content'])
    for i, chunk in enumerate(sem_chunks):
        all_chunks.append(chunk)
        chunk_metadata.append({
            'title': doc['title'],
            'source': doc['source'],
            'chunk_id': i
        })
print(f" Total semantic chunks: {len(all_chunks)}")

# 3. Embeddings & FAISS index
print("Creating embeddings...")
embeddings = embedding_model.encode(all_chunks, show_progress_bar=True)
index = faiss.IndexFlatIP(embeddings.shape[1])
index.add(embeddings.astype('float32'))
print(f"FAISS index with {index.ntotal} vectors")

# 4. Cross-encoder for reranking
ce_tokenizer = CE_Tok.from_pretrained("cross-encoder/ms-marco-MiniLM-L-6-v2")
ce_model     = CE_Mod.from_pretrained("cross-encoder/ms-marco-MiniLM-L-6-v2").to(model.device)

def rerank_candidates(query: str, cands: list, top_k: int = 3):
    pairs = [(query, c['chunk']) for c in cands]
    inputs = ce_tokenizer(pairs, return_tensors='pt', truncation=True, padding=True).to(model.c
    scores = ce_model(**inputs).logits.squeeze(-1).tolist()
    ranked = sorted(zip(cands, scores), key=lambda x: x[1], reverse=True)
    return [c for c,_ in ranked[:top_k]]

def retrieve_relevant_chunks(query: str, base_k: int = 10):
    # initial FAISS retrieval
    q_emb = embedding_model.encode([query])
    scores, idxs = index.search(q_emb.astype('float32'), base_k)
    cands = []
    for score, idx in zip(scores[0], idxs[0]):
        if score >= 0.1:
            cands.append({
                'chunk': all_chunks[idx],
                'metadata': chunk_metadata[idx],
                'score': float(score)
            })
    # rerank top candidates
    top = rerank_candidates(query, cands, top_k=3)
```

```
    if not top:
        top = [{
            'chunk': FALLBACK_MEDICAL_TEXT,
            'metadata': {'title':'Fallback Knowledge Base','source':'builtin'},
            'score': 1.0
        }]
    return top
```

```
 Processing medical texts into semantic chunks...
 Total semantic chunks: 2
Creating embeddings...
```

Batches: 100% 1/1 [00:00<00:00, 9.27it/s]

```
FAISS index with 2 vectors
```

Medical Chatbot with RAG Integration

```
from jinja2 import Template

PROMPT_TEMPLATE = """
You are a medical researcher providing evidence-based answers.

Context:
{% for src in sources %}
- {{src.metadata.title}}: “{{src.chunk[:150]}}…” (score: {{src.score:.2f}})
{% endfor %}

Question: {{question}}

Please answer in this format:
1. **Summary** (2-3 sentences)
2. **Detailed Explanation** (bullet points)
3. **References** (list source titles)

Answer:
"""

def generate_medical_response(query: str, history: list):
    # 1) Short-term memory
    mem = "".join(f"User: {q}\nAssistant: {a}\n" for q,a in history[-3:])
    # 2) Retrieval & structural prompt
    chunks = retrieve_relevant_chunks(query)
    prompt_body = Template(PROMPT_TEMPLATE).render(sources=chunks, question=query)
    full_prompt = mem + prompt_body

    # 3) Outline pass
    inputs = tokenizer(full_prompt, return_tensors="pt", truncation=True, max_length=1024).to(m
    outline_ids = model.generate(
        **inputs,
        max_new_tokens=100,
        temperature=0.3,
        top_k=50,
        top_p=0.9,
        repetition_penalty=1.2,
        no_repeat_ngram_size=3,
        early_stopping=True
    )
    outline = tokenizer.decode(outline_ids[0], skip_special_tokens=True)

    # 4) Expansion pass
```

```
    exp_prompt = full_prompt + "\n\nOutline:\n" + outline + "\n\nFull Answer:"
    inputs2 = tokenizer(exp_prompt, return_tensors="pt", truncation=True, max_length=1024).to(m
    out_ids = model.generate(
        **inputs2,
        max_new_tokens=200,
        temperature=0.3,
        top_k=50,
        top_p=0.9,
        repetition_penalty=1.2,
        no_repeat_ngram_size=3,
        early_stopping=True
    )
    answer = tokenizer.decode(out_ids[0], skip_special_tokens=True).strip()
    return answer, chunks
```

Gradio Interface

```
import gradio as gr

def chatbot_interface(message: str, history: list) -> tuple:
    # Add medical disclaimer if not present
    disclaimer = ""
    if "medical advice" not in message.lower():
        disclaimer = "\n\n⚠️ **Medical Disclaimer**: This information is for educational purpos

    try:
        # Generate response with RAG (must accept history if your backend uses memory)
        response, sources = generate_medical_response(message, history)
        # Add source citations
        source_info = "\n\n **Sources consulted:**\n"
        for i, source in enumerate(sources[:2], 1):  # Top 2 sources
            source_info += f"{i}. {source['metadata']['title']} (Score: {source['score']:.3f})\
        full_response = response + disclaimer + source_info
        # Update history
        history.append([message, full_response])
        return history, ""
    except Exception as e:
        error_response = f"Sorry, I encountered an error: {str(e)}\n\nPlease try rephrasing you
        history.append([message, error_response])
        return history, ""

print(" Launching Gradio interface...")

with gr.Blocks() as iface:
    gr.Markdown("# 🩺 Medical Chatbot with RAG")
    gr.Markdown("*Powered by Camel-5B and medical knowledge retrieval*")

    chatbot = gr.Chatbot(height=400)

    with gr.Row():
        msg = gr.Textbox(
            label="Ask a medical question",
            placeholder="e.g., What are the symptoms of diabetes?",
            lines=2
        )
        send_btn = gr.Button("Send")
    clear = gr.Button("Clear Conversation")

    # Setup event handlers
    def respond_and_clear(message, history):
```

```
            return chatbot_interface(message, history)

        send_btn.click(respond_and_clear, [msg, chatbot], [chatbot, msg])
        msg.submit(respond_and_clear, [msg, chatbot], [chatbot, msg])
        clear.click(lambda: [], None, chatbot)

        gr.Examples(
            examples=[
                "What are the symptoms of cardiovascular disease?",
                "How is diabetes managed?",
                "What are the risk factors for heart disease?",
                "Explain asthma treatment options",
                "What preventive care screenings are recommended?"
            ],
            inputs=msg
        )

        gr.Markdown("---")
        gr.Markdown(f"**System Info:** {len(medical_texts)} medical sources • {len(all_chunks)} knc

    iface.launch(share=True, debug=True)
```

```
Launching Gradio interface...
Colab notebook detected. This cell will run indefinitely so that you can see errors and log
* Running on public URL: https://c0606a5ef83785f5c0.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio
```

No interface is running right now

```
Keyboard interruption in main thread... closing server.
Killing tunnel 127.0.0.1:7862 <> https://c0606a5ef83785f5c0.gradio.live
```