UNIVERSIDADE FEDERAL DE SERGIPE

DISCIPLINA:

PROCC0111 – TÓPICOS AVANÇADOS EM ENGENHARIA DE SOFTWARE E SISTEMAS
DE INFORMAÇÃO I – ENGENHARIA DE PROMPTS

ALUNO:

MARCELO MOREIRA WEST

PROFESSOR:

DR. GLAUCO DE FIGUEIREDO CARNEIRO

ATIVIDADE 3

ATIVIDADE	**IMPLEMENTAÇÃO DE PARTE DO CÓDIGO COMO UM AGENTE IA PARA RECUPERAÇÃO EFICIENTE DAS INFORMAÇÕES MÉDICAS.**
PROFESSOR	**Glauco de Figueiredo Carneiro**
ALUNO	**Marcelo Moreira West**

1. CÓDIGO DO CHATBOT

```
# Instalar bibliotecas (executar no Google Colab)
!pip install --upgrade openai gradio PyMuPDF --quiet
!pip install --upgrade openai gradio PyMuPDF faiss-cpu --quiet

import os
import openai
import gradio as gr
import fitz # PyMuPDF
import faiss
import numpy as np

class MedicalAssistantAgent:
      """
      Um agente médico que utiliza GPT-4o e RAG (Retrieval-Augmented
Generation) para responder perguntas
      com base em informações extraídas de PDFs.
      """

def __init__(self, caminho_pdfs):
      """
      Inicializa o agente, configurando a chave da API da OpenAI e o
caminho para os PDFs.
      """
      self.api_key = os.getenv("OPENAI_API_KEY")
      if not self.api_key:
            raise ValueError("A chave da API não foi encontrada. Defina
a variável de ambiente                          'OPENAI_API_KEY'.")
      openai.api_key = self.api_key
      self.caminho_pdfs = caminho_pdfs

def extrair_texto_pdf(self, caminho_pdf):
      """
      Extrai texto de um arquivo PDF.
      """
      texto = ""
      with fitz.open(caminho_pdf) as doc:
      for pagina in doc:
            texto += pagina.get_text()
      return texto

def dividir_em_chunks(self, texto, tamanho=500, sobreposicao=100):
      """
      Divide o texto em chunks de tamanho fixo com sobreposição.
      """
      palavras = texto.split()
      chunks = []
      for i in range(0, len(palavras), tamanho - sobreposicao):
```

```
            chunk = " ".join(palavras[i:i + tamanho])
            if chunk:
                  chunks.append(chunk)
      return chunks

def gerar_embedding(self, texto):
      """
      Gera embeddings para um texto usando a API da OpenAI.
      """
      response = openai.Embedding.create(
      input=texto,
      model="text-embedding-ada-002"
      )
      return np.array(response['data'][0]['embedding'],
dtype=np.float32)

def indexar_pdf(self, caminho_pdf):
      """
      Indexa os chunks de um PDF em um banco vetorial FAISS.
      """
      texto = self.extrair_texto_pdf(caminho_pdf)
      chunks = self.dividir_em_chunks(texto)
      embeddings = [self.gerar_embedding(chunk) for chunk in chunks]

      index = faiss.IndexFlatL2(len(embeddings[0]))
      index.add(np.array(embeddings))

      return index, chunks


def recuperar_chunks(self, pergunta, index, chunks, k=3):
      """
      Recupera os k chunks mais relevantes para uma pergunta.
      """
      pergunta_emb = self.gerar_embedding(pergunta)
      _, indices = index.search(np.array([pergunta_emb]), k)
      return [chunks[i] for i in indices[0]]

def consultar_gpt_rag(self, pergunta, nome_pdf):
      """
      Consulta o modelo GPT-4o com informações recuperadas de um PDF.
      """
      caminho_pdf = os.path.join(self.caminho_pdfs, nome_pdf)
      index, chunks = self.indexar_pdf(caminho_pdf)
      trechos_relevantes = self.recuperar_chunks(pergunta, index,
chunks)

      contexto = "\n---\n".join(trechos_relevantes)
```

```
        prompt = f"""
            Você é um assistente médico especializado em doenças
respiratórias.

            Use as informações abaixo extraídas de documentos para
responder a pergunta do usuário:

        {contexto}

        Pergunta: {pergunta}
        """

        try:
            response = openai.ChatCompletion.create(
            model="gpt-4",
            messages=[
            {"role": "system", "content": "Você é um assistente médico
confiável, que usa informações de                contexto com
responsabilidade."},
            {"role": "user", "content": prompt}
            ],
        temperature=0.5,
        max_tokens=1000
        )
        return response['choices'][0]['message']['content']
        except Exception as e:
        return f"Erro: {str(e)}"

def listar_pdfs(self):
        """
        Lista os arquivos PDF disponíveis no diretório configurado.
        """
        return [f for f in os.listdir(self.caminho_pdfs) if
f.endswith(".pdf")]

def iniciar_interface(self):
        """
        Inicia a interface Gradio para interação com o agente.
        """
        interface = gr.Interface(
        fn=self.consultar_gpt_rag,
        inputs=[
        gr.Textbox(lines=4, label="Digite sua pergunta"),
        gr.Dropdown(choices=self.listar_pdfs(), label="Selecione um PDF")
        ],
        outputs="text",
        title="RAG Médico com GPT-4o e PDF",
```

```
        description="Faça perguntas sobre doenças respiratórias. O modelo buscará as respostas mais relevantes   nos arquivos PDF usando RAG (Retrieval-Augmented Generation)."
        )
        interface.launch(share=True)

# Executa o agente
if __name__ == "__main__":
        try:
                caminho_pdfs = "/content/" # Substitua pelo caminho onde os PDFs estão armazenados
                agente = MedicalAssistantAgent(caminho_pdfs)
                agente.iniciar_interface()
        except ValueError as e:
                print(e)
```

2. DESCRIÇÃO DA IMPLEMENTAÇÃO

2.1. Encapsulamento em uma classe:

Foi criada a classe 'MedicalAssistantAgent' com os Métodos: 'extrair_texto_pdf', 'dividir_em_chunks', 'gerar_embedding', 'indexar_pdf', 'recuperar_chunks' e 'consultar_gpt_rag'.

2.2. Interface Gradio:

A interface Gradio foi integrada no método 'iniciar_interface' para permitir interação com o agente.

2.3. Modularidade:

O código é modular e reutilizável, permitindo fácil manutenção e expansão.

2.4. Configuração do caminho dos PDFs:

O caminho dos PDFs é configurado no construtor da classe e usado em todos os métodos relacionados.

3. INSTRUÇÕES DE EXECUÇÃO

3.1. Certifique-se de que os PDFs estão no diretório configurado (`/content/` ou outro caminho que você definir).

3.2. Instale as dependências necessárias:

pip install openai gradio pymupdf faiss-cpu numpy

3.3. Configure a variável de ambiente ‘OPENAI_API_KEY’ com sua chave da API:

export OPENAI_API_KEY="sua-chave-aqui"

3.4. Execute o programa:

python3 medical_assistant_agent.py

4. RESULTADO DA EXECUÇÃO:

```
(venv) marcelo-west@serverwest:~/UFBA/agent$ python3 chatbot_agent.py
Bem-vindo ao chatbot GPT-4o! Digite 'sair' para encerrar.
Você: quem é você?
Chatbot: Eu sou um modelo de linguagem desenvolvido pela OpenAI, conhecido como ChatGPT. Estou aqui para ajudar a
 responder perguntas, fornecer informações e interagir com você sobre uma variedade de tópicos. Como posso ajudar
 hoje?
Você: quais são os sintomas da covid-19?
Chatbot: Os sintomas da COVID-19 podem variar amplamente entre os indivíduos, mas os mais comuns incluem:

1. **Febre** ou calafrios
2. **Tosse** seca
3. **Fadiga**
4. **Dificuldade para respirar** ou falta de ar
5. **Dores musculares** ou corporais
6. **Dor de cabeça**
7. **Perda de paladar ou olfato**
8. **Dor de garganta**
9. **Congestão** ou coriza
10. **Náusea** ou vômito
11. **Diarreia**

Esses sintomas podem aparecer de 2 a 14 dias após a exposição ao vírus. Além disso, algumas pessoas infectadas co
m o coronavírus podem ser assintomáticas, ou seja, não apresentam sintomas, mas ainda podem transmitir o vírus pa
ra outras pessoas.

É importante observar que a gravidade dos sintomas pode variar de leve a grave, e algumas pessoas, especialmente
aquelas com condições médicas preexistentes ou idosos, podem ter um risco maior de desenvolver complicações grave
s. Se você ou alguém que você conhece estiver apresentando sintomas graves, como dificuldade respiratória, dor ou
 pressão persistente no peito, confusão, incapacidade de acordar ou permanecer acordado, ou coloração azulada nos
 lábios ou rosto, deve procurar atendimento médico de emergência imediatamente.
```