**Documentação**

**Projeto IAVS: Inteligência Artificial para a Vigilância em Saúde**

**Assistente de IA Local**

Este projeto demonstra como construir um assistente de IA que responde a perguntas com base em um conjunto de documentos fornecidos, tudo executado localmente no seu computador. Utilizamos o **Streamlit** para a interface, o **Ollama** para rodar o modelo de IA e o **LangChain** para orchestrar o processo de busca e resposta, no sistema Windows.

**Pré-requisitos**

Para começar, certifique-se de que você tem as seguintes ferramentas instaladas:

* **VS Code**: Um editor de código. Faça o download em [code.visualstudio.com](https://code.visualstudio.com/).
* **Python 3.8+**: A linguagem de programação do projeto. Baixe a versão mais recente em [python.org](https://www.python.org/downloads/).
* **Ollama**: A ferramenta para rodar o LLM localmente. Faça o download e instale em [ollama.com](https://ollama.com/).

**Passo 1: Instalação e Configuração Inicial**

1. **Instale o Ollama e o Modelo**:
   * Abra o **Terminal** (ou PowerShell no Windows).
   * Instale o Ollama a partir do site oficial, seguindo as instruções para o seu sistema operacional.
   * Após a instalação, verifique se o Ollama está funcionando digitando ollama -v no terminal.
   * Baixe o modelo de linguagem llama2 (ou outro modelo de sua preferência) com o seguinte comando:

Bash

ollama pull llama2

* + **Verifique se o Ollama está em execução em segundo plano**: Ele deve estar rodando como um serviço no seu computador. Você pode verificar isso com os comandos:
    - **Windows**: tasklist | findstr "ollama.exe"
    - Se não estiver rodando, reinicie seu computador ou procure por "Ollama" nos seus aplicativos e inicie-o.

1. **Crie a Estrutura do Projeto**:
   * Crie nova pasta em um local de sua preferência, por exemplo, assistente-llm-local. No caso estudado, usamos a pasta “IAVS\_PROJETO-main”.
   * Dentro dessa pasta, crie uma subpasta chamada documents. Esta pasta irá armazenar todos os seus arquivos de texto (.pdf ou .txt) que servirão como base de conhecimento para o assistente.
   * Crie um arquivo Python chamado app.py na raiz da pasta assistente-llm-local.
2. **Configure o Ambiente Virtual**:
   * Abra o **VS Code**.
   * No VS Code, vá em **File > Open Folder...** (ou Arquivo > Abrir Pasta...) e selecione a pasta assistente-llm-local que você criou.
   * Abra o **Terminal Integrado** do VS Code. Você pode fazer isso indo em **Terminal > New Terminal** (ou Terminal > Novo Terminal) ou usando o atalho Ctrl + (crase).
   * Crie um ambiente virtual para o seu projeto para isolar as dependências:

Bash

python -m venv venv

* + Ative o ambiente virtual:

Bash

.\venv\Scripts\activate

**Passo 2: Instalação das Bibliotecas**

Com o ambiente virtual ativado no terminal do VS Code, instale as bibliotecas Python necessárias para o projeto. Para manter as dependências organizadas, é uma boa prática criar um arquivo requirements.txt.

1. Crie um arquivo chamado requirements.txt na raiz da sua pasta assistente-llm-local (no mesmo nível de app.py e da pasta documents).
2. Copie e cole as seguintes linhas no arquivo requirements.txt:

Plaintext

streamlit

langchain

langchain-ollama

langchain-community

chromadb

pypdf

sentence-transformers

1. Salve o arquivo requirements.txt.
2. No terminal do VS Code (com o ambiente virtual ativado), execute o comando para instalar todas as bibliotecas listadas:

Bash

pip install -r requirements.txt

* + Este processo pode levar alguns minutos, pois algumas bibliotecas são grandes, especialmente o sentence-transformers que baixa o modelo de embedding.

**Passo 3: Implementação do Código Python (app.py)**

Abra o arquivo app.py que você criou no Passo 1.2 e cole o código completo abaixo. Este código contém a lógica para carregar seus documentos, criar o banco de dados de conhecimento, interagir com o modelo Ollama e apresentar a interface via Streamlit.

Python

import streamlit as st # Para criar a interface de usuário web.

import os # Para interagir com o sistema operacional, como listar arquivos em uma pasta.

from langchain\_ollama import OllamaLLM # Conecta ao modelo de linguagem rodando via Ollama.

from langchain.document\_loaders import PyPDFLoader, TextLoader # Carrega arquivos PDF e TXT.

from langchain.text\_splitter import RecursiveCharacterTextSplitter # Divide textos longos em pedaços (chunks).

from langchain.vectorstores import Chroma # Banco de dados vetorial para armazenar embeddings.

from langchain.embeddings import SentenceTransformerEmbeddings # Gera representações numéricas (embeddings) do texto.

from langchain.chains import RetrievalQA # Orquestra a busca nos documentos e a geração da resposta.

from langchain.prompts import PromptTemplate # Para criar templates de prompt personalizados para o LLM.

# Função para carregar e processar os documentos da pasta 'documents'

def processar\_documentos(pasta\_documentos):

docs = [] # Lista para armazenar os documentos carregados.

# Verifica se a pasta de documentos existe.

if not os.path.exists(pasta\_documentos):

st.error(f"A pasta '{pasta\_documentos}' não foi encontrada. Certifique-se de que ela está no mesmo diretório do arquivo app.py.")

return None # Retorna None se a pasta não existir para evitar erros.

# Itera sobre cada arquivo na pasta de documentos.

for arquivo in os.listdir(pasta\_documentos):

caminho\_arquivo = os.path.join(pasta\_documentos, arquivo) # Constrói o caminho completo para o arquivo.

try:

# Carrega arquivos PDF usando PyPDFLoader.

if arquivo.endswith(".pdf"):

loader = PyPDFLoader(caminho\_arquivo)

docs.extend(loader.load()) # Adiciona o conteúdo do PDF à lista de documentos.

st.info(f"PDF '{arquivo}' carregado com sucesso.")

# Carrega arquivos TXT usando TextLoader.

elif arquivo.endswith(".txt"):

loader = TextLoader(caminho\_arquivo, encoding='utf-8') # Assume codificação UTF-8 para TXT.

docs.extend(loader.load()) # Adiciona o conteúdo do TXT à lista de documentos.

st.info(f"TXT '{arquivo}' carregado com sucesso.")

except Exception as e:

# Em caso de erro ao carregar um arquivo, exibe um aviso.

st.warning(f"Não foi possível carregar o arquivo '{arquivo}': {e}. Por favor, verifique o formato e o conteúdo.")

# Verifica se algum documento foi carregado.

if not docs:

st.error("Nenhum documento foi carregado. A lista de documentos está vazia. Certifique-se de que há arquivos .pdf ou .txt válidos na pasta 'documents'.")

return None # Retorna None se nenhum documento foi carregado.

# Divide os documentos em pedaços (chunks) menores.

# chunk\_size: tamanho máximo de cada pedaço de texto.

# chunk\_overlap: caracteres de sobreposição entre os pedaços para manter o contexto.

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=200)

docs\_chunks = text\_splitter.split\_documents(docs)

# Cria embeddings (representações numéricas) para cada pedaço de texto.

embeddings = SentenceTransformerEmbeddings(model\_name="all-MiniLM-L6-v2")

# Armazena os chunks e seus embeddings no banco de dados vetorial ChromaDB.

db = Chroma.from\_documents(docs\_chunks, embeddings)

return db # Retorna o banco de dados Chroma configurado.

# --- Configuração da Interface Streamlit e Lógica Principal ---

st.title("Assistente de IA Local com Ollama e Streamlit") # Título da aplicação na interface web.

st.write("Faça perguntas com base nos documentos carregados na pasta `documents`.") # Descrição.

# Carrega e processa os documentos apenas uma vez, na primeira execução do aplicativo.

# O 'st.session\_state' garante que o banco de dados 'db' seja persistido entre as interações do usuário.

if "db" not in st.session\_state:

with st.spinner("Processando documentos... Esta etapa pode levar alguns minutos na primeira execução."):

st.session\_state.db = processar\_documentos("documents")

if st.session\_state.db:

st.success("Documentos processados! Você já pode fazer perguntas.")

else:

st.error("Falha ao processar documentos. Verifique as mensagens acima para detalhes.")

# Continua a execução apenas se o banco de dados 'db' foi carregado com sucesso.

if "db" in st.session\_state and st.session\_state.db is not None:

# Inicializa o modelo de linguagem (LLM) do Ollama.

llm = OllamaLLM(model="llama2")

# Define o template de prompt para instruir o LLM.

# Ele inclui o contexto dos documentos e instrui o LLM a responder em português.

template = """Use as seguintes partes do contexto para responder à pergunta do usuário.

Se você não souber a resposta, apenas diga que não a conhece. Não tente inventar uma resposta.

Responda em português.

Contexto: {context}

Pergunta: {question}

Resposta em português:"""

# Cria uma instância de PromptTemplate a partir do template definido.

QA\_CHAIN\_PROMPT = PromptTemplate.from\_template(template)

# Configura a cadeia de Retrieval-Augmented Generation (RAG).

# llm: O modelo de linguagem a ser usado.

# chain\_type="stuff": Método para passar todos os documentos relevantes para o LLM.

# retriever: Componente que busca os documentos mais relevantes do ChromaDB.

# chain\_type\_kwargs: Argumentos adicionais para a cadeia, incluindo o prompt personalizado.

qa\_chain = RetrievalQA.from\_chain\_type(

llm=llm,

chain\_type="stuff",

retriever=st.session\_state.db.as\_retriever(),

chain\_type\_kwargs={"prompt": QA\_CHAIN\_PROMPT}

)

# Interface de usuário para a pergunta do usuário.

pergunta\_usuario = st.text\_input("Sua pergunta:", "")

# Processa a pergunta se o usuário digitou algo.

if pergunta\_usuario:

with st.spinner("Gerando resposta..."): # Exibe um spinner enquanto a resposta é gerada.

resposta = qa\_chain.run(pergunta\_usuario) # Executa a cadeia RAG com a pergunta.

st.write("---") # Linha divisória na interface.

st.write(f"\*\*Resposta:\*\* {resposta}") # Exibe a resposta formatada.

else:

# Mensagem exibida se o banco de dados não foi carregado com sucesso.

st.info("Por favor, verifique os erros acima e os arquivos na pasta 'documents' para que o assistente possa começar.")

**Passo 4: Adicione seus Documentos**

1. **Prepare seus Arquivos**: Coloque todos os seus documentos (.pdf e/ou .txt) na pasta documents que você criou no Passo 1.2.
   * Certifique-se de que os arquivos .txt estão salvos com codificação **UTF-8** para evitar problemas de leitura.
   * Se for um .pdf, verifique se ele possui uma camada de texto selecionável (não é uma imagem escaneada).

**Passo 5: Execute o Projeto**

1. **Certifique-se de que o Ollama está ativo**: Confirme que o servidor Ollama está rodando em segundo plano no seu computador.
2. **Abra o Terminal no VS Code**: Volte para o terminal integrado do VS Code onde o ambiente virtual está ativado.
3. **Inicie a Aplicação Streamlit**: Execute o comando a seguir:

Bash

streamlit run app.py

1. **Acesse o Aplicativo**: Seu navegador padrão abrirá automaticamente uma nova guia com o aplicativo Streamlit.
   * Na primeira vez que você executar, o Streamlit exibirá um spinner (Processando documentos...) enquanto a função processar\_documentos carrega, divide e cria os embeddings dos seus arquivos. Isso pode levar alguns minutos, dependendo da quantidade e tamanho dos documentos.
   * Após o processamento, uma mensagem de sucesso será exibida.

**Passo 6: Interaja com o Assistente**

1. No campo "Sua pergunta:", digite suas perguntas relacionadas ao conteúdo dos documentos que você carregou.
2. Pressione Enter.
3. O assistente de IA buscará as informações relevantes nos seus documentos e gerará uma resposta em português.

**Solução de Problemas Comuns**

* **Expected Embeddings to be non-empty list or numpy array, got [] in upsert.**: Este erro significa que nenhum documento foi carregado ou processado com sucesso.
  + Verifique se a pasta documents existe e está no diretório correto.
  + Certifique-se de que há arquivos .pdf ou .txt na pasta documents.
  + Verifique se os arquivos .txt estão em **UTF-8**.
  + Certifique-se de que os arquivos .pdf contêm texto selecionável.
  + As mensagens de st.error e st.warning no código podem ajudar a identificar o arquivo problemático.
* **O Ollama não está respondendo ou não carrega o modelo**:
  + Confirme se o Ollama está rodando em segundo plano.
  + Verifique se você baixou o modelo llama2 (ou o modelo que especificou no código) com ollama pull llama2.
  + Tente reiniciar o Ollama ou o seu computador.
* **Respostas em inglês (ou outro idioma)**:
  + Certifique-se de que o PromptTemplate foi adicionado corretamente ao seu código, instruindo o LLM a responder em português.
  + Salve o arquivo app.py para que o Streamlit recarregue as mudanças.
* **Erros de biblioteca (ModuleNotFoundError)**:
  + Certifique-se de que você ativou o ambiente virtual ((venv)) no terminal do VS Code antes de rodar pip install -r requirements.txt.
  + Reinstale as bibliotecas usando pip install -r requirements.txt.

Com esta documentação, você tem todas as informações necessárias para replicar e entender o projeto!