**1. Introdução**

O objetivo deste relatório é documentar os passos necessários para a criação de um chatbot utilizando as bibliotecas LlamaIndex, Chroma DB e modelos de embeddings do HuggingFace. O processo inclui a instalação das bibliotecas, leitura e processamento de documentos, criação de embeddings e configuração do banco de dados para armazenar e consultar as informações necessárias para a interação com o chatbot.

**2. Passos para Criar o Chatbot**

**2.1 Instalação das Bibliotecas Necessárias**

Primeiro, é necessário instalar as bibliotecas que facilitam a leitura de documentos, o processamento de dados e a construção do índice:

bash

Copiar

!pip install -q llama-index

!pip install -q llama-index-embeddings-huggingface

!pip install -q llama-index-vector-stores-chroma

!pip install -q llama-index-llms-groq

Essas bibliotecas fornecem as ferramentas essenciais para lidar com documentos, integrar embeddings e configurar o banco de dados.

**2.2 Leitura de Documentos com LlamaIndex**

O próximo passo é importar e ler os documentos a partir de uma pasta específica. Usamos a classe SimpleDirectoryReader para ler os arquivos e carregá-los em uma variável.

python

Copiar

from llama\_index.core import SimpleDirectoryReader

documentos = SimpleDirectoryReader(input\_dir='documentos')

# Visualizando os arquivos carregados

documentos.input\_files

docs = documentos.load\_data()

Esses documentos são armazenados na variável docs e podem ser manipulados conforme necessário.

**2.3 Pré-processamento de Dados: Segmentação de Sentenças**

A etapa seguinte envolve a divisão dos documentos em partes menores (nós), para facilitar o processamento e a criação de embeddings. Utilizamos o SentenceSplitter para dividir o texto em segmentos de tamanho adequado.

python

Copiar

from llama\_index.core.node\_parser import SentenceSplitter

node\_parser = SentenceSplitter(chunk\_size=1200)

# Dividindo os documentos em nós

nodes = node\_parser.get\_nodes\_from\_documents(docs, show\_progress=True)

Esses nós representam trechos dos documentos e são a base para a criação do índice.

**2.4 Integração com Embeddings do HuggingFace**

Para criar embeddings dos nós, utilizamos o modelo multilingual-e5-large do HuggingFace. A classe ChromaEmbeddingWrapper é criada para integrar os embeddings com o banco de dados Chroma.

python

Copiar

from llama\_index.embeddings.huggingface import HuggingFaceEmbedding

class ChromaEmbeddingWrapper:

def \_\_init\_\_(self, model\_name: str):

self.model = HuggingFaceEmbedding(model\_name=model\_name)

def \_\_call\_\_(self, input):

return self.model.embed(input)

embed\_model\_chroma = ChromaEmbeddingWrapper(model\_name='intfloat/multilingual-e5-large')

**2.5 Configuração do Banco de Dados com Chroma DB**

Agora, configuramos o banco de dados Chroma para armazenar os embeddings. Usamos o cliente do Chroma e criamos uma coleção para armazenar os dados.

python

Copiar

import chromadb

db = chromadb.PersistentClient(path='./chroma\_db')

chroma\_client = db

collection\_name = 'docs\_info'

try:

chroma\_collection = chroma\_client.get\_or\_create\_collection(name=collection\_name)

embadding\_function = embed\_model\_chroma

except Exception as e:

print(f'Erro ao criar coleção {e}')

**2.6 Criação do Índice com LlamaIndex**

Após a criação do banco de dados e o processamento dos dados, criamos um índice vetorial usando os nós gerados e o modelo de embeddings. O VectorStoreIndex é usado para armazenar os dados processados.

python

Copiar

from llama\_index.vector\_stores.chroma import ChromaVectorStore

from llama\_index.core import StorageContext

vector\_store = ChromaVectorStore(chroma\_collection=chroma\_collection)

storage\_context = StorageContext.from\_defaults(vector\_store=vector\_store)

embed\_model = HuggingFaceEmbedding(model\_name='intfloat/multilingual-e5-large')

index = VectorStoreIndex(nodes, storage\_context=storage\_context, embed\_model=embed\_model)

**2.7 Carregando o Índice para Uso**

Agora que o índice foi criado, podemos carregá-lo para realizar consultas com base nas informações armazenadas.

python

Copiar

from llama\_index.core import load\_index\_from\_storage

index = load\_index\_from\_storage(storage\_context, embed\_model=embed\_model)

**2.8 Integração com a API da Groq**

Para otimizar o desempenho da consulta, podemos integrar o modelo da Groq, utilizando sua API. A seguir, está a chave da API Groq para fazer a autenticação.

python

Copiar

Groq\_api = 'gsk\_D6qheWgXIaQ5jl3Pu8LNWGdyb3FYJXU0RvNNoIpEKV1NreqLAFnf'

**2.9 Consultas ao Banco de Dados**

Agora, o chatbot pode fazer consultas ao banco de dados Chroma com base no índice carregado. Com isso, o sistema estará pronto para interagir com os usuários, respondendo de acordo com os dados processados e armazenados.

python

Copiar

from llama\_index.llms.groq import Groq

**2.10 Execução do Chatbot**

Finalmente, o chatbot pode ser executado, onde o usuário faz perguntas e o sistema retorna respostas com base nas informações indexadas.

**3. Conclusão**

Neste relatório, abordamos os principais passos para a criação de um chatbot baseado em documentos, utilizando as bibliotecas LlamaIndex, HuggingFace e Chroma DB. O processo envolveu a leitura de documentos, criação de embeddings, armazenamento de dados em um banco de dados vetorial e a configuração de um modelo para consultas rápidas e precisas. Com essa abordagem, é possível construir chatbots poderosos e eficientes para diversas aplicações.