Roteiro de Atividades

Este roteiro foi pensado para que suas habilidades adquiridas ao longo dos primeiros 8 encontros sejam executadas e consolidadas.

Uma grande aplicação para a rotina em diversas áreas é a utilização de modelos de linguagem (LLM) para auxílio em geração de estrutura de relatórios, e-mails, criação de códigos e resolução de problemas relacionados a eles, entre outras atividades.

Pensando nisso, esta atividade será toda desenvolvida utilizando modelo LLM em um contexto exposto por vocês, como possível aplicação. Divirta-se!

**Contexto**

Neste primeiro momento, vamos utilizar um exemplo de um assistente, já completo, que iremos entender cada elemento e sua função. A próxima etapa, feita por vocês, envolverá a construção e adequação deste assistente em um ambiente com API para acesso ao modelo.

**Importar as bibliotecas**

import requests

import json

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

requests: envio de solicitação para a API do modelo (llama)

nltk: Natural Language Toolkit fornece funções de tokenização e remoção de stopwords, como “o”, “a”, “de”, e assim por diante.

**# Baixar stopwords do NLTK**

nltk.download("punkt")

nltk.download("stopwords")

**# URL do Ollama rodando localmente (após ser instalado, esta URL é criada)**

OLLAMA\_URL = "http://localhost:11434/api/generate"

Este bloco de código abaixo é muito importante para esta atividade em específico. A ideia é definir algumas palavras-chave e um fluxo de trabalho relacionado. Após a sua definição, quando uma perguntar for feita ao modelo, via *prompt*, o modelo além de retornar a resposta ele também define a qual fluxo de trabalho a pergunta pertence.

**# Dicionário de palavras-chave e fluxos de trabalho**

fluxos = {

"Atividades e Planos de Corrida": "nadar",

"Atividades e Planos de Pedal": "padalar",

"Atividades e Planos de Natação": "correr",

}

Este bloco pode ser alterado dentro do contexto que achar necessário, basta manter na primeira parte à esquerda “x”: as palavras-chave e na direita :”y” o nome do fluxo correspondente. A estrutura da variável **fluxo** é um dicionário no formato **chave:valor**. Observação: para mais de uma palavra-chave, o código deve ser alterado, no bloco de identificação do fluxo.

Bloco de código para extração das palavras-chave de uma pergunta.

def extrair\_palavras\_chave(texto):

"""Extrai palavras-chave de uma pergunta removendo stopwords."""

tokens = word\_tokenize(texto.lower()) # Tokenização e conversão para minúsculas

stop\_words = set(stopwords.words("portuguese"))

palavras\_chave = [t for t in tokens if t.isalnum() and t not in stop\_words]

return palavras\_chave

Bloco de código para obtenção de resposta do modelo. A pergunta é passada como argumento da função, enviada via URL do Ollama e solicitado ao modelo definido na variável model.

def obter\_resposta\_llama(pergunta):

"""Envia a pergunta para o LLaMA e retorna a resposta."""

payload = {

"model": "llama3",

"prompt": pergunta,

"stream": False

}

resposta = requests.post(OLLAMA\_URL, json=payload)

return resposta.json()["response"] if resposta.status\_code == 200 else "Erro na resposta"

Bloco de código para associação das palavras-chave extraídas aos fluxos de trabalho definidos.

def determinar\_fluxo(pergunta):

"""Determina o fluxo de trabalho com base nas palavras-chave da pergunta."""

palavras\_chave = extrair\_palavras\_chave(pergunta)

for palavra in palavras\_chave:

if palavra in fluxos:

return f"Fluxo identificado: {fluxos[palavra]}"

return "Nenhum fluxo específico identificado."

Bloco de loop principal do Assistente

if \_\_name\_\_ == "\_\_main\_\_":

while True:

pergunta = input("Digite sua pergunta (ou 'sair' para encerrar): ")

if pergunta.lower() == "sair":

break

fluxo = determinar\_fluxo(pergunta)

resposta = obter\_resposta\_llama(pergunta)

print(fluxo)

print("Resposta do Assistente:", resposta)

O código está completo! Para sua execução e verificação, vamos criar um arquivo **python** com ele completo e executar. Para esta atividade, qualquer IDE pode ser usada. Vamos usar o **VisualCode** para criar, analisar e executar.

**- Arquivo Python criado**

**- Validação do código**

**- Execução sem erros**

**- Validação da tarefa diretamente no terminal**

**ATIVIDADE 1**

O assistente visto na tarefa anterior é útil e eficiente. Porém, é acessado localmente apenas por quem executa o código. Para disponibilizar este assistente para sua equipe ou empresa é preciso utilizar uma forma de expor a execução utilizando uma API, centralizando as solicitações.

Utilizar os blocos abaixo, incompletos, para montar seu arquivo **api.py** e colocar em execução. Os blocos que estiverem incompletos devem ser preenchidos corretamente da maneira que achar melhor!

**Bloco 01**

from fastapi import FastAPI, Response

from pydantic import BaseModel

import requests

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from typing import List

import time

from prometheus\_client import Counter, Summary, generate\_latest, CONTENT\_TYPE\_LATEST

**Bloco 02**

nltk.download("punkt")

nltk.download("stopwords")

**Bloco 03**

app = FastAPI()

**Bloco 04**

fluxos = {

"X": ["Y1", "Y2", "Y3”],

Preencha com quantos fluxos forem necessários!

}

**Bloco 05**

OLLAMA\_URL = <http://localhost:11434/api/generate>

*--- CRIE 2 métricas, ao menos:*

PREDICTION\_TIME = #completar...

PREDICTION\_INFERENCE = #completar...

**Bloco 06**

**ATIVIDADE 3: compare o código de extração de palavras-chave e determinação de fluxos abaixo com o que foi feito na primeira atividade, do assistente e identifique as diferenças.**

class Pergunta(BaseModel):

pergunta: str

def extrair\_palavras\_chave(texto: str) -> List[str]:

"""Extrai palavras-chave da pergunta"""

stop\_words = set(stopwords.words("portuguese"))

tokens = word\_tokenize(texto.lower()) # Tokeniza e converte em minúsculas

palavras\_chave = [t for t in tokens if t.isalnum() and t not in stop\_words]

return palavras\_chave

def determinar\_fluxo(palavras\_chave: List[str]) -> str:

"""Identifica o fluxo de trabalho baseado nas palavras-chave"""

for fluxo, palavras in fluxos.items():

# Verifica se alguma palavra-chave do fluxo aparece nas palavras extraídas

if any(palavra in palavras\_chave for palavra in palavras):

return f"{fluxo.capitalize()}"

return "Nenhum fluxo específico identificado."

def obter\_resposta\_llama(pergunta: str) -> str:

"""Obtém a resposta do modelo LLaMA"""

payload = {

"model": "llama3.2",

"prompt": pergunta,

"stream": False

}

resposta = requests.post(OLLAMA\_URL, json=payload)

return resposta.json().get("response", "Erro ao obter resposta")

**Bloco 07**

**O bloco “app.post” está incompleto. É preciso incorporar os comandos para associação das métricas e também de apresentação na tela da métrica, dentro do comando *return*.**

@app.get("/")

def home():

return {"message": "API llama funcionando!"}

@app.get("/metrics")

def get\_metrics():

return Response(generate\_latest(), media\_type=CONTENT\_TYPE\_LATEST)

@app.post("/pergunta/")

async def fazer\_pergunta(pergunta: Pergunta):

"""Processa a pergunta, envia ao LLaMA e retorna a resposta"""

palavras\_chave = extrair\_palavras\_chave(pergunta.pergunta)

fluxo = determinar\_fluxo(palavras\_chave)

# Obter resposta do LLaMA

resposta\_llama = obter\_resposta\_llama(pergunta.pergunta)

return {"fluxo": fluxo, "resposta": resposta\_llama}

Com o código finalizado, execute o comando **uvicorn api:app --host 0.0.0.0 --port 8000** para iniciar a API. Acesse via browser e testa a API.

**ATIVIDADE 3**

Acessar a página web da API é uma tarefa de validação. Quando em produção, a exposição do modelo é útil para incorporar em outras ferramentas ou ambientes. Para ver esta integração, podemos criar uma página WEB em HTML simples.

Vamos obter ajuda de uma IA Generativa (a de sua preferência) e solicitar que forneça um código HTML para integração com API da FastAPI que estamos utilizando. Na página teremos uma caixa de texto pra escreve e enviar o prompt e receber a resposta também.

**Adicionando arquivos para customização da tarefa**

A tarefa de um assistente interno, para respostas customizadas à fluxos internos, pode ser integrada à documentos, por exemplo PDF e TXT, para que a solicitação via prompt seja primeiro consultada nos documentos e, só então, no modelo LLM escolhido.

Para esta tarefa, uma opção é o uso do **RAG – Retrieval-Augmented Generation**. Ele tem dois passos principais, sendo o primeiro a busca nos documentos e o uso do modelo LLM para gerar as respostas. Para esta implementação, os documentos são adicionados a um **diretório** qualquer e este é apresentado ao arquivo **api.py** para consulta com seu caminho em uma variável.

A seguir, pode-se encontrar um código muito parecido com o construído e utilizado nas atividades anteriores. Aqui, a estrutura de busca nos documentos é adicionada.

from fastapi import FastAPI

from pydantic import BaseModel

import requests

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

import os

from langchain.document\_loaders import PyPDFLoader, TextLoader

from langchain.vectorstores import Chroma

from langchain.embeddings import HuggingFaceEmbeddings

from langchain.chains import RetrievalQA

from langchain.prompts import PromptTemplate

from typing import List

nltk.download("punkt")

nltk.download("stopwords")

# Inicializa a aplicação FastAPI

app = FastAPI()

# Dicionário de fluxos de trabalho com múltiplas palavras-chave

fluxos = {

"reunião": ["reunião", "agenda", "marcar encontro", "reuniões"],

"relatório": ["relatório", "gerar relatório", "criar relatório", "relatórios"],

"projeto": ["projeto", "gestão de projetos", "gerenciamento de projetos"],

"financeiro": ["financeiro", "contas", "relatório financeiro", "finanças"],

}

OLLAMA\_URL = "http://localhost:11434/api/generate" # Substitua com a URL do Ollama

**# Pasta onde estão os documentos**

**PASTA\_DOCUMENTOS = "documents"**

**# Carregar e indexar documentos da pasta**

**def carregar\_documentos():**

**documentos = []**

**for arquivo in os.listdir(PASTA\_DOCUMENTOS):**

**caminho\_arquivo = os.path.join(PASTA\_DOCUMENTOS, arquivo)**

**if arquivo.endswith(".pdf"):**

**loader = PyPDFLoader(caminho\_arquivo)**

**elif arquivo.endswith(".txt"):**

**loader = TextLoader(caminho\_arquivo)**

**else:**

**continue**

**documentos.extend(loader.load())**

**return documentos**

**# Criar o índice vetorial para consulta eficiente**

**embeddings = HuggingFaceEmbeddings(model\_name="sentence-transformers/all-MiniLM-L6-v2")**

**index = Chroma.from\_documents(carregar\_documentos(), embeddings)**

class Pergunta(BaseModel):

pergunta: str

def extrair\_palavras\_chave(texto: str) -> List[str]:

"""Extrai palavras-chave da pergunta"""

stop\_words = set(stopwords.words("portuguese"))

tokens = word\_tokenize(texto.lower()) # Tokeniza e converte em minúsculas

palavras\_chave = [t for t in tokens if t.isalnum() and t not in stop\_words]

return palavras\_chave

def determinar\_fluxo(palavras\_chave: List[str]) -> str:

"""Identifica o fluxo de trabalho baseado nas palavras-chave"""

for fluxo, palavras in fluxos.items():

# Verifica se alguma palavra-chave do fluxo aparece nas palavras extraídas

if any(palavra in palavras\_chave for palavra in palavras):

return f"Fluxo relacionado a: {fluxo.capitalize()}"

return "Nenhum fluxo específico identificado."

# Função para obter resposta do LLaMA com a consulta dos documentos

**def obter\_resposta\_llama(pergunta: str) -> str:**

**# Primeiro, buscar nos documentos usando o índice**

**docsearch = index.as\_search\_engine()**

**resultado\_busca = docsearch.search(pergunta)**

**# O contexto da busca será usado para gerar a resposta final**

**contexto = " ".join([res["content"] for res in resultado\_busca["results"][:3]])**

# Usar LLaMA para gerar a resposta final

payload = {

"model": "llama3.2",

"prompt": f"{contexto}\n\nPergunta: {pergunta}",

"stream": False

}

resposta = requests.post(OLLAMA\_URL, json=payload)

return resposta.json().get("response", "Erro ao obter resposta")

@app.post("/pergunta/")

async def fazer\_pergunta(pergunta: Pergunta):

"""Processa a pergunta, envia ao LLaMA e retorna a resposta"""

palavras\_chave = extrair\_palavras\_chave(pergunta.pergunta)

fluxo = determinar\_fluxo(palavras\_chave)

# Obter resposta do LLaMA com consulta aos documentos

resposta\_llama = obter\_resposta\_llama(pergunta.pergunta)

return {"fluxo": fluxo, "resposta": resposta\_llama}

**ATIVIDADE 4**

Analise os blocos em negrito e identifique as mudanças. Entenda as mudanças e o que cada parte dos comandos de extração de informação dos documentos faz.

Tente executar este código, que está completo. Lembre-se que uma pasta com documentos será necessária.