UNIVERSIDADE FEDERAL DE SÃO PAULO
CAMPUS SÃO JOSÉ DOS CAMPOS

TRABALHO PRÁTICO INDIVIDUAL DE ENG. SW

Caroline Maximo 163.650

Docente: Professor Dr. Fabio Fagundes Silveira

SÃO JOSÉ DOS CAMPOS - SP
JANEIRO - 2025

Sumário

1 Introdução

Este projeto é parte de um experimento inicial para integrar modelos de Inteligência Artificial em um aplicativo que visa aproximar pessoas para treinarem juntas. O aplicativo, que constitui parte do trabalho final do curso, terá um chat onde usuários podem interagir e receber sugestões de rotinas de treino. Assim, foi concebida esta prova de conceito que une duas abordagens diferentes de modelos de linguagem: **Ollama LLM** (através da biblioteca `langchain_ollama`) e **OpenAI Chat** (através da biblioteca `openai`).

O objetivo é que, no futuro, esse recurso seja inserido no chat do aplicativo, permitindo a geração rápida de treinos personalizados, otimizando a rotina de usuários que desejam encontrar parceiros de treino e, ao mesmo tempo, receber aconselhamento de um "personal trainer virtual" a qualquer momento.

2 Arquivos do Projeto e Estrutura

2.1 Arquivo `ollamaAI.py`

Figura 1: ollamaAI.py

```python
# Cria o objeto de prompt utilizando o template
prompt = ChatPromptTemplate.from_template(template)

# Encadeia prompt + modelo
chain = prompt | model

def handle_conversation():
    context = ""
    print("Iniciando conversa! Digite 'exit' para sair.")
    while True:
        user_input = input("Você: ")
        if user_input.lower() == "exit":
            break

        # Invoca a cadeia (chain)
        result = chain.invoke({"context": context, "question": user_input})

        print("Personal:", result)
        # Atualiza o contexto
        context += f"\nUser: {user_input}\nAI: {result}"

if __name__ == "__main__":
    handle_conversation()
```

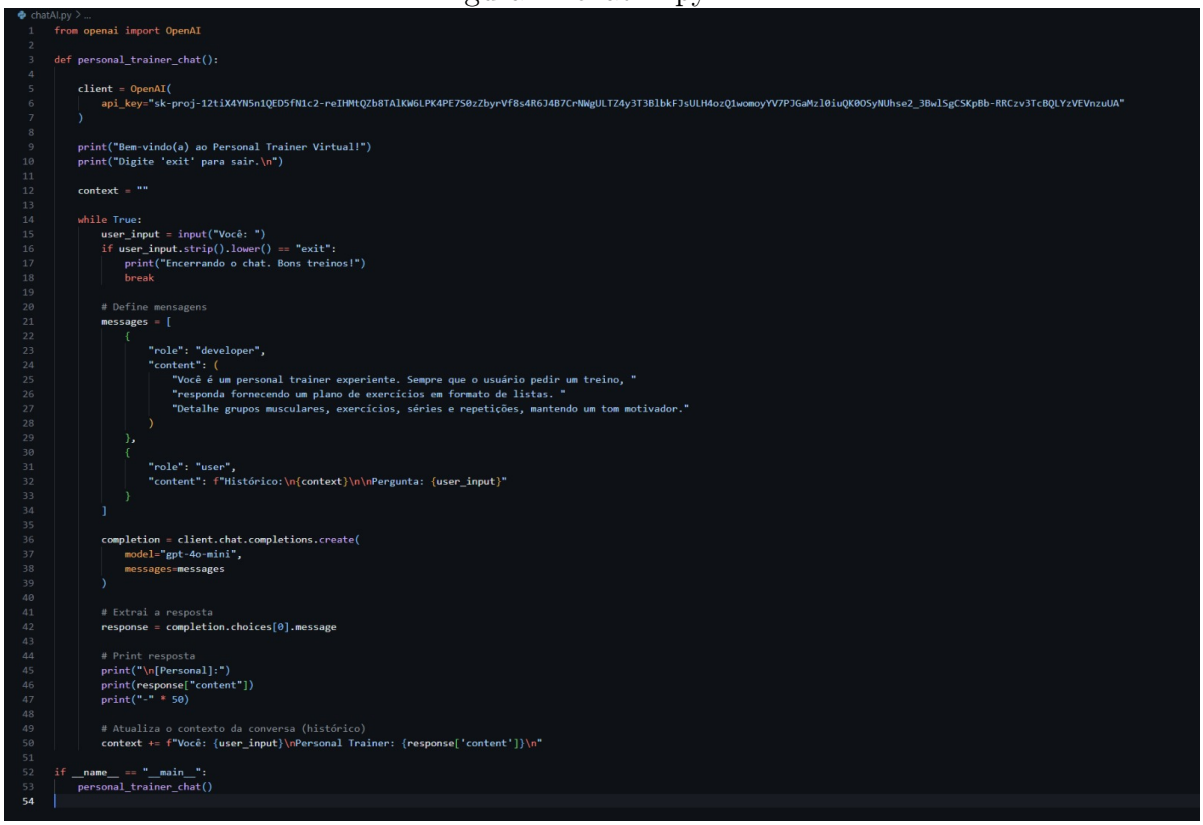
Descrição e Funcionamento:

- O arquivo `ollamaAI.py` utiliza a biblioteca `langchain_ollama` para interagir com o modelo `OllamaLLM` (no caso, `"gemma2:2b"`).
- O **template** define que a IA deve se comportar como um *personal trainer* e responder em formato de listas, detalhando exercícios, grupos musculares, séries e repetições.
- Com a criação do objeto `prompt` (via `ChatPromptTemplate`), esse template é combinado ao modelo. A variável `chain` representa o "encadeamento" entre `prompt` e o `model`.
- Na função `handle_conversation()`, o programa entra em um **loop** que:
  1. Pede input do usuário.
  2. Passa esse input (e o contexto acumulado) para o modelo Ollama via `chain.invoke(...)`.
  3. Recebe a resposta e imprime na tela, ao mesmo tempo atualizando o histórico de conversa em `context`.

- Caso o usuário digite `"exit"`, a conversa é encerrada.

2.2 Arquivo `chatAI.py`

Figura 2: chatAI.py



```python
from openai import OpenAI

def personal_trainer_chat():

    client = OpenAI(
        api_key="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"
    )

    print("Bem-vindo(a) ao Personal Trainer Virtual!")
    print("Digite 'exit' para sair.\n")

    context = ""

    while True:
        user_input = input("Você: ")
        if user_input.strip().lower() == "exit":
            print("Encerrando o chat. Bons treinos!")
            break

        # Define mensagens
        messages = [
            {
                "role": "developer",
                "content": (
                    "Você é um personal trainer experiente. Sempre que o usuário pedir um treino, "
                    "responda fornecendo um plano de exercícios em formato de listas. "
                    "Detalhe grupos musculares, exercícios, séries e repetições, mantendo um tom motivador."
                )
            },
            {
                "role": "user",
                "content": f"Histórico:\n{context}\n\nPergunta: {user_input}"
            }
        ]

        completion = client.chat.completions.create(
            model="gpt-4o-mini",
            messages=messages
        )

        # Extrai a resposta
        response = completion.choices[0].message

        # Print resposta
        print("\n[Personal]:")
        print(response["content"])
        print("-" * 50)

        # Atualiza o contexto da conversa (histórico)
        context += f"Você: {user_input}\nPersonal Trainer: {response['content']}\n"

if __name__ == "__main__":
    personal_trainer_chat()
```

Descrição e Funcionamento:

- O arquivo `chatAI.py` faz a conexão com a API da OpenAI através da classe `OpenAI`, fornecendo a `api_key` diretamente no código (apenas para testes locais).
- A função `personal_trainer_chat()` controla o fluxo do chat:
    1. Inicia um loop de `input` para leitura das perguntas do usuário.
    2. Monta a lista de **mensagens** (`messages`), incluindo uma mensagem de papel (`"role": "developer"`) descrevendo que a IA é um personal trainer.
    3. Faz a requisição ao método `client.chat.completions.create(...)` informando o modelo `gpt-4o-mini`.
    4. Recebe a resposta, imprime no terminal e atualiza o `context` para reter o histórico.

- O usuário pode encerrar a conversa ao digitar `"exit"`.

2.3 Arquivo `main.py`

Figura 3: main.py

```python
import ollamaAI
import chatAI

def main():
    print("Escolha o modelo de IA que deseja usar:")
    print("1) Ollama")
    print("2) Chat GPT (OpenAI)")

    opcao = input("Digite sua opção: ").strip()

    if opcao == "1":
        # Chama a função do Ollama
        ollamaAI.handle_conversation()
    elif opcao == "2":
        # Chama a função da OpenAI
        chatAI.personal_trainer_chat()
    else:
        print("Opção inválida. Finalizando...")

if __name__ == "__main__":
    main()
```

Descrição e Funcionamento:

- O arquivo `main.py` centraliza a escolha de qual motor de IA utilizar:
  1. Importa tanto `ollamaAI` quanto `chatAI`.
  2. Imprime as opções e lê via `input()` qual IA o usuário deseja.
  3. Executa a função correspondente:
     - `ollamaAI.handle_conversation()` para usar o modelo `Ollama`.
     - `chatAI.personal_trainer_chat()` para usar a API da OpenAI.
  4. Caso a resposta não seja reconhecida, exibe uma mensagem de erro e encerra.
- Essa organização facilita a manutenção e a expansão futura do projeto, pois permite adicionar novos "motores" de IA apenas seguindo o mesmo padrão de import e chamada de função.

3 Ferramentas e Bibliotecas Utilizadas

- **Python 3.10.9**: Linguagem de programação principal do projeto.
- **openai** (`pip install openai`): Biblioteca oficial da OpenAI para integração com os modelos de linguagem.
- **langchain_ollama** (`pip install langchain-ollama`): Fornece classes e métodos para conexão com modelos Ollama.
- **LangChain Core** (`langchain_core.prompts`): Pacote auxiliar que facilita a criação de *prompts* mais elaborados para LLMs.

4 Conclusão

Com este protótipo, demonstramos como é possível integrar diferentes modelos de linguagem (Ollama e OpenAI) em uma mesma aplicação para gerar planos de treino personalizados. Esta abordagem se encaixa diretamente na proposta do aplicativo, que funciona como uma plataforma para unir pessoas que desejam encontrar parceiros de treino e, ao mesmo tempo, obter orientação rápida de um personal trainer virtual.

Portanto, no contexto do projeto final do curso, a ideia seria integrar esta solução diretamente ao chat do aplicativo. Assim, os usuários não apenas poderão encontrar parceiros de treino, mas também receber dicas e sugestões personalizadas em tempo real, tornando a experiência mais envolvente, prática e completa para todos.