Instituto Infnet

GRADUAÇÃO EM BANCO DE DADOS

DESENVOLVIMENTO DE DATA-DRIVEN APPS COM PYTHON [24E4_3]

WANDERSON RAFAEL MENDONÇA BATISTA

TESTE DE PERFORMANCE – TP2

PROF. FERNANDO GUIMARÃES FERREIRA

RIO DE JANEIRO, 2024

Os códigos estão disponíveis em https://github.com/wanderaf/wanderaf-datadrive_tp02

Parte 1 FastAPI

Questão 1: Crie uma aplicação simples em FastAPI que utilize o modelo GPT-2 da HuggingFace para gerar textos a partir de uma entrada fornecida via requisição HTTP.

O aplicativo deve:

Receber uma frase de entrada como JSON.

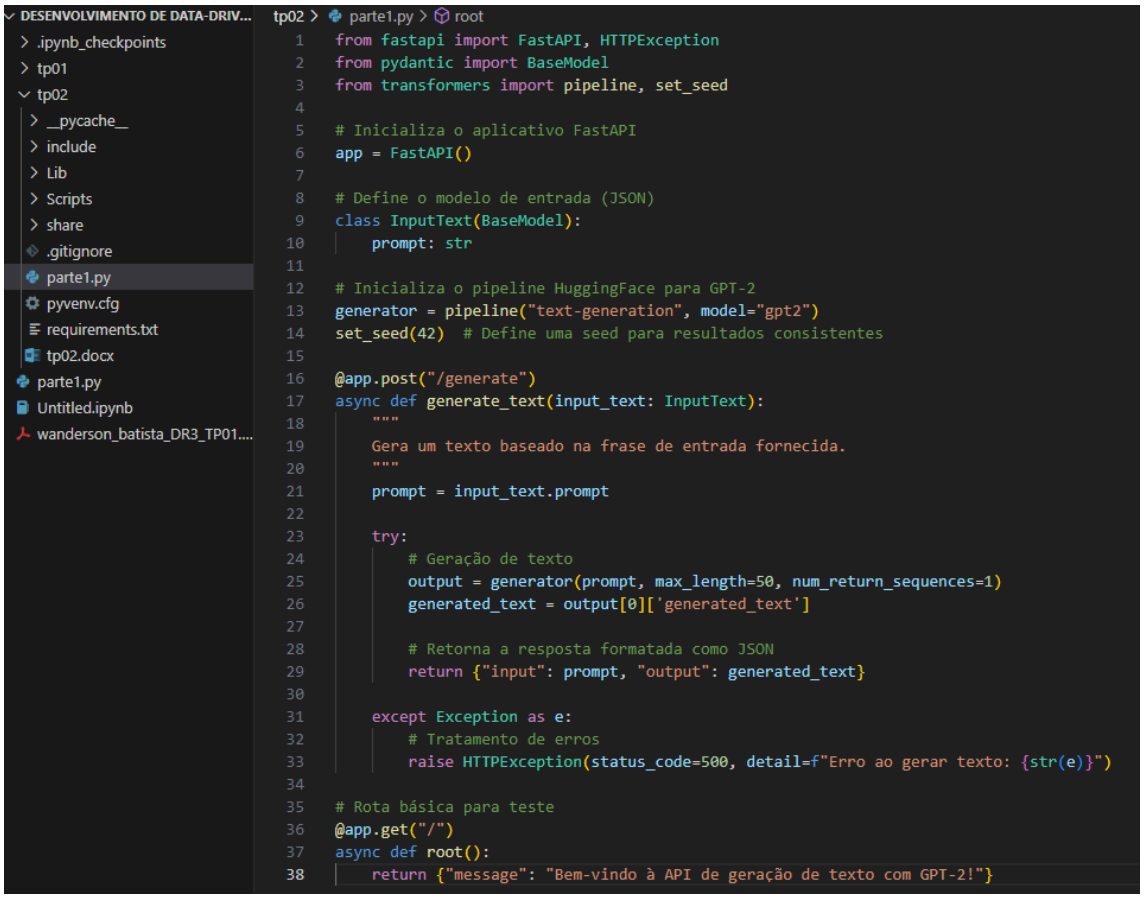
Utilizar a biblioteca transformers do HuggingFace para gerar um texto de saída.

Retornar o texto gerado em uma resposta HTTP.

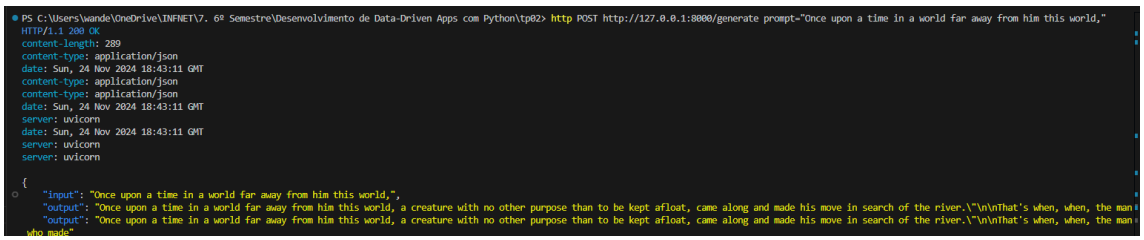
O que é esperado:

O aplicativo deve gerar uma continuação de texto a partir de uma frase de entrada e retornar a resposta formatada como JSON.

Código utilizado



Saída



Questão 2: Crie um aplicativo FastAPI que utiliza o modelo de tradução Helsinki-NLP/opus-mt-en-fr da HuggingFace para traduzir textos do inglês para o francês.

A aplicação deve:

Receber um texto em inglês via uma requisição HTTP.

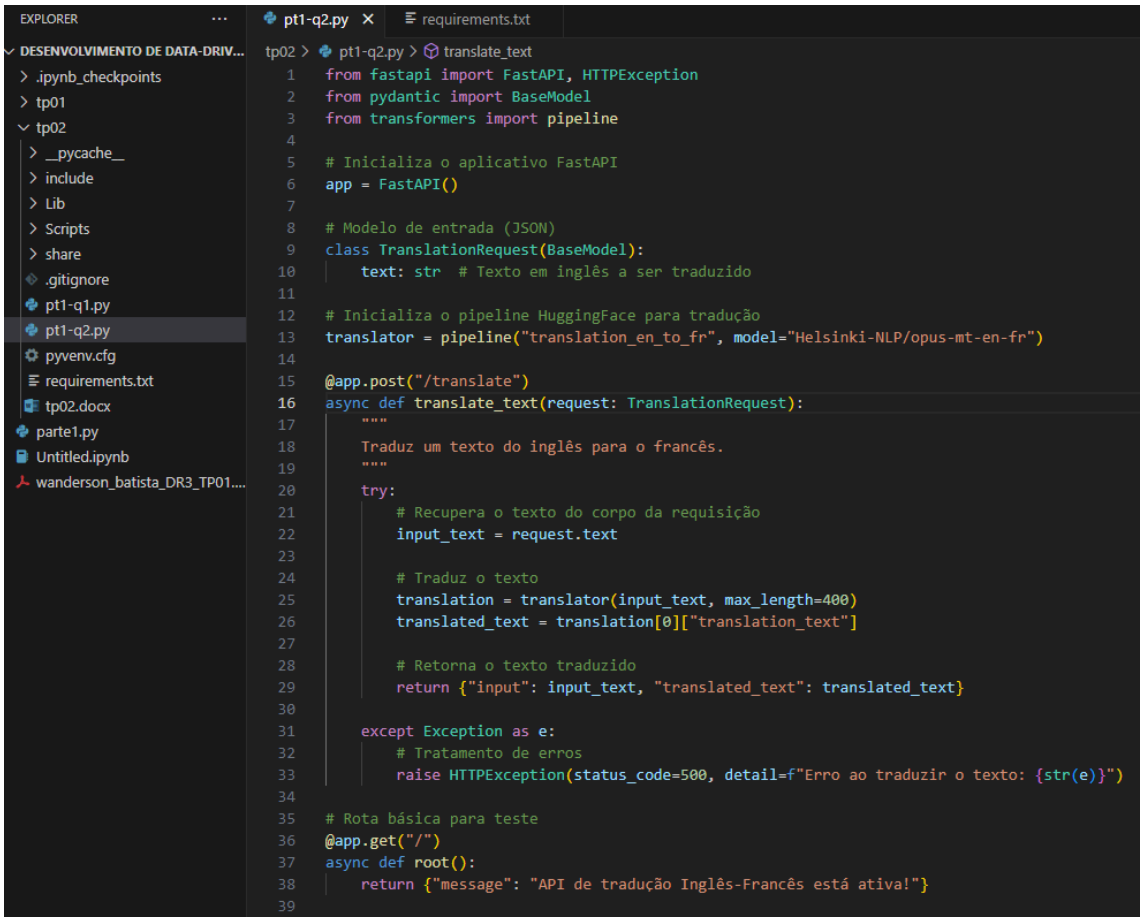
Traduzir o texto para o francês utilizando o modelo de tradução.

Retornar o texto traduzido em uma resposta JSON.

O que é esperado:

A API deve receber um texto em inglês e retornar sua tradução para o francês, processando tanto frases curtas quanto textos mais longos.

Código utilizado



Saída

```
tp POST http://127.0.0.1:8000/translate text="Hello, how are you?"
HTTP/1.1 200 OK
content-length: 81
content-type: application/json
HTTP/1.1 200 OK
content-length: 81
content-type: application/json
content-length: 81
content-type: application/json
content-type: application/json
date: Sun, 24 Nov 2024 19:05:31 GMT
server: uvicorn

{
    "input": "Hello, how are you?",
    "translated_text": "Bonjour, comment allez-vous ?"
server: uvicorn

{
```

```
PS C:\Users\wande\OneDrive\INFNET\7. 6º Semestre\Desenvolvimento de Data-Driven Apps com Python\tp02> http POST http:/
/127.0.0.1:8000/translate text="Artificial intelligence is transforming industries across the globe. From healthcare a
nd education to finance and entertainment, AI-powered solutions are improving efficiency, enabling innovation, and unl
ocking new possibilities. As the technology continues to evolve, it is crucial to address the ethical chalHTTPHTTP/1.1
HTTP/1.1 200 OK
content-length: 710
content-type: application/json
date: Sun, 24 Nov 2024 19:13:03 GMT
server: uvicorn

{
    "input": "Artificial intelligence is transforming industries across the globe. From healthcare and education to fi
nance and entertainment, AI-powered solutions are improving efficiency, enabling innovation, and unlocking new possibi
lities. As the technology continues to evolve, it is crucial to address the ethical challenges and ensure that AI is d
eveloped and deployed responsibly for the benefit of humanity.",
    "translated_text": "L'intelligence artificielle transforme les industries à travers le monde. Des soins de santé e
t de l'éducation au financement et au divertissement, les solutions alimentées par l'IA améliorent l'efficacité, perme
ttent l'innovation et ouvrent de nouvelles possibilités."
}


PS C:\Users\wande\OneDrive\INFNET\7. 6º Semestre\Desenvolvimento de Data-Driven Apps com Python\tp02>
```

Questão 3: Com base na API desenvolvida na Questão 2 (Parte1), explique as principais limitações do modelo de tradução utilizado.

Enumere e discuta:

Limitações quanto à precisão da tradução.

Desafios de tempo de resposta e desempenho em grande escala.

Restrições de custo e escalabilidade.

Limitações na tradução de gírias, expressões idiomáticas ou linguagem de contexto.

Existe limitações quanto à precisão, especialmente em gírias, expressões idiomáticas e linguagem de contexto, resultando frequentemente em traduções literais ou interpretações inadequadas. Além disso, o modelo não diferencia nuances regionais do francês, o que pode gerar traduções que soam artificiais para certos públicos. A falta de contexto em frases isoladas é outro desafio, já que o modelo não considera parágrafos inteiros, levando a inconsistências em textos mais complexos.

Em termos de desempenho, o modelo demanda recursos computacionais elevados, especialmente para entradas maiores ou múltiplas requisições simultâneas, o que aumenta a latência e dificulta a escalabilidade. Os custos para infraestrutura robusta, como GPUs, podem ser altos, seja em execução local ou na nuvem. Para mitigar essas limitações, é possível realizar um ajuste fino do modelo em corpus especializado, utilizar processamento em lote para melhorar a eficiência e adotar modelos menores ou quantizados para reduzir custos sem comprometer demasiadamente a precisão.

Questão 4: Com base no modelo GPT-2 utilizado na Questão 1 (Parte 1), explique as principais limitações do modelo no contexto da geração de texto.

Discuta:

A coerência do texto gerado.

Possíveis falhas ou incoerências geradas por LLMs.

Desempenho e questões de latência.

Limitações na geração de conteúdo apropriado.

O modelo apresenta limitações na coerência, podendo perder o foco, repetir ideias ou gerar informações desconexas. Ele também pode produzir conteúdo factualmente incorreto ou inadequado, já que não possui mecanismos para filtrar informações sensíveis ou enviesadas. Além disso, a geração de texto é intensiva em recursos, podendo causar latência em sistemas menos robustos. Essas limitações tornam o GPT-2 mais adequado para tarefas criativas e genéricas, mas menos confiável para aplicações que exigem precisão e controle contextual.

Parte 2 LangChain

Questão 1: Desenvolva um protótipo utilizando LangChain que simule um chatbot simples com Fake LLM.

A aplicação deve:

Receber um input de texto via FastAPI.

Retornar uma resposta simulada pelo Fake LLM.

O que é esperado:

O protótipo deve simular um chatbot básico que responde a perguntas pré-definidas.

A arquitetura deve ser simples, e você deve explicar a importância de usar Fake LLM para testes rápidos.

Desenhe um diagrama simples da arquitetura do aplicativo, detalhando as principais etapas do fluxo de dados.

requirements.txt × pt2-q1.py ●

tp02 > pt2-q1.py > root

```python
from fastapi import FastAPI
from pydantic import BaseModel
from langchain.llms.fake import FakeListLLM
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

# Inicializa o aplicativo FastAPI
app = FastAPI()

# Define o modelo de entrada
class ChatRequest(BaseModel):
    user_input: str  # Mensagem enviada pelo usuário

# Configura o FakeListLLM com respostas predefinidas
responses = [
    "Hello! How can I assist you today?",
    "I'm just a bot, but I'm functioning perfectly!",
    "I'm sorry, I don't understand your question.",
]
fake_llm = FakeListLLM(responses=responses)

# Define o template do prompt
prompt_template = PromptTemplate(input_variables=["user_input"], template="{user_input}")
chat_chain = LLMChain(llm=fake_llm, prompt=prompt_template)

@app.post("/chat")
async def chat_with_bot(request: ChatRequest):
    """
    Simula uma conversa com o chatbot.
    """
    user_input = request.user_input
    response = chat_chain.run({"user_input": user_input})
    return {"user_input": user_input, "bot_response": response}

# Rota básica para teste
@app.get("/")
async def root():
    return {"message": "Chatbot com Fake LLM está ativo!"}
```

Saídas

```
PS C:\Users\wande\OneDrive\INFNET\7. 6º Semestre\Desenvolvimento de Data-Driven Apps com Python\tp02>
PS C:\Users\wande\OneDrive\INFNET\7. 6º Semestre\Desenvolvimento de Data-Driven Apps com Python\tp02> http POST http://1
27.0.0.1:8000/chat user_input="Hello"
>> Can you provide code in Python to handle texts that have accuracy?Can you provide code in Python to handle texts that
 have accuracy?
HTTP/1.1 200 OK
content-length: 86
content-type: application/json
date: Sun, 24 Nov 2024 20:34:55 GMT
server: uvicorn

{
    "bot_response": "I'm just a bot, but I'm functioning perfectly!",
    "user_input": "Hello"
}
```

```
    "bot_response": "I'm sorry, I don't understand your question.",
{
    "bot_response": "I'm sorry, I don't understand your question.",
    "user_input": "Can you provide code in Python to handle texts that have accuracy?"
    "bot_response": "I'm sorry, I don't understand your question.",
    "user_input": "Can you provide code in Python to handle texts that have accuracy?"
    "user_input": "Can you provide code in Python to handle texts that have accuracy?"
}
```

Diagrama

O FakeLLM torna-se importante para realização de testes rápidos e econômicos, uma vez que permite validar fluxos sem custos adicionais, possui baixa latência, já que não necessita de chamadas externas, sua personalização é simples uma vez que é fácil modificar para cobrir cenários específicos e permite escalabilidade para modelos reais, após validação o FakeLLM pode ser substituído por um modelo real.

Questão 2: Desenvolva um aplicativo que utilize LangChain para integrar a API da OpenAI.

O aplicativo deve:

Receber um texto em inglês via FastAPI.

Traduzir o texto para o francês utilizando um modelo da OpenAI via LangChain.

Retornar o texto traduzido em uma resposta JSON.

O que é esperado:

O aplicativo deve funcionar como uma API de tradução, semelhante à questão 2 (Parte 1), mas utilizando a OpenAI via LangChain.

A aplicação deve gerenciar as chamadas à API da OpenAI e retornar a tradução com baixa latência.

Forneça um diagrama da arquitetura da aplicação, destacando os componentes principais, como FastAPI, LangChain, e OpenAI.

Código

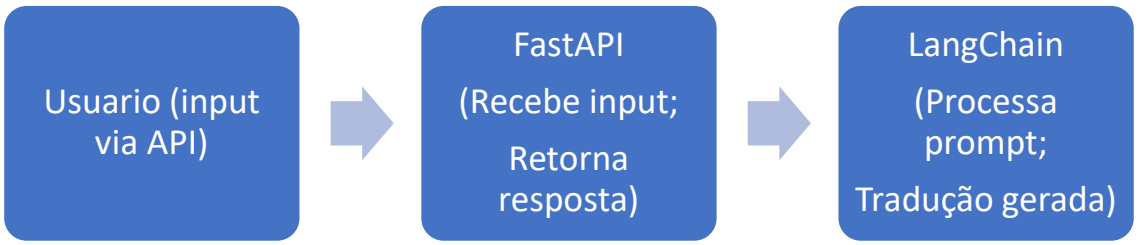
requirements.txt | pt2-q2.py | teste.py | .env

tp02 > pt2-q2.py > root

```python
from fastapi import FastAPI
from pydantic import BaseModel
from langchain.chat_models import ChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
import os
from dotenv import load_dotenv

# Configuração da API OpenAI
load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")

# Inicializa o aplicativo FastAPI
app = FastAPI()

# Define o modelo de entrada
class TranslationRequest(BaseModel):
    text: str  # Texto em inglês a ser traduzido

# Configura o modelo OpenAI via LangChain
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

# Define o template do prompt para tradução
prompt_template = PromptTemplate(
    input_variables=["text"],
    template="Translate the following text to French:\n{text}"
)

# Cria a cadeia LLM com o modelo e o prompt
translation_chain = LLMChain(llm=llm, prompt=prompt_template)

@app.post("/translate")
async def translate_text(request: TranslationRequest):
    """
    Traduz o texto do inglês para o francês.
    """
    input_text = request.text
    try:
        # Executa a tradução usando LangChain
        translated_text = translation_chain.run({"text": input_text})
        return {"input_text": input_text, "translated_text": translated_text}
    except Exception as e:
        return {"error": str(e)}

# Rota básica para teste
@app.get("/")
async def root():
    return {"message": "API de tradução com OpenAI está ativa!"}
```

Saída

```
PS C:\Users\wande\OneDrive\INFNET\7. 6º Semestre\Desenvolvimento de Data-Driven
tp POST http://127.0.0.1:8000/translate text="Hello, how are you?"
HTTP/1.1 200 OK
content-length: 82
tp POST http://127.0.0.1:8000/translate text="Hello, how are you?"
HTTP/1.1 200 OK
content-length: 82
HTTP/1.1 200 OK
content-length: 82
content-type: application/json
date: Sun, 24 Nov 2024 22:00:44 GMT
content-type: application/json
date: Sun, 24 Nov 2024 22:00:44 GMT
date: Sun, 24 Nov 2024 22:00:44 GMT
server: uvicorn

{
    "input_text": "Hello, how are you?",
    "translated_text": "Bonjour, comment ça va ?"
```



Questão 3: Crie uma API semelhante à Questão 2 (Parte 2), mas que utilize o modelo Helsinki-NLP/opus-mt-en-de da HuggingFace para traduzir textos do inglês para o alemão.

A aplicação deve:

Receber um texto em inglês via FastAPI.

Utilizar o LangChain para gerenciar as chamadas ao modelo HuggingFace.

Retornar o texto traduzido para o alemão como resposta JSON.

O que é esperado:

O objetivo é que a aplicação funcione de maneira semelhante às Questões 2 (Parte 1) e 2 (Parte 2), mas desta vez integrando LangChain com HuggingFace.

O modelo a ser utilizado deve ser o Helsinki-NLP/opus-mt-en-de.

Forneça um diagrama detalhado da arquitetura da aplicação, destacando as interações entre FastAPI, LangChain, e HuggingFace.

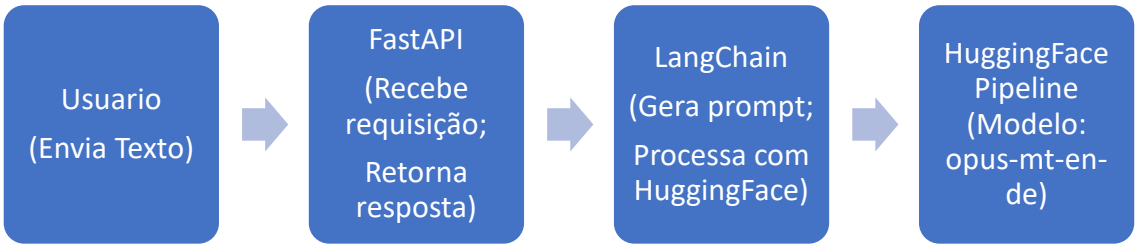
requirements.txt | pt2-q2.py | pt2-q3.py | teste.py | .env

tp02 > pt2-q3.py > root

```python
from fastapi import FastAPI
from pydantic import BaseModel
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate
from langchain.llms import HuggingFacePipeline
from transformers import pipeline

# Inicializa o aplicativo FastAPI
app = FastAPI()

# Define o modelo de entrada
class TranslationRequest(BaseModel):
    text: str  # Texto em inglês a ser traduzido

# Configura o pipeline HuggingFace para o modelo Helsinki-NLP
translator_pipeline = pipeline("translation", model="Helsinki-NLP/opus-mt-en-de")

# Configura o LangChain com o pipeline HuggingFace
llm = HuggingFacePipeline(pipeline=translator_pipeline)

# Define o template do prompt para LangChain
prompt_template = PromptTemplate(
    input_variables=["text"],
    template="Translate the following text to German:\n{text}"
)

# Cria a cadeia LLM com LangChain
translation_chain = LLMChain(llm=llm, prompt=prompt_template)

@app.post("/translate")
async def translate_text(request: TranslationRequest):
    """
    Traduz o texto do inglês para o alemão.
    """
    input_text = request.text
    try:
        # Executa a tradução usando LangChain
        translated_text = translation_chain.run({"text": input_text})
        return {"input_text": input_text, "translated_text": translated_text}
    except Exception as e:
        return {"error": str(e)}

# Rota básica para teste
@app.get("/")
async def root():
    return {"message": "API de tradução com HuggingFace está ativa!"}
```

Saída

```
PS C:\Users\wande\OneDrive\INFNET\7. 6º Semestre\Desenvolvimento de Data-Driven Apps com Python\tp02> http POST ht
tp://127.0.0.1:8000/translate text="Hello, how are you?"
HTTP/1.1 200 OK
content-length: 130
content-type: application/json
date: Sun, 24 Nov 2024 22:37:59 GMT
server: uvicorn
    "input_text": "Hello, how are you?",
    "translated_text": "Übersetzen Sie den folgenden Text auf Deutsch: Hallo, wie geht es Ihnen?"
}
```



Questão 4: Com base na implementação da Questão 2 (Parte 2), explique as principais limitações de utilizar LangChain para integrar a API da OpenAI.

Discuta os seguintes aspectos:

Latência de resposta.

Limites de uso da API da OpenAI.

Desafios de escalabilidade e custo.

Qualidade das traduções geradas em comparação com outros modelos.

A latência pode ser significativa devido ao tempo de processamento da API e à camada adicional de LangChain, tornando-a menos ideal para sistemas que exigem respostas em tempo real. Além disso, os limites de uso impostos pela OpenAI, como restrições de tokens e requisições por minuto, podem interromper o serviço em cenários de alta demanda, especialmente sem um plano pago robusto.

Os custos também são um desafio, pois cada requisição é cobrada com base no número de tokens, e prompts maiores podem aumentar os gastos de forma imprevisível. Em termos de qualidade, embora a OpenAI ofereça traduções geralmente boas, elas podem ser inferiores às de modelos especializados como DeepL ou Helsinki-NLP, especialmente em textos técnicos ou regionais.

Questão 5: Com base na aplicação desenvolvida na 3 (Parte 2), explique as limitações de usar LangChain para integrar o modelo HuggingFace de tradução.

Discuta aspectos como:

Desempenho e tempo de resposta.

Consumo de recursos computacionais.

Possíveis limitações no ajuste fino do modelo.

Comparação com o uso direto da API HuggingFace.

Embora LangChain simplifique a integração com modelos HuggingFace e organize o fluxo de tradução, sua aplicação apresenta desafios de desempenho, alta demanda computacional e limitações para personalização avançada. Além disso, em casos simples, o uso direto da API HuggingFace pode ser mais eficiente, reduzindo sobrecarga e aumentando o controle sobre o pipeline. A escolha entre LangChain e a API direta depende do equilíbrio entre simplicidade de implementação e necessidades de desempenho ou personalização.

Questão 6: Com base nas questões 1-2 (Parte 1) e 2-3 (Parte 2), desenvolva uma tabela comparativa que aborde os seguintes critérios:

Facilidade de uso/configuração.

Latência e desempenho.

Flexibilidade para diferentes modelos.

Custo e escalabilidade.

Adequação para protótipos versus aplicações em produção.

A comparação deve ser apresentada em formato de tabela, com colunas dedicadas a cada critério e linhas comparando FastAPI puro com LangChain.

Critério	FastAPI Puro	LangChain
Facilidade de uso/configuração	Simples de configurar para tarefas específicas, mas exige maior esforço manual para gerenciar prompts e integrações com APIs ou modelos.	Abstrai a configuração de prompts e integrações, tornando o fluxo mais organizado e fácil de expandir. Ideal para aplicações complexas.
Latência e desempenho	Baixa latência, pois não adiciona camadas extras de abstração. Depende apenas	Latência um pouco maior devido à abstração adicional e ao gerenciamento de fluxo

Critério	FastAPI Puro	LangChain
	do tempo de resposta do modelo ou API integrada.	interno. Pode ser um gargalo em aplicações de tempo real.
Flexibilidade para diferentes modelos	Integra facilmente APIs ou bibliotecas, mas requer código adicional para alternar entre diferentes modelos (ex.: OpenAI, HuggingFace).	Altamente flexível, permitindo alternar entre diferentes modelos com modificações mínimas, ideal para fluxos complexos.
Custo e escalabilidade	Geralmente mais eficiente em custo, pois elimina camadas intermediárias. Escalabilidade depende da otimização da infraestrutura de backend.	Custos podem ser mais altos devido à sobrecarga da abstração e prompts mais extensos. Requer planejamento para escalabilidade em produção.
Adequação para protótipos versus aplicações em produção	Melhor para aplicações simples ou sistemas de produção onde o controle total e desempenho são cruciais. Menos ideal para prototipagem rápida.	Ideal para prototipagem rápida devido à facilidade de uso e abstração. Pode ser ajustado para produção, mas exige otimizações para desempenho e custo.