Folha de Dicas: Fundamentos da Construção de Agentes de IA usando RAG e LangChain

Pacote/Método	Descrição	Exemplo de código

Gerar texto	Este trecho de código gera sequências de texto com base na entrada e não calcula o gradiente para gerar a saída.	`# Gerar texto`
		`output_ids = model.generate(`
		`  inputs.input_ids,`
		`  attention_mask=inputs.attention_mask,`
		`  pad_token_id=tokenizer.eos_token_id,`
		`  max_length=50,`
		`  num_return_sequences=1`
		`)`
		`output_ids`
		`ou`
		`com torch.no_grad():`
		`  outputs = model(**inputs)`
		`outputs`

função formatting_prompts_func_no_response	A função prompt gera prompts de texto formatados a partir de um conjunto de dados usando as instruções do conjunto de dados. Ela cria strings que incluem apenas a instrução e um espaço reservado para a resposta.	`def formatting_prompts_func(mydataset):`
		`  output_texts = []`
		`  for i in range(len(mydataset['instruction'])):`
		`  text = (`
		`  f"### Instrução:\n{mydataset['instruction'][i]}"`
		`  f"\n\n### Resposta:\n{mydataset['output'][i]}"`
		`  )`
		`  output_texts.append(text)`
		`  return output_texts`
		`def formatting_prompts_func_no_response(mydataset):`
		`  output_texts = []`
		`  for i in range(len(mydataset['instruction'])):`
		`  text = (`
		`  f"### Instrução:\n{mydataset['instruction'][i]}"`
		`  f"\n\n### Resposta:\n"`
		`  )`
		`  output_texts.append(text)`
		`  return output_texts`

torch.no_grad()	Este trecho de código ajuda a gerar sequências de texto a partir da função de pipeline. Ele garante que os cálculos de gradiente sejam desativados e otimiza o desempenho e o uso de memória.	`com torch.no_grad():`
		`  # Devido à limitação de recursos, aplique a função apenas em 3 registro`
		`  pipeline_iterator= gen_pipeline(instructions_torch[:3],`
		`  max_length=50, # isso é definido como 50 de`
		`  num_beams=5,`
		`  early_stopping=True,)`
		`generated_outputs_lora = []`
		`for text in pipeline_iterator:`
		`  generated_outputs_lora.append(text[0]["generated_text"])`

Objeto de modelo de inferência mixtral-8x7b-instruct-v01 watsonx.ai	Ajusta os parâmetros para expandir os limites de criatividade e comprimento da resposta.	`model_id = 'mistralai/mixtral-8x7b-instruct-v01'`
		`parameters = {`
		`  GenParams.MAX_NEW_TOKENS: 256,  # isso controla o número máximo de toke`
		`  GenParams.TEMPERATURE: 0.5, # isso controla a aleatoriedade ou criativi`
		`}`
		`credentials = {`
		`  "url": "https://us-south.ml.cloud.ibm.com"`
		`}`
		`project_id = "skills-network"`
		`model = ModelInference(`
		`  model_id=model_id,`
		`  params=parameters,`
		`  credentials=credentials,`
		`  project_id=project_id`
		`)`

Pacote/Método	Descrição	Exemplo de código

Templates de prompt de string	Usados para formatar uma única string e geralmente são utilizados para entradas mais simples.	`from langchain_core.prompts import PromptTemplate`
		`prompt = PromptTemplate.from_template("Me conte uma piada {adjective} sobre`
		`input_ = {"adjective": "engraçada", "topic": "gatos"}  # cria um dicionário`
		`prompt.invoke(input_)`

Templates de prompt de chat	Usados para formatar uma lista de mensagens. Esses "templates" consistem em uma lista de templates em si.	`from langchain_core.prompts import ChatPromptTemplate`
		`prompt = ChatPromptTemplate.from_messages([`
		`("system", "Você é um assistente útil"),`
		`("user", "Me conte uma piada sobre {topic}")`
		`])`
		`input_ = {"topic": "gatos"}`
		`prompt.invoke(input_)`

Placeholder de mensagens	Este template de prompt é responsável por adicionar uma lista de mensagens em um local específico. Mas se você quiser que o usuário passe uma lista de mensagens que você iria inserir em um local específico, o trecho de código fornecido é útil.	`from langchain_core.prompts import MessagesPlaceholder`
		`from langchain_core.messages import HumanMessage`
		`prompt = ChatPromptTemplate.from_messages([`
		`("system", "Você é um assistente útil"),`
		`MessagesPlaceholder("msgs")`
		`])`
		`input_ = {"msgs": [HumanMessage(content="Qual é o dia depois de terça-feira`
		`prompt.invoke(input_)`

Selecionador de exemplos	Se você tiver muitos exemplos, pode ser necessário selecionar quais incluir no prompt. O Selecionador de Exemplos é a classe responsável por fazer isso.	`from langchain_core.example_selectors import LengthBasedExampleSelector`
		`from langchain_core.prompts import FewShotPromptTemplate, PromptTemplate`
		
		`# Exemplos de uma tarefa fictícia de cri`
		
		`exemplos = [`
		`{“input”: “feliz”, “output”: “triste”},`
		`{“input”: “alto”, “output”: “baixo”},`
		`{“input”: “energético”, “output”: “letárgico”},`
		`{“input”: “ensolarado”, “output”: “nublado”},`
		`{“input”: “ventoso”, “output”: “calmo”},`
		`]`
		`exemplo_prompt = PromptTemplate(`
		`input_variables=[“input”, “output”],`
		`template=”Entrada: {input}\nSaída: {output}”,`
		`)`
		`seletor_exemplo = LengthBasedExampleSelector(`
		`examples=exemplos,`
		`example_prompt=exemplo_prompt,`
		`max_length=25,  # O comprimento máximo que os exemplos formatados devem`
		`)`
		`prompt_dinâmico = FewShotPromptTemplate(`
		`example_selector=seletor_exemplo,`
		`example_prompt=exemplo_prompt,`
		`prefix=”Dê o antônimo de cada entrada”,`
		`suffix=”Entrada: {adjetivo}\nSaída:”,`
		`input_variables=[“adjetivo”],`
		`)`

Pacote/Método	Descrição	Exemplo de código

Parser JSON	Esse parser de saída permite que os usuários especifiquem um esquema JSON arbitrário e consultem LLMs para saídas que estejam em conformidade com esse esquema.	`from langchain_core.output_parsers import JsonOutputParser`
		`from langchain_core.pydantic_v1 import BaseModel, Field`
		Defina a estrutura de dados desejada.
		`class Joke(BaseModel):`
		`  setup: str = Field(description=”pergunta para preparar uma piada”)`
		`  punchline: str = Field(description=”resposta para resolver a piada”)`
		E uma consulta destinada a solicitar a
		`joke_query = “Conte-me uma piada.”`
		Configurar um parser + injetar instruç
		`output_parser = JsonOutputParser(pydantic_object=Joke)`
		`format_instructions = output_parser.get_format_instructions()`
		`prompt = PromptTemplate(`
		`  template=”Responda à consulta do usuário.\n{format_instructions}\n{quer`
		`  input_variables=[“query”],`
		`  partial_variables={“format_instructions”: format_instructions},`
		`)`
		`chain = prompt | mixtral_llm | output_parser`
		`chain.invoke({“query”: joke_query})`

Parser de lista separada por vírgulas	Este parser de saída pode ser usado quando você deseja retornar uma lista de itens separados por vírgulas.	`from langchain.output_parsers import CommaSeparatedListOutputParser`
		`output_parser = CommaSeparatedListOutputParser()`
		`format_instructions = output_parser.get_format_instructions()`
		`prompt = PromptTemplate(`
		`  template="Responda à consulta do usuário. {format_instructions}\nListe`
		`  input_variables=["subject"],`
		`  partial_variables={"format_instructions": format_instructions},`
		`)`
		`chain = prompt | mixtral_llm | output_parser`

Objeto de documento	Contém informações sobre alguns dados no LangChain. Possui dois atributos: **page_content: str:** Este atributo contém o conteúdo do documento. **metadata: dict:** Este atributo contém metadados arbitrários associados ao documento. Pode ser usado para rastrear vários detalhes, como o ID do documento, nome do arquivo e assim por diante.	`from langchain_core.documents import Document`
		`Document(page_content="""Python é uma linguagem de programação interpretada`
		`  A filosofia de design do Python enfatiza a legibili`
		`  metadata={`
		`  'my_document_id' : 234234,`
		`  'my_document_source' : "Sobre Python",`
		`  'my_document_create_time' : 1680013019`
		`  })`

text_splitter	Em um nível alto, os divisores de texto funcionam da seguinte forma:
	• Divida o texto em pequenos fragmentos semanticamente significativos (geralmente frases).
	• Comece a combinar
		`text_splitter = CharacterTextSplitter(chunk_size=200, chunk_overlap=20, sep`
		`chunks = text_splitter.split_documents(document)`
		`print(len(chunks))`

Pacote/Método	Descrição	Exemplo de código
	esses pequenos fragmentos em um fragmento maior até alcançar um determinado tamanho (medido por alguma função). • Assim que atingir esse tamanho, transforme esse fragmento em seu próprio pedaço de texto e comece a criar um novo fragmento com alguma sobreposição (para manter o contexto entre os fragmentos).	

Modelos de embedding	Modelos de embedding são projetados especificamente para interagir com embeddings de texto. Embeddings geram uma representação vetorial para uma determinada peça de texto. Isso é vantajoso, pois permite que você conceitue texto dentro de um espaço vetorial. Consequentemente, você pode realizar operações como busca semântica, onde identifica peças de texto que são mais semelhantes dentro do espaço vetorial.	`from ibm_watsonx_ai.metanames import EmbedTextParamsMetaNames`
		`embed_params = {`
		`  EmbedTextParamsMetaNames.TRUNCATE_INPUT_TOKENS: 3,`
		`  EmbedTextParamsMetaNames.RETURN_OPTIONS: {"input_text": True},`
		`}`
		`from langchain_ibm import WatsonxEmbeddings`
		`watsonx_embedding = WatsonxEmbeddings(`
		`  model_id="ibm/slate-125m-english-rtrvr",`
		`  url="https://us-south.ml.cloud.ibm.com",`
		`  project_id="skills-network",`
		`  params=embed_params,`
		`)`

Recuperador apoiado por vetor	Um recuperador que usa um armazenamento vetorial para recuperar documentos. É uma camada leve em torno da classe de armazenamento vetorial para fazê-la se conformar à interface do recuperador. Utiliza os métodos de busca implementados por um armazenamento vetorial, como busca de similaridade e MMR (relevância marginal máxima), para consultar os textos no armazenamento vetorial. Como construímos uma busca de documentos em armazenamento vetorial, é muito fácil construir um recuperador.	`retriever = docsearch.as_retriever()`
		`docs = retriever.invoke("Langchain")Um recuperador de armazenamento vetoria`
		`Como construímos uma busca de documentos em armazenamento vetorial, é muito`

Classe ChatMessageHistory	Uma das classes utilitárias principais que sustentam a maioria (senão todas) as módulos de memória é a classe ChatMessageHistory. Esta camada super leve fornece métodos convenientes para salvar HumanMessages, AIMessages e depois buscá-los todos.	`from langchain.memory import ChatMessageHistory`
		`chat = mixtral_llm`
		`history = ChatMessageHistory()`
		`history.add_ai_message("oi!")`
		`history.add_user_message("qual é a capital da França?")`

Pacote/Método	Descrição	Exemplo de código

langchain.chains	Este trecho de código usa uma LangChain, biblioteca para construir aplicações de modelo de linguagem, criando uma cadeia para gerar recomendações de pratos populares com base nas localizações especificadas. Também configura as configurações de inferência do modelo para processamento adicional.	`from langchain.chains import LLMChain`
		`template = """Seu trabalho é sugerir um prato clássico da área que os usuár`
		`{location}`
		`SUA RESPOSTA:`
		`"""`
		`prompt_template = PromptTemplate(template=template, input_variables=['locat`
		cadeia 1
		`location_chain = LLMChain(llm=mixtral_llm, prompt=prompt_template, output_k`
		`location_chain.invoke(input={'location':'China'})`

Cadena sequencial simples	Cadeias sequenciais permitem que a saída de um LLM seja usada como entrada para outro. Essa abordagem é benéfica para dividir tarefas e manter o foco do seu LLM.	`from langchain.chains import SequentialChain`
		`template = """Dada uma refeição {meal}, forneça uma receita curta e simples`
		`SUA RESPOSTA:`
		`"""`
		`prompt_template = PromptTemplate(template=template, input_variables=['meal'`
		cadeia 2
		`dish_chain = LLMChain(llm=mixtral_llm, prompt=prompt_template, output_key='`
		`template = """Dada a receita {receita}, estime quanto tempo eu preciso para`
		`SUA RESPOSTA:`
		`"""`
		`prompt_template = PromptTemplate(template=template, input_variables=['recei`
		cadeia 3
		`recipe_chain = LLMChain(llm=mixtral_llm, prompt=prompt_template, output_key`
		cadeia geral
		`overall_chain = SequentialChain(chains=[location_chain, dish_chain, recipe_`
		`input_variables=['location'],`
		`output_variables=['meal', 'recipe', '`
		`verbose= True)`

load_summarize_chain	Este trecho de código usa a biblioteca LangChain para carregar e usar uma cadeia de sumarização com um modelo de linguagem específico e tipo de cadeia. Este tipo de cadeia será aplicado a dados da web para imprimir um resumo resultante.	`from langchain.chains.summarize import load_summarize_chain`
		`chain = load_summarize_chain(llm=mixtral_llm, chain_type="stuff", verbose=F`
		`response = chain.invoke(web_data)`
		`print(response['output_text'])n`

TextClassifier	Representa um classificador de texto simples que usa uma camada de embedding, uma camada linear oculta com ativação ReLU e uma camada linear de saída. O construtor recebe os seguintes argumentos: num_class: O número de classes para classificar. freeze: Se deve congelar a camada de embedding.	`from torch import nn`
		`class TextClassifier(nn.Module):`
		`def __init__(self, num_classes,freeze=False):`
		`super(TextClassifier, self).__init__()`
		`self.embedding = nn.Embedding.from_pretrained(glove_embedding.vecto`
		`# Um exemplo de adição de camadas adicionais: Uma camada linear e u`
		`self.fc1 = nn.Linear(in_features=100, out_features=128)`
		`self.relu = nn.ReLU()`
		`# A camada de saída que fornece as probabilidades finais para as cl`
		`self.fc2 = nn.Linear(in_features=128, out_features=num_classes)`
		`def forward(self, x):`
		`# Passar a entrada pela camada de embedding`
		`x = self.embedding(x)`
		`# Aqui você pode usar uma média simples`
		`x = torch.mean(x, dim=1)`
		`# Passar as embeddings agrupadas pelas camadas adicionais`
		`x = self.fc1(x)`
		`x = self.relu(x)`
		`return self.fc2(x)`

Pacote/Método	Descrição	Exemplo de código

Treinar o modelo	Este trecho de código descreve a função para treinar um modelo de aprendizado de máquina usando PyTorch. Esta função treina o modelo por um número especificado de épocas, rastreia-as e avalia o desempenho no conjunto de dados.	`def train_model(model, optimizer, criterion, train_dataloader, valid_datalo`
		`    cum_loss_list = []`
		`    acc_epoch = []`
		`    best_acc = 0`
		`    file_name = model_name`
		`    for epoch in tqdm(range(1, epochs + 1)):`
		`        model.train()`
		`        cum_loss = 0`
		`        for _, (label, text) in enumerate(train_dataloader):`
		`            optimizer.zero_grad()`
		`            predicted_label = model(text)`
		`            loss = criterion(predicted_label, label)`
		`            loss.backward()`
		`            torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)`
		`            optimizer.step()`
		`            cum_loss += loss.item()`
		`        #print("Loss:", cum_loss)`
		`        cum_loss_list.append(cum_loss)`
		`        acc_val = evaluate(valid_dataloader, model, device)`
		`        acc_epoch.append(acc_val)`
		`        if acc_val > best_acc:`
		`            best_acc = acc_val`
		`            print(f"Nova melhor precisão: {acc_val:.4f}")`
		`            #torch.save(model.state_dict(), f"{model_name}.pth")`
		`    #save_list_to_file(cum_loss_list, f"{model_name}_loss.pkl")`
		`    #save_list_to_file(acc_epoch, f"{model_name}_acc.pkl")`

llm_model	Este trecho de código define a função 'llm_model' para gerar texto usando o modelo de linguagem da plataforma mistral.ai, especificamente o modelo 'mitral-8x7b-instruct-v01'. A função ajuda a personalizar os parâmetros de geração e interage com os serviços de aprendizado de máquina do IBM Watson'.	`def llm_model(prompt_txt, params=None):`
		`    model_id = 'mistralai/mixtral-8x7b-instruct-v01'`
		`    default_params = {`
		`        "max_new_tokens": 256,`
		`        "min_new_tokens": 0,`
		`        "temperature": 0.5,`
		`        "top_p": 0.2,`
		`        "top_k": 1`
		`    }`
		`    if params:`
		`        default_params.update(params)`
		`    parameters = {`
		`        GenParams.MAX_NEW_TOKENS: default_params["max_new_tokens"],  # isso`
		`        GenParams.MIN_NEW_TOKENS: default_params["min_new_tokens"], # isso`
		`        GenParams.TEMPERATURE: default_params["temperature"], # isso contro`
		`        GenParams.TOP_P: default_params["top_p"],`
		`        GenParams.TOP_K: default_params["top_k"]`
		`    }`
		`    credentials = {`
		`        "url": "https://us-south.ml.cloud.ibm.com"`
		`    }`
		`    project_id = "skills-network"`
		`    model = Model(`
		`        model_id=model_id,`
		`        params=parameters,`
		`        credentials=credentials,`
		`        project_id=project_id`
		`    )`
		`    mixtral_llm = WatsonxLLM(model=model)`
		`    response  = mixtral_llm.invoke(prompt_txt)`
		`    return response`

Prompt de zero-shot	O aprendizado zero-shot é crucial para testar a capacidade de um modelo de aplicar seu conhecimento pré-treinado a novas tarefas não vistas sem treinamento adicional.	`prompt = """Classifique a seguinte afirmação como verdadeira ou falsa:`
		`            'A Torre Eiffel está localizada em Berlim.'`
		`            Resposta:`
		`"""`
		`response = llm_model(prompt, params)`
		`print(f"prompt: {prompt}\n")`
		`print(f"resposta : {response}\n")`

Pacote/Método	Descrição	Exemplo de código
	Essa capacidade é valiosa para avaliar as habilidades de generalização do modelo.	

Prompt de one-shot	Exemplo de aprendizado one-shot onde o modelo recebe um único exemplo para ajudar a guiar sua tradução do inglês para o francês. O prompt fornece um par de tradução de exemplo, "Como está o tempo hoje?" traduzido para "Comment est le temps aujourd'hui?" Este exemplo serve como guia para o modelo entender o contexto da tarefa e o formato desejado. O modelo é então encarregado de traduzir uma nova frase, "Onde fica o supermercado mais próximo?" sem mais orientações.	`params = {`
		`  "max_new_tokens": 20,`
		`  "temperature": 0.1,`
		`}`
		`prompt = """Aqui está um exemplo de traduzir uma frase do inglês para o fra`
		`  Inglês: “Como está o tempo hoje?”`
		`  Francês: “Comment est le temps aujourd'hui?”`
		`  Agora, traduza a seguinte frase do inglês para o francês:`
		`  Inglês: “Onde fica o supermercado mais próximo?”`
		`"""`
		`response = llm_model(prompt, params)`
		`print(f"prompt: {prompt}\n")`
		`print(f"resposta : {response}\n")`

Prompt de few-shot	Este trecho de código classifica emoções usando uma abordagem de aprendizado few-shot. O prompt inclui vários exemplos onde afirmações estão associadas às suas respectivas emoções.	`#parâmetros  `max_new_tokens` para 10, o que restringe o modelo a gerar res`
		`params = {`
		`  "max_new_tokens": 10,`
		`}`
		`prompt = """Aqui estão alguns exemplos de classificação de emoções em afirm`
		`  Afirmação: 'Acabei de ganhar minha primeira maratona!'`
		`  Emoção: Alegria`
		`  Afirmação: 'Não consigo acreditar que perdi minhas chaves novam`
		`  Emoção: Frustração`
		`  Afirmação: 'Meu melhor amigo está se mudando para outro país.'`
		`  Emoção: Tristeza`
		`  Agora, classifique a emoção na seguinte afirmação:`
		`  Afirmação: 'Aquele filme foi tão assustador que tive que cobrir`
		`"""`
		`response = llm_model(prompt, params)`
		`print(f"prompt: {prompt}\n")`
		`print(f"resposta : {response}\n")`

Prompt de cadeia de pensamento (CoT)	A técnica de prompting de Cadeia de Pensamento (CoT), projetada para guiar o modelo por uma sequência de etapas de raciocínio para resolver um problema.
	
	A técnica CoT envolve estruturar o prompt instruindo o modelo a “Desmembrar cada etapa do seu cálculo.” Isso incentiva o modelo a incluir etapas de raciocínio explícitas, imitando processos de resolução de problemas semelhantes aos humanos.
		`params = {`
		`  "max_new_tokens": 512,`
		`  "temperature": 0.5,`
		`}`
		`prompt = """Considere o problema: 'Uma loja tinha 22 maçãs. Eles venderam 1`
		`  Quantas maçãs há agora?’`
		`  Desmembre cada etapa do seu cálculo`
		`"""`
		`response = llm_model(prompt, params)`
		`print(f"prompt: {prompt}\n")`
		`print(f"resposta : {response}\n")`

Auto-consistência	Este trecho de código determina o resultado consistente para	`params = {`
		`  "max_new_tokens": 512,`
		`}`
		`prompt = """Quando eu tinha 6 anos, minha irmã tinha metade da minha idade.`

Pacote/Método	Descrição	Exemplo de código
	problemas relacionados à idade e gera múltiplas respostas. O dicionário 'params' especifica o número máximo de tokens para gerar respostas.	Forneça três cálculos e explicações independentes e, em seguida """ response = llm_model(prompt, params) print(f"prompt: {prompt}\n") print(f"resposta : {response}\n")

Template de prompt	Um conceito chave no LangChain, ajuda a traduzir a entrada do usuário e parâmetros em instruções para um modelo de linguagem. Isso pode ser usado para guiar a resposta de um modelo, ajudando-o a entender o contexto e gerar uma saída relevante e coerente baseada em linguagem.	model_id = 'mistralai/mixtral-8x7b-instruct-v01'
		parameters = {
		GenParams.MAX_NEW_TOKENS: 256, # isso controla o número máximo de toke
		GenParams.TEMPERATURE: 0.5, # isso controla a aleatoriedade ou criativi
		}
		credentials = {
		"url": "https://us-south.ml.cloud.ibm.com"
		}
		project_id = "skills-network"
		model = Model(
		model_id=model_id,
		params=parameters,
		credentials=credentials,
		project_id=project_id
		)
		mixtral_llm = WatsonxLLM(model=model)
		mixtral_llm

Sumarização de texto	Agente de sumarização de texto projetado para ajudar a resumir o conteúdo que você fornece ao LLM. Você pode armazenar o conteúdo a ser resumido em uma variável, permitindo o uso repetido do prompt.	content = """ O rápido avanço da tecnologia no século 21 transformou várias indús Inovações como inteligência artificial, aprendizado de máquina e a Por exemplo, ferramentas de diagnóstico impulsionadas por IA estão Além disso, plataformas de aprendizado online estão tornando a educ Esses desenvolvimentos tecnológicos não apenas estão aumentando a p """ template = """Resuma o {content} em uma frase. """ prompt = PromptTemplate.from_template(template) llm_chain = LLMChain(prompt=prompt, llm=mixtral_llm) response = llm_chain.invoke(input = {"content": content}) print(response["text"])
Resposta a perguntas	Um agente que permite ao LLM aprender com o conteúdo fornecido e responder perguntas com base no que aprendeu. Ocasionalmente, se o LLM não tiver informações suficientes, pode gerar uma resposta especulativa. Para gerenciar isso, você instruirá especificamente para responder com "Inseguro sobre a resposta" se não tiver certeza sobre a resposta correta.	content = """ O sistema solar consiste no Sol, oito planetas, suas luas, planetas Os planetas internos—Mercúrio, Vênus, Terra e Marte—são rochosos e Os planetas externos—Júpiter, Saturno, Urano e Netuno—são muito mai """ question = "Quais planetas no sistema solar são rochosos e sólidos?" template = """ Responda a {question} com base em {content}. Responda "Inseguro sobre a resposta" se não tiver certeza sobre Resposta: """ prompt = PromptTemplate.from_template(template) output_key = "answer" llm_chain = LLMChain(prompt=prompt, llm=mixtral_llm, output_key=output_key) response = llm_chain.invoke(input = {"question":question ,"content": conten print(response["answer"])
Geração de código	Um agente projetado para gerar consultas	description = """ Recupere os nomes e endereços de e-mail de todos os clientes da tab A tabela 'purchases' contém uma coluna 'purchase_date'

Pacote/Método	Descrição	Exemplo de código
	SQL com base em descrições dadas. Ele interpreta os requisitos da sua entrada e os traduz em código SQL executável.	""" template = """ Gere uma consulta SQL com base na {description} Consulta SQL: """ prompt = PromptTemplate.from_template(template) output_key = "query" llm_chain = LLMChain(prompt=prompt, llm=mixtral_llm, output_key=output_key) response = llm_chain.invoke(input = {"description":description}) print(response["query"])
Interpretação de papéis	Configura o LLM para assumir papéis específicos conforme definido por nós, permitindo que siga regras predeterminadas e se comporte como um chatbot orientado a tarefas.	role = """ mestre do jogo """ tone = "envolvente e imersivo" template = """ Você é um especialista {role}. Eu tenho esta pergunta {question Resposta: """ prompt = PromptTemplate.from_template(template) output_key = "answer" llm_chain = LLMChain(prompt=prompt, llm=mixtral_llm, output_key=output_key)
class_names	Este trecho de código mapeia rótulos numéricos para suas respectivas descrições textuais para classificar tarefas. Este código ajuda no aprendizado de máquina a interpretar a saída do modelo, onde as previsões do modelo são numéricas e devem ser apresentadas em um formato mais legível para humanos.	class_names = {0: "negativo", 1: "positivo"} class_names

read_and_split_text	Envolve abrir o arquivo, ler seu conteúdo e dividir o texto em parágrafos individuais. Cada parágrafo representa uma seção das políticas da empresa. Você também pode filtrar quaisquer parágrafos vazios para limpar seu conjunto de dados.	def read_and_split_text(filename):
		with open(filename, 'r', encoding='utf-8') as file:
		text = file.read()
		# Dividir o texto em parágrafos (divisão simples por caracteres de nova
		paragraphs = text.split('\n')
		# Filtrar quaisquer parágrafos vazios ou entradas indesejadas
		paragraphs = [para.strip() for para in paragraphs if len(para.strip())
		return paragraphs
		
		Leia o arquivo de texto e divida-o em
		
		paragraphs = read_and_split_text('companyPolicies.txt')
		paragraphs[0:10]

encode_contexts	Este trecho de código codifica uma lista de textos em embeddings usando content_tokenizer e context_encoder. Este código ajuda a iterar	def encode_contexts(text_list):
		# Codifica uma lista de textos em embeddings
		embeddings = []
		for text in text_list:
		inputs = context_tokenizer(text, return_tensors='pt', padding=True,
		outputs = context_encoder(**inputs)
		embeddings.append(outputs.pooler_output)
		return torch.cat(embeddings).detach().numpy()

Pacote/Método	Descrição	Exemplo de código
	por cada texto na lista de entrada, tokeniza e codifica, e então adiciona o pooler_output à lista de embeddings. Os embeddings resultantes são armazenados nas variáveis context_embeddings e geram embeddings a partir de dados textuais para várias aplicações de processamento de linguagem natural (NLP).	**você agora codificaria esses parágrafo** `context_embeddings = encode_contexts(paragraphs)`

import faiss	FAISS (Facebook AI Similarity Search) é uma biblioteca eficiente desenvolvida pelo Facebook para busca de similaridade e agrupamento de vetores densos. FAISS é projetada para busca de similaridade rápida, o que é particularmente valioso ao lidar com grandes conjuntos de dados. É altamente adequada para tarefas em processamento de linguagem natural onde a velocidade de recuperação é crítica. Ela lida efetivamente com grandes volumes de dados, mantendo o desempenho mesmo à medida que os tamanhos dos conjuntos de dados aumentam.	`import faiss`
		Converter lista de arrays numpy em um
		`embedding_dim = 768  # Isso deve corresponder à dimensão dos seus embedding`
		`context_embeddings_np = np.array(context_embeddings).astype('float32')`
		Criar um índice FAISS para as embeddi
		`index = faiss.IndexFlatL2(embedding_dim)`
		`index.add(context_embeddings_np)  # Adiciona as embeddings de contexto ao í`

search_relevant_contexts	Este trecho de código é útil para buscar contextos relevantes para uma determinada pergunta. Ele tokeniza a pergunta usando o question_tokenizer, codifica a pergunta usando question_encoder e pesquisa em um índice para recuperar o contexto relevante com base na embedding da pergunta.	`def search_relevant_contexts(question, question_tokenizer, question_encoder`
		`"""`
		`Busca os contextos mais relevantes para uma determinada pergunta.`
		`Retorna:`
		`tuple: Distâncias e índices dos k contextos relevantes.`
		`"""`
		`# Tokeniza a pergunta`
		`question_inputs = question_tokenizer(question, return_tensors='pt')`
		`# Codifica a pergunta para obter a embedding`
		`question_embedding = question_encoder(**question_inputs).pooler_output.`
		`# Pesquisa no índice para recuperar os k contextos relevantes`
		`D, I = index.search(question_embedding, k)`
		`return D, I`

generate_answer_without_context	Este trecho de código gera respostas usando o prompt inserido sem exigir contexto adicional. Ele tokeniza as perguntas de entrada usando o tokenizer, gera o texto de saída usando o modelo e decodifica o texto gerado para obter a resposta.	`def generate_answer_without_context(question):`
		`# Tokeniza a pergunta de entrada`
		`inputs = tokenizer(question, return_tensors='pt', max_length=1024, trun`
		`# Gera a saída diretamente da pergunta sem contexto adicional`
		`summary_ids = model.generate(inputs['input_ids'], max_length=150, min_l`
		`# Decodifica e retorna o texto gerado`
		`answer = tokenizer.decode(summary_ids[0], skip_special_tokens=True)`
		`return answer`

Pacote/Método	Descrição	Exemplo de código

Gerando respostas com contextos DPR	Respostas são geradas quando o modelo utiliza contextos recuperados via DPR, que se espera que melhorem a relevância e a profundidade da resposta:	`def generate_answer(contexts):`
		`  # Concatena os contextos recuperados para formar a entrada para o BART`
		`  input_text = ' '.join(contexts)`
		`  inputs = tokenizer(input_text, return_tensors='pt', max_length=1024, tr`
		`  # Gera a saída usando o BART`
		`  summary_ids = model.generate(inputs['input_ids'], max_length=150, min_l`
		`  return tokenizer.decode(summary_ids[0], skip_special_tokens=True)`

função aggregate_embeddings	A função aggregate_embeddings recebe índices de tokens e suas respectivas máscaras de atenção, e usa um modelo BERT para converter esses tokens em embeddings de palavras. Em seguida, filtra as embeddings de tokens preenchidos com zero e calcula a média das embeddings para cada sequência. Isso ajuda a reduzir a dimensionalidade dos dados enquanto retém as informações mais importantes das embeddings.	`def aggregate_embeddings(input_ids, attention_masks, bert_model=bert_model)`
		`  """`
		`  Converte índices de tokens e máscaras em embeddings de palavras, filtra`
		`  e agrega calculando a média das embeddings para cada sequência de entra`
		`  """`
		`  mean_embeddings = []`
		`  # Processa cada sequência no lote`
		`  print('número de entradas',len(input_ids))`
		`  for input_id, mask in tqdm(zip(input_ids, attention_masks)):`
		`  input_ids_tensor = torch.tensor([input_id]).to(DEVICE)`
		`  mask_tensor = torch.tensor([mask]).to(DEVICE)`
		`  with torch.no_grad():`
		`  # Obtém as embeddings de palavras do modelo BERT`
		`  word_embeddings = bert_model(input_ids_tensor, attention_mask=m`
		`  # Filtra as embeddings nas posições onde a máscara é zero`
		`  valid_embeddings_mask=mask_tensor[0] != 0`
		`  valid_embeddings = word_embeddings[valid_embeddings_mask,:]`
		`  # Calcula a média das embeddings filtradas`
		`  mean_embedding = valid_embeddings.mean(dim=0)  mean_em`
		`  # Concatena as médias das embeddings de todas as sequências no lote`
		`  aggregated_mean_embeddings = torch.cat(mean_embeddings)`
		`  return aggregated_mean_embeddings`

text_to_emb	Projetado para converter uma lista de strings de texto em suas respectivas embeddings usando um tokenizer predefinido.	`def text_to_emb(list_of_text,max_input=512):`
		`  data_token_index  = tokenizer.batch_encode_plus(list_of_text, add_speci`
		`  return question_embeddings`

process_song	Converte tanto as perguntas de adequação predefinidas quanto as letras da canção em "embeddings RAG" e mede a similaridade entre elas para determinar a adequação.	`import re`
		`def process_song(song):`
		`  # Remove quebras de linha da canção`
		`  song_new = re.sub(r'[\n]', ' ', song)`
		`  # Remove apóstrofos da canção`
		`  song_new = [song_new.replace("\'", "")]`
		`  return song_new`

RAG_QA	Este trecho de código realiza perguntas e respostas usando embeddings de perguntas e fornece embeddings. Ele ajuda a remodelar os resultados para processamento, ordenando os índices em ordem decrescente e imprimindo as 'n-	`def RAG_QA(embeddings_questions, embeddings, n_responses=3):`
		`  # Calcula o produto escalar entre as embeddings de perguntas e as embed`
		`  dot_product = embeddings_questions @ embeddings.T`
		`  # Remodela os resultados do produto escalar para um tensor 1D para proc`
		`  dot_product = dot_product.reshape(-1)`
		`  # Ordena os índices dos resultados do produto escalar em ordem decresce`
		`  sorted_indices = torch.argsort(dot_product, descending=True)`
		`  # Converte os índices ordenados em uma lista para iteração mais fácil.`
		`  sorted_indices = sorted_indices.tolist()`
		`  # Imprime as 'n_responses' principais da lista ordenada, que correspond`
		`  for index in sorted_indices[:n_responses]:`
		`  print(yes_responses[index])`

Pacote/Método	Descrição	Exemplo de código
	respostas' com base nos maiores valores de produto escalar.	

model_name_or_path	Este trecho de código define o nome do modelo como 'gpt2' e inicializa o token e o modelo usando o modelo GPT-2. Neste código, adiciona tokens especiais para preenchimento mantendo o comprimento máximo da sequência em 1024.	`# Define o nome ou caminho do modelo`
		`model_name_or_path = "gpt2"`
		
		`# Inicializar tokenizer e modelo`
		
		`tokenizer = GPT2Tokenizer.from_pretrained(model_name_or_path, use_fast=True`
		`model = GPT2ForSequenceClassification.from_pretrained(model_name_or_path, n`
		
		`# Adicione tokens especiais se necessári`
		
		`tokenizer.pad_token = tokenizer.eos_token`
		`model.config.pad_token_id = model.config.eos_token_id`
		
		`# Defina o comprimento máximo`
		
		`max_length = 1024`

add_combined_columns	Este trecho de código combina o prompt com as respostas escolhidas e rejeitadas em um exemplo de conjunto de dados. Ele combina com o 'Human:' e 'Assistant:' para clareza. Esta função modifica cada exemplo na divisão 'train' do conjunto de dados, criando novas colunas 'prompt_chosen' e 'prompt_rejected' com o texto combinado.	`# Defina uma função para combinar 'prompt' com as respostas 'chosen' e 'rej`
		`def add_combined_columns(example):`
		`    # Combine 'prompt' com a resposta 'chosen', formatando com os rótulos "`
		`    example['prompt_chosen'] = "\n\nHuman: " + example["prompt"] + "\n\nAss`
		`    # Combine 'prompt' com a resposta 'rejected', formatando com os rótulos`
		`    example['prompt_rejected'] = "\n\nHuman: " + example["prompt"] + "\n\nA`
		`    # Retorne o exemplo modificado`
		`    return example`
		
		`# Aplique a função a cada exemplo no spl`
		
		`dataset[‘train’] = dataset[‘train’].map(add_combined_columns)`

RetrievalQA	Este trecho de código cria um exemplo para 'RetrievalQA' usando um modelo de linguagem e um recuperador de documentos.	`qa = RetrievalQA.from_chain_type(llm=flan_ul2_llm,`
		`                                  chain_type="stuff",`
		`query = "qual é a política móvel?"`
		`qa.invoke(query)`

Skills Network