**Tutorial básico de Langchain - RAG - Retrieval Augmented Generation - Vídeo 2**

**Transcrito por** [**TurboScribe.ai**](https://turboscribe.ai/pt/?ref=docx_export_upsell)**.** [**Atualize para Ilimitado**](https://turboscribe.ai/pt/subscribed?ref=docx_export_upsell) **para remover esta mensagem.**

Quer aprender tudo sobre RAG? Fica comigo nesse vídeo. Oi, pessoal, aqui é o Rafa. Nesse vídeo, é o segundo vídeo da nossa série de Lankchain, e a gente vai trazer aqui um assunto muito pedido.

Muitas pessoas me chamaram na DM e pediram para falar de RAG, falar de RAG, e hoje a gente vai começar nessa jornada. Esse é o vídeo introdutório. Eu não vou explicar para vocês todas as nuances de RAG.

RAG pode ser já uma área de estudos por si só, e a ideia é mostrar para vocês que estão começando a brincar com a biblioteca Lankchain, como a gente vai fazer essa integração para fazer o RAG. Mas o que é esse tal de RAG? RAG é Retrieval Augmented Generation, que basicamente é o seguinte, é você pegar os seus dados proprietários e utilizar esses dados proprietários para responder perguntas com qualquer LLM, como short GPT, como LLM, Gemini, qualquer outro LLM que vai sair no futuro. E qual é a importância disso? Nem todos os tipos de dados podem ser enviados na API da OpenAI.

Nem todos os tipos de dados, dados sensíveis, por exemplo, se você está fazendo uma aplicação jurídica, que tem algum processo, dados sigilosos, você não quer que esses dados vão parar nas bases de dados da OpenAI. Se você tem algumas questões de sigilo, seja industrial, criativo, se você está desenvolvendo um projeto novo, uma música nova, você quer auxiliar as LLMs, você quer criar aplicações com LLMs, mas você não pode depender exclusivamente das APIs públicas, certo? E essa é uma estratégia para você misturar a parte do contexto justamente com a parte da LLM. Você tem o melhor dos dois mundos.

E existem várias técnicas para a gente fazer isso. Existem vários LLMs. É legal que você pode depois testar aquela modularidade que a gente viu lá no vídeo sobre o LAMY Chain, que a gente tem a ideia das cadeias, a gente sempre vai estar juntando processamento nessas cadeias e existem componentes específicos já para extrair dados da web, para fazer parças de PDF, parça de PDF é literalmente, você pode pegar um PDF de alguma coisa que interessa, carregar nele facilmente usando esses componentes que a LAMY Chain te propõe.

Uma vez que está carregado, você vai pegar esses documentos e pode fazer perguntas para o seu PDF, perguntas para o seu documento. Por exemplo, de hoje, a gente vai utilizar um componente que vai fazer um web scrapping numa página da Wikipedia para ajudar a gente a responder uma pergunta que o ChatGPT, por exemplo, não consegue responder. A ideia de hoje é mostrar para vocês um tutorial básico de como se virar, como fazer um HAG básico.

E na medida que a gente foi evoluindo com essa playlist, a ideia aqui é trazer aplicações cada vez mais profissionais. A única coisa que às vezes demora um pouco para fazer esses vídeos porque quanto mais profissional é a aplicação, mais trabalho dá, mais coisas para explicar, mas a gente está aprendendo passo a passo aqui, acho que você não está com pressa, super aconselho você também pegar a documentação da LAMY Chain, que apesar de ela mudar bastante, essa é uma parte, já aviso, bem chata de lidar com a LAMY Chain, praticamente toda semana tem uma função nova, eles estão depreciando funções que não existem mais, e você tem que estar correndo atrás do prejuízo e entendendo como é o novo jeito correto de fazer as coisas. Esse é um preço que se paga por você estar na fronteira da tecnologia, no estado da arte da ciência, é justamente esse, é ser o pioneiro, é você lidar com essas questões no começo.

E você vai lidar com muita função quebrada, muita função que semana passada você aprendeu a fazer de um jeito, na semana seguinte não é, essa sensação de frustração é normal. E quanto mais cênico você fica na carreira, é isso que eu espero de você, que o mercado espera de você, que você saiba lidar com essa diversidade, que você saiba entender o que está acontecendo. Como esse movimento de LLM tem menos de, diria, dois, três anos, a maturidade da biblioteca está longe de ser madura, e cada semana a gente tem novidades diferentes, tem maneiras de utilizar a biblioteca diferente, e a gente tem que lidar com isso.

E a minha ideia aqui também na playlist é disseminar esse conhecimento, eu vi que tem pouco conteúdo falando sobre LAMY Chain aqui no Brasil, quero trazer algo interessante aqui para vocês que vocês possam extrair valor. Mas, cheio de teoria, vamos agora para a prática. O nosso exemplo vai ser bem simples, né? O que eu vou fazer? Eu vou pegar aquela função que eu criei para mostrar para vocês como funcionava a LAMY Chain básica no primeiro tutorial, vou reformular ela para ela responder perguntas em relação ao Oscar.

Simples assim. Eu vou passar o nome do filme, vou passar o ano, e eu vou perguntar para ela quantos Oscars, por exemplo, o filme A Vida é Bela ganhou. Quero essa tipo de informação.

Só que tem um porém, eu vou usar o LLN, o ChatGPT 8.5 Turbo, que ele foi atualizado até o ano passado. Então, ele não tem informações ainda do Oscar desse ano. Então, se eu perguntar para ele, por exemplo, quantos Oscars o Oppenheimer ganhou, estamos em 2024, quando estou gravando esse vídeo, o ChatGPT 3.5 não vai saber responder.

Ele vai falar, não sei, isso não aconteceu ainda. Essa é a informação que ele tem. E o que eu vou fazer? Eu vou criar uma outra estrutura, junto com o LAMY Chain também, que vai fazer um web scrapping de uma parte da Wikipédia sobre o filme Oppenheimer, que já tem as informações de quantos Oscars ele ganhou.

E eu vou, utilizando a LAMY Chain, pegar esse web scrapping, pegar o meu LLN, que eu instanciei, e vou juntar eles, e vou fazer com o modelo 3.5, que não tem acesso a essa informação na base de dados dele, de treino, responder essa pergunta. Essa é um teste conceitual para vocês mostrarem, entenderem o que o HAG pode fazer por vocês. Nesse processo, eu tenho que usar um Vector Database.

Eu vou usar o FICE, que é uma alternativa open source. Também existe o ChromeDB. Existem outras alternativas, já a Enterprise, que te oferecem outras coisas de Vector Database.

E isso também é um outro mundo à parte. Como você criar esse Vector Database, como você fazer a gestão dele, onde você vai alocar ele. Mas aqui, eu vou fazer tudo local.

Eu vou criar um Vector Database para fazer o parte de uma parte da Wikipédia. Mas você pode imaginar que você estivesse fazendo uma aplicação profissional. Eu gosto de usar o exemplo do escritório de advocacia, que tem processos sigilosos.

Então, por exemplo, ele não usaria o LLM da OpenAI, mas poderia usar um LLM, por exemplo, que é aberto no Facebook, que ele não envia dados para fora, ele usa a API interna, você consegue baixar o LLM no seu servidor, ter controle e gestão sobre esse LLM. Você pode utilizar o LLM treinado para responder perguntas do processo para um advogado que está querendo se inteirar do processo, por aí vai. Ao invés de ele ler 500 mil páginas que todo advogado já deve ter lidado na vida do processo, ele pode simplesmente perguntar em um prompt o que aconteceu com o fone de tal, qual a data que aconteceu.

E o mais legal disso é que o parser que ele faz para o Vector Database, você consegue achar a referência. Então, não é uma informação jogada. Ele não vai te dar só, por exemplo, o Oppenheimer ganhou sete Oscars.

Não, ele vai te dar o Oppenheimer ganhou sete Oscars e eu estou te respondendo isso baseado nesse contexto que você me deu. Que, no caso, por exemplo, da aplicação de direito, o advogado podia ir lá no processo e ver o que de fato está no processo, se o chat não alucinou e por aí vai. Então, o HAG tem aplicações muito poderosas.

E é esse gostinho que eu quero dar aqui para vocês hoje. Então, vamos lá para o tutorial, botar a mão na massa e entender como é que a gente faz esse HAG aqui utilizando a biblioteca LangChain. Pessoal, aqui eu abri nossa aplicação que a gente fez no tutorial zero de LangChain.

Exatamente o estado de como estava, só dando um recordado aqui. A gente usou o LLM da OpenAI mesmo, a gente usou o prompt template e usou o LLM chain para amarrar, para juntar as duas coisas aqui. O prompt está aqui nessas duas linhas.

Aqui a minha chain está juntando o meu LLM com esse prompt. E aqui eu consigo enviar via parâmetros o animal, a cor e ele me responde aqui. Baseando um pouco nessa aplicação, eu criei já essa versão aqui para juntar.

Como é a mesma coisa, eu não perdi muito tempo refazendo esse espaço com você. Mas, basicamente, tem praticamente as mesmas dependências aqui. Lembrando que o luz.env é para a gente expandir a nossa chave da OpenAI que está aqui no .env. Se você tem dúvida do que está acontecendo aqui, eu sugiro que você busque lá no primeiro vídeo como é que você entra e vai no site da OpenAI, como você gera essa chave secreta e como você carrega ela aqui.

Mas é basicamente essa dependência aqui. Temos aqui os requirements de tudo que a gente vai precisar aqui para instalar. Depois eu libero isso aqui para vocês puderem replicar.

E eu só mudei a função, não é mais animais, é a função oscar. Aqui ela recebe o nome de um filme como parâmetro. Opa, aqui está errado, né? Aqui precisa do ano também, né? Filme e ano.

Vou juntar o meu prompt com o meu LLM através dessa chain. Vou chamá-la de oscarchain e vou passar para ela aqui o filme e o ano que eu recebi ela como um parâmetro. Tem uma diferença aqui que eu também tirei o LLM dessa função.

Vocês já vão entender daqui a pouco por quê. Porque eu quero usar esse LLM também na minha outra aplicação, na minha revisão com o HAG. Eu vou usar o mesmo objeto instanciado aqui para vocês verem que não tem diferença, que é ele mesmo, não tem nenhum problema.

Aqui eu instancei o modelo que eu quero, o 3.1.5 Turbo, justamente porque é o modelo que é o default dele, mas deixei explícito aqui para vocês verem que é o modelo antigo, não é o GPT-4, ele não tem acesso às informações, principalmente do Oscar, do que aconteceu em 2024, que foi recente, inclusive, na data que eu estou gravando o vídeo aqui. Esse preprocess aqui vocês não precisam se preocupar. Como tinha uns warnings acontecendo no outro vídeo, justamente por causa de algumas bibliotecas que estão sendo atualizadas, algumas funções que não tem mais, eu criei aqui umas funções para ele não aparecer para a gente, mas não influenciar nada no processo.

Pode se considerar isso aqui, só encapsulei ele aqui para ficar mais bonitinho. Então, basicamente, quando a gente rodar esse arquivo, ele vai chamar para a função Oscar, vai passar aqui o parâmetro OpenRM, vai passar o 1.2.24 e vai passar esse LLM que foi instanciado aqui. Inclusive, pode colocar aqui embaixo, não deixa lá em cima.

E vai printar a resposta aqui dentro do texto. E o que eu espero é que, como eu estou usando esta versão aqui do chat de GPT, ele não vai conseguir responder. Então, vamos lá fazer esse teste.

Python, chatGPT.py. Ele vai demorar um pouco processando. E aí, está a resposta. Não é possível determinar quantos Oscars o filme OpenRM e 1.2.24, pois ainda estamos chegando nesse ano.

E não há como prever os resultados de premiações futuras. É o que a gente esperava. A gente sabe que o chat de GPT-3, principalmente esse modelo, ele tem um certo delay.

E o que a gente vai fazer agora é exatamente aquele processo que eu falei para vocês. Eu vou aqui na página Wikipedia, aqui essa página atualizada, eu vou passar esse link para um web scrapping. Ele vai fazer o web scrapping dessa página, vai colocar essa página dentro de um vector database.

Eu estou usando o Files. E a partir desse vector database, ele vai conseguir usar esse texto como contexto para gerar a resposta. Então, a primeira etapa desse processo é justamente fazer esse parser.

E eu vou criar aqui um outro arquivo chamado vectorDB.py. Justamente para fazer este processo. A primeira dependência que a gente vai precisar é o web parser. From langchain.

Outline to read. Alguns comentários sobre essa primeira linha. Essa parte de langchain community são pessoas que não são do time principal do core da langchain que criam essas ferramentas.

Então, alguém, algum desenvolvedor que já precisava, estava interessado em ajudar a comunidade, ele já criou esse web parser. Como dependência, ele tem o BeautifulSoup, que é uma biblioteca de Python já para fazer o web scrapping. Então, ela já está pronta para isso.

E a gente vai testar essa parte agora, ver como é que funciona. Então, eu vou criar um loader aqui, vou instanciar esse cara e vou passar uma URL para ele. Vou copiar, inclusive, essa URL aqui do Fionn, passar como string e vou colocar o resultado dele dentro de uma variável chamada docs.

Vou fazer o loader, vou fazer esse loader carregar. E vou dar um print aqui para vocês verem o que tem aqui dentro. Então, vamos aqui, python, vectorDB.py. Ele vai fazer esse web scrapping e vai printar aqui.

Temos aqui, se vocês verem, é muita informação, porque é uma página inteira de Wikipédia. Aqui está o Fionn, está demais. Temos aqui, ele está dentro, ele me passou uma lista, tem um objeto document, que tem a propriedade page content aqui.

E deve ter uns metadados aqui, mostrando para ele a source, por aí vai. Está vendo aqui a source, o link que ele usou, o título, bem interessante, a língua que está. É um parser bem poderoso, ele já conseguiu te dar várias informações.

Isso aqui vai ser importante, porque, de novo, como eu falei, não é só responder, existe como você criar algumas aplicações onde você responde a aplicação e ele te dá a referência para você ir lá e conferir, certo? Mas esse é o primeiro passo, isso aqui não é um vector database. Eu só simplesmente fui lá na Wikipédia, fiz o web scrapping e coloquei na memória. Está dentro desse docs aqui, toda essa documentação.

Eu quero pegar esse documento aqui e eu quero colocar agora dentro de um vector database. E a gente tem alguns passos para isso. O primeiro passo dele é transformar todo esse emaranhado de textos em embeddings.

Se você tem alguma dúvida do que é um embedding, eu estou fazendo uma playlist aqui também, que eu estou lendo os principais papers sobre NLP. E esses papers sobre NLP, eles trazem uma grande revisão bibliográfica e explica o que é um embedding. Mas basicamente é pegar todas essas strings e transformar em um vector.

Esse vector tem um significado semântico, no seguinte sentido. Palavras próximas vão ter vetores próximos. Palavras diferentes vão ter vetores diferentes.

Então, se eu falar a palavra rottweiler, pincher, são bernardo, essas palavras vão estar próximas porque são rastros de cachorro. Diferente se eu falar a palavra carburador, biela, pistão, que são componentes de um motor mecânico, certo? Pode acontecer da palavra pincher e biela estarem no mesmo contexto? Pode, mas é muito raro. É muito mais crível a gente imaginar que as palavras biela, visabrequim e outras componentes do motor apareçam juntos nos textos da internet.

Se a gente lembrar como é que é treinado os modelos LLMs, eles usam textos da internet, faz todo o sentido do mundo a gente usar essa estatística, certo? Então, o que a gente vai fazer aqui agora? A gente vai transformar esses textos em embeddings, vai quebrar em pequenos pedaços e vai armazenar ele num vector database. Para isso, eu também tenho já bibliotecas que fazem isso, que fazem essa transformação. Vou usar novamente o da OpenAI, justamente porque é o que está mais, vamos dizer assim, avançado e tudo mais, mas se eu tivesse, se eu importasse, por exemplo, com assistindo os dados, eu poderia usar a biblioteca IEMA, que ela não tem conexão com a OpenAI, não envia dados para lá e por aí vai.

Você tem uma questão de compliance para usar. Tem toda a comunidade open source aí para isso. Esse é o embeddings.

Além do embeddings, eu vou pegar aqui de novo o da Community, só que agora eu vou pegar do vector stores. Esse vector stores são basicamente bibliotecas que fazem esse parser. Então aí você vai achar o ChromoDB, você vai achar o Files, você vai achar diversas soluções de bancos de dados vetoriais, certo? Então, recapitulando, esse cara aqui é para transformar as coisas em embeddings, esse aqui é de fato o vector database.

E uma última dependência que a gente precisa é uma dependência para quebrar esse texto gigante que está aparecendo aqui embaixo em pequenos pedaços, pequenos chunks, como se fala em inglês. É interessante para o LLM a gente colocar um número limitado de texto, justamente porque ele tem uma limitação de tokens. Então, se a gente passar um número gigantesco, a gente não vai conseguir fazer o parser, vai ter problemas com a API.

De novo, a gente já tem um componente aqui que faz isso para a gente. Esse cara aqui vai quebrar o texto em pequenos chunks, esse vai transformar em embeddings cada um desses chunks e esse aqui vai armazenar num vectorDB, tá? Já temos todas as nossas dependências aqui e agora vamos utilizá-las. Vamos tirar esse cara daqui.

Deixa eu abaixar aqui e organizar um pouco a tela, não precisamos mais do restante de informação. O que eu vou fazer agora? Eu vou criar uma variável embeddings e vou instanciar esses embeddings aqui. Lembrando que se você não setou aqui o seu in, ele vai reclamar, ele vai chear.

Como você está usando os embeddings da OpenAI, você precisa ter uma chave de API, porque isso é cobrado. Vou criar o teste split agora, que é justamente instanciar esse outro cara aqui para ele justamente quebrar o meu texto em pequenos chunks e vou passar os meus documentos para ele, que é justamente testsplitter.split documents. E vou passar esses docs.

Se você lembrar, aqui dentro é o resultado desse print gigantesco aqui embaixo, esse objeto. Como ele já foi feito na comunidade numchain, essas coisas já se conversam, então não tem nenhum problema de interoperabilidade. Ele já espera receber esse tipo de docs aqui, mas aqui você não precisa passar só o webscrap, não.

Se você carregou ele numa variável document, aqui... Deixa eu ver aqui no início, vou dar um pouco mais. Opa, foi demais agora. Se você colocou ele num document, está vendo aqui embaixo, você pode passar um PDF, você pode passar a transcrição de um vídeo, tem diversos componentes.

É muito legal a gente explorar o que a comunidade está fazendo aí, porque dá a ideia de aplicações muito interessantes para a gente, ver o que tem nesses repositórios aqui. É um bom exercício para vocês fazerem depois. Mas beleza, já fiz todo esse processo de quebrar em partes, agora eu tenho, de fato, que transformar eles em vetores e mandar eles para o meu vector database.

Vou instanciar o files aqui. E from documents. Vou passar os meus documents e vou passar a minha função de embeddings.

O que esse cara vai fazer aqui? Ele vai pegar todos esses documentos que ele quebrou, tem vários pedacinhos de documentos aqui dentro. Aquele testando Wikipedia foi quebrado em vários pedacinhos nessa linha. E ele vai utilizar os embeds que você instanciou da linha 9 aqui em cima para transformar cada string dessa em um vetor, certo? E vai armazenando a memória.

A gente não vai criar um banco de dados, mas sim, isso aqui é um banco de dados. Então a gente consegue criar ele no modelo cliente-servidor, a gente consegue deixar ele armazenado em um local seguro, na nuvem. Tudo que você já aprendeu sobre banco de dados é válido para um banco de dados vetorial da mesma maneira.

A gente não vai fazer isso aqui porque é um exemplo, eu quero mostrar para vocês toda essa trajetória, como funciona esse primeiro exercício de HAG, mas fique sabendo que profissionalmente a gente não faz isso. A gente quer que o banco de dados persista, justamente por isso que a gente precisa que ele esteja pendurado em algum servidor seguro. A última questão que a gente vai fazer aqui agora, a gente vai fazer o seguinte, vou englobar esse cara aqui em uma função, vou chamar isso aqui de def, deixa eu pensar, url to vector, vou tirar esse cara daqui, vou usar ele como variável, para o card vou ficar um pouquinho mais organizado, a gente vai criar um outro arquivo, não vai chamar ele aqui, só essa instância, return, e eu vou querer retornar o retriever.

O retriever é como se fosse o recuperar, o recuperador, se eu não me engano, essa é a tradução da palavra em inglês. E o que é o recuperador? É você buscar as informações. E esse vector aqui, o próprio faz, esse objeto aqui, ele te dá um método para transformar ele nesse componente, o retriever.

Acho que eu estou meio aqui, se eu não me engano. Então, eu vou devolver esse componente. E o que esse cara vai fazer? Eu passo uma url, ele vai fazer o webscrapping para mim, vai carregar, vai quebrar em pequenos pedaços de texto, vai transformar em embeddings e alocar num vetor database, e vai devolver esse vetor database já como retriever, que é o método que eu espero passar ele para um LLN via lang chain, ele me devolva o contexto da minha aplicação, a parte mais interessante de acordo com o meu prompt.

Tem uma matemática muito interessante por trás, como ele faz isso, a gente não vai entrar no mérito, mas basicamente o que ele vai fazer? Ele vai pegar o teu prompt, a tua pergunta, vai transformar num embedding também, utilizando essa mesma técnica, vai utilizar uma similaridade de cosseno para achar qual é o pedacinho daqui que faz mais sentido com o que você está perguntando, e vai buscar só esse pedacinho. Isso é o que o retriever vai devolver. É exatamente o que essa função retriever faz aqui.

Então, a gente já tem essa parte da solução, essa função eu passo uma url e ela devolve, talvez melhor chamar de retriever. Retriever fica mais correto, acredito. Vou agora fazer o seguinte, ao invés de continuar trabalhando aqui, vou criar um outro arquivo, o arquivo rag, para a gente diferenciar.

rag.py. E agora, de novo, o que a gente vai fazer aqui? A gente vai criar agora um template, vou criar novamente uma chain, só que ao invés de criar a chain, eu vou pegar essa minha chain e passar para uma retriever chain. Então, são vários elos de uma cadeia. Lembra, chain é uma corrente, certo? Elas estão juntas.

Eu vou unir primeiro um llm com um prompt, e essa instância, depois eu unir com o retriever, para me dar a resposta que eu quero. Então, agora eu vou acelerar essa parte aqui do vídeo, para não entediar vocês, mas basicamente eu vou simplesmente carregar algumas bibliotecas que a gente vai usar. Bom, aqui o que eu fiz? Eu vou mostrar para vocês um outro jeito de se criar prompts.

Aqui a gente criou ele utilizando esse prompt template, e aqui eu vou usar outra estratégia, o chat prompt template. Ele tem algumas sutis diferenças, mas não são muitas, não. Essa do chat é justamente quando eu quero criar agentes, que eu quero interagir com ele, que eu quero treinar aquela carinha de chat de IPT que a gente conhece, sabe, que a gente fala uma pergunta, ele age como um ser humano respondendo.

Basicamente, essa diferença é sutil, mas existe. Ele vai tentar te responder como se fosse um ser humano. A outra, ele vai te dar uma resposta seca, porque ele está interessado só na informação.

E essa aqui é uma chain genérica. Repare que ele está bem do mesmo módulo, que ela vai, basicamente, essa aqui a gente vai usar para juntar o LLM com o prompt que a gente criou aqui, e depois tem essa outra que é para juntar o nosso retriever com esse elemento que a gente criou aqui a partir dessa chain. Então, é sempre juntando componentes.

Essa é a lógica do lang chain. E aqui eu vou fazer explicitamente para vocês verem. Eu vou primeiro importar essa função nossa do URL to Retriever.

Então, from vector b, import URL to Retriever. E vou fazer o seguinte. Eu não vou criar um LLM novo aqui.

Eu vou fazer questão de importar esse LLM aqui, que está nesse arquivo, que está instanciado nesse arquivo, para a gente ter certeza que ele está usando o mesmo LLM com os mesmos parâmetros. Então, aqui eu vou simplesmente vir aqui e vir from. Chat.

Opa. GPT, import LLM. Então, não vou instanciar nada.

Ele vai ser instanciado nesse objeto aqui e vou colocá-lo aqui dentro. Vou usar esse LLM. Então, vou chamar aqui document chain igual.

Vamos começar a juntar as coisas. Esse stuff, eu vou usar aqui o LLM. Ah, espera aí.

Antes de juntar alguma coisa, você precisa criar o prompt primeiro, né? Fui mais apressado. Vamos pegar aqui o prompt. Eu vou utilizar esse primeiro objeto aqui.

Só uma outra maneira de fazer isso. Eu vou usar o método from template. Esse método from template, ele exige apenas duas coisas, né? Eu vou criar um template a partir de texto aqui e necessariamente eu preciso de duas variáveis.

Tem que ser desse jeito, senão não vai funcionar. Você pode ir lá depois da documentação, dar uma bugada, copiar e colar esse valor. Você vê que esse é um template que precisa ter um contexto e precisa ter um input, tá? Isso é muito importante porque a gente vai usar isso.

Essas variáveis vão ser utilizadas na nossa estratégia de RAG, né? A gente vai passar uma pergunta e a gente quer que tenha esse contexto para ter a resposta, né? Então, a gente tem aqui input. E só para garantir que ele não vai nos ensinar, eu vou colocar aqui, ó. Responda a pergunta com base apenas no contexto. Se você escrever em inglês, em francês, ele tem esse suporte também, mas estamos aqui fazendo um tutorial em língua portuguesa, por que não, né? Vamos escrever o nosso prompt aqui em português.

Ele vai receber um contexto. Que vai vir de onde? A gente vai vir desse retriever, vai vir exatamente daqui, ó. Esse cara aqui que vai me dar o contexto, como eu falei para vocês. E o input é a minha pergunta.

Eu vou passar uma pergunta para ele, ele vai pegar a minha pergunta que está dentro desse input aqui, vai transformar no embedding, vai usar esse retriever aqui para pegar o texto que está no files mais próximo possível e vai carregá-lo nesse lugar aqui como contexto, tá? Então, a partir disso, bom, agora o que eu tenho que fazer? Eu tenho que passar o meu retriever para ele, certo? Então, eu vou usar aqui a minha função urlToRetriever. E agora sim, eu vou novamente pegar o link aqui da Wikipédia. E agora eu vou criar a minha chain, eu vou combinar essas coisas.

Eu já tenho aqui o meu LLM que veio aqui, ó, do chat.pt, combinado com esse prompt aqui em cima, certo? Dentro da mesma corrente. Agora eu vou pegar essa corrente e quero combinar com esse retriever. E eu vou usar esse objeto que está aqui, createRetrievalChain, certo? Abre e fecha ele, retriever, e vou passar o documentChain, certo? Essa é a lógica por trás da LangChain, a gente sempre vai estar colocando elos nessa corrente, em linguagem filosófica até, né? E aqui a gente praticamente já tem tudo o que a gente precisa, né? Vou criar a nossa response e vou agora aqui simplesmente chamar ela como invoke.

Invoke era o antigo run, se você estudou LangChain há um tempo atrás, antigamente eles chamavam isso de run. Mas acho que tem, tipo, semanas, sabe, que eles trocaram isso aqui para invoke. Aqui eu tenho que passar o meu input, tem que ser igual a esse cara aqui, que ele vai substituir aqui na pergunta.

Então eu vou colocar aqui a minha pergunta. Quantos Oscars o filme Oppenheimer, espero que seja assim, ganhou em 2024. E ele, quando eu executar essa linha aqui, ele vai pegar esse input, vai colocar ele nesse prompt, vai executar o retriever, vai pegar o chunk, que é mais parecido com essa pergunta, colocar nesse contexto, aí ele vai rodar esse LLM com essa pergunta, esse contexto, e vai tirar a resposta.

Interessante. E vamos ver aqui o print. Vamos pegar a resposta e colocar aqui o answer dele.

E agora o momento que a gente estava esperando, vou dar uma limpada aqui no texto.

**Este arquivo é mais longo que 30 minutos.**

[**Atualize para Ilimitado**](https://turboscribe.ai/pt/subscribed?ref=docx_export_upsell) **em** [**TurboScribe.ai**](https://turboscribe.ai/pt/?ref=docx_export_upsell) **para transcrever arquivos de até 10 horas.**