

[VER PLANOS](#)[PROGRAMAÇÃO](#)[FRONT-END](#)[DATA SCIENCE](#)[INTELIGÊNCIA ARTIFICIAL](#)[DEVOPS](#)[UX & DESIGN](#)[MOBILE](#)[INOVAÇÃO & GESTÃO](#)

Alura > Cursos de Inteligência Artificial > Cursos de IA para Programação >  
Conteúdos de IA para Programação >  
Primeiras aulas do curso LangChain: criando chatbots inteligentes com RAG

## LangChain: criando chatbots inteligentes com RAG

### Reduzindo alucinações - Apresentação

0:00 / 2:57

### Apresentando o curso e objetivos

[VER PLANOS](#)

RAG, como é chamado no LangChain. Teremos várias aulas abordando o desenvolvimento, definições e outros aspectos importantes. Começaremos do básico até a criação de um chatbot interessante que poderá ser colocado em prática.

## Introduzindo o instrutor

Meu nome é **Eduardo Pena** e serei o instrutor deste curso. Sou cientista de dados sênior e líder em uma empresa do setor automotivo. Tenho formação em estatística pela Unicamp, sou de Campinas, e possuo pós-graduação em inteligência artificial pela PUC Minas e em ciência de dados pela Federal do Paraná. Tenho cerca de 12 anos de experiência no mundo dos dados e também uma experiência significativa no ensino, com cerca de 4 a 5 anos lecionando ciência de dados, inteligência artificial, Python, R e programação em geral. É uma área que me interessa muito.

Espero que apreciem as aulas que vamos desenvolver, pois acredito que será um material enriquecedor e estou bastante empolgado para compartilhá-lo com vocês.

## Estrutura do curso e conteúdo das aulas

No curso de LangChain, teremos uma visão geral que será abordada de forma incremental. Os conceitos serão explicados de maneira que cada aula esteja interligada com a anterior, portanto, é importante prestar atenção em cada uma para evitar dúvidas e garantir a compreensão completa dos tópicos.

Na primeira aula, começaremos explicando a arquitetura RAG. Na segunda aula, abordaremos o armazenamento vetorial, utilizando *Files* (arquivos) e o Chroma. A terceira aula será dedicada aos *Embeds* (incorporações) de alta performance, um assunto mais avançado e interessante. Em seguida, discutiremos *Pipelines* (fluxos de trabalho) de dados complexos.

Na quinta aula, trataremos de cadeias de conversação robustas. A sexta aula será focada na avaliação dos modelos, com discussões envolvendo LangSmith e Hagas, o que promete ser bastante interessante. Na sétima aula, exploraremos *Hybrid Search* (busca híbrida) e outras técnicas avançadas.

Finalmente, na oitava e última aula, realizaremos um projeto prático, o projeto Capstone, no qual implementaremos um *chatbot* (assistente virtual) com um tema relevante ao



VER PLANOS

## Reduzindo alucinações - Estrutura do curso

0:00 / 10:20

## Introduzindo a arquitetura RAG na prática

Na aula de hoje, vamos discutir a arquitetura RAG na prática. O objetivo é entender como fazer a inteligência artificial responder com base em informações específicas e atualizadas. Esse é o cerne do que o RAG resolve. Quando falamos em informações específicas, referimo-nos a dados da empresa, como documentos internos ou catálogos de produtos. Por "atualizadas", queremos dizer informações que não estão necessariamente no treinamento original do modelo, mas que são sempre atualizadas.

Esses objetivos abordam os principais problemas que discutiremos nesta aula: alucinações e conhecimento desatualizado. Modelos de linguagem, como o GPT, são treinados com bases de dados gigantes, mas fixas e limitadas. Isso gera um problema significativo que precisamos resolver, e possivelmente temos uma solução que comentaremos em breve. Esse problema é sério e, para resolvê-lo, utilizamos o RAG.

## Problemas de alucinações e conhecimento desatualizado

[VER PLANOS](#)

Não tem acesso a informações privadas da empresa, documentos internos ou bases específicas.

A questão das alucinações é bastante discutida. Se já trabalhamos com modelos de LLM, provavelmente já ouvimos falar sobre alucinações. Esses problemas ocorrem quando os modelos inventam respostas ao não saberem algo, criando informações que parecem plausíveis, mas são falsas. Além disso, há o contexto limitado, pois os modelos não conseguem acessar diretamente documentos, catálogos de produtos ou bases específicas.

## Solução: Retrieval Augmented Generation (RAG)

A solução para esses problemas é o RAG, ou *Retrieval Augmented Generation*. O RAG combina o melhor dos dois mundos: a busca precisa de um motor de busca com a capacidade de conversação de uma LLM. Assim, temos a busca mais geração, o que nos permite obter respostas inteligentes e confiáveis.

O RAG consiste em três componentes principais:

**Retrieval (Recuperação):** Envolve a busca de informações relevantes em documentos para responder a uma pergunta. O sistema busca documentos relevantes na base de conhecimento.

**Augmentation (Aumento):** Refere-se à melhoria das capacidades do modelo, inserindo informações encontradas como contexto no *prompt* do usuário.

**Generation (Geração):** Refere-se à capacidade natural das LLMs de gerar um texto coerente.

A combinação desses componentes cria um sistema híbrido mais poderoso do que qualquer componente isolado.

## Funcionamento do processo RAG

Vamos explorar como funciona o processo RAG em etapas claras para entender melhor seu funcionamento. A primeira etapa é a recuperação, que consiste em encontrar trechos de informações mais relevantes em documentos para responder a uma pergunta. O sistema busca documentos relevantes na base de conhecimento. Em

[VER PLANOS](#)

Ele utiliza, nessa primeira etapa, uma busca semântica, não apenas palavras-chave. Quando afirmamos que ele aumenta o *prompt*, queremos dizer que ele adiciona informações relevantes ao contexto da pergunta. A instrução de usar apenas aquele contexto é crucial para evitar alucinações, pois, às vezes, o modelo não sabe a resposta e acaba inventando. Por isso, enfatizamos o uso exclusivo do contexto. Esse processo ocorre de forma automática e em tempo real.

## Explorando o aumento do contexto e componentes do RAG

Como funciona o aumento do contexto? O truque do REG é fornecer ao LLM um livro de consulta específico para cada pergunta, permitindo uma resposta precisa e baseada em fatos. O processo envolve uma pergunta do usuário, seguida por uma busca semântica. O sistema busca documentos relevantes e aumenta o contexto, adicionando documentos ao *prompt*. A LLM responde com base no contexto, gerando uma resposta.

Por exemplo, se a pergunta do usuário for sobre a política de devolução para produtos eletrônicos, o sistema recupera o contexto: produtos eletrônicos podem ser devolvidos em até 30 dias com a nota fiscal, mas itens danificados não são elegíveis. A resposta gerada seria: "Nossa política permite a devolução de produtos eletrônicos em até 30 dias, desde que você apresente a nota fiscal e o produto não esteja danificado." Assim, a resposta é gerada com base em um contexto recuperado, iniciando com a pergunta do usuário. Esse é um processo interessante.

Alguns componentes essenciais do REG serão discutidos mais adiante. Esses componentes principais incluem os *embeds*, que são representações numéricas, vetores que capturam o significado semântico dos textos, permitindo busca por similaridade. Basicamente, são os DNAs da informação. Na próxima aula, entraremos em mais detalhes sobre os *embeds*, juntamente com um banco de dados vetorial. Teremos uma aula específica para isso, abordando a questão de armazenar os dados, indexando os *embeds* para busca rápida e eficiente. Exemplos incluem FIZE, CHROMA, PINECONE, entre outros.

## Detalhando o processo de chunk e geração de embed

O processo de *chunk* é importante, pois divide o documento em partes menores. O modelo de linguagem, a LLM, é o cérebro de todo o processo. O fluxo completo do REG

[VER PLANOS](#)

atividades em suas metas para facilitar a recuperação.

Na geração do *embed*, cada *chunk* é convertido em um vetor numérico, representando seu significado semântico. A indexação vetorial armazena os *embeds* em um banco de dados vetorial para busca eficiente. Na consulta e recuperação, a pergunta do usuário é convertida em *embeds*. O computador não entende textos ou palavras, então precisamos converter para *embeds*. Esse processo de conversão busca *chunks* relevantes, e, por fim, a LLM gera a resposta baseada nas perguntas e nos *chunks* recuperados como contexto.

## Explorando os benefícios e casos de uso do RAG

O RAG é uma revolução em diversos setores e problemas. Ele reduz alucinações, trabalha com conhecimento atualizado e oferece um custo-benefício interessante. A questão da transparência permite saber qual fonte foi usada para gerar a resposta, facilitando auditorias. A privacidade é importante, pois dados sensíveis podem permanecer dentro da infraestrutura, sem exposição externa, garantindo segurança.

Diversos casos de uso do RAG incluem atendimento ao cliente, suporte técnico e médico, análise de documentos legais, pesquisa e desenvolvimento. O RAG pode ajudar a solucionar muitos problemas. Vamos explorar essa ferramenta poderosa durante o curso. Esta aula é uma parte mais teórica; sempre começaremos com teoria, seguida por duas partes práticas para aplicar os conceitos aprendidos. Nos vemos na próxima aula. Muito obrigado!

## Reduzindo alucinações - Definições iniciais



VER PLANOS

0:00 / 12:35

## Introduzindo a prática do RAG

Sejam bem-vindos à segunda parte da **Aula 1**. Na primeira parte, discutimos a teoria do RAG (*Retrieve, Augment, Generation*). Aprendemos sobre o problema que o RAG busca resolver e entendemos cada uma das letras que compõem o acrônimo: *retrieve*, *augment* e *generation*. Exploramos como funciona o fluxo de um processo de RAG, incluindo a busca de dados, a divisão em partes menores (*chunks*), o trabalho com *embeds* e outros aspectos que serão aprofundados ao longo do curso.

Agora, vamos colocar em prática o que aprendemos sobre o RAG. Para isso, utilizaremos o Colab. No Colab, veremos o RAG em ação com o Gemini. O Gemini oferece algumas soluções gratuitas, permitindo que obtenhamos uma chave gratuita para testar e implementar. Escolhemos o Gemini como exemplo, mas poderíamos utilizar outra LLM, como a OpenAI. A chave da OpenAI é fácil de criar, embora tenha um custo, que não é muito alto, especialmente para testes e aprendizado. O processo de implementação muda pouco ao utilizar a OpenAI.

## Explorando o uso do RAG

Vamos explorar como o RAG resolve problemas reais de LLM, algo que já vimos na teoria e agora veremos na prática. Discutiremos as diferenças entre o RAG e o *prompt* tradicional. O *prompt* tradicional, às vezes, não consegue recuperar certas informações que o RAG consegue. Queremos obter informações específicas de forma mais eficaz.

[VER PLANOS](#)

PDF diretamente no Guia para aplicar o processo de RAG.

Relembrando, o RAG é essencial no mercado devido à atualização de dados, respostas verificadas, custos otimizados e aplicações reais importantes.

## Instalando bibliotecas necessárias

Primeiro, vamos instalar algumas bibliotecas necessárias utilizando o `pip install`. As bibliotecas principais que utilizaremos são o LangChain, o Google Gen AI e o PyPDF. Além dessas, é interessante instalar o LangChain Community, que oferece soluções adicionais para trabalhar com nosso modelo de LLM. A instalação pode levar algum tempo, mas, uma vez concluída, estaremos prontos para prosseguir.

```
!pip install langchain langchain-google-genai pypdf  
!pip install langchain-community
```

## Configurando o ambiente de desenvolvimento

Após a instalação das bibliotecas, começaremos o desenvolvimento. Primeiramente, precisamos importar o módulo `os`, que nos permitirá acessar algumas informações.

```
import os
```

Em seguida, configuraremos nossa API, trazendo a chave da API do Google. Para isso, acesse [eaestudio.google.com](http://eaestudio.google.com), crie sua chave API, copie e cole-a no código. É importante lembrar que essa chave é secreta e não deve ser compartilhada, pois pode gerar custos dependendo do modelo utilizado.

```
os.environ["GOOGLE_API_KEY"] = "coloque sua chave aqui"
```

É muito importante não compartilhar a chave de acesso, principalmente não comitá-la em nenhum local, pois isso pode expô-la e permitir que outra pessoa a utilize. Devemos



VER PLANOS

Mas nunca a compreendi.

## Comparando o prompt tradicional com o RAG

Vamos executar o código. Já temos nossa chave pré-configurada. Vamos tentar fazer algumas comparações entre o *Prompter* tradicional e o *Reg*, de forma mais prática. Para começar, vamos trazer algumas das bibliotecas que importamos anteriormente.

```
from langchain.google_genai import ChatGoogleGenerativeAI  
from langchain.prompts import ChatPromptTemplate
```

Definimos nosso modelo de LLM com o GNI. Vamos criar um objeto `chat Google Generative AI`, onde especificamos o modelo desejado e um parâmetro de temperatura (`temperature`), que será definido como zero. Isso determinará se a resposta será mais determinística ou não. O modelo escolhido será o Gemini 1.5 Pro Latest.

```
llm = ChatGoogleGenerativeAI(model="gemini-1.5-pro-latest", temperature=0)
```

## Executando um exemplo de prompt tradicional

Vamos fazer um exemplo. O exemplo 1 será um *prompt* tradicional, sem *Reg*. Vamos tentar entender as diferenças. Faremos uma pergunta simples: "Qual é a política de home office da nossa empresa?" Colocaremos essa pergunta para o LLM responder.

```
pergunta = "Qual é a política de home office da nossa empresa?"  
prompt_tradicional = ChatPromptTemplate.from_template(  
    "Responda a seguinte pergunta: {pergunta}"  
)  
chain_tradicional = prompt_tradicional | llm  
resposta_tradicional = chain_tradicional.invoke({"pergunta": pergunta})  
print(resposta_tradicional.content)
```

[VER PLANOS](#)

(chain). Utilizamos um operador, o `pipe`, que o LangChain usa para criar essa cadeia de execução. Basicamente, conectamos o *prompt* com a LLM, de um lado a pergunta e do outro a LLM. Executamos o *prompt* tradicional e obtemos uma resposta.

Utilizamos o `chain_traditional.invoke` para invocar a pergunta, substituindo a parte da pergunta com a que desejamos. Ao rodar o código, imprimimos a resposta tradicional com `resposta_traditional.content`. Definimos a pergunta e o *prompt* desejado, criamos a cadeia e invocamos a pergunta dentro do nosso *prompt*. A LLM respondeu que não é uma empresa e não possui política de home office, pois é um modelo de linguagem treinado pela Google e não possui informações sobre a empresa.

Na próxima parte da primeira aula, veremos a utilização do *Reg*, onde colocaremos um documento para responder a pergunta. Será mais interessante, pois aplicaremos o *Reg* para responder a pergunta. Muito obrigado e até o próximo vídeo.

## Sobre o curso LangChain: criando chatbots inteligentes com RAG

O [curso LangChain: criando chatbots inteligentes com RAG](#) possui **291 minutos de vídeos**, em um total de **65 atividades**. Gostou? Conheça nossos outros [cursos de IA para Programação](#) em [Inteligência Artificial](#), ou leia nossos [artigos de Inteligência Artificial](#).

Matricule-se e comece a estudar com a gente hoje! Conheça outros tópicos abordados durante o curso:

Reducindo alucinações

Embedding e armazenamento vetorial

Otimizando a performance

Pipelines complexos

Cadeias de conversação

Avaliação de resultados

Busca híbrida

Projeto final

[VER PLANOS](#)

## até 44% OFF

**1 ano**    **2 anos**    OFERTA

O maior desconto!

### PLUS

Impulsiona a sua carreira com os melhores cursos e faça parte da maior comunidade tech.

R\$ 109

22% OFF

12x **R\$85,02**

1 ano de Alura

À vista R\$1.020,24

[MATRICULE-SE](#)

Matricule-se no plano PLUS e garanta:

22 Carreiras

1.997 Cursos

Acesso a TODOS os cursos por 1 ano

Certificado

Mentorias em grupo com especialistas

Comunidade exclusiva

Acesso ao conteúdo das Imersões

[VER PLANOS](#)

## PRO

Acelere o seu aprendizado com a IA da Alura e prepare-se para o mercado internacional.

R\$ 149

22% OFF

12x **R\$116,22**

1 ano de Alura

À vista R\$1.394,64

[MATRICULE-SE](#)

Todos os benefícios do PLUS e mais vantagens exclusivas:

**Luri**, a inteligência artificial da Alura

**Alura Língua** - Inglês e Espanhol

A melhor opção para sua carreira

## ULTRA LAB

Para estudantes ultra comprometidos atingirem seu objetivo mais rápido.

R\$ 299

22% OFF

12x **R\$233,22**

1 ano de Alura

À vista R\$2.798,64

[VER PLANOS](#)

### Todos os benefícios do PRO e mais vantagens exclusivas:

**Luri**, com mensagens ILIMITADAS

**Luri Vision**, a IA que enxerga suas dúvidas

**6 Ebooks** da Casa do Código

**Talent Lab**



Pague com cartão de crédito em até 12x, PayPal, NuPay ou Pix

Garantimos cancelamento gratuito em até 7 dias

[Conheça os Planos para Empresas](#)

### Nossas redes e apps



[Institucional](#)

[A Alura](#)

[VER PLANOS](#)[Para Empresas](#)[Plataforma](#)[Para Sua Escola](#)[Depoimentos](#)[Política de Privacidade](#)[Instrutores\(as\)](#)[Compromisso de Integridade](#)[Dev em <T>](#)[Termos de Uso](#)[Luri, a inteligência artificial da Alura](#)[Documentos Institucionais](#)[IA Conference 2025](#)[Status](#)[Cursos imersivos](#)[Certificações](#)[Uma empresa do grupo Alun](#)

## Conteúdos

[Alura Cases](#)

## Fale Conosco

[Email e telefone](#)[Imersões](#)[Perguntas frequentes](#)[Artigos](#)[Podcasts](#)[Artigos de educação corporativa](#)[Imersão Dev Agentes de IA Google](#)

## Novidades e Lançamentos

[bins.br@gmail.com](#)[ENVIAR](#)

[VER PLANOS](#)

## CURSOS

### Cursos de Programação

Lógica | Python | PHP | Java | .NET | Node JS | C | Computação | Jogos | IoT

### Cursos de Front-end

HTML, CSS | React | Angular | JavaScript | jQuery

### Cursos de Data Science

Ciência de dados | BI | SQL e Banco de Dados | Excel | Machine Learning | NoSQL | Estatística

### Cursos de Inteligência Artificial

IA para Programação | IA para Dados

### Cursos de DevOps

AWS | Azure | Docker | Segurança | IaC | Linux

### Cursos de UX & Design

Usabilidade e UX | Vídeo e Motion | 3D

### Cursos de Mobile

Flutter | iOS e Swift | Android, Kotlin | Jogos

### Cursos de Inovação & Gestão

Métodos Ágeis | Softskills | Liderança e Gestão | Startups | Vendas

## CURSOS UNIVERSITÁRIOS FIAP

Graduação | Pós-graduação | MBA