

[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 Flash Skills: Agentes de IA com LangGraph

Flash Skills: Agentes de IA com LangGraph

Configurando o LLM - Apresentação

0:00 / 0:59

Apresentando o curso e o instrutor

Olá, sejam bem-vindos a este curso de Flash Skills Agentes de IA. Meu nome é **Alan Spadini** e, para fins de acessibilidade, vou me autodescrever.

Audiodescrição: Alan é um homem branco, com cabelos castanhos. Ele está no estúdio da Alura, com uma parede colorida e uma estante ao fundo.

Introduzindo o LandGraph e seus usos

Neste curso, vamos trabalhar com o LandGraph para a construção de agentes.

O *LandGraph* é um dos *frameworks* mais requisitados pelo mercado quando se trata da construção de agentes e multiagentes. Neste curso, vamos aprender a construir agentes responsáveis pela busca na web e em bases de dados e informações específicas. Além disso, trabalharemos com agentes roteadores, que serão capazes de decidir qual agente específico escolher, dependendo da tarefa requisitada.

Convidando para os próximos vídeos

Esperamos vê-los nos próximos vídeos deste curso. Até lá! [♪]

Configurando o LLM - Conectando-se com um LLM

0:00 / 14:47



Introduzindo o projeto de agentes de IA

Neste curso, vamos trabalhar na construção de agentes de IA para resolver um problema específico. Nosso objetivo é criar um agente de IA capaz de buscar informações na internet, como se fosse um agente que coleta links de sites sobre uma pesquisa ou informação específica e elabora uma resposta com base nessa pesquisa. Queremos que a resposta do nosso LLM (Modelo de Linguagem Grande) seja a mais atualizada possível, por isso estamos desenvolvendo este projeto.

Para construir este projeto, utilizaremos o Google Colab, que é um *notebook* que nos permitirá desenvolver o projeto de forma sequencial. Para agilizar o processo, já realizamos a instalação das bibliotecas necessárias ao longo do curso. Deixaremos uma atividade escrita para que seja possível acessar cada um desses comandos e executar apenas a célula de código dentro do Google Colab. Basta copiar os comandos e pressionar Shift + Enter no Google Colab para que ele execute a instalação. Um ponto de atenção é que ele exibirá um botão de "Restart Session" para que possamos reiniciar a sessão e rodar todo o nosso código sem problemas. Após clicar em "Restart Session", a sessão será reiniciada e poderemos iniciar a construção do nosso código.

Instalando bibliotecas necessárias

Para começar, vamos garantir que temos as bibliotecas corretas instaladas. Podemos fazer isso com os seguintes comandos:

```
%pip uninstall -y google-ai-generativelanguage google-generativeai langchain  
%pip install google-ai-generativelanguage==0.6.15  
%pip install -U langchain-google-genai  
%pip install -U langchain  
%pip install -U langchain-community  
%pip install -U langgraph  
%pip install arxiv
```



Para trabalhar com uma LLM e construir agentes, precisaremos de uma LLM. Neste curso, utilizaremos a LLM do Gemini, mas é possível usar qualquer LLM que o *framework* do LangChain permita. Na documentação do LangChain, temos a opção de escolher entre diversas LLMs diferentes, como a LLM do Chat GPT da OpenAI, a da Bedrock, vinculada aos modelos do Cloud, ou mesmo os modelos através da biblioteca OLAMA, que permite a utilização de modelos abertos. Optamos por utilizar a do Gemini, pois, no momento da gravação, eles permitem a utilização em um nível gratuito.

Obtendo e configurando a chave de API

Para obter uma chave de API, necessária para utilizar essa LLM, devemos acessar o site do Google AI Studio. No Google AI Studio, acessamos uma página onde podemos clicar em "Get API Key". Ao clicar, uma tela será exibida, permitindo a criação de uma chave de API. Podemos, por exemplo, deletar uma chave existente e criar uma nova. É necessário vincular a chave a um projeto no Google Cloud, onde podemos criar uma chave de API atual. Caso ainda não tenhamos um projeto no Google Cloud, não há problema, pois é possível criar uma chave gratuita para uso. Se escolhermos um projeto do Google Cloud com faturamento configurado, a chave será paga. Para quem está utilizando pela primeira vez, a chave será uma versão gratuita.

A ideia é copiar essa chave e inseri-la no Google Colab. No Google Colab, precisamos inserir essa chave de forma segura. Uma maneira de fazer isso é clicar em "Secrets" no canto esquerdo da tela e adicionar um novo *secret*. Podemos nomeá-lo, por exemplo, como "chave" e colar a chave. Logo abaixo, no Google Colab, há uma instrução de como copiar esse *secret* para o nosso código. Podemos copiar o trecho de código exibido, colar e substituir o *secret* pelo nome da nossa chave. No nosso caso, foi "chave", mas já temos outra chave do Gemini pré-configurada, então utilizaremos esse nome. Copiamos, trocamos e colocamos o nome "Gemini API Key" como o nome dessa chave.

Configurando variáveis de ambiente

Este é o comando que devemos usar para copiar nossa chave, e colocaremos à frente de `UserData` uma variável para armazenar a informação da chave. Nomearemos essa variável como `API_Key` para guardar a chave, e queremos tornar essa chave uma variável de ambiente. Precisamos de uma variável de ambiente com um nome específico para que a biblioteca que utilizaremos a reconheça.

Para isso, faremos a importação do módulo `os`, utilizando `import os`, que nos permitirá acessar informações do sistema. Após definir `API_Key`, digitaremos `os.environ`, abriremos e fecharemos colchetes, e entre aspas colocaremos `Google_API_Key`, que será igual a `API_Key`. Assim, a chave será armazenada dentro das variáveis de ambiente como `Google_API_Key`. Note que demos um nome diferente para a Gemini API Key, mas não é necessário ser diferente; a ideia é que precisamos ter o nome `Google_API_Key` dentro das variáveis de ambiente para armazenar essa chave.

```
import os
api_key = userdata.get('GEMINI_API_KEY')
os.environ['GOOGLE_API_KEY'] = api_key
```

Utilizando a biblioteca LinkedIn e definindo a LLM

O LinkedIn, que é a biblioteca que vamos começar utilizando, reconhece esse nome como uma chave do Google. Vamos executar o comando e essa chave está guardada. Como mencionado, vamos começar utilizando o LinkedIn, uma biblioteca do mesmo grupo do Lenggraph, que nos permitirá fazer as primeiras interações com a LLM. Queremos descobrir uma informação específica: qual é o impacto da IA em uma área específica. Esse é o tipo de busca que queremos realizar.

Para isso, vamos chamar a LLM do Google através do LinkedIn. Realizamos a importação da biblioteca e agora vamos definir a LLM que estamos utilizando, colocando-a na variável LLM. A LLM será igual ao Google Generative AI. Neste momento, não adotaremos a opção de temperatura. Vamos definir o modelo que estamos utilizando. Dentro do Google AI Studio, temos a opção de diversos modelos, mas neste momento queremos utilizar o Gemini 2.0 Flash. Definimos o modelo como Gemini-2.0-Flash.

```
from langchain_google_genai import ChatGoogleGenerativeAI  
llm = ChatGoogleGenerativeAI(model='gemini-2.0-flash')
```

Criando e utilizando o template de prompt

Com a LLM definida, queremos passar a instrução para ela utilizando um *template* de *prompt*. O *prompt* é a instrução que estamos passando. Para isso, precisamos importar o *prompt template*. Dentro desse *prompt template*, colocaremos a informação que queremos descobrir. Definimos *modelo_d_prompt* como igual ao *prompt template*, onde colocamos o *template* "me diga quais os impactos da IA no assunto {assunto}". A variável *input_variables* será igual a ["assunto"].

```
from langchain.prompts import PromptTemplate  
  
modelo_de_prompt = PromptTemplate(  
    template="Me diga quais os impactos da IA no assunto {assunto}",  
    input_variables=['assunto'])  
)
```

O que foi feito aqui é que queremos saber quais os impactos da IA dentro de um assunto específico. O assunto entre chaves é uma variável, pois queremos saber os impactos da IA em várias áreas diferentes. Podemos simplesmente trocar a variável assunto e pedir por um assunto diferente, como medicina, agricultura, entre outros.

Executando a cadeia de processos e obtendo respostas

Para rodar esse *prompt* com um assunto específico, precisamos chamar uma cadeia dentro do LangChain. Definimos uma cadeia que será igual ao modelo de *prompt* que definimos. Passamos esse texto para a LLM, que irá processá-lo e nos dar uma resposta sobre o assunto. Queremos gerar uma saída no formato de *String*. Precisamos importar o *StringOutputParser* para isso.

```
from langchain_core.output_parsers import StrOutputParser  
  
cadeia = modelo_de_prompt | llm | StrOutputParser()
```

Com a sequência do processo definida, precisamos invocar essa cadeia para obter a resposta. Colocamos a resposta dentro de uma variável chamada *resposta*, que será igual a *cadeia.invoke({ "assunto": "agricultura" })*. A chamada da cadeia é feita e, após algum tempo, a LLM nos dá a resposta. Imprimimos essa resposta, que nos informa sobre o impacto da IA na agricultura. A inteligência artificial está transformando a agricultura de diversas maneiras.

```
resposta = cadeia.invoke({ 'assunto': 'Agricultura' })  
print(resposta)
```

Considerações finais e próximos passos

Podemos ficar satisfeitos com essa resposta ou não, pois a informação pode estar desatualizada. Pode ser interessante buscar essa informação na web. No entanto, estamos contando com o conhecimento registrado na LLM, que pode ser visto como um arquivo salvo e fixo, sem consultar a internet. Se quisermos que a LLM consulte a internet, precisamos adicionar essa capacidade. Vamos explorar como fazer isso no próximo vídeo. Até lá!

Fazendo as primeiras buscas - Rodando o LLM com tools

0:00 / 14:25



Introduzindo a necessidade de busca na web

Já conseguimos chamar a LLM e fazer com que ela respondesse uma questão para nós. No entanto, queremos que a LLM busque informações na web para garantir que a resposta à nossa requisição esteja atualizada. Como podemos fazer isso? Como podemos permitir que a LLM acesse a web, por exemplo, para executar uma busca? Para isso, precisamos criar uma ferramenta.

Vamos criar uma ferramenta baseada em uma biblioteca desenvolvida pela comunidade do LangChain, chamada Tavilli. O Tavilli possui um site próprio, e, para utilizá-lo, também precisaremos de uma API. O interessante é que o site do Tavilli permite o uso até um determinado nível gratuito.

Configurando a conta e chave de API do Tavilli

Para começar, precisamos criar uma conta no Tavilli. Vamos deixar uma atividade mostrando como criar essa conta. Basta pegar a chave de API do Tavilli, que aparecerá na tela, e copiá-la. O procedimento será semelhante ao que fizemos com o Gemini. Vamos no símbolo de chave, no lado esquerdo, em "Secrets", e colocamos uma variável com essa chave. No nosso caso, já temos configurada essa chave do Tavilli, que nomeamos como `tavilli`, tudo em minúsculo.

Para utilizar essa ferramenta, que já foi preparada pela comunidade do LangChain, faremos o *import* dela. Vamos usar o comando:

```
from langchain_community.tools.tavily_search import TavilySearchResults
```

Além disso, precisaremos de outra funcionalidade do LangChain, que é o `tool`, para informar que estamos definindo uma ferramenta. Usaremos o comando:

```
from langchain_core.tools import tool
```

Definindo a função de busca na web

Após isso, executamos o comando com *shift + Enter*. Caso haja algum erro de digitação, corrigimos e executamos novamente. Com os *imports* feitos corretamente, precisamos pegar a chave de API do Tavilli e colocá-la dentro de uma variável de ambiente. Definimos isso com:

```
os.environ["TAVILLI_API_KEY"] = userdata.get("tavilli")
```

Executamos novamente com *shift + Enter*, e a variável de ambiente estará definida.

Agora, podemos chamar o `TavilliSearchResults`. No entanto, não queremos utilizá-lo de forma direta. Queremos definir a chamada do `TavilliSearchResults` dentro de uma função Python. Isso demonstra que podemos definir como ferramentas quaisquer funções Python que conseguirmos criar, não sendo necessário que estejam vinculadas a uma ferramenta pré-definida pela comunidade do LangChain. Podemos chamar ou definir quaisquer ferramentas.

Implementando a função de busca e testando

Vamos definir uma função Python normal, bem definida e detalhada. A função será:

```
def busca_web(query: str) -> list:  
    """Busca na web por um termo específico"""
```

A query será nossa pesquisa, o termo de busca, por exemplo, "o impacto da IA na agricultura". Essa entrada estará no formato de *string*. O formato de saída será uma lista. Dentro da função, colocaremos uma descrição do que ela faz: "Busca na web por um termo específico".

Para definir que essa função é uma ferramenta, colocamos `@tool` antes dela. Em seguida, fazemos a chamada da biblioteca Tavilli, que será armazenada na variável `tavilli_search`. Definimos:

```
@tool  
def busca_web(query: str) -> list:  
    """Busca na web por um termo específico"""  
  
    tavilli_search = TavilySearchResults(max_results=2, search_depth="advanced")  
    resultado_busca = tavilli_search.invoke(query)  
    return resultado_busca
```

Isso estabelece algumas restrições para as buscas, como o número máximo de resultados e a profundidade da busca.

Integrando a função com a LLM

Por fim, podemos testar nossa função chamando-a diretamente:

```
busca_web("IA na Agricultura")
```

Ao pressionarmos *shift* mais "Enter", obtemos um resultado de busca. É importante notar que isso não está relacionado à chamada da API do Gemini. Estamos utilizando a API do Tavilli para realizar a busca e retornar um link com o resultado encontrado na web. O Tavilli nos fornece o título do site, o link e um resumo do conteúdo. Para isso, é necessário definir um *search depth* (profundidade de busca), *advanced* (avançado) e um número máximo de *tokens*. O Tavilli possui uma inteligência artificial que resume o conteúdo do site, o que é bastante interessante.

Essa funcionalidade é uma função que encapsula o Tavilli. Poderíamos ter outras funcionalidades, como uma conta matemática, que seria uma ferramenta definida para a LLM usar. No entanto, a LLM ainda não tentou utilizar essa ferramenta. Para que a LLM utilize essa ferramenta e resuma ainda mais, verificando, por exemplo, o número de links obtidos e fazendo um resumo da estrutura, podemos definir nossas ferramentas.

Configurando a LLM com ferramentas

Definimos as ferramentas com:

```
tools = [busca_web]
```

Podemos fornecer várias ferramentas para a LLM, mas, pela experiência, quanto menos ferramentas uma LLM específica utilizar, menos confusa ela ficará. Inicialmente, utilizaremos apenas a ferramenta de busca web. Após pressionar *shift* mais "Enter", fornecemos essa ferramenta para a LLM.

Criamos uma nova variável, `llm_com_ferramenta`, que será igual a:

```
llm_com_ferramenta = llm.bind(tools=tools)
```

Assim, agregamos essa informação à LLM original que definimos. Após pressionar *shift* mais "Enter", utilizamos o *prompt template* com o mesmo modelo de *prompt* que já construímos, restringindo o uso apenas às ferramentas definidas. Não queremos que a LLM utilize seu conhecimento prévio, pois desejamos que ela chame a ferramenta.

Testando a integração com a LLM

Definimos `input_variables`, relacionado ao assunto que definimos, e criamos uma cadeia com:

```
modelo_de_prompt = PromptTemplate(  
    template="Usando apenas as tools disponíveis me diga quais os impactos de  
    input_variables=[\"assunto\"]  
)
```

```
cadeia = modelo_de_prompt | llm_com_ferramenta | StrOutputParser()
```

Após pressionar *shift* mais "Enter", o processo é semelhante ao que fizemos anteriormente. Invocamos a cadeia com:

```
resposta = cadeia.invoke({"assunto": "Agricultura"})
resposta
```

No entanto, a resposta impressa estava vazia, com apenas duas aspas. Para entender o que ocorreu, utilizamos outro modelo para chamar a LLM com *prompt*. Passamos o *prompt* de forma mais direta, sem utilizar um *prompt template*, apenas com as ferramentas disponíveis. Perguntamos sobre os impactos da IA na agricultura, sem incluir variáveis. Após pressionar *shift* mais "Enter", chamamos novamente a LLM de outra forma, definindo *resposta* como:

```
prompt = """Usando apenas as tools disponíveis me diga quais os impactos da
resposta = llm_com_ferramenta.invoke(prompt)
```

Analisando os resultados e próximos passos

Ao pressionar *shift* mais "Enter", imprimimos a resposta:

```
print(resposta)
```

Ainda assim, o conteúdo impresso estava vazio, mas recebemos mais informações como saída. A LLM percebeu que precisava chamar a ferramenta, identificou o nome *busca_web* e os argumentos, que são os impactos da IA na agricultura. Ela não utilizou o *prompt* completo, apenas a parte relevante para a busca. A resposta incluiu *metadata* e outras informações extras, mas não gerou a saída esperada. A LLM apenas identificou a necessidade de chamar a ferramenta e parou nesse ponto.

Para resolver isso, precisaremos de algo que realize mais passos. No próximo vídeo, começaremos a trabalhar com agentes. Até o próximo vídeo.

Sobre o curso Flash Skills: Agentes de IA com LangGraph

O [curso Flash Skills: Agentes de IA com LangGraph](#) possui **92 minutos de vídeos**, em um total de **19 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:

Configurando o LLM

Fazendo as primeiras buscas

Aperfeiçoando buscas com múltiplos agentes

Montando uma visualização

Escolha a duração do seu plano e aproveite 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

App Android e iOS para estudar onde quiser

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

MATRICULE-SE**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

Sobre nós

Trabalhe na Alura

Para Empresas

Para Sua Escola

Política de Privacidade

Compromisso de Integridade

Termos de Uso

Documentos Institucionais

Status

Uma empresa do grupo Alun

A Alura

Como Funciona

Formações

Plataforma

Depoimentos

Instrutores(as)

Dev em <T>

Luri, a inteligência artificial da Alura

IA Conference 2025

Cursos imersivos

Certificações

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](#)

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

CURSOS UNIVERSITÁRIOS FIAP

Graduação | Pós-graduação | MBA