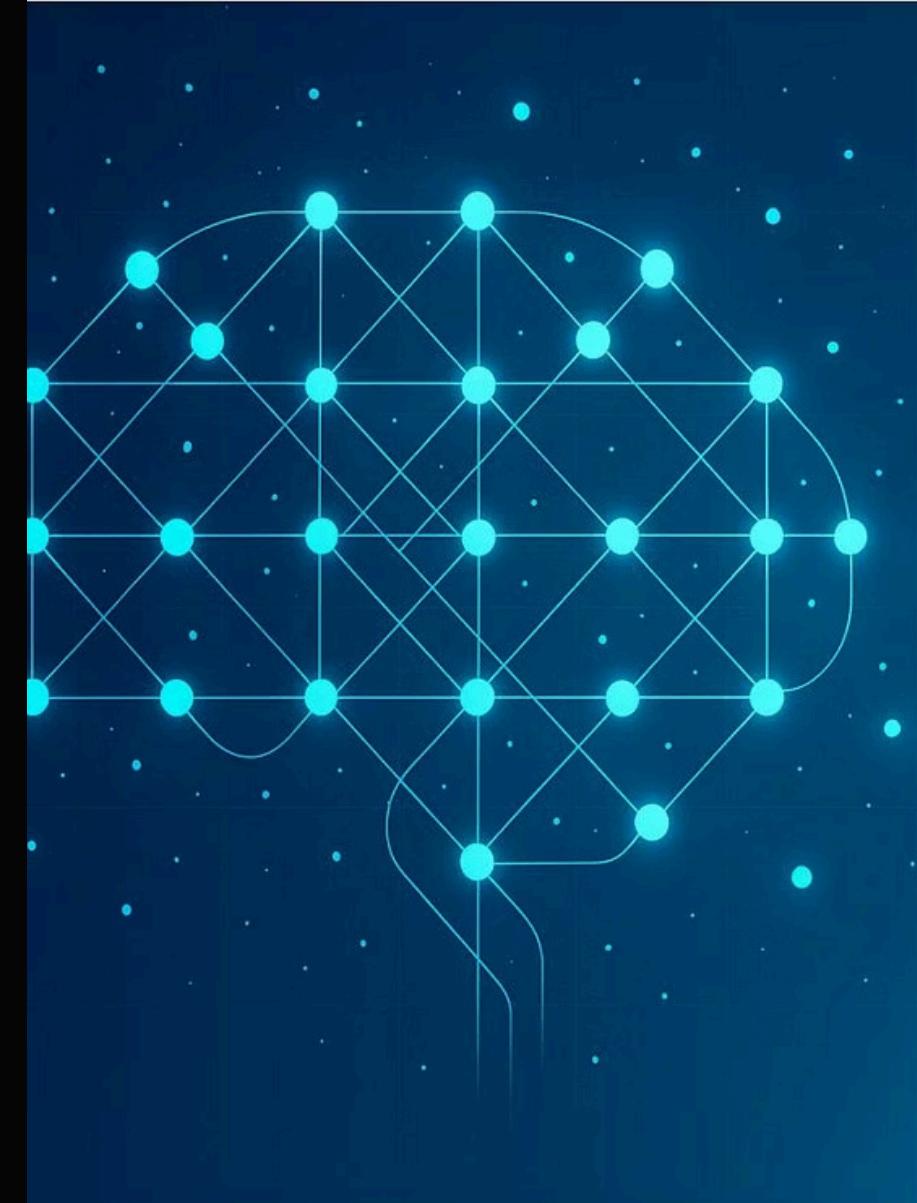


# Engenharia de Agentes de IA

Domine a tecnologia que está redefinindo o mundo. Torne-se um arquiteto de sistemas inteligentes e autônomos.



# Módulo 1: Fundamentos de IA Generativa

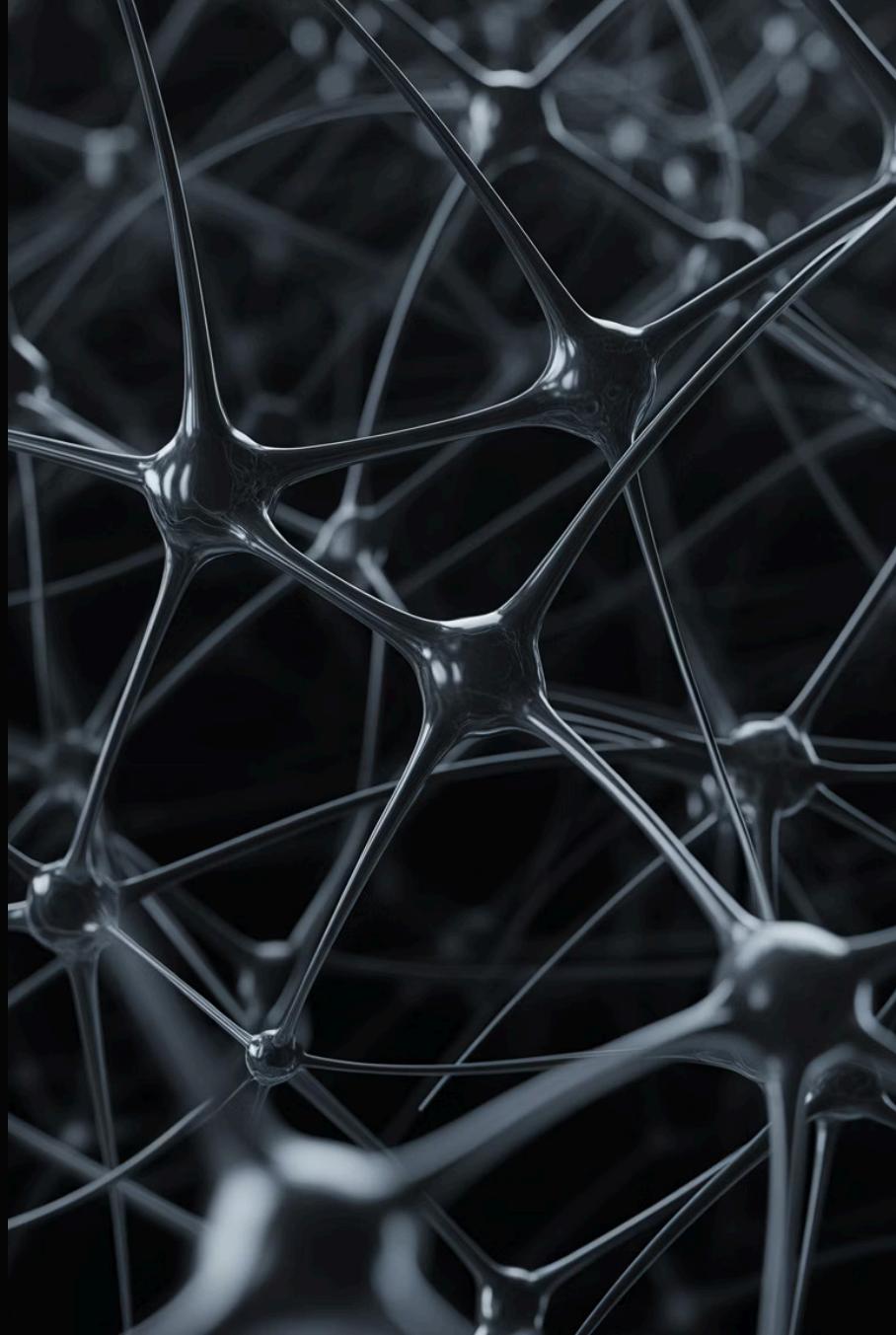
Duração

8 horas de conteúdo intensivo

Nível

Iniciante - nenhum conhecimento prévio  
necessário

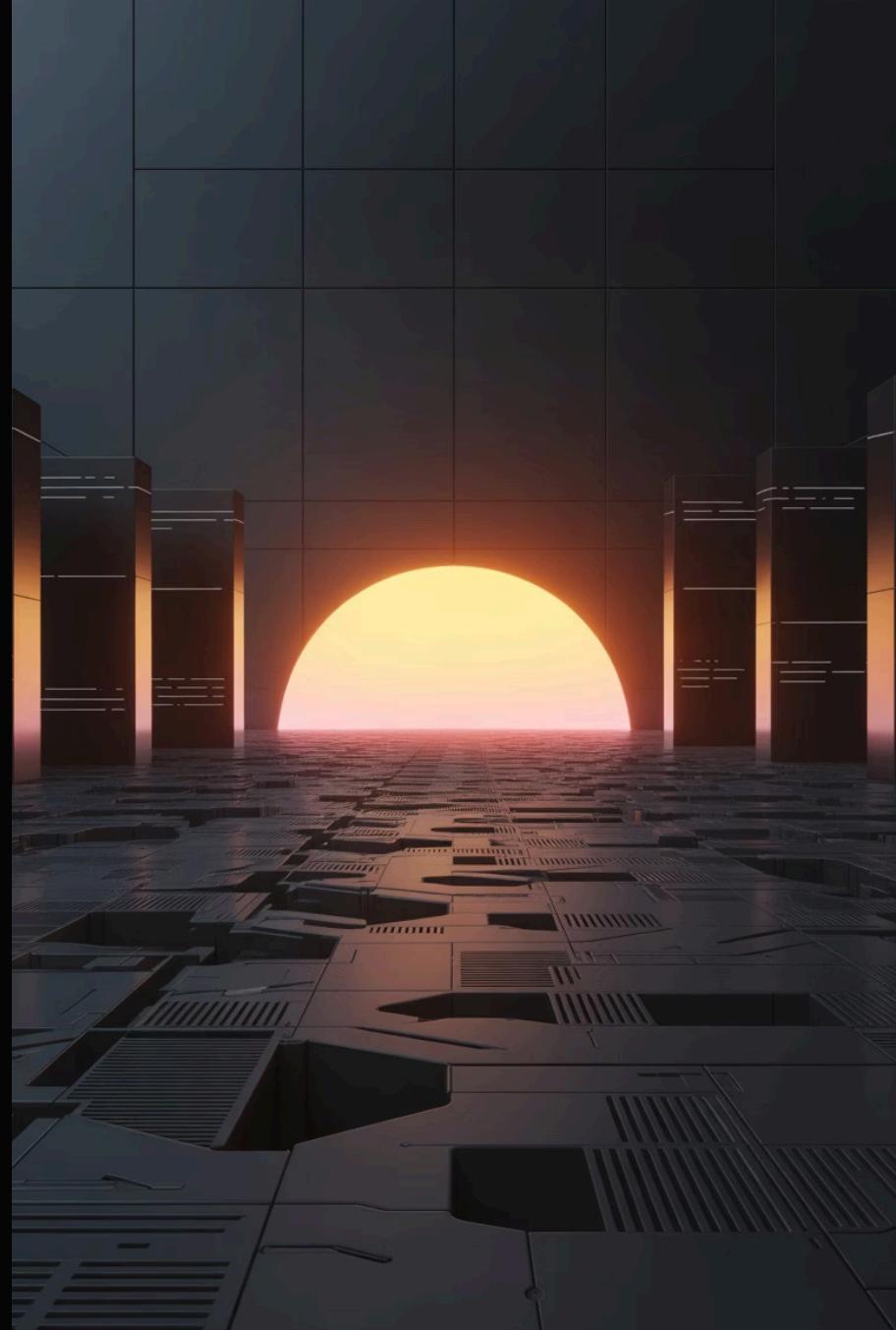
"A IA não pensa. Ela prevê. E previsão é poder."



# O Despertar: Sua Jornada Começa

Você está prestes a dominar a tecnologia que está redefinindo o mundo. Não como um mero usuário, mas como um arquiteto. Este é o seu primeiro passo para se tornar um Engenheiro de Agentes de IA, aprendendo a linguagem das máquinas que aprendem.

Bem-vindo ao ponto de partida da sua jornada. Neste módulo, vamos desmistificar a "mágica" por trás da Inteligência Artificial Generativa. Você não precisa de nenhum conhecimento prévio em IA; apenas curiosidade e a vontade de construir o futuro. Vamos mergulhar nos conceitos que sustentam modelos como o ChatGPT, Midjourney e outros, estabelecendo a base sólida sobre a qual construiremos sistemas complexos e autônomos nos módulos seguintes.





# A Evolução da Inteligência Artificial

- 1 IA Simbólica  
Sistemas baseados em regras rígidas programadas por humanos. Poderosa, mas limitada.
- 2 Machine Learning  
Sistemas que aprendem a partir de dados, superando as limitações das regras fixas.
- 3 Deep Learning  
Redes neurais profundas que processam informações em múltiplas camadas de abstração.
- 4 Transformers (2017)  
A arquitetura revolucionária que permitiu escalabilidade sem precedentes e abriu as portas para os LLMs modernos.



# A Arquitetura Transformer

## Encoder

Lê e comprehende a informação de entrada. Analisa cada parte da sequência e constrói uma representação matemática rica em contexto. É como ler uma frase e entender não apenas as palavras, mas as relações entre elas.

## Decoder

Gera uma nova sequência de dados com base na compreensão do encoder. Prevê a próxima palavra (ou pixel) mais provável, uma de cada vez, até completar a tarefa.

- ❑ **Insight:** O mecanismo de atenção permite que o modelo pese a importância de diferentes palavras na sequência de entrada ao processar uma palavra específica. Ao traduzir "O gato sentou no tapete", a atenção garante que o modelo associe "sentou" com "gato" e "tapete", entendendo o contexto da ação.

# O Mecanismo de Atenção

O mecanismo de atenção é o coração do Transformer. Ele permite que o modelo foque nas partes mais relevantes da informação de entrada ao processar cada elemento. Pense nele como um holofote que ilumina as palavras mais importantes em cada momento do processamento.

01

## Análise de Relevância

O modelo calcula scores de atenção para cada palavra em relação às outras

02

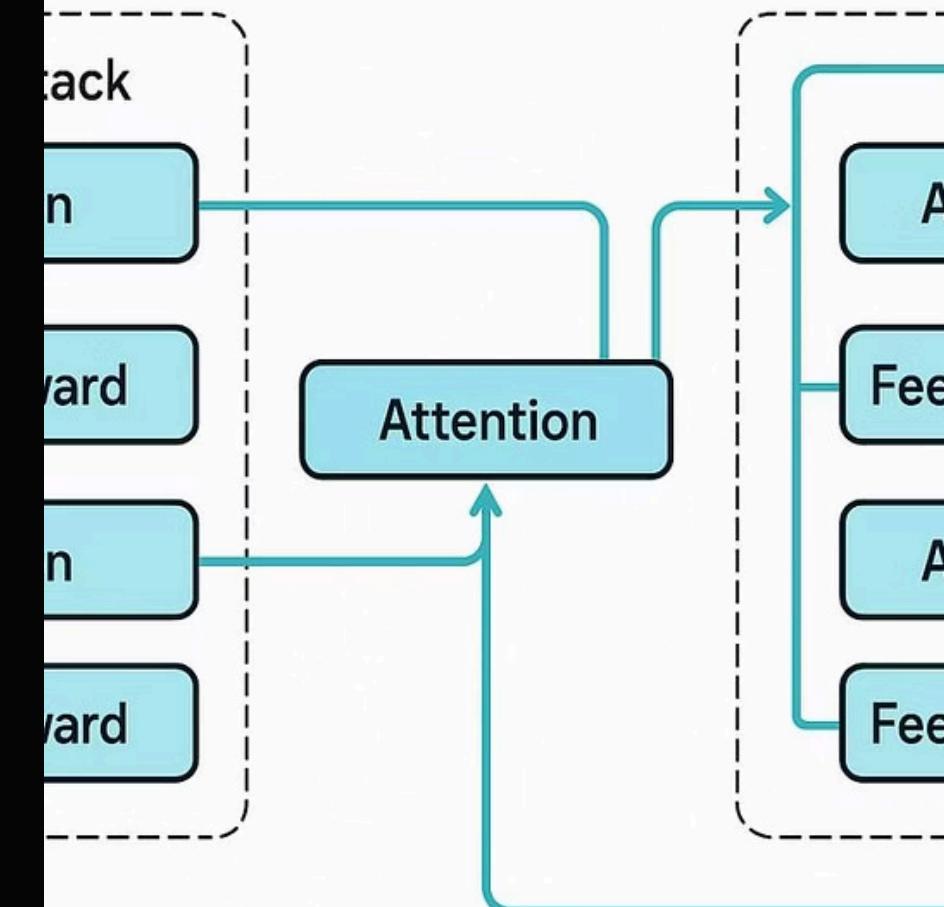
## Ponderação Contextual

Palavras mais relevantes recebem maior peso no processamento

03

## Representação Enriquecida

Cada palavra é representada considerando seu contexto completo



# Tokenização e Embeddings

Modelos de linguagem não leem palavras; eles leem números. O processo de converter texto em números que a máquina pode entender é fundamental e ocorre em duas etapas essenciais.

A horizontal flow diagram consisting of two dark grey chevron-shaped blocks. The first block on the left has the number '1' in white. The second block on the right has the number '2' in white.

1

## 1. Tokenização

O texto é quebrado em pedaços menores, chamados tokens. Um token pode ser uma palavra, parte de uma palavra ou até mesmo um único caractere. Por exemplo, "IA Generativa" pode ser tokenizada em ["IA", "Genera", "tiva"].

2

## 2. Embeddings

Cada token é mapeado para um vetor numérico de alta dimensão. Esse vetor captura o significado semântico do token. Palavras com significados semelhantes terão vetores próximos no espaço vetorial.

Imagine um dicionário onde cada palavra aponta para coordenadas em um mapa 3D. Palavras relacionadas a "realeza" estariam agrupadas em uma região, enquanto palavras sobre "tecnologia" estariam em outra. É isso que os embeddings fazem, mas em centenas ou milhares de dimensões.

# Anatomia de um LLM

Um LLM é, em essência, uma pilha massiva de camadas de Transformer, treinada em uma quantidade colossal de dados da internet. Essa escala é o que permite o comportamento emergente que vemos.



## Camadas Superficiais

Aprendem gramática e sintaxe básica



## Camadas Intermediárias

Capturam relações semânticas e contexto

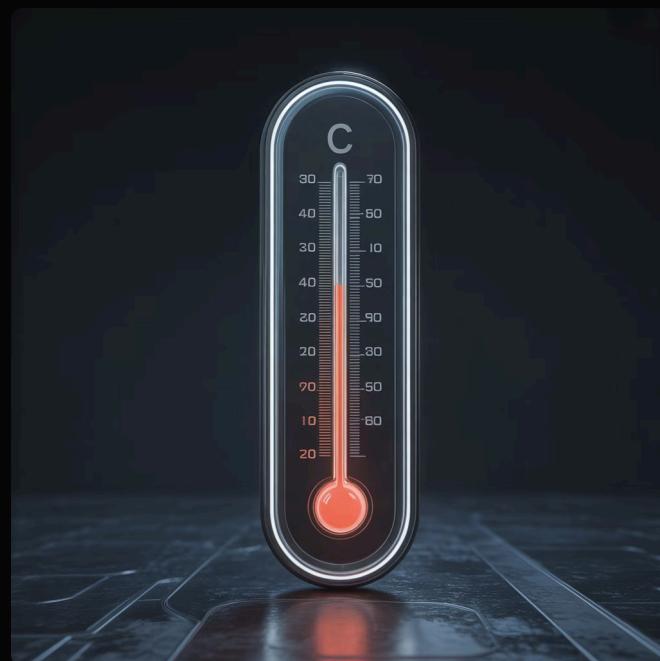


## Camadas Profundas

Desenvolvem raciocínio abstrato e estilo

# Parâmetros de Geração de Texto

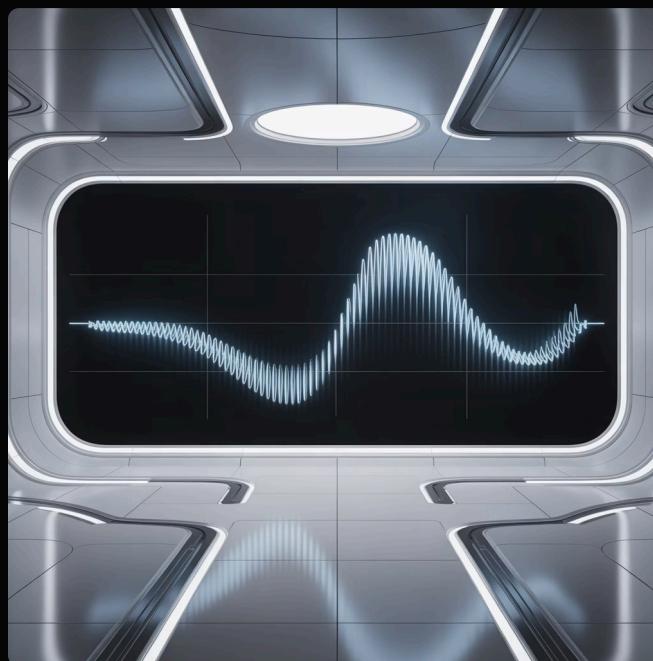
Temperature



**Baixa (0.1):** Respostas previsíveis e focadas

**Alta (1.5):** Respostas criativas e diversas, mas com maior risco de erros

Top-p



**Nucleus Sampling:** Seleciona o menor conjunto de tokens cuja probabilidade acumulada excede o valor p

**Exemplo:** Com top-p=0.9, considera apenas os tokens que compõem os 90% mais prováveis

Top-k



**Limitação:** Restringe a seleção aos k tokens mais prováveis

**Controle:** Oferece um equilíbrio entre diversidade e coerência

- ☐ **Teste você mesmo:** Use um playground de LLM e experimente gerar o mesmo prompt com diferentes valores de temperature e top-p. Observe como a previsibilidade e a "criatividade" da resposta mudam drasticamente.

# Além do Texto: Multimodalidade

A IA Generativa vai muito além da linguagem. Os mesmos princípios dos Transformers podem ser aplicados a outros tipos de dados, criando uma IA verdadeiramente multimodal.



## Modelos de Imagem

Stable Diffusion, Midjourney e DALL-E usam difusão para remover ruído e criar imagens coerentes a partir de prompts de texto. É como um escultor que começa com mármore ruidoso e esculpe a obra de arte.



## Modelos de Áudio

Whisper da OpenAI aplica Transformers para transcrição de fala para texto com precisão impressionante. Modelos Text-to-Speech geram vozes humanas realistas a partir de texto.



## Modelos de Vídeo

Sora e similares combinam compreensão de texto, geração de imagens e consistência temporal para criar clipes de vídeo a partir de prompts simples. A fronteira atual da IA generativa.

# Resumo: Fundamentos de IA Generativa

## IA Generativa

Baseada em prever o próximo item em uma sequência usando padrões aprendidos de vastos conjuntos de dados

## Transformer

Arquitetura chave com Encoder, Decoder e mecanismo de Atenção que revolucionou o processamento de linguagem

## Tokenização & Embeddings

Como a IA converte texto em vetores numéricos com significado semântico

## Parâmetros de Geração

Temperature, top-p e top-k controlam a criatividade e previsibilidade das respostas

## Multimodalidade

Aplicação dos mesmos princípios a imagens, áudio e vídeo para criar sistemas completos



# Projeto Prático: Gerador de Ideias Multimodal

01

## Objetivo

Crie um script em Python que use uma API de LLM para gerar uma ideia de produto inovadora

02

## Expansão

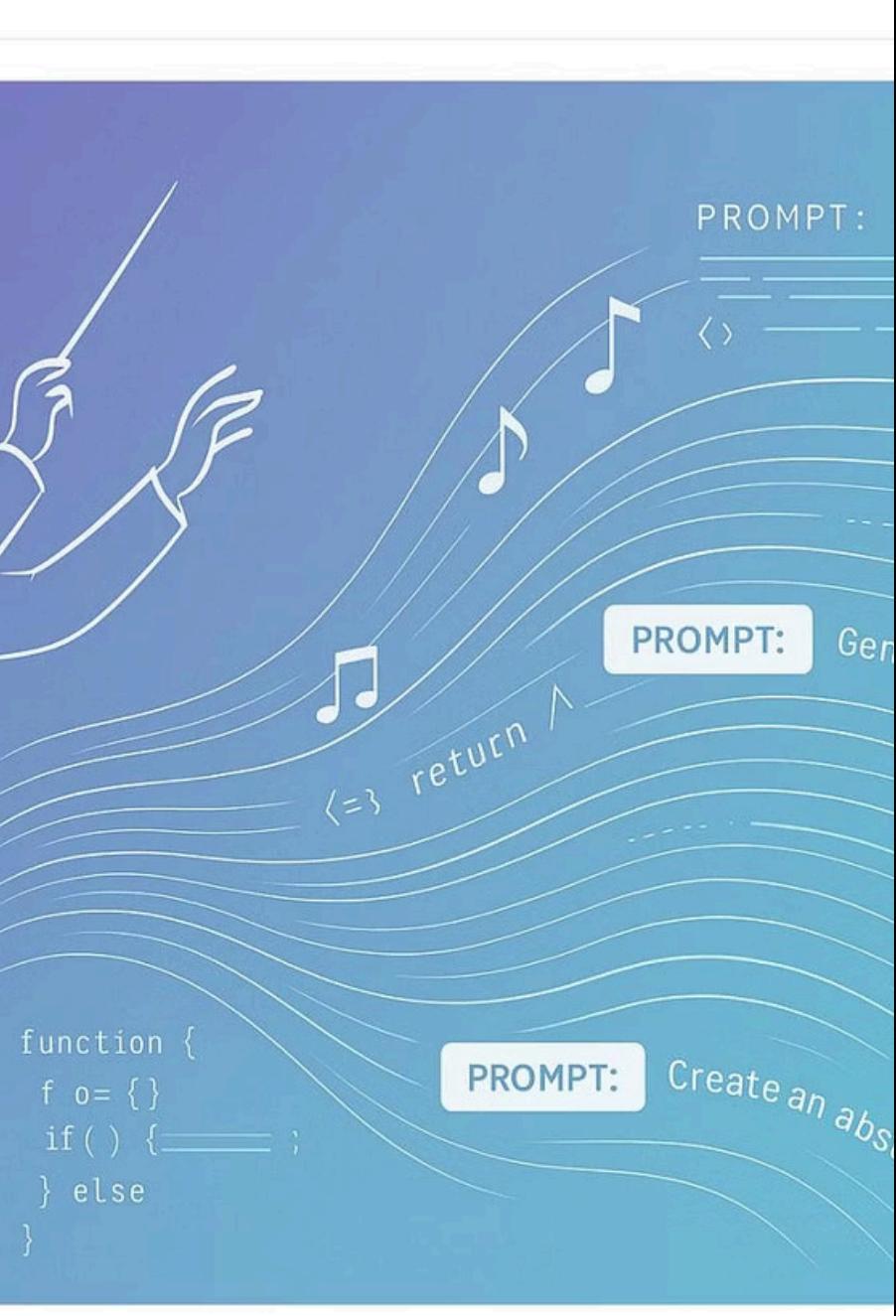
A partir da ideia gerada, use o mesmo LLM para criar um prompt detalhado para um modelo de geração de imagem

03

## Visualização

Use uma API de geração de imagem (como Stable Diffusion) para criar um conceito visual do produto

Este projeto irá solidificar sua compreensão de como diferentes modalidades de IA podem ser orquestradas para um único objetivo criativo, demonstrando o poder da integração multimodal.



## Módulo 2: Engenharia de Prompts Avançada

Duração

6 horas de prática intensiva

Nível

Iniciante-Intermediário

"Palavras são código. Aprenda a programar com linguagem natural."

# A Arte de Conversar com IA

Você entendeu a máquina. Agora, você vai aprender a sua linguagem. Não a linguagem de programação, mas a linguagem do pensamento. A engenharia de prompts é a arte de esculpir a intenção em palavras, guiando a vasta inteligência da IA para um propósito específico.

Se o Módulo 1 foi sobre entender o motor, este módulo é sobre aprender a dirigir. Um LLM, por mais poderoso que seja, é um instrumento. A qualidade da música que ele produz depende inteiramente da habilidade do maestro. Aqui, você se tornará esse maestro. Vamos transformar a maneira como você interage com a IA, passando de simples perguntas para instruções complexas e estratégicas que extraem o máximo potencial desses modelos.

# Anatomia de um Prompt Perfeito



## 1. Persona (Role)

Defina quem o LLM deve ser. "Você é um especialista em marketing digital..." Isso prepara o modelo com o tom, o conhecimento e o estilo certos.



## 2. Contexto (Context)

Forneça as informações de fundo necessárias para a tarefa. Inclua dados, restrições e o objetivo final.



## 3. Tarefa (Task)

Descreva clara e inequivocamente o que você quer que o modelo faça. Use verbos de ação: "Analise", "Resuma", "Crie", "Traduza".



## 4. Formato (Format)

Especifique como a saída deve ser estruturada. "Retorne em JSON", "Crie uma tabela com três colunas", "Escreva em tom profissional".

# Zero-Shot vs. Few-Shot Prompting

## Zero-Shot

Você pede ao modelo para realizar uma tarefa sem fornecer nenhum exemplo. Funciona bem para tarefas gerais, mas pode falhar em tarefas complexas ou de nicho.

**Exemplo:** "Traduza este texto para inglês: [texto]"

## Few-Shot

Você fornece ao modelo alguns exemplos (geralmente de 2 a 5) de entradas e saídas desejadas antes de fazer o pedido final. Isso ajuda o modelo a entender o padrão e o formato exatos que você espera.

**Exemplo:** "Entrada: 'Olá' → Saída: 'Hello'. Entrada: 'Obrigado' → Saída: 'Thank you'. Agora traduza: 'Bom dia'"

- **Insight:** O Few-Shot Prompting é uma das técnicas mais poderosas para melhorar a precisão e a confiabilidade de um LLM sem a necessidade de fine-tuning (retreinamento), que é um processo muito mais caro e complexo.



# Chain of Thought (CoT)

Para tarefas que exigem lógica, matemática ou planejamento em várias etapas, um prompt simples não é suficiente. Precisamos ensinar o modelo a "pensar passo a passo".

A técnica de Cadeia de Pensamento consiste em instruir o modelo a externalizar seu processo de raciocínio antes de dar a resposta final. Ao simplesmente adicionar a frase "Pense passo a passo" ou "Vamos raciocinar sobre isso", você força o modelo a detalhar sua lógica, o que reduz drasticamente os erros em problemas complexos.

# Exemplo Prático de Chain of Thought

## Prompt Padrão

"João tem 5 maçãs. Ele compra mais 3 caixas de maçãs, cada uma com 4 maçãs. Quantas maçãs ele tem agora?"

**Resposta (potencialmente errada):** 12

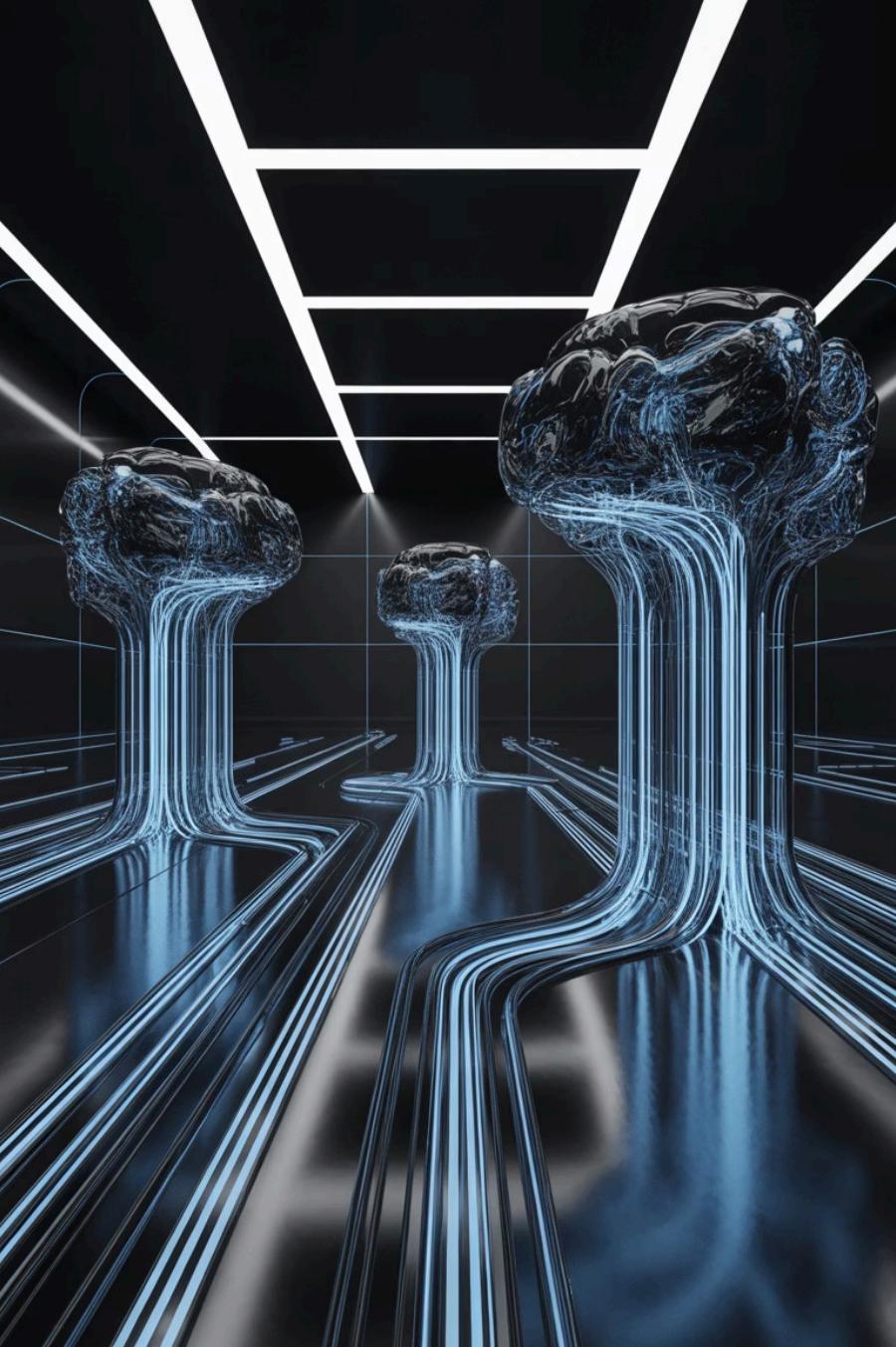
## Prompt com CoT

"João tem 5 maçãs. Ele compra mais 3 caixas de maçãs, cada uma com 4 maçãs. Quantas maçãs ele tem agora? **Pense passo a passo.**"

**Resposta (correta):**

1. Primeiro, vamos calcular quantas maçãs João comprou nas 3 caixas
2. São 3 caixas com 4 maçãs cada, então  $3 \times 4 = 12$  maçãs
3. João já tinha 5 maçãs
4. Somando:  $5 + 12 = 17$  maçãs

**Resposta Final:** João tem 17 maçãs.



# Técnicas Avançadas de Raciocínio

1

## Chain of Thought (CoT)

Instrui o modelo a externalizar seu raciocínio passo a passo antes da resposta final

2

## Self-Consistency

Gera múltiplas cadeias de pensamento e escolhe a resposta que aparece com mais frequência, como um comitê votando

3

## Tree of Thoughts (ToT)

Explora diferentes caminhos de raciocínio como galhos de uma árvore, avaliando estados intermediários e retrocedendo se necessário

# System Prompt: A Constituição do Agente

Quando passamos de simples prompts para a criação de agentes autônomos, a engenharia de prompts se torna a base da "personalidade" e do comportamento do agente. O prompt inicial, muitas vezes chamado de System Prompt, define a identidade, as regras e os objetivos do agente.

## Persona Detalhada

Quem é o agente, quais são suas habilidades, qual é o seu tom de comunicação?

## Objetivo Principal

Qual é a sua razão de existir? Qual é o objetivo final que ele deve sempre buscar?

## Ferramentas

Quais ferramentas ele pode usar e como deve usá-las de forma eficaz?

## Restrições e Guardrails

O que o agente não pode fazer? Quais são as regras de segurança e ética?

## Processo de Raciocínio

Como ele deve pensar? (Ex: Usar ReAct, sempre justificar decisões)

# Resumo: Engenharia de Prompts

## Anatomia do Prompt

Persona, Contexto, Tarefa e Formato são os quatro pilares de um prompt eficaz

## Few-Shot Prompting

Fornecer exemplos para guiar o modelo aumenta drasticamente a precisão

## Chain of Thought

Instruir o modelo a pensar passo a passo reduz erros em tarefas complexas

## Técnicas Avançadas

Self-Consistency e Tree of Thoughts para problemas que exigem múltiplos caminhos de raciocínio

## System Prompt

A "constituição" que define identidade, objetivos e regras de um agente de IA



# Projeto Prático: Análise de Sentimento com CoT

01

## Objetivo

Desenvolva um script em Python que recebe uma avaliação de produto e a classifica como "Positiva", "Negativa" ou "Neutra"

02

## Implementação

Crie um prompt que use Chain of Thought. Instrua o modelo a identificar pontos-chave, avaliar o sentimento de cada ponto e consolidar em uma classificação geral

03

## Teste

Teste com avaliações ambíguas (ex: "O produto é lindo, mas quebrou no primeiro dia") e verifique se a cadeia de pensamento ajuda o modelo a chegar a uma conclusão mais nuançada

Este projeto demonstrará o poder do raciocínio estruturado na resolução de tarefas ambíguas e complexas.

# Módulo 3: Frameworks de Desenvolvimento - LangChain

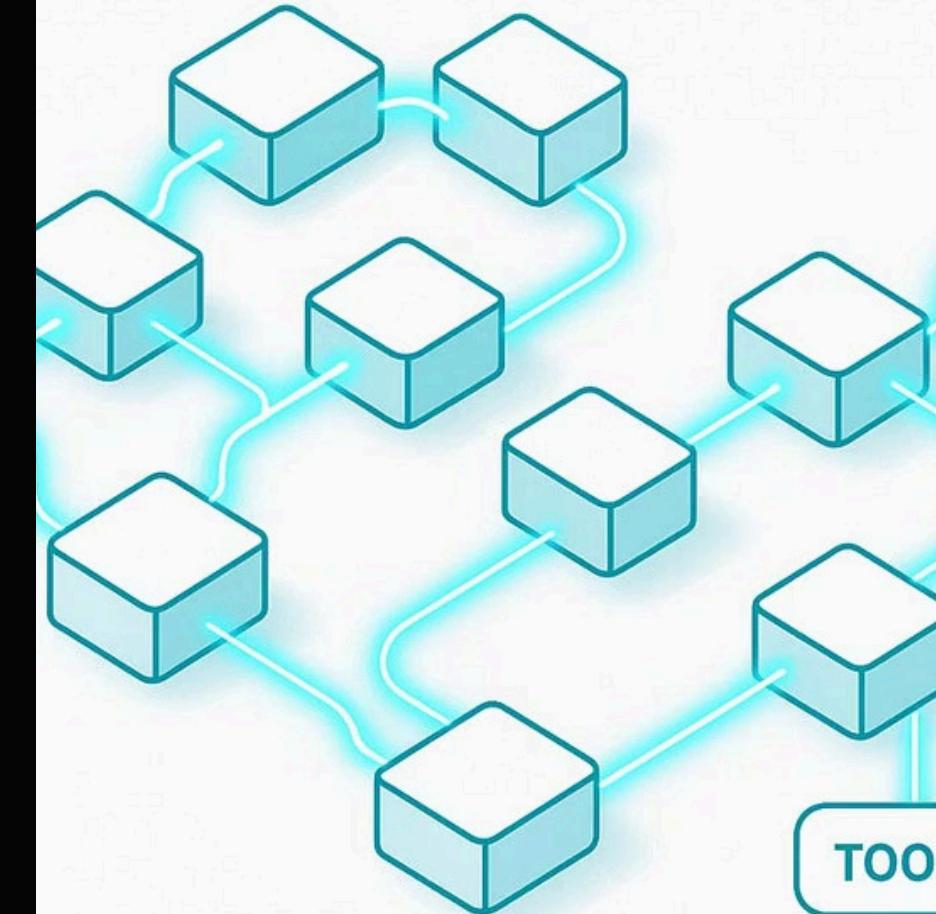
Duração

8 horas de desenvolvimento prático

Nível

Intermediário

"Não construa do zero. Orquestre o que já existe."



# A Construção: De Conversadores a Construtores

Você aprendeu a falar com a IA. Agora, você vai dar a ela mãos para agir e uma mente para lembrar. Este é o momento em que passamos de meros conversadores a verdadeiros construtores de aplicações. Com LangChain, você não está mais limitado a uma única interação; você está construindo sistemas inteligentes e persistentes.

Bem-vindo à oficina do Engenheiro de Agentes. Se os LLMs são o motor e os prompts são o volante, LangChain é o chassi, o sistema de transmissão e todo o conjunto que transforma um motor potente em um veículo funcional. Este framework de código aberto nos dá os blocos de construção para criar aplicações de IA que vão muito além de um simples chatbot.

# Arquitetura Modular do LangChain



## Models

Interações com vários provedores de LLMs: OpenAI, Hugging Face, Anthropic, Cohere e dezenas de outros



## Chains

Combinam LLMs com outras ferramentas ou outros LLMs em sequências lógicas



## Tools

Funções que os agentes podem usar para interagir com o mundo exterior



## Prompts

Ferramentas para gerenciar e otimizar prompts, incluindo templates e seletores de exemplos



## Agents

LLM atua como motor de raciocínio que decide qual ferramenta usar para resolver um problema



## Memory

Permite que Chains e Agents se lembrem de interações passadas

# Chains: Conectando os Pontos

Uma Chain é a estrutura mais fundamental em LangChain. A mais simples, a LLMChain, consiste em um PromptTemplate, um Model e um OutputParser. Ela pega a entrada do usuário, formata o prompt, envia para o LLM e analisa a saída.

O poder real vem das Sequential Chains, onde a saída de uma chain se torna a entrada da próxima, permitindo fluxos de trabalho complexos. Por exemplo, uma chain pode gerar um nome de produto e a próxima pode escrever uma descrição de marketing para esse nome.

- **Insight:** Pense nas Chains como pipelines de montagem para o processamento de linguagem. Cada estação (LLM ou ferramenta) realiza uma tarefa específica antes de passar o trabalho para a próxima, criando um produto final muito mais sofisticado do que qualquer estação poderia fazer sozinha.



# RAG: Retrieval-Augmented Generation

Esta é uma das aplicações mais poderosas e transformadoras dos LLMs. Por padrão, um LLM só conhece a informação com a qual foi treinado, que pode estar desatualizada ou não conter dados privados da sua empresa. O RAG resolve isso.

RAG é a técnica de conectar um LLM a uma fonte de dados externa e dinâmica. Em vez de responder apenas com seu conhecimento interno, o modelo primeiro recupera informações relevantes dessa fonte de dados e, em seguida, usa essas informações para gerar uma resposta informada e contextualizada.

# Fluxo de Trabalho do RAG

01

## Indexing (Indexação)

Seus documentos (PDFs, TXTs, páginas web) são quebrados em pedaços menores (chunks)

02

## Embedding

Cada chunk é transformado em um vetor numérico usando um modelo de embedding

03

## Storage (Armazenamento)

Esses vetores são armazenados em um banco de dados vetorial (Chroma, Pinecone, FAISS)

04

## Retrieval (Recuperação)

A pergunta do usuário é convertida em embedding e o sistema busca os chunks mais semanticamente semelhantes

05

## Generation (Geração)

Os chunks recuperados são inseridos no prompt junto com a pergunta, e o LLM gera uma resposta contextualizada

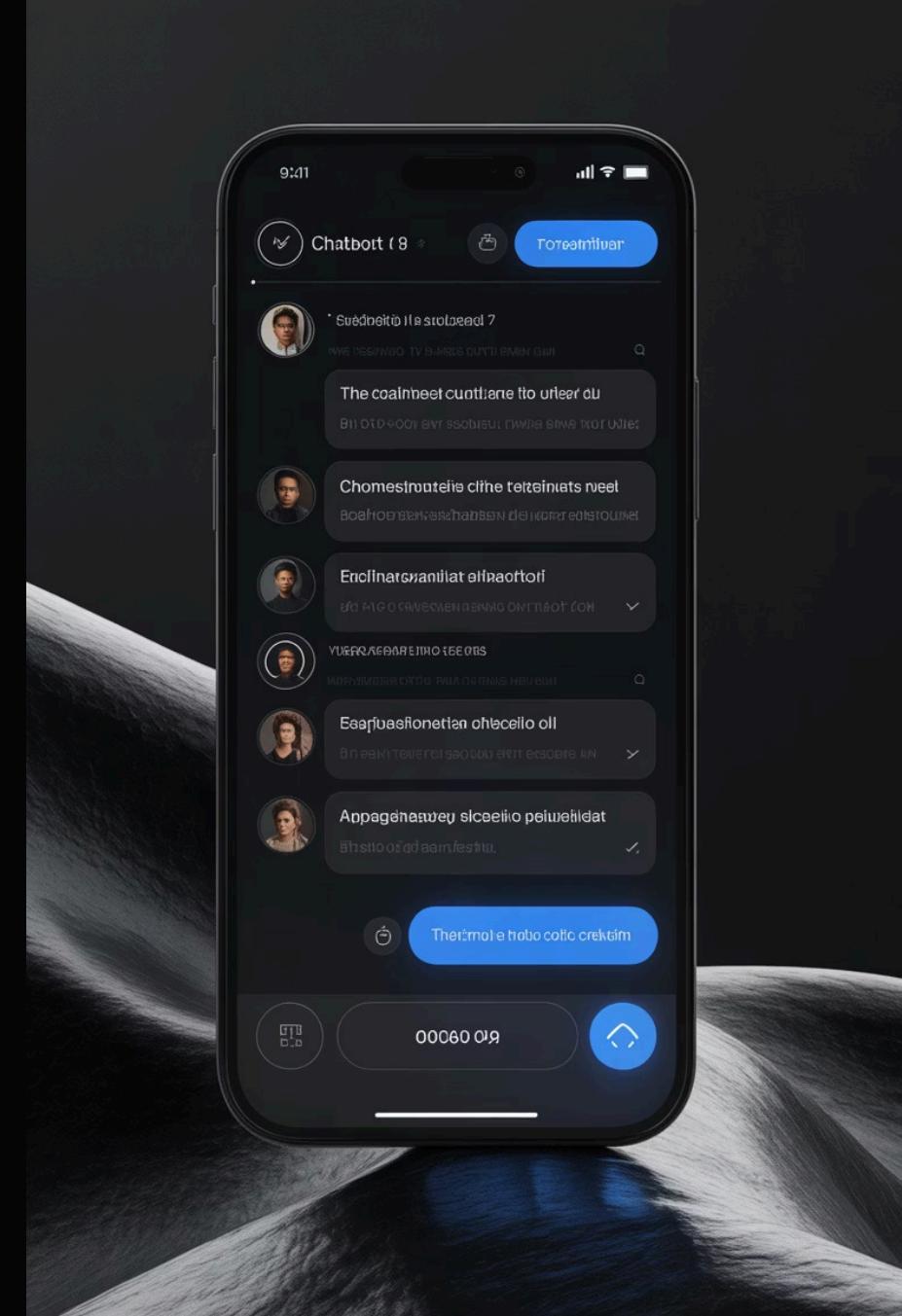
# Exemplo Prático de RAG

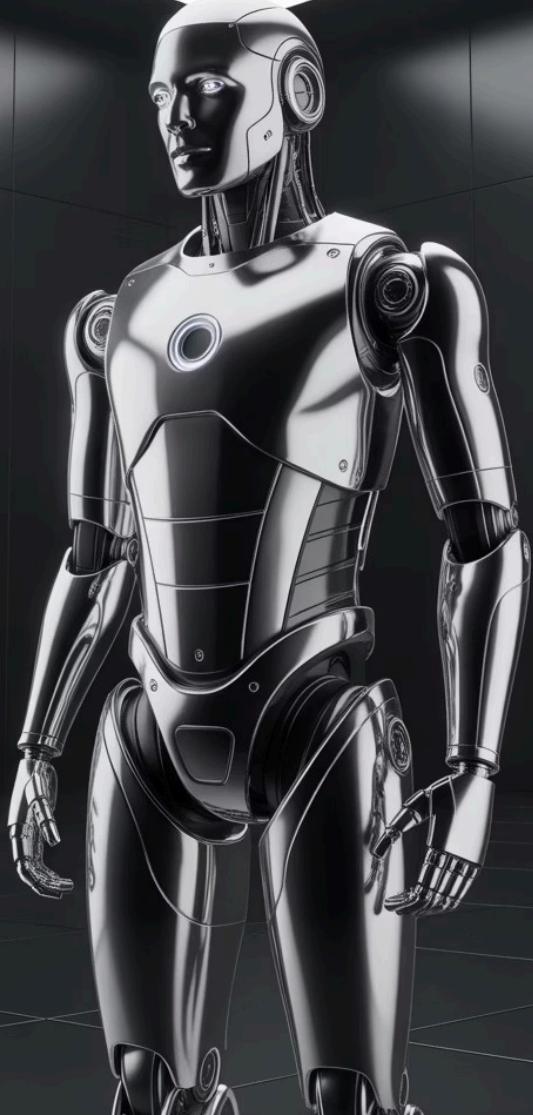
Um chatbot de atendimento ao cliente usando RAG. Em vez de ter respostas pré-programadas, ele pode consultar em tempo real todo o manual de produtos da empresa.

**Cenário:** Quando um cliente pergunta "Qual é a garantia do modelo X?", o sistema:

1. Converte a pergunta em um vetor de embedding
2. Busca no banco de dados vetorial as seções do manual mais relevantes
3. Encontra a seção específica sobre garantias do modelo X
4. Passa essa informação para o LLM junto com a pergunta original
5. O LLM gera uma resposta precisa e atualizada baseada no manual oficial

O resultado é um atendimento sempre atualizado, preciso e baseado em documentação oficial, sem necessidade de retreinar o modelo.





# Agentes LangChain

Se as Chains são ferrovias com um caminho fixo, os Agentes são veículos todo-terreno. Com um agente, você não define uma sequência rígida de ações. Em vez disso, você dá ao LLM um conjunto de ferramentas e um objetivo, e ele decide por si mesmo qual ferramenta usar, em que ordem, para atingir esse objetivo.

Isso é possível através de uma técnica de prompting chamada **ReAct (Reason + Act)**.

# O Ciclo ReAct

## Thought (Pensamento)

Com base na pergunta do usuário, o agente pensa sobre qual é o próximo passo lógico e qual ferramenta seria útil

## Observation

O agente recebe o resultado da ferramenta (ex: "25 graus, ensolarado")



O ciclo se repete. O agente agora tem uma nova observação e pensa no próximo passo (ex: "Eu tenho a informação, agora preciso formatá-la e apresentá-la ao usuário").

# Tipos de Agentes LangChain

## Conversational

Otimizado para conversas, usando memória para manter o contexto ao longo de múltiplas interações

## Self-Ask with Search

Especializado em responder perguntas quebrando-as em sub-perguntas e buscando as respostas

## OpenAI Functions/Tools

Agentes que usam a capacidade nativa de alguns modelos da OpenAI de chamar funções, tornando-os mais confiáveis

- ☐ **Teste você mesmo:** Construa um agente simples com duas ferramentas: uma ferramenta de busca na web e uma ferramenta de calculadora. Faça a pergunta: "Qual é a idade do atual presidente dos EUA elevada à segunda potência?". Observe como o agente primeiro usa a busca para encontrar a idade e, em seguida, usa a calculadora para fazer o cálculo.

# Ecossistema de Integrações

O verdadeiro poder do LangChain reside em seu vasto ecossistema de integrações. Ele abstrai a complexidade de interagir com centenas de ferramentas, modelos e serviços diferentes, permitindo que você os conecte com apenas algumas linhas de código.

## Modelos

Suporte nativo para OpenAI, Anthropic (Claude), Google (Gemini), Cohere, e dezenas de modelos open-source via Hugging Face

## Bancos de Dados Vetoriais

Integrações com mais de 50 Vector Stores, desde soluções locais como Chroma e FAISS até serviços em nuvem como Pinecone e Weaviate

## APIs e Serviços

Centenas de ferramentas pré-construídas para interagir com serviços como Google Drive, Notion, Zapier, Wikipedia e muito mais



# Resumo: LangChain

## Framework de Orquestração

LangChain é um framework modular para construir aplicações de IA complexas

## Chains

Sequências lógicas de ações para fluxos de trabalho definidos

## RAG

Técnica de conectar LLMs a fontes de dados externas para respostas contextualizadas

## Agentes

Usam o LLM como motor de raciocínio para decidir dinamicamente quais ferramentas usar

## ReAct

O ciclo de Pensamento-Ação-Observação que impulsiona os agentes autônomos

# Projeto Prático: ChatPDF

01

## Objetivo

Criar uma aplicação web simples (usando Streamlit ou Gradio) onde o usuário pode fazer upload de um PDF e fazer perguntas sobre seu conteúdo

02

## Implementação RAG

Use LangChain para carregar e dividir o PDF em chunks, criar embeddings, armazenar em um Vector Store (FAISS) e criar uma RetrievalQA Chain

03

## Interface

Crie uma interface de chat onde o usuário pode fazer perguntas e ver as respostas geradas pelo seu sistema RAG

Este projeto é um portfólio fantástico e ensina a base da maioria das aplicações de IA corporativas atuais.



# Módulo 4: Agentes Autônomos com Agno

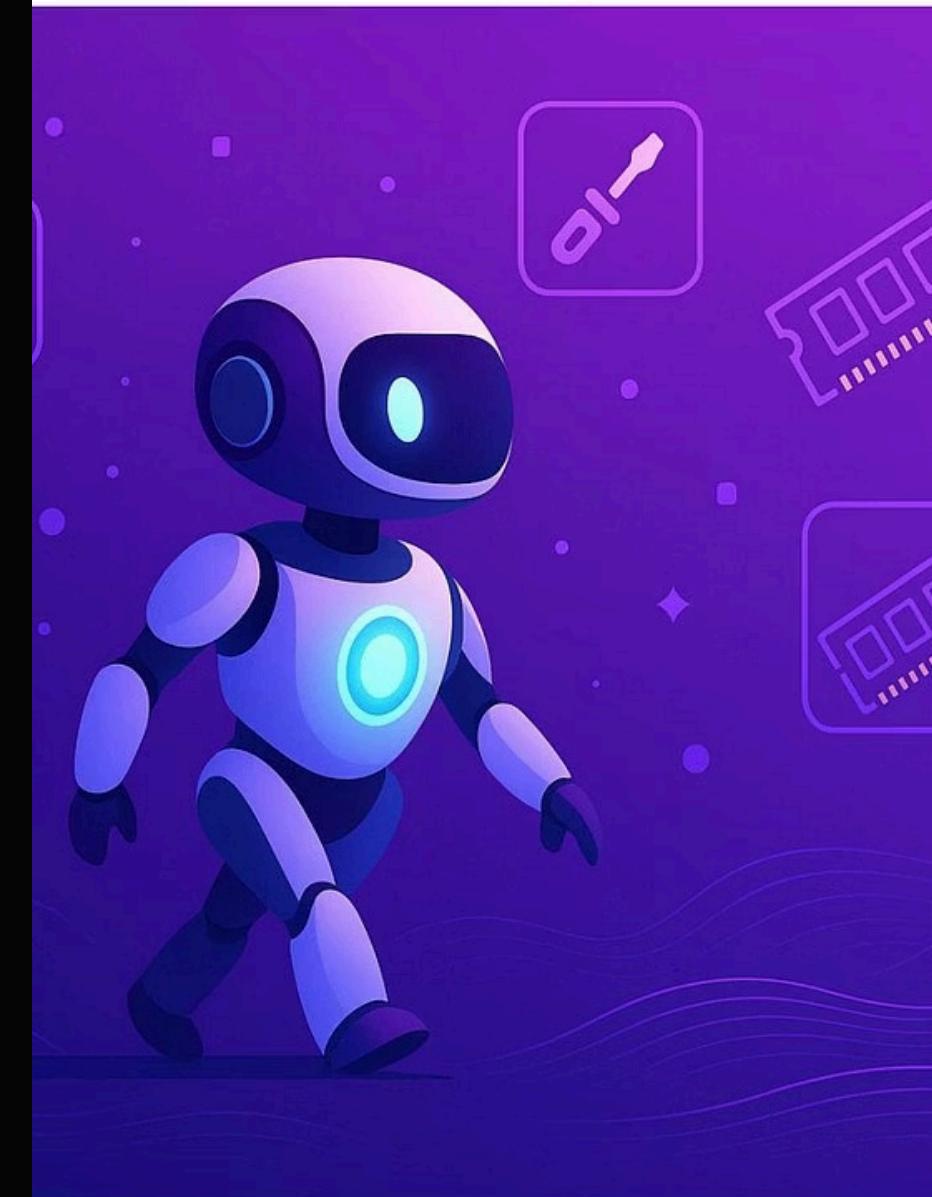
Duração

6 horas de desenvolvimento focado

Nível

Intermediário

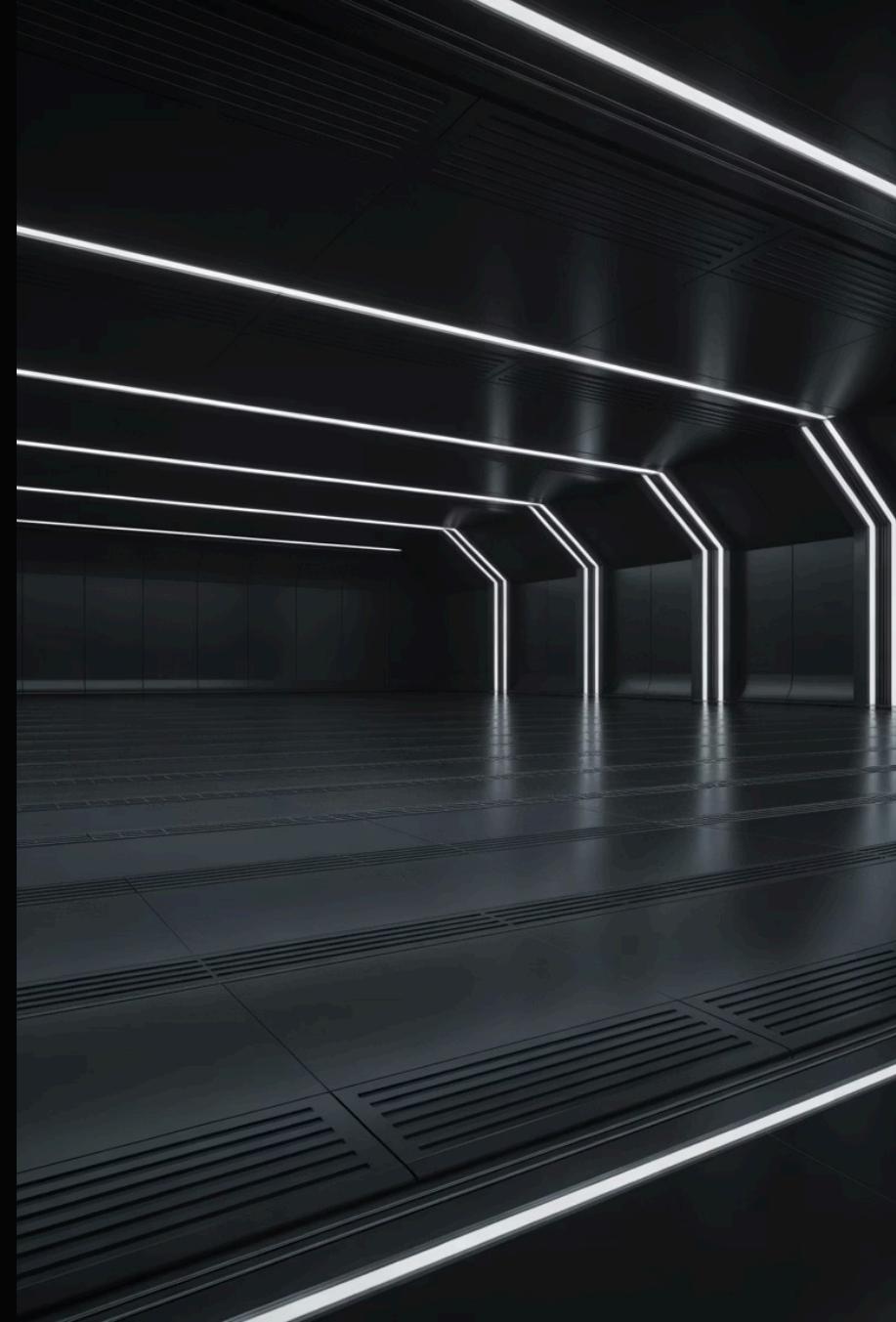
"Simplicidade é a sofisticação máxima."



# A Elegância da Simplicidade

Você dominou a complexidade da orquestração com LangChain. Agora, vamos explorar a elegância da simplicidade. Agno nos convida a pensar de forma diferente sobre a construção de agentes, focando em um design minimalista e em um fluxo de trabalho intuitivo. É a prova de que o poder não precisa vir da complexidade.

Depois de mergulhar na vastidão de opções do LangChain, Agno surge como um contraponto refrescante. Criado com a filosofia de ser "dolorosamente simples", este framework open-source foca em fazer uma coisa e fazê-la excepcionalmente bem: criar agentes de IA autônomos.



# A Filosofia Agno



## Simplicidade Extrema

A estrutura do código é linear e fácil de seguir. Não há abstrações complexas ou cadeias aninhadas. Você define um agente, dá a ele ferramentas e o executa.



## Foco no Agente

Agno é, antes de tudo, um framework para construir agentes. Ele não tenta ser um canivete suíço para todas as aplicações de LLM, mas sim a melhor ferramenta para criar entidades autônomas.



## Depuração Transparente

O log e o rastreamento do processo de pensamento do agente são claros e explícitos, tornando muito mais fácil entender por que um agente tomou uma determinada decisão.

# Quando Usar Agno?

## Prototipagem Rápida

Para criar e testar um agente autônomo rapidamente sem configuração complexa

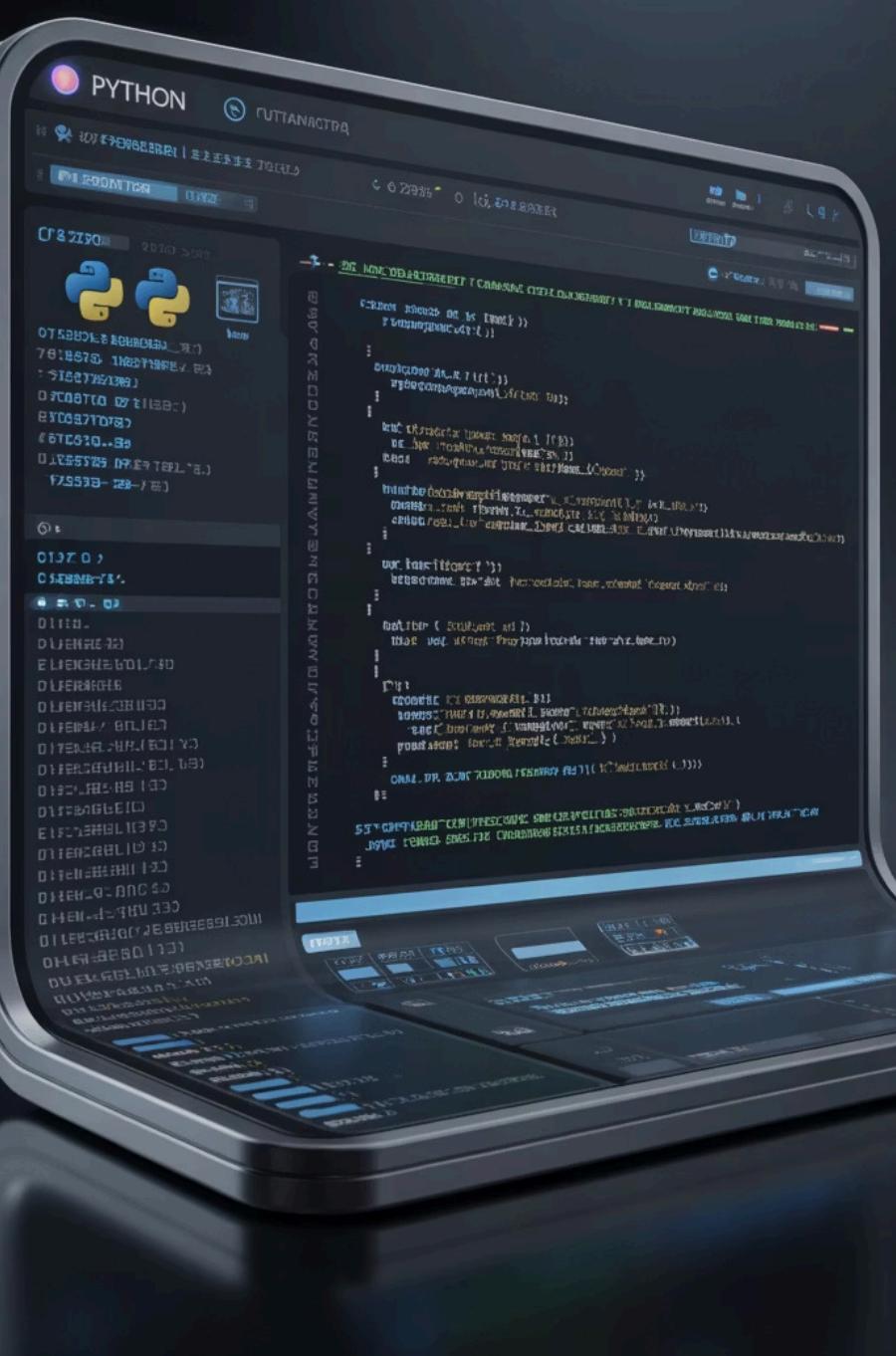
## Casos de Uso Focados

Quando seu objetivo principal é um agente que usa ferramentas, em vez de uma aplicação complexa de processamento de dados

## Clareza e Manutenibilidade

Quando a facilidade de entender e manter o código é uma prioridade máxima





# Criando seu Primeiro Agente Agno

Construir um agente com Agno é um processo notavelmente direto. A estrutura principal envolve a classe Agent e a definição de Tools.

01

## Importar Agent

```
from agno import Agent
```

02

## Definir Ferramentas

Crie funções Python simples e decore-as com @tool

03

## Instanciar o Agente

Crie uma instância de Agent, passando as ferramentas que ele pode usar

04

## Executar

Chame o método run() do agente com a sua solicitação

# Exemplo de Código Agno

```
from agno import Agent, tool
import requests

@tool
def search_web(query: str) -> str:
    """Busca na web por uma determinada consulta."""
    # Lógica para chamar uma API de busca
    return f"Resultados para: {query}"

# Instancia o agente com a ferramenta de busca
my_agent = Agent(tools=[search_web])

# Executa o agente
result = my_agent.run("Qual é a capital da França?")
print(result)
```

- ❑ **Insight:** Note a simplicidade. A docstring da função search\_web é usada automaticamente pelo Agno para que o LLM entenda o que a ferramenta faz. Isso torna o código auto-documentado e fácil de entender tanto para humanos quanto para a IA.



# RAG e Memória em Agno

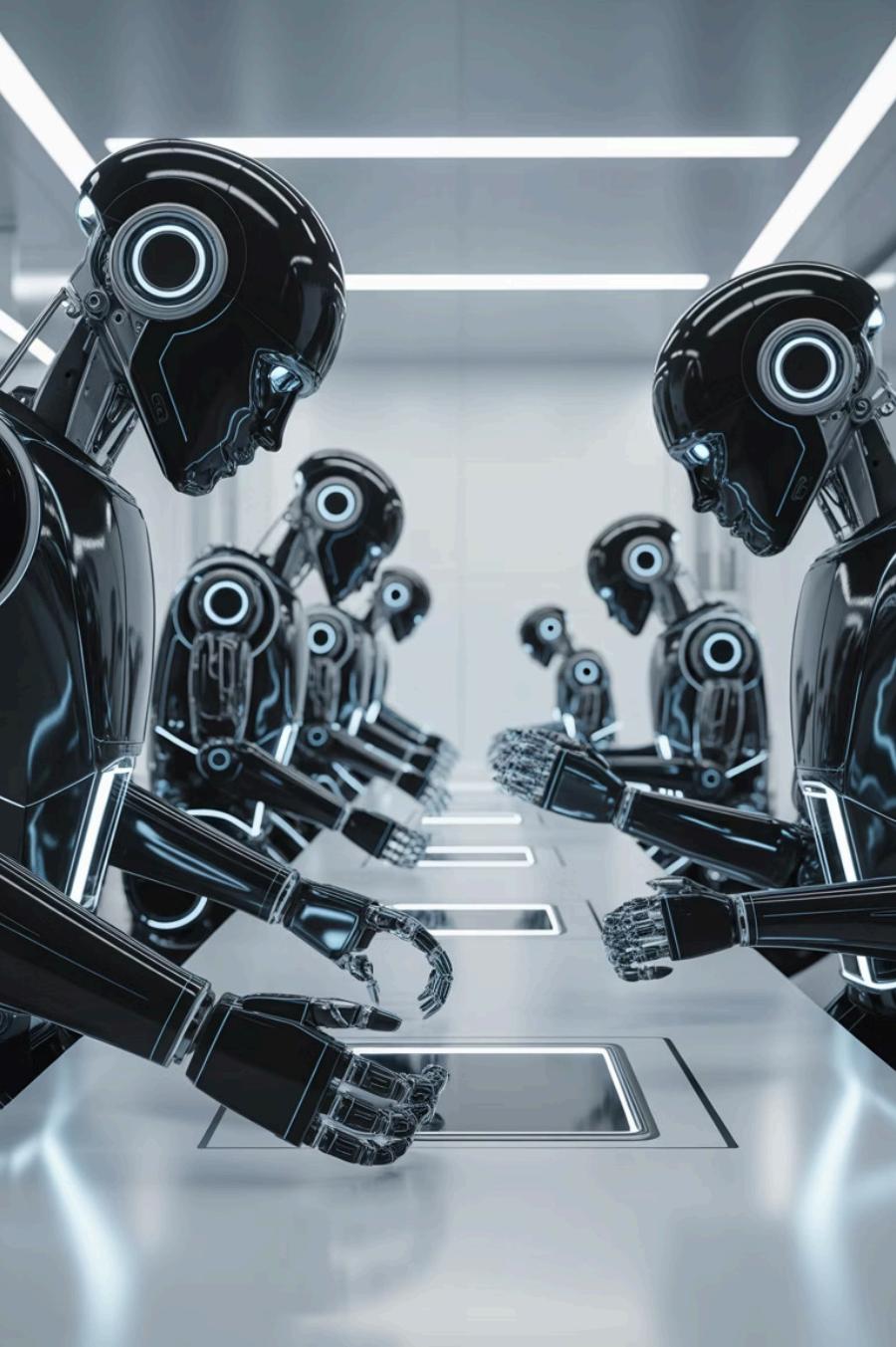
Embora simples, Agno é poderoso e suporta os mesmos padrões avançados que vimos em LangChain, como RAG e memória, mas com sua própria abordagem minimalista.

## RAG com Agno

Para implementar RAG, você simplesmente cria uma ferramenta (@tool) que realiza a busca em um banco de dados vetorial. O agente aprenderá a usar essa ferramenta sempre que precisar de informações de uma base de conhecimento específica.

## Memória Persistente

Agno suporta memória através do Storage, que pode ser configurado para salvar o histórico de interações em arquivos locais ou em um banco de dados. Isso permite que o agente mantenha o contexto ao longo de múltiplas execuções.



# Sistemas Multi-Agentes em Agno

Assim como em outros frameworks, Agno permite a criação de sistemas onde múltiplos agentes colaboram. A abordagem de Agno para isso é, novamente, baseada na simplicidade: um agente pode ser uma ferramenta para outro agente.

- 1
- 2
- 3

## Agente Gerente

Recebe o objetivo principal do usuário e orquestra os especialistas

## Agente de Pesquisa

Especializado em buscar informações na web usando ferramentas de busca

## Agente de Escrita

Especializado em redigir textos com base nas informações fornecidas

O Gerente pode primeiro chamar o Agente de Pesquisa para coletar dados e, em seguida, passar esses dados para o Agente de Escrita para compilar um relatório.

# Resumo: Agno

## Filosofia

Simplicidade, foco no agente e depuração transparente

## Estrutura Básica

Defina funções Python, decore-as com @tool e passe-as para uma instância de Agent

## Auto-documentação

As docstrings das ferramentas são usadas para que o LLM entenda sua funcionalidade

## RAG e Memória

Implementados de forma simples através de ferramentas customizadas e Storage

## Multi-Agentes

Criados tratando agentes especialistas como ferramentas para um agente gerente

# Projeto Prático: Assistente de Tarefas

01

## Objetivo

Construir um agente que possa gerenciar uma lista de tarefas simples (adicionar, listar, remover)

02

## Implementação

Crie três ferramentas Agno: add\_task(task: str), list\_tasks() e remove\_task(task\_number: int). Instancie um agente com essas ferramentas.

03

## Teste

Interaja com seu agente em linguagem natural. Peça: "Adicione 'comprar leite' à minha lista", "Mostre minhas tarefas", "Remova a primeira tarefa"

Este projeto destaca a beleza e a simplicidade de Agno na criação de agentes autônomos e funcionais com o mínimo de código.





## Módulo 5: Sistemas Multi-Agentes com CrewAI

Duração

8 horas de orquestração avançada

Nível

Avançado

"Sozinho você vai rápido. Em equipe, você vai longe."

# A Maestria: Orquestrando Equipes de IA

Você evoluiu de um conversador para um construtor, e de um construtor para um arquiteto. Agora, você se tornará um orquestrador. Este é o ápice da engenharia de agentes: a criação de sistemas onde múltiplos especialistas de IA colaboram, delegam e resolvem problemas que nenhum agente conseguia sozinho.

Bem-vindo à sala de comando. Até agora, trabalhamos com agentes agindo de forma isolada ou em sequências simples. CrewAI eleva o jogo ao introduzir um framework robusto para a colaboração entre agentes. Inspirado em equipes humanas eficazes, ele nos permite definir papéis, delegar tarefas e orquestrar um processo colaborativo para resolver problemas complexos.



# Por que Sistemas Multi-Agentes?



## Especialização

Cada agente pode ser um especialista em um domínio específico (pesquisa, escrita, análise de código), levando a um desempenho de maior qualidade em cada subtarefa



## Paralelismo

Múltiplas tarefas podem ser executadas em paralelo por diferentes agentes, acelerando significativamente a resolução do problema



## Robustez

Se um agente falha, outro pode potencialmente assumir sua função, tornando o sistema mais resiliente a erros



## Modularidade

É mais fácil desenvolver, depurar e manter agentes menores e especializados do que um único agente monolítico e complexo

# Arquitetura CrewAI

CrewAI formaliza a colaboração entre agentes através de alguns conceitos-chave que espelham equipes humanas eficazes.

## Agents

Os especialistas da sua equipe. Cada agente tem um role (papel), um goal (objetivo), um backstory (contexto) e tools (ferramentas)

## Tasks

As tarefas específicas que precisam ser concluídas. Cada tarefa tem uma description, um agent designado e um expected\_output

## Tools

As mesmas ferramentas que vimos nos frameworks anteriores. Funções que os agentes usam para interagir com o mundo exterior

## Crew

A equipe completa, composta pelos agents e tasks

## Process

Define como as tarefas serão executadas: Sequencial (uma após a outra) ou Hierárquico (um gerente orquestra dinamicamente)

# Exemplo de Código CrewAI

## VEJA NA PRÁTICA (Código):

```
from crewai import Agent, Task, Crew, Process

# Agente 1: Pesquisador
researcher = Agent(
    role='Pesquisador de Mercado',
    goal='Encontrar as últimas tendências em IA',
    backstory='Você é um especialista em analisar o mercado de tecnologia',
    tools=[search_tool]
)

# Agente 2: Escritor
writer = Agent(
    role='Escritor de Conteúdo',
    goal='Escrever um artigo de blog envolvente',
    backstory='Você é um redator de tecnologia com talento para storytelling',
    tools=[]
)

# Tarefa 1: Pesquisar
research_task = Task(
    description='Pesquise e resuma as 3 principais tendências de IA para 2024',
    agent=researcher,
    expected_output='Um resumo de 3 parágrafos sobre as tendências'
)

# Tarefa 2: Escrever
write_task = Task(
    description='Use o resumo da pesquisa para escrever um artigo de 500 palavras',
    agent=writer,
    expected_output='Um artigo de blog formatado em Markdown'
)

# Montando a Crew
blog_crew = Crew(
    agents=[researcher, writer],
    tasks=[research_task, write_task],
    process=ProcessSEQUENTIAL
)

# Executando a Crew
result = blog_crew.kickoff()
```

# Design de Agentes Especializados

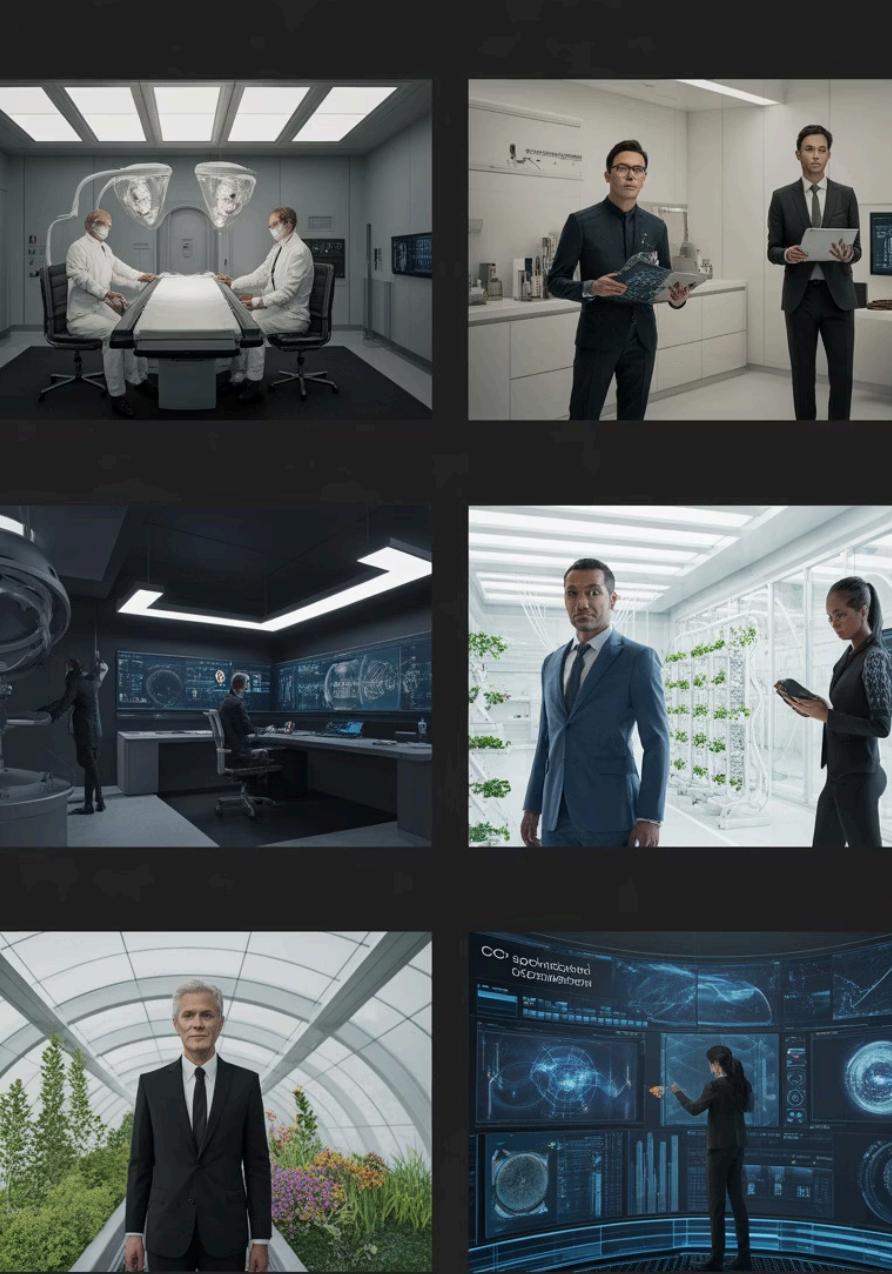
## Princípio da Responsabilidade Única

Cada agente deve ter um papel claro e focado. Evite criar agentes "faz-tudo". Um agente especializado em pesquisa não deve também ser responsável por escrita e análise de dados.

## Backstory Detalhado

Invista tempo criando um backstory rico. Dê ao seu agente uma história, uma personalidade e uma área de especialização. Isso funciona como um mega-prompt que guia todo o seu comportamento.

- **Insight:** A beleza do CrewAI está na clareza da definição de papéis e responsabilidades. O backstory de um agente é um poderoso mecanismo de prompting que o prepara para executar sua função com a persona e o conhecimento corretos, melhorando drasticamente a qualidade do seu trabalho.



# Definição de Tarefas Claras

## Descrição Explícita

A descrição da tarefa deve ser inequívoca. Diga ao agente exatamente o que fazer, sem ambiguidades

## Contexto e Dependências

Para tarefas sequenciais, você pode passar o resultado de tarefas anteriores usando {task.output} na descrição da tarefa seguinte

## Expected Output

Seja muito específico sobre o formato e o conteúdo do resultado esperado. Isso ajuda o agente a entender o que é "concluído"



# Processo Hierárquico

No processo hierárquico, você designa um agente como manager\_llm. Este gerente não executa tarefas diretamente, mas analisa o objetivo geral e delega as tarefas para os agentes apropriados na ordem que julgar mais eficiente.

Isso permite uma flexibilidade muito maior e é ideal para problemas complexos onde o fluxo de trabalho não é linear. O gerente pode:

- Avaliar qual agente é mais adequado para cada subtarefa
- Decidir a ordem de execução dinamicamente com base nos resultados intermediários
- Redistribuir tarefas se um agente falhar ou produzir resultados insatisfatórios
- Coordenar múltiplos agentes trabalhando em paralelo



# Observabilidade com AgentOps

Um dos maiores desafios em sistemas multi-agentes é entender o que está acontecendo. Por que um agente tomou uma decisão? Onde está o gargalo? CrewAI se integra nativamente com o AgentOps, uma plataforma de observabilidade projetada para agentes de IA.

## TM Rastreamento de Execução

Visualize o fluxo completo de tarefas, incluindo os pensamentos, ações e observações de cada agente

## \$ Análise de Custos

Monitore o consumo de tokens e os custos de API em tempo real para otimizar gastos

## 00 Métricas de Desempenho

Analise a latência, a taxa de sucesso e outras métricas para otimizar sua crew

# Resumo: CrewAI

## Sistemas Multi-Agentes

Equipes de agentes especializados superam agentes generalistas em problemas complexos

## Arquitetura

Composta por Agents, Tasks, Tools e Process (Sequencial ou Hierárquico)

## Design de Agentes

Foco na especialização através de role, goal e um backstory detalhado

## Processos

Sequencial para fluxos lineares e Hierárquico para orquestração dinâmica por um gerente

## Observabilidade

Integração com AgentOps para rastrear, depurar e otimizar desempenho e custos



# Projeto Prático: Agência de Conteúdo Automatizada

01

## Objetivo

Criar uma crew que, a partir de um tópico, pesquisa, escreve e edita um artigo de blog completo

02

## Agentes

Pesquisador (busca informações), Escritor (cria rascunho), Editor (revisa e corrige)

03

## Tarefas

Tarefa de Pesquisa → Tarefa de Escrita (usa output da pesquisa) → Tarefa de Edição (usa output da escrita)

04

## Execução

Use processo Sequencial. Dê um tópico (ex: "O impacto da IA na medicina") e observe a crew produzir um artigo completo

Este projeto é uma demonstração impressionante do poder da colaboração entre agentes e simula um fluxo de trabalho do mundo real de forma totalmente autônoma.

# Módulo 6: Model Context Protocol (MCP)

Duração

6 horas de integração avançada

Nível

Avançado

"Padronização é o caminho para a escalabilidade."





# Construindo Pontes Universais

Você construiu agentes e orquestrou equipes. Agora, você vai construir as pontes que conectam seus agentes a qualquer ferramenta no universo digital. O Model Context Protocol (MCP) é a chave para a interoperabilidade universal, transformando qualquer API, script ou sistema legado em uma ferramenta acessível para a sua IA.

Este módulo aborda um dos desafios mais críticos na engenharia de agentes: a comunicação com o mundo exterior. Cada ferramenta e API tem sua própria estrutura, e a criação de integrações customizadas é um trabalho constante. O MCP surge como uma proposta de padrão aberto para resolver isso.



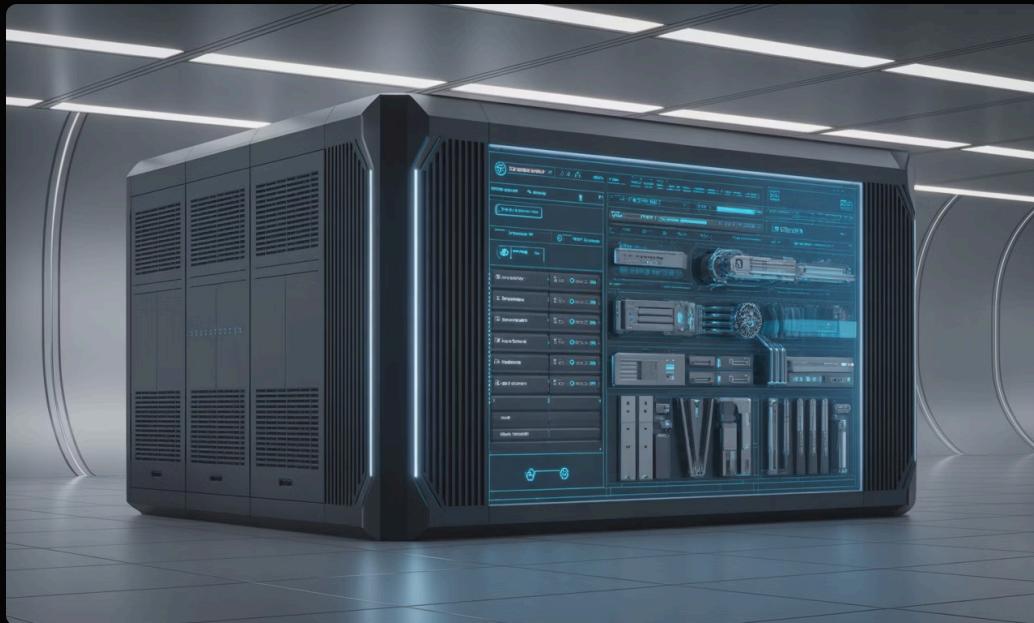
# O Problema que o MCP Resolve

Atualmente, para que um agente LangChain ou CrewAI use uma nova API, um desenvolvedor precisa escrever um "wrapper" ou "toolkit" específico. Se você então quiser usar essa mesma ferramenta com outro sistema de IA (como o Claude Desktop ou o CursorAI), você precisa escrever outro wrapper. O MCP elimina essa redundância.

Com o MCP, você expõe suas ferramentas em um servidor MCP. Qualquer cliente compatível com MCP pode então se conectar a esse servidor, descobrir automaticamente quais ferramentas estão disponíveis (junto com suas descrições e parâmetros) e usá-las sem precisar de código de integração customizado.

# Arquitetura Cliente-Servidor do MCP

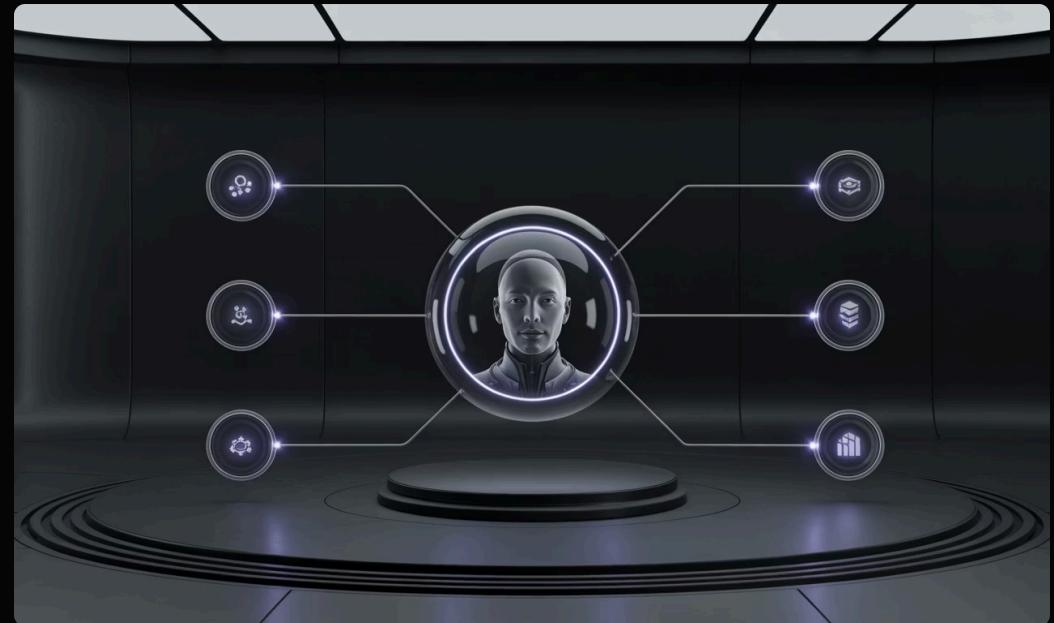
Servidor MCP (Tool Provider)



É um servidor web que expõe uma ou mais ferramentas. Ele tem um endpoint principal (/mcp) que retorna uma lista de ferramentas disponíveis, suas descrições e como chamá-las. É o seu catálogo de serviços.

Pense no MCP como o padrão USB para ferramentas de IA: qualquer dispositivo (ferramenta) que siga o padrão pode ser conectado a qualquer computador (agente).

Cliente MCP (Agent)



É o sistema de IA (um agente CrewAI, um chatbot, uma IDE) que sabe como ler o catálogo do servidor MCP e fazer chamadas para as ferramentas de acordo com a especificação.