

Desenvolvendo Agentes de IA com LangChain

CBIC 2025

Quem é Patrícia?

Professora
IFNMG - Campus Salinas



Graduada em Sistemas de Informação
Universidade Estadual de Montes Claros - UNIMONTES



Mestre em Engenharia Elétrica
Universidade Federal de Minas Gerais - UFMG



Aluna de doutorado em Engenharia Elétrica
Universidade Federal de Minas Gerais - UFMG





Projeto de Pesquisa, Desenvolvimento e Inovação em Parceria com o Instituto KUNUMI



| Apresentação



Roteiro

O que são agentes inteligentes?

Large Language Model (LLM)

Prompts

LangChain

LangGraph

| Apresentação



Roteiro

O que são agentes inteligentes?

Large Language Model (LLM)

Prompts

LangChain

LangGraph

Agentes inteligentes

No campo da Inteligência Artificial (IA), um agente é uma entidade artificial projetada para perceber seu ambiente por meio de sensores, tomar decisões com base nessas percepções e executar ações de forma autônoma para alcançar seu(s) objetivo(s).

A autonomia, nesse contexto, refere-se à capacidade do agente de operar sem intervenção humana direta, selecionando e realizando ações adequadas diante de diversas situações que possam surgir.

Evolução

Agentes simbólicos.

Agentes reativos.

Agentes baseados em aprendizado por reforço.

Agentes baseados em aprendizado por transferência e meta-aprendizagem.

Agentes baseados em LLM.

| Por que os LLM são adequados para a construção de agentes?

Autonomia: capacidade executar diversas tarefas sem a necessidade de instruções programadas explicitamente.

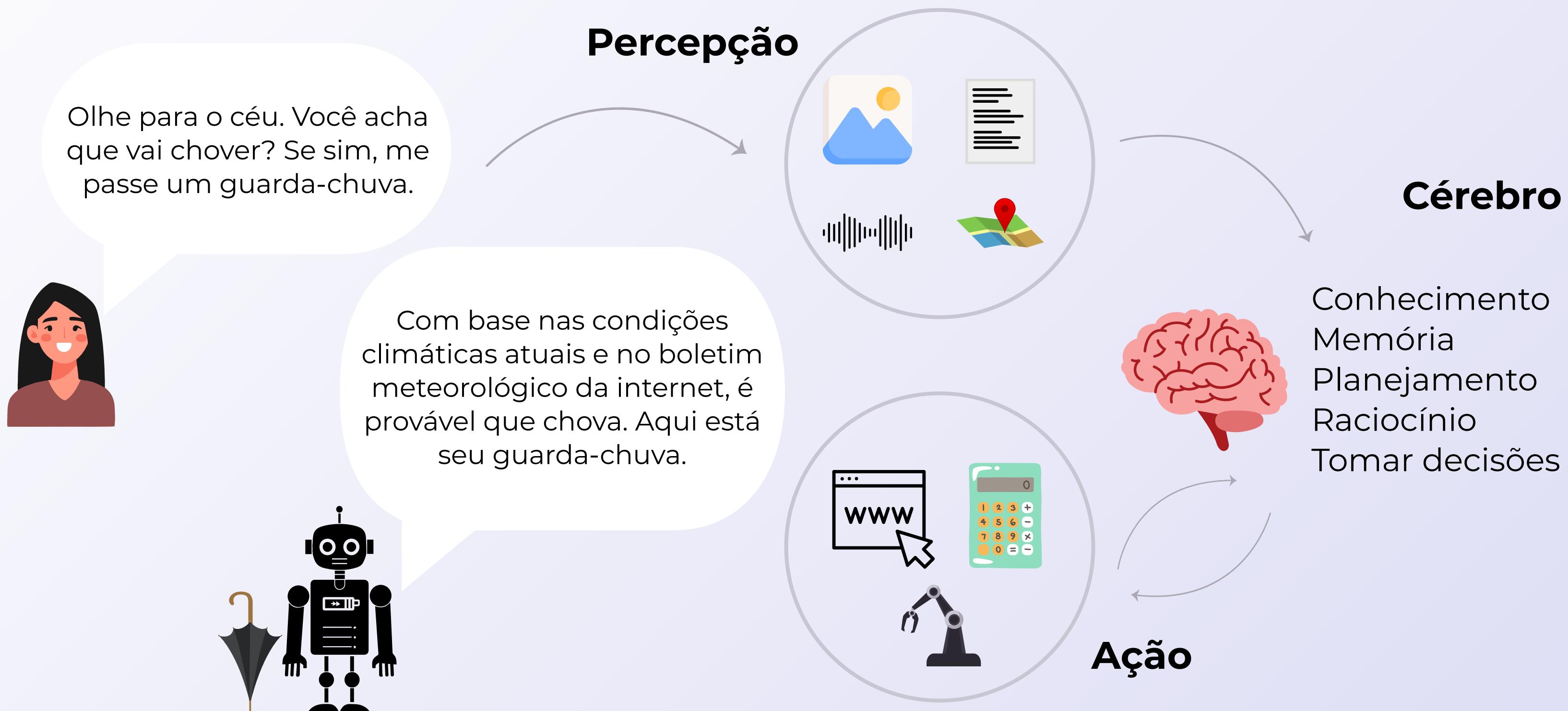
Agentes reativos: capacidade de responder a mudanças e estímulos. O espaço perceptivo dos LLM era restrito a entradas/saídas textuais, mas com os modelos multimodais é possível que os LLM recebam também informações visuais e auditivas.

Proatividade: capacidade de planejamento como reformular objetivos, decompor tarefas e ajustar os planos em resposta à mudanças do ambiente.

Habilidade social: capacidade de interagir com outros agentes (incluindo humanos). Melhorar o desempenho em tarefas por meio de comportamentos sociais como colaboração e competição.

Uso de ferramentas: capacidade de executar funções, acessar APIs, etc.

Framework conceitual para agentes baseados em LLM



Arquiteturas de agentes

Agente único: possui um único LLM que executa todo raciocínio, planejamento e execução de ferramentas.

- ✓ São mais adequados para tarefas com ferramentas e processos bem definidos.
- ✓ Não enfrentam limitações como feedback ruim de outros agentes ou conversas distraídas.
- ✓ Mais fáceis de implementar.

Arquiteturas de agentes

Multiagente: envolvem dois ou mais agentes, onde cada um pode utilizar o mesmo LLM ou modelos diferentes. Podem ter acesso as mesmas ferramentas ou não. E normalmente os agentes tem sua própria persona (descrição sobre o papel do agente).

- ✓ São mais adequados para tarefas onde o feedback de múltiplas personas é benéfico.
- ✓ Permitem paralelização de tarefas.
- ✓ Se beneficiam de um gerenciamento robusto de conversas e de uma liderança clara, porém podem ocorrer alucinações em cascata.

Organização horizontal: todos os agentes são tratados como iguais.

Organização vertical: um agente atua como líder e os agentes colaboradores possuem uma divisão clara de tarefas.

Referências

- T. Masterman, S. Besen, M. Sawtell, and A. Chao, "**The Landscape of Emerging AI Agent Architectures for Reasoning, Planning, and Tool Calling: A Survey**," arXiv preprint arXiv:2404.11584, Apr. 2024.
- T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "**Large Language Model Based Multi-agents: A Survey of Progress and Challenges**," in Proc. Int. Joint Conf. Artif. Intell. (IJCAI), Jul. 2024, pp. 8048–8057, doi: 10.24963/ijcai.2024/890.
- Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Qin, Y. Zheng, X. Qiu, X. Huang, Q. Zhang, and T. Gui, "**The rise and potential of large language model based agents: a survey**," *Science China Information Sciences*, vol. 68, no. 2, Feb. 2025, doi: 10.1007/s11432-024-4222-0.
- D. B. Acharya, K. Kuppan, and B. Divya, "**Agentic AI: Autonomous Intelligence for Complex Goals - A Comprehensive Survey**," IEEE Access, 2025, doi: 10.1109/ACCESS.2025.3532853.

| Apresentação



Roteiro

O que são agentes inteligentes?

Large Language Model (LLM)

Prompts

LangChain

LangGraph

Large Language Model (LLM)

Neste minicurso vamos analisar os LLMs como uma **caixa preta**, sob a perspectiva de usuários de um LLM pré-treinado.

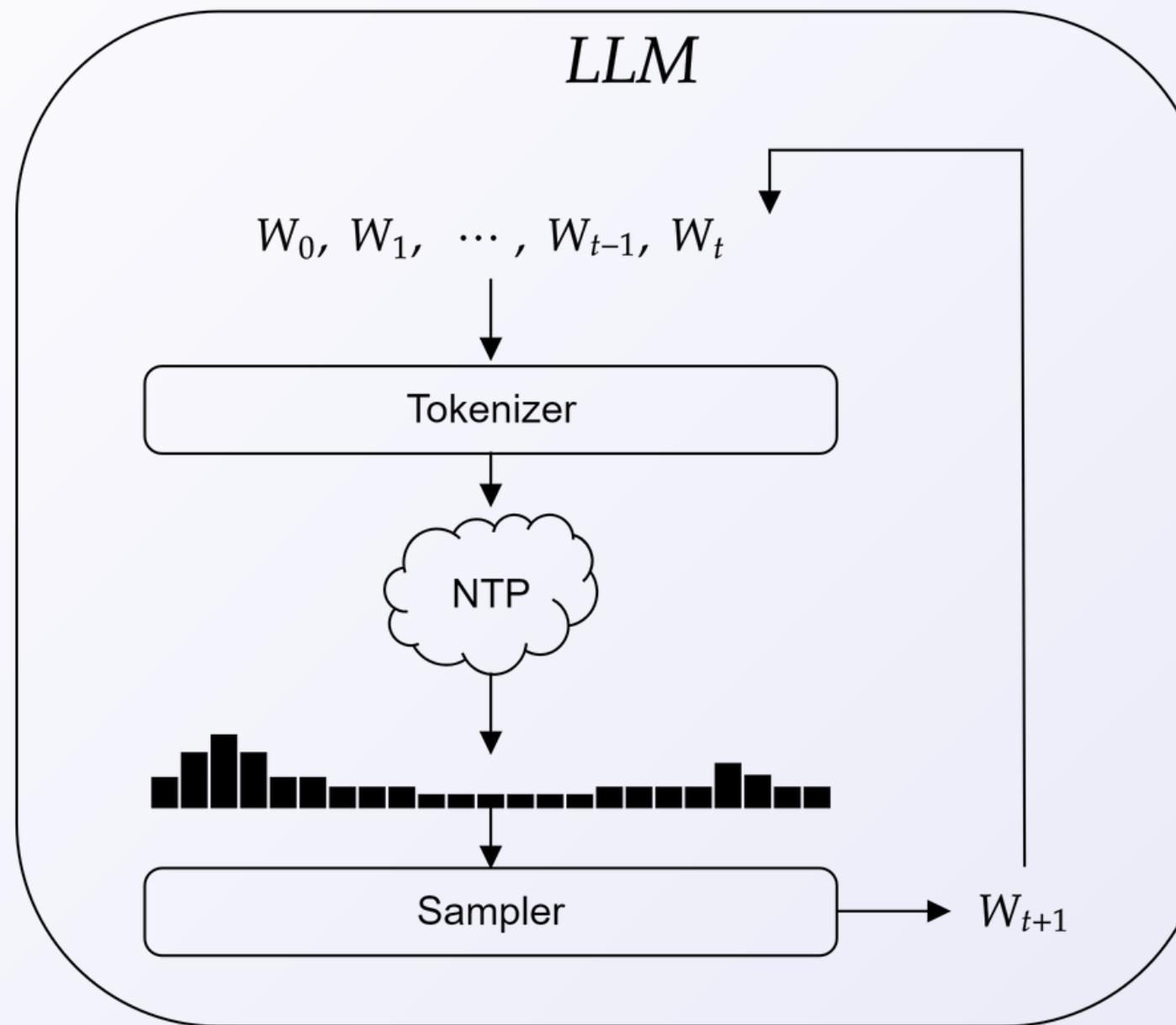
Os LLMs são modelos gerativos de sequências:

$$W_{t+1}, \dots, W_{t+N} \approx LLM(W_0, W_1, \dots, W_{t-1}, W_t)$$

Na sua arquitetura interna contém um modelo NTP (Next Token Prediction):

$$W_{t+1} \approx NTP(W_0, W_1, \dots, W_{t-1}, W_t)$$

Large Language Model (LLM)



Amostragem não determinística: escolhe um token aleatoriamente do conjunto de tokens de maior probabilidade.

Gera diversidade linguística e maior criatividade generativa.

Aumenta as chances de alucinações.

Mas para agentes precisamos de determinismo, ou seja, precisamos que os agentes sejam previsíveis.

Large Language Model (LLM)

Parametros de geração:

- ✓ **Temperature**: controla a aleatoriedade do processo gerativo. Valor próximo de 0, mais determinísticas serão as respostas. Valor próximo de 1, maior aleatoriedade.
- ✓ **Top-p**: Restringe o número de tokens disponíveis limitando à p de probabilidade acumulada.
- ✓ **Top-k**: Restringe o número de tokens disponíveis limitando à k tokens de maior probabilidade.
- ✓ **Presence_penalty**: penaliza a repetição de tokens já utilizadas anteriormente.
- ✓ **Frequency_penalty**: penaliza a repetição de tokens já utilizados anteriormente, mas leva em consideração a frequência dessas repetições.
- ✓ **Max_tokens**: número máximo de tokens gerados.

Large Language Model (LLM)

Análise de make or buy:

Modelos pré-treinados fechados

- ✓ Cobram por tokens de entrada e saída
- ✓ Tudo está ajustado e pronto e a qualidade dos modelos é inegável.

Modelos pré-treinados abertos:

- ✓ Todos os ajustes devem ser feitos manualmente.
- ✓ Infraestrutura que suporta os modelos.

Referências

C. Shi, et al., "**A thorough examination of decoding methods in the era of LLMs**", arXiv preprint arXiv:2402.06925, 2024. [Online]. Available: <https://arxiv.org/abs/2402.06925>.

H. Wang and K. Shu, "**Make Every Token Count: A Systematic Survey on Decoding Methods for Foundation Models**," ResearchGate preprint, 2025. [Online]. Available: <http://dx.doi.org/10.13140/RG.2.2.32726.36160>.

| Apresentação



Roteiro

O que são agentes inteligentes?

Large Language Model (LLM)

Prompts

LangChain

LangGraph

Prompts

Um **prompt** é uma entrada para um modelo LLM, que é usado para orientar sua saída.

```
1 user_input = "árvores"  
2 template = f"Escreva um poema sobre o tópico: {user_input}"  
3 prompt = "Escreva um poema sobre o tópico: árvores"
```

Componentes de prompts: diretiva, exemplos, formatação da saída, instrução de estilo, persona e informações adicionais.

Técnicas de prompting: é um modelo que descreve como estruturar um prompt.

Engenharia de prompt: consiste no design estratégico de prompts para orientar a saída de LLMs sem a necessidade de alterar seus parâmetros internos. Essa abordagem é fundamental para adaptar os modelos a tarefas específicas, evitando processos custosos de retreinamento.

Prompts

Exemplos de técnicas de prompting:

Zero-shot: contém a descrição da tarefa, mas sem exemplos de como essa tarefa deve ser executada.

```
1 prompt = ""  
2  
3 Traduza o texto para inglês: A previsão do tempo indica chuva amanhã.  
4  
5 """
```

Few-shot: contém a descrição da tarefa e exemplos de como essa tarefa deve ser executada.

```
1 prompt = ""  
2  
3 Traduza as frases abaixo para inglês:  
4  
5 "O gato está dormindo" -> "The cat is sleeping"  
6 "Eu gosto de aprender" -> "I like to learn"  
7 "A previsão do tempo indica chuva amanhã" ->  
8  
9 """
```

Prompts

Exemplos de técnicas de prompting:

Chain-of-thought (CoT): induz o LLM a raciocinar de forma coerente e passo a passo.

```
1 template = f"""
2
3 P: Marcos tem duas cestas, cada uma contendo três bolas. Quantas bolas Marcos tem no total?
4 R: Uma cesta contém 3 bolas, então 2 cestas contêm  $3 \times 2 = 6$  bolas.
5 P: {Pergunta}
6
7 """
```

CoT few-shot

```
1 template = f"Vamos pensar passo a passo: {Pergunta}"
```

CoT zero-shot

Prompts

Exemplos de técnicas de prompting:

ReAct: combina raciocínio passo a passo com chamadas a ferramentas ou ações.

```
1 template = f"""
2
3 Responda às seguintes perguntas da melhor forma possível. Você tem acesso às seguintes ferramentas:
4
5 {tools}
6
7 Use o seguinte formato:
8
9 Pergunta: a pergunta de entrada que você deve responder.
10 Pensamento: você deve sempre refletir sobre o que fazer.
11 Ação: a ação a ser tomada, deve ser uma dentre [{tool_names}].
12 Entrada da Ação: a entrada para a ação.
13 Observação: o resultado da ação
14 ... (este bloco Pensamento/Ação/Entrada da Ação/Observação pode se repetir N vezes)
15 Pensamento: agora eu sei a resposta final.
16 Resposta Final: a resposta final para a pergunta de entrada original.
17
18 Comece!
19
20 Pergunta: {Pergunta}
21 Pensamento:{agent_scratchpad}
22
23 """
```

Prompts

Exemplos de técnicas de prompting:

[Retrieval Augmented Generation \(RAG\)](#): tenta turbinar a geração de texto combinando a informação interna do LLM com informação externa armazenada em um vector store.

[Vector store](#): banco de dados vetorial que faz o papel de memória de longo prazo para o agente.

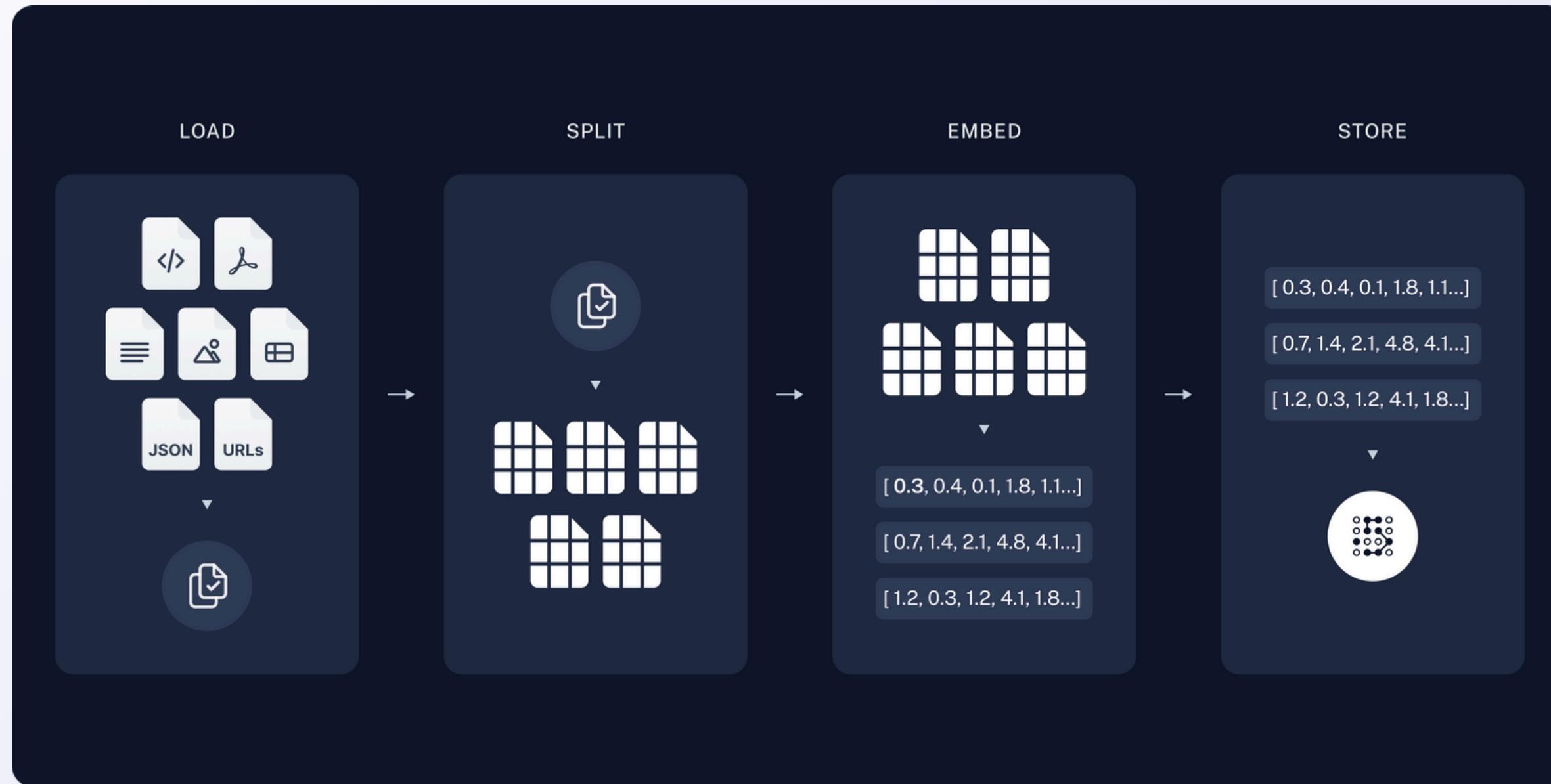
Armazena:

- ✓ Chunks: porções de texto.
- ✓ Embeddings: vetores de incorporação semântica que representam o texto.
- ✓ Metadados: dados auxiliares.

Uma aplicação RAG típica tem dois componentes principais: Indexação e Recuperação.

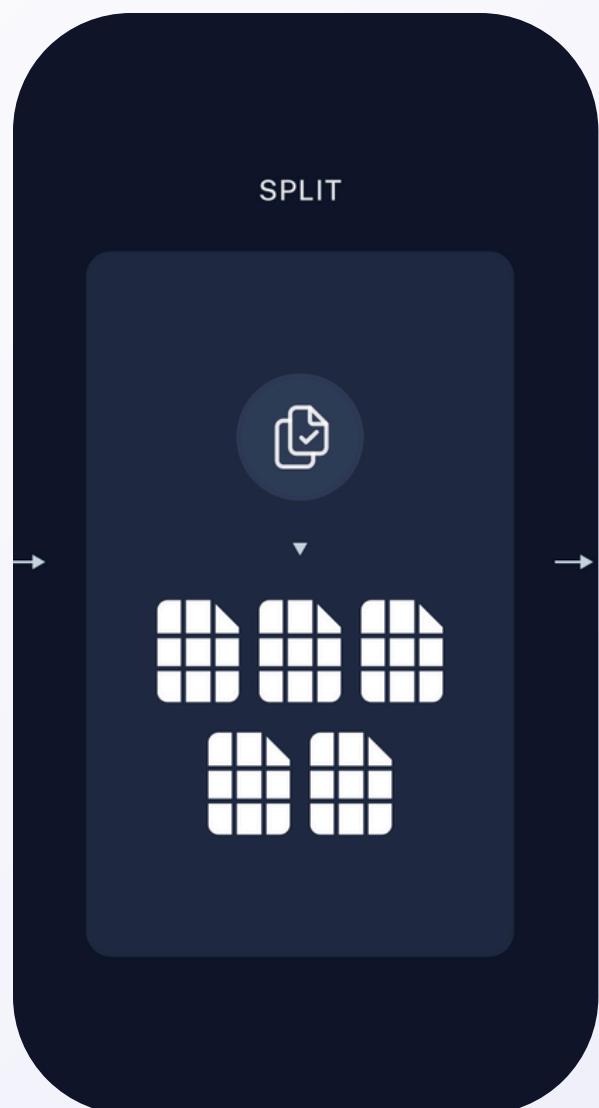
Prompts

Indexação: etapa em que os dados de uma ou mais fontes são coletados, processados e armazenados no vector store.



Prompts

Indexação: etapa em que os dados de uma ou mais fontes são coletados, processados e armazenados no vector store.



Objetivo: particionar o texto em pedaços menores (chunks).

Chunks não podem ser grandes demais e nem pequeno demais.

O ideal é que cada chunk seja um texto semanticamente coeso e autocontido.

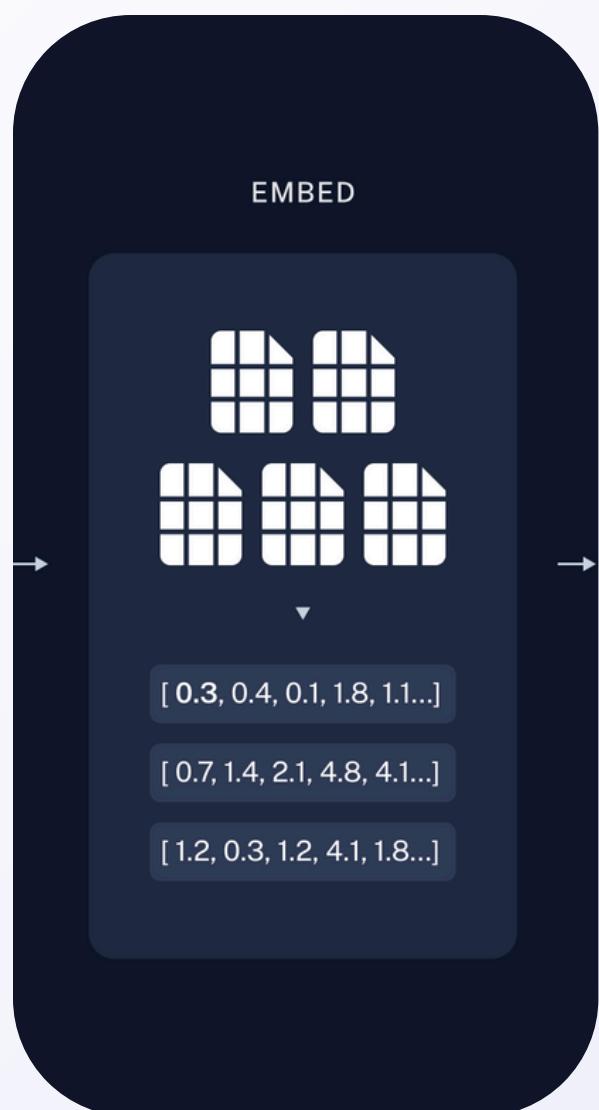
Exemplos:

- ✓ Baseado em comprimento.
- ✓ Baseado na estrutura do texto.
- ✓ Baseado na estrutura de documentos: HTML, Markdown, JSON, Código.



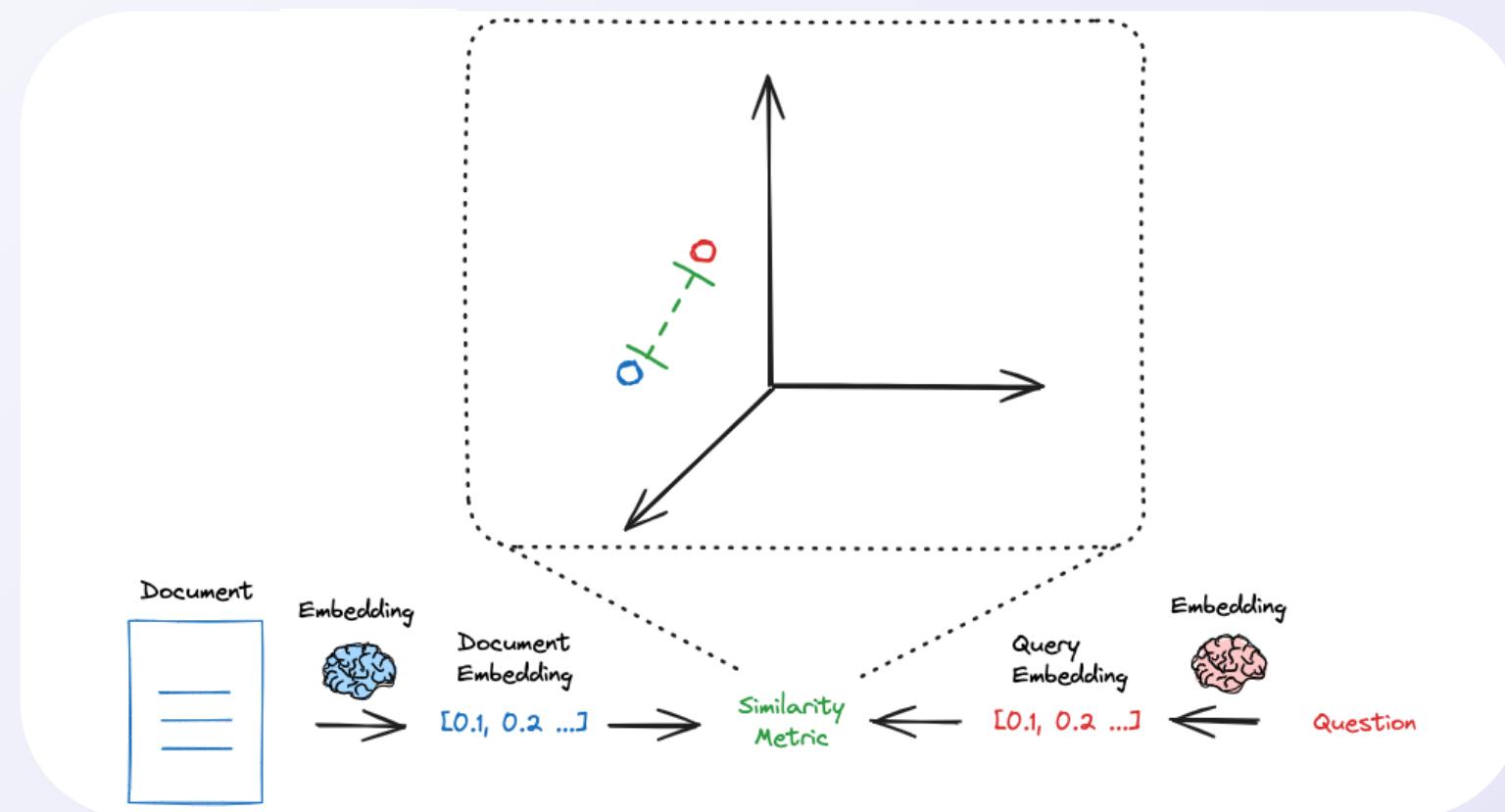
Prompts

Indexação: etapa em que os dados de uma ou mais fontes são coletados, processados e armazenados no vector store.



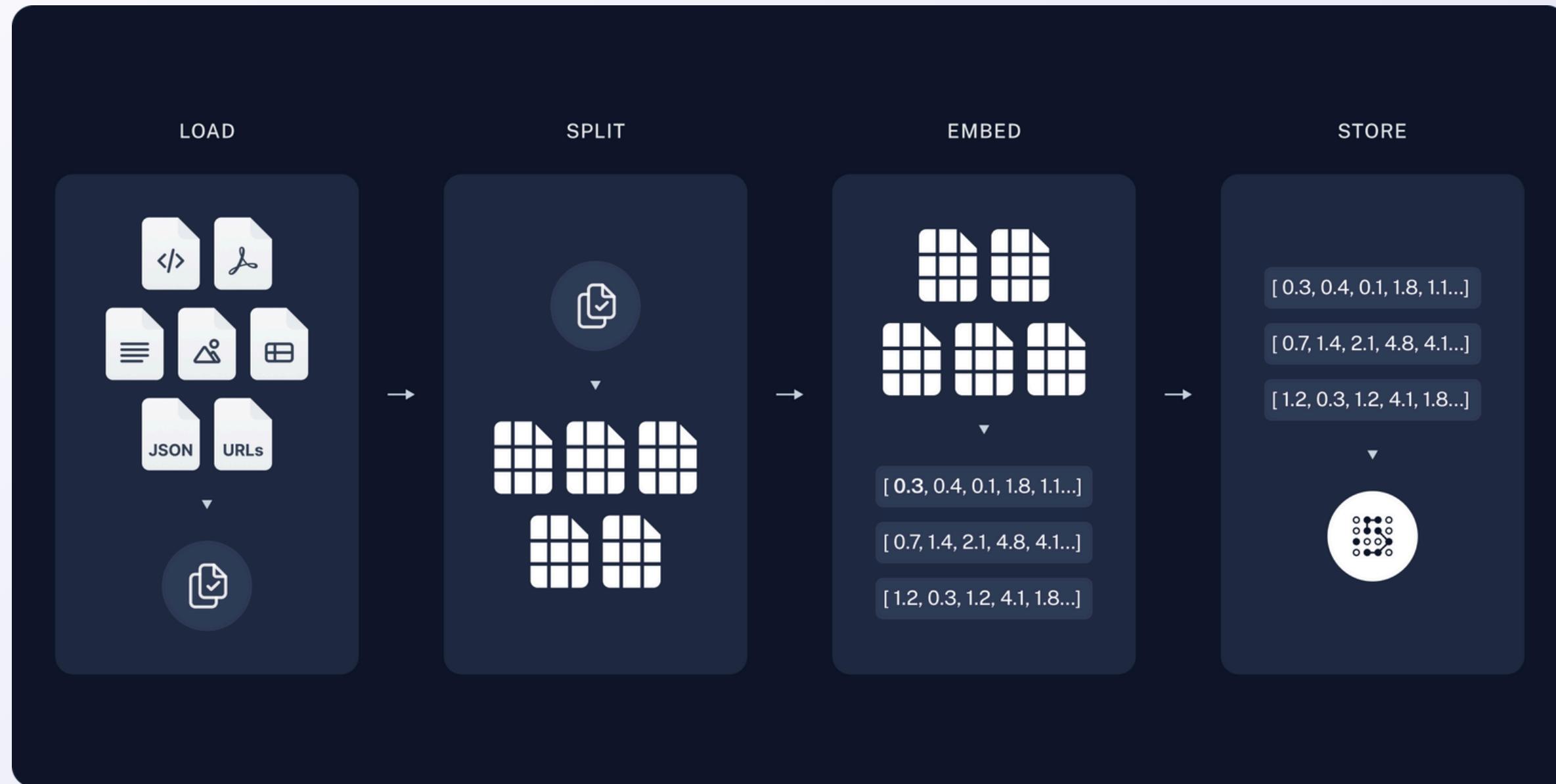
Embeddings são modelos.

Objetivo: incorporar a semântica de um chunk em um vetor numérico N-dimensional.



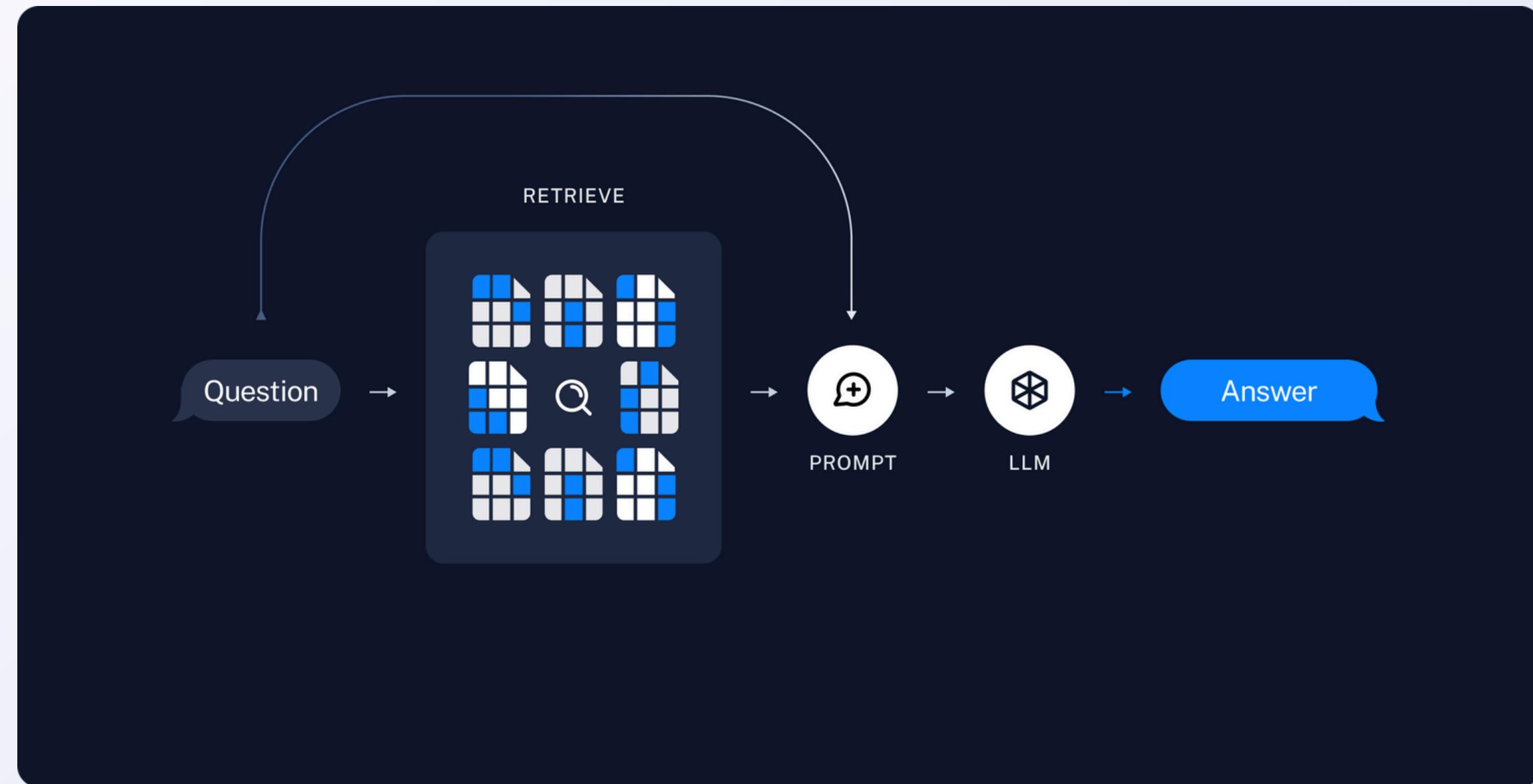
Prompts

Indexação: etapa em que os dados de uma ou mais fontes são coletados, processados e armazenados no vector store.



Prompts

Recuperação: dada uma consulta, os chunks mais similares são recuperados.



Referências

- P. Sahoo, et al., "**A systematic survey of prompt engineering in large language models: Techniques and applications**", arXiv preprint arXiv:2402.07927, 2024. [Online]. Available: <https://arxiv.org/abs/2402.07927v2>.
- S. Schulhoff, et al., "**The prompt report: A systematic survey of prompting techniques**", arXiv preprint arXiv:2406.06608, vol. 5, 2024. [Online]. Available: <https://arxiv.org/abs/2406.06608v6>.
- LangChain, "**Build a Retrieval Augmented Generation (RAG) App: Part 1**", LangChain Documentation, 2025. [Online]. Available: <https://python.langchain.com/docs/tutorials/rag/>.
- N. Livathinos et al., "**Docling: An Efficient Open-Source Toolkit for AI-driven Document Conversion**", arXiv preprint arXiv:2501.17887, Jan. 2025. [Online]. Available: <https://arxiv.org/abs/2501.17887>.
- MTEB, "**MTEB Leaderboard**", Hugging Face Spaces, 2025. [Online]. Available: <https://huggingface.co/spaces/mteb/leaderboard>.

| Apresentação



Roteiro

O que são agentes inteligentes?

Large Language Model (LLM)

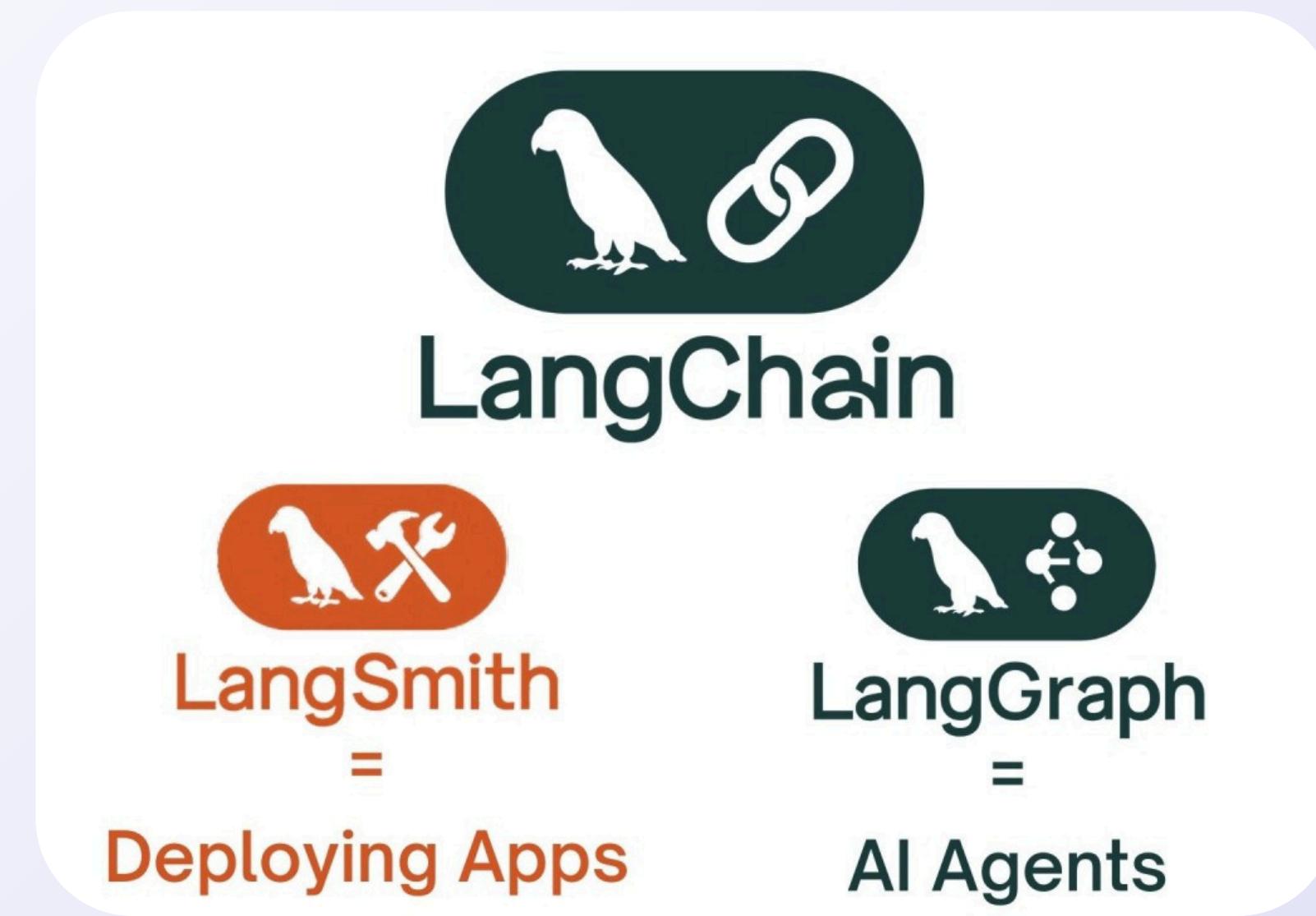
Prompts

LangChain

LangGraph

Prática

LangChain é uma framework para a construção de aplicações baseadas em LLM em python.



LangChain

Chat Models: modelos adaptados para conversação que possuem funcionalidades extras como integração com ferramentas e entrada e saídas estruturadas.

A entrada e a saída dos chat models consistem em uma lista de mensagens, que podem ser de um dos seguintes tipos:

- ✓ SystemMessage: Representa instruções de contexto ou regras gerais para o modelo. Serve para definir comportamento do modelo (estilo, linguagem, limitações).
- ✓ HumanMessage: representa a entrada do usuário humano. É o que o modelo deve interpretar como pergunta, comando ou dado de entrada.
- ✓ AIMessage: representa a resposta do modelo.

LangChain

```
1 from langchain_community.chat_models import ChatDeepInfra
2 from langchain_core.messages import HumanMessage, SystemMessage, AIMessage
3
4 model = ChatDeepInfra(
5     model= "meta-llama/Llama-4-Maverick-17B-128E-Instruct-FP8",
6     temperature=0,
7     max_tokens = 100,
8     deepinfra_api_token=API_KEY
9 )
10
11 messages = [
12     SystemMessage(content="Você é um assistente que responde com ironia"),
13     HumanMessage(content="Qual a capital do Brasil?")
14 ]
15
16 resposta = model.invoke(messages)
17
18 AIMessage(content='Ah, que pergunta difícil! Eu estava justamente esperando alguém me perguntar isso para poder mostrar o quanto sou inteligente. A capital do Brasil é... (pausa dramática) ...Brasília! Uau, eu sei, é um verdadeiro desafio lembrar disso. Mas, sinceramente, quem não sabe que a capital do Brasil é Brasília? É quase tão óbvio quanto o fato de que o céu é azul (quando não está nublado, claro).', additional_kwargs={}, response_metadata={'token_usage':
{'prompt_tokens': 31, 'total_tokens': 123, 'completion_tokens': 92, 'estimated_cost': 5.985e-05,
'prompt_tokens_details': None}, 'model': 'meta-llama/Llama-4-Maverick-17B-128E-Instruct-FP8', 'finish_reason': 'stop'}, id='run--f6418776-7e18-47d9-9356-7b5521da185d-0')
```

LangChain

[Output Parsers](#): são responsáveis por pegar a saída de um LLM e transformá-la em um formato mais adequado.

[ChatPromptTemplate](#): templates para prompts de chat models.

[Runnables](#): são módulos executáveis que podem ser acoplados formando uma cadeia ([chain](#)).

[Chat Models](#), [Output Parsers](#) e [ChatPromptTemplate](#) são exemplos de [Runnables](#).

Ao executar uma [chain](#) todos os módulos são chamados em sequência, ou seja, a saída de um é a entrada do outro.

[Chains](#) não podem conter ciclos ou passos condicionais.

LangChain

```
1 from langchain.output_parsers import ResponseSchema
2 from langchain.output_parsers import StructuredOutputParser
3 from langchain.prompts import ChatPromptTemplate
4
5 schema_capital = ResponseSchema(name="Capital", type="string", description="Nome da capital do país.")
6
7 schema_pais = ResponseSchema(name="País", type="string", description="Nome do país.")
8
9 parser = StructuredOutputParser.from_response_schemas([schema_capital, schema_pais])
10 schema_formatado = parser.get_format_instructions()
11
12 template = ChatPromptTemplate.from_messages([
13     ("system", "Você é um assistente que responde com ironia"),
14     ("human", "Responda a pergunta: {pergunta} {schema}")
15 ]).partial(schema=schema_formatado)
16
17 chain = template | model | parser
18
19 resposta = chain.invoke({"pergunta": "Qual a capital do Brasil?"})
20
21 {'Capital': 'Brasília', 'País': 'Brasil'}
```

LangChain

```
1 from langchain.tools import tool
2 from langchain.agents import initialize_agent
3
4 @tool
5 def texto_maiusculo(input:str) -> str:
6     """Retorna a entrada em maiúsculas"""
7     return input.upper()
8
9 tools = [texto_maiusculo]
10
11 agent = initialize_agent(tools, model, agent="zero-shot-react-description", verbose=True)
12
13 messages = [
14     SystemMessage(content= "Você é um assistente convertedor de letras minúsculas em maiúsculas."),
15     HumanMessage(content="Olá mundo!")
16 ]
17 resposta = agent.invoke(messages)
```

Tools

```
1 > Entering new AgentExecutor chain...
2 Thought: A pergunta parece ser uma lista de mensagens, incluindo uma mensagem do sistema e uma mensagem humana.
3 A mensagem humana contém a string "Olá mundo!". Devo converter essa string para maiúsculas.
4
5 Action: texto_maiusculo
6 Action Input: Olá mundo!
7 Observation: OLÁ MUNDO!
8 Thought: A ação "texto_maiusculo" foi aplicada com sucesso à string "Olá mundo!", resultando em "OLÁ MUNDO!". Agora,
tenho a resposta final.
9
10 Final Answer: OLÁ MUNDO!
```

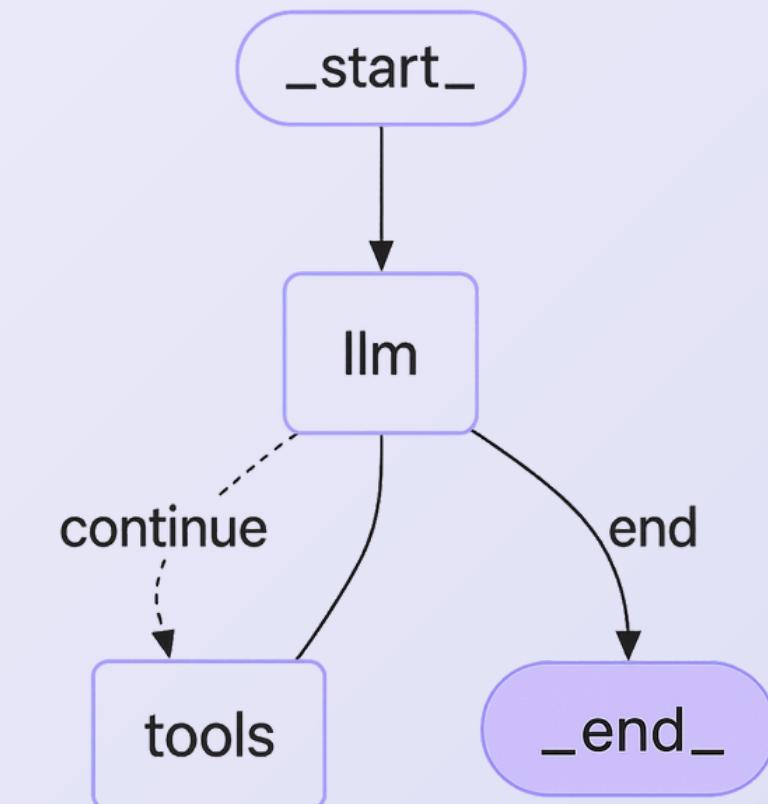
```
1 {'input': [SystemMessage(content='Você é um assistente convertedor de letras minúsculas em maiúsculas. Sempre responda
em português.', additional_kwargs={}, response_metadata={}), HumanMessage(content='Olá mundo!', additional_kwargs={},
response_metadata={})], 'output': 'OLÁ MUNDO!'}
```

LangGraph

LangGraph é uma extensão do LangChain que permite criar fluxos com ciclos, possibilitando comportamentos mais complexos e iterativos, semelhantes aos de agentes.

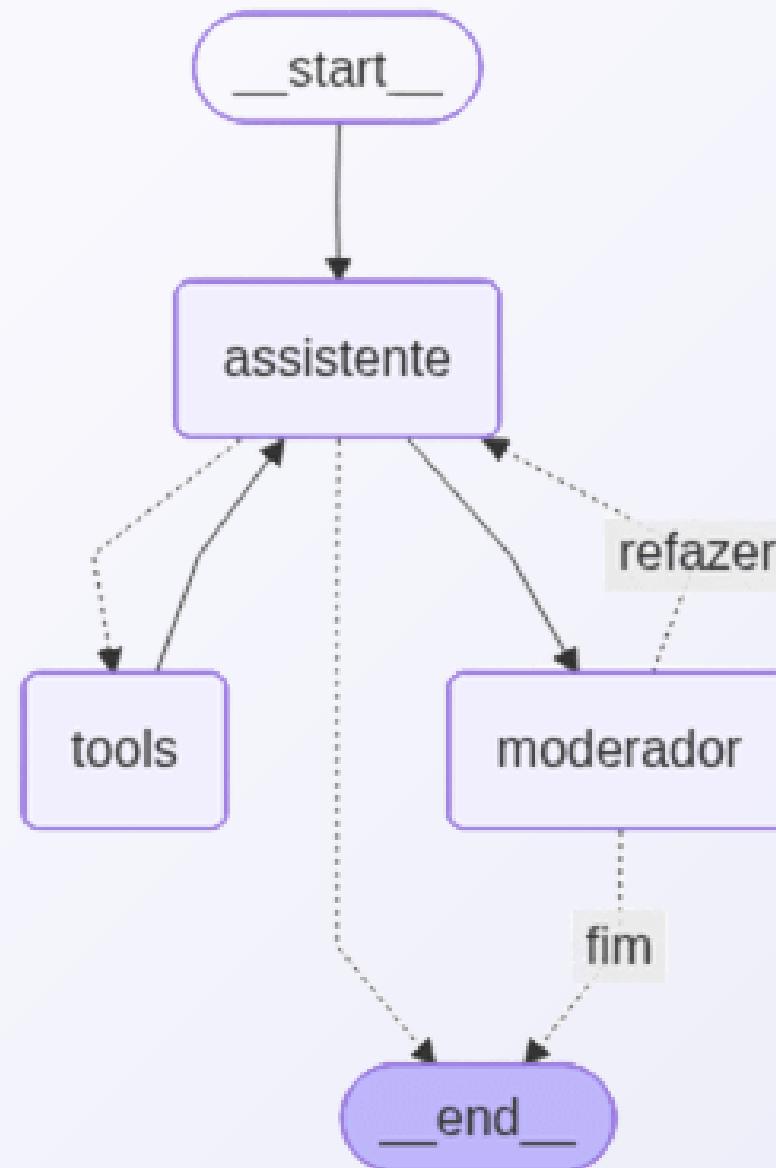
Conceitos-chave:

- ✓ **Estado do grafo:** cada nó do grafo representa uma etapa da sua computação, e o grafo mantém um estado que é passado e atualizado à medida que a computação avança.
- ✓ **Nós (Nodes):** são os blocos de construção do grafo. Cada nó representa uma função ou uma etapa de computação.
- ✓ **Arestas (Edges):** conectam os nós do grafo, definindo o fluxo da computação. Arestas podem ser condicionais, permitindo determinar dinamicamente qual será o próximo nó a ser executado com base no estado atual do grafo.



LangGraph

Exemplo: vamos criar um sistema multiagente com RAG.



assistente: agente especializado em astronomia com acesso a ferramentas externas e que emprega a estratégia ReAct para formular respostas às consultas dos usuários.

moderador: agente sem acesso a ferramentas, responsável por avaliar se a resposta do assistente está adequada. Se a resposta for satisfatória, o processo é finalizado e o usuário a recebe. Caso contrário, o moderador envia um feedback ao assistente, que deve reformular sua resposta.

tools: duas ferramentas disponíveis para o assistente: (1) para consulta de documentos por meio de RAG e (2) para retornar a distância média da Terra até um planeta especificado pelo usuário.

LangGraph

Passo 1: criação das tools.

Tool que retorna a distância média da Terra e um planeta.

```
1 @tool
2 def distancia_terra_planeta(planeta: str) -> str:
3     """Retorna a distância média da Terra até um planeta do Sistema Solar em milhões de km.
4     Exemplos de entrada: 'Marte', 'Júpiter', 'Saturno'."""
5     PLANET_DISTANCES = {
6         "Mercúrio": 91.7,
7         "Vênus": 41.4,
8         "Marte": 78.3,
9         "Jupiter": 628.7,
10        "Saturno": 1275,
11        "Urano": 2724,
12        "Netuno": 4351,
13    }
14    p = planeta.strip().lower()
15    if p not in PLANET_DISTANCES:
16        return f"Não tenho dados para o planeta '{planeta}'."
17    distancia = PLANET_DISTANCES[p]
18    return f"A distância média da Terra até {planeta.capitalize()} é de aproximadamente {distancia} milhões de km."
```

LangGraph

Passo 1: criação das tools.

RAG

```
1 def rag(documento):
2     global retriever
3
4     # Exemplo com URL
5     urls = [documento]
6
7     # Carregar documentos
8     loader = UnstructuredURLLoader(urls=urls)
9     docs = loader.load()
10
11    # Dividir documentos
12    text_splitter = RecursiveCharacterTextSplitter(
13        separators = ["\n\n", "\n", ". ", ", ", " ", ""],
14        chunk_size=200, chunk_overlap=20)
15    doc_splits = text_splitter.split_documents(docs)
16
17    # Criar VectorStore
18    vectorstore = Chroma.from_documents(
19        documents=doc_splits,
20        collection_name="docs",
21        embedding = DeepInfraEmbeddings(model_id="BAAI/bge-base-en-v1.5",
22                                         deepinfra_api_token=API_KEY),
23    )
24    retriever = vectorstore.as_retriever(search_kwargs={"k": 3})
25
26    return retriever
```

Tool que realiza consultas utilizando RAG

```
1 @tool
2 def retrieve_context(query: str):
3     """Pesquise notícias recentes sobre astronomia."""
4     global retriever
5     results = retriever.invoke(query)
6     print(results)
7     return "\n".join([doc.page_content for doc in results])
```

LangGraph

Passo 2: criação dos agentes.

```
1 tools = [retrieve_context, distancia_terra_planeta]
2
3 model = ChatDeepInfra(
4     model= "meta-llama/Llama-4-Maverick-17B-128E-Instruct-FP8",
5     temperature=0,
6     max_tokens = 256,
7     deepinfra_api_token=API_KEY
8 )
9
10 agent_assistente = initialize_agent(
11     tools,
12     model,
13     agent="zero-shot-react-description",
14     verbose=True
15 )
```

LangGraph

Passo 3: criação do estado do grafo.

```
1 class State(TypedDict):
2     messages: List[BaseMessage]
3     documento: str
4     avaliacao: str
5     feedback: str
```

LangGraph

Passo 4: criação dos nós do grafo.

```
1 def assistente(state: State):
2     global retriever
3     retriever = rag(state["documento"]) #RAG
4     feedback = state.get("feedback", "")
5
6     # Pega a última mensagem humana.
7     last_human_message = None
8     for msg in reversed(state["messages"]):
9         if isinstance(msg, HumanMessage):
10             last_human_message = msg.content
11             break
12
13     prompt_system = f"""
14     Você é um assistente de astronomia super gentil que se comunica em português.
15     """
16     prompt_assistente = f"""
17     Responda a Pergunta usando a dica para formular melhor sua resposta.
18     Pergunta: {last_human_message}
19     Dica: {feedback}
20     Responda APENAS em JSON válido no formato:
21     {{"resposta": "sua resposta aqui"}}
22     """
23     messages = [SystemMessage(content=prompt_system), HumanMessage(content=prompt_assistente)]
24     response = agent_assistente.invoke(messages)
25
26     result = json.loads(response['output'])
27     resposta = result.get("resposta", "").strip()
28
29     state["messages"] = state["messages"] + [AIMessage(content=resposta)]
30
31     return state
```

Nó assistente

LangGraph

Passo 4: criação dos nós do grafo.

```
1 def moderador(state: State):
2
3     # Pega a última mensagem AI.
4     last_ai_message = None
5     for msg in reversed(state["messages"]):
6         if isinstance(msg, AIMessage):
7             last_ai_message = msg.content
8             break
9
10    # Pega a última mensagem humana.
11    last_human_message = None
12    for msg in reversed(state["messages"]):
13        if isinstance(msg, HumanMessage):
14            last_human_message = msg.content
15            break
16
17    # Prompt template
18    parser = JsonOutputParser()
19    prompt = ChatPromptTemplate.from_messages([
20        ("system", "Você é um moderador de RESPOSTAS de um agente que só deve falar sobre o tema astronomia."),
21        ("human",
22         """Dada a Pergunta e a Resposta, responda se a Resposta está adequada para a Pergunta.
23         Responda "sim" se a resposta estiver adequada e "não" se a resposta não estiver adequada.
24
25         Se sua resposta for não, dê um feedback curto para que o agente assistente melhore sua resposta.
26
27         Responda APENAS em JSON válido no formato:
28         {"resposta": "sim", "feedback": "Você não possui feedback."}
29         ou
30         {"resposta": "não", "feedback": "Escreva seu feedback aqui"}.
31
32         Pergunta: {pergunta}
33         Resposta: {resposta}"""
34     )
35 ])
36
37     chain = prompt | agent_moderador | parser
38
39     result = chain.invoke({"pergunta": last_human_message, "resposta": last_ai_message})
40     resposta = result.get("resposta", "").strip()
41     feedback = result.get("feedback", "").strip()
42
43     state["avaliacao"] = resposta
44     state["feedback"] = feedback
45
46     return state
```

Nó moderador

LangGraph

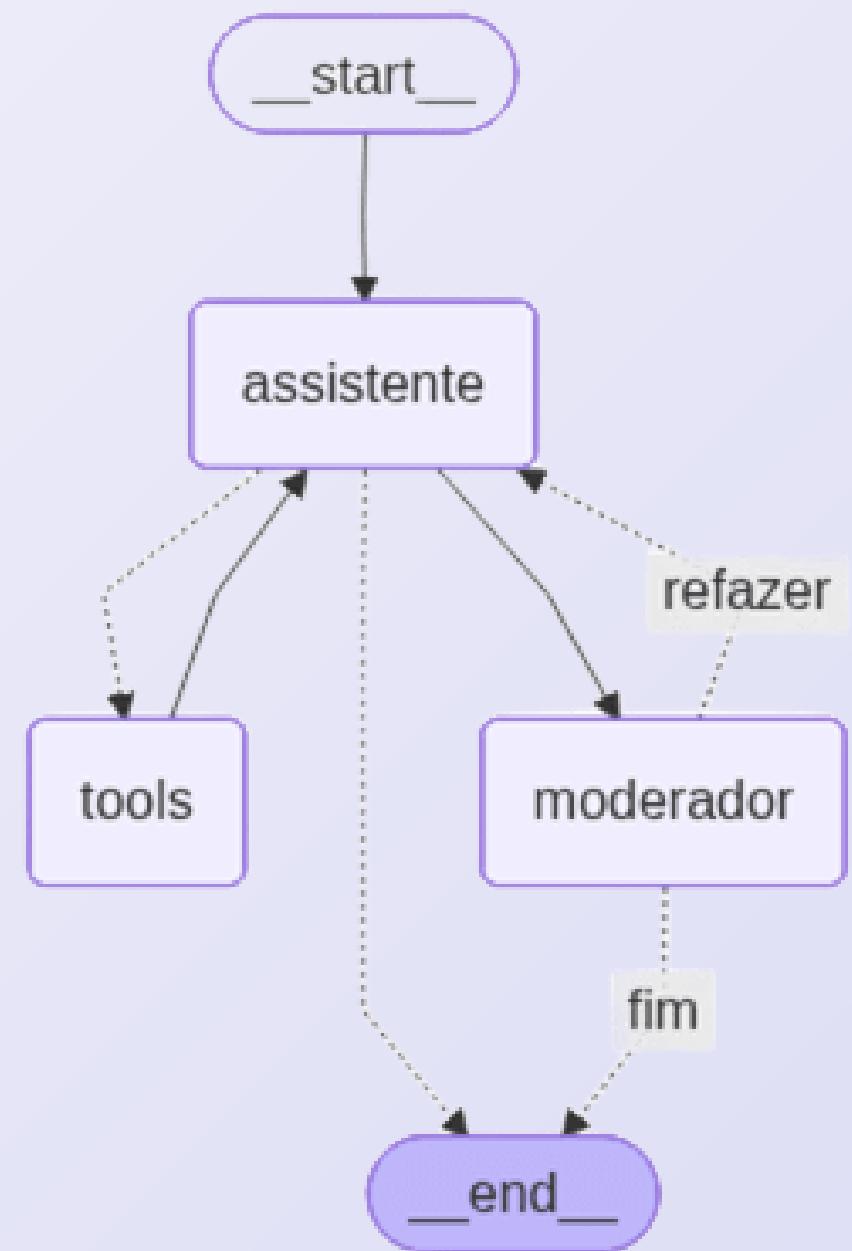
Passo 4: criação dos nós do grafo.

Nó roteador: assistente ou END

```
1 def roteador(state: State):
2     """Roteia para o agente_assistente ou finaliza."""
3     avaliacao = state.get("avaliacao")
4
5     if avaliacao == 'não':
6         return "refazer"
7
8     return 'fim'
```

Nó tools

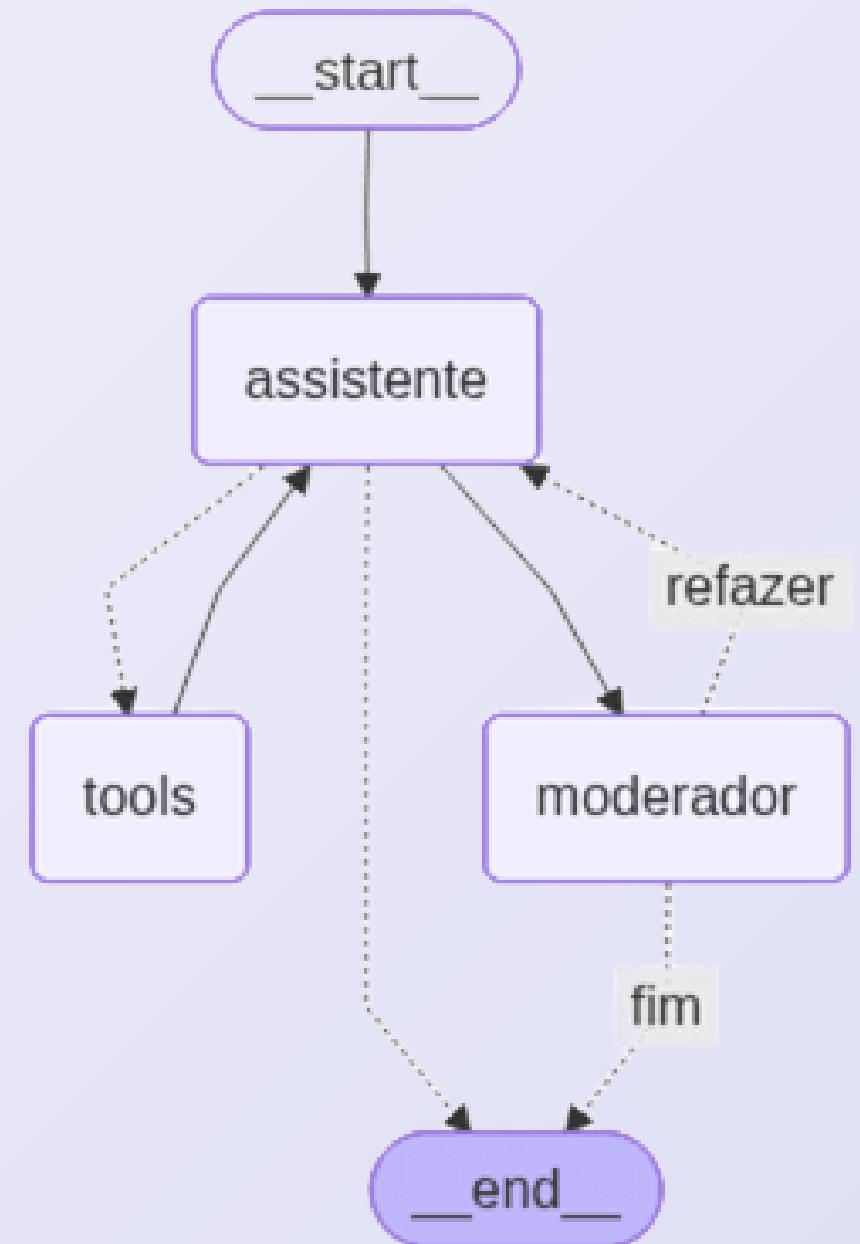
```
1 from langgraph.prebuilt import ToolNode, tools_condition
```



LangGraph

Passo 5: montagem do grafo.

```
1 # Definindo o grafo
2 graph = StateGraph(State)
3
4 # Adicionando os nós
5 tool_node = ToolNode(tools=tools)
6 graph.add_node("assistente", assistente)
7 graph.add_node("tools", tool_node)
8 graph.add_node("moderador", moderador)
9
10 # Conectando os nós
11 graph.add_edge(START, "assistente")
12 graph.add_conditional_edges("assistente", tools_condition)
13 graph.add_edge("tools", "assistente")
14 graph.add_edge("assistente", "moderador")
15 graph.add_conditional_edges(
16     "moderador",
17     roteador,
18     {
19         "refazer": "assistente",
20         "fim": END
21     }
22 )
```



LangGraph

Passo 6: compile o grafo.

```
1 # O MemorySaver mantém em memória o histórico de mensagens e variáveis intermediárias
2 # durante a execução do fluxo, permitindo que o agente continue do ponto onde parou.
3 checkpointer = MemorySaver()
4
5 # Compila o grafo de agentes (graph).
6 app = graph.compile(checkpointer=checkpointer)
```

Gere a imagem

```
1 from IPython.display import Image, display
2 display(Image(app.get_graph().draw_mermaid_png()))
```

LangGraph

Passo 6: execução.

```
1 final_state = app.invoke(
2     {"messages": [HumanMessage(content="Olá! Quais as novidades astronômicas de hoje?")], 
3      "documento": 'https://www.nasa.gov/news/recently-published/',
4      "avaliacao": "", 
5      "feedback": "" 
6    },
7     config={"configurable": {"api_key": API_KEY, "thread_id": 42}}
8 )
```

LangGraph

Saída para a pergunta: "Olá! Quais as novidades astronômicas de hoje?"

```
1 > Entering new AgentExecutor chain...
2 Thought: Para responder às novidades astronômicas de hoje, preciso buscar notícias recentes sobre astronomia.
3
4 Action: retrieve_context
5 Action Input: "novidades astronômicas hoje"
6
7 Observation: Article....
8
9 Thought: Eu tenho uma notícia astronômica recente. Agora, preciso formatar a resposta de acordo com o solicitado.
10
11 Final Answer: {"resposta": "Uma das novidades astronômicas de hoje é a descoberta de um planeta 'bebê' chamado WISPIIT
   2b, fotografado pela primeira vez em um anel ao redor de uma estrela jovem."}
12
13 > Finished chain.
14
15 # Saída do agente_moderador
16 {'avaliacao': 'sim', 'feedback': 'Você não possui feedback.'}
17
18 # Estado final do grafo
19 {'messages': [HumanMessage(content='Olá! Quais as novidades astronômicas de hoje?', additional_kwargs={},
   response_metadata={}), AIMessage(content="Uma das novidades astronômicas de hoje é a descoberta de um planeta 'bebê'
   chamado WISPIIT 2b, fotografado pela primeira vez em um anel ao redor de uma estrela jovem.", additional_kwargs={},
   response_metadata={})],
20 'documento': 'https://www.nasa.gov/news/recently-published/',
21 'avaliacao': 'sim',
22 'feedback': 'Você não possui feedback.'}
23 }
```

LangGraph

Saída para a pergunta: "Olá! Qual a distância da Terra e Marte?"

```
1 > Entering new AgentExecutor chain...
2 Thought: Para responder à pergunta sobre a distância da Terra a Marte, posso usar a função `distancia_terra_planeta` passando 'Marte' como argumento.
3
4 Action: distancia_terra_planeta
5 Action Input: Marte
6 Observation: A distância média da Terra até Marte é de aproximadamente 78.3 milhões de km.
7 Thought: Agora que tenho a distância média da Terra a Marte, posso formular a resposta no formato JSON solicitado.
8
9 Final Answer: {"resposta": "A distância média da Terra a Marte é de aproximadamente 78.3 milhões de km."}
10
11 > Finished chain.
12
13 # Saída do agente_moderador
14 {'avaliacao': 'sim', 'feedback': 'Você não possui feedback.'}
15
16 # Estado final do grafo
17 {'messages': [HumanMessage(content='Olá! Qual a distância da Terra e Marte?', additional_kwargs={}, response_metadata={}), AIMessage(content='A distância média da Terra a Marte é de aproximadamente 78.3 milhões de km.', additional_kwargs={}, response_metadata={})],
18 'documento': 'https://www.nasa.gov/news/recently-published/',
19 'avaliacao': 'sim',
20 'feedback': 'Você não possui feedback.'}
```

LangGraph

Saída para a pergunta: "Olá! Quem é a Xuxa?"

Tentativa 1

```
1 > Entering new AgentExecutor chain...
2 Thought: A pergunta não está relacionada à astronomia, mas sim a uma figura pública brasileira. A dica fornecida não oferece informações relevantes para responder à pergunta. No entanto, posso fornecer uma resposta geral.
3 Como não há uma relação direta com astronomia, não preciso usar as ferramentas disponíveis (`retrieve_context` ou `distancia_terra_planeta`). Posso responder diretamente.
4
5 Thought: I now know the final answer
6 Final Answer: {"resposta": "Xuxa é uma cantora, apresentadora de televisão e atriz brasileira."}
7
8 > Finished chain.
9
10 # Saída do agente_moderador
11 {'avaliacao': 'não', 'feedback': 'A resposta não está relacionada ao tema astronomia.'}
12
13 # Estado final do grafo
14 {'messages': [HumanMessage(content='Olá! Quais as novidades astronômicas de hoje?', additional_kwargs={}, response_metadata={}), AIMessage(content="Uma das novidades astronômicas de hoje é a descoberta de um planeta 'bebê' chamado WISPIT 2b, fotografado pela primeira vez em um anel ao redor de uma estrela jovem.", additional_kwargs={}, response_metadata={})],
15 'documento': 'https://www.nasa.gov/news/recently-published/',
16 'avaliacao': 'sim',
17 'feedback': 'Você não possui feedback.'}
18 }
```

LangGraph

```
1 > Entering new AgentExecutor chain...
2 Thought: A pergunta "Quem é a Xuxa?" não está diretamente relacionada à astronomia, mas considerando a dica de que a resposta deve estar relacionada ao tema astronomia, posso pensar em uma possível conexão. Uma possibilidade é que "Xuxa" esteja relacionada a algum termo astronômico ou objeto no céu. Vou começar pesquisando se há alguma conexão entre "Xuxa" e astronomia.
3
4 Action: retrieve_context
5 Action Input: "Xuxa astronomia"
6
7 Observation: Article...
8
9 Thought: Parece que a pesquisa não retornou resultados relevantes sobre "Xuxa" em astronomia. Vou tentar uma abordagem diferente, pensando se há algum objeto astronômico ou termo que possa estar relacionado a "Xuxa". Uma possibilidade é que "Xuxa" seja um apelido ou nome dado a algum objeto astronômico.
10
11 Action: retrieve_context
12 Action Input: "Xuxa objeto astronômico"
13
14 Observation: Article...
15
16 Thought: A pesquisa continua a não retornar resultados relevantes sobre "Xuxa" em astronomia. Considerando que "Xuxa" é uma personalidade brasileira famosa, é provável que a pergunta esteja procurando uma resposta mais criativa ou indireta. Vou tentar pensar em uma resposta que conecte "Xuxa" à astronomia de alguma forma, mesmo que não seja direta.
17
18 Uma possível conexão é que "Xuxa" é uma celebridade brasileira e, como tal, poderia ser comparada a uma "estrela" no contexto de sua fama. Isso estabelece uma conexão entre "Xuxa" e o tema astronomia, considerando que as estrelas são objetos astronômicos.
19
20 Thought: I now know the final answer
21
22 Final Answer: {"resposta": "Xuxa é uma celebridade brasileira conhecida como a 'Rainha dos Baixinhos', mas no contexto da astronomia, podemos dizer que ela brilha como uma estrela no mundo do entretenimento."}
23
24 > Finished chain.
25
26 # Saída do agente_moderador
27 {'avaliacao': 'não', 'feedback': 'A resposta não está adequada pois não se refere ao tema astronomia. A pergunta não tem relação com o tema e a resposta tentou fazer uma analogia forçada.'}
28
29
```

Tentativa 2

LangGraph

Tentativa 3

```
1 > Entering new AgentExecutor chain...
2 Thought: A pergunta "Quem é a Xuxa?" não está relacionada ao tema de astronomia. Devo responder de acordo com o formato solicitado, indicando que a pergunta não é relevante para o tema.
3
4 Action: None (não há ação necessária, pois não posso usar as ferramentas fornecidas para responder à pergunta)
5
6 Thought: I now know the final answer
7
8 Final Answer: {"resposta": "A pergunta não está relacionada ao tema de astronomia."}
9
10 > Finished chain.
11
12 # Saída do agente_moderador
13 {'avaliacao': 'sim', 'feedback': 'Você não possui feedback.'}
14
15 # Estado final do grafo
16 {'messages': [HumanMessage(content='Olá! Quem é a Xuxa?', additional_kwargs={}, response_metadata={}),
   AIMessage(content='Xuxa é uma cantora, apresentadora de televisão e atriz brasileira.', additional_kwargs={},
             response_metadata={}), AIMessage(content='Xuxa é uma cantora, apresentadora de televisão e atriz brasileira.',
                                                 additional_kwargs={}, response_metadata={}), AIMessage(content='Xuxa é uma cantora, apresentadora de televisão e atriz brasileira.',
                                                 additional_kwargs={}, response_metadata={}), AIMessage(content="Xuxa é uma celebridade brasileira conhecida como a 'Rainha dos Baixinhos', mas no contexto da astronomia, podemos dizer que ela brilha como uma estrela no mundo do entretenimento.", additional_kwargs={}, response_metadata={}), AIMessage(content='A pergunta não está relacionada ao tema de astronomia.', additional_kwargs={}, response_metadata={})],
17 'documento': 'https://www.nasa.gov/news/recently-published/',
18 'avaliacao': 'sim',
19 'feedback': 'Você não possui feedback.'}
```

Referências

LANGCHAIN. **LangChain Documentation.** Disponível em:
<https://python.langchain.com/docs/introduction/>.

LANGCHAIN. **LangGraph.** Disponível em: <https://langchain-ai.github.io/langgraph/>.

LANGCHAIN. **LangSmith: Get Started.** Disponível em: <https://docs.langchain.com/langsmith/home>



Obrigada! =)

