

DOMINE LLMs COM LANGCHAIN



CONTEÚDO DO CURSO

- Intuição sobre LLMs
- LLM com Hugging Face
- LLM com Langchain
- Aplicações RAG
- Agentes e tools
- Projeto 1: transcrição e compreensão de vídeo
- Projeto 2: chatbot com memória e interface
- Projeto 3: converse com seus documentos

- Pré-requisitos: lógica de programação e programação básica em Python

O QUE SÃO LLMs

- **LLMs (Large Language Models)** - “grandes modelos de linguagem” ou “Modelos de linguagem de grande escala” - são modelos de Inteligência Artificial (IA) que utilizam técnicas de Aprendizado de Máquina para entender e gerar linguagem humana
- Utilizam Processamento de Linguagem Natural (**PLN**) para interagir com a linguagem humana.

O QUE SÃO LLMs

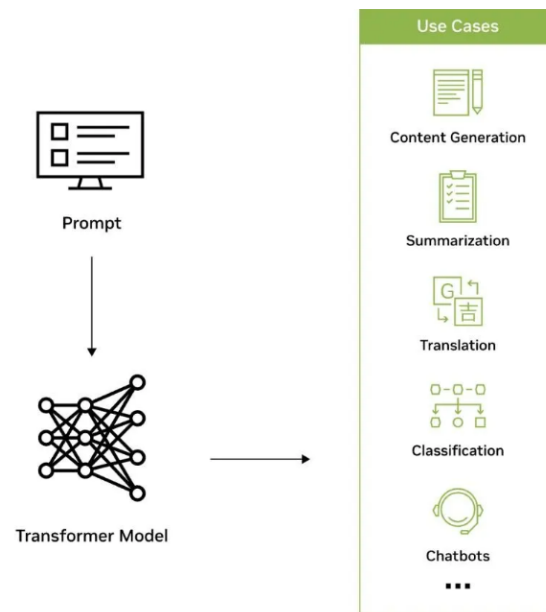
- É importante esclarecer e saber a diferença de todos esses conceitos:
 - **Inteligência Artificial (IA)**
 - **Machine Learning (ML)**
 - **Processamento de Linguagem Natural (PLN)**
 - **Redes neurais artificiais (ANNs)**
 - **IA Generativa**
 - **LLMs**

COMO AS LLMs FUNCIONAM

- Utilizam aprendizado não-supervisionado para entender a linguagem.
- Um modelo de machine learning é alimentado com conjuntos de dados (centenas de bilhões de palavras e frases) os quais são estudados e aprendidos com base em exemplo.
- O modelo aprende com esses exemplos sem instruções humanas explícitas.
- Extrai informações dos dados, cria conexões e "aprende" sobre a linguagem.
- Como resultado, o modelo captura as relações complexas entre palavras e frases.

COMO AS LLMs FUNCIONAM

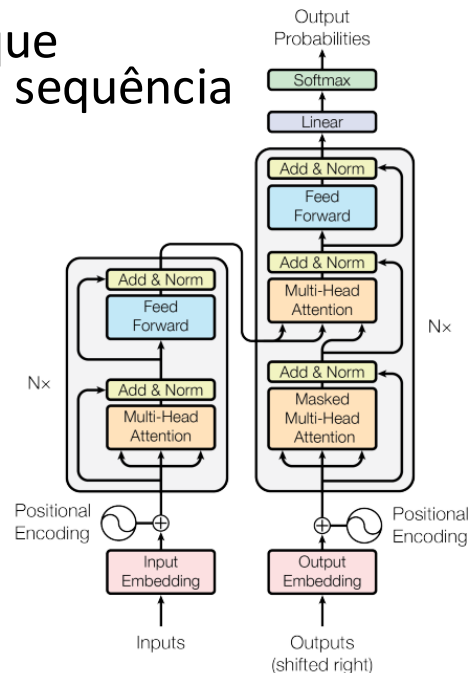
- LLMs requerem recursos computacionais significativos para processar dados.
- Utilizam unidades de processamento gráfico (GPUs) para lidar com cálculos complexos.
- O uso de GPUs é fundamental, o que acelera o treinamento e a operação dos **transformadores**



Créditos da imagem : [NVIDIA](#)

TRANSFORMADORES (TRANSFORMERS)

- Um transformador é uma arquitetura de deep learning que transforma ou altera uma sequência de entrada em uma sequência de saída.
- Foi apresentada no paper "Attention Is All You Need" ("*Atenção é tudo que você precisa*"), de autoria de oito cientistas do Google.
- A arquitetura do transformador melhora a capacidade dos modelos de ML ao capturar dependências contextuais em sequências de dados, como palavras em uma frase.
- Com bilhões de parâmetros, o transformador analisa padrões e nuances complexas da linguagem.



Arquitetura do Transformer.

Mais detalhes no paper oficial:

<https://arxiv.org/abs/1706.03762>

TRANSFORMADORES (TRANSFORMERS)

- Por exemplo, em "Qual é a cor do céu?", o modelo identifica a relação entre "cor", "céu" e "azul" para gerar "O céu é azul".
- É uma rede neural que aprende o contexto e, assim, o significado com o monitoramento de relações em dados sequenciais como as palavras desta frase.



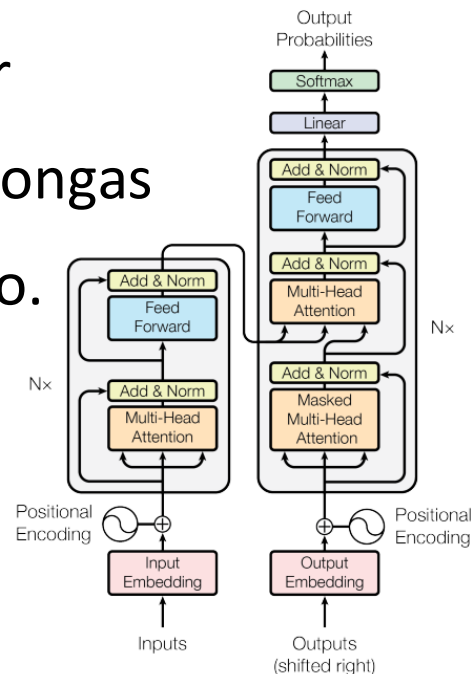
Guia ilustrado dos transformers: <https://jalammar.github.io/illustrated-transformer/>

ARQUITETURAS ANTERIORES

- **RNN:** Tinham muitas limitações para processar sequências muito longas. Embora fossem capazes de capturar dependências temporais, seu desempenho decaía com o aumento da sequência.
- **LSTM:** Consegue processar sequências maiores graças a mecanismos inovadores, que permite a preservação de informações importantes por períodos mais longos. No entanto, ainda enfrentam dificuldades de escalabilidade e eficiência em treinamento com grandes volumes de dados.
- **Transformers:** Superaram essas limitações ao eliminar a dependência sequencial e utilizar mecanismos de atenção que capturam relações contextuais entre todas as partes de uma sequência simultaneamente.

COMO FUNCIONA A ARQUITETURA DO TRANSFORMER

- Redes tradicionais: codificador e decodificador
- Desafios: perdem-se detalhes em sequências longas
- Solução proposta: Mecanismo de Auto atenção.
- Isso torna os transformadores mais eficientes e eficazes, especialmente com textos longos e grandes conjuntos de dados.



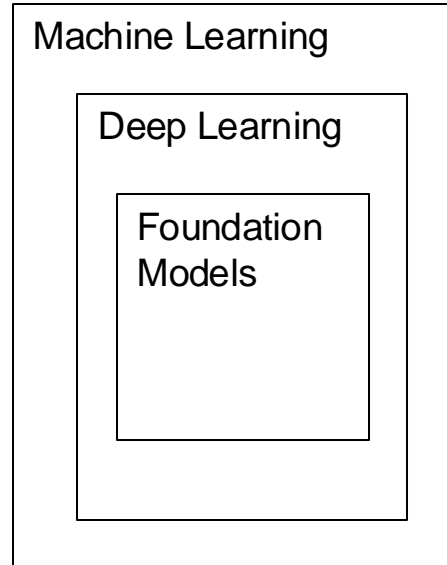
Arquitetura do Transformer.

Mais detalhes no paper oficial:

<https://arxiv.org/abs/1706.03762>

MODELOS DE FUNDAÇÃO

- Os LLMs são uma classe dos chamados *Foundation Models* – Modelos de fundação (modelos de base, ou Modelos Fundacionais - em tradução livre).
- O termo Foundation Models foi popularizado pela Universidade de Stanford em 2021, que também fundou o Centro de Pesquisa em Foundation Models (CRFM) para estudar modelos como BERT, GPT-3 e CLIP.
- Enquanto LLMs são modelos específicos para linguagem e portanto focam em tarefas relacionadas a texto, os Foundation Models têm um escopo mais amplo, podendo ser aplicados a diversas finalidades; como aplicações envolvendo imagens, áudios, vídeos, sinais 3D e muito mais.



FUNCIONAMENTO DE LLMs

- Visão geral: Entender seus componentes principais, como os tokens (entradas) e logits (saídas).
- Encoder-Decoder e GPT: Revisite a arquitetura Transformer de encoder-decoder e, especificamente, a arquitetura GPT, que é apenas decodificadora e utilizada em muitos LLMs atuais.
- Tokenização: Converta dados de texto brutos em tokens que o modelo possa entender, geralmente dividindo o texto em palavras ou subpalavras.
- Mecanismos de Atenção: Compreenda a teoria por trás dos mecanismos de atenção, como auto-atenção e atenção de produto escalar, que permitem ao modelo focar em diferentes partes da entrada ao produzir uma saída.
- Geração de texto: Explore as diferentes formas de gerar sequências de saída. Estratégias comuns incluem decodificação gananciosa, beam search, top-k sampling e nucleus sampling.

FUNCIONAMENTO DE LLMs – MATEMÁTICA

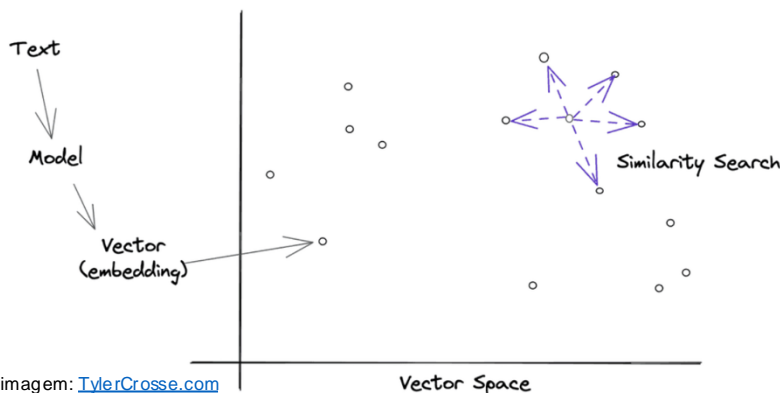
- Álgebra linear: Essencial para compreender muitos algoritmos, especialmente em deep learning. Conceitos-chave incluem vetores, matrizes, determinantes, autovalores e autovetores, espaços vetoriais e transformações lineares.
- Cálculo: Necessário para a otimização de funções contínuas em muitos algoritmos de aprendizado de máquina. Conceitos importantes incluem derivadas, integrais, limites e séries, além de cálculo multivariável e gradientes.
- Probabilidade e estatística: Crucial para entender como os modelos aprendem com os dados e fazem previsões. Conceitos-chave incluem teoria da probabilidade, variáveis aleatórias, distribuições de probabilidade, expectativas, variância, covariância, correlação, testes de hipóteses, intervalos de confiança, estimativa de máxima verossimilhança e inferência bayesiana.

FUNCIONAMENTO DE LLMs

- [The Illustrated Transformer](#) por Jay Alammar: uma explicação visual e intuitiva do modelo Transformer.
- [The Illustrated GPT-2](#) por Jay Alammar: Focado na arquitetura GPT, que é muito semelhante à do Llama.
- [Visual intro to Transformers](#) por 3Blue1Brown: introdução visual simples e fácil de entender de Transformers
- [LLM Visualization](#) por Brendan Bycroft: visualização 3D incrível do que acontece dentro de um LLM.
- [nanoGPT](#) por Andrej Karpathy: um vídeo de 2 horas no YouTube para reimplementar o GPT do zero (para programadores).
- [Attention? Attention!](#) por Lilian Weng: Apresenta a importância e necessidade do conceito de atenção de uma forma detalhada e mais formal.
- [Decoding Strategies in LLMs](#): Fornece código e uma introdução visual às diferentes estratégias de decodificação para gerar texto.

EMBEDDINGS

- **Embeddings** são representações numéricas de dados textuais.
- No contexto de LLMs, os embeddings são usados para transformar palavras ou sentenças em vetores que o modelo pode entender e processar.
- Para texto, grandes modelos de linguagem como BERT e ChatGPT são ótimos para gerar incorporações que capturam o significado semântico do texto.

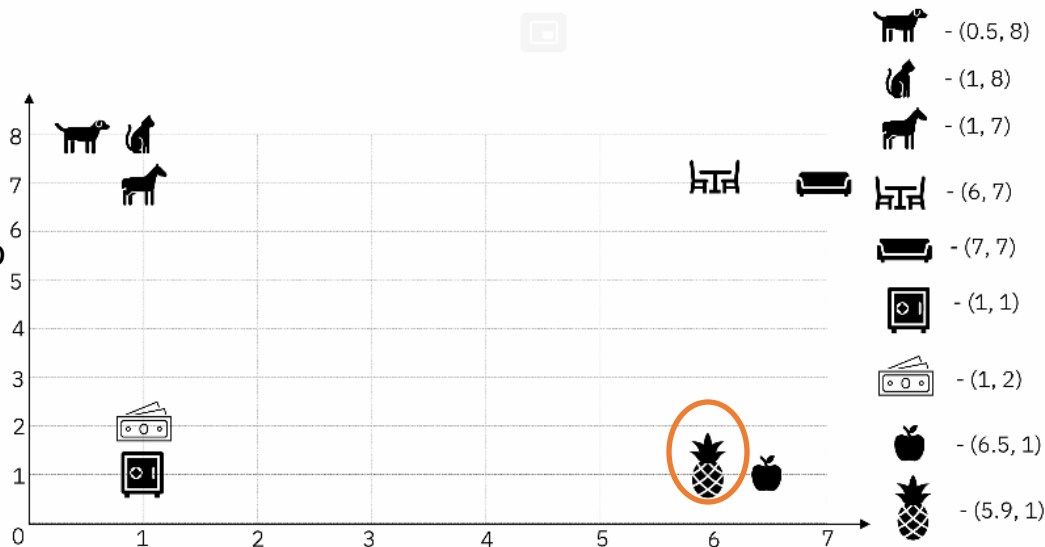


Créditos da imagem: [TylerCrosse.com](https://tylertcrosse.com)

EMBEDDINGS

- O modelo aprende a separar e agrupar esses textos com base em suas similaridades.
- Quando esse modelo recebe uma palavra nova, como "abacaxi", ele sabe exatamente onde colocar - muito provavelmente na região onde estão outras frutas.

Note que palavras (nesse caso, vetores) similares, tendem a se agrupar no espaço vetorial. São pontos com coordenadas próximas.

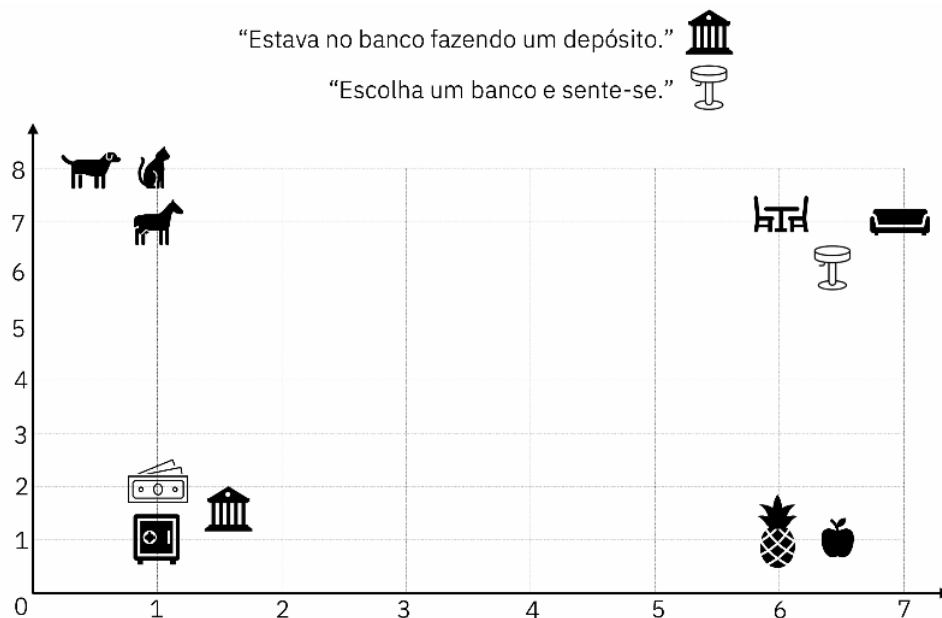


Créditos da imagem: brains.dev

EMBEDDINGS

- E para a palavra "banco"?
- Aqui entra a importância do Mecanismo de Atenção: permite ao modelo entender esse contexto.
- Por exemplo, em "Estava no banco fazendo um depósito." e "Escolha um banco e sente-se.", "banco" pode se referir a uma instituição financeira ou a um assento.

Com o contexto, o modelo posiciona corretamente os embeddings, diferenciando entre os significados.



Créditos da imagem: brains.dev

EMBEDDINGS E TOKENS

- Tokens são unidades menores de texto criadas através do processo de tokenização, que converte texto em números. No contexto de LLMs, pode ser uma palavra inteira, parte de uma palavra ou até mesmo um caractere individual.
- Enquanto o Embedding é a representação vetorial do texto em um espaço multidimensional, o Token é uma representação mais estática de unidades de texto.
- Tokens são convertidos em números para que o modelo possa entender e gerar texto.
- Esse mapeamento é feito usando um vocabulário. Cada token tem um índice específico.

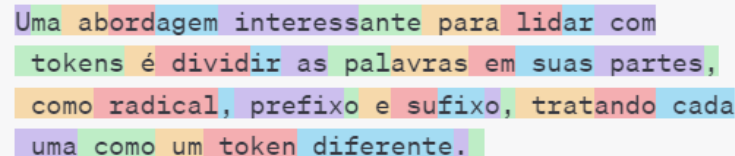
LETRA	Índice	LETRA	Índice	LETRA	Índice
A	1	I	9	Q	17
B	2	J	10	R	18
C	3	K	11	S	19
D	4	L	12	T	20
E	5	M	13	U	21
F	6	N	14	V	22
G	7	O	15	X	23
H	8	P	16	Z	24

Neste exemplo, a letra “i” vira o número 9. A letra “a” vira o número 1. Letra “e” vira o número 5, “x” vira 23... e assim por diante.

TOKENS

- Uma abordagem interessante para lidar com tokens é dividir as palavras em suas partes, como radical, prefixo e sufixo, tratando cada uma como um token diferente.
- Muitas palavras no nosso idioma compartilham o mesmo radical. Da mesma forma, sufixos em algumas palavras podem ser reutilizados em outros.
- Para explorar mais o processo de Tokenização, utilize o Tokenizer da Open AI
- <https://platform.openai.com/tokenizer>

Tokens	Characters
42	163



Uma abordagem interessante para lidar com tokens é dividir as palavras em suas partes, como radical, prefixo e sufixo, tratando cada uma como um token diferente.

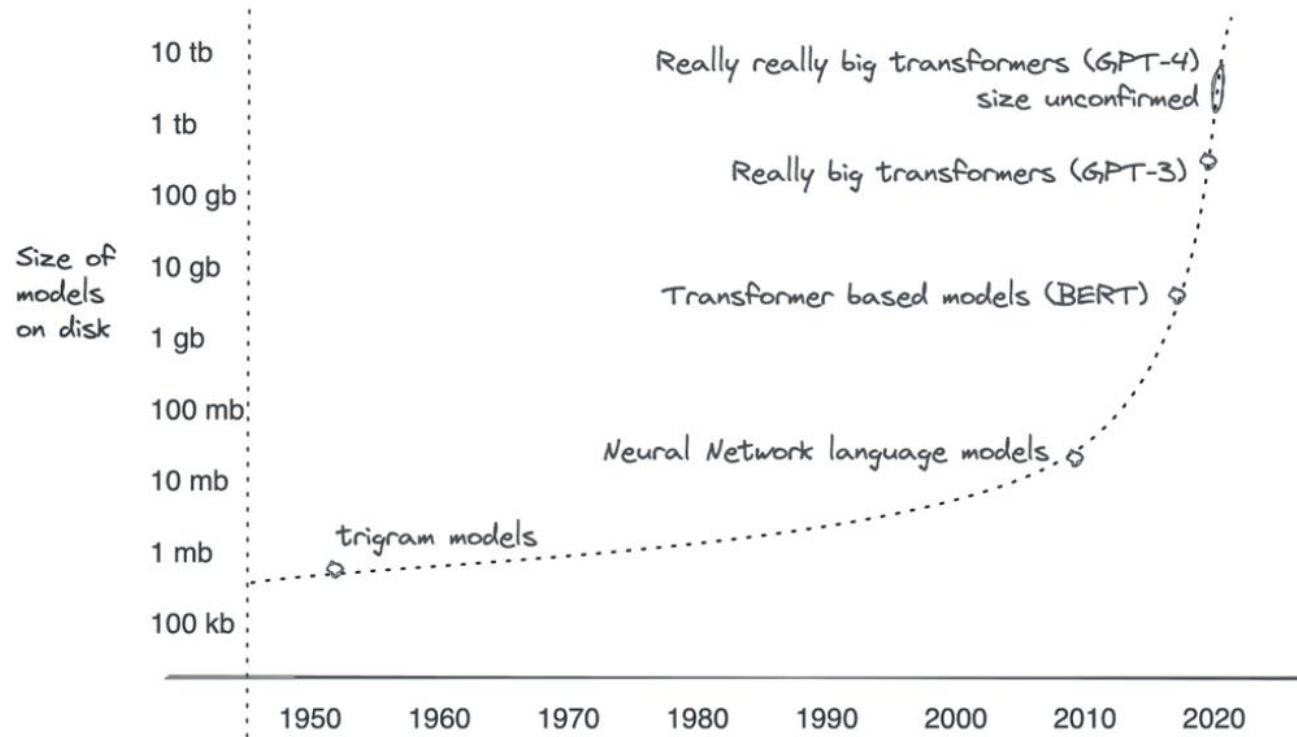
EVOLUÇÃO E CONTEXTO HISTÓRICO

- As primeiras tentativas de modelos de linguagem datam do século passado, quando pesquisadores começaram a explorar a ideia de ensinar máquinas a entenderem e gerarem linguagem humana.
- Na década de 90 as linguagens evoluíram para modelos estatísticos, analisando padrões linguísticos. Projetos em larga escala eram limitados pela capacidade de processamento.
- Os avanços em Machine Learning nos anos 2000 aumentaram a complexidade dos modelos de linguagem. A internet forneceu uma grande quantidade de dados de treinamento.
- Em 2012, avanços em deep learning e conjuntos de dados maiores levaram ao desenvolvimento dos transformadores pré-treinados generativos (GPT).

EVOLUÇÃO E CONTEXTO HISTÓRICO

- A jornada dos modelos de LLMs como conhecemos hoje começou com os modelos de fundação como o GPT e o BERT, que estabeleceram a base para sistemas mais sofisticados.
- Em 2018 o Google apresentou o BERT, um grande avanço na arquitetura de modelos de linguagem. Dois anos depois, a OpenAI lançou o GPT-3, com 175 bilhões de parâmetros, estabelecendo um novo padrão de desempenho.
- Com o lançamento do ChatGPT em 2022, modelos como o GPT-3 se tornaram amplamente acessíveis via interface web, aumentando a conscientização pública sobre LLMs.
- Em 2023, modelos de código aberto ganharam atenção por demonstrarem resultados impressionantes, como Dolly 2.0, LLaMA, Alpaca e Vicuna. O GPT-4 é lançado, definindo novos padrões de tamanho e desempenho. Em 2024, modelos open source demonstraram superar modelos proprietários em diversas tarefas, competindo diretamente com os modelos mais modernos como ChatGPT e Gemini.

LINHA DO TEMPO DAS LLMs



Créditos da imagem: [TylerCrosse.com](https://tylercrosse.com)

POR QUE LLMs SÃO TÃO IMPORTANTES HOJE?

Automação e eficiência

LLMs realizam tarefas relacionadas à linguagem, desde suporte ao cliente até análise de dados e geração de conteúdo. Além disso, reduzem custos operacionais e liberam recursos humanos para tarefas mais estratégicas.

Geração de insights

Verificam rapidamente grandes volumes de texto, como redes sociais, avaliações e artigos. Podem ajudar a estudar tendências do mercado e analisar o feedback dos clientes, orientando decisões empresariais.

Aperfeiçoamento da experiência do cliente

Oferecem conteúdo altamente personalizado, aumentando o engajamento e melhorando a experiência dos usuários. Ex: chatbots para atendimento ao cliente 24/7, mensagens de marketing personalizadas, traduções e comunicação entre distintas culturas.

EXEMPLOS DE APLICAÇÕES COM LLMS

- Chatbots e assistentes virtuais
- Geração de código e depuração
- Geração de conteúdo
- Resumo de conteúdo
- Tradução entre idiomas
- Classificação e agrupamento de texto
- Análise de sentimento

DESAFIOS E LIMITAÇÕES DOS LLMs

- Custo
- Privacidade e segurança
- Precisão e viés

ALUCINAÇÕES EM LLMs

- Alucinações em LLMs ocorrem quando o modelo gera informações incorretas ou fictícias que parecem plausíveis.
- Isso acontece porque os LLMs são projetados para prever a próxima palavra ou frase com base em padrões aprendidos, mas não possuem compreensão real do mundo.
- Podem combinar palavras de maneira coerente, mas sem garantir a veracidade dos dados.
- Esses erros podem ser críticos em aplicações que exigem precisão factual, como atendimento ao cliente ou relatórios médicos.
- Portanto, é importante validar as informações fornecidas por LLMs antes de utilizá-las como referência.

ALUCINAÇÕES EM LLMs

As alucinações podem ser causadas por:

- Overfitting, erros de codificação e decodificação, e viés de treinamento (semelhante à falta de diversidade nos dados de treinamento, o que irá limitar a capacidade de generalizar para novos dados), entre outros.

Pesquisadores e profissionais estão adotando várias abordagens para abordar o problema das alucinações em LLMs.

- Isso inclui melhorar a diversidade de dados de treinamento, eliminar vieses inerentes, usar melhores técnicas de regularização e empregar treinamento adversário e aprendizado por reforço

ÉTICA E PREOCUPAÇÕES COM LLMs

- Os LLMs podem amplificar preconceitos prejudiciais e gerar conteúdo discriminatório, baseando-se em estereótipos presentes nos dados de treinamento.
- A produção de conteúdo discriminatório e tóxico é uma preocupação central, pois os dados de treinamento muitas vezes incluem estereótipos socioculturais. Isso pode levar à geração de textos prejudiciais contra grupos desfavorecidos, com base em raça, gênero, religião, etnia, etc.
- Questões de privacidade são críticas, pois os LLMs treinam com grandes volumes de dados pessoais, potencialmente vazando informações sensíveis como números de previdência e endereços.
- LLMs permitem facilmente gerar desinformação, produzindo conteúdo falso que parece confiável, o que pode ter impactos negativos nas esferas social, cultural e política.

MODELOS

- Os modelos de LLMs podem ser **proprietários (Closed Source)** ou de **código aberto (Open Source)**.
- Modelos proprietários são criados por empresas privadas e têm restrições de uso, enquanto modelos de código aberto são gratuitos e modificáveis, incentivando a colaboração e inovação. Modelos proprietários podem possuir trilhões de parâmetros e capturam nuances linguísticas profundas, mas mais parâmetros nem sempre garantem melhor desempenho.
- Modelos de código aberto, embora geralmente possam ser menores em termos de parâmetro, têm mostrado capacidade de competir com os proprietários, principalmente quando bem ajustados e otimizados pela comunidade. Além disso, a diferença de tamanho máximo entre modelos proprietários e open source está cada vez menor.

MODELOS – ESCOLHENDO ENTRE CÓDIGO ABERTO OU PROPRIETÁRIO

Vantagens de modelos Open Source

- **Transparência:** Permitem total visibilidade sobre arquitetura, dados de treino e processos, o que é crucial para conformidade e mitigação de riscos.
- **Personalização e flexibilidade:** Com acesso ao código-fonte, é possível ajustar os modelos para atender necessidades específicas.
- **Custos:** Geralmente mais econômicos, sem taxas de licenciamento, beneficiando startups e pequenas empresas.
- **Comunidade e suporte:** Aproveitam a contribuição de uma comunidade global de desenvolvedores para melhorias contínuas.

Vantagens de modelos Proprietários

- **Suporte dedicado:** Oferecem suporte técnico especializado, essencial para empresas que dependem dos modelos.
- **Desempenho otimizado:** Frequentemente possuem ajustes específicos que podem superar os modelos de código aberto em certas tarefas.
- **Segurança:** Podem fornecer garantias adicionais de segurança, importantes para empresas com dados sensíveis.

QUANDO USAR MODELOS OPEN SOURCE

- Controle total sobre os dados
- Permitem personalização do código-fonte, ajustando modelos às necessidades específicas e facilitando a otimização contínua.
- Possuem custo benefício superior pois são geralmente gratuitas, o que reduz custos e beneficia startups e pequenas empresas.
- Embora modelos proprietários ofereçam suporte técnico e otimizações, soluções open source proporcionam maior transparência e controle, com inovação impulsionada pela comunidade global.

MODELOS MENORES - SMALL LANGUAGE MODELS (SLM)

Modelos de linguagem pequenos ganharam muita popularidade por usarem muito menos recursos computacionais, o que democratiza o uso em ambiente local.

- SLMs funcionam quase tão bem quanto modelos maiores, mas são mais rápidos e práticos para dispositivos móveis e sistemas em tempo real.
- "Pequeno" refere-se ao menor número de parâmetros, variando de alguns milhões a algumas centenas de milhões, comparado a bilhões em modelos maiores.
- Alguns especialistas preferem ainda manter o rótulo de SLM para modelos bem menores, com menos de 1 Bilhão de parâmetros: por exemplo o GPT-2 Small (117 milhões), TinyBERT (15 milhões)
- Ou ainda, algo entre 1~3 bilhões de parâmetros, como por exemplo o Google Gemma 2B que possui 2 bilhões.

Enquanto os LLMs, como o GPT-3, possuem bilhões ou até trilhões de parâmetros, os SLMs têm uma quantidade significativamente menor: na faixa de milhões a poucos bilhões de parâmetros.

	Llama 3	Meta 8 billion parameters
	Phi-3	Microsoft 3.8 billion - 7 billion parameters
	Gemma	Google 2 billion - 7 billion parameters
	Mixtral 8x7B	Mistral AI 7 billion parameters
	OpenELM	Apple 0.27 billion - 3 billion parameters

Créditos da imagem: [Data Science Dojo](https://data-science-dojo.com/)

MODELOS MENORES - SMALL LANGUAGE MODELS (SLM)

- A desvantagem é que podem não ter o mesmo desempenho em tarefas complexas, como entender contexto ou gerar texto detalhado.
- Demonstram um excelente custo-benefício considerando a quanto foi reduzido em tamanho.

Comparação com a versão maior (70B) com a menor (8B) do Llama 3

	Meta Llama 3 8B	Mistral 7B		Gemma 7B	
		Published	Measured	Published	Measured
MMLU 5-shot	66.6	62.5	63.9	64.3	64.4
AGIEval English 3-5-shot	45.9	--	44.0	41.7	44.9
BIG-Bench Hard 3-shot, CoT	61.1	--	56.0	55.1	59.0
ARC-Challenge 25-shot	78.6	78.1	78.7	53.2 0-shot	79.1
DROP 3-shot, F1	58.4	--	54.4	--	56.3

	Meta Llama 3 70B	Gemini Pro 1.0	Mixtral 8x22B
		Published	Measured
MMLU 5-shot	79.5	71.8	77.7
AGIEval English 3-5-shot	63.0	--	61.2
BIG-Bench Hard 3-shot, CoT	81.3	75.0	79.2
ARC-Challenge 25-shot	93.0	--	90.7
DROP 3-shot, F1	79.7	74.1 variable-shot	77.6

Desempenho do modelo pré-treinado Llama 3 – Fonte: Meta

MODELOS MENORES - SMALL LANGUAGE MODELS (SLM)

O modelo open source Phi-3-mini é considerado revolucionário para a categoria de SLM.

Foi lançado pela Microsoft no segundo semestre de 2024 e é baseado em uma arquitetura decoder-only do transformer, com comprimento de contexto de 4.000 tokens.

	Phi-3-mini 3.8b	Phi-3-small 7b	Phi-3-medium 14b	Phi-2 2.7b	Mistral 7b	Gemma 7b	Llama-3-In 8b	Mixtral 8x7b	GPT-3.5 version 1106
MMLU (5-Shot) [HBK*21]	68.8	75.7	78.0	56.3	61.7	63.6	66.5	70.5	71.4
HellaSwag (5-Shot) [ZHB*19]	76.7	77.0	82.4	53.6	58.5	49.8	71.1	70.4	78.8
ANLI (7-Shot) [NWD*20]	52.8	58.1	55.8	42.5	47.1	48.7	57.3	55.2	58.1
GSM-8K (8-Shot; CoT) [CKB*21]	82.5	89.6	91.0	61.1	46.4	59.8	77.4	64.7	78.1
MedQA (2-Shot) [JFO*20]	53.8	65.4	69.9	40.9	50.0	49.6	60.5	62.2	63.4
AGIEval (0-Shot) [ZCG*23]	37.5	45.1	50.2	29.8	35.1	42.1	42.0	45.2	48.4
TriviaQA (5-Shot) [JCWZ17]	64.0	58.1	73.9	45.2	75.2	72.3	67.7	82.2	85.8
Arc-C (10-Shot) [CCE*18]	84.9	90.7	91.6	75.9	78.6	78.3	82.8	87.3	87.4
Arc-E (10-Shot) [CCE*18]	94.6	97.0	97.7	88.5	90.6	91.4	93.4	95.6	96.3
PIQA (5-Shot) [BZGC19]	84.2	86.9	87.9	60.2	77.7	78.1	75.7	86.0	86.6
SociQA (5-Shot) [BZGC19]	76.6	79.2	80.2	68.3	74.6	65.5	73.9	75.9	68.3

Mais informações no paper: <https://arxiv.org/abs/2404.14219>

MODELOS MENORES - SMALL LANGUAGE MODELS (SLM)

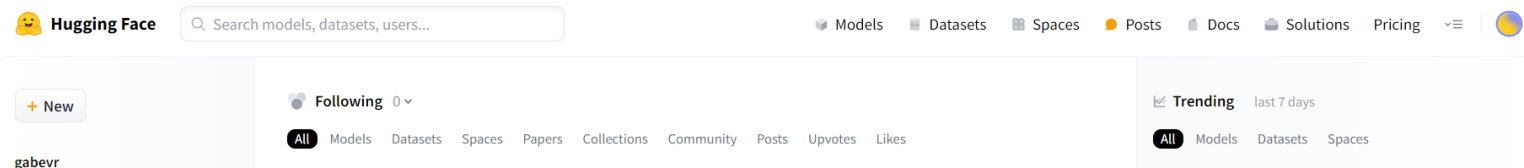
- **O que torna os SLM tão interessantes?**
 - Acessíveis e econômicos: Podem ser executados (em modo de inferência) em dispositivos com recursos limitados, como laptops ou GPUs menos potentes.
 - Mais fáceis de personalizar: Modelos pequenos podem ser ajustados (fine-tuned) mais rapidamente e em apenas uma única GPU
 - Mais eficientes em energia: Modelos de linguagem pequenos exigem menos recursos computacionais, tornando-os mais eficientes em termos de energia
 - Valiosos para propósitos educacionais: São mais gerenciáveis e, portanto, mais fáceis de entender e ajustar.

COMO ESCOLHER OS MODELOS

- Para escolher os modelos mais modernos (estado-da-arte) de LLMs, é crucial estar constantemente atualizado com os lançamentos mais recentes em repositórios de tecnologia, pois novos modelos surgem frequentemente.
- Dada a rápida evolução e o crescente interesse na área, os modelos apresentados aqui podem rapidamente se tornar “desatualizados” de certo modo, já que novos avanços são feitos a cada trimestre e novos modelos são publicados a todo momento.
- Portanto, acompanhar as inovações é essencial para garantir a relevância e a eficácia dos modelos utilizados.
- Neste curso, abordaremos uma implementação que simplifica a troca de LLMs, permitindo testar diferentes modelos sem a necessidade de reescrever todo o código, o que facilita bastante a experimentação e consequentemente levará a melhores resultados.

ONDE ENCONTRAR MODELOS

- Para se manter atualizado com os modelos mais recentes de LLMs, é essencial pesquisar constantemente em repositórios como o Hugging Face, consultar artigos comparativos e acompanhar leaderboards, como veremos a seguir.
- A chave é aprender a se adaptar e a incorporar novos modelos e técnicas conforme eles surgem, garantindo que você sempre trabalhe com as soluções mais avançadas e relevantes.



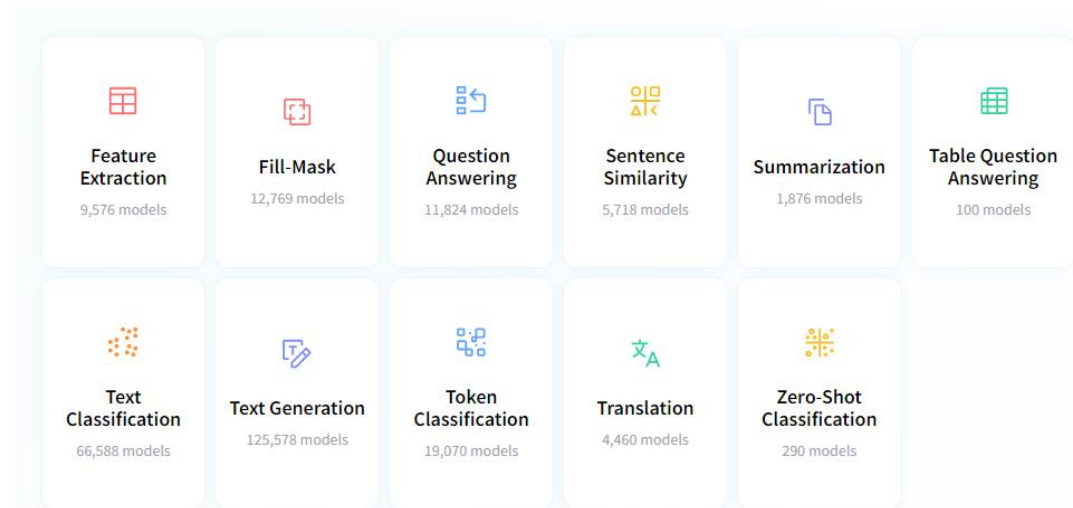
<https://huggingface.co>

HUGGING FACE

- Hugging Face é um repositório de código aberto que oferece modelos, datasets e ferramentas.
- É especialmente conhecido por sua biblioteca Transformers, projetada para aplicações de PLN.
- Além da biblioteca Transformers, há milhares de modelos de linguagem disponíveis gratuitamente no Hugging Face.
- Hugging Face se destacou como um ator principal no ecossistema de LLMs, oferecendo uma plataforma e ferramentas de código aberto que simplificam o uso desses modelos.
- Essa acessibilidade tem capacitado desenvolvedores e pesquisadores a aproveitarem o poder dos LLMs com facilidade e eficiência.

HUGGING FACE - COMO NAVEGAR E ENCONTRAR MODELOS

Acesse: <https://huggingface.co/tasks>



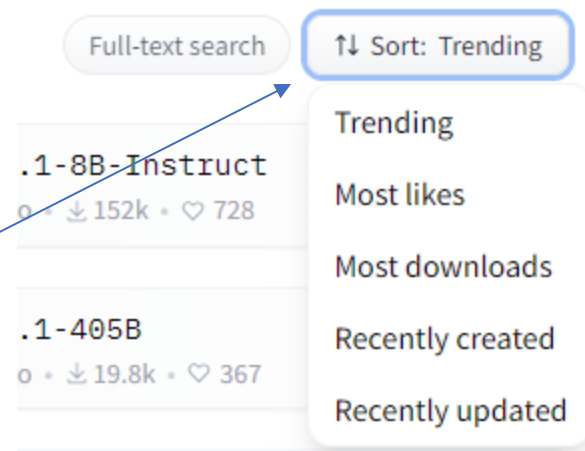
Além de modelos PLN, o Hugging Face também reúne inúmeros modelos de outras áreas da IA, como Visão Computacional. Além disso, dentro da própria plataforma você tem acesso a demonstrações, casos de uso, conjuntos de dados e mais.

HUGGING FACE - COMO NAVEGAR E ENCONTRAR MODELOS

Para acessar LLMs, acesse a opção “Text Generation”, ou acesse o link abaixo

https://huggingface.co/models?pipeline_tag=text-generation&sort=trending

Dica: filtrar por modelos em tendência, ou com maior número de curtidas ou downloads.

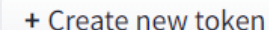


HUGGING FACE – CRIANDO CONTA E GERANDO UM API TOKEN

- Embora não seja obrigatório no geral, há alguns casos mais específicos onde é necessário possuir uma conta, como por exemplo modelos que exigem liberação.
- Portanto, é interessante fazer já seu cadastro: <https://huggingface.co/join>

Gerando um chave de API

- Acesse <https://huggingface.co/settings/tokens>
- Selecione “Create new token”
- Defina a permissão de leitura (READ)
- Copie a chave



A criação deste token é totalmente gratuita e não há cobrança para carregar modelos.

HUGGING FACE – CRIANDO CONTA E GERANDO UM API TOKEN

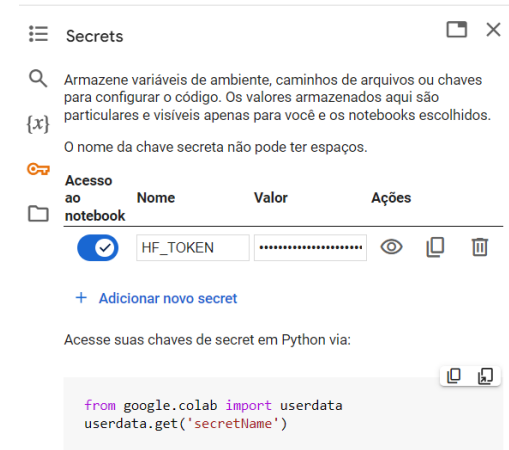
Para usar o token no notebook do Google Colab, use esse código abaixo:

- `os.environ["HF_TOKEN"] = getpass.getpass()`

E no campo que aparecer, cole a sua chave da Hugging Face.

Alternativa: você pode também adicioná-lo às chaves secretas (secret keys) do Colab assim:

- Abra seu notebook Google Colab.
- No lado esquerdo da página, você verá um ícone de chave. Clique nele.
- Clique em “Adicionar uma chave” e insira seu token de acesso.



HUGGING FACE


- A API do Hugging Face se integra muito bem com o LangChain, um framework open source que conecta LLMs, fontes de dados e outras funcionalidades com uma sintaxe unificada.
- Ao longo desse curso, exploraremos essa poderosa ferramenta. Mas de início aprenderemos a implementar LLMs usando a biblioteca Transformers.
- Embora o LangChain forneça mais opções, saber esse modo pode ser útil caso você esteja testando um modelo novo e recém-publicado que ainda não possui tanta compatibilidade. Mesmo com o LangChain, ao lidar com literalmente milhares de modelos diferentes, pode haver certa incompatibilidade ao carregá-los.

 **LangChain**
+
Hugging Face 

ACESSANDO MODELOS NO HUGGING FACE

- Alguns modelos mais específicos exigem aprovação para poder baixar e usar. Porém, para a maioria não é necessário
- Caso seja, então geralmente basta preencher um formulário para se obter permissão de acesso.
- A liberação costuma leva poucos minutos, mas dependendo da quantidade de requisições o tempo pode variar.

Como por exemplo o modelo Llama 3 da Meta
<https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct>

 You need to agree to share your contact information to access this model

The information you provide will be collected, stored, processed and shared in accordance with the [Meta Privacy Policy](#).

LLAMA 3.1 COMMUNITY LICENSE AGREEMENT

Llama 3.1 Version Release Date: July 23, 2024

"Agreement" means the terms and conditions for use, reproduction, distribution and modification of the Llama Materials set forth herein.

"Documentation" means the specifications, manuals and documentation accompanying Llama 3.1 distributed by Meta at <https://llama.meta.com/doc/overview>.

"Licensee" or "you" means you, or your employer or any other person or entity (if you are entering into this Agreement on such person or...

Após liberado, deve chegar uma resposta em seu email.

This is to let you know your request to access model "meta-llama/Meta-Llama-3.1-70B" on huggingface.co has been accepted by the repo authors.

You can now access the repo [here](#), or view all your gated repo access requests [in your settings](#).

Cheers,

The Hugging Face team

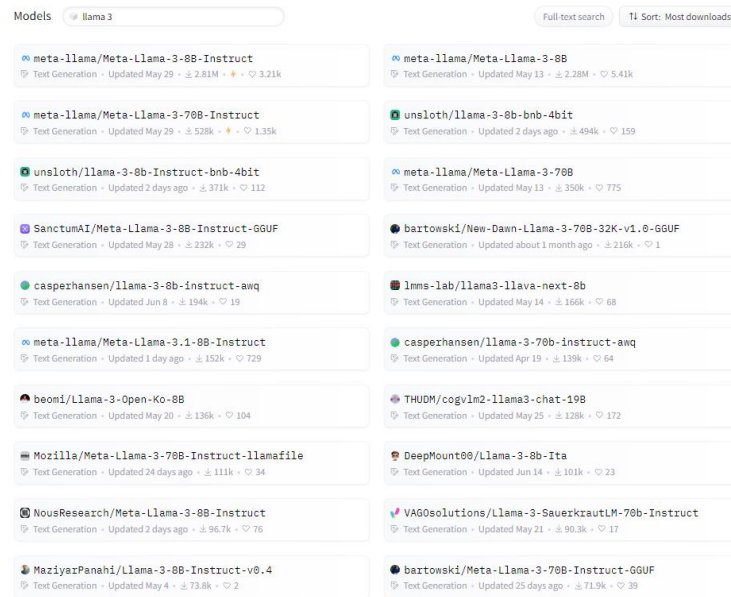
Clique no botão dentro dessa seção e preencha os dados pedidos

ACESSANDO MODELOS NO HUGGING FACE

Como alternativa, você pode usar variações desses modelos, versões fine-tuned publicadas pela comunidade.

Por exemplo, veja ao lado quantas opções aparecem ao pesquisarmos por “llama 3”

Podemos ser mais específicos também, pesquisar pelo formato do modelo (exemplo: GGUF) ou outras características, como por exemplo “portuguese” (veremos mais tarde o porquê pode ser interessante ser mais específico em nossa busca)



https://huggingface.co/models?pipeline_tag=text-generation&sort=trending

TIPOS DE MODELOS

- Ao navegar pelos repositórios de modelos de IA generativa (como o Hugging Face) observará modelos listados com os sufixos 'instruct' ou 'chat'.
- Isso porque há pelo menos três tipos principais de LLMs:
- **Modelos Base (base models)** - Modelos Base passam apenas pelo pré-treinamento e completam textos com as palavras mais prováveis.
- **Modelos Ajustados para Instruções (Instruct-tuned)** - Modelos Ajustados para Instruções passam por uma etapa adicional de ajuste para instruções, melhorando a capacidade de seguir comandos específicos.
- **Modelos de Chat (chat models)** - foram ajustados para funcionar em chatbots, portanto, podem ser mais apropriados para conversas. Alguns modelos de chat podem ser chamados de instrução e vice-versa dependendo da situação, então para facilitar você irá encontrar situações onde um modelo usado para chat é classificado como de instrução.

TIPOS DE MODELOS

- A versão 'instruct' do modelo foi ajustada para seguir instruções fornecidas. Esses modelos 'esperam' ser solicitados a fazer algo.
- Modelos com o sufixo 'chat' foram ajustados para funcionar em chatbots, 'esperando' estar envolvidos em uma conversa com diferentes atores.
- Em contraste, modelos não ajustados para instruções simplesmente geram uma saída que continua a partir do prompt.
- Exemplo de qual modelo usar:
- Para criar chatbots, implementar RAG ou usar agentes, use modelos 'instruct' ou 'chat'.
- Em caso de dúvida, use um modelo 'instruct'.

TIPOS DE MODELOS

Modelos base são bons em prever palavras subsequentes com base em grandes volumes de texto, mas tendem a gerar respostas genéricas, pois são treinados para completar frases, não para responder diretamente a perguntas.

- Por esse motivo, ao perguntar a um modelo base *"Onde vivem os pinguins?"*, uma resposta plausível dada por ele seria: *"Do que se alimentam? Como eles são? [...]"*

Por que isso ocorre? Lembre-se que os LLMs são projetados para completar frases prevendo as palavras mais prováveis com base no texto anterior.

- Por exemplo, para "O céu é", eles provavelmente completarão com "azul" porque é a sequência mais comum.
- Isso ocorre porque os modelos base são treinados para prever a próxima palavra em grandes volumes de texto.
- Eles não respondem perguntas diretamente, pois, estatisticamente, perguntas muitas vezes seguem outras perguntas em seus dados de treinamento.
- Portanto, seu comportamento reflete essas estatísticas, não uma falta de inteligência por parte do modelo.

TIPOS DE MODELOS

Uma possível solução seria modificar um pouco o prompt, para darmos uma dica ao modelo.

- Pegando o exemplo dos pinguins, poderíamos adicionar após a pergunta “*Os pinguins vivem*” – ficando assim: “*Onde vivem os pinguins? Os pinguins vivem*”.
- Com isso teríamos o resultado esperado, pois lembre-se que esse tipo de modelo completa a frase.

Isso que fizemos também é conhecido como **engenharia de prompt**.

Até um certo ponto pode funcionar bem, mas há limitações.

Além disso, não é assim que os grandes e impressionantes modelos LLMs que conhecemos funcionam. Por exemplo, não precisamos dar essas dicas para o ChatGPT, ele já entende o que deve fazer. Isso porque esses modelos são do tipo *Instruct-tuned*.

Para fazer isso e assim chegar a esses resultados superiores e mais adequados a determinados cenários (como por exemplo, expectativa de uma resposta, ou simulação de conversa) é utilizado uma técnica conhecida como **fine-tuning**.

FINE-TUNING – AJUSTANDO MODELOS

- Fine-tuning é uma etapa adicional no processo de criação de um modelo, aprimorando sua capacidade de realizar tarefas específicas.
- Envolve pegar o modelo base pré-treinado e treiná-lo mais com um dataset menor e mais especializado, relevante para a tarefa desejada.
- Para o exemplo de instruções: O objetivo não é ensinar o modelo sobre a informação em si (como "Onde vivem os pinguins?" ou "Qual é a capital da Inglaterra?"), mas sim ensinar que quando uma pergunta é feita, esperamos uma resposta nesse formato.
- O processo de fine-tuning ajusta os parâmetros do modelo para melhor se alinhar às nuances e requisitos dessas tarefas, como responder perguntas, sem perder a compreensão geral da linguagem que já possui.

Os dados fornecidos para treinamento são rotulados, com exemplos de gerações para um prompt específico.

Exemplo de um dataset de instruções:

- *Input: " Onde vivem os pinguins?"*
- *Output: "Vivem principalmente na zona da Antártida [...]"*.
- *Input: "Qual é a capital da Inglaterra?"*
- *Output: "A capital da Inglaterra é Londres".*

MODELOS AJUSTADOS PARA INSTRUÇÕES (INSTRUCT-TUNED)

Portanto, ao desenvolver aplicações com LLM geralmente estamos mais interessados nos modelos **Instruct-tuned** pois são ajustados para receber instruções.

- Depois do pré-treino, onde assimilam uma enorme quantidade de texto sem uma finalidade específica além de reproduzir esses textos, os modelos passam por mais uma rodada de treinamento.
- Essa segunda rodada é conhecida como Tuning (Ajuste), ou mais precisamente, Instruct Tuning (ajuste para instruções).
- Primeiro, o modelo adquire conhecimento e, em seguida, aprende a tornar esse conhecimento mais útil para os usuários.
- Nesse momento, o modelo aprende que, ao receber uma pergunta, deve gerar uma resposta, e não outra pergunta.

MODELOS AJUSTADOS PARA INSTRUÇÕES (INSTRUCT-TUNED)

- Esse ajuste do modelo para seguir instruções é realizado por meio de uma segunda etapa de treino.
- Alimentamos o LLM com um dataset de instruções durante essa nova fase de treino.
- Ensina-se ao modelo que, quando há uma pergunta na entrada, a saída deve ser uma resposta a essa pergunta.
- Quando há um pedido de atividade, o modelo deve executar essa atividade.
- Um exemplo de dataset de instruções Open Source em português é o Cabrita, uma variação brasileira do dataset do Llama.
- <https://github.com/22-hours/cabrita>

TIPOS DE MODELOS

- Por exemplo o modelo Llama 3, modelo open source da empresa Meta e que no segundo trimestre de 2024 demonstrou superar em diversas tarefas versões atuais e recentes de modelos proprietários como o próprio ChatGPT.

<https://huggingface.co/meta-llama/Meta-Llama-3-8B>

[∞ meta-llama/](https://huggingface.co/meta-llama/Meta-Llama-3-8B)**Meta-Llama-3-8B**

<https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct>

[∞ meta-llama/](https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct)**Meta-Llama-3-8B-Instruct**

ESCOLHENDO MODELOS

Detalhes que são interessantes ficar de olho na hora de escolher o modelo:

- Data Freshness – a data máxima dos dados de treinamento mais recentes. Ou seja, até que data o modelo possui conhecimento.

Por exemplo, o Meta-Llama-3.1-8B-Instruct possui até dezembro de 2023.

- Na página de descrição do modelo pode haver também outras informações interessantes, como idiomas suportados

Avaliar modelos com Benchmarks

- Além disso, pode ser interessante se atentar às métricas, embora seja mais relevante acessar outros locais, leaderboards ou curadorias e teste comparativos, para ver como os modelos se comportam em benchmarks amplamente aceitos.

LEADERBOARD (TABELAS DE CLASSIFICAÇÃO)

Leaderboards são classificações que comparam diferentes modelos de LLMs com base em métricas de desempenho. Eles ajudam a:

- Avaliar e comparar a eficácia de diversos modelos.
- Identificar os modelos mais atualizados e eficazes.
- Fornecer insights sobre qual modelo é mais adequado para diferentes aplicações.
- Ajudar desenvolvedores e pesquisadores na escolha de modelos, mostrando quais são os mais bem avaliados em termos de precisão e eficiência.

Essas classificações são essenciais para se manter atualizado sobre os avanços no campo de LLMs.

- Exemplo de um bastante referenciado: <https://chat.lmsys.org/?leaderboard>

MODELOS PARA IDIOMAS ESPECÍFICOS

- Caso formos desenvolver uma aplicação onde as respostas serão retornadas em inglês então podemos usar como referência apenas esses placares mais conhecidos.
- No entanto, como nossa intenção é desenvolver aplicações no idioma português, é interessante que procuremos por modelos que boas avaliações em datasets desse idioma desejado.
- Para isso, podemos procurar por leaderboards focados no idioma desejado.
- Por exemplo, para o português:
https://huggingface.co/spaces/eduagarcia/open_pt_llm_leaderboard

E aqui uma lista atualizado com melhores modelos: <https://huggingface.co/collections/eduagarcia/portuguese-llm-leaderboard-best-models-65c152c13ab3c67bc4f203a6>

MODELOS PARA IDIOMAS ESPECÍFICOS

Aqui você pode filtrar por diversos atributos diferentes – como por exemplo tamanho do modelo (quantos bilhões de parâmetros), ou tipo (open source ou proprietários).

Além disso, pode ordenar a tabela com base na métrica/dataset mais interessante, basta clicar sobre o nome da coluna na tabela

LLM Benchmark

Metrics

About

Submit here!

Changelog

Search for your model (separate multiple queries with ';' and press ENTER...)

Select columns to show

☒ Average

☒ ENEM

☒ BLUEX

☒ OAB Exams

☒ ASSIN2 RTE

☒ ASSIN2 STS

☒ FAQUAD NLI

☒ HateBR

☒ PT Hate Speech

☒ tweetSentBR

☐ Type

☐ Architecture

☐ Precision

☐ Merged

☐ Hub License

☐ #Params (B)

☐ Hub

☐ Model sha

☐ Revision

☐ Evaluation Time (s)

☐ Leaderboard Average

☐ NPM (Average)

☐ Main Language

Hide models

☐ Proprietary

☐ Private or deleted

☐ Contains a merge/moerge

☒ Flagged

☐ MoE

Model types

☒ pretrained

☒ language adapted (FP, FT, ...)

☒ fine-tuned/lp on domain-specific datasets

☐ chat (RLHF, DPO, IFT, ...)

☒ base merges and moerges

☒ proprietary (closed)

☐ ?

Precision

☒ float16

☒ bfloat16

☒ 8bit

☒ 4bit

☒ GPTQ

☐ ?

Model sizes (in billions of parameters)

☐ ?

☒ ~1B

☒ ~3B

☒ ~8B

☒ ~16B

☒ ~35B

☒ ~70B

☒ 100B+

Model Main Language

☒ Portuguese

☒ English

☒ Chinese

☒ Spanish

☒ Other

☐ ?

AMBIENTE PARA IMPLEMENTAÇÃO

Usaremos o Colab para aproveitar a GPU gratuita, recomendamos fazer por lá pelo menos num primeiro momento.

Observação sobre o Colab: se atingir a quota de uso da GPU é necessário esperar algumas horas para poder usar novamente.

Caso aconteça, você pode:

- Executar localmente (caso não possua um hardware bom, precisa ser via API)
- Usar a soluções pagas (como a Open AI)
- Usar outro serviço que disponibiliza GPU gratuita, como o SageMaker Studio Lab ou Kaggle

IMPLEMENTAÇÃO COM BIBLIOTECA TRANSFORMERS (HUGGINGFACE)

Modelo escolhido para os testes iniciais: Phi-3 Mini-4K-Instruct

<https://huggingface.co/microsoft/Phi-3-mini-4k-instruct>

- Justificativa: é open source, acessível e consegue responder bem em português (embora ainda seja melhor em inglês). Verá que muitos modelos não entendem esse idioma, e os que entendem são muito pesados para executarmos em nosso ambiente, ou seja, precisamos acessar via alguma API ou interface web, como o ChatGPT. Porém nesse momento queremos explorar soluções open source, para obtermos maior liberdade.
- Os modelos Phi oferecem muitas das mesmas capacidades dos LLMs, mas são menores em tamanho e treinados em conjuntos de dados menores, tornando-os mais econômicos e acessíveis.
- A família Phi-3, incluindo modelos como o Phi-3-mini, apresenta desempenho impressionante em vários benchmarks.

DETALHES DO MODELO

microsoft/Phi-3-mini-4k-instruct

Organização (*microsoft*):

- Pode ser empresa ou pessoa que mantém o modelo. Esta parte indica que o modelo é desenvolvido e mantido pela Microsoft, empresa líder em tecnologia com ampla pesquisa e desenvolvimento em IA e aprendizado de máquina.

Família do modelo (*Phi-3-mini*):

- Phi-3: Isso denota a terceira iteração da série de modelos Phi da Microsoft.
- Mini: Isso indica que este modelo em particular é uma versão menor dentro da família Phi-3. Ele foi projetado para ser eficiente e leve, ao mesmo tempo em que oferece desempenho robusto.

DETALHES DO MODELO

microsoft/Phi-3-mini-4k-instruct

4k (tamanho): Isso se refere ao tamanho da janela de contexto do modelo ou comprimento da sequência, que neste caso é de 4.000 tokens. Isso significa que o modelo pode manipular até 4.000 tokens em uma única entrada, permitindo que ele processe e gere sequências de texto mais longas de forma mais eficaz.

Instruct: conforme citado anteriormente, isso indica que o modelo é ajustado para tarefas de acompanhamento de instruções. Modelos com "instruct" em seu nome são normalmente otimizados para entender e gerar respostas com base em instruções ou prompts específicos, tornando-os adequados para tarefas como resposta a perguntas, geração de texto guiado e muito mais.

TOKENIZER

- Em nossa configuração, também precisamos carregar o tokenizer associado ao modelo. O tokenizer é crucial para preparar dados de texto em um formato que o modelo possa entender.
- Um tokenizador converte texto bruto em tokens, que - conforme comentado anteriormente - são representações numéricas que o modelo pode processar.
- Ele também converte os tokens de saída do modelo de volta em texto legível por humanos.
- Os tokenizadores lidam com tarefas como dividir texto em palavras ou subpalavras, adicionar tokens especiais e gerenciar mapeamento de vocabulário.

TOKENIZER

Definição do Tokenizer

- Gerenciamento de vocabulário: O tokenizador inclui um vocabulário que mapeia cada token para um identificador exclusivo. Isso garante que as mesmas palavras ou subpalavras sejam consistentemente representadas pelos mesmos tokens.
- Tokens especiais: O tokenizador manipula tokens especiais como [CLS], [SEP] e [PAD], que são usados para propósitos específicos no processamento de modelos (por exemplo, indicar o início ou o fim de uma sequência, preencher sequências mais curtas).
- Manipulando sequências longas: Por exemplo o modelo Phi-3-mini-4k-instruct. Dado que este modelo é otimizado para uma janela de contexto de até 4.000 tokens (4k), o tokenizador gerencia sequências longas de forma eficiente, truncando-as ou preenchendo-as adequadamente conforme necessário.

PIPELINE

O que é uma pipeline?

- Um pipeline é uma abstração que simplifica o uso de modelos pré-treinados para uma variedade de tarefas de PNL. Ele fornece uma API unificada para diferentes tarefas, como geração de texto, classificação de texto, tradução e muito mais.
- Os pipelines lidam com todo o processo de obtenção de dados de entrada brutos, pré-processamento usando o tokenizador, execução pelo modelo e pós-processamento da saída do modelo para produzir resultados legíveis por humanos.
- Essa interface de alto nível abstrai as complexidades de trabalhar diretamente com modelos e tokenizadores, facilitando a execução de tarefas comuns com código mínimo.

ENGENHARIA DE PROMPT

- **Engenharia de prompt** é a arte de formular e otimizar instruções fornecidas a modelos de IA generativa para obter respostas mais precisas e relevantes.
- Ela envolve a criação de perguntas ou comandos que orientam o modelo a produzir a saída desejada, maximizando a eficiência e a qualidade da resposta.
- É um processo bastante empírico, portanto, é interessante testar bem as variações.

Exemplo Simples:

- **Antes da Engenharia de Prompt:**

Pergunta: "Conta uma história."

Resposta: "Era uma vez uma princesa que morava em um castelo."

- **Depois da Engenharia de Prompt:**

Pergunta: "Conta uma história sobre uma princesa corajosa que salva seu reino de um dragão usando sua inteligência."

Resposta: "Era uma vez uma princesa corajosa que, usando sua inteligência, salvou seu reino de um dragão aterrorizante com um plano engenhoso."

MELHORANDO RESULTADOS COM ENGENHARIA DE PROMPT

- Sempre confira se o seu prompt não poderia ser mais específico. Se mesmo melhorando o prompt estiver com dificuldades de atingir o resultado esperado (e após experimentar outros parâmetros), então o modelo não é tão apropriado para essa tarefa.
- Por exemplo, para geração de código: Uma ideia/sugestão de prompt que poderia ser conveniente para a geração de código, caso queira usar a LLM como seu co-piloto: *"Refatore utilizando conceitos como SOLID, Clean Code, DRY, KISS e caso seja possível aplique um ou mais design patterns adequados visando escalabilidade e performance, criando uma estrutura de pastas organizadas e separando por arquivos"* (e claro, aqui você pode modificar à vontade)
- Observação: as vezes é melhor manter o prompt simples e não muito elaborado. Colocar informações ou referências diferentes demais pode "confundir". Portanto é bastante interessante ir adicionado ou removendo termo por termo caso esteja experimentando e atrás de melhores resultados

ENGENHARIA DE PROMPT

Alguns exemplos de ferramentas e serviços online que podem auxiliar na engenharia de prompt



Por exemplo o LangSmith, que faz parte do ecossistema LangChain <https://smith.langchain.com/hub>

MELHORANDO RESULTADOS COM MUDANÇA DE MODELOS

- A engenharia de prompt pode melhorar apenas até certo ponto, dependendo da complexidade do pedido. Para alcançar resultados mais assertivos você pode buscar modelos maiores e mais modernos com mais parâmetros (lembre-se da troca eficiência x qualidade das respostas).
- Aqui pode ser muito útil as dicas fornecidas anteriormente sobre procurar modelos mais recentes, modelos em tendência ou que possuem maiores pontuações para benchmarks específicos.
- Outra opção ainda é procurar por modelos com foco na tarefa desejada, por exemplo em geração de código ou em conversas/chat.
- Vamos pegar o caso de geração de código; para esse objetivo, você poderia usar o modelo deepseek-coder de 6.7B por exemplo (ou procurar por outros com esse foco)

QUANTIZAÇÃO

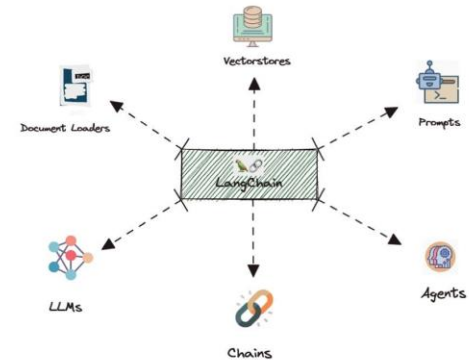
- A **quantização** reduz a precisão dos números usados para representar os parâmetros do modelo, diminuindo o uso de memória e os requisitos computacionais.
- Ao invés de vez de usar números de ponto flutuante de 32 bits (float32), o modelo pode usar números de 16 bits (float16) ou até de 8 bits (int8).
- Este processo pode reduzir significativamente o tamanho do modelo e acelerar a inferência sem causar uma perda substancial na precisão.
- **Benefícios para LLMs:**
 - Permite rodar grandes modelos em hardware com recursos limitados.
 - Mantém um bom desempenho sem comprometer significativamente a precisão.
 - Facilita a execução de LLMs em dispositivos móveis e sistemas em tempo real.
 - Ideal para quem usa plataformas como o Google Colab com recursos gratuitos.
- Em nossos exemplos iniciais, usamos o quantização de 4 bits (do módulo BitsAndBytesConfig da biblioteca Transformers) <https://huggingface.co/blog/4bit-transformers-bitsandbytes>

QUANTIZAÇÃO

- O fine-tuning também pode ser usado para melhorar desempenho, além de adicionar comportamentos desejáveis (ou remover comportamentos indesejáveis).
- No entanto, o fine-tuning de modelos muito grandes é caro; para ter uma ideia, o ajuste fino regular de 16 bits de um modelo LLama com 65 bilhões de parâmetros requer mais de 780 GB de memória de GPU.
- Abordagens de quantização como a QLoRA ajudam a reduzir os requisitos de memória para fine-tuning de um modelo com 65 bilhões de parâmetros de mais de 780 GB de memória de GPU para menos de 48 GB, sem degradar o tempo de execução ou o desempenho preditivo em comparação com uma linha de base totalmente ajustada com 16 bits.

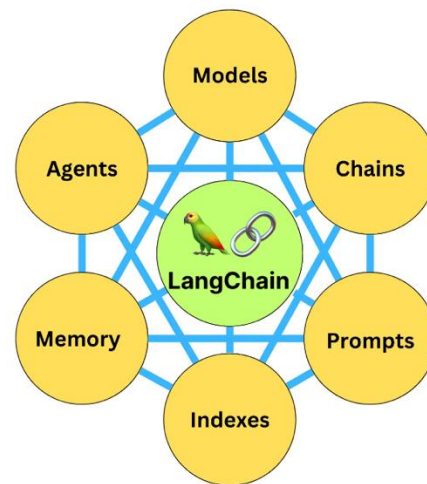
LANGCHAIN

- É uma biblioteca de código aberto projetada para facilitar a integração de LLMs com várias fontes de dados e funcionalidades adicionais.
- Desenvolvido por Harrison Chase e lançado em outubro de 2022.
- Fornece uma sintaxe unificada que simplifica o uso de LLMs em diferentes contextos, como chatbots, análises de texto e sistemas de perguntas e respostas.
- LangChain torna mais fácil combinar LLMs com outras ferramentas e serviços.
- Além disso, permite que desenvolvedores criem soluções avançadas de processamento de linguagem natural de forma eficiente e escalável.
- É uma solução muito importante para deixar as LLMs “programáticas” e criar aplicações próprias.



LANGCHAIN – PRINCIPAIS COMPONENTES

- **Modelos:** Oferece uma interface padrão para interações com uma ampla gama de LLMs.
- **Prompts:** Ferramentas para simplificar a criação e tratamento de prompts dinâmicos.
- **Chains** (Corrente, Cadeia ou Sequencia): Interface padrão para encadear LLMs em aplicações complexas, permitindo a ligação entre múltiplos modelos ou outros módulos especializados.
- **Memória:** Módulos que permitem o gerenciamento e alteração de conversas anteriores, essencial para chatbots que precisam relembrar interações passadas para manter coerência.
- **Agentes:** Equipados com um kit de ferramentas abrangente, podem escolher quais ferramentas usar com base nas informações do usuário.
- **Índices:** Métodos para organizar documentos (que contém dados proprietários, por exemplo) de forma a facilitar a interação eficaz com LLMs.



Créditos da imagem: [ByteByteGo](#)

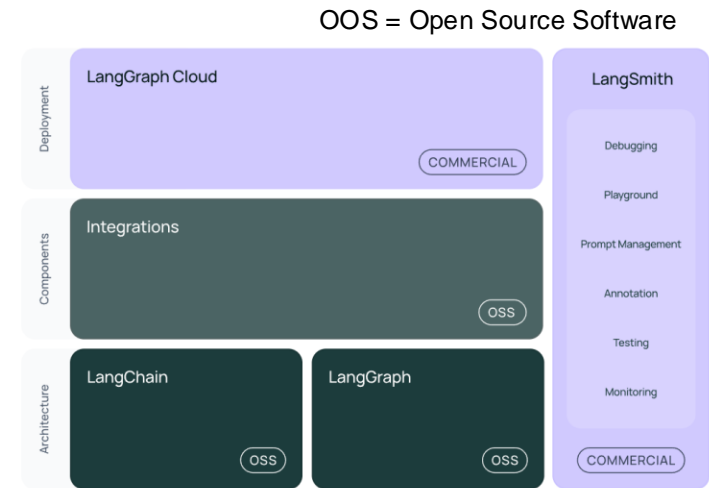
IMPLEMENTAÇÃO COM LANGCHAIN

Considerações importantes:

- O LangChain é uma biblioteca em constante desenvolvimento, projetada para facilitar a construção de aplicações alimentadas por grandes modelos de linguagem.
- É importante prestar atenção aos avisos (warnings) que aparecem durante o uso, pois algumas funções podem ser depreciadas e deixar de funcionar a partir de determinadas versões.
- Manter a biblioteca sempre na mesma versão pode evitar problemas de depreciação, mas é recomendado atualizar periodicamente para ter acesso aos recursos mais modernos e otimizados.
- O LangChain representa uma solução de ponta, e a atualização frequente é o preço a pagar para usufruir das melhores ferramentas disponíveis para construção de aplicações avançadas.
- Portanto, dependendo de quando estiver assistindo, algumas funções podem precisar atualização no nome. Caso seja necessário, atualizaremos o Colab (ou você pode verificar a mensagem do warning, que em alguns casos explica a mudança necessária)

ECOSSISTEMA LANGCHAIN

- langchain-core: Abstrações básicas e LangChain Expression Language (LCEL).
- langchain-community: Integrações de terceiros. Pacotes parceiros (por exemplo, langchain-openai, langchain-anthropic, etc.): Algumas integrações foram divididas em seus próprios pacotes leves que dependem apenas do langchain-core.
- langchain: Chains, Agentes e Estratégias de Retrieval que compõem a arquitetura cognitiva de uma aplicação.
- LangGraph: Para construir aplicações robustas e com estado para múltiplos atores com LLMs, modelando etapas como arestas e nós em um gráfico. Integra-se perfeitamente com LangChain, mas pode ser usado sem ele.
- LangServe: Para implementar chains do LangChain como APIs REST.
- LangSmith: Uma plataforma para desenvolvedores que permite depurar, testar, avaliar e monitorar aplicações LLM.



<https://python.langchain.com/v0.2/docs/introduction/>

LANGCHAIN + HUGGING FACE

- A integração de Hugging Face com LangChain traz diversos benefícios, conforme citado anteriormente.
- Hugging Face disponibiliza uma ampla variedade de modelos pré-treinados que podem ser facilmente incorporados às suas aplicações.
- LangChain facilita essa incorporação, fornecendo uma interface uniforme e ferramentas extras para melhorar desempenho e eficiência.
- Com essas ferramentas combinadas, você pode se focar mais na lógica de negócios e menos nas questões técnicas.

LANGCHAIN – MODELOS

- Uma das principais vantagens do LangChain é que ele permite trabalhar facilmente com diversos modelos. Alguns modelos são melhores para determinadas tarefas ou oferecem um melhor custo-benefício. Portanto, você provavelmente irá explorar diferentes modelos durante seus testes.
- O LangChain tem integrações com muitos provedores de modelos (OpenAI, Cohere, Hugging Face, Anthropic, Google, etc.) e expõe uma interface padrão para interagir com todos esses modelos.
- No LangChain, ao trabalhar com modelos nós definimos o que é chamado de **LLM Wrapper**. Um wrapper é como uma "embalagem" que facilita a utilização dos Grandes Modelos de Linguagem em aplicações.
- Já a **LLM** em si funciona como o "cérebro" ou motor da aplicação, realizando o processamento de linguagem natural.

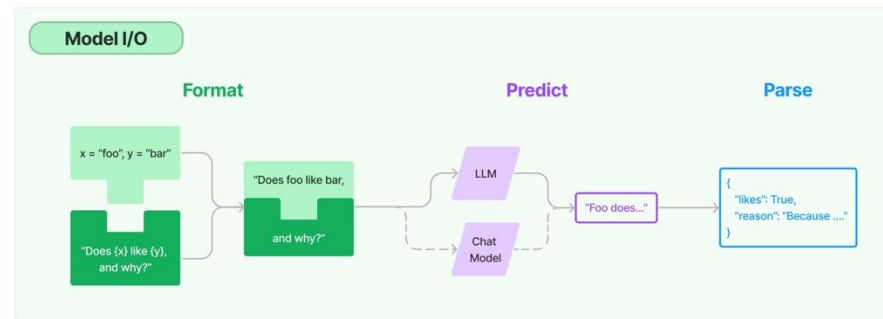
LANGCHAIN – ENTRADA E SAÍDA DE MODELOS

A biblioteca divide o gerenciamento de modelo em **LLMs** e **Chat Models**.

- LLMs - LangChain não serve seus próprios LLMs, mas fornece uma interface padrão para interagir com muitos LLMs diferentes. Para ser específico, essa interface recebe como entrada uma string e retorna uma string.
- Chat Models - usa mensagens de chat como entradas e retorna mensagens de chat como saídas.

Conforme desenvolvemos nossa aplicação, iremos entender as diferenças e casos de uso para cada.

O LangChain fornece os blocos de construção para interagir com qualquer modelo de linguagem.



Fonte da imagem: [Documentação do LangChain](#)

COMPONENTES - LLMS E CHAT MODELS

Componente - Chat Models

- Modelos de linguagem mais novos usam sequências de mensagens como entradas e retornam mensagens de chat como saídas.
- Esses modelos permitem atribuir funções distintas às mensagens, diferenciando entre IA, usuários e instruções do sistema.
- Embora trabalhem com mensagens de entrada e saída, os wrappers LangChain permitem que esses modelos recebam strings como entrada. Isso facilita o uso de modelos de chat no lugar de LLMs tradicionais.
- Quando uma string é passada como entrada, ela é convertida em uma HumanMessage e então processada pelo modelo.

Componente - LLMs

- Modelos de linguagem que recebem uma string como entrada e retornam uma string.
- Tradicionalmente, esses são modelos mais antigos (modelos mais novos geralmente são modelos de chat).
- Embora os modelos subjacentes trabalhem com string in / string out, os wrappers LangChain permitem que esses modelos recebam mensagens como entrada.
- As mensagens recebidas são formatadas em uma string antes de serem passadas para o modelo. LangChain não hospeda nenhum LLM, mas depende de integrações de terceiros.

COMPONENTES - LLMS E CHAT MODELS

- Ou seja: LLMS de texto puro de entrada/saída de texto tendem a ser mais antigos ou de nível mais baixo.
- Muitos modelos populares são mais bem usados como modelos de chat / bate-papo (chat completion models), mesmo para casos de uso que não sejam de chat.
- O LangChain prioriza o Chat Models por estar mais associado a seu uso com modelos mais modernos (pelo menos na versão atual).

MENSAGENS

Alguns modelos de linguagem pegam uma lista de mensagens como entrada e retornam uma mensagem. Existem alguns tipos diferentes de mensagens.

No LangChain, todas as mensagens têm uma propriedade ``role``, ``content`` e ``response_metadata``.

- A função (role) descreve quem está dizendo a mensagem (ex: human, system). LangChain tem diferentes classes de mensagem para diferentes funções.
- A propriedade conteúdo (content) descreve o conteúdo da mensagem, podendo ser: uma string (a maioria dos modelos lida com esse tipo de conteúdo); ou uma lista de dicionários (isso é usado para entrada multimodal, onde o dicionário contém informações sobre esse tipo de entrada e esse local de entrada)
- A propriedade `response_metadata` contém metadados adicionais sobre a resposta. Os dados aqui são frequentemente específicos para cada provedor de modelo. É aqui que informações como *log-probs* (probabilidades de log) e uso de token podem ser armazenadas.

OPÇÕES DE QUANTIZAÇÃO

Como alternativa, ao invés de realizarmos a quantização nós mesmos usando o BitsAndBytesConfig, podemos baixar modelos já quantizados e também em formatos de arquivo compactos e eficientes, como o **GGUF**, em comparação com os formatos convencionais como Pytorch e Safetensors.

Outras abordagens de otimização:

- **GPTQ** - Generalized Post-Training Quantization
- **AWQ** - Activation-aware Weight Quantization

No Hugging Face podemos encontrar centenas de modelos convertidos para esse formato.

- Por exemplo para o formato GGUF, acesse aqui e confira https://huggingface.co/models?pipeline_tag=text-generation&sort=trending&search=gguf

Para realizar a inferência de modelos nesse formato nós podemos usar a biblioteca Llama.cpp, que possui compatibilidade com o LangChain.

FORMATOS DE ARQUIVO COMPACTOS E EFICIENTES PARA LLMS

GGML: Criado por Georgi Gerganov. Facilita o uso de modelos em vários hardwares, incluindo CPUs.

- Limitações do GGML: Problemas de compatibilidade e necessidade de ajustes manuais complexos.

GGUF: A Evolução do GGML, lançado em agosto de 2023. Desenvolvido pela comunidade de IA, incluindo Gerganov.

- Objetivo: Resolver as limitações do GGML.
- Oferece maior flexibilidade e estabilidade, ao mesmo tempo mantendo a compatibilidade com modelos antigos.

Vantagens e Desvantagens do GGUF

- Vantagens: Experiência de usuário aprimorada, suporte a uma variedade de modelos, incluindo os LLaMA da Meta e muitos outros.
- Desvantagens: Conversão de modelos existentes pode ser demorada e exigir adaptação dos usuários.

MODELOS DE PROMPT (PROMPT TEMPLATES)

- Os modelos de prompt (Prompt Templates) ajudam a traduzir a entrada e os parâmetros do usuário em instruções para um modelo de linguagem.
- Pode ser usado para orientar a resposta de um modelo, ajudando-o a entender o contexto e gerar uma saída relevante e coerente baseada em linguagem.
- Isso principalmente facilita a criação de prompts de maneiras variáveis.
- Com o Langchain, temos uma maneira eficiente de conectar isso aos diferentes LLMs que existe.
- Para mudar a LLM, basta alterar o código anterior de carregamento, e o código seguinte permanece igual. Ou seja, é um modo muito mais eficiente caso esteja querendo desenvolver aplicações profissionais.

```
from langchain_core.prompts import PromptTemplate

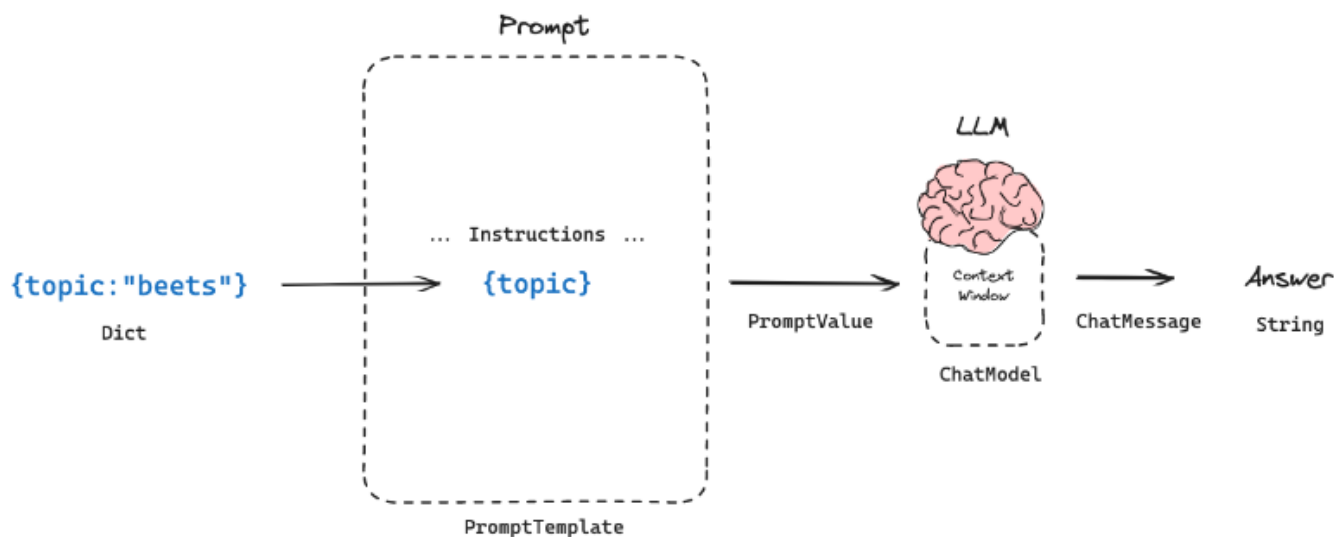
prompt_template = PromptTemplate.from_template("Escreva um poema sobre {topic}")

prompt_template.invoke({"topic": "abacates"})

StringPromptValue(text='Escreva um poema sobre abacates')
```

MODELOS DE PROMPT (PROMPT TEMPLATES)

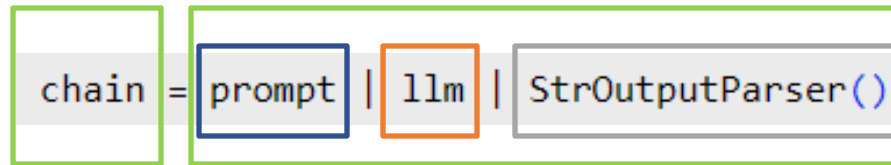
Visualização dos Prompt Templates no LangChain



Créditos da imagem: [Jacob Lee](#)

CHAINS (CORRENTES)

- As Chains do LangChain (que podem ser traduzidas para Corrente, Cadeias ou ainda Sequências) permitem ligar diversos prompts para construir aplicações complexas, facilitando a criação de uma aplicação de IA ao quebrar problemas maiores em partes menores.
- Elas funcionam ao encadear componentes, onde a saída de um se torna a entrada do próximo, criando uma sequência lógica de operações.
- Em outras palavras, permitem conectar e "amarrar" diferentes componentes, possibilitando a criação de sequências de prompts.
- Chains são estáticas (hard-coded), esperando sempre um tipo específico de entrada.



- A Linguagem de Expressão de Langchain (LCEL - LangChain Expression Language), é uma maneira declarativa de encadear os componentes da biblioteca
- A LCEL é uma abstração de alguns conceitos interessantes do Python em um formato que permite uma camada de código "minimalista" para construir cadeias de componentes LangChain.
- Ela permite um desenvolvimento rápido de chains pois como podemos ver, a sintaxe é bastante prática e flexível. Além disso, permite incorporar recursos avançados como streaming, assíncrono, execução paralela e muito mais.

LCEL

- Desde a versão v0.2 do Langchain, é incentivado o uso do LCEL para maior praticidade, flexibilidade e recursos adicionais que eles oferecem
- Um modo de verificar o aumento de versatilidade é comparar as sintaxes, conforme pode ser visto abaixo

Modo anterior (legado)

```
from langchain.chains import LLMChain
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI

prompt = ChatPromptTemplate.from_messages(
    [{"user", "Tell me a {adjective} joke"}],
)

chain = LLMChain(llm=ChatOpenAI(), prompt=prompt)

chain.invoke({"adjective": "funny"})
```

Com LCEL

```
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI

prompt = ChatPromptTemplate.from_messages(
    [{"user", "Tell me a {adjective} joke"}],
)

chain = prompt | ChatOpenAI() | StrOutputParser()

chain.invoke({"adjective": "funny"})
```

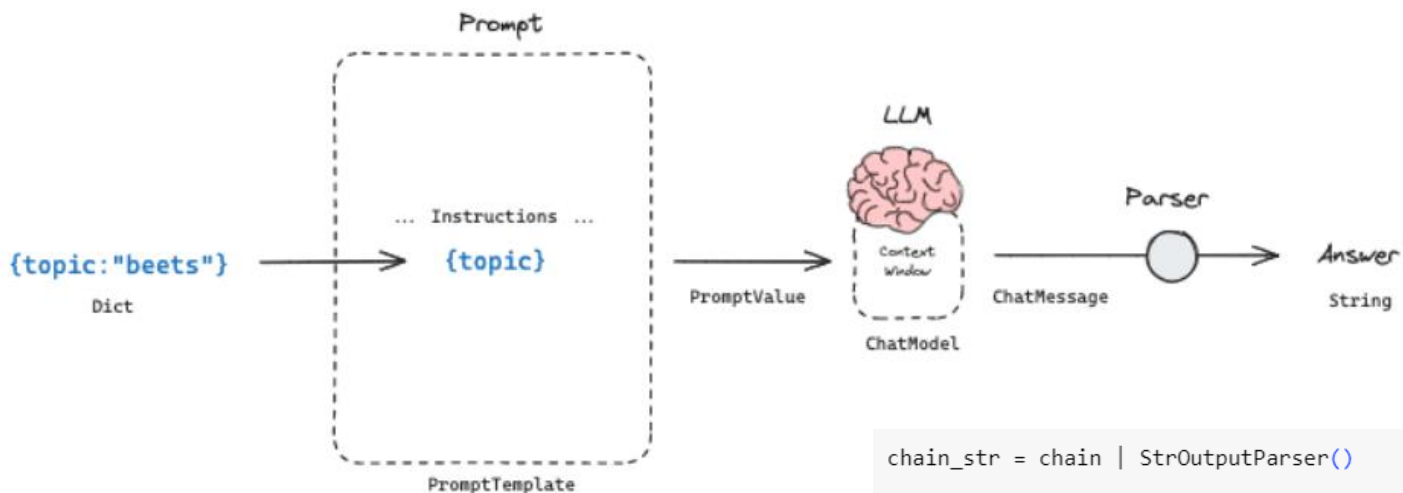
OUTPUT PARSER

- LangChain possui um componente chamado Output Parser (que significa "analisador de saída", tradução livre), que, como o nome indica, é responsável por processar a saída de um modelo em um formato mais acessível ou adequado para nosso objetivo.
- Isso é muito útil quando você está usando o LLMs para gerar qualquer forma de dados estruturados.

Por exemplo o StrOutputParser: esse analisador apenas converte a saída de um modelo de linguagem em uma string. Se o modelo for um modelo carregado pelo componente LLM (e, portanto, produzir uma string), ele apenas passa por essa string. Se a saída for um ChatModel (e, portanto, produzir uma mensagem), ele passa pelo atributo `.content` da mensagem.

CHAINS / OUTPUT PARSER

Estendendo uma chain usando o StrOutputParser por exemplo



```
chain_str = chain | StrOutputParser()  
  
# Isso é equivalente a:  
# chain_str = prompt | llm | StrOutputParser()
```

Créditos da imagem: [Jacob Lee](#)

IMPLEMENTAÇÃO LANGCHAIN LOCALMENTE

Preparando o Ambiente Local

- Instalar Python
- Instalação do Visual Studio Code (recomendamos essa pois é a que explicaremos o processo, mas pode usar outra IDE de sua preferência)

IMPLEMENTAÇÃO LANGCHAIN LOCALMENTE

Instalar os comandos pip para instalação do langchain e demais bibliotecas

- Para fazer isso no VS Code, selecione: Terminal > New Terminal
(Ctrl + Shift + ')

Então, digite os comandos de instalação pip para as bibliotecas que instalamos no Colab:

```
pip install -q transformers einops accelerate bitsandbytes
```

```
pip install -q langchain langchain_community [...]
```

Obs: para o pytorch, recomendamos usar esse comando para se evitar incompatibilidade:

```
pip install torch==2.3.1 torchvision torchaudio --index-url  
https://download.pytorch.org/whl/test/cu121
```

IMPLEMENTAÇÃO EM MÁQUINA LOCAL – PROBLEMAS COMUNS

Se ocorrer problema de comando pip não reconhecido (no Windows)

```
PS C:\Users\gabri> pip --version
pip : O termo 'pip' não é reconhecido como nome de cmdlet, função, arquivo de script ou programa operável. Verifique a grafia do nome ou, se um caminho tiver sido incluído, veja se o caminho está correto e tente novamente.
No linha:1 caractere:1
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (pip:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException
```

Solução:

- Pesquisar pela versão do Python (menu iniciar), botão direito > Abrir local do Arquivo
- Depois, seleciona Python 3.12 (ou a versão que estiver usando) e seleciona novamente Abrir local do arquivo
- Após isso, abra a pasta Scripts e com a pasta aberta, copie o caminho da pasta Exemplo: C:\Users\<seu_usuario>\AppData\Local\Programs\Python\Python312\Scripts
- Após copiado, vá até "Este Computador" e em local vazio clique com o botão direito e selecione Propriedades
- Depois, vá em Configurações avançadas do sistema > Variáveis do ambiente
- Em "Variáveis de usuário" abra a pasta Path
- Clique em novo e cole o caminho copiado
- Após isso, clique OK em tudo e feche as janelas

IMPLEMENTAÇÃO EM MÁQUINA LOCAL – PROBLEMAS COMUNS

Se não encontrar a biblioteca instalada

- Selecione View > Command Pallete
(ou pressione F1, ou Ctrl + Shift + P)
- Digite: Python: Select Interpreter
e selecione Enter

IMPLEMENTAÇÃO EM MÁQUINA LOCAL - API KEYS

Configurando as API Keys

- Após instalada as bibliotecas necessárias, podemos criar um projeto novo, onde colocaremos os arquivos.
- Para carregar os tokens, ao invés do `os.environ` usaremos um método mais prático (e recomendado para esse caso)
- Utilizaremos a biblioteca `dotenv`, que simplifica a gestão de variáveis de ambiente ao armazená-las em um arquivo `.env`.
- Este arquivo contém pares de key-values que representam configurações sensíveis e específicas do ambiente, como senhas e tokens.
- A biblioteca `python-dotenv` lê o conteúdo do arquivo e carrega essas variáveis no ambiente de execução da aplicação.
- Isso melhora a segurança, facilita a configuração e oferece flexibilidade entre diferentes ambientes de desenvolvimento. Assim, é possível realizar mudanças de configuração sem alterar o código-fonte, o que simplifica o desenvolvimento local.

IMPLEMENTAÇÃO EM MÁQUINA LOCAL - API KEYS

Configurando as API Keys

Para realizar esse método, siga as etapas:

- Primeiro, instale o dotenv no terminal => `pip install python-dotenv`
- Dentro do seu arquivo .py, faça a importação => `from dotenv import load_dotenv`
- Crie o arquivo .env e deixe-o no diretório raiz do projeto.
- Nesse arquivo .env, você vai colocar nele todas as Keys que usamos até então (e outras que serão usadas no futuro), nesse formato abaixo, com 1 chave por linha

```
HUGGINGFACE_API_KEY=#####
```

```
OPENAI_API_KEY=#####
```

- Por fim, chame o método `load_dotenv()` no código (de preferência na primeira linha logo após as importações).
- Com isso, as Keys serão carregadas em seu programa

IMPLEMENTAÇÃO EM MÁQUINA LOCAL

Para rodar localmente, iremos fazer testes com uma solução proprietária via API e uma solução open source que pode ser executada 100% localmente de modo gratuito.

Para modelos proprietários

- Iremos usar o **ChatGPT**. É especialmente útil quando não tiver hardware tão bom para rodar os modelos localmente, ou não queremos nos preocupar com isso.
- Desse modo, caso queira rodar os projetos em sua máquina local, você pode usar a integração com o ChatGPT da Open AI (mas você pode usar outro serviço de sua preferência que se conecte à uma API, como Google / Gemini ou Anthropic / Claude)

Para modelos open source em ambiente local

- Iremos usar o **Ollama**. Caso procure por uma solução gratuita para execução de modelos open source localmente nós recomendamos o Ollama, altamente indicado devido à maior performance e sua compatibilidade com o LangChain.
- Com o Ollama podemos executar as LLMs de modo mais prático com CPU, não sendo necessário o uso da GPU para modelos menores.

IMPLEMENTAÇÃO EM MÁQUINA LOCAL

Usando modelos ChatGPT (ou outros modelos proprietários)

Precisa antes fazer a instalação usando esse comando no terminal:

```
pip install -qU langchain-openai
```

Quanto ao código, é o mesmo que está no Colab.

Outros modelos proprietários ou via API

Caso queira usar outros serviços que fornecem LLMs, a lógica é a mesma para os outros modelos (caso possua integração direta com o LangChain)

IMPLEMENTAÇÃO EM MÁQUINA LOCAL COM OLLAMA

- **Ollama** é uma ferramenta de código aberto que possibilita a execução, criação e compartilhamento dos LLMs diretamente no seu próprio computador.
- A ferramenta é compatível com sistemas operacionais como macOS, Linux e Windows.
- Sua instalação é facilitada, podendo ser feita de modo prático diretamente pelo site oficial ou através de imagens Docker.
- O Ollama pode ser integrado ao LangChain. A primeiro momento, utilizaremos através da interface do próprio Ollama. E depois usaremos pelo LangChain.



ollama.com

IMPLEMENTAÇÃO EM MÁQUINA LOCAL COM OLLAMA

Instalação do Ollama

- 1) Acessar o site <https://ollama.com/>
 - Clique no botão download e selecione seu sistema operacional
- 2) Processo de instalação - nenhuma configuração necessária, apenas clicar para Prosseguir
- 3) Para rodar e baixar o modelo, execute o seguinte comando:
`ollama run phi3`

Lista de modelos disponíveis para baixar pelo Ollama:

<https://ollama.com/library>

IMPLEMENTAÇÃO EM MÁQUINA LOCAL COM OLLAMA

Testes no Ollama

- Escreva algo como “olá, como vai você?” para rodar seu primeiro exemplo.
- O resultado que você está vendo está sendo executado localmente, ou seja, poderia ser feito sem acesso à internet.

Mais opções

- Digite ``/?`` para mostrar os comandos disponíveis
- Exemplos:
 - `/show info`
 - `/show template`

INTEGRAÇÃO DO OLLAMA COM LANGCHAIN

- 1) Instalação do pacote: `pip install -qU langchain-ollama`
- 2) Baixe o modelo desejado através do comando abaixo, que você deve também executar no terminal:

`ollama pull <nome do modelo>`

Exemplo: `ollama pull llama3`

Observação: caso você tenha baixado e executado anteriormente o comando `ollama run <modelo>` para esse modelo então não precisa executar o comando de pull agora, pois o modelo está salvo localmente.

Mas se estiver usando pela primeira vez esse modelo em sua máquina (ou se ocorrer qualquer erro no carregamento) use o comando **ollama pull** para puxar o modelo do repositório.

Para listar todos os modelos baixados: `ollama list`

Os modelos ficam salvos em:

- Windows: `C:\Users\<usuario>\.ollama\models`
- MacOS: `~/ .ollama/models`
- Linux: `/usr/share/ollama/.ollama/models`

Consultar documentação atualizada caso não encontre nesses locais

INTEGRAÇÃO DO OLLAMA COM LANGCHAIN

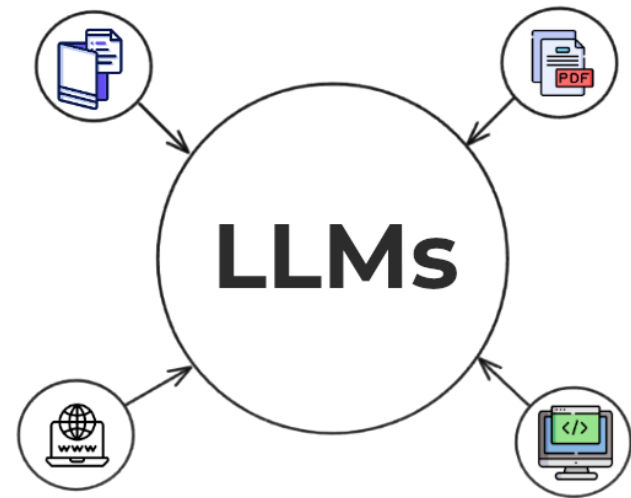
- Após feita a configuração e baixado o modelo, basta importar o método `ChatOllama()`
- Aqui você deve passar o nome do modelo. Os demais parâmetros são opcionais.
- Eles são basicamente os mesmos das outras classes com “Chat” no nome e que já exploramos anteriormente. Mas caso deseje consultar mais detalhes sobre os demais parâmetros, acesse:

https://api.python.langchain.com/en/latest/chat_models/langchain_ollama.chat_models.ChatOllama.html

– *checar o código .py*

RAG

- **Retrieval-Augmented Generation** (RAG) é uma técnica que combina modelos de linguagem com mecanismos de recuperação de informações para melhorar a geração de texto.
- Em vez de confiar apenas nos dados armazenados internamente pelo modelo, RAG recupera informações relevantes de fontes externas, como documentos ou bancos de dados, para fornecer respostas mais precisas e informadas.



PRINCIPAIS DESAFIOS DOS LLMs QUE RAG AJUDA A RESOLVER

- Conhecimento de Domínio (*domain knowledge*) - LLMs podem não ter informações específicas de uma área específica, tornando suas respostas menos precisas.
 - Solução: com o RAG permite a integração de dados específicos de um domínio a partir de fontes externas, melhorando a precisão e a relevância das respostas.
- Alucinações (*hallucinations*) - LLMs às vezes geram respostas factualmente incorretas ou incoerentes, conhecidas como "alucinações".
 - Solução: RAG utiliza sistemas de recuperação para acessar informações verificadas de fontes externas em tempo real, reduzindo a probabilidade de alucinações e aumentando a confiabilidade das respostas.
- Dados de treinamento não recentes (*training data cut off*) - as LLMs têm um conhecimento limitado ao período em que foram treinados, ignorando eventos ou informações recentes.
 - Solução: RAG pode buscar informações atualizadas em bases de dados ou na internet, garantindo que as respostas estejam sempre atualizadas e relevantes.

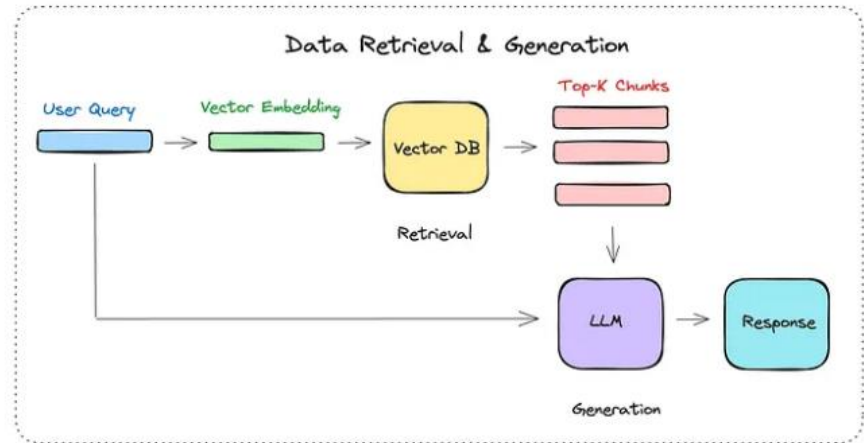
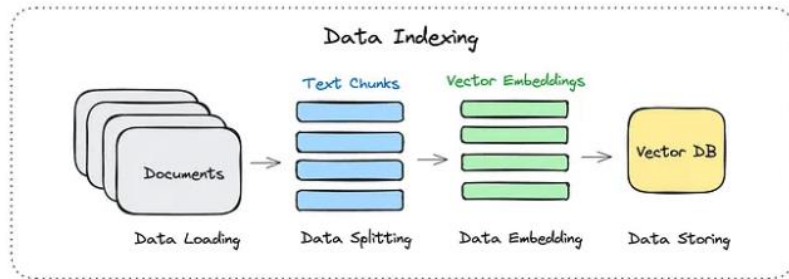
RAG – VANTAGENS

- Acesso a conhecimento atualizado
- Redução de alucinações
- Integração de conhecimento de domínio
- Versatilidade e Inteligência
- Melhora na relevância das respostas
- Capacidade de recuperação dinâmica
- Aprimoramento da compreensão linguística

RAG – EXEMPLOS DE APLICAÇÕES

- Atendimento ao Cliente
- Pesquisa Acadêmica
- Assistentes Jurídicos
- Medicina e Diagnóstico
- Desenvolvimento de Produtos

RAG – PIPELINE BÁSICA



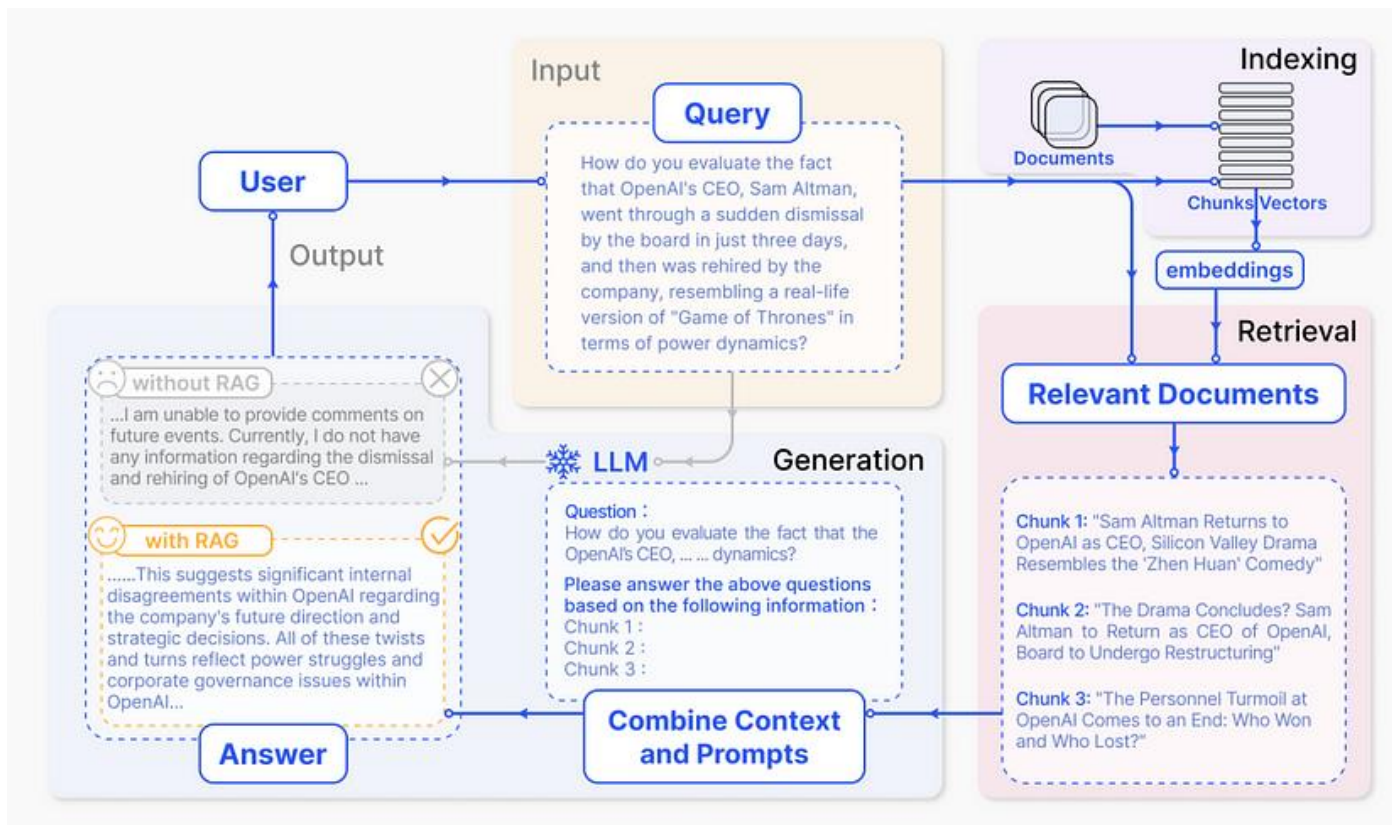
Créditos da imagem: [DrJulija's Notebook](#)

RAG - CONCEITOS E FUNCIONAMENTO

Uma aplicação RAG convencional pode ser dividida em dois processos:

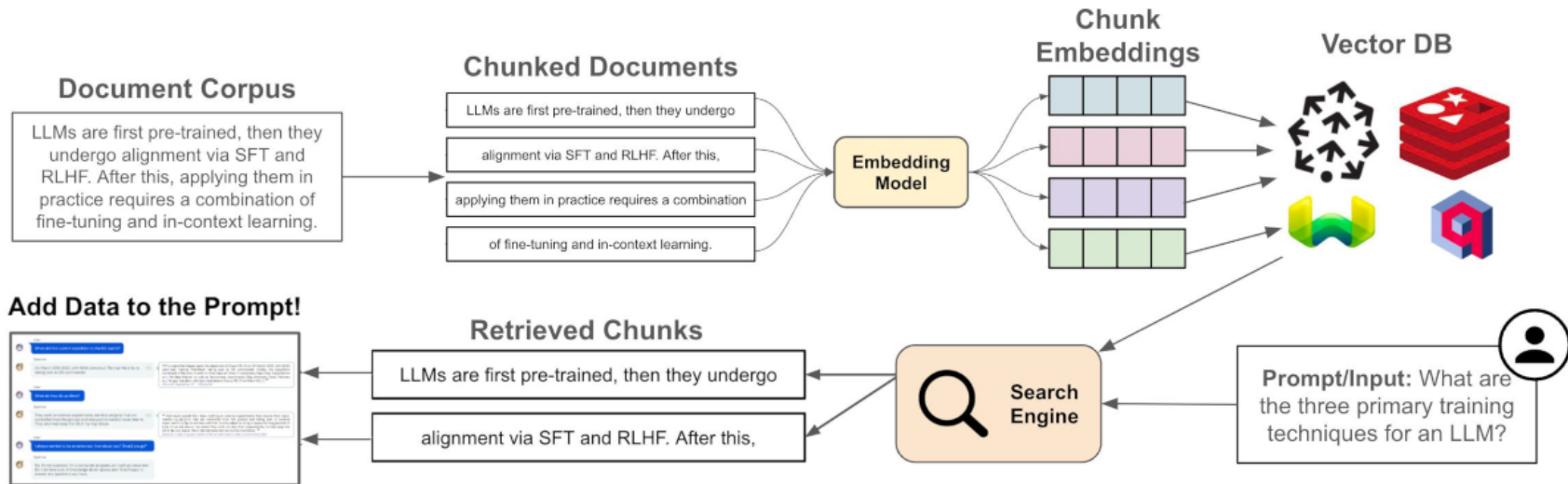
- **Indexação (*Indexing*):**
 - Constitui de um pipeline para ingerir dados de uma fonte e indexá-lo.
 - O método típico é converter todos os dados privados em embeddings armazenados em um banco de dados vetorial.
- **Recuperação e geração (*Retrieval and generation*):**
 - É a chain em si, que processa o pedido do usuário e recupera os dados relevantes do índice, depois passando-os para o modelo usar como contexto.
- O LangChain tem vários componentes projetados para ajudar a criar aplicativos de perguntas e respostas e aplicativos RAG de modo geral.

RAG - PIPELINE



Créditos da imagem: paper "RAG for Large Language Models: A Survey"

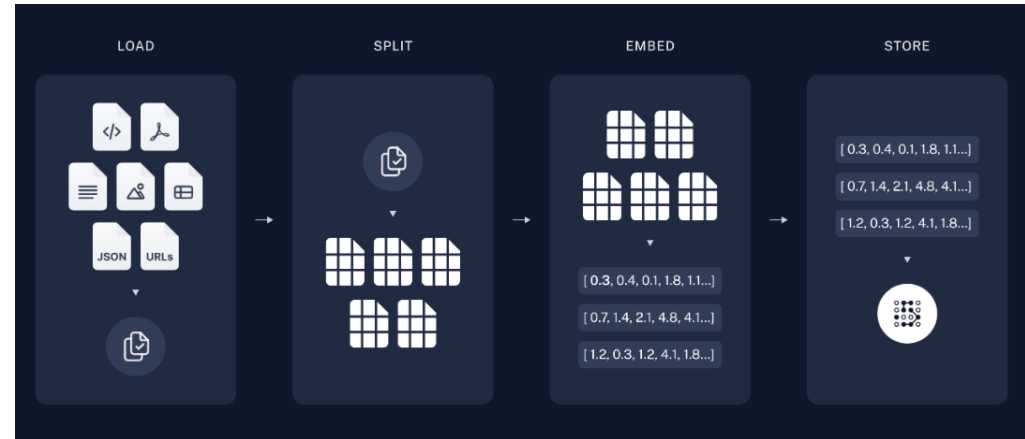
RAG - PIPELINE



Créditos da imagem: A Practitioners Guide to Retrieval Augmented Generation (RAG)

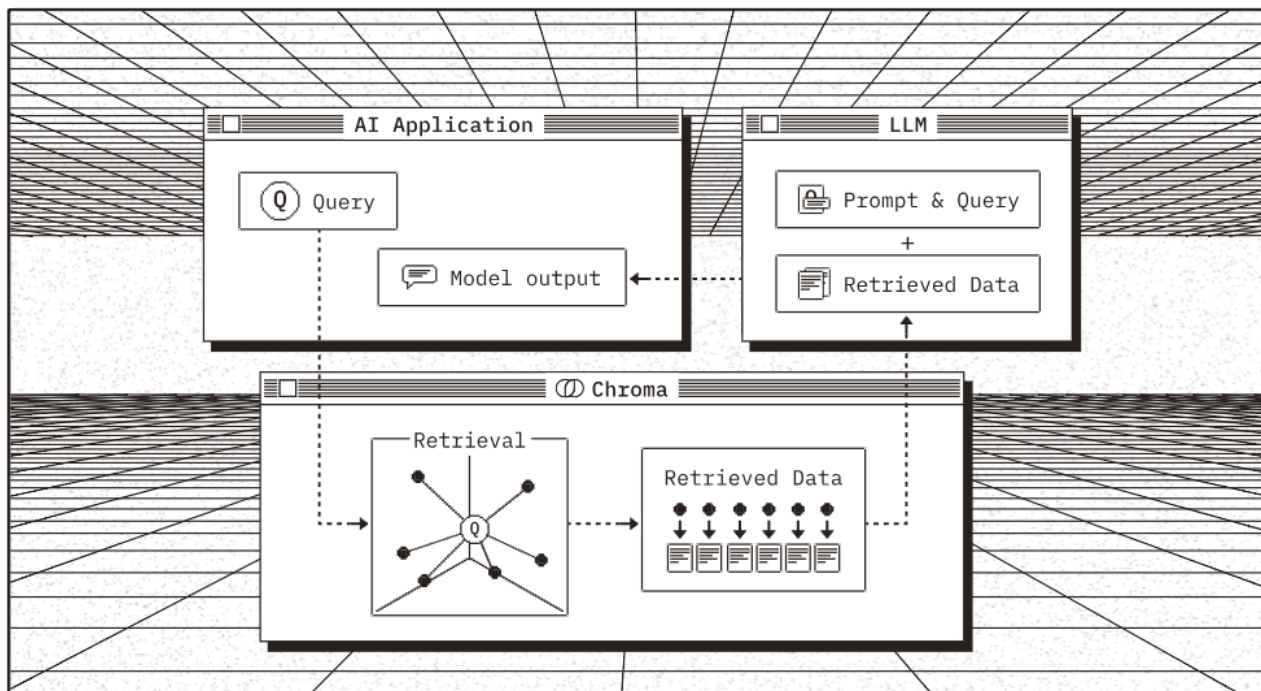
RAG - INDEXAÇÃO

- **1) Carregar:** Primeiramente, precisamos carregar nossos dados. No LangChain temos várias funções (chamados de *Document Loaders*) que facilitam esse processo.
- **2) Dividir os dados em pequenos pedaços:** Divisores de texto quebram grandes documentos em partes menores, facilitando tanto a indexação quanto a passagem ao modelo, pois partes grandes são mais difíceis de pesquisar (por exemplo, livros inteiros ou documentos com centenas de páginas) e portanto não cabem na janela de contexto do modelo.
- **3) Armazenar:** Precisamos de um local para armazenar e indexar esses pedaços, para que possam ser pesquisadas posteriormente. Após o embedding, os dados podem então ser armazenados em um banco de dados vetorial (VectorStore) para recuperação posterior.



BANCO DE DADOS VETORIAL - VISUALIZAÇÃO

Resumo do processo: Converter uma query em embeddings para que depois possam ser recuperadas e serem fornecidas como contexto junto à pergunta, que será passada como entrada para a LLM



Créditos: site do Chroma DB (<https://www.trychroma.com>)

BANCO DE DADOS VETORIAL

Existem várias ferramentas e serviços para vector stores e bancos de dados semânticos

Para um comparativo completo, acesse:

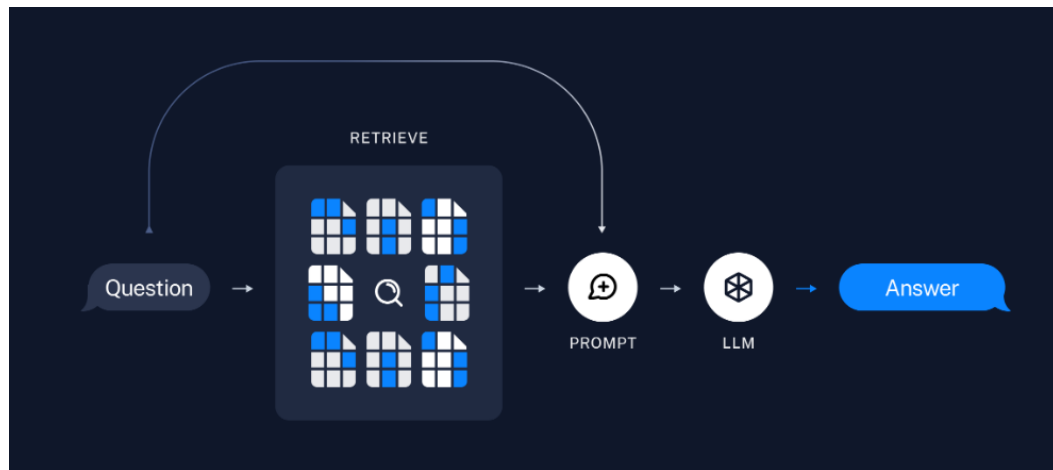
<https://superlinked.com/vector-db-comparison?via=topaitools>

Acessar documentação langchain para ver quais possui integração

<https://python.langchain.com/v0.2/docs/integrations/vectorstores/>

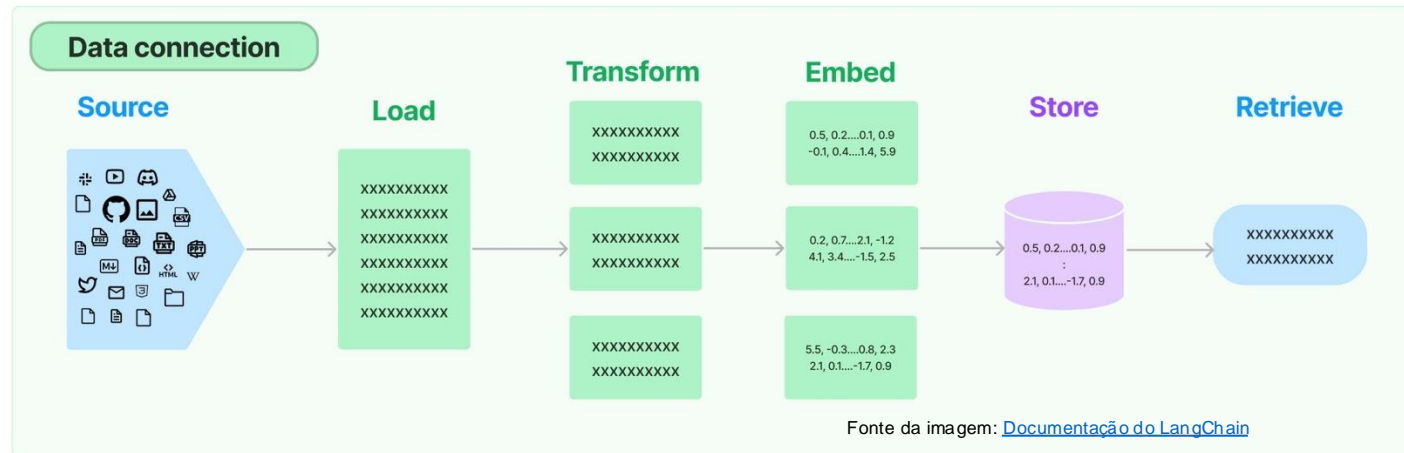
RAG – RECUPERAÇÃO E GERAÇÃO

- **4) Recuperar:** Dada a entrada de um usuário, as divisões relevantes são recuperadas do armazenamento usando um retriever.
- **5) Gerar:** o modelo então produz uma resposta usando um prompt que inclui a pergunta e os dados recuperados.
- O modelo receberá como input a pergunta do usuário + contexto



LANGCHAIN – CONEXÃO DE DADOS E RECUPERAÇÃO

- Dados externos são recuperados e então passados para o LLM ao fazer a etapa de geração.
- Significa que é possível usar LLMs para processar nossos documentos ou outros recursos como páginas na internet.
- O LangChain fornece todos os blocos de construção para aplicativos RAG - do simples ao complexo.

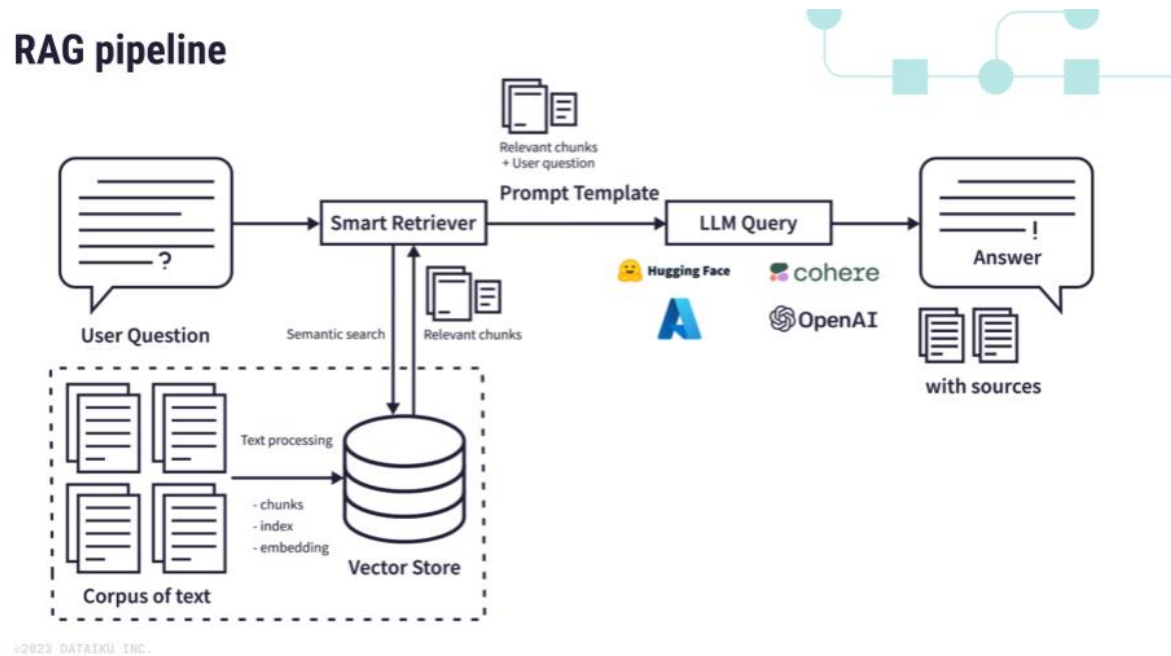


LANGCHAIN – TERMOS E CLASSES PARA RETRIEVAL

- **Document Loaders** - Carregadores de documentos carregam documentos de várias fontes diferentes. O LangChain oferece centenas de carregadores de documentos e integrações com outros grandes provedores, como AirByte e Unstructured.
- **Text Splitting** - Uma parte essencial da recuperação de informações é buscar apenas as partes relevantes dos documentos. O LangChain oferece vários algoritmos de transformação para dividir documentos grandes em partes menores, otimizados para tipos específicos de documentos.
- **Text Embedding Models** - A criação de embeddings para documentos captura o significado semântico do texto, permitindo encontrar rapidamente partes semelhantes. O LangChain integra vários provedores de embeddings, oferecendo uma interface padrão para alternar entre modelos facilmente.
- **Vector Stores** - Com a ascensão dos embeddings, há uma necessidade de bancos de dados para armazená-los e pesquisá-los eficientemente. O LangChain integra mais de 50 serviços diferentes, permitindo escolher o mais adequado às suas necessidades e fornecendo uma interface padrão para alternar entre eles.
- **Retrievers** - Depois que os dados estão no banco de dados, é necessário recuperá-los. O LangChain suporta diversos algoritmos de recuperação, oferecendo métodos simples de busca semântica e uma coleção de algoritmos adicionais para aumentar o desempenho.
- **Indexing** - A API de Indexação do LangChain sincroniza seus dados de qualquer fonte em uma loja de vetores, ajudando a evitar a duplicação de conteúdo, reescrita de conteúdo inalterado e recálculo de embeddings sobre conteúdo inalterado.

RAG – EXEMPLO PRÁTICO

Diagrama de um pipeline de RAG para aplicação de perguntas e respostas



Créditos da imagem: [data iku](#)

RAG – EXEMPLO PRÁTICO

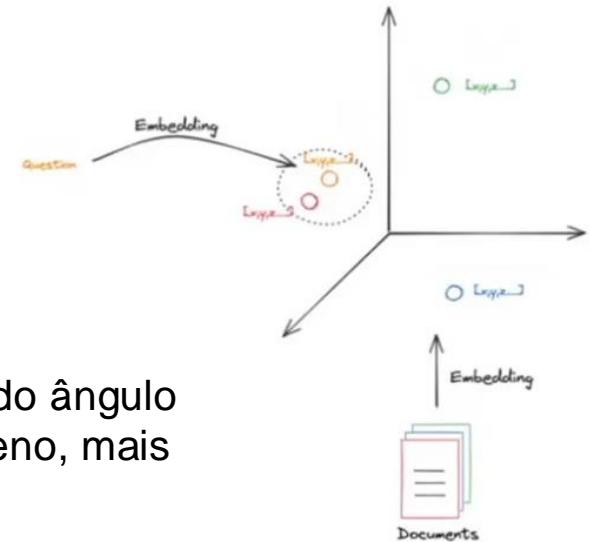
Diagrama de um pipeline de RAG para aplicação de perguntas e respostas

1. Reunir documentos para complementar o conhecimento base de um LLM sobre uma determinada área, como políticas internas, documentos financeiros, documentação técnica ou artigos de pesquisa.
2. Usar um vector store (como FAISS, Pinecone ou ChromaDB) para dividir os dados textuais em partes menores e vetorizá-los, armazenando o resultado em um banco de conhecimento otimizado para busca semântica.
3. Quando um usuário faz uma pergunta, o sistema de recuperação inteligente usa o vector store para complementar a pergunta com informações relevantes.
4. O prompt aumentado é usado para consultar o LLM, fornecendo respostas mais precisas juntamente com suas fontes.

RAG – RETRIEVAL

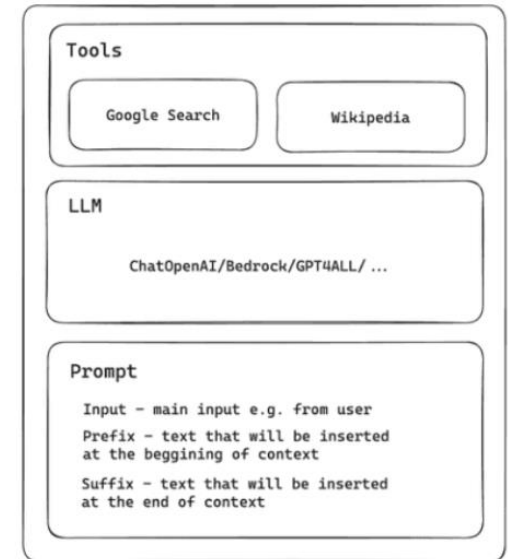
Exemplos de Busca por Similaridade

- Consiste em encontrar documentos ou informações que são semelhantes a uma consulta baseada em suas representações numéricas.
- Busca por Similaridade: Compara os vetores numéricos (embeddings) da consulta e dos documentos para encontrar os mais semelhantes via KNN.
- Similaridade Cosseno: método que mede o cosseno do ângulo entre dois vetores. Quanto maior a similaridade cosseno, mais semelhantes são os vetores.
- Como buscar um livro em uma biblioteca descrevendo seu conteúdo. O sistema encontra livros com conteúdos semelhantes à sua descrição comparando as representações numéricas dos conteúdos dos livros.



AGENTES E TOOLS

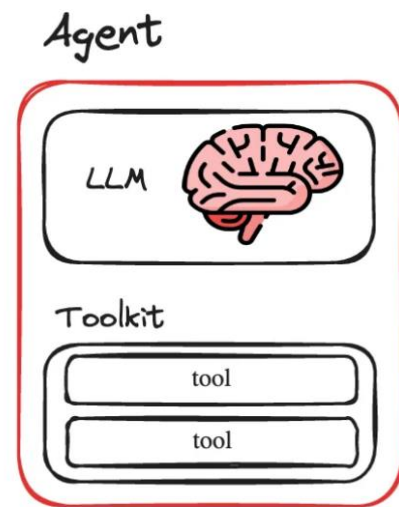
- LLMs encontram dificuldades com determinadas tarefas que envolvem lógica, cálculo matemáticos ou pesquisa.
- A definição de agentes LLM é bastante ampla: são todos os sistemas que utilizam grandes modelos de linguagem como motor e podem realizar ações em seu ambiente com base em observações.
- O agente usa um motor de raciocínio para determinar quais ações tomar para obter um resultado - agente integrado com os mecanismo de pesquisa da Google e a Wikipedia.



Créditos da imagem: [Bright Inventions](#)

AGENTES E TOOLS

- **Tools** (Ferramentas) são os principais componentes de agentes que realizam tarefas individuais.
- As tools são basicamente apenas métodos/classes aos quais o agente tem acesso que podem fazer coisas como: fazer uma pesquisa na web, interagir com um índice do mercado de ações por meio de uma API, atualizar um evento do Google Agenda ou executar uma consulta em um banco de dados.
- Uma coleção de Ferramentas no LangChain é chamada de **Toolkit**.



Créditos da imagem:
[Sami Maameri](#)
- [Towards Data Science](#)

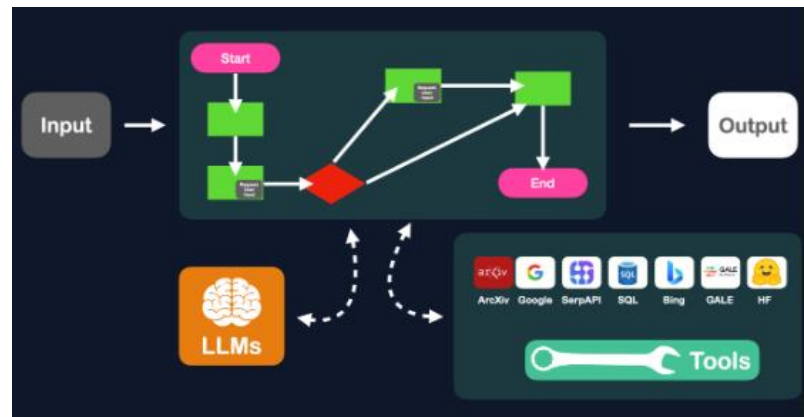
AGENTES VS. CHAINS

Agentes



- Os agentes possuem um nível maior de flexibilidade, autonomia real e capacidade de raciocínio.
- Podem selecionar ferramentas conforme necessário, em uma sequência que considerem adequada.

Chains



Créditos da imagem: [Cobus Greyling](#)

- Podemos ter controle direto sobre a tarefa ou direção da conversa.
- Valioso especialmente em situações onde é crucial ter controle detalhado sobre o fluxo da tarefa ou da conversa.

VANTAGENS DOS AGENTES

- Flexibilidade
- Raciocínio dinâmico
- Manutenção simplificada
- Capacidade de integração
- Produção de ações
- Adequação a tarefas complexas
- Realimentação de resultados

AGENTES – CASOS DE USO

- Negócios
- Saúde
- Atendimento ao cliente
- Geração e gestão de conteúdo
- Tendências emergentes e direções futuras

AGENTES – EXPLORANDO O POTENCIAL

Um panorama geral sobre o estado atual (e futuro) dos agentes e seu potencial

- Você pode encontrar um bom review do panorama geral dos agentes no artigo **The Rise and Potential of Large Language Model Based Agents: A Survey**, por Xi et al., 2023.

< Papers arxiv:2309.07864

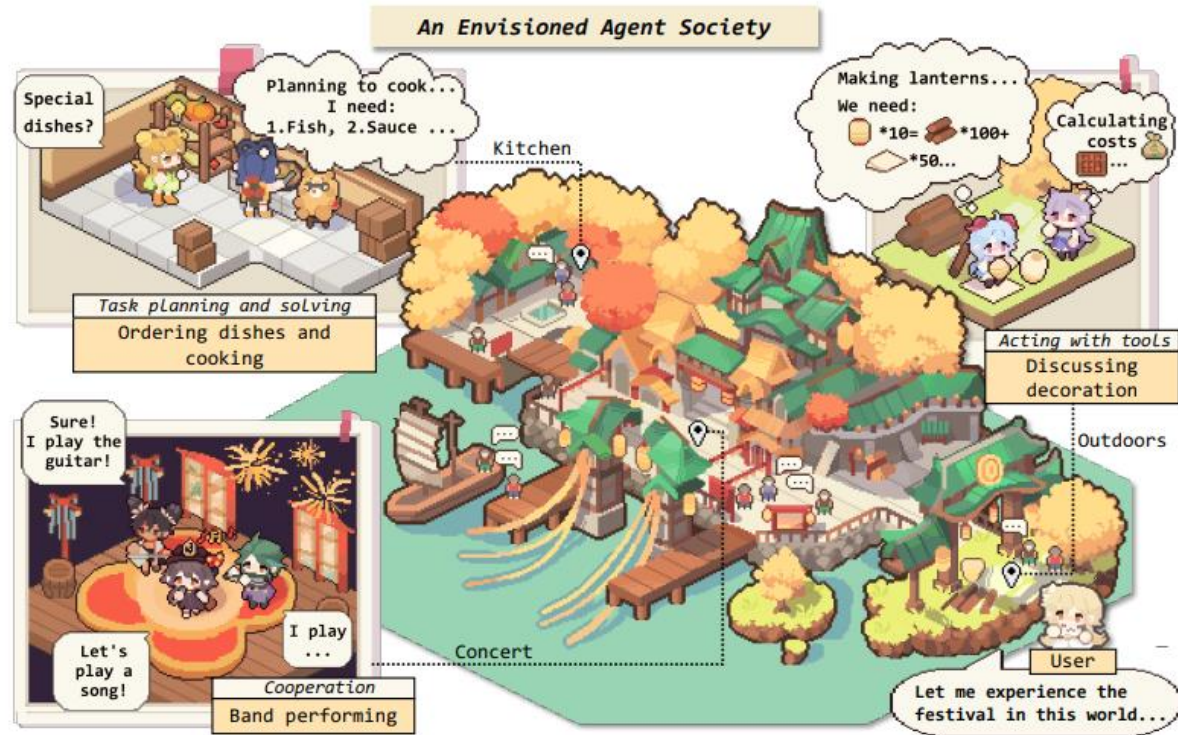
The Rise and Potential of Large Language Model Based Agents: A Survey

Published on Sep 14, 2023

Authors: Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He,  Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Qin Liu, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou + 8 authors

<https://arxiv.org/abs/2309.07864>

AGENTES – EXPLORANDO O POTENCIAL



Créditos: [The Rise and Potential of Large Language Model Based Agents: A Survey](#)

CHAIN-OF-THOUGHT - CADEIA DE PENSAMENTO

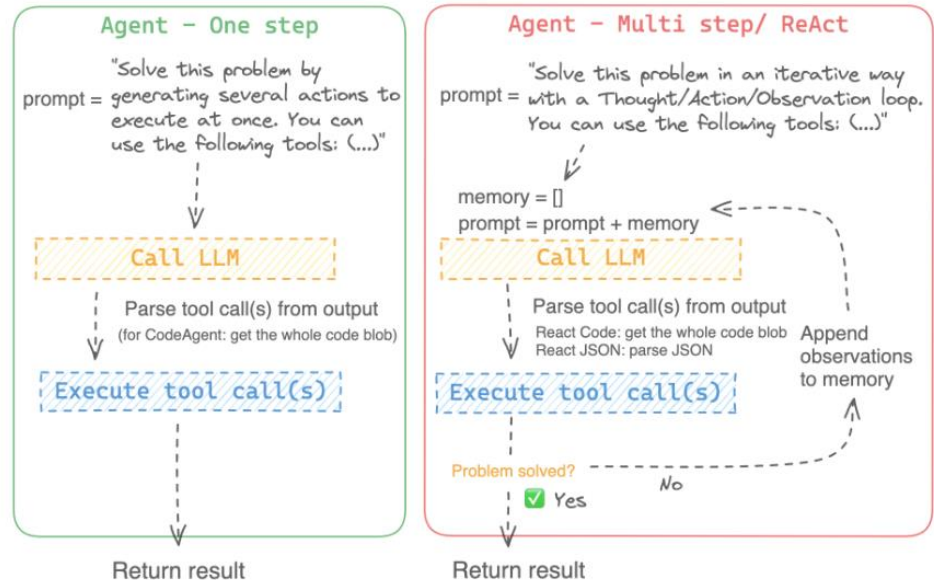
- Uma das maneiras de resolver o problema de raciocínio complexo em LLM é usar o prompting Chain-Of-Thought (cadeia de pensamento).
- O raciocínio complexo pode ser desafiador mesmo para as maiores LLMs hoje, especialmente tarefas que incluem múltiplas etapas.
- Para alcançar sua tarefa, agentes podem usar várias iterações do ciclo de:
Percepção \Rightarrow Reflexão \Rightarrow Ação

Standard Prompting	Chain-of-Thought Prompting
<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>	<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>
<p>Model Output</p> <p>A: The answer is 27. ❌</p>	<p>Model Output</p> <p>A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅</p>

Para saber detalhes mais profundos sobre a teoria, consulte o paper sobre o **Chain-of-Thought Prompting Elicits Reasoning in Large Language Models** (<https://arxiv.org/abs/2201.11903>)

ReAct (REASONING AND ACTIONS)

- O ReAct é uma abordagem para a construção de agentes. É definido com base na concatenação de duas palavras, **raciocínio** ("Reasoning") e **atuação** ("Acting").
- No prompt, descrevemos o modelo, quais ferramentas ele pode usar, e pedimos que ele pense "passo a passo" (também chamado de comportamento de cadeia de pensamento, ou Chain-of-Thought) para planejar e executar suas próximas ações para alcançar a resposta final.



Créditos da imagem: [Aymeric Roucher](#), [Hugging Face](#)

AGENTES – ReAct E CHAIN-OF-THOUGHT

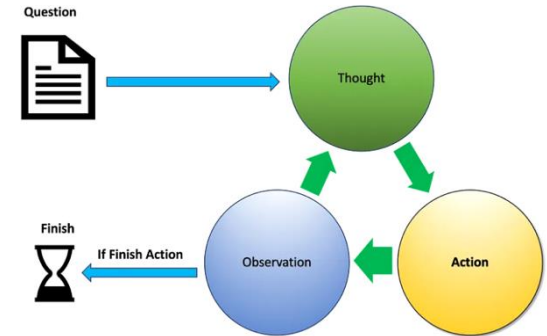
- ReAct permite que LLMs realizem raciocínio e ações específicas para tarefas.
- Combina raciocínio em cadeia (chain of thought) com planejamento de ações, melhorando o desempenho em várias tarefas e superando problemas como alucinações.
- Para resolver perguntas complexas, o LLM adota um processo iterativo.
- Primeiro, o LLM gera um pensamento sobre o problema e identifica uma ação a ser tomada. As ações podem incluir chamadas de API, como buscar dados na Wikipedia.
- O LLM observa os resultados das ações e, se necessário, gera novos pensamentos e ações até encontrar a resposta.

AGENTES E ReAct

Na estrutura ReAct, o LLM pode escolher entre um número limitado de ações que são definidas por um conjunto de instruções que são anexadas ao texto do prompt de perguntas do LLM.

Por exemplo, o artigo do ReAct abrange três espaços de ação. Projeta uma Web API simples da Wikipedia com três tipos de ações para dar suporte à recuperação interativa de informações.

- *search[entity]* que retorna as primeiras 5 frases da página wiki da entidade correspondente, se existir, ou então sugere as 5 principais entidades semelhantes do mecanismo de busca da Wikipedia
- *lookup[string]* que retornaria a próxima frase na página que contém a string, simulando a funcionalidade Ctrl+F no navegador.
- *finish[answer]* que finalizaria a tarefa atual com a resposta



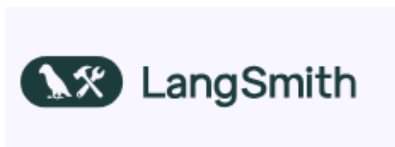
Mais detalhes e teoria aprofundada no paper “ReAct: Synergizing Reasoning and Acting in Language Models”
<https://arxiv.org/pdf/2210.03629>

AGENTES NO LANGCHAIN – POSSÍVEIS LIMITAÇÕES

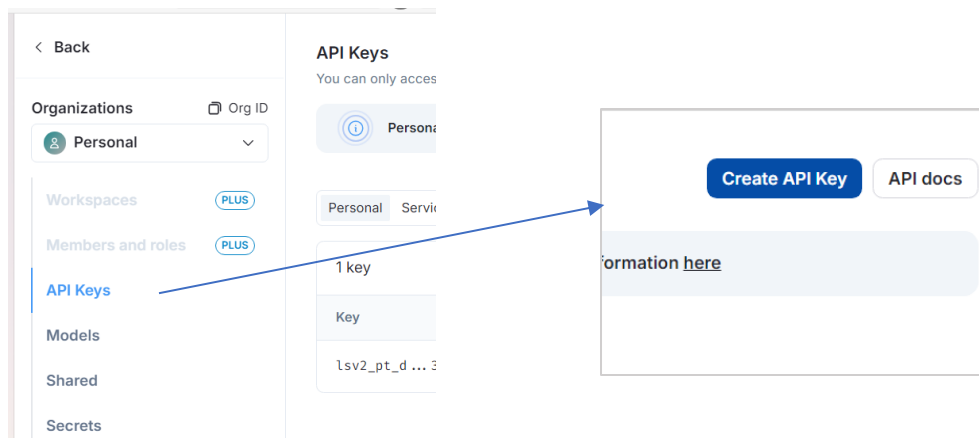
- O LangChain é um framework bastante rico e completo, mas atualmente talvez o seu ponto mais fraco seja a falta de compatibilidade com modelos open source para criação de agentes de IA.
- A documentação muitas vezes se encontra desatualizada e carente de informações, dificultando a resolução de problemas. A comunidade de IA destaca essas limitações, especialmente em relação à clareza e atualidade das informações.
- Agents e tools têm grande potencial de uso, permitindo orquestrar interações complexas entre diferentes ferramentas e LLMs. No entanto, devido à documentação sobre seu funcionamento ainda ser vaga e muitas vezes impossível de produzir (levando a comportamentos inesperados e indesejados), dificulta a compreensão e a implementação eficaz.
- Pode ser que seja resolvido em um futuro breve. Em todo o caso, se permanecer os mesmos comportamentos, avalia-se testar outro framework para Agentes de IA (como **LlamaIndex** ou **Crew AI**), ou criar uma solução manualmente.

LANGSMITH

- Além de fornecer um hub contendo milhares de prompts da comunidade, o LangSmith oferece uma ferramenta de tracing, usada para depuração.
- Ele contém as informações completas de todas as entradas e saídas de cada etapa do aplicação.
- À medida que essas aplicações se tornam cada vez mais complexas, torna-se crucial poder inspecionar exatamente o que está acontecendo dentro da sua cadeia ou agente.

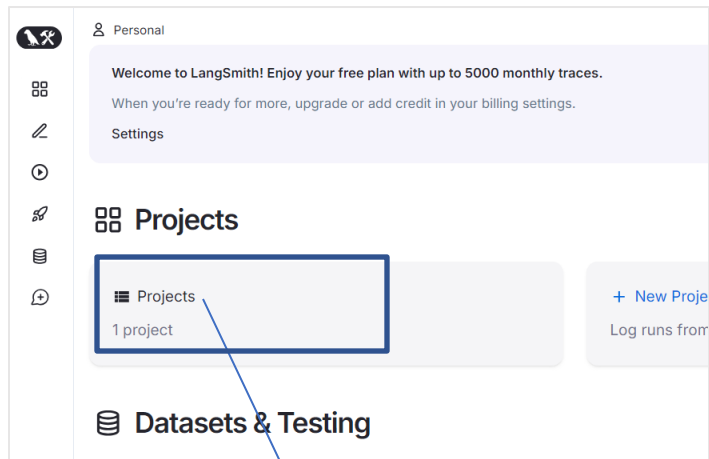


Para gerar um key, acesse:
<https://smith.langchain.com>



LANGSMITH

Para visualizar seus registros, acesse:



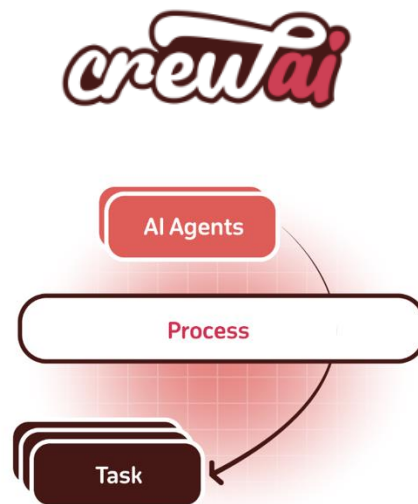
Name ↑↓	Feedback (7D)	Run Count (7D)	Error Rate (7D) ↑↓
default		16	19%

>	✓	AgentExecutor	Onde fica Linz? Qual ...	Linz é a terceira ...
>	✓	AgentExecutor	Qual é o valor de mer...	O valor de merca...
>	✓	AgentExecutor	Qual o valor de merca...	O valor de merca...

The screenshot shows the 'AgentExecutor' trace view. It displays the input, output, and feedback for a specific run. The input is 'input: Onde fica Linz? Qual a população?'. The output is 'Linz é a terceira maior cidade da Áustria, localizada na região da Alta Áustria, com uma população de cerca de 208.000 habitantes.' The feedback is '1'. The trace also shows the execution of various tools like 'PromptTemplate', 'ChatOpenAI', and 'tavily_search_results_json'.

SOLUÇÕES PARA APLICAÇÕES MAIS COMPLEXAS COM AGENTES

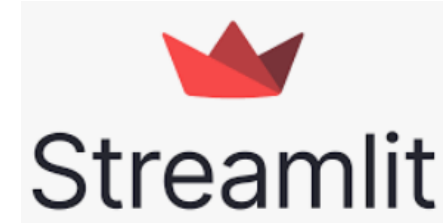
- O CrewAI é um framework de ponta para a orquestração de agentes de IA autônomos que desempenham papéis específicos.
- Ele facilita a inteligência colaborativa, permitindo que os agentes trabalhem juntos de forma fluida e enfrentem tarefas complexas.
- Utilizando o CrewAI, é possível criar plataformas de assistentes inteligentes, equipes automatizadas de atendimento ao cliente ou grupos de pesquisa multiagente, onde cada agente assume um papel e compartilha objetivos em um sistema coeso.
- Além disso, o CrewAI pode ser mais interessante e completo que o LangChain devido à sua melhor integração com modelos open source, oferecendo uma solução robusta e flexível para diversas aplicações de IA.



crewai.com

CRIAÇÃO DE INTERFACES COM STREAMLIT

- O Streamlit é um framework open source que facilita a criação de aplicações web interativas para dados e machine learning com poucas linhas de código Python.
- Ele permite que os desenvolvedores transformem scripts de dados em aplicativos web de forma rápida e eficiente.
- Streamlit é conhecido por sua flexibilidade, simplicidade e capacidade de integrar facilmente com bibliotecas de visualização e machine learning.
- Utilizando o Streamlit, poderemos criar interfaces interativas para nossas aplicações de LLMs de maneira mais rápida e eficiente, acelerando o desenvolvimento e a implementação de nossas soluções.



USO DO STREAMLIT EM AMBIENTE LOCAL

Configurando o Streamlit

Instalação

- use o comando `pip install -q streamlit`

Para inicializar

- use o comando `streamlit run <seu_arquivo>.py`


USO DO STREAMLIT NO COLAB


- Soluções como ngrok e localtunnel são necessárias para expor servidores locais a endereços públicos da internet de maneira segura e rápida, gerando uma URL Pública para seu Localhost.
- Elas são especialmente úteis durante o desenvolvimento de aplicações web, permitindo que os desenvolvedores compartilhem seu trabalho em andamento com colegas ou clientes sem precisar configurar um servidor público.
- O conceito de túnel refere-se ao processo de criar uma conexão segura que encapsula o tráfego de rede entre dois pontos. Em contexto de ngrok e localtunnel, um túnel cria uma ponte entre um servidor local e a internet, permitindo acesso externo ao servidor local.


Para mais informações sobre como usar, acesse o Colab do projeto 1


CHATBOT CUSTOMIZADO COM MEMÓRIA E INTERFACE


Seu assistente virtual

 Olá, sou o seu assistente virtual! Como posso ajudar você?

 qual é o maior planeta do sistema solar?

 Olá! O maior planeta do sistema solar é Júpiter! Com um diâmetro de aproximadamente 142.984 quilômetros, Júpiter é mais de 11 vezes maior que a Terra e tem uma massa de cerca de 318 vezes maior que a da Terra. Além disso, Júpiter é um gás gigante, composto principalmente por hidrogênio e hélio, e é conhecido por suas características únicas, como suas bandas de nuvens e seus grandes olhos (os quais são, na verdade, grandes ciclones).

 e o menor?

 O menor planeta do sistema solar é Mercúrio! Com um diâmetro de aproximadamente 4.879 quilômetros, Mercúrio é o menor planeta do sistema solar e tem uma massa de cerca de 0,38 vezes a da Terra. Mercúrio é um planeta rochoso e tem uma órbita muito próxima ao Sol, o que faz com que sua superfície atinja temperaturas extremas. Além disso, Mercúrio é conhecido por sua velocidade de rotação muito rápida, que faz com que seu período de rotação seja apenas de 59 dias.

Digite sua mensagem aqui...



Testes com Streamlit para geração da interface de nossa aplicação

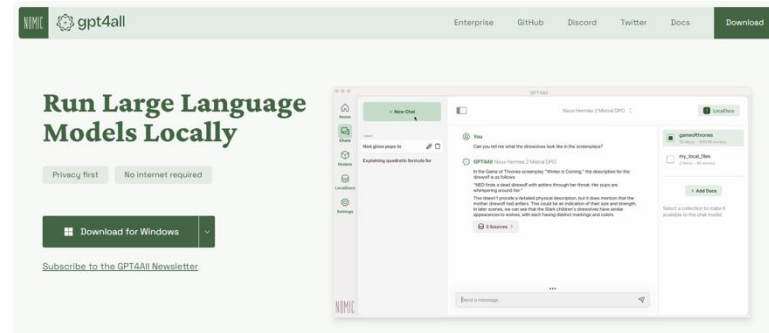
SOLUÇÕES ALTERNATIVAS OPEN SOURCE PRONTAS E INTUITIVAS PARA EXECUÇÃO EM AMBIENTE LOCAL

- Além do Ollama, existem outras soluções que permitem uma implementação otimizada de LLMs em máquinas locais.
- Essas soluções se popularizaram por democratizar o uso dos grandes modelos de linguagem, permitindo rodar inteiramente na sua máquina local com sobrecarga mínima e sem mesmo exigir uma GPU.
- Além disso, muitas dessas soluções já vem com uma interface, o que permite que um público mais leigo consiga fazer seu uso.
- Por exemplo
- Como focamos em uma solução mais “programática” então a escolha de nossa stack apresentada faz mais sentido.
- No entanto, são soluções que podem ser interessantes dependendo do seu caso de uso, ainda mais se a ideia é obter uma inferência rápida e com pouco esforço de configuração.

SOLUÇÕES ALTERNATIVAS OPEN SOURCE PRONTAS E INTUITIVAS PARA EXECUÇÃO EM AMBIENTE LOCAL

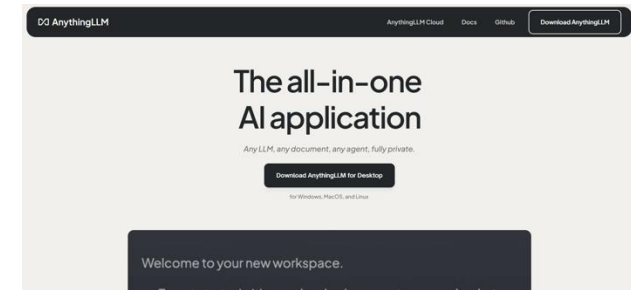
- O GPT4All foi projetado para rodar em PCs modernos ou relativamente modernos sem precisar de conexão com a internet ou mesmo de uma GPU.
- A ideia do GPT4All é fornecer uma plataforma de código aberto e gratuita onde as pessoas possam executar grandes modelos de linguagem em seus computadores.
- Seu uso se popularizou devido à praticidade de uso.
- Oferece interface pronta para uso e bastante intuitiva.
- Possui também integração com o LangChain.

<https://gpt4all.io/index.html>



SOLUÇÕES ALTERNATIVAS OPEN SOURCE PRONTAS E INTUITIVAS PARA USO LOCAL

- AnythingLLM simplifica a integração de vários serviços de IA, como OpenAI, GPT-4 e bancos de dados vetoriais (por exemplo, LangChain, Pinecone e ChromaDB) em um pacote coeso que aumenta a produtividade exponencialmente.
- Um dos seus principais pontos fortes é a capacidade de rodar inteiramente na sua máquina local com sobrecarga mínima, sem exigir GPU.
- Permite transformar qualquer documento, recurso ou parte do conteúdo em contexto que qualquer LLM pode usar como referências durante o chat.
- É um modo prático de implementação de soluções com RAG por exemplo, semelhante ao sistema que desenvolvemos no projeto 2.



<https://anythingllm.com>