

AKCIT -Grandes Modelos de Linguagem

The book cover features a dark background with a wavy pattern of blue and purple dots on the right side. The AKCIT logo is at the top left, and a purple ribbon banner across the middle contains the title 'Grandes Modelos de Linguagem'. Below the title is the author information: 'Autoria: Sávio Salvarino Teles de Oliveira, Daniel Fazzioni, David O'Neil Campos Ferreira'. To the left, there are icons of a screen, a cube, and a brain. The 'Organizadores:' section lists Taciana Novo Kudo, Renata Dutra Braga, Deborah Silva Alves Fernandes, Cristiane Bastos Rocha Ferreira, and Arlindo Rodrigues Galvão Filho. The bottom left corner has the text 'Cegraf UFG'. A circular stamp in the bottom right corner reads 'DISTRIBUIÇÃO GRATUITA' and 'VENDA PROIBIDA'.

AKCIT -Grandes Modelos de Linguagem

Acessível com VLibras

Grandes Modelos de Linguagem

Autoria:
Sávio Salvarino Teles de Oliveira
Daniel Fazzioni
David O'Neil Campos Ferreira

Organizadores:
Taciana Novo Kudo
Renata Dutra Braga
Deborah Silva Alves Fernandes
Cristiane Bastos Rocha Ferreira
Arlindo Rodrigues Galvão Filho

Cegraf UFG

DISTRIBUIÇÃO GRATUITA
VENDA PROIBIDA



Universidade Federal de Goiás

Reitora

Angelita Pereira de Lima

Vice-Reitor

Jesiel Freitas Carvalho

Diretora do Cegraf UFC

Maria Lucia Kons

Conselho Editorial da Coleção Formação no AKCIT

Anderson da Silva Soares

Arlindo Rodrigues Galvão Filho

Deborah Silva Alves Fernandes

Juliana Pereira de Souza Zinader

Renata Dutra Braga

Taciana Novo Kudo

Telma Woerle de Lima Soares

Equipe de produção:

Amanda Souza Vitor

Ana Laura Sene Amâncio Zara

Ana Luísa Silva Gonçalves

Caio Barbosa Dias

Daiane Souza Vitor

Dandra Alves de Souza

Davi Oliveira Gomes

Guilherme Correia Dutra

Iuri Vaz Miranda

Isadora Yasmim da Silva

Júlia de Souza Nascimento

Layne Grazielle Souza Dias

Luciana Dantas Soares Alves

Luis Felipe Ferreira Silva

Luiza de Oliveira Costa

Luma Wanderley de Oliveira

Pedro Vitor Silveira Fajardo

Suse Barbosa Castilho

Vinícius Pereira Espíndola

Wagner Wilson Furtado

Wanderley de Souza Alencar



Grandes Modelos de Linguagem

Autoria:

Sávio Salvarino Teles de Oliveira
Daniel Fazzioni
David O'Neil Campos Ferreira



Organizadores:

Taciana Novo Kudo
Renata Dutra Braga
Deborah Silva Alves Fernandes
Cristiane Bastos Rocha Ferreira
Arlindo Rodrigues Galvão Filho

Cegraf UFG

2024

Grandes Modelos de Linguagem

Instituições responsáveis

Universidade Federal de Goiás (UFG)

Centro de Competência Embrapii em Tecnologias Imersivas, denominado AKCIT (Advanced Knowledge Center for Immersive Technologies)

Centro de Excelência em Inteligência Artificial (CEIA)

Instituições financiadoras

Empresa Brasileira de Pesquisa e Inovação Industrial (Embrapii)

Governo do Estado de Goiás

Empresas parceiras do AKCIT

Apoio

Universidade Federal de Goiás (UFG)

Pró-Reitoria de Pesquisa e Inovação (PRPI-UFG)

Instituto de Informática (INF-UFG)



LISTA DE ABREVIATURAS E SIGLAS

APIs *Application Programming Interface* - Interface de Programação de Aplicações

AWS *Amazon Web Services* - Serviços Web da Amazon

BERT *Bidirectional Encoder Representations from Transformers* - Representações de Codificador Bidirecional de Transformadores



CNN *Convolutional Neural Networks* - Redes Neurais Convolucionais

CPU *Central Processing Unit* - Unidade Central de Processamento

ELMo *Embeddings from Language Model* - Incorporações do Modelo de Linguagem

FLOPs *Floating Point Operations per Second* - Operações de Ponto Flutuante por Segundo

FTaaS *Fine-Tuning-as-a-Service* - Ajuste Fino como Serviço

GPT *Generative Pre-trained Transformer* - Transformador
Generativo Pré-treinado

GPU *Graphics Processing Units* - Unidades de Processamento Gráfico

HMM *Hidden Markov Models* - Modelos Ocultos de Markov

IAG Inteligência Artificial Generativa

ICL *In-Context Learning* - Aprendizado In-contexto

LLMs *Large Language Models* - Grandes Modelos de Linguagem

LMs *Language Models* - Modelos de Linguagem

LoRA *Low-Rank Adaptation* - Adaptação de Baixa Dimensão



LSTM *Long Short-Term Memory* - Memória de Longo e Curto Prazo

MLN *Neural Language Models* - Modelos de Linguagem Neurais

MLP *Probabilistic Language Models* - Modelos de Linguagem Probabilísticos

MLOps *Machine Learning Operations* - Operações de Aprendizado de Máquina

MP *Multilayer Perceptron* - Perceptron Multicamadas

PLN Processamento de Linguagem Natural

RAG *Retrieval Augmentation Generation* - Recuperação e Geração com Aumento

RAM *Random Access Memory - Memória de Acesso Aleatório*

RNN *Recurrent Neural Networks - Redes Neurais Recorrentes*

TPUs *Tensor Processing Units - Unidades de Processamento Tensor*

UFG Universidade Federal de Goiás

VRAM *Video Random Access Memory - Memória de Acesso Aleatório para Vídeo*

LISTA DE EQUAÇÕES, FIGURAS E TABELAS

Equação 1 - Memória necessária para rodar um *Large Language Model*



Equação 2 - Exemplo do Llama 70B com precisão de 16 bits

Equação 3 - Quantização do Llama 70B com precisão de 4 bits

Figura 1 - Mapa mental que ilustra a evolução e a taxonomia dos modelos de linguagem

Figura 2 - Evolução da capacidade de resolução de tarefas em quatro gerações de Modelos de Linguagem

Figura 3 - Comparação entre a abordagem usual de treinamento de modelos

Figura 4 - Tendências dos números cumulativos de artigos do arXiv que contêm as frases-chave "*language model*" (desde junho de 2018) (a) e "*large language model*" (desde outubro de 2019) (b)

Figura 5 - Tarefa de completar textos utilizando grandes modelos de linguagem baseados em arquitetura *Transformer*

Figura 6 - Exemplo de aplicação do parâmetro *Top-k* ($k=2$) na geração de texto

Figura 7 - Exemplo de aplicação do parâmetro *Top-p* ($p=0,95$) na geração de texto

Figura 8 - Impacto da temperatura na distribuição de probabilidades em modelos de linguagem

Figura 9 - Distribuição de probabilidade em função da temperatura

Figura 10 - Treinamento de um grande modelo de linguagem como *Transformer*

Figura 11 - Ciclo de *design* e aplicação de IA Generativa



Figura 12 - Exemplo de *prompt*

Figura 13 - Exemplo de *prompt* sem formatação

Figura 14 - Exemplo de *prompt* com formatação

Figura 15 - *Prompt* incorreto

Figura 16 - *Prompt* correto

Figura 17 - Exemplos de *prompts* correto e incorreto

Figura 18 - Avaliação da qualidade de saída do sistema

Figura 19 - Principais componentes da estratégia de testes em engenharia de *prompts*

Figura 20 - Criação de um *prompt* para geração de resumos de textos longos

Figura 21 - Desenvolvimento de um assistente virtual para responder perguntas complexas

Figura 22 - Ilustração do aprendizado em contexto, onde *Large Language Models* usam *prompts* com exemplos para fazer previsões

Figura 23 - Cenário de exemplo de *few-shot learning*

Figura 24 - Estágios de desenvolvimento de um *Large Language Model*

Figura 25 - Ajuste fino de um grande modelo de linguagem utilizando bases de dados privadas (conjunto de dados de um problema real)

Figura 26 - Reparametrização utilizando a Técnica *Low-Rank Adaptation* (LoRA)



Figura 27 - Comparativo da quantidade de parâmetros treináveis por camada no *fine-tuning* do Llama 3.1 na versão com 3 bilhões de parâmetros, utilizando *Low-Rank Adaptation* (LoRA) com rank 128

Figura 28 - Vantagens e desafios do *fine-tuning*

Figura 29 - Visão geral de como funciona o *Retrieval Augmentation Generation* (RAG) com três etapas

Figura 30 - Exemplo de cenário com uso de *Retrieval Augmentation Generation* (RAG)

Figura 31 - Comparações entre os três paradigmas do *Retrieval Augmentation Generation* (RAG)

Figura 32 - Arquitetura simplificada de memória da *Central Processing Units* (CPU) e *Graphics Processing Units* (GPU)

Figura 33 - Quatro abordagens para colocar *Large Language Models* em produção (AWS Generative AI Summit)

Figura 34 - Matriz de otimização da acurácia de *Large Language Models*

Figura 35 - Boas práticas para garantir que o *Large Language Model* funcione de maneira eficaz

Tabela 1 - Comparativo entre respostas do modelo *Base* com o modelo *Instruct*

SUMÁRIO

Apresentação

Unidade I - Modelos de Linguagem e Grandes Modelos de Linguagem



1.1 Modelos de Linguagem	10
1.1.1 Conexão entre os Modelos de Linguagem	11
1.1.2 Modelos de Linguagem Probabilísticos	14
1.1.2.1 Aplicações	15
1.1.3 Modelos de Linguagem Neurais	16
1.1.3.1 Aplicações	17
1.1.4 Modelos de Linguagem Pré-Treinados	17
1.1.4.1 Aplicações	20
1.2 Grandes Modelos de Linguagem	20
1.2.1 Grandes Modelos de Linguagem com <i>Transformers</i>	21
1.2.2 Geração de Texto por Amostragem em Arquiteturas <i>Transformer</i>	24
1.2.2.1 Amostragem Top-k	25
1.2.2.2 Amostragem <i>Nucleus</i> ou Top-p	27
1.2.2.3 Temperatura	29
1.3 Treinamento de Grandes Modelos de Linguagem	34
1.3.1 Relação entre Número de <i>Tokens</i> , Tamanho do Modelo e Custo de Treinamento	37

1.3.2 Corpora de Treinamento para Grandes Modelos de Linguagem	39
1.4 Desafios dos Grandes Modelos de Linguagem	40
Unidade II - Engenharia de Prompts	45
2.1 Introdução à Engenharia de <i>Prompts</i>	45
2.2 Princípios de Engenharia de <i>Prompts</i>	49
2.2.1 Forneça a Direção	49
2.2.2 Especifique o Formato	50
2.2.3 Forneça Exemplos	53
2.2.3.1 Diversidade de Casos	53
2.2.3.2 Detalhe e Clareza	54
2.2.4 Avalie a Qualidade	55
2.2.5 Faça a Divisão do Trabalho	60
2.2.5.1 Exemplos de Aplicação da Divisão do Trabalho	60
2.2.5.2 Por que Dividir o Trabalho Ajuda a IA?	62
Unidade III - <i>In-context Learning</i> e <i>Fine-tuning</i> em Large Language Models	66
3.1 <i>In-context Learning</i>	66
3.1.1 Desafios e oportunidades do <i>In-context Learning</i>	69
3.2 <i>Fine-tuning</i> em Large Language Models	70
3.2.1 Métodos de <i>Fine-tuning</i>	75
3.2.2 Vantagens e Desvantagens do <i>Fine-tuning</i>	
Unidade IV - Retrieval Augmentation Generation	81
4.1 Como Funciona o <i>Retrieval Augmentation Generation</i> ?	81
4.2 Arquiteturas <i>Retrieval Augmentation Generation</i>	83
4.2.1 Arquitetura <i>Naive</i>	85
4.2.2 Arquitetura Avançada	86
4.2.3 Arquitetura Modular	86
4.3 <i>Retrieval Augmented Generation</i> vs. <i>Fine-tuning</i>	87
4.4 Benefícios do <i>Retrieval Augmented Generation</i>	88
4.5 Desafios e Limitações	88
Unidade V - Integração e Escalabilidade de Soluções com Large Language Models	93
5.1 Escalabilidade de Soluções Large Language Models	93
5.2 Capacidade Computacional para Adoção de Large Language Models nas Empresas	94
5.2.1 Métricas de Capacidade Computacional para Large Language Models	94
5.2.2 Implantação do Large Language Model	96
5.2.3 Requisitos de GPU para Implantação de um Large Language Model em uma Infraestrutura Local	99



5.3 Otimização de Latência e Acurácia	101
5.3.1 Otimização de Latência dos <i>Large Language Models</i>	101
5.3.2 Otimização da Acurácia dos <i>Large Language Models</i>	103
5.4 Integração de Soluções <i>Large Language Models</i>	106
5.5 Disponibilizar Soluções <i>Large Language Models</i> para Usuários Finais	107
5.6 Estudos de Caso e Exemplos Práticos	107
5.6.1 Chatbots para Atendimento ao Cliente	108
5.6.2 Personalização de Conteúdo com <i>Large Language Models</i>	108
5.6.3 Tradução Automática	109
Unidade VI - Encerramento	112
6.1 Sugestões para Ação Futura e Leitura Adicional	113
6.2 Leitura Adicional	114
6.3 Reflexões Finais	114
Referências	115

APRESENTAÇÃO



Prezado(a) Participante,

Seja bem-vindo(a) ao Microcurso Grandes Modelos de Linguagem. Este Microcurso faz parte da Coleção Formação e Capacitação do Centro de Competências Imersivas, uma parceria entre a Embrapii e a Universidade Federal de Goiás (UFG).

A oferta deste Microcurso é motivada pela crescente importância dos Grandes Modelos de Linguagem (*Large Language Models [LLMs]*) no cenário da Inteligência Artificial (IA) e suas diversas aplicações em áreas como Processamento de Linguagem Natural (PLN), desenvolvimento de chatbots, tradução automática e muito mais.

O objetivo deste Microcurso é fornecer a você uma base sobre o funcionamento dos LLMs, suas capacidades e limitações. Abordaremos desde os conceitos básicos, como arquiteturas de modelos e técnicas de treinamento, até tópicos mais avançados, como a construção de *prompts* eficazes, a integração de LLMs em sistemas existentes e as questões de escalabilidade.

Você terá a oportunidade de aprender com exemplos práticos, exercícios e *notebooks* interativos, que irão auxiliar a dominar as ferramentas e técnicas para utilizar LLMs em seus próprios projetos.

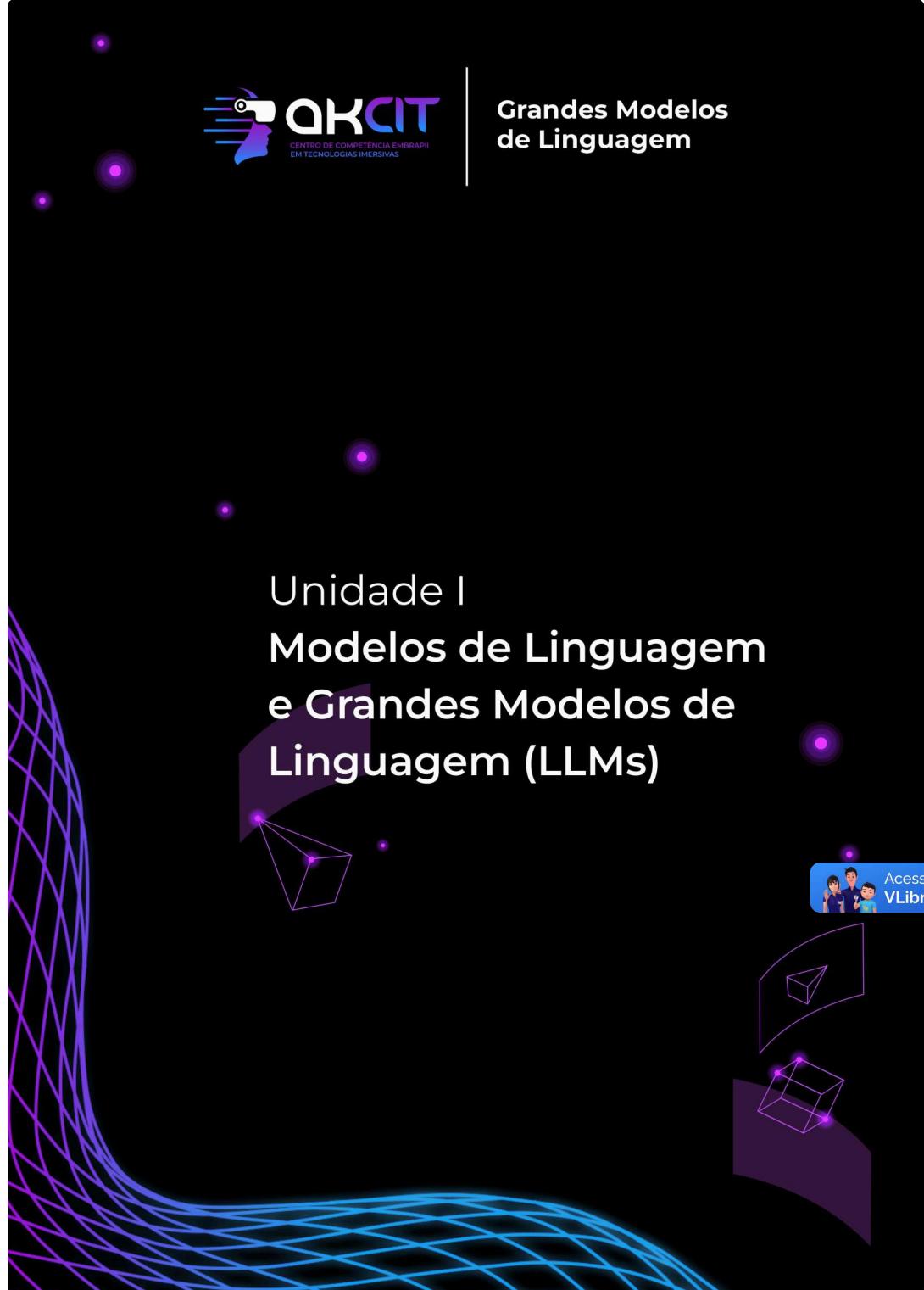
Desejamos um excelente estudo!





Grandes Modelos
de Linguagem

Unidade I
**Modelos de Linguagem
e Grandes Modelos de
Linguagem (LLMs)**



UNIDADE I - MODELOS DE LINGUAGEM E GRANDES MODELOS DE LINGUAGEM

1.1 MODELOS DE LINGUAGEM

A tarefa de prever o futuro é muito complicada. Mas e prever algo que pareça mais fácil, como as próximas palavras que alguém dirá? Qual palavra, por exemplo, provavelmente seguirá a frase "Por favor, entreguem seus trabalhos de ..."? Talvez, a maioria de vocês concluiu que uma palavra muito provável é "casa", mas não "geladeira".

Nesta seção, formalizamos essa intuição apresentando modelos que atribuem uma probabilidade a cada próxima palavra possível. Modelos que atribuem probabilidades a palavras ou sequências de palavras futuras são chamados de modelos de linguagem ou *Language Models* (LMs). Por que desejaríamos prever as próximas palavras? Os LMs são treinados apenas prevendo palavras! Como veremos ao longo deste livro, eles aprendem muito sobre linguagem apenas sendo treinados para prever as próximas palavras a partir de palavras vizinhas.

Os LMs também podem calcular a probabilidade de uma frase inteira. Por exemplo, a frase "O gato deitou-se sobre o tapete macio" possui uma probabilidade muito maior de aparecer em um texto do que a mesma sequência de palavras em uma ordem diferente: "Macio tapete o gato deitou-se sobre".

Por que importa qual a probabilidade de uma frase ou quão provável é a próxima palavra? Em muitas aplicações de PLN, podemos usar a probabilidade como uma forma de escolher uma frase ou palavra melhor em vez de uma menos apropriada. Por exemplo, um sistema de tradução automática que precisa escolher entre "O gato sentou sobre o tapete" e "O gato assentou sobre o tapete". A primeira frase, sendo mais comum e natural em português, terá uma probabilidade maior, guiando o sistema para a tradução correta. Da mesma forma, em um corretor ortográfico, a palavra "cachorro" terá uma probabilidade muito maior do que "caxorro", permitindo a correção automática do erro. A análise de probabilidade também é aplicada em assistentes virtuais, prevendo a próxima palavra mais provável em uma frase e oferecendo sugestões ao usuário durante a digitação.

Em termos mais técnicos, um LLM é um algoritmo treinado em um conjunto massivo de dados textuais para prever e gerar texto. Ele pode ser visto como um super-leitor que, após ler uma grande quantidade de documentos textuais, pode adivinhar a próxima palavra de uma frase ou até escrever um texto inteiro com base em algumas palavras-chave. Mas o que torna os LMs tão importantes? É a capacidade de comunicar e



gerar linguagem similar à humana. No nosso dia a dia, os LMs já estão presentes em:

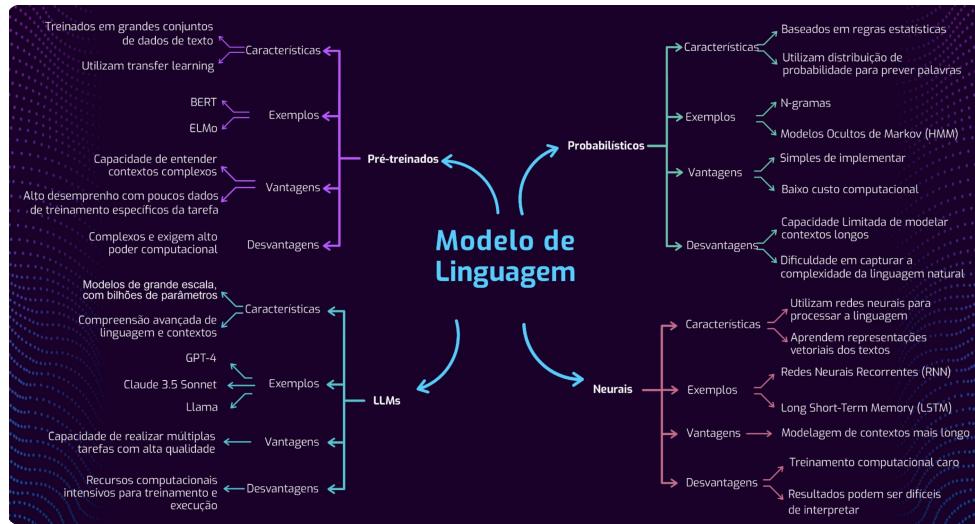
- **Assistentes virtuais:** Siri®, Alexa® e Google Assistente® utilizam LMs para entender comandos de voz e responder às perguntas dos usuários.
- **Tradutores automáticos:** plataformas como o Google Tradutor® usam LMs para traduzir textos de forma cada vez mais precisa e natural.
- **Preenchimento automático e sugestões de texto:** ao escrever um email ou mensagem, os LMs predizem a próxima palavra, corrigem erros ortográficos e até sugerem respostas completas.
- **Geração de conteúdo:** blogs e sites podem ser, em parte, escritos por LMs, que criam textos criativos e informativos.
- **Análise de sentimentos:** empresas utilizam LMs para analisarem comentários e menções nas redes sociais, identificando a opinião do público sobre seus produtos e serviços.

1.1.1 CONEXÃO ENTRE OS MODELOS DE LINGUAGEM

Para melhor compreender a evolução e a diversidade desses modelos, na Figura 1 é apresentado um mapa mental que funciona como um guia visual. Nele, exploramos desde os modelos mais básicos, como os N-gramas, até o estado da arte com os LLMs, passando pelos modelos probabilísticos e modelos pré-treinados, destacando suas características distintivas e funcionalidades. No mapa mental, são organizados diferentes tipos de LMs de forma hierárquica, classificando-os em quatro principais categorias: modelos probabilísticos, modelos neurais, modelos pré-treinados e LLMs.



Figura 1 - Mapa mental que ilustra a evolução e a taxonomia dos modelos de linguagem



Fonte: autoria própria.

Modelos de Linguagem Probabilísticos (MLPs) são caracterizados por serem baseados em regras estatísticas, utilizando distribuições de probabilidades para prever palavras em uma sequência de texto. Exemplos incluem N-gramas e Modelos Ocultos de Markov (HMM). Esses modelos têm como vantagem a simplicidade de implementação e baixo custo computacional, mas possuem limitações na modelagem de contextos longos e na captura da complexidade da linguagem natural.

Modelos Neurais utilizam redes neurais para processar a linguagem, aprendendo representações vetoriais dos textos. Alguns exemplos são as Redes Neurais Recorrentes (RNN) e Long Short-Term Memory (LSTM). Esses modelos têm a capacidade de modelar contextos mais longos, o que melhora sua capacidade de entender a linguagem. Entretanto, seu treinamento é computacionalmente caro e os resultados podem ser difíceis de interpretar.

Modelos Pré-treinados são treinados em grandes conjuntos de dados de texto e utilizam técnicas de transferência de aprendizado (*transfer learning*). Exemplos incluem *Bidirectional Encoder Representations from Transformers* (BERT) e *Embeddings from Language Model* (ELMo), que são conhecidos por conseguir processar contextos complexos e pelo alto desempenho em tarefas específicas com poucos dados de treinamento adicionais. Esses modelos, entretanto, são complexos e exigem alto poder computacional.

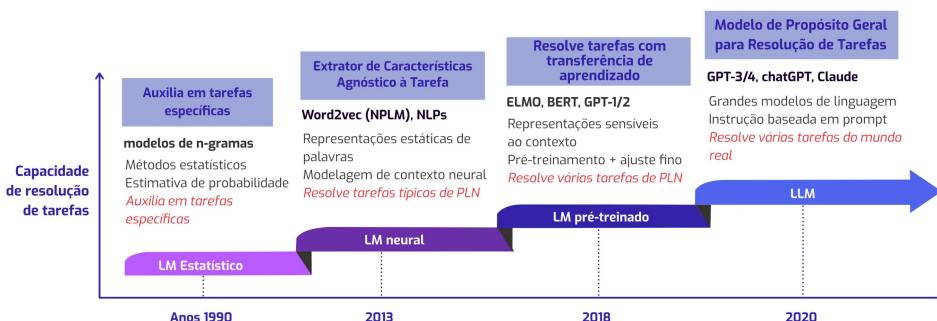
LLMs são caracterizados por sua escala massiva, com bilhões de parâmetros, o que lhes confere uma compreensão avançada de linguagem e contextos. O GPT-4, Claude 3.5, Mistral e Llama são exemplos de famílias de modelos que são classificados como LLMs. Esses modelos têm a capacidade de realizar múltiplas tarefas com alta qualidade, como geração de texto, tradução automática e sumarização de texto. Contudo, sua desvantagem reside no alto custo computacional.

necessário tanto para o treinamento quanto para a execução do modelo em produção. Um estudo revelou que o ChatGPT pode consumir 1 litro de água a cada 100 palavras geradas pela ferramenta. A água é utilizada como um mecanismo para resfriar os supercomputadores que rodam o GPT (Li *et al.*, 2023).

Na Figura 2, é apresentada a evolução dos LMs ao longo do tempo, desde os anos 1990 até os dias atuais. No início, os modelos eram baseados em métodos estatísticos, como os modelos *n-gram*, que serviam como auxiliares em tarefas específicas, realizando estimativas de probabilidade para resolver problemas limitados. Esses modelos possuíam uma capacidade de resolução de tarefas muito limitada, sendo utilizados apenas em tarefas específicas dentro do domínio da linguagem.

Com o avanço da tecnologia, surgiram modelos mais avançados, como o Word2Vec e outros modelos baseados em redes neurais, que foram categorizados como modelos de linguagem neurais (MLN). Esses modelos eram capazes de aprender representações de palavras de maneira estática e realizar modelagem de contexto neural, permitindo a resolução de tarefas típicas de PLN de forma mais generalizada. Mais tarde, em 2018, a introdução dos modelos pré-treinados, como ELMo, BERT, e as primeiras versões do GPT, trouxe uma mudança r  permitir a transferência de aprendizado entre diferentes tarefas de PLN, resolvendo uma variedade de tarefas com o uso de pré-treinamento e *fine-tuning*. Finalmente, a evolução dos modelos resultou na criação dos LLMs, como o GPT-4, que são capazes de resolver diversas tarefas do mundo real, utilizando escalabilidade e *prompts* para completar tarefas complexas.

Figura 2 - Evolução da capacidade de resolução de tarefas em quatro gerações de Modelos de Linguagem



Fonte: autoria própria.

1.1.2 MODELOS DE LINGUAGEM PROBABILÍSTICOS

Antes do surgimento das redes neurais profundas e da aprendizagem profunda, o PLN se apoiava em MLP. Suponha que você esteja tentando

adivinhar a próxima palavra em uma frase. O que você faria? Provavelmente, se basearia nas palavras que já foram ditas, certo? É exatamente assim que funcionam os MLP. Eles aprendem a prever a próxima palavra em uma sequência, analisando um grande conjunto de textos e identificando padrões de linguagem.

Eles podem ser vistos como detetives da linguagem que investigam textos em busca de pistas sobre quais palavras costumam aparecer juntas e com que frequência. Essas pistas são traduzidas em probabilidades, que indicam a chance de uma determinada palavra aparecer após outra.

Um conceito central nos MLP é a probabilidade de uma sequência de palavras, que determina a chance de uma sequência específica aparecer em um determinado contexto. Por exemplo, um modelo treinado em um *corpus* de notícias seria capaz de determinar que a sequência "O presidente anunciou um novo" tem maior probabilidade de ser seguida pela palavra "decreto" do que pela palavra "sorvete".

Um dos modelos probabilísticos mais simples e eficazes são os N-gramas que calculam a probabilidade de uma palavra ocorrer com base nas $n-1$ palavras anteriores. Modelos populares como unigramas (consideram apenas a palavra atual), bigramas (consideram a palavra atual e a anterior) e trigramas (consideram as duas palavras anteriores a atual) são amplamente utilizados em diversas tarefas de PLN, como correção ortográfica e predição de texto (Jurafsky; Martin, 2024). A intuição de modelos N-gramas é que em vez de calcular a probabilidade de uma palavra dado o histórico de palavras anteriores, nós queremos aproximar o histórico utilizando apenas as últimas palavras.



1.1.2.1 APLICAÇÕES

Os MLP são interconectados e podem ser combinados para criar sistemas. Por exemplo, modelos N-gramas podem ser usados como base para modelos de tópicos, fornecendo informações contextuais sobre as palavras. LMs podem ser integrados a sistemas de tradução automática, usando sua capacidade de analisar a estrutura sintática para realizar traduções mais precisas. As aplicações dos MLP abrangem diversas áreas, como:

- **Previsão de texto:** implementados em teclados de *smartphones* e editores de texto para sugerir a próxima palavra ou frase.
- **Tradução automática:** tradução de textos de uma língua para outra, utilizando LMs para modelar a estrutura das línguas envolvidas.

- **Análise de sentimentos:** identificar a opinião expressa em um texto, seja ela positiva, negativa ou neutra.
- **Classificação de documentos:** auxiliam na categorização automática de textos em tópicos predefinidos.

Apesar de sua utilidade, os MLP enfrentam vários desafios. Um dos principais problemas é a maldição da dimensionalidade: à medida que o tamanho do vocabulário e a complexidade das relações entre as palavras aumentam, a quantidade de dados necessários para treinar um modelo preciso cresce exponencialmente. Além disso, os MLP têm dificuldade em capturar o significado de longo alcance e o contexto de uma frase, limitando sua capacidade de lidar com tarefas mais complexas de compreensão da linguagem.

Apesar dessas limitações, os MLPs são a base para o desenvolvimento da área de PLN. As técnicas e os resultados obtidos com esses modelos continuam a ser utilizados em conjunto com as abordagens mais recentes, como as arquiteturas *Transformers*.

1.1.3 MODELOS DE LINGUAGEM NEURAIS

Os MLN (Turney; Pantel, 2010; Collobert et al., 2011; Mikolov et al., 2010) definem a probabilidade de sequências de palavras por meio neurais, por exemplo, *perceptron* multicamadas e RNN. Os MLN possuem muitas vantagens sobre os MLP, já que podem lidar com sentenças e documentos muito mais longos, podem generalizar melhor sobre contextos de palavras similares e serem mais precisos na predição de palavras. Por outro lado, MLN são muito mais complexos, mais lentos e precisam de mais recursos computacionais para treinar, além de serem menos interpretáveis do que MLP. Por isso, para algumas tarefas menores, um MLP ainda é a ferramenta certa.

Os MLN foram desenvolvidos para construir uma solução unificada e de ponta a ponta para várias tarefas de PLN (Bengio; Ducharme; Vincent, 2000). Um dos principais modelos, denominado *Word2vec* (Kombrink et al., 2011; Mikolov et al., 2013) foi proposto para construir uma rede neural simplificada para aprender representações distribuídas de palavras, que se mostraram muito eficazes em uma variedade de tarefas de PLN. Esses estudos iniciaram o uso de MLN para aprendizado de representação, tendo um impacto importante no campo de PLN.

Os MLN representam palavras por meio de *embeddings*, o que permite que MLN generalizem melhor que MLP para dados não vistos. Por exemplo, suponha que o LM recebeu esta frase na etapa de treinamento:



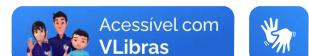
Eu tenho que garantir que o gato seja alimentado.

Vamos supor que, na etapa de treinamento, o modelo não recebeu exemplos com as palavras "seja alimentado" depois da palavra "cão". Nosso conjunto de testes possui o prefixo "Eu esqueci de garantir que o cão seja". Qual é a próxima palavra? Um LM N-grama irá prever "alimentado" depois de "que o gato seja", mas não depois de "que o cão seja". Mas um MLN, sabendo que "gato" e "cão" possuem *embeddings* similares, será capaz de generalizar a partir do contexto "gato" para atribuir uma probabilidade alta o suficiente para "alimentado" mesmo depois de ver "cão".

1.1.3.1 APLICAÇÕES

Os MLN, como redes recorrentes (RNN) e memória de longo prazo e curto prazo (LSTM), têm sido utilizados em diversas aplicações de PLN no mundo real. Algumas das principais aplicações desses modelos incluem:

1. **Classificação de texto:** as capacidades de processamento de linguagem das RNNs e LSTMs permitem utilizá-las para a classificação de textos em diferentes categorias, como sentimento, tópico, spam etc.
2. **Geração de diálogo:** criação de agentes conversacionais capazes de manter diálogos fluidos com seres humanos.
3. **Detecção de anomalias em texto:** as LSTMs podem ser usadas para detectar anomalias e desvios em textos, como identificar fraudes em documentos financeiros ou detectar conteúdo inadequado em postagens online.
4. **Extração de informações:** podem ser treinados para extrair informações estruturadas, como nomes de entidades, relações e fatos, a partir de textos não estruturados.



As redes neurais são treinadas em grandes conjuntos de dados de texto para aprender padrões, relações semânticas e até mesmo nuances de estilos da linguagem. Mas, os limites dos MLN estão se tornando cada vez maiores com o surgimento de novos modelos, como o *Transformer*, que utiliza mecanismos de atenção para capturar relacionamentos de longo prazo entre palavras, superando algumas das limitações de RNNs e LSTMs.

1.1.4 MODELOS DE LINGUAGEM PRÉ-TREINADOS

Os LM pré-treinados são sistemas de IA que aprendem a compreender e gerar texto a partir da análise de grandes conjuntos de dados. Esses modelos são treinados em uma ampla variedade de textos,

como livros, artigos, websites e diálogos, adquirindo um conhecimento profundo sobre a estrutura, o significado e o uso da linguagem natural. Devido ao seu treinamento em larga escala, esses modelos requerem uma enorme quantidade de recursos computacionais, como poder de processamento com *Graphics Processing Units* (GPUs) ou *Tensor Processing Units* (TPUs) e armazenamento.

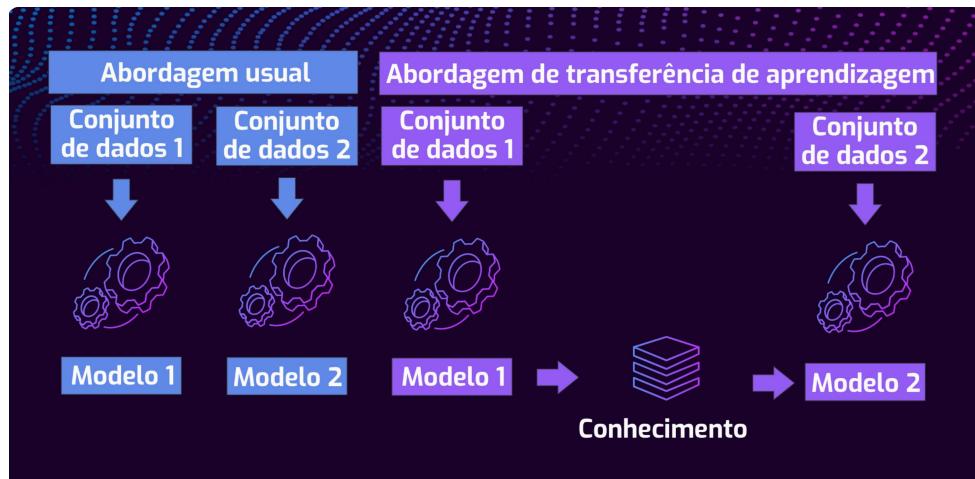
Uma das principais características desses modelos é a sua capacidade de transferência de aprendizado. Após serem pré-treinados em grandes conjuntos de dados genéricos, eles podem ser ajustados rapidamente a tarefas e domínios específicos, usando apenas uma fração dos dados e dos recursos computacionais necessários para construir um modelo do zero.

Na Figura 3, são comparadas duas abordagens de treinamento de modelos de aprendizado de máquina. Na abordagem usual, dois modelos distintos são treinados separadamente com diferentes conjuntos de dados, resultando em dois modelos específicos e independentes. Já na abordagem de transferência de aprendizado, um modelo é inicialmente treinado com um conjunto de dados (conjunto de dados 1), e o conhecimento adquirido é transferido para um segundo modelo, que é então refinado com um novo conjunto de dados (conjunto de dados 2). Isso permite que o segundo modelo aprenda anterior, acelerando o treinamento e melhorando o desempenho em tarefas específicas. Essa técnica é muito útil em LLMs, onde modelos pré-treinados em grandes volumes de dados são ajustados para aplicações especializadas por meio de um processo denominado ***fine-tuning*** ou **ajuste fino**.

Nessa comparação entre a abordagem usual de treinamento de modelos, onde cada modelo é treinado do zero com um conjunto de dados específico, e a abordagem de *Transfer Learning*, o conhecimento de um modelo pré-treinado é transferido para outro modelo, permitindo um treinamento especializado.

Figura 3 - Comparação entre a abordagem usual de treinamento de modelos





Fonte: adaptada de Singh (2024).

Fine-tuning é um processo em que um modelo de IA, que já foi treinado em uma grande quantidade de dados gerais, é ajustado para realizar uma tarefa específica. Por exemplo, um cozinheiro pode saber fazer de tudo um pouco, mas deseja se tornar um especialista em fazer pizzas. Ele já sabe cozinhar, então ele não precisa aprender do zero, mas desenvolver as habilidades para fazer a melhor pizza possível. Esse treinamento adicional, focado em pizzas, é como o *fine-tuning*.

No contexto de um modelo de IA, o "cozinheiro" é o modelo pré-treinado, e o "treinamento para fazer pizzas" é o processo de *fine-tuning*. Esse ajuste é feito usando um novo conjunto de dados que é mais relevante para a tarefa específica que você quer que o modelo faça. Por exemplo, se o modelo original foi treinado com textos gerais da internet, o *fine-tuning* poderia ser feito com textos médicos, se o objetivo for criar um modelo para auxiliar em diagnósticos médicos. Assim, o modelo ajustado consegue desempenhar a nova tarefa de forma mais precisa do que se tivesse que treiná-lo do zero apenas com os dados específicos.

Nos últimos anos, vários LMs pré-treinados têm se destacado. O ELMo (Peters et al., 2018), por exemplo, foi uma das primeiras propostas para capturar representações de palavras com reconhecimento de contexto, usando uma rede LSTM bidirecional. Já o BERT (Devlin et al., 2019), baseado na arquitetura *Transformer* (Vaswani, 2017), trouxe avanços ao pré-treinar LMs bidirecionais em larga escala. Seguindo o paradigma de "pré-treinamento e ajuste fino", outros modelos como o GPT-2 (Radford et al., 2019) e o BART (Fedus; Zoph; Shazeer, 2022) também foram propostos, introduzindo diferentes arquiteturas ou estratégias de pré-treinamento. Esses LMs pré-treinados tornaram-se referência na área, cada qual com suas características e funcionalidades particulares e são a base para a criação dos LLMs.



1.1.4.1 APLICAÇÕES

Os LMs pré-treinados impulsionaram avanços em diversas áreas, tais como:

- **Geração de texto:** o GPT, com sua capacidade de gerar textos coerentes e criativos, se destaca nesse quesito, sendo utilizado em chatbots, criação de conteúdo e ferramentas de escrita.
- **Tradução automática:** modelos como o BERT e GPT podem ser adaptados para tarefas de tradução, utilizando sua compreensão da linguagem para gerar traduções mais precisas e fluentes.
- **Sumarização de texto:** LMs pré-treinados podem ser utilizados para extrair as informações principais de um texto, gerando resumos concisos e informativos.
- **Análise de sentimento:** o ELMo e o BERT, com sua capacidade de entender o contexto das palavras, podem ser utilizados para analisar o sentimento de um texto, identificando se ele expressa sentimentos positivos, negativos ou neutros.

Os LLMs, como GPT-4, se baseiam em princípios e técnicas dos modelos pré-treinados, mas possuem tamanhos e capacidades ainda maiores. Esses modelos representam a vanguarda da pesquisa em PLN, abrindo novas possibilidades para a interação humano-computador com a criação dos LLMs.



1.2 GRANDES MODELOS DE LINGUAGEM

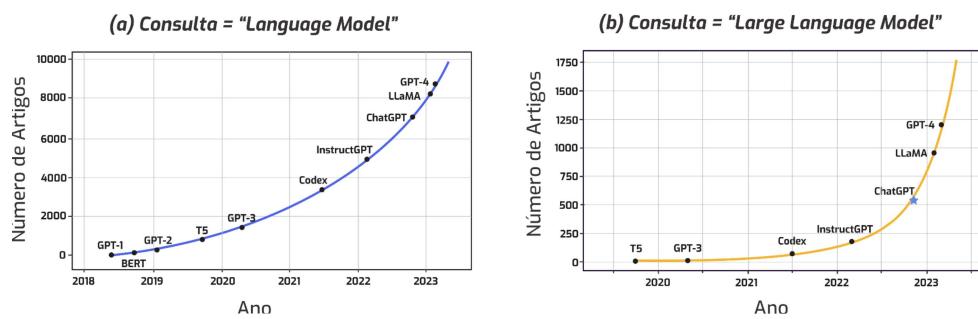
Pesquisadores descobriram que a ampliação dos LMs pré-treinados (por exemplo, a ampliação do tamanho do modelo ou do tamanho dos dados) muitas vezes leva a uma capacidade de modelo aprimorada em tarefas específicas (seguindo a lei da escalabilidade de Kaplan *et al.*, 2020). Vários estudos exploraram o limite de desempenho treinando um modelo pré-treinado cada vez maior (por exemplo, o GPT-3 com 175 bilhões de parâmetros e o PaLM com 540 bilhões de parâmetros). Embora o escalonamento seja conduzido principalmente no tamanho do modelo (com arquiteturas e tarefas de pré-treinamento semelhantes), esses LLMs apresentam comportamentos diferentes dos modelos menores (por exemplo, BERT com 330 milhões de parâmetros e GPT-2 com 1,5 bilhão de parâmetros) e demonstram habilidades impressionantes, denominadas habilidades emergentes (Wei *et al.*, 2022) ao resolver uma série de tarefas complexas. Por exemplo, o GPT-3 pode resolver tarefas que o GPT-2 não consegue fazer de forma satisfatória.

O termo "grandes modelos de linguagem" refere-se a LMs pré-treinados de grande porte (Hoffmann *et al.*, 2022; Taylor *et al.*, 2022; Wei

et al., 2022; Shanahan, 2024). Na Figura 4, é mostrado como esses modelos atraem cada vez mais atenção da pesquisa com o aumento acentuado de artigos do arXiv relacionados a LLMs após o lançamento do ChatGPT.

As estatísticas são calculadas usando correspondência exata ao consultar as frases-chave no título ou resumo por meses. Os marcos importantes com o lançamento de novos LLMs estão indicados nos gráficos.

Figura 4 - Tendências dos números cumulativos de artigos do arXiv que contêm as frases-chave "*language model*" (desde junho de 2018) (a) e "*large language model*" (desde outubro de 2019) (b)



Fonte: adaptada de Zhao (2023).

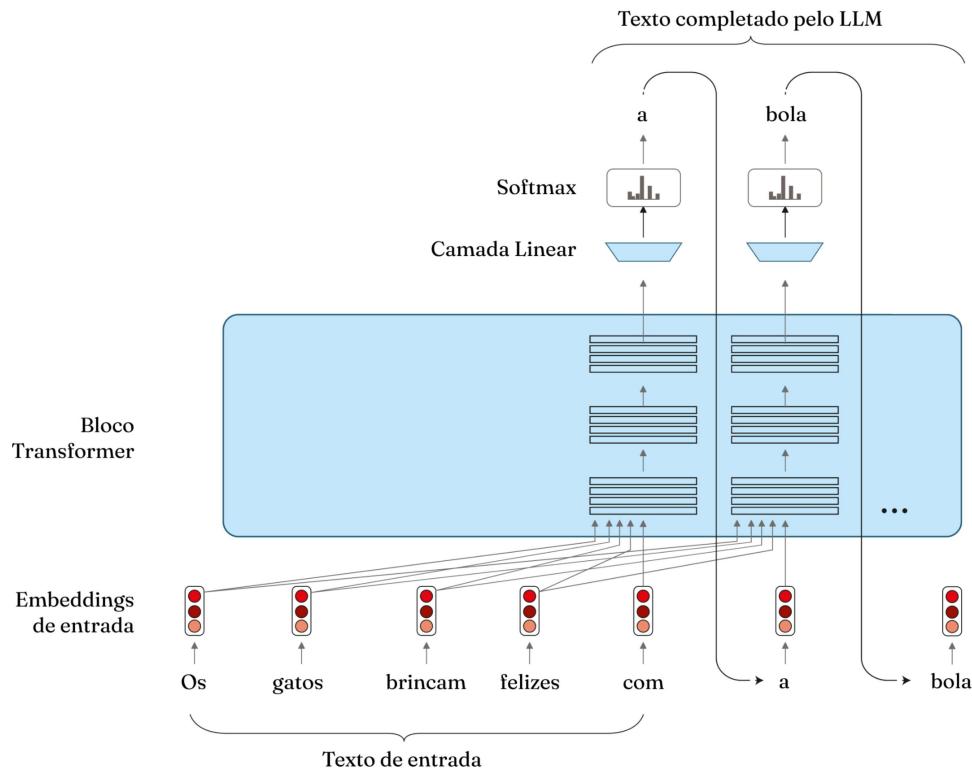
1.2.1 GRANDES MODELOS DE LINGUAGEM COM TRANSFORMERS



A arquitetura *Transformer* foi a propulsora da evolução da área de PLN. Diferente de modelos anteriores, como as RNNs, a arquitetura *Transformer* permite paralelizar o processamento e lidar com sequências de texto muito longas. Essa arquitetura permitiu o desenvolvimento de LLMs capazes de gerar texto de forma condicional, com base em um *prompt* de entrada. A habilidade da arquitetura *Transformer* de acessar todo o contexto anterior, incluindo as próprias saídas geradas, é a base para a tarefa de geração de texto.

Considere a simples tarefa de completar a sentença “Os gatos brincam felizes com ...”, ilustrada na Figura 5. Aqui, um modelo de linguagem recebe um prefixo de texto e é solicitado a gerar um possível complemento. Observe que, à medida que o processo de geração prossegue, o modelo tem acesso direto ao contexto, bem como a todas as suas próprias saídas geradas subsequentemente (pelo menos o máximo que couber na grande janela de contexto). Essa capacidade de incorporar a totalidade do contexto anterior e as saídas geradas a cada etapa é a chave para o poder dos LLMs construídos a partir da arquitetura *Transformer*.

Figura 5 - Tarefa de completar textos utilizando grandes modelos de linguagem baseados em arquitetura *Transformer*



Fonte: adaptada de Zhao (2023).

Mas, por que devemos nos preocupar em prever as próximas palavras? O diferencial da modelagem de linguagem é que muitas tarefas práticas de PLN podem ser formuladas como prever palavras e que um LM poderoso o suficiente pode resolvê-las com um alto grau de precisão. Por exemplo, podemos considerar a tarefa de análise de sentimentos como modelagem de linguagem, fornecendo a um LM um contexto como:

O sentimento da frase “Eu gosto de filmes de ação” é:

Devemos comparar a seguinte probabilidade condicional das palavras “positivo” e a palavra “negativo” para ver qual é maior. Se a palavra “positivo” for mais provável, dizemos que o sentimento da frase é positivo; caso contrário, dizemos que o sentimento é negativo.

Também podemos classificar tarefas mais complexas como previsão de palavras. Considere a tarefa de responder a perguntas simples, onde o sistema recebe alguma pergunta e deve dar uma resposta textual. Podemos considerar a tarefa de responder a perguntas como previsão de palavras, fornecendo a um LM uma pergunta e um token como ‘R’ sugerindo que uma resposta deve vir a seguir:

Q: Quem escreveu o livro “A Origem das Espécies”? R:

Se pedirmos a um LM para calcular quais palavras têm probabilidades mais altas, podemos ver que Charles é muito provável.

Em seguida, se escolhermos Charles e continuarmos:

Q: Quem escreveu o livro “A Origem das Espécies”? R: Charles.

Agora, podemos ver que Darwin é a palavra mais provável e podemos selecioná-la.

Quais palavras devemos gerar a cada etapa? Uma maneira simples de gerar palavras é sempre gerar a palavra mais provável, dado o contexto. Isso é chamado de decodificação gulosa. Um algoritmo guloso é aquele que faz uma escolha local ótima, quer venha a ser ou não a melhor escolha no final (Edmonds, 1971). Assim, na decodificação gulosa, a cada etapa da geração, a saída é escolhida calculando a probabilidade para cada saída possível (cada palavra no vocabulário) e, em seguida, escolhendo a palavra de maior probabilidade.

Na prática, não usamos decodificação gulosa com LLMs. Um grande problema com a decodificação gulosa é que, como as palavras que ela escolhe são (por definição) previsíveis, o texto resultante é genérico e frequentemente bastante repetitivo. Na verdade, a decodificação gulosa é tão previsível que é até determinística; se o contexto for idêntico e o modelo probabilístico for o mesmo, a decodificação gulosa sempre resultará na geração exatamente da mesma sentença. Na maioria das outras tarefas, entretanto, as pessoas preferem textos que foram gerados por métodos mais avançados, chamados métodos de amostragem. Esses introduzem um pouco mais de diversidade nas gerações. Veremos como fazer isso na próxima seção.



1.2.2 GERAÇÃO DE TEXTO POR AMOSTRAGEM EM ARQUITETURAS TRANSFORMER

Ao gerar um texto com um LM, a cada passo, o modelo seleciona o próximo *token* com base nas escolhas anteriores, utilizando probabilidades dos *tokens*. Isso significa que *tokens* com alta probabilidade têm mais chances de serem escolhidas. Para facilitar o entendimento, nos próximos exemplos, iremos assumir que um *token* é uma palavra e o modelo tem como objetivo predizer a próxima palavra. Se o modelo atribui uma probabilidade de 70% para a palavra "bola" e 10% para a palavra "zebra" no contexto da sentença "Os gatos brincam felizes com a ___", é mais provável que a palavra "bola" seja escolhida.

A amostragem é o processo pelo qual o modelo escolhe a próxima palavra em uma sequência com base nas probabilidades que ele calcula. Esse processo de escolher a próxima palavra é chamado de **decodificação** e acontece de forma autoregressiva, ou seja, a cada passo, escolhemos a próxima palavra com base nas palavras que já foram

geradas anteriormente. Isso significa que o modelo "lê" o texto que já foi gerado e, com base nesse contexto, decide qual será a próxima palavra.

Um dos métodos mais comuns para fazer essa decodificação em LLMs é chamado de **amostragem**, introduzindo um grau de aleatoriedade na escolha das palavras. Em vez de escolher sempre a palavra com a maior probabilidade, o modelo seleciona a próxima palavra com base na distribuição de probabilidade calculada, permitindo que palavras menos prováveis sejam escolhidas ocasionalmente. Essa abordagem pode resultar em textos mais diversos e criativos, pois o modelo é induzido a explorar diferentes caminhos e possibilidades. O modelo também pode produzir resultados menos coerentes ou fora de contexto, uma vez que a escolha aleatória pode levar a combinações de palavras inesperadas ou menos lógicas.

Por isso, em vez de usar uma amostragem aleatória, existem métodos que ajudam a evitar a seleção dessas palavras muito improváveis. Uma técnica popular é o *top-k sampling*, que limita as escolhas às k palavras mais prováveis, e o *top-p sampling* (também conhecido como *nucleus sampling*), que seleciona palavras com base em um intervalo de probabilidade cumulativa. Essas abordagens ajudam a manter a qualidade do texto gerado enquanto ainda permitem alguma flexibilidade.



Nas próximas seções deste livro, vamos explorar algumas técnicas de amostragem que permitem ajustar o equilíbrio entre qualidade e diversidade na geração de texto com LMs. Essas técnicas possibilitam priorizar palavras com probabilidades mais altas (para geração de textos mais coerentes) ou dar mais chances a palavras com probabilidades medianas (para gerar textos mais criativos e diversificados). A escolha da técnica e dos parâmetros adequados dependerá do tipo de texto que se deseja gerar e dos objetivos da tarefa.

1.2.2.1 AMOSTRAGEM TOP-K

A amostragem top-k é uma técnica que visa equilibrar a previsibilidade e a criatividade na geração de texto por modelos baseados na arquitetura *Transformer*. Em vez de optar pela palavra com a maior probabilidade, como no método de decodificação gulosa, a amostragem top-k introduz um grau de aleatoriedade controlada, permitindo que o modelo explore mais opções sem comprometer a qualidade do texto gerado.

O top-k funciona restringindo o espaço de busca do modelo às k palavras mais prováveis, conforme calculadas pelas suas pontuações de probabilidade. Inicialmente, o modelo ordena todas as palavras possíveis

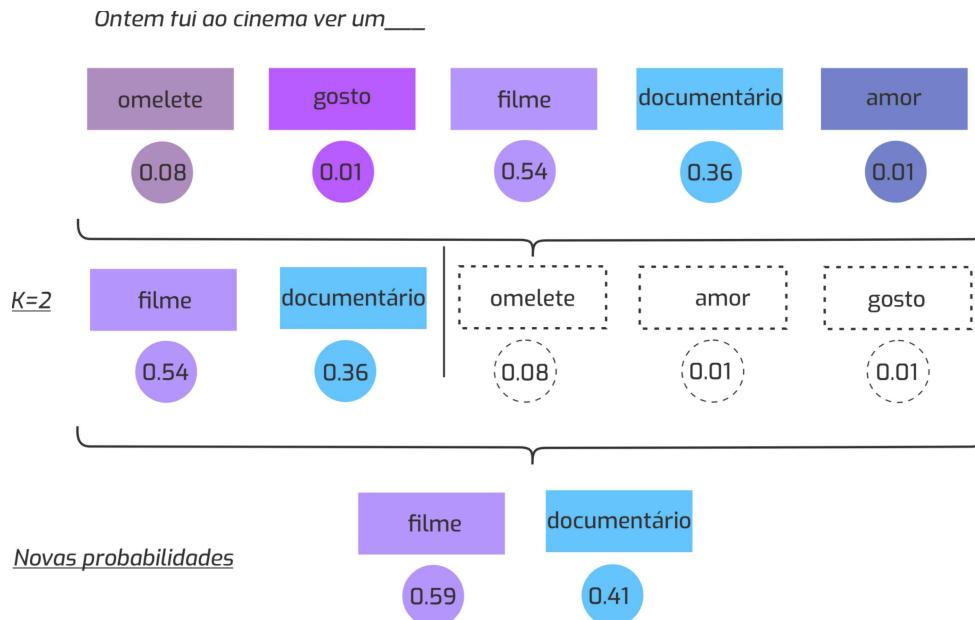
com base nas suas probabilidades. Em seguida, em vez de escolher diretamente a palavra mais provável, ele limita a escolha apenas às k palavras mais prováveis e amostra aleatoriamente entre essas opções. Esse método permite que o modelo gere textos que são ao mesmo tempo coerentes e variados, evitando a repetição constante de escolhas muito prováveis e o uso de palavras muito improváveis.

Na Figura 6, podemos ver um exemplo de top-k com k=2, para escolher a próxima palavra da frase “*Ontem fui ao cinema ver um*”. Primeiro, o modelo atribui probabilidades a várias palavras possíveis, como “filme”, “documentário”, “omelete”, entre outras. Porém, ao aplicar o top-k, apenas as duas palavras com as maiores probabilidades (“filme” e “documentário”) são mantidas, descartando as outras opções. Como resultado, as probabilidades das palavras restantes são normalizadas, aumentando as chances de escolher uma das palavras selecionadas. Nesse exemplo, a probabilidade de “filme” aumenta para 0,59 e a de “documentário” para 0,41 (somando 1), mostrando como o top-k influencia a geração de texto, restringindo a variedade de palavras e focando nas opções mais prováveis.

Na Figura 6, são apresentadas apenas as duas palavras com maiores probabilidades (“filme” e “documentário”) são mantidas, enquanto as demais são descartadas, resultando em uma redistribuição de probabilidades.



Figura 6 - Exemplo de aplicação do parâmetro Top-k (k=2) na geração de texto



Fonte: imagem adaptada de Viejo (2023).

Uma das principais vantagens da amostragem Top-k é que ela pode ser ajustada para diferentes necessidades de geração de texto. Por exemplo, se o valor de k for baixo, o modelo restringe suas escolhas a um

pequeno número de palavras, resultando em texto mais previsível e consistente, mas possivelmente menos criativo. Por outro lado, um valor maior de k permite que o modelo explore mais opções, aumentando a diversidade do texto gerado, mas com o risco de introduzir incoerências ou gerar sequências menos lógicas, aumentando a probabilidade de “alucinação”.

Entretanto, a escolha do valor de k não deve ser feita de forma arbitrária. Ela deve considerar o contexto e os objetivos da geração de texto. Em aplicações onde a precisão e a consistência são importantes, como na tradução automática ou na geração de respostas em assistentes virtuais, valores menores de k podem ser preferíveis. Já em tarefas criativas, como a geração de histórias ou poesia, um k maior pode ser mais adequado para aumentar a originalidade e a criatividade.

1.2.2.2 AMOSTRAGEM NUCLEUS OU TOP-P

Um problema com a amostragem top-k é que k é fixo, mas a forma da distribuição de probabilidade sobre as palavras difere em diferentes contextos. Se definirmos k = 10, às vezes as 10 palavras principais serão muito prováveis e incluirão a maior parte da massa de probabilidade, mas, outras vezes, a distribuição de probabilidade será mais plana e as 10 palavras principais incluirão apenas uma pequena parte da massa de probabilidade.



Além da técnica de amostragem top-k, outra técnica popular para controle da geração de texto em modelos com arquitetura *Transformer* é a amostragem top-p. Ela oferece uma abordagem dinâmica para selecionar palavras, ajustando automaticamente o tamanho do conjunto de palavras consideradas para cada passo da geração, com base na probabilidade cumulativa.

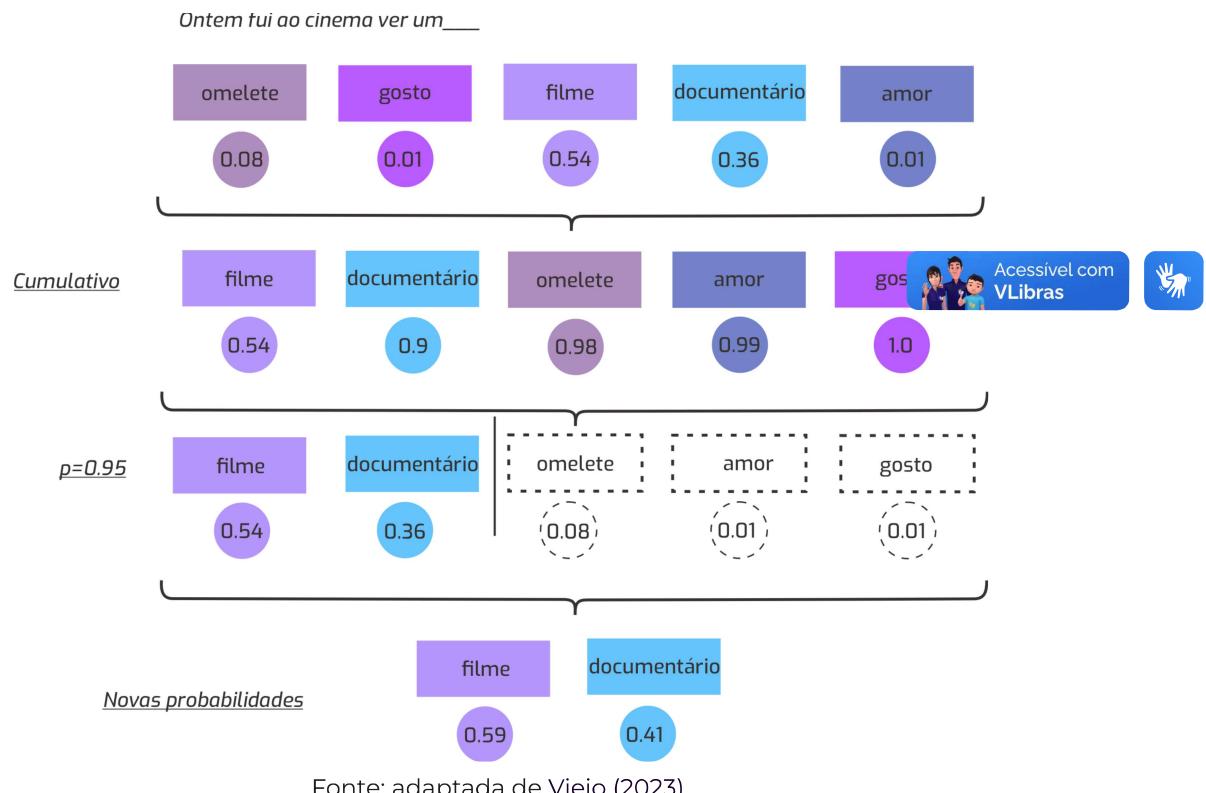
Na amostragem top-p, em vez de limitar a escolha a um número fixo de palavras, como na top-k, o modelo seleciona o menor subconjunto de palavras cujo somatório das probabilidades atinge ou excede um valor de limiar predefinido, chamado de p. Esse subconjunto é chamado de “núcleo”. Uma vez identificado o núcleo, a próxima palavra é escolhida aleatoriamente dentro desse conjunto, com base nas probabilidades originais.

Na Figura 7, é apresentado um exemplo da amostragem top-p, tendo como entrada a frase “Ontem fui ao cinema ver um”. O modelo inicialmente atribui probabilidades a várias palavras possíveis, como “filme”, “documentário”, “omelete”, entre outras. À medida que as probabilidades são somadas, elas formam uma distribuição cumulativa. Com top-p configurado para 0,95, as palavras “filme” e “documentário”

são selecionadas, pois suas probabilidades combinadas atingem o limite definido, enquanto as palavras com probabilidades mais baixas, como "omelete", "amor" e "gostar", são descartadas. Isso resulta em uma redistribuição das probabilidades das palavras selecionadas, com "filme" recebendo uma probabilidade de 0,59 e "documentário" de 0,41. O top-p é uma técnica que permite manter a diversidade de geração de texto, limitando a seleção apenas às palavras mais relevantes de acordo com o contexto.

Na Figura 7, a seguir, na aplicação do parâmetro top-p, as palavras mais prováveis são selecionadas até que a soma cumulativa das probabilidades atinja o valor de 0,95. As palavras com menores probabilidades são descartadas, resultando em uma redistribuição das probabilidades entre as palavras restantes.

Figura 7 - Exemplo de aplicação do parâmetro top-p ($p=0,95$) na geração de texto



A principal vantagem da amostragem top-p em relação à top-k é a sua flexibilidade. Em situações onde uma única palavra é bem mais provável do que as outras, o núcleo pode ser pequeno, similar a um valor baixo de k , garantindo que o modelo não introduza opções incoerentes. Quando várias palavras têm probabilidades comparáveis, o núcleo pode se expandir para incluir mais opções, permitindo uma maior diversidade na geração de texto. Essa adaptabilidade pode resultar em saídas que combinam coerência com criatividade, dependendo do contexto.

A escolha do valor de p é importante para o desempenho do modelo. Valores baixos de p resultam em núcleos pequenos, o que pode fazer com que o texto gerado seja mais previsível, semelhante ao comportamento com um baixo valor de k . Já valores altos de p permitem a inclusão de mais palavras no núcleo, aumentando a diversidade do texto gerado, mas também o risco de se introduzir palavras menos coerentes com o contexto. Um valor típico de p utilizado é 0,9, o que permite que o modelo mantenha uma boa combinação de coerência e criatividade.

O top- p é útil em tarefas onde a flexibilidade e a adaptabilidade são necessárias, como na criação de diálogos ou textos criativos, onde o modelo deve manter uma resposta relevante ao contexto, mas também deve explorar diferentes possibilidades de forma dinâmica. Ao ajustar p conforme a necessidade da aplicação, é possível gerar saídas que são ao mesmo tempo variadas e coerentes, atendendo a vários contextos e propósitos na geração de texto.

1.2.2.3 TEMPERATURA

A temperatura é um hiperparâmetro que controla o nível de aleatoriedade e criatividade do texto gerado. Ela funciona como um fator de ajuste que modifica a distribuição de probabilidades das palavras previstas pelo modelo, influenciando diretamente a seleção da palavra na sequência.



A temperatura impacta nas probabilidades preditivas do modelo. Especificamente, as probabilidades são divididas pela temperatura antes de serem passadas por uma função de softmax para converter essas pontuações em uma distribuição final de probabilidades. Quando a temperatura é maior que 1, as probabilidades se tornam mais uniformes, o que aumenta a diversidade do texto gerado ao permitir que o modelo escolha palavras menos prováveis com maior frequência. Isso pode resultar em saídas mais criativas, mas também pode aumentar o risco de incoerências.

Por outro lado, quando a temperatura é menor que 1, as diferenças entre as probabilidades das palavras são amplificadas. Nesse caso, o modelo tende a escolher as palavras mais prováveis com mais frequência, resultando em um texto mais previsível e consistente, mas menos criativo. Uma temperatura muito baixa pode fazer com que o modelo gere saídas muito repetitivas ou até mesmo que ele trave em loops de repetição.

Um valor de temperatura baixo (< 1 ex: 0,2 ou 0,5) possui as seguintes características:

- Ideal quando se espera uma resposta mais precisa.
- Uma temperatura de 0 significa que o *token* com maior probabilidade geralmente será selecionado.
- Concentra a maior parte da massa de probabilidade nas próximas palavras mais prováveis.
- Isso resulta em gerações de texto mais coerentes e repetitivas, mas com menos criatividade ou exploração de opções menos prováveis.

Um valor de temperatura alto ≥ 1 (ex: 1,5 ou 2,0) possui as seguintes características:

- Torna as previsões do modelo mais dispersas e “incertas”.
- Ele achata e distribui a distribuição de probabilidade mais uniformemente entre as palavras.
- Isso permite que o modelo experimente com mais frequência escolhas de palavras menos prováveis, permitindo gerações de texto mais criativas, exploratórias e “surpreendentes”.
- Aumenta a probabilidade de escolhas menos coerentes e com alucinação.

O ajuste da temperatura permite que os desenvolvedores controlam o estilo e a natureza do texto gerado conforme a necessidade da aplicação. Em contextos onde a precisão e a coerência são necessários, como em traduções ou respostas automatizadas, uma temperatura baixa pode ser ideal. Em tarefas criativas, como a geração de histórias ou poesia, uma temperatura mais alta pode incentivar a exploração de ideias novas e inesperadas. Em resumo, a temperatura é como um botão que você pode ajustar para encontrar o equilíbrio perfeito entre precisão e criatividade.

- **Temperatura baixa:** ideal para resumos, traduções e textos que exigem precisão e clareza.
- **Temperatura alta:** perfeita para poemas, roteiros e textos que buscam a originalidade e a criatividade.

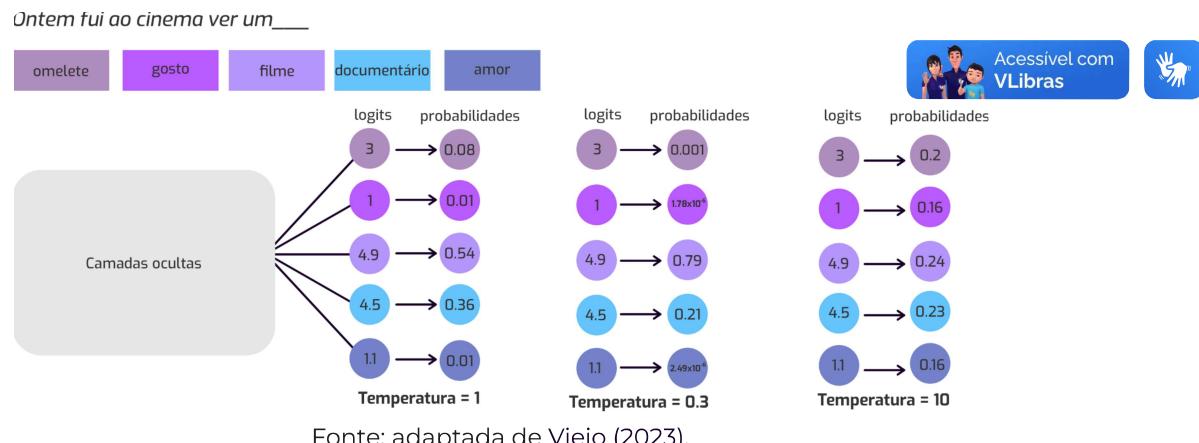
Na Figura 8, é ilustrado como a temperatura afeta a probabilidade de diferentes palavras em uma tarefa de geração de texto, onde o objetivo é completar a frase "Ontem fui ao cinema ver um ...". Cada círculo azul representa uma possível palavra, com os valores numéricos indicando os *logits* (à esquerda) e as probabilidades correspondentes (à direita) para cada temperatura. *Logits* são as saídas não normalizadas do modelo antes de escolher qual será a próxima palavra a ser gerada. A função

softmax é aplicada para converter esses logits em probabilidades, e o modelo seleciona a próxima palavra com base nessas probabilidades.

Observe que com a temperatura em 1 (valor padrão), a palavra "filme" possui a maior probabilidade (0,54), seguida por "documentário" (0,36). Aumentar a temperatura para 10 torna as probabilidades mais uniformes, dando chances maiores para palavras como "omelete" e "amor", que antes eram menos prováveis. Ao contrário, diminuir a temperatura para 0,3 concentra a probabilidade em "filme" (0,79), tornando a escolha da palavra mais "previsível" e menos criativa.

Na Figura 8, a seguir, é ilustrado como a temperatura afeta a seleção da próxima palavra com base nos *logit scores* calculados pelo modelo. À esquerda, com temperatura igual a 1, as probabilidades refletem as diferenças nos *logit scores*. No centro, com uma temperatura de 0,3, as diferenças são ampliadas, resultando em uma maior concentração de probabilidade na palavra mais provável. À direita, com temperatura 10, as diferenças são suavizadas, distribuindo a probabilidade mais uniformemente entre as opções.

Figura 8 - Impacto da temperatura na distribuição de probabilidades em modelos de linguagem



Fonte: adaptada de Viejo (2023).

Compreender os efeitos da temperatura é importante para otimizar o desempenho do modelo em diferentes tarefas, desde a geração de respostas precisas até a produção de conteúdo criativo. A fórmula da **temperatura** em LMs é usada para controlar a "aleatoriedade" ou "exploração" na geração de texto ao mostrar a próxima palavra (*token*) com base nas probabilidades previstas pelo modelo:

$$P(w_i) = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_i}{T}\right)}$$

onde:

- $P(w_i)$ é a probabilidade ajustada da palavra w_i ,

- z_i é o *logit* (valor bruto) para a palavra w_i ,
- T é a **temperatura**, que controla a suavização da distribuição,
- A soma no denominador é feita sobre todos os *logits* z_j .

A temperatura é uma maneira de regular o grau de exploração do modelo na geração de texto, afetando a previsibilidade e a variedade das respostas. A seguir, são apresentadas como a temperatura funciona internamente nos LLMs:

- **Cálculo das probabilidades:** o LM calcula os *logits* z_i para cada palavra no vocabulário, baseado no contexto das palavras anteriores.
- **Divisão pela temperatura:** os *logits* são divididos pelo valor da temperatura:
 - Se **temperatura < 1**: as probabilidades se tornam mais extremas, ou seja, probabilidades mais altas se tornam mais altas, e as mais baixas se tornam mais baixas.
 - Se **temperatura > 1**: as probabilidades se tornam menos extremas, aproximando-se de 0.
- **Aplicação de softmax:** após o ajuste pela temperatura, aplica-se a função *softmax* para converter essas probabilidades distribuição de probabilidade final, que soma 1 e determina a seleção das palavras durante a geração de texto.



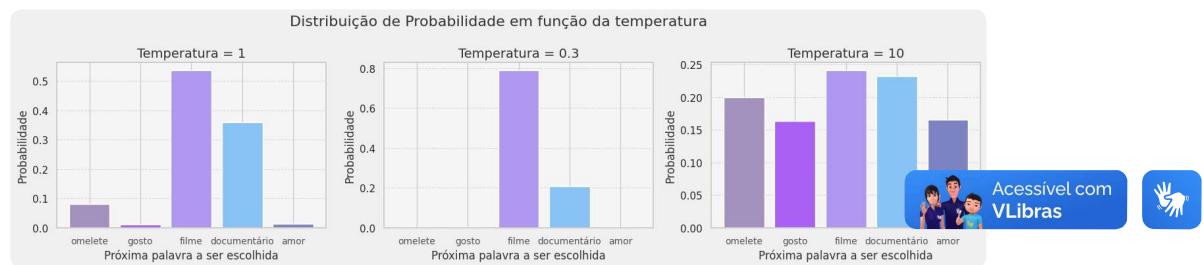
Na Figura 9, podemos ver como a aplicação da temperatura influencia a distribuição de probabilidades. Quando o LM calcula as probabilidades iniciais para as palavras possíveis, ele na verdade trabalha com *logit scores*. É aqui que a temperatura aparece: ao dividir os *logit scores* pela temperatura, ajusta-se a "extremidade" dessas pontuações antes de passá-las pela função *softmax*, que transforma esses valores ajustados em probabilidades finais.

- Temperatura = 1 (à esquerda na figura): nesse caso, os *logit scores* são divididos por 1, o que significa que não há modificação nas diferenças relativas entre eles. A função *softmax* então converte esses *logit scores* diretamente em probabilidades, refletindo as diferenças originais de confiança do modelo. Como resultado, a palavra com o *logit* mais alto ("filme", com logit 4,9) tem uma probabilidade considerável (0,54), mas outras palavras também têm grandes chances de serem escolhidas.
- Temperatura < 1 (0,3) (no centro): os *logit scores* são divididos por um valor menor que 1, o que faz com que as diferenças entre eles

sejam ampliadas. As probabilidades se tornam mais extremas: a palavra com o maior *logit* ("filme") tem agora uma probabilidade de 0,79, enquanto as outras praticamente desaparecem como opções (probabilidades muito próximas de zero). Isso ocorre porque a função *softmax* transforma as diferenças aumentadas nos *logit scores* em probabilidades ainda mais concentradas, privilegiando fortemente a palavra mais provável.

- Temperatura > 1 (10) (à direita): quando a temperatura é maior que 1, os *logit scores* são divididos por um valor maior, o que achata as diferenças entre eles. Na função *softmax*, isso se traduz em uma distribuição de probabilidades mais uniforme, onde mesmo palavras com *logit scores* relativamente baixos recebem uma probabilidade maior do que normalmente teriam. Isso permite que o modelo explore uma mais opções de palavras, tornando a escolha final mais aleatória e menos previsível.

Figura 9 - Distribuição de probabilidade em função da temperatura



Fonte: autoria própria.

A temperatura ajusta o equilíbrio entre previsibilidade e criatividade na geração de texto por LLMs, controlando a distribuição de probabilidades das palavras. Temperaturas mais baixas tornam o modelo mais conservador, enquanto temperaturas altas aumentam a diversidade. Esse ajuste, embora útil na inferência, depende da qualidade do treinamento do modelo, que ensina o modelo a gerar distribuições precisas. Assim, treinamento e temperatura se complementam para criar um modelo eficaz e flexível.

1.3 TREINAMENTO DE GRANDES MODELOS DE LINGUAGEM

Para treinar uma arquitetura *Transformer* como um LM, temos como entrada um *corpus* de texto como material de treinamento e, a cada etapa de tempo t , pedimos ao modelo para prever o próximo *token*. Chamamos esse modelo de auto-supervisionado porque não precisamos adicionar nenhum rótulo com a resposta; a sequência natural de *tokens* é sua própria supervisão. Nós treinamos o modelo para

minimizar o erro na previsão da próxima *token* na sequência de treinamento, usando a entropia cruzada como a função de perda.

Nos exemplos desta seção, iremos utilizar exemplos de predição da próxima palavra para facilitar o entendimento. Podemos utilizar como exemplo a frase "Os gatos brincam felizes com a". O modelo precisa prever qual palavra vem a seguir. Como ele já viu muitas frases similares durante o treinamento, ele pode prever que a próxima palavra será "bola", "lá", ou "caixa". Seu trabalho é adivinhar a palavra que melhor se encaixa no contexto.

Treinar LLMs é como ensinar um computador a prever e gerar textos. O treinamento de um LLM envolve alimentar o modelo com grandes quantidades de texto. Esse texto pode ser retirado da internet, livros, artigos, entre outros. A ideia é que, com essa quantidade enorme de dados, o modelo consiga aprender padrões na linguagem, como quais palavras costumam aparecer juntas e em quais contextos.

Esse treinamento é chamado de auto-supervisionado. Isso significa que o modelo aprende sozinho, sem precisar que um humano diga a resposta certa para cada frase. Em vez disso, ele usa o próprio texto como orientação. Se a frase for "Os gatos brincam felizes com a", o modelo sabe que "bola" é a resposta correta e ajusta suas "configurações internas" para melhorar suas previsões futuras.



Na Figura 10, é ilustrada a abordagem geral de treinamento. Em cada etapa, dadas todas as palavras anteriores, a camada final do *transformer* produz uma distribuição de saída sobre todo o vocabulário. Cada palavra é convertida em um *embedding*, que é então processado pelos blocos do *Transformer*. A cada etapa, o modelo tenta prever a próxima palavra, recebendo "retorno" do quanto a resposta está correta e ajustando seus parâmetros. No final, o modelo gera uma distribuição de probabilidade sobre todas as palavras possíveis, escolhendo a palavra com a maior probabilidade como sua previsão.

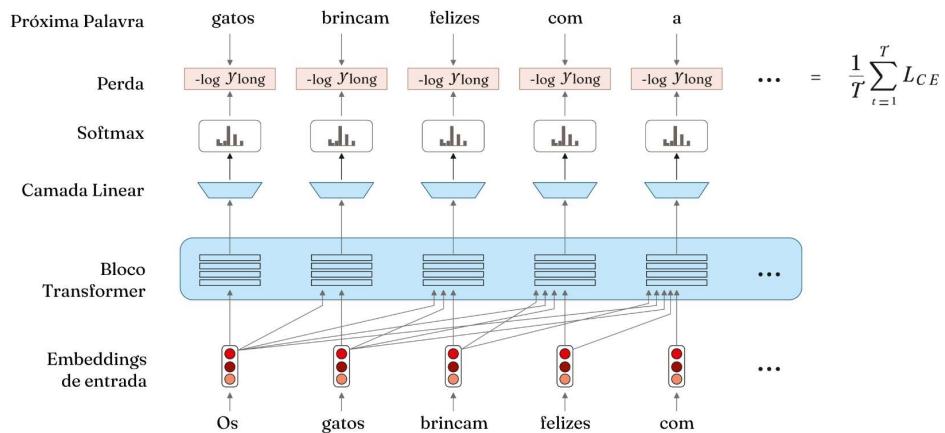
Na imagem, vemos um modelo que, a cada nova palavra recebida, tenta prever a próxima. Por exemplo, se o modelo já recebeu as palavras "Os gatos brincam felizes com a", ele agora tenta prever "bola". Se a previsão estiver correta, o modelo ganha um retorno positivo; se não, ele ajusta suas configurações para melhorar na próxima vez.

Durante o treinamento, a probabilidade atribuída à palavra correta é usada para calcular a perda de entropia cruzada para cada item na sequência. Como acontece com as RNNs, a perda para uma sequência de treinamento é a perda média de entropia cruzada em toda a sequência. Os pesos na rede são ajustados para minimizar a perda

média da entropia cruzada sobre a sequência de treinamento por meio de descida de gradiente. Ou seja, o modelo ajusta seus pesos para prever melhor as palavras corretas em uma frase, usando um cálculo para medir o quanto longe suas previsões estão da resposta certa, e melhora com base nesses erros.

Na Figura 10, é ilustrado como funciona o treinamento de um modelo como o *Transformer*, que é uma das arquiteturas mais usadas para criar LLMs.

Figura 10 - Treinamento de um grande modelo de linguagem como Transformer



Fonte: adaptada de Jurafsky; Martin (2024).



Esse ciclo de previsão e correção se repete milhões de vezes com diferentes frases e contextos até que o modelo fique muito bom em prever palavras e gerar textos que façam sentido. No final do processo, o modelo se torna capaz de realizar tarefas complexas, como responder perguntas, resumir textos e até mesmo escrever histórias.

No caso da modelagem de linguagem, a distribuição correta vem do conhecimento da próxima palavra. Isso é representado como um vetor *one-hot* correspondente ao vocabulário, onde a entrada para a próxima palavra real é 1 e todas as outras entradas são 0. Assim, a perda de entropia cruzada para modelagem de linguagem é determinada pela probabilidade que o modelo atribui à próxima palavra correta.

Assim, em cada posição de palavra t da entrada, o modelo recebe como entrada a sequência correta de *tokens* e os usa para calcular uma distribuição de probabilidade sobre as próximas palavras possíveis, a fim de calcular a perda do modelo para o próximo *token*. A ideia de que sempre damos ao modelo a sequência de histórico correta para prever a próxima palavra (em vez de alimentar o modelo com seu melhor caso da etapa de tempo anterior) é denominada *teacher forcing*. Isso permite que todos os *tokens* de *output* sejam corrigidos, ajustando a distribuição de probabilidade dos *tokens* preditos para cada um dos *tokens*.

propagados no modelo, além de economizar processamento ao garantir um treinamento mais eficiente.

Modelos grandes são geralmente treinados preenchendo a janela de contexto completa com texto. Se os documentos forem mais curtos que isso, vários documentos são agrupados na janela com um *token* especial de fim de texto entre eles. O tamanho do lote para a descida de gradiente é geralmente muito grande.

O treinamento de LLMs é um processo contínuo e iterativo. Os pesquisadores estão constantemente buscando novas técnicas e arquiteturas para aprimorar a capacidade dos modelos. Novas formas de representar as palavras, novos métodos de treinamento e novas estruturas de redes neurais são exploradas para tornar os LLMs cada vez mais robustos e versáteis.

O objetivo final é criar modelos que não apenas imitem a linguagem humana, mas que também sejam capazes de compreendê-la profundamente, gerando textos criativos, informativos e úteis para uma variedade de tarefas. O treinamento de LLMs é um processo que envolve uma combinação de dados massivos, algoritmos avançados e um ciclo constante de aprendizado. É por meio desse processo que os LLMs se tornam capazes de gerar textos que se assemelham àqueles escritos por humanos, abrindo possibilidades para a comunicação, a criatividade e a resolução de problemas.



1.3.1 RELAÇÃO ENTRE NÚMERO DE TOKENS, TAMANHO DO MODELO E CUSTO DE TREINAMENTO

O treinamento de LLMs envolve uma complexa interação entre o número de *tokens*, o tamanho do modelo e o custo de treinamento. Esses três fatores estão relacionados e impactam diretamente tanto no desempenho computacional e na acurácia do modelo final. Existe um equilíbrio entre o número de *tokens*, o tamanho do modelo e o custo de treinamento. Aumentar qualquer um desses fatores pode melhorar o desempenho do modelo, mas ao custo de maior complexidade e recursos financeiros.

O número de *tokens* refere-se à quantidade de pedaços de texto em que os dados de treinamento são divididos. Um *token* pode ser uma palavra, uma parte de uma palavra ou até mesmo um símbolo de pontuação. Quanto mais *tokens* um modelo é exposto durante o treinamento, mais informações ele pode absorver. Isso permite que o modelo comprehenda nuances e padrões da linguagem de maneira mais precisa. Porém, utilizar mais *tokens* também significa mais ciclos de

processamento, aumentando o tempo e os recursos necessários para o treinamento.

O tamanho do modelo é geralmente medido pelo número de parâmetros, que são os componentes ajustáveis que o modelo usa para fazer previsões. Modelos maiores têm mais parâmetros, o que lhes permite capturar e modelar mais detalhes sobre a linguagem. Quanto maior o número de parâmetros, maior é a capacidade de processamento necessária para treinar o modelo. Isso significa que modelos maiores demandam mais memória e poder computacional, o que aumenta o custo de treinamento.

O custo de treinamento de um LM grande é uma função direta do número de *tokens* e do tamanho do modelo. Modelos com mais parâmetros treinados em corpora de grande escala (com bilhões ou até trilhões de *tokens*) requerem um volume grande de recursos computacionais. Isso inclui o uso de processadores especializados, como GPUs e TPUs, que são caros tanto em termos de *hardware* quanto de energia. Além disso, o tempo necessário para treinar esses modelos aumenta exponencialmente com o tamanho, o que pode resultar em custos que chegam a milhões de dólares.

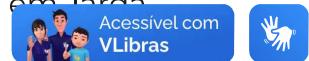
O GPT-3, com seus 175 bilhões de parâmetros e treinado em 300 bilhões de *tokens*, foi um marco na área de IA, permitindo que o modelo aprenda novas tarefas com apenas alguns exemplos no *prompt*, sem a necessidade de ajustes em seus pesos. A configuração específica de 175 bilhões de parâmetros e 300 bilhões de *tokens* não foi arbitrária. Ela foi baseada em um estudo cuidadoso das leis de escalabilidade para redes neurais, como descrito no artigo "Scaling Laws for Neural Language Models" (Kaplan et al., 2020), publicado pela OpenAI, em 2020.

Esse artigo introduziu, pela primeira vez, equações que permitiam estimar a qualidade de um modelo neural antes mesmo de seu treinamento, considerando três variáveis principais: o tamanho do modelo (número de parâmetros), a quantidade de dados de treinamento (número de *tokens*) e a quantidade de computação necessária (medida em *Floating Point Operations per Second [FLOPs]*). A OpenAI, ao extrapolar essas equações, determinou que, dado o orçamento computacional disponível na época, a melhor configuração seria um modelo com 175 bilhões de parâmetros, treinado com 300 bilhões de *tokens*. Esta proporção de aproximadamente dois *tokens* por parâmetro foi seguida por outros projetos, como o OPT-175B da Meta, treinado com a mesma proporção, e o PaLM do Google, que possuía 540 bilhões de parâmetros e foi treinado com 780 bilhões de *tokens*.



Posteriormente, pesquisadores da DeepMind identificaram que havia um erro na metodologia usada no artigo da OpenAI, que superestimava o número de parâmetros e subestimava a quantidade de tokens necessária para otimizar a qualidade do modelo. Com a correção dessas equações, a equipe da DeepMind desenvolveu o modelo Chinchilla (Hoffmann et al., 2022). Eles descobriram que, para um treinamento com computação otimizada, o tamanho do modelo e o número de *tokens* de treinamento devem ser escalados igualmente: quando o tamanho do modelo dobra, o número de *tokens* de treinamento também deve dobrar. Esse ajuste mostrou que treinar com muito mais *tokens* do que anteriormente considerado levava a uma qualidade superior, mesmo com um custo computacional semelhante.

A partir daí, iniciou-se uma corrida por mais dados de treinamento, com modelos como o GPT-4 (embora o número exato de *tokens* não seja público), PaLM 2 (Google) e Megatron-Turing NLG (NVIDIA e Microsoft) adotando proporções de *tokens* por parâmetro cada vez maiores. Essa história mostra que detalhes aparentemente pequenos, como a escolha da proporção ideal entre *tokens* e parâmetros, podem ter um impacto profundo no sucesso dos LMs. A busca por esse equilíbrio continua a evoluir, com implicações diretas não apenas na qualidade dos modelos, mas também no custo e na viabilidade de seu treinamento em larga escala.



1.3.2 CORPORA DE TREINAMENTO PARA GRANDES MODELOS DE LINGUAGEM

Corpora de treinamento, conjuntos de dados usados para treinar modelos de aprendizado de máquina, são obtidos por meio de uma variedade de métodos, dependendo da tarefa e dos recursos disponíveis. Um método comum é a coleta de dados da web, onde se utilizam técnicas como *web scraping* para extrair texto, imagens, áudio e vídeo de sites, plataformas de mídia social e outras fontes online.

O texto da web é geralmente retirado de corpora de páginas da web coletadas automaticamente, como o common crawl, uma série de snapshots de toda a web produzidos pela organização sem fins lucrativos Common Crawl (Common Crawl, 2024), que possuem bilhões de páginas da web. Existem várias limpezas de dados de rastreamento comum, como o *Colossal Clean Crawled Corpus* (Raffel et al., 2020), um corpus de 156 bilhões de *tokens* de inglês que é filtrado de várias maneiras (removendo linguagem não natural como código, frases com palavras ofensivas de uma lista de bloqueio, etc.). O que há nesses dados? Uma análise sugere que, em grande parte, são documentos de

texto de patentes, Wikipédia e sites de notícias (Dodge *et al.*, 2021). A Wikipédia desempenha um papel importante em muitos treinamentos de LMs, assim como vários corpora de livros. Os modelos GPT3, por exemplo, são treinados principalmente na web (429 bilhões de *tokens*), conjuntos de textos de livros (67 bilhões de *tokens*) e Wikipédia (3 bilhões de *tokens*).

Dados existentes, como livros, artigos científicos, notícias e transcrições de conversas, podem ser reutilizados para formar um *corpus*. Bibliotecas digitais, repositórios de dados abertos e coleções privadas podem ser fontes de treinamento. A qualidade e a representatividade desses dados devem ser avaliadas antes do uso.

A colaboração em massa (*crowdsourcing*) envolve a contribuição de um grande número de pessoas na criação e anotação de dados, frequentemente por meio de plataformas online. Essa abordagem permite a coleta rápida de grandes quantidades de dados a um custo relativamente baixo. Esse tipo de dado pode ser utilizado, mas é importante garantir mecanismos de controle de qualidade devido à confiança da informação.

Em alguns casos, dados gerados automaticamente são utilizados. Isso pode incluir texto sintético gerado por LMs, imagens criadas por algoritmos de geração de imagens ou dados simulados de ambientes virtuais. Dados gerados automaticamente podem ser úteis para aumentar *corpora* existentes ou para criar dados para tarefas específicas.

A criação manual de dados é outra abordagem, onde especialistas humanos criam e anotam os dados, garantindo alta qualidade e precisão. Esse método é mais trabalhoso e caro, mas pode melhorar o resultado de tarefas que exigem conhecimento especializado ou onde a disponibilidade de dados é limitada. Frequentemente, a criação manual é combinada com outras técnicas para aumentar o tamanho do *corpus*.

A combinação de diferentes fontes de dados também é uma prática comum para criar *corpora* mais abrangentes e representativas. A combinação pode envolver a fusão de *corpora* existentes, a adição de dados coletados da web ou a integração de dados criados manualmente. Uma tarefa a ser realizada neste tipo de dado é garantir a consistência e a compatibilidade dos diferentes conjuntos de dados durante o processo de combinação.

Como esses *corpora* de treinamento são muito grandes, é provável que contenham muitos exemplos naturais que podem ser úteis para tarefas de NLP, como pares de perguntas e respostas (por exemplo, de



listas de perguntas frequentes), traduções de frases entre vários idiomas, documentos junto com seus resumos e assim por diante.

Em todos os casos, a consideração de questões éticas, como privacidade, direitos autorais e vieses presentes nos dados, devem ser levadas em conta durante todo o processo de obtenção de *corpora* de treinamento. A qualidade, a representatividade e o tamanho do *corpus* afetam diretamente o desempenho do modelo de aprendizado de máquina treinado nele.

1.4 DESAFIOS DOS GRANDES MODELOS DE LINGUAGEM

Os LLMs são ferramentas com grande potencial, mas é importante usá-los com responsabilidade e consciência. Por exemplo, LMs são propensos a dizer coisas falsas, um problema chamado “alucinação”. Os LMs são treinados para gerar texto previsível e coerente, mas os algoritmos probabilísticos que vimos até agora não têm como garantir que o texto gerado seja correto ou verdadeiro. Isso pode ocasionar diversos problemas para qualquer aplicativo onde os fatos são importantes!

Outro desafio é que os LMs podem gerar linguagem tóxica (Gehman *et al.*, 2020), onde mesmo prompts atóxicos podem levar LLMs a produzirem discurso de ódio e abusar de seus usuários. LMs também geram estereótipos (Cheng, Durmus, Jurafsky, 2023) e atitudes negativas (Sheng, *et al.*, 2019; Brown, *et al.*, 2020) sobre muitos grupos demográficos.

Uma fonte de vieses são os dados de treinamento (Gehman *et al.*, 2020), onde os conjuntos de dados de treinamento de LM extensa incluem texto tóxico extraído de sites proibidos. Existem outros vieses além da toxicidade: os dados de treinamento são desproporcionalmente gerados por autores dos Estados Unidos da América e de países desenvolvidos. Essas amostras tendenciosas provavelmente distorcem a geração resultante em direção às perspectivas ou tópicos deste grupo de países. Além disso, os LMs podem amplificar os vieses demográficos e preconceituosos.

LMs também podem ser usados por agentes maliciosos para gerar texto para desinformação, *phishing* ou outras atividades socialmente prejudiciais (Brown, *et al.*, 2020). Pesquisas mostram que LLMs podem gerar texto que emula extremistas online (Mcguffie; Newhouse, 2020), com o risco de amplificar movimentos extremistas e sua tentativa de radicalizar e recrutar.



LMs também apresentam problemas de privacidade, pois podem vazar informações sobre seus dados de treinamento. Assim, é possível que um adversário extraia texto de dados de treinamento de um LM, como o nome, número de telefone e endereço de uma pessoa (Mcguffie; Newhouse, 2020; Giray, 2023). Isso é um problema se LLMs são treinados em conjuntos de dados privados, como registros eletrônicos de saúde.

Relacionada à privacidade está a questão dos direitos autorais. LLMs podem ser treinados em documentos protegidos por direitos autorais. Em alguns países, como os Estados Unidos da América, a doutrina do uso justo permite que conteúdo protegido por direitos autorais seja usado para construir LMs, mas não se eles forem usados para gerar texto que concorra com o mercado para o texto no qual são treinados.

Encontrar maneiras de mitigar todos esses danos é uma importante área de pesquisa atual em PLN. No mínimo, analisar os dados usados para pré-treinar LLMs é importante como uma forma de entender as questões de toxicidade, viés, privacidade e uso justo, tornando importante que os LMs forneçam informações completas e replicáveis sobre os *corpora* usados para treiná-los. Modelos de código aberto podem especificar seus dados de treinamento exatos. Os requisitos para que os modelos sejam transparentes de tais maneiras também estão em processo de incorporação aos regulamentos de vários



nacionais.



SAIBA MAIS...

Artigos acadêmicos e publicações:

- *Attention is All You Need* (Vaswani et al., 2017): este artigo apresenta a arquitetura *Transformer*, que é a base dos LLMs.
- *Scaling Laws for Neural Language Models* (Kaplan et al., 2020): discute como o aumento do tamanho dos LMs impacta diretamente a performance e as capacidades emergentes.

Ferramentas e notebooks interativos:

- Utilize o *Hugging Face* para testar diversos LMs e ajustar parâmetros como temperatura e top-p sampling.

Disponível em: <https://huggingface.co/>.

Vídeos e cursos online:

- Curso de PLN da *Stanford University*: Esse curso aborda em detalhes as técnicas utilizadas para construir e ajustar LLMs. Disponível em: <https://web.stanford.edu/class/cs224n/>.
- *Deep Learning Specialization* no *Coursera*, oferecido por Andrew Ng: Um ótimo recurso para aprender sobre redes neurais e transformers. Disponível em: <https://www.coursera.org/specializations/deep-learning>.

Livro recomendado:

- *Speech And Language Processing*, 3^a edição (Jurafsky, 2024).



PARA RELEMBRAR...

Nessa Unidade, exploramos os fundamentos dos LLMs, desde os tradicionais modelos probabilísticos, como n-gramas, até os avançados LLMs baseados em arquiteturas Transformer. Também discutimos a evolução dos LMs, destacando a transição de modelos neurais para os grandes modelos pré-treinados e suas aplicações, incluindo assistentes virtuais, tradutores automáticos e corretores de texto.

Além disso, explicamos como os LLMs são treinados em grandes quantidades de dados, o que lhes permite gerar texto similar ao humano e realizar diversas tarefas, como tradução, geração de texto e análise de sentimentos. Aqui estão os pontos-chave abordados nessa Unidade:

- LMs: modelos que prevêem a probabilidade da próxima palavra em uma sequência, atribuindo probabilidades a palavras ou sequências de palavras.
- LLMs: versões maiores e mais poderosas dos LMs, treinadas em conjuntos de dados massivos, com bilhões ou até

trilhões de parâmetros. Eles exibem uma compreensão profunda da linguagem e do contexto.

- Modelos probabilísticos: usam regras estatísticas e probabilidades para prever palavras, como os modelos N-gramas. São mais simples, mas menos eficazes em contextos longos.
- Modelos neurais: utilizam redes neurais (RNNs, LSTMs) para processar a linguagem. Capturam melhor o contexto, mas são computacionalmente mais caros.
- Modelos pré-treinados (ex: BERT, ELMo): treinados em grandes conjuntos de dados e depois ajustados para tarefas específicas. Oferecem bom desempenho com poucos dados adicionais.
- *Transfer learning*: técnica que permite transferir o conhecimento adquirido em uma tarefa para outra tarefa relacionada, acelerando o treinamento e melhorando o desempenho em tarefas específicas.
- *Fine-tuning* (ajuste fino): adaptação de um modelo pré-treinado para uma tarefa específica, ajustando seus parâmetros com um conjunto de dados menor e especialista em tarefas específicas.
- Escalabilidade: capacidade de um sistema (incluindo de lidar com o aumento da demanda (dados, usuários, complexidade) sem perder desempenho computacional ou acurácia.





Grandes Modelos
de Linguagem

Unidade II
**Engenharia
de *prompts***



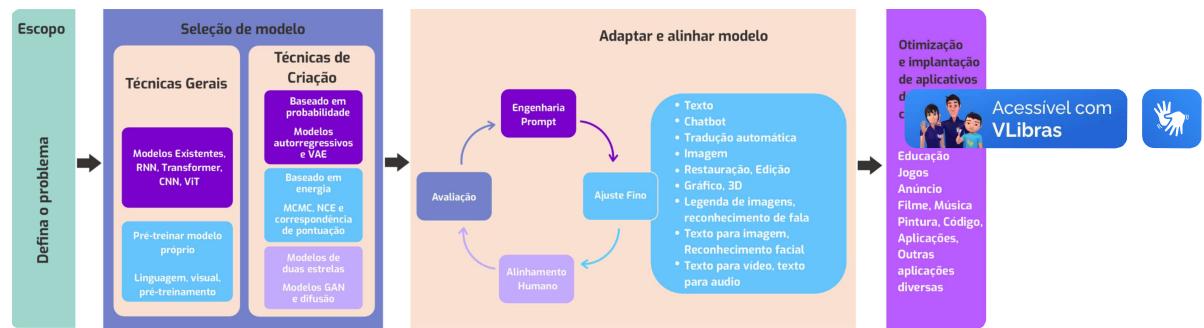
UNIDADE II - ENGENHARIA DE *PROMPTS*

2.1 INTRODUÇÃO À ENGENHARIA DE PROMPTS

A IA está alterando a maneira como interagimos com a tecnologia. De textos e imagens a código e música, a IA generativa nos permite criar conteúdo de forma rápida. Mas para aproveitar todo o potencial dessas ferramentas, precisamos aprender a nos comunicar com elas de forma eficaz.

Este conceito se manifesta no processo de desenvolvimento de aplicações baseadas em Inteligência Artificial Generativa (IAG), como ilustrado na Figura 11. Esta *pipeline* começa com a definição do escopo do projeto e se estende até a otimização e implantação final das aplicações alimentadas por *Large Language Models*. Na Figura 11, é apresentado um fluxograma detalhado que pode ser dividido em três fases principais: seleção do modelo, adaptação e alinhamento do modelo, além da otimização e implantação do modelo.

Figura 11 - Ciclo de design e aplicação de IA Generativa



Fonte: adaptada de Hadi et al. (2024).

Dessa forma, o ciclo segue as seguintes etapas:

- 1) Escopo: definir o problema e o escopo;
- 2) Seleção de modelo: seleção de modelo com a escolha das técnicas de modelagem adequadas, como modelos autorregressivos, modelos baseados em energia ou modelos de difusão;
- 3) Adaptação e alinhamento de modelo: otimização para tarefas específicas por meio da engenharia de *prompts*, ajuste fino do modelo ou alinhamento humano;
- 4) Otimização e implantação do modelo ajustado para várias aplicações nos domínios de educação, jogos, mídia e muito mais.

Na fase de **Seleção do Modelo**, os desenvolvedores podem optar por técnicas gerais, que incluem o uso de modelos existentes (como RNN, Transformer e Convolutional Neural Networks [CNN]) ou a criação de modelos próprios pré-treinados. Alternativamente, podem explorar

técnicas de criação mais específicas, como modelos autoregressivos baseados em verossimilhança ou modelos baseados em energia.

A fase de **Adaptação e Alinhamento do Modelo** se baseia em *prompts* bem elaborados para a tarefa específica. Esta etapa envolve três processos principais: engenharia de *prompts*, avaliação e *fine-tuning*. A engenharia de *prompts* é particularmente importante pois permite ajustar o comportamento do modelo para tarefas específicas sem necessidade de retreinamento completo.

A fase final de **Otimização e Implantação** foca na entrega de aplicações alimentadas por LLMs em diversos domínios, como educação, jogos, saúde, entretenimento e desenvolvimento de software.

Os *prompts* gerados na fase de Adaptação e Alinhamento do Modelo podem ser usados para ajustar LMs para tarefas específicas, como gerar texto, traduzir idiomas, resumir texto ou responder a perguntas. Um *prompt* é o texto de entrada fornecido ao modelo para gerar uma resposta. Ele serve como instrução, contexto e direcionamento para o LLM, influenciando o conteúdo, o estilo e a estrutura da saída gerada. A Engenharia de *Prompt* permite que os usuários otimizem o desempenho do modelo para uma variedade de tarefas. Por meio de *prompts* bem formulados, os usuários podem orientar o modelo para gerar saídas desejáveis.



Mas, afinal o que é engenharia de *prompt*? Engenharia de *prompt* é o processo de descobrir *prompts* que geram de forma confiável resultados úteis ou desejados, ou seja, a habilidade de elaborar entradas para modelos de IA generativa, como o ChatGPT, a fim de obter os resultados desejados. **Não se trata apenas de fazer perguntas; trata-se de comunicar seus objetivos com clareza e precisão, guiando a IA para gerar respostas relevantes, criativas e de alta qualidade.**

Essa abordagem envolve converter uma ou mais tarefas em um conjunto de dados baseado em *prompts* e treinar um LM usando uma técnica conhecida como “aprendizado baseado em *prompts*” ou “aprendizado de *prompts*” (Giray, 2023; Gero, Liu, Chilton, 2022). A engenharia de *prompts* é uma disciplina recente que se concentra em desenvolver e otimizar *prompts* para utilizar efetivamente LMs de grande escala em diversas aplicações e áreas de pesquisa (White et al., 2023).

Em termos simples, a engenharia de *prompt* é a arte de elaborar a instrução - o *prompt* - para um LM, tal como o ChatGPT, a fim de obter o resultado desejado. É como dar ao modelo um mapa detalhado, com a rota traçada e o destino destacado, para que ele possa navegar pelo

universo da linguagem e trazer as informações mais relevantes e precisas para você.

Aqui está um exemplo simples de uma entrada de *prompt* para um gerador de nomes de produtos (inspirado em um dos exemplos do OpenAI), e a saída resultante do ChatGPT.

Entrada:

Liste os 5 principais livros na área de processamento de linguagem natural

Saída:

Claro, aqui estão os cinco principais livros na área de Processamento de Linguagem Natural (PLN):

- "Speech and Language Processing" de Daniel Jurafsky e James H. Martin
- "Natural Language Processing with Python" de Steven Bird, Ewan Klein e Edward Loper
- "Foundations of Statistical Natural Language Processing" de Christopher D. Manning e Hinrich Schütze
- "Neural Network Methods in Natural Language Processing" de Yoav Goldberg
- "Deep Learning for Natural Language Processing" de Nitish Srivastava, Sumit Pandey e Karan Jain



Ao empregar técnicas de engenharia de *prompts*, conseguimos aproveitar o potencial dos LMs, destacando suas capacidades em vários domínios. Esse cenário abre novas opções para melhorar sistemas de IA e aumentar seu desempenho em diversas aplicações, desde geração de texto até síntese de imagens e além.

Ao desenvolver um *prompt*, o custo de *prompts* ineficientes pode ser alto, tanto em termos financeiros (taxas do OpenAI) quanto em tempo dedicado à correção de erros. Por isso, o conteúdo do *prompt* é importante e alguns pontos devem ser considerados:

• Clareza e Especificidade

- Seja preciso na descrição do que você busca. Por exemplo, em vez de "pedir os principais livros", especifique "os cinco principais livros".
- Forneça contexto incluindo informações relevantes sobre os livros, como público-alvo, funcionalidades e valores da marca.
- Forneça exemplos de livros que você considera adequados para contextualizar o modelo.

- **Controle da Saída**

- Limite a quantidade de caracteres ou palavras para evitar saídas muito longas ou curtas.
- Use técnicas como "*few-shot learning*" (orientação do modelo com poucos exemplos) ou "*chain-of-thought prompting*" (orienta o modelo a explicar seu raciocínio passo a passo) para guiar o modelo em direção à saída desejada.
- Implemente mecanismos para filtrar saídas inadequadas, como nomes de livros que não são da área de PLN.

- **Otimização de Custo**

- Utilize frases concisas e elimine informações irrelevantes para reduzir o custo das taxas do OpenAI.
- Experimente diferentes estruturas de *prompt* e compare os resultados para encontrar a melhor relação custo-benefício.

- **Qualidade do prompt e da saída**

- Permita que os usuários personalizem o *prompt* com informações específicas de livros, artigos ou livros.
- Implemente mecanismos de avaliação da qualidade das saídas geradas para manter a consistência e a relevância.



Ao criar um prompt otimizado, você garante um sistema que irá gerar uma saída mais eficiente, confiável e escalável, que atenderá às necessidades de milhares de usuários.

2.2 PRINCÍPIOS DE ENGENHARIA DE PROMPTS

O livro "*Prompt Engineering for Generative AI*" (Phoenix, Taylor, 2024) apresenta uma lista com cinco princípios a serem seguidos na criação de *prompts*. Esses princípios não são dicas ou truques efêmeros, mas sim convenções geralmente aceitas que são úteis para trabalhar com LMs. Esses princípios são independentes do modelo e devem funcionar para melhorar seu *prompt*, independentemente do modelo que você está usando.

- **Forneça a direção:** descreva o estilo desejado em detalhes.
- **Especifique o formato:** defina quais regras seguir e a estrutura necessária da resposta.

- **Forneça exemplos:** insira um conjunto diversificado de casos de teste onde a tarefa foi realizada corretamente.
- **Avalie a qualidade:** identifique erros e classifique as respostas, testando o que impulsiona o desempenho.
- **Faça a divisão do trabalho:** dvide as tarefas em várias etapas, interligadas para objetivos complexos.

Cada um desses itens será comentado nas subseções seguintes.

2.2.1 FORNEÇA A DIREÇÃO

A clareza das instruções permite obter resultados mais relevantes ao utilizar MLN. Fornecer uma direção clara ajuda a guiar o modelo para compreender exatamente o que é esperado na resposta. A direção deve ser explícita e evitar ambiguidades, detalhando o objetivo da tarefa de maneira específica. Fornecer a direção significa orientar o modelo de forma precisa e detalhada sobre o que você espera como resultado. Isso pode ser feito por meio de:

- **Descrever o estilo desejado em detalhes:** se você deseja um texto formal ou informal, técnico ou criativo, é preciso especificar.
- **Dar exemplos concretos:** mostre ao modelo exemplos de resultados desejáveis ajuda a ilustrar suas expectativas.
- **Definir o contexto:** forneça informações relevantes sobre o público-alvo, a finalidade do texto ou o público alvo do resultado do prompt.
- **Utilizar analogias:** compare a tarefa com algo familiar ao modelo para facilitar a compreensão.



O exemplo abaixo "Escreva um artigo breve sobre os benefícios da IA na medicina, com um tom informativo e acessível" ilustra como fornecer a direção correta (Figura 12).

Figura 12 - Exemplo de Prompt

Entrada:

"Escreva um artigo breve sobre os benefícios da inteligência artificial na medicina, com um tom informativo e acessível."

Saída do ChatGPT:

"A inteligência artificial (IA) está revolucionando a medicina ao melhorar diagnósticos, personalizar tratamentos e otimizar a gestão hospitalar. Com algoritmos avançados, a IA pode analisar grandes volumes de dados clínicos, identificando padrões que podem escapar ao olho humano..."

Fonte: autoria própria.

O exemplo acima define o **objetivo claro** de produzir um artigo sobre os benefícios da inteligência artificial na medicina, detalha o **tipo de conteúdo desejado** ao especificar um "artigo breve", e **estabelece o tom e estilo** como "informativo e acessível". Essas informações direcionam o ChatGPT a gerar um texto conciso, objetivo e de fácil compreensão sobre o tema, evitando respostas excessivamente técnicas ou com tom inadequado.

2.2.2 ESPECIFIQUE O FORMATO

Especificando o formato garante que a IA gere uma saída no formato que você deseja. Se você não especificar o formato, a IA pode gerar uma saída em um formato que é difícil de usar. Você pode especificar o formato fornecendo à IA um exemplo do formato que você deseja ou especificando as regras que a saída deve seguir. Você deve definir as regras e a estrutura da resposta esperada, incluindo detalhes como o tamanho, a formatação e o estilo da saída do modelo.

Existe, com frequência, alguma sobreposição entre o primeiro e segundo princípio, Dar Direção e Especificar Formato (Phoenix, Taylor, 2024). Este último trata da definição do tipo de saída desejada, por exemplo, formato JSON ou o formato de uma foto de banco de imagens. O primeiro trata do estilo de resposta desejado, independentemente do formato, por exemplo, nomes de produtos no estilo de Steve Jobs ou uma imagem de uma reunião de negócios no estilo de Van Gogh. Quando há conflitos entre estilo e formato, geralmente é melhor resolvê-los descartando qualquer elemento que seja menos importante para o resultado final.

Quanto mais detalhado e específico for o *prompt* em relação ao formato da resposta, maior a probabilidade do modelo gerar um resultado satisfatório. Para isso, siga estas etapas ao formular seu *prompt*:

- Seja específico sobre como a resposta deve ser organizada. Use frases como:
 - "Responda na forma de uma lista numerada."
 - "Estruture sua resposta em três parágrafos concisos."
 - "Apresente seus argumentos em um formato de tabela com colunas para 'Prós' e 'Contras'!"
- Forneça exemplos de formato da saída. Se possível, forneça um exemplo simples da estrutura desejada logo após a instrução.
 - "Liste os ingredientes necessários para uma receita de bolo":

- Farinha
- Açúcar
- ..."
- Defina restrições de formato e conteúdo. Por exemplo:
 - "Limite sua resposta a 100 palavras."
 - "Inclua pelo menos três exemplos concretos para cada ponto."
 - "Não use jargões técnicos na explicação."
- Priorize as informações. Se houver elementos obrigatórios que a resposta deve conter, deixe isso claro:
 - "Certifique-se de incluir o ano de publicação, o autor e uma breve sinopse para cada livro."

Veja um exemplo de *prompt* sem formatação na Figura 13 e com formatação na Figura 14.

Figura 13 - Exemplo de *prompt* sem formatação

Pergunta:

Quais são os principais pontos turísticos de Paris e seus horários de funcionamento?

Resposta:

O Arco do Triunfo é imperdível e está aberto das 10h às 23h. A Torre Eiffel é outro ícone, aberta das 9h30 às 23h45. O Museu do Louvre, com a Mona Lisa, abre das 9h às 18h, exceto às terças.



Fonte: autoria própria.

Figura 14 - Exemplo de *prompt* com formatação

Pergunta

Gostaria de uma tabela com os principais pontos turísticos de Paris e seus horários de funcionamento. A tabela deve ter as seguintes colunas: Nome do Ponto Turístico, Horário de Abertura e Horário de Fechamento.

Resposta

Nome do Ponto Turístico	Horário de Abertura	Horário de Fechamento
Arco do Triunfo	10h	23h
Torre Eiffel	9h30	23h45
Museu do Louvre	9h	18h (fechado às terças)

Fonte: autoria própria.

No primeiro *prompt* (sem formatação), o modelo responde com um texto corrido, dificultando a leitura e comparação das informações. No segundo *prompt* (com formatação), a instrução clara de formatação em tabela, incluindo os nomes das colunas, indica ao modelo que ele deve

retornar a informação no formato tabular, tornando a leitura mais fácil e organizada.

2.2.3 FORNEÇA EXEMPLOS

Inserir exemplos diversificados é uma boa prática para orientar o modelo de PLN sobre o tipo de resposta esperada. Exemplos funcionam como uma referência prática, demonstrando de forma clara como a tarefa deve ser realizada. Eles ajudam a estabelecer um padrão de qualidade e fornecem uma base concreta para o LM na criação da resposta.

Ao fornecer exemplos, é importante cobrir muitos casos para mostrar diferentes variações possíveis da tarefa. Por exemplo, se a tarefa é escrever e-mails, você pode incluir e-mails formais, informais, de agradecimento, de reclamação, entre outros. Dessa forma, o modelo pode adaptar sua saída com base nos exemplos fornecidos, resultando em uma melhor correspondência com as expectativas do usuário. De forma geral, você deve seguir os seguintes princípios:

- **Mostre o que você considera uma boa resposta:** use exemplos de alta qualidade e relevantes para a tarefa.
- **Varie os exemplos:** apresente exemplos diversos que cobrem diferentes aspectos da tarefa para que o modelo generalizar melhor.
- **Use poucos exemplos, mas relevantes:** comece com um ou poucos exemplos e adicione mais se necessário. Nem sempre mais é melhor.



2.2.3.1 DIVERSIDADE DE CASOS

Apresente diferentes variações da tarefa para que o LM entenda a amplitude da sua aplicação. Inclua exemplos que representem casos diversos para o modelo, testando sua capacidade de lidar com nuances. Para que o modelo foque em cenários reais, apresente exemplos específicos da tarefa que deseja que ele realize.

Você deve ilustrar a amplitude da tarefa com casos simples e complexos.

Abaixo temos um exemplo incorreto (Figura 15) e outro correto de *prompts* (Figura 16) que apresentam exemplos com diversidade de casos. O exemplo correto demonstra a aplicação em diferentes contextos (informal e formal), explorando a capacidade do modelo de adaptar a tradução.

Figura 15 - Prompt incorreto

Prompt:

"Traduza as seguintes frases para o inglês, adaptando o nível de formalidade:
 Senhora, gostaria de uma xícara de café?
 O curso de inteligência artificial é top demais!"

Fonte: autoria própria.

Figura 16 - Prompt correto*Prompt:*

"Traduza as seguintes frases para o inglês, adaptando o nível de formalidade:
 Senhora, gostaria de uma xícara de café?
 O curso de inteligência artificial é top demais!"

Exemplos:

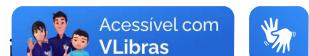
Informal: E aí, beleza? → What's up?
 Formal: Prezado Senhor, agradecemos a sua colaboração.→
 Dear Sir, we appreciate your collaboration."

Fonte: autoria própria.

O *prompt* correto apresenta exemplos com diferentes níveis de complexidade, explorando variações de linguagem formal e informal. Isso permite que o modelo aprenda a lidar com diferentes nuances e gere traduções mais precisas.

2.2.3.2 DETALHE E CLAREZA

Forneça todos os detalhes necessários para cada exemplo, informações contextuais, instruções detalhadas e saídas esperadas. Você deve utilizar uma linguagem simples e clara, evitando termos técnicos complexos que o modelo pode não compreender. Apresente os exemplos de forma consistente, com uma estrutura clara e organizada e inclua exemplos que demonstrem tanto as saídas corretas quanto as incorretas para que o modelo aprenda a identificar padrões.



O *prompt* deve conter informações completas para o modelo compreender a tarefa e o resultado esperado. Abaixo temos um exemplo incorreto e outro correto de *prompts* com detalhe e clareza. O exemplo correto define claramente a tarefa (classificação de sentimento), fornece a frase e indica a saída esperada (negativo), facilitando a compreensão do modelo (Figura 17).

Figura 17 - Exemplos de Prompts correto e incorreto

Prompt Incorreto*Prompt:*

"Classifique o sentimento das frases abaixo:
A aula de IA é ótima!
Estou cansado de esperar na fila!"

Exemplos:

"Que dia lindo! O sentimento desta frase deve ser positivo.
O trânsito está um caos. Esta sentença possui sentimento negativo."

Prompt Correto*Prompt:*

"Classifique o sentimento das frases abaixo como positivo, negativo ou neutro:
A aula de IA é ótima!
Estou cansado de esperar na fila!"

Exemplos:

"Frase: Que dia lindo! Sentimento: Positivo
Frase: O trânsito está um caos. Sentimento: Negativo
Frase: O gato está dormindo. Sentimento: Neutro"

Fonte: autoria própria.

O *prompt* incorreto não especifica as categorias de sentimentos desejadas (positivo, negativo, neutro) e fornece exemplos de saídas esperadas com falta de clareza. O *prompt* correto define a tarefa de forma clara, as categorias de sentimentos e fornece exemplos esperadas para cada categoria, facilitando o entendimento do modelo.



2.2.4 AVALIE A QUALIDADE

Avaliar a qualidade das respostas geradas é importante para identificar erros e refinar os *prompts* para melhorar o desempenho do modelo. A avaliação deve ser minuciosa, considerando tanto a precisão das informações quanto a adequação ao contexto e formato especificado. Identificar falhas comuns e categorizar as respostas com base em sua qualidade permite ajustar os *prompts* de maneira mais eficaz.

Sem um sistema de avaliação correto, existe o risco de utilizar *prompts* que geram respostas imprecisas, inconsistentes ou até mesmo inadequadas para a tarefa em questão. Por exemplo, se um modelo comete erros frequentes de coerência, isso pode indicar a necessidade de revisões nas instruções fornecidas. A avaliação contínua permite iterar sobre os *prompts*, aprimorando progressivamente a performance do modelo de NLP.

Avaliar e iterar sobre os *prompts* pode levar a reduções no tamanho do *prompt* e, portanto, no custo e na latência do seu sistema. Se você

conseguir encontrar outro *prompt* que tenha um desempenho tão bom quanto (ou melhor), mas que use um *prompt* mais curto, você poderá reduzir o custo financeiro e o tempo de resposta do seu sistema. Em muitos casos, você descobrirá neste processo que muitos elementos de um *prompt* complexo são descartáveis.

Existem diversas abordagens para avaliar a qualidade de *prompts*, cada uma com suas próprias vantagens e desvantagens. Uma abordagem simples, porém eficaz, é a avaliação manual por meio de um sistema de classificação, onde o usuário avalia as respostas geradas pelo modelo utilizando um sistema de "joinha positivo" e "joinha negativo". Na Figura 18, é apresentado um exemplo dessa técnica, que apesar de subjetiva, permite identificar rapidamente padrões e desvios nas respostas, especialmente quando comparamos diferentes versões de um mesmo *prompt*.

Figura 18 - Avaliação da qualidade de saída do sistema



Fonte: autoria própria.

Outra abordagem é a avaliação programática, que se baseia na comparação das respostas geradas com um conjunto de respostas de referência, previamente definidas. Essa técnica é útil em tarefas que exigem maior precisão, como em casos de tradução automática ou geração de código. Métricas que medem a similaridade entre as frases geradas e as frases de referência são frequentemente utilizadas nesse tipo de avaliação.

Por fim, a avaliação de *prompts* também pode ser realizada por meio de testes em cenários reais de uso. Nesse tipo de avaliação, o *prompt* é integrado a um sistema real e a sua eficácia é medida em relação aos objetivos de negócio, tais como redução na quantidade de chamadas ao suporte. Essa abordagem, apesar de ser a mais completa, também é a mais complexa e dispendiosa, exigindo um planejamento cuidadoso e uma infraestrutura robusta para coleta e análise de dados.

A escolha da melhor abordagem para avaliação de *prompts* depende do contexto específico da tarefa, dos recursos disponíveis e dos objetivos a serem alcançados. A estratégia de testes em engenharia de *prompts* envolve a criação de um processo estruturado para avaliar e refinar *prompts* de maneira contínua e interativa. Essa abordagem é importante para identificar o que funciona e o que não funciona, permitindo melhorias contínuas na interação com LMs. A seguir, são detalhados os principais componentes desta estratégia na Figura 19.

Figura 19 - Principais componentes da estratégia de testes em engenharia de *prompts*



Fonte: autoria própria.

Existem diversos projetos no GitHub focados na avaliação de *prompts* e LMs. Esses projetos oferecem ferramentas, *benchmarks* (trata-se de um padrão de referência) e *frameworks* (coleções de componentes de software) que auxiliam na análise e melhoria dos *prompts* utilizados com LLMs. Aqui estão alguns dos principais projetos:

1. *LM Evaluation Harness*

- **Descrição:** *framework* de avaliação com poucos exemplos para LMs, que fornece uma variedade de tarefas e *benchmarks*.
- **Link:** [EleutherAI/lm-evaluation-harness](https://github.com/EleutherAI/lm-evaluation-harness)

2. *OpenAI Evals*

- Descrição: avaliação de LMs que permite criar, compartilhar e executar avaliações para LMs, facilitando a análise sistemática de *prompts* e respostas.
- **Link:** <https://github.com/openai/evals>

3. *DeepEval*

- **Descrição:** *framework* para realizar testes em LMs, permitindo identificar pontos fortes e fracos nas capacidades de geração de texto.
- **Link:** <https://github.com/confident-ai/deepeval>



4. *Langfuse*

- **Descrição:** plataforma open-source (de código aberto) que ajuda na observabilidade e avaliação de LMs. Ela fornece ferramentas para coletar, visualizar e analisar interações com LMs, facilitando a identificação de problemas e a medição de desempenho.
- **Link:** <https://github.com/langfuse/langfuse>

5. *LangSmith*

- **Descrição:** ferramenta desenvolvida pelo *LangChain* que se concentra na avaliação, monitoramento e melhoria contínua de LMs. Ela oferece uma plataforma abrangente para rastrear o desempenho dos LMs em várias tarefas, com foco na criação e ajuste de fluxos de trabalho de PLN.
- **Link:** <https://www.langchain.com/langsmith>

2.2.5 FAÇA A DIVISÃO DO TRABALHO

Suponha que você precisa explicar um trabalho complicado para alguém. Se você explicar tudo de uma vez, a pessoa pode se confundir. Mas se você dividir o trabalho em etapas menores e explicar cada uma com clareza, fica mais fácil de entender, certo? É exatamente isso que o **princípio da Divisão do Trabalho** faz na Engenharia de *Prompts*.

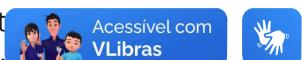
A ideia central da divisão do trabalho é bastante intuitiva, refletindo uma prática comum em engenharia: a decomposição de problemas complexos em partes menores e mais gerenciáveis. No contexto da engenharia de *prompts*, isso se traduz em dividir uma tarefa complexa, que exige várias etapas de processamento de informação, em uma sequência de *prompts* menores e interdependentes. Cada *prompt* individual se concentra em uma parte específica da tarefa, e a saída de um *prompt* serve como entrada para o próximo, criando assim um fluxo de trabalho encadeado.

Em vez de pedir à IA para resolver um problema gigante de uma só vez, você pode separar esse problema em pequenas tarefas. Cada tarefa recebe um "comando" específico (o *prompt*), e a resposta de uma tarefa vira informação para a próxima, como um quebra-cabeça.

2.2.5.1 EXEMPLOS DE APLICAÇÃO DA DIVISÃO DO TRABALHO

A decomposição de tarefas é uma estratégia importante na engenharia de *prompt*, onde um problema complexo é dividido em etapas menores e mais gerenciáveis. Isso ajuda a criar *prompts* mais precisos e eficazes. Nas Figuras 20 e 21, a seguir, estão disponíveis dois exemplos práticos de como essa abordagem pode ser aplicada.

Figura 20 - Criação de um *prompt* para geração de resumos de textos longos



Exemplo 1: Criação de um *Prompt* para Geração de Resumos de Textos Longos

Vamos considerar o desafio de criar um *prompt* para que uma IA gere resumos de textos longos de forma clara e concisa. A tarefa pode ser decomposta da seguinte maneira:

**1 Identificação dos pontos principais**

O primeiro passo é criar um *prompt* que instrua a IA a identificar os pontos principais do texto. Por exemplo: "Identifique os três pontos principais abordados neste texto." Isso direciona a IA a focar nas informações mais relevantes.

**2 Elaboração de resumos parciais**

Com os pontos principais identificados, o próximo *prompt* pode pedir à IA para elaborar pequenos resumos de cada ponto principal. Um exemplo de *prompt* seria: "Resuma cada um dos pontos principais em uma ou duas frases." Isso permite que o texto seja dividido em seções menores, facilitando o processamento.

**3 Combinação e refinamento do resumo**

Após a elaboração dos resumos parciais, o *prompt* final pode instruir a IA a combinar esses resumos em um texto único e coeso. Um exemplo seria: "Combine os resumos em um único parágrafo coeso que capture a essência do texto original."

**4 Revisão e ajustes finais**

Finalmente, um *prompt* adicional pode ser utilizado para revisar o resumo e ajustar a clareza e a concisão. Por exemplo: "Revise o parágrafo para garantir que esteja claro e conciso, e remova qualquer informação desnecessária."

Fonte: autoria própria.

Essa decomposição em etapas permite que a IA execute a tarefa complexa de maneira mais eficaz e controlada, resultando em resumos de alta qualidade (Figura 21).

Figura 21 - Desenvolvimento de um assistente virtual para responder perguntas complexas



Acessível com VLBras

**Exemplo 2: Desenvolvimento de um Assistente Virtual para Responder Perguntas Complexas**

Vamos imaginar que o objetivo seja criar um assistente virtual que responda perguntas complexas, como "Como funciona a mudança climática?". A tarefa pode ser decomposta da seguinte forma:

**1 Divisão da pergunta em subperguntas**

O primeiro *prompt* pode instruir a IA a dividir a pergunta complexa em subperguntas mais simples.

- Por exemplo: "Divida a pergunta 'Como funciona a mudança climática?' em três subperguntas mais simples.
 - "O que é efeito estufa?"
 - "Quais são os principais gases de efeito estufa?"
 - "Como o aquecimento global afeta o clima?".

**2 Resposta a cada subpergunta**

Para cada subpergunta, um novo *prompt* pode ser criado para gerar respostas detalhadas.

- Por exemplo: "Explique o que é o efeito estufa." e assim por diante para cada subpergunta.

**3 Integração das respostas**

Com as respostas às subperguntas prontas, o próximo passo é integrá-las em uma única resposta coesa.

- Um exemplo de *prompt* seria: "Combine as respostas às subperguntas em uma explicação coesa de como funciona a mudança climática."

**4 Simplificação e ajuste da resposta final**

Finalmente, pode-se utilizar um *prompt* para simplificar e ajustar a resposta final, garantindo que ela seja acessível para o público-alvo.

- Por exemplo: "Simplifique a resposta para que seja compreensível por estudantes do ensino médio."

Fonte: autoria própria.

2.2.5.2 POR QUE DIVIDIR O TRABALHO AJUDA A IA?

Pense em como ensinamos algo novo a uma criança. Se a gente apresentar um monte de informação complicada de uma vez só, ela provavelmente vai ficar confusa e desistir. Mas se dividirmos a tarefa em etapas menores, explicando cada passo com paciência, a criança terá muito mais chances de aprender e se sentir motivada.

Com a IA, a lógica é a mesma! A "Divisão do Trabalho" funciona como um guia para a IA, tornando tarefas complexas muito mais fáceis de entender e resolver. Em vez de enfrentar um desafio gigante de uma vez só, a IA pode focar em cada etapa com atenção, o que traz uma série de benefícios:

- **Melhora da compreensibilidade:** ao decompor um problema complexo em partes menores, o modelo de IA pode processar e entender as informações de forma mais eficaz.
- concentrando-se em uma única tarefa por vez, o modelo tem maior probabilidade de gerar resultados precisos e consistentes, reduzindo a ocorrência de "alucinações".
- **Maior controle e visibilidade:** a divisão do trabalho torna o processo de interação com a IA mais transparente, permitindo que os usuários acompanhem o raciocínio da IA passo a passo e identifiquem as causas de erros.
- **Possibilidade de reutilização:** os *prompts* individuais, desenvolvidos para cada etapa da tarefa, podem ser reutilizados em outros contextos, otimizando o processo de desenvolvimento e ampliando as possibilidades de aplicação da IA.



Dividir o trabalho é como dar um mapa para a IA, guiando-a passo a passo para encontrar soluções. Essa técnica funciona para diversas tarefas, desde escrever histórias criativas até resolver problemas complexos, reduzindo as tarefas em outras menores facilmente solucionadas pelos modelos.

NOTEBOOK COLAB:

https://colab.research.google.com/drive/1QZnRnb5h6COd_uHCSmNI3Gs4aEV4fixN

SAIBA MAIS...



Livros:

- *Prompt Engineering for Generative AI* de James Phoenix e Mike Taylor: o livro aborda estratégias e princípios de engenharia de *prompts* para diferentes modelos de IA, incluindo o ChatGPT.
- *Deep Learning for Natural Language Processing* de Stephan Raaijmakers: o livro fornece uma base para o campo de NLP e como os *prompts* desempenham um papel nele.

Artigos e publicações:

- Explore o arXiv.org, um repositório de artigos científicos, para encontrar pesquisas recentes sobre engenharia de *prompts* e tópicos relacionados.

Ferramentas online:

- Explore o OpenAI *Playground* para testar diferentes *prompts* no modelo GPT-4 e entender melhor como eles funcionam.



Comunidades online:

- Participe de fóruns e comunidades online, como o fórum do Hugging Face e o OpenAI Community, para discutir ideias e aprender com outros especialistas.



PARA RELEMBRAR...

Nessa Unidade, exploramos o conceito de Engenharia de *Prompts* e como isso pode ajudá-lo a otimizar sua interação com LLMs como o ChatGPT. Vamos relembrar os principais pontos:

- Engenharia de *prompts*: processo de elaboração de *prompts* eficientes para obter resultados desejáveis de um modelo de IA. É o processo de projetar e refinar a entrada (o *prompt*) que você fornece a um LM como o ChatGPT para guiá-lo a gerar resultados úteis e relevantes.
- *Prompt*: instruções que damos a um LLM para que ele possa gerar uma resposta adequada, ou seja, uma instrução ou conjunto de instruções que é fornecido a um LM para realizar uma tarefa. Pode ser uma pergunta, uma instrução, um exemplo de entrada e saída ou uma combinação desses.
- Clareza e Especificidade: *prompts* eficazes são claros, concisos e específicos. Evite ambiguidades e direcione o modelo exatamente para o que você espera como resultado.
- Contexto: forneça contexto suficiente para o modelo entender o que você está buscando. Isso pode incluir exemplos, informações sobre o público-alvo e restrições de formato.
- Formato: especifique o formato da resposta desejada, seja uma lista, um parágrafo ou um código.
- Exemplos: inclua exemplos relevantes para ilustrar o que você espera do modelo.
- Divisão do trabalho: para tarefas complexas, divida-as em etapas menores e crie *prompts* específicos para cada etapa, para guiar a IA mais eficazmente.
- Avaliação e iteração: é importante avaliar continuamente a qualidade das saídas geradas e ajustar os *prompts* com base nos resultados para melhorar a performance do modelo.





Grandes Modelos
de Linguagem

Unidade III
*In-context learning
e fine-tuning em LLMs*



UNIDADE III - IN- CONTEXT LEARNING E

FINE-TUNING EM LARGE LANGUAGE MODELS

3.1 IN-CONTEXT LEARNING

Quando falamos de aprendizado em *machine learning*, geralmente pensamos em um processo que envolve a exposição do modelo a uma grande quantidade de dados rotulados, seguido de um longo processo de otimização para ajustar os pesos internos do modelo. O *in-context learning* permite que os LMs aprendam e executem novas tarefas apenas a partir de exemplos fornecidos no contexto da própria interação, sem qualquer atualização explícita dos parâmetros internos. Em vez de ajustar seus parâmetros internos para cada tarefa, os LLMs se baseiam em exemplos fornecidos. Em outras palavras, o modelo "aprende" a tarefa durante a interação, utilizando os exemplos fornecidos para generalizar e prever a próxima saída. Este é um tipo de aprendizado dinâmico e temporário, onde o conhecimento é aplicado dentro do contexto atual e não retido após a sessão.

Os LLMs são pré-treinados em vastos conjuntos de dados para aprender as estatísticas subjacentes da linguagem. Durante o uso, quando um exemplo de tarefa é apresentado ao modelo, ele utiliza seu conhecimento pré-existente e o contexto fornecido para gerar uma resposta apropriada. Por exemplo, em vez de ser treinado com um conjunto de dados de tradução inglês-português, você fornece alguns exemplos de pares de frases inglês-português e o LLM é capaz de traduzir novos textos com precisão. É como se o LLM aprendesse a tarefa observando os exemplos fornecidos.

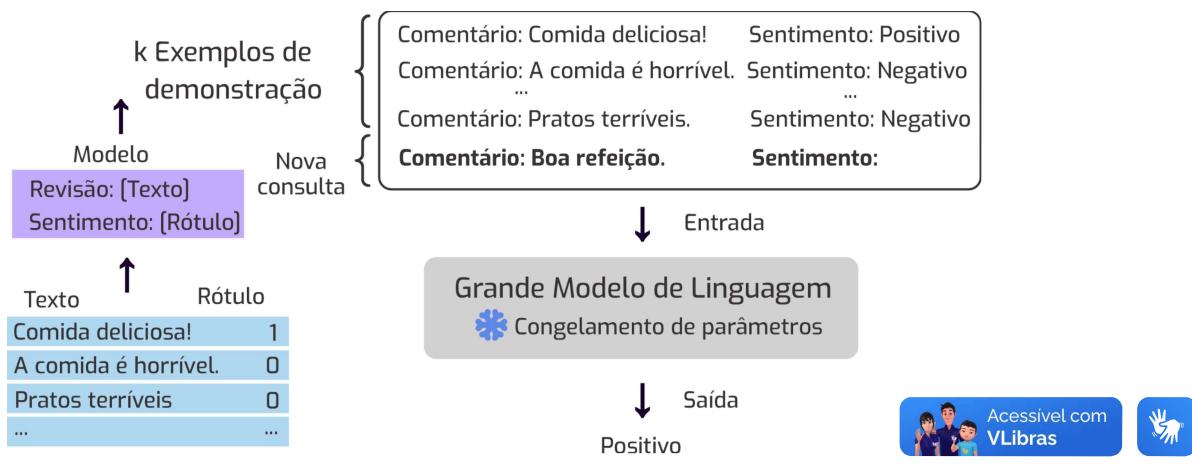
Na figura 22, vemos um exemplo de como o ICL funciona. O modelo é apresentado a alguns exemplos de demonstração, como "Comentário: Comida Deliciosa! Sentimento: Positivo" e "Comentário: A comida é horrível. Sentimento: Negativo". Esses exemplos são como o nosso amigo "estudando" a tarefa de classificar sentimentos. O modelo então recebe uma nova frase, como "Comentário: Boa refeição!", e deve prever o sentimento com base nos exemplos anteriores. O maior diferencial é que o modelo não é treinado com esses exemplos de forma tradicional, com atualizações de parâmetros. Em vez disso, ele "aprende" com a analogia, buscando padrões nos exemplos fornecidos e aplicando-os à nova frase.



É como se o modelo estivesse "lendo" as frases de demonstração e, a partir delas, construindo um entendimento do que define um sentimento "positivo" ou "negativo". É uma forma de aprendizado mais intuitiva e próxima ao raciocínio humano, sem a necessidade de um treinamento formal.

Na Figura 22, a seguir, apresentamos a ilustração do aprendizado em contexto que requer um contexto de *prompt* contendo alguns exemplos de demonstração escritos em LMs natural. Ao receber este *prompt* e uma consulta como entrada, os LLMs são responsáveis por fazer as previsões.

Figura 22 - Ilustração do aprendizado em contexto, onde Large Language Models usam *prompts* com exemplos para fazer previsões



Os exemplos podem ser usados para diversas tarefas, como resolução de problemas matemáticos, sem a necessidade de serem treinados especificamente para cada uma delas. O ICL é uma forma de interagir com os LLMs, utilizando a linguagem natural como interface e aproveitando a capacidade de aprendizado por analogia que os modelos demonstram.

Suponha que você tem um amigo que aprende muito rápido. Você mostra a ele uma foto de um kiwi, fruta que ele nunca viu antes, e diz "Isso é um kiwi". Instantaneamente, ele consegue identificar outras fotos de kiwis e até descrever seu sabor exótico. Essa capacidade de aprender com pouca informação prévia é semelhante ao que chamamos de *in-context learning* em LLMs. Assim como nosso amigo, os LLMs podem ser ensinados a realizar novas tarefas sem longos treinamentos.

O *in-context learning* descreve o processo interno da rede neural, pelo qual um LM, pré-treinado de forma não-supervisionada, utiliza as habilidades e os padrões reconhecidos durante o pré-treinamento para se adaptar rapidamente ou reconhecer a tarefa desejada ao receber uma nova sequência de entrada (Brown, et al., 2020). Embora o pré-

treinamento exponha o modelo a um conjunto vasto de dados, o aprendizado *in-context* acontece no momento da inferência, permitindo ao modelo inferir a tarefa a ser realizada baseando-se apenas no contexto fornecido dentro da própria sequência de entrada, que pode conter sub-tarefas repetidas.

In-context learning e *few-shot learning* estão intimamente relacionados porque ambos envolvem o uso de exemplos fornecidos no contexto da entrada para guiar a resposta do modelo, sem a necessidade de um novo treinamento. *In-context learning* é o processo mais amplo, no qual o modelo utiliza informações disponíveis no contexto para responder a uma tarefa específica.

No *few-shot learning*, o modelo recebe alguns exemplos concretos da tarefa que precisa realizar. Esses exemplos, não vistos no treinamento do modelo, são suficientes para que ele entenda o padrão de resposta esperado. Isso faz com que o *few-shot learning* funcione como uma adaptação capaz de melhorar a resposta do modelo ao situá-lo em um contexto com exemplos relevantes, mas sem precisar de um ajuste persistente nos parâmetros do modelo, ou seja, sem *fine-tuning*.

Com alguns exemplos, o modelo pode reconhecer padrões e realizar a tarefa com maior precisão. Quanto mais exemplos ele recebe, melhor ele se torna em generalizar e aplicar o conhecimento a novas situações. Na Figura 23, a seguir, é apresentado um cenário de *few-shot learning*.

Figura 23 - Cenário de exemplo de *few-shot learning*

Exemplo:

Para ter certeza que seu amigo entende o conceito de frutas cítricas, você mostra a ele fotos de limões, laranjas e limas, dizendo "Todas essas frutas são cítricas e têm um sabor azedo". Agora, ele reconhecerá facilmente outras frutas cítricas como as tangerinas. A inferência *few-shot* oferece ao LLM alguns exemplos (geralmente entre 3 e 5) para ilustrar a tarefa. Quanto mais exemplos ele recebe, melhor ele se sai, assim como nosso amigo, que se torna um especialista em frutas depois de ver diversos exemplos.



Fonte: autoria própria.

As maiores vantagens do *in-context learning* são (Dong *et al.*, 2023):

- **Flexibilidade:** permite que o LM se adapte a diferentes tarefas e domínios, sem a necessidade de retreinamento. É possível, por exemplo, usar o mesmo modelo para escrever um poema, traduzir um texto e responder a perguntas sobre um determinado assunto, sem precisar configurar cada tarefa individualmente.
- **Eficiência:** o aprendizado em contexto é um processo rápido e eficiente, pois não exige o treinamento completo do modelo para

cada nova tarefa. Isso é importante para aplicações em tempo real, onde a velocidade é um parâmetro necessário.

- **Personalização:** podemos personalizar o LM para se adaptar a necessidades específicas, fornecendo informações relevantes e exemplos específicos para a tarefa em questão. Por exemplo, um LM poderia escrever textos com o seu estilo particular, ou ser capaz de responder a perguntas sobre um tema específico que lhe interessa.

3.1.1 DESAFIOS E OPORTUNIDADES DO *IN-CONTEXT LEARNING*

Apesar de suas vantagens, o aprendizado in-contexto tem suas limitações. Primeiro, ele é altamente dependente da qualidade e quantidade dos exemplos fornecidos. Se os exemplos no contexto não forem representativos ou estiverem mal formulados, o modelo pode falhar em generalizar corretamente. Além disso, o aprendizado in-contexto é limitado pela capacidade do modelo de manter o contexto em sua memória, o que pode ser um desafio para tarefas complexas ou de longo prazo.

Para tarefas complexas que exigem uma compreensão profunda do contexto, o aprendizado em contexto pode não ser suficiente. Nesses casos, o LM pode precisar de treinamento específico para realizar a tarefa com precisão. Alguns trabalhos (Kaplan et al., 2020; MCguffie; Newhouse, 2020) indicam que modelos maiores, com mais parâmetros, tendem a ser melhores no aprendizado in-contexto. Esses modelos possuem uma maior capacidade de captar padrões complexos e generalizar a partir de poucos exemplos. No entanto, isso também levanta questões sobre custo computacional e financeiro, uma vez que treinar e executar modelos maiores requer mais recursos computacionais.

Apesar desses desafios, o *in-context learning* abre um universo de oportunidades para a IA. Essa habilidade permite que os LMs se adaptem rapidamente a novas tarefas e domínios, tornando-os ferramentas para diversas aplicações, desde a automação de tarefas repetitivas até a criação de novas formas de interação com a tecnologia.

O *in-context learning* é um campo de pesquisa em constante evolução, com novas descobertas e avanços sendo feitos a todo momento. Pesquisadores estão trabalhando para aprimorar a robustez e a generalização da ICL, tornando os LMs mais confiáveis e versáteis. Com o avanço da pesquisa, podemos esperar que o *in-context learning* se



torne uma característica cada vez mais importante nos LMs, permitindo que eles aprendam e se adaptem a um mundo em constante mudança.

3.2 FINE-TUNING EM LARGE LANGUAGE MODELS

O desenvolvimento de um LLM normalmente segue um processo dividido em estágios, conforme ilustrado na Figura 24. Inicialmente, o modelo é treinado em um corpus massivo de dados não supervisionados, abrangendo diversos tópicos e estilos de linguagem. Esse estágio permite que o modelo aprenda representações amplas da linguagem. Após essa fase, o modelo passa por um ajuste fino (*fine-tuning*), onde é treinado em conjuntos de dados mais específicos, direcionados a tarefas ou domínios particulares. Isso possibilita que o modelo refine suas capacidades para desempenhar instruções ou tarefas especializadas com maior precisão.

Figura 24 - Estágios de desenvolvimento de um *Large Language Model*



*Existem linhas de pesquisa para iniciar os pesos dos LLM. *Em modelos consolidados pode haver estágios com aprendizado por reforço. Fonte: autoria própria.

A principal diferença entre as respostas de um modelo base e um modelo ajustado para instruções (modelo *instruct*) está no comportamento e adequação às tarefas específicas, como ilustrado na Tabela 1. O modelo base, após o pré-treinamento em um grande corpus de dados não supervisionados, gera respostas genéricas baseadas em padrões estatísticos de linguagem. Ele pode completar frases, prever palavras e gerar textos coerentes, mas não é otimizado para seguir comandos específicos ou interagir de maneira direcionada.

Já um modelo *instruct* é ajustado por meio de *fine-tuning* em um conjunto de dados específico, no qual ele aprende a seguir instruções e fornecer respostas mais estruturadas e apropriadas ao contexto. O *instruct* geralmente entende melhor perguntas ou solicitações diretas e responde de forma mais clara e focada na tarefa. Enquanto o modelo base pode gerar uma resposta mais aberta ou ambígua, o modelo

instruct é projetado para ser mais útil e preciso, atendendo às necessidades específicas do usuário.

Tabela 1 - Comparativo entre respostas do modelo *Base* com o modelo *Instruct*

Modelo	Meta-Llama-3.1-8B	Meta-Llama-3.1-8B-Instruct
Prompt	Era uma vez	
Resposta	Era uma vez um rei que tinha um filho muito inteligente. O rei queria que seu filho se casasse com a filha de outro rei, mas o filho não queria. O rei ficou muito bravo e disse que se o filho não se casasse com a filha do outro rei, ele seria morto. O filho ficou muito triste e disse que não queria se casar com a filha do outro rei. O rei ficou ainda mais bravo e disse que se o filho não se casasse com a filha do outro rei, ele seria morto...	Era uma vez, em uma pequena cidade, um jovem chamado Pedro. Ele era conhecido por sua habilidade em resolver problemas e sua capacidade de ouvir as pessoas. Pedro era um ouvinte atento e um conselheiro sábio, e as pessoas da cidade vinham a ele com seus problemas e preocupações. Um dia, um homem idoso veio a Pedro e disse: "Meu filho, estou preocupado com a minha filha. Ela é uma jovem inteligente e talentosa..."
Prompt	O que é <i>fine-tuning</i> ?	
Resposta	O que é <i>fine-tuning</i> ? O que é <i>fine-tuning</i> ?...	O <i>fine-tuning</i> é um processo de treinamento de modelos de linguagem que envolve ajustar o modelo para realizar uma tarefa específica. Isso é feito adicionando camadas ou ajustando as camadas existentes do modelo para melhorar o desempenho em uma tarefa específica.

Fonte: autoria própria.

Acessível com
VLibras



Embora os modelos *instruct* sejam ajustados para seguir comandos específicos e fornecer respostas mais úteis, eles ainda estão limitados pela natureza dos dados com os quais foram treinados. Ou seja, mesmo após o ajuste fino para seguir instruções, o comportamento do modelo é altamente dependente dos padrões observados nos dados de treinamento, os quais raramente representam a complexidade e a diversidade dos cenários do mundo real. Pois, se o modelo foi ajustado para fornecer respostas baseadas em certos tipos de dados, ele pode falhar ao lidar com contextos inéditos ou com dados que fogem desses padrões. Por exemplo, em uma situação onde o contexto cultural ou técnico difere drasticamente dos dados fornecidos durante o *fine-tuning*, o modelo *instruct* pode fornecer respostas que, embora pareçam corretas, não são adequadas ou contextualizadas para a realidade específica do usuário.

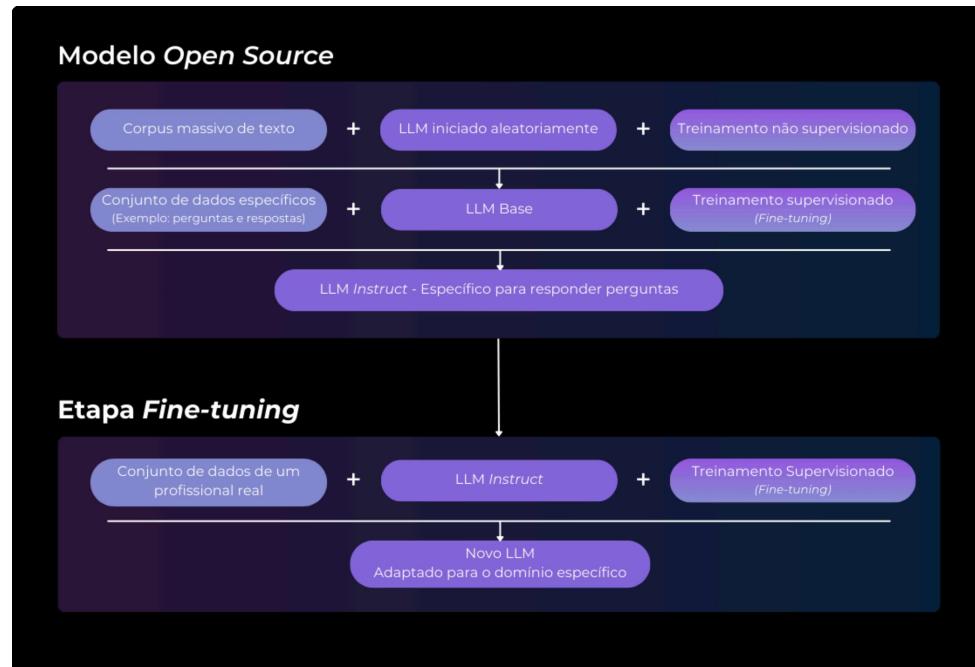
Assim como um estudante, um LLM pode responder a perguntas sobre diversos tópicos, mas não tem expertise em um assunto específico. O *fine-tuning* é como oferecer a esse estudante um curso de especialização, fornecendo-lhe dados específicos e instruindo-o a se concentrar em uma área particular.

No contexto dos LLMs, o *fine-tuning* consiste em treinar o modelo pré-treinado em um conjunto de dados menor e específico para a tarefa desejada. Esse processo ajusta os parâmetros do modelo para que ele se torne mais preciso na realização dessa tarefa específica. Por exemplo, se quisermos criar um LLM especializado em resumir textos científicos, podemos fazer *fine-tuning* em um conjunto de dados de artigos científicos, com resumos de alta qualidade.

Na Figura 25, é apresentado o processo de *fine-tuning* de um LLM usando um banco de conhecimento privado. O processo começa com o pré-treinamento de um LLM base em um grande conjunto de dados. Este modelo pré-treinado então passa por um processo de ajuste supervisionado usando um banco de conhecimento específico e privado. Essa etapa de ajuste adapta o LLM ao domínio ou tarefa específica definida pelos dados privados, resultando em um LLM melhor adequado para aplicações especializadas. O diagrama mostra as etapas envolvidas, destacando o papel do banco de conhecimento privado na personalização do LLM para necessidades específicas (Figura 25).



Figura 25 - Ajuste fino de um grande modelo de linguagem utilizando bases de dados privadas (conjunto de dados de um problema real)



Fonte: autoria própria.

Essa abordagem envolve a continuação do treinamento de um modelo já pré-treinado em um novo conjunto de dados que é relevante

para uma tarefa específica. A vantagem do *fine-tuning* é que ele permite que o modelo aproveite o conhecimento adquirido durante a fase inicial de pré-treinamento, economizando tempo e recursos em comparação ao treinamento de um modelo a partir do zero.

Durante o *fine-tuning*, o modelo é exposto a um novo conjunto de dados, que pode ser muito menor em comparação ao conjunto de dados utilizado no pré-treinamento. Isso permite ao modelo ajustar seus parâmetros de acordo com as peculiaridades da nova tarefa, resultando em um desempenho otimizado.

Uma outra aplicação importante do *fine-tuning*, além das tarefas gerativas mencionadas, é a adaptação do modelo para realizar tarefas específicas e complexas. Normalmente, os LLMs são conhecidos por suas habilidades gerativas, como escrever textos ou responder perguntas. No entanto, com o ajuste fino, é possível transformar o modelo em uma ferramenta voltada para tarefas não gerativas, como classificação.

Por exemplo, em vez de gerar respostas, o modelo pode ser treinado para realizar análise de sentimento, identificando se um texto apresenta emoções positivas, negativas ou neutras. Da mesma forma, o modelo pode ser ajustado para identificar e classificar a toxicidade em discurso, detectando automaticamente comportamentos prejudiciais, como discurso de ódio ou assédio.



Essas novas capacidades são alcançadas por meio da exposição do modelo a conjuntos de dados rotulados, onde cada exemplo de texto é associado a uma categoria ou rótulo específico. Durante o *fine-tuning*, o modelo ajusta seus parâmetros para aprender a associar padrões de linguagem a esses rótulos, transformando-se em uma ferramenta de classificação eficiente.

Essa flexibilidade mostra que os LLMs, embora conhecidos por suas capacidades gerativas, podem ser refinados para diversas tarefas especializadas, indo além da geração de texto e ampliando seu impacto em áreas como a análise de dados e a moderação de conteúdo.

3.2.1 MÉTODOS DE *FINE-TUNING*

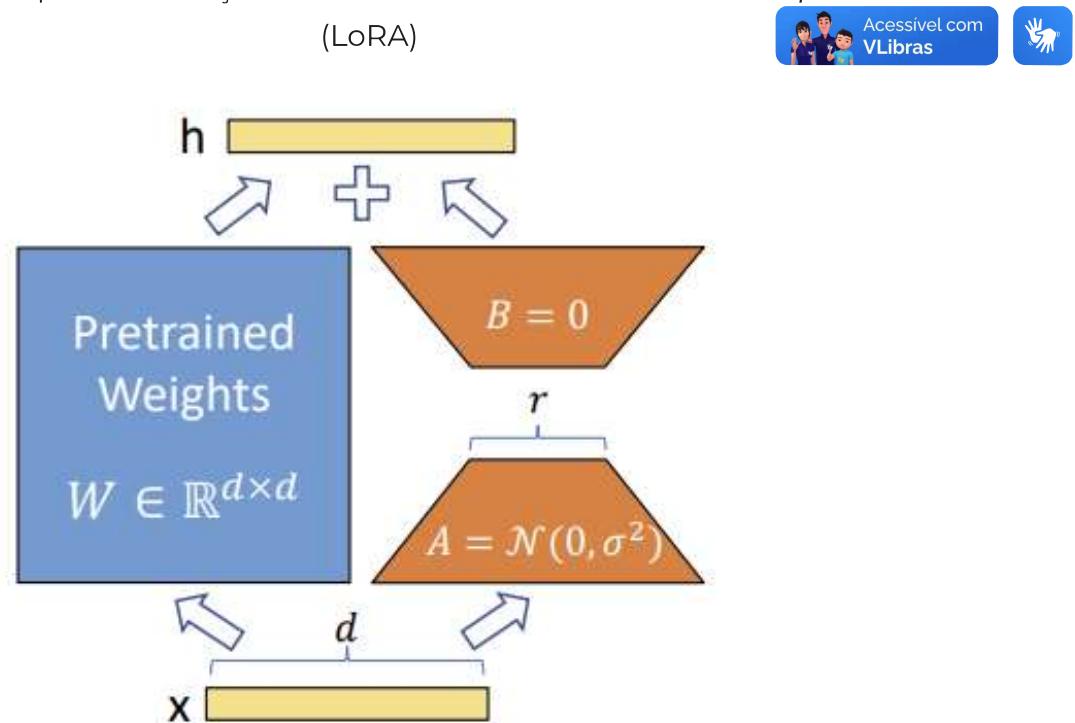
Recentemente, técnicas de ajuste fino mais otimizadas em termos de parâmetros, conhecidas como *Parameter-Efficient Fine-Tuning* (PEFT), têm ganhado destaque. Essas técnicas, como o *prefix fine-tuning*, *prompt-tuning* e *adapters*, envolvem a atualização de apenas um subconjunto de parâmetros do modelo, reduzindo a demanda por recursos computacionais. Métodos como *Low-Rank Adaptation* (LoRA) (Hu et al., 2022), foram desenvolvidos para otimizar os custos computacionais dos modelos.

LoRA é uma técnica desenvolvida para tornar o ajuste fino de LLMs mais apropriado em termos de uso de recursos computacionais. Em vez de atualizar todos os parâmetros do modelo durante o treinamento, o LoRA introduz uma abordagem que ajusta apenas um pequeno subconjunto de parâmetros que são de baixo custo computacional.

O conceito central do LoRA é a decomposição de matrizes em fatores de baixo *rank* (ou seja, menor dimensão) que representam as informações mais importantes do modelo. Em termos práticos, LoRA insere pequenas camadas lineares paralelas nas camadas originais do modelo. Durante o ajuste fino, apenas esses fatores de baixo *rank* são atualizados, enquanto os parâmetros originais do modelo permanecem congelados, como ilustrado na Figura 26. Isso reduz a quantidade de memória e computação necessárias para realizar o ajuste fino, sem comprometer o desempenho do modelo.

A principal ideia por trás do LoRA é adicionar um pequeno "adaptador" (um módulo de rede neural) ao LLM original. Esse adaptador é treinado para a tarefa desejada, como tradução de idiomas ou geração de texto criativo. O LLM base permanece intacto, mas o adaptador aprende a ajustar as saídas do LLM para a tarefa em questão.

Figura 26 - Reparametrização utilizando a Técnica Low-Rank Adaptation



Fonte: adaptada de Hu et al. (2022).

Na Figura 26, a matriz azul (*Pretrained Weights*) representa os pesos pré-treinados do modelo, que permanecem inalterados, enquanto as matrizes A e B são geradas e configuradas como treináveis.

No caso do LoRA, a adaptação não ocorre em todas as camadas do modelo, como é visível na Figura 26. Por exemplo, as matrizes de *embedding*, que atuam como uma "memória" do modelo, armazenando representações aprendidas de palavras ou dados de entrada durante o pré-treinamento, permanecem inalteradas. Essas matrizes carregam tanto a representação vetorial de cada *token* do vocabulário do modelo, quanto o conhecimento valioso que o modelo adquiriu ao ser treinado em grandes volumes de dados, como uma memória robusta sobre o entendimento geral do mundo. Alterar essas matrizes poderia causar o esquecimento de informações críticas — o que é conhecido como esquecimento catastrófico.

Por isso, LoRA intervém principalmente nas camadas de projeção linear, dentro da arquitetura de atenção. Nessa técnica, são inseridas matrizes de baixo *rank* (A e B, na Figura 26) que ajustam os pesos dessas camadas específicas, enquanto os pesos pré-treinados, representados pela matriz azul, permanecem intocados. Ao focar apenas nas camadas necessárias, o modelo pode se ajustar eficientemente a novas tarefas sem perder a "memória" de seu conhecimento prévio, garantindo uma adaptação eficiente sem comprometer a base aprendida no pré-treinamento.

Na Figura 27, há um comparativo da quantidade de parâmetros treináveis por camada no *fine-tuning* do modelo Llama 3.1, com 3 bilhões de parâmetros, utilizando LoRA com rank 128. Essa comparação ilustra de forma clara como o LoRA reduz o número de parâmetros ajustados em comparação com o *fine-tuning* tradicional, ao treinar apenas uma fração dos parâmetros.

Figura 27 - Comparativo da quantidade de parâmetros treináveis por camada no *fine-tuning* do Llama 3.1 na versão com 3 bilhões de parâmetros, utilizando *Low-Rank Adaptation* (LoRA) com rank 128



Fonte: autoria própria.

3.2.2 VANTAGENS E DESVANTAGENS DO *FINE-TUNING*

O *fine-tuning* oferece diversas vantagens, algumas delas podem ser observadas na Figura 28, a seguir.

Figura 28 - Vantagens e desafios do *fine-tuning*



Vantagens e desvantagens do fine-tuning

O fine-tuning oferece diversas vantagens:

Mas, o Fine-Tuning também apresenta alguns desafios:



Especialização:

permite criar LLMs que se destacam em tarefas específicas, como tradução, geração de código ou escrita criativa.



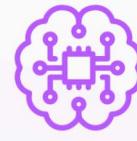
Personalização:

adapta o modelo para atender às necessidades de um usuário ou organização particular.



Melhor desempenho:

melhora a precisão e a qualidade das saídas do modelo em tarefas específicas.



Custos computacionais:

o processo de Fine-Tuning exige grande quantidade de memória e recursos computacionais, especialmente para grandes modelos de linguagem.



Necessidade de dados:

para um Fine-Tuning eficaz, é necessário um conjunto de dados específico e de alta qualidade para a tarefa desejada.



Risco de overfitting:

se o conjunto de dados for muito pequeno ou não representativo, o modelo pode adaptar muito bem a ele, mas apresentar um desempenho ruim em dados novos.



Acessível com
VLibras



Fonte: autoria própria.

NOTEBOOK COLAB:

<https://colab.research.google.com/drive/15I-avFQIaKOGfYJ7fZr7juAfHsk6GNQa>



SAIBA MAIS...

Artigos científicos:

- "Language Models are Few-Shot Learners" (Brown, et al., 2020): este artigo introduziu o conceito de aprendizado *in-*

context e demonstrou a capacidade dos LLMs de generalizar a partir de poucos exemplos.

- "LoRA: Low-Rank Adaptation of Large Language Models" (Hu et al., 2022): explore a técnica LoRA para *fine-tuning* de LLMs, reduzindo custos computacionais.
- Pesquise por artigos sobre "*in-context learning*", "*few-shot learning*", "*fine-tuning*" e "*parameter-efficient fine-tuning*" no arXiv.org para se manter atualizado sobre as pesquisas mais recentes.

Artigos do Medium:

- Desmistificando o *fine tuning* de LLMs na prática: PEFT, LoRA, QLORA e Hambúrgueres
- *Low-Rank Adaptation (LoRA): From intuition to implementation to interview questions*
- Minds and Machines — AI for mental health support, fine-tuning LLMs with LoRA in practice
- *Fine-tuning LLMs using LoRA*
- *Creating your own ChatGPT: a guide to fine-tuning LLMs with LoRA*
- *LoRA for fine-tuning LLMs explained with code example*



PARA RELEMBRAR...

Nessa Unidade, exploramos duas importantes técnicas para adaptar e otimizar LLMs: In-Context Learning e Fine-tuning. Aqui estão os principais conceitos:

- ICL: capacidade do LLM de aprender novas tarefas a partir de exemplos fornecidos durante a interação, sem atualização explícita dos parâmetros. O modelo aprende com os exemplos mostrados, como se estivesse sendo treinado no momento.

- *Zero-Shot Learning*: o modelo realiza uma tarefa sem nenhum exemplo prévio, apenas com seu conhecimento pré-treinado.
- *One-Shot Learning*: o modelo recebe apenas um exemplo para aprender e se adaptar à tarefa.
- *Few-Shot Learning*: o modelo recebe poucos exemplos (geralmente 3 a 5) para aprender a tarefa.
- *Fine-tuning*: processo de treinamento de um modelo pré-treinado em um conjunto de dados menor e específico para uma tarefa, ajustando seus parâmetros para otimizar o desempenho nessa tarefa. É como especializar o modelo em uma área particular.
- LoRA: técnica que adiciona pequenas camadas lineares ao LLM original, treinando apenas essas camadas e mantendo o resto do modelo congelado.



Unidade IV
Retrieval
Augmentation
Generation (RAG)



UNIDADE IV - RETRIEVAL

AUGMENTATION GENERATION

Até agora, exploramos como os LLMs podem gerar texto, aprendendo padrões complexos a partir de dados massivos. Existe um desafio inerente a essa abordagem: a informação que o modelo possui está limitada ao que foi armazenado durante seu treinamento. Por exemplo, um LLM treinado antes de 2020 não teria conhecimento da pandemia de COVID-19. Essa limitação pode ser problemática quando precisamos que o modelo acesse informações factuais precisas e atualizadas.

É aqui que entra a Geração Aumentada por Recuperação, ou RAG, do inglês *Retrieval Augmented Generation*. Essa técnica permite que os LLMs acessem e utilizem informações externas em tempo real, expandindo suas capacidades. O RAG pode ser comparado a um "cérebro" adicional para o LLM, um grande repositório de informações ao qual ele pode recorrer sempre que precisar.

4.1 COMO FUNCIONA O RETRIEVAL AUGMENTATION GENERATION?



Na Figura 29, é ilustrado o funcionamento de um sistema RAG. Começamos com um usuário que formula uma pergunta sobre o CEO do OpenAI, Sam Altman, e sua saída e retorno à empresa. O LLM, como um modelo pré-treinado, não tem acesso a informações atualizadas sobre esse evento. É aqui que o RAG entra em ação.

O sistema RAG realiza uma indexação de um conjunto de documentos, neste caso, artigos de notícias relevantes ao assunto da pergunta. Essa indexação envolve a divisão do texto do documento em vários pedaços, denominados *chunks*. Por meio de um processo de recuperação, o sistema identifica os *chunks* mais relevantes à pergunta do usuário.

Em seguida, o sistema combina o contexto da pergunta original com as informações extraídas dos documentos relevantes, formando um *prompt* mais completo para o LLM. Esse *prompt* enriquece o LLM com informações contextuais, permitindo que ele gere uma resposta mais precisa e completa.

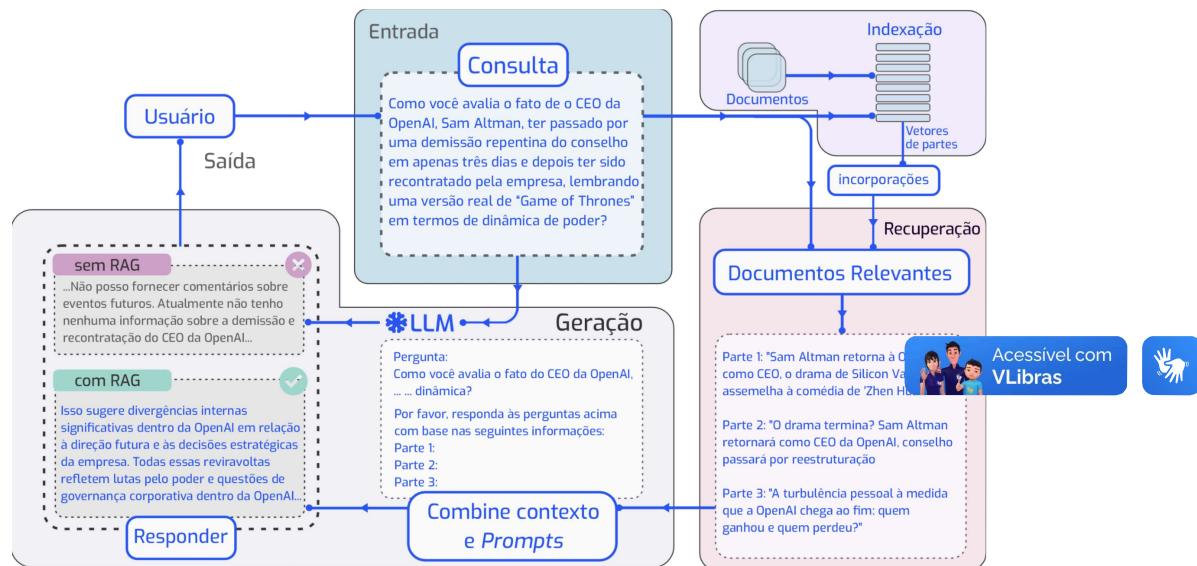
O sistema RAG, ao incorporar informações externas, amplia o conhecimento do LLM, tornando-o mais capaz de lidar com perguntas

complexas e atualizadas. Essa técnica permite que os LLMs possam acessar e processar informações de forma mais dinâmica.

Na Figura 29, é demonstrado como o processo de RAG pode ser dividido em três etapas, que correspondem a:

1. **Indexação:** documentos são divididos em trechos, codificados em vetores e armazenados em um banco de dados vetorial.
2. **Recuperação:** recupera os k principais trechos mais relevantes para a pergunta com base na similaridade semântica.
3. **Geração:** insere a pergunta original e os trechos recuperados em um LLM para gerar a resposta final.

Figura 29 - Visão geral de como funciona o Retrieval Augmentation Generation (RAG) com três etapas



Fonte: adaptada de Gao et al. (2023).

A indexação envolve a criação de um índice da base de dados, onde os documentos são organizados de maneira a facilitar a recuperação das informações relevantes. Os documentos são divididos em unidades menores (*chunks*), como frases ou parágrafos, e transformados em representações vetoriais (*embeddings*). Este processo envolve transformar o conteúdo textual (*chunks*) em representações vetoriais (*embeddings*) e armazenar estas representações vetoriais em um banco de dados. Os *embeddings* capturam o significado semântico do texto e são armazenadas em um índice, facilitando a busca por similaridade, em vez da busca textual.

O RAG inicia com o envio de uma consulta do usuário no formato textual. Quando o sistema recebe uma consulta, ele transforma a consulta em uma representação vetorial. O sistema precisa do texto da consulta no formato vetorial (*embeddings*) para conseguir buscar os *chunks* mais relevantes na base de dados vetorial, previamente

armazenada. Ou seja, o sistema compara os *embeddings* dos *chunks* dos documentos com o *embedding* da consulta para definir quais *chunks* possuem maior similaridade com a consulta. Estes *chunks* são recuperados e fornecidos ao LLM como contexto adicional.

A etapa de geração utiliza o LLM para formular a resposta final. O modelo combina o contexto recuperado com a consulta do usuário e gera uma resposta, aproveitando tanto o conhecimento externo quanto sua capacidade de processamento de linguagem natural. Leia o exemplo na Figura 30.

Figura 30 - Exemplo de cenário com uso de *Retrieval Augmentation Generation* (RAG)

Exemplo:

Podemos ter um cenário com um usuário perguntando: "Quem ganhou o Oscar de melhor ator em 2024?". O sistema RAG indexaria uma base de dados de cinema, como o IMDb (Internet Movie Database - base online sobre a indústria do entretenimento com dados sobre filmes, séries de TV, celebridades e outros). Na etapa de recuperação, o sistema buscaria por documentos relacionados a "Oscar de melhor ator" e "2024". Os documentos relevantes seriam recuperados e fornecidos ao LLM. A etapa de geração combinaria esses documentos com a consulta original e geraria a resposta: "Cillian Murphy ganhou o Oscar de melhor ator em 2024 por sua atuação em Oppenheimer".

Fonte: autoria própria.



4.2 ARQUITETURAS RETRIEVAL AUGMENTATION GENERATION

A evolução das arquiteturas de RAG tem acompanhado o avanço das técnicas de aprendizado profundo e das necessidades crescentes por respostas mais precisas e contextualmente relevantes. As arquiteturas podem ser classificadas em três categorias principais: **Naive**, **Avançada** e **Modular**. Cada uma dessas arquiteturas oferece diferentes abordagens para a integração de recuperação e geração, variando em complexidade e desempenho.

Na Figura 31, são mostradas três arquiteturas RAG diferentes: *naive RAG*, *advanced RAG* e *modular RAG*:

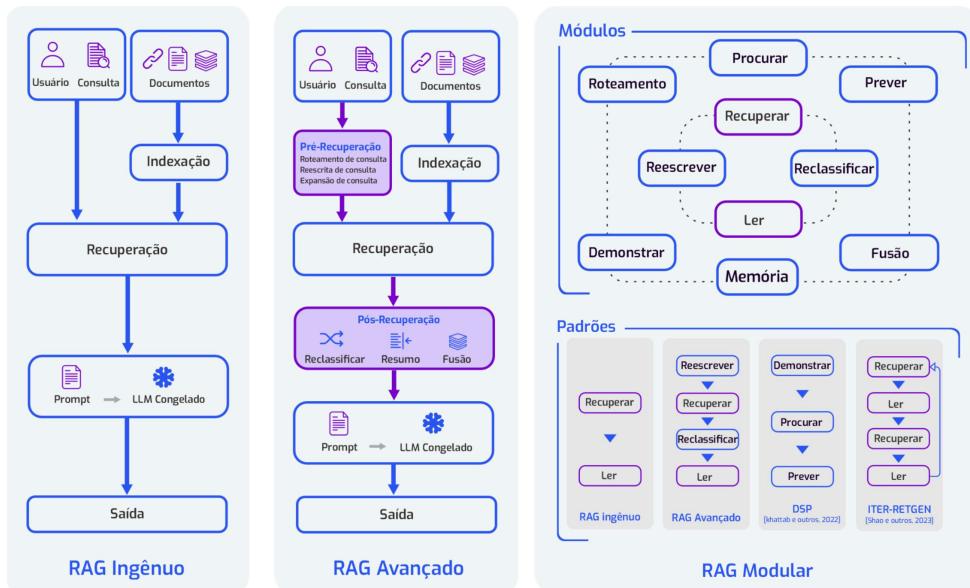
- *Naive RAG* é a arquitetura mais simples. Os documentos são indexados no formato vetorial e uma consulta é enviada para a etapa de recuperação. Os documentos recuperados são então usados para criar um *prompt* que é enviado para um LLM. O LLM então gera uma saída com base nas informações dos documentos recuperados.
- *Advanced RAG* adiciona uma etapa de pós-recuperação ao *pipeline*. Esta etapa pode incluir tarefas como re-ranqueamento,

sumarização e fusão, que ajudam a melhorar a relevância e a qualidade dos documentos recuperados. O *prompt* é então gerado com base nesses documentos aprimorados e enviados para o LLM para gerar a saída.

- *Modular RAG* oferece a arquitetura mais flexível e avançada, dividindo o sistema RAG em módulos reutilizáveis, como demonstração, pesquisa, leitura, recuperação, re-rankeamento, reescrita e previsão. Esses módulos podem ser montados em vários *pipelines* para atender a necessidades específicas. Por exemplo, o *pipeline* DSP (demonstração, pesquisa, previsão) mostrado na Figura 30 usa os módulos demonstração, pesquisa e previsão para gerar uma saída. A natureza modular do *modular RAG* permite uma maior flexibilidade e personalização em comparação com o *naive* e *advanced RAG*.

Na Figura 31, são demonstrados os três paradigmas dos modelos RAG. O *Naive RAG* (ou RAG ingênuo), mostrado à esquerda, consiste principalmente em três partes: indexação, recuperação e geração. O RAG avançado, exibido ao centro, propõe múltiplas estratégias de otimização em torno da pré-recuperação e pós-recuperação, mantendo um processo semelhante ao do Naive RAG, ainda seguindo uma estrutura em cadeia. O RAG modular, apresentado à direita, herda e evoluí do paradigma anterior, oferecendo maior flexibilidade geral.

Figura 31 - Comparação entre os três paradigmas do Retrieval Augmentation Generation (RAG)



Fonte: adaptada de Gao et al. (2023).

4.2.1 ARQUITETURA NAIVE

A arquitetura *Naive* é a forma mais básica e direta de implementação de um sistema RAG. Nesta abordagem, o processo de recuperação e geração é realizado de forma bastante simples e linear. O *pipeline* de um sistema *Naive* pode ser descrito da seguinte forma:

- **Recuperação simples:** quando uma consulta é realizada, o sistema busca documentos ou fragmentos de texto relevantes em uma base de dados usando técnicas de recuperação básicas, como a busca por palavras-chave ou modelos de recuperação tradicionais.
- **Geração direta:** após recuperar os documentos mais relevantes, o LM utiliza essas informações como contexto adicional para gerar a resposta. O processo é uma concatenação das informações recuperadas com a entrada original, seguida da geração do texto.

A arquitetura *Naive* possui diversas limitações. A eficácia do sistema depende diretamente da qualidade das informações recuperadas, que, por sua vez, está atrelada à capacidade limitada dos métodos de recuperação tradicionais. Além disso, a ausência de mecanismos de refinamento entre as etapas de recuperação e geração pode resultar em respostas que, embora baseadas em fatos, carecem de coerência ou precisão.



4.2.2 ARQUITETURA AVANÇADA

A arquitetura Avançada de RAG melhora o *pipeline* *Naive*, incorporando técnicas mais otimizadas tanto na recuperação quanto na geração. Esta arquitetura é projetada para otimizar o processo em todas as etapas, resultando em uma integração entre os componentes do sistema.

- **Recuperação:** nesta abordagem, a recuperação é realizada usando técnicas mais avançadas, como *embeddings* semânticos e modelos baseados em aprendizado profundo, como BERT ou variantes. Esses modelos são capazes de entender o contexto e a semântica das consultas, permitindo a recuperação de documentos que não apenas correspondem às palavras-chave, mas que são contextualmente relevantes.
- **Re-ranking e filtragem:** antes de passar as informações recuperadas para a etapa de geração, a arquitetura avançada realiza um processo de *re-ranking*. Essa etapa adicional reordena e refina os resultados da recuperação, filtrando aqueles que são mais relevantes e confiáveis. Esse *re-ranking* pode ser baseado

em critérios adicionais, como a confiança do modelo ou a relevância semântica.

- **Geração contextualizada:** com uma base de informações mais refinada e relevante, o LM é capaz de gerar respostas mais coerentes e precisas. A geração não é apenas uma simples concatenação; o modelo pode reescrever ou adaptar a informação recuperada para que ela se integre de maneira natural e contextualizada à resposta.

4.2.3 ARQUITETURA MODULAR

A arquitetura modular é a mais flexível e escalável das três abordagens. Ela trata as etapas de recuperação e geração como módulos independentes, que podem ser combinados ou substituídos conforme necessário. Essa modularidade permite uma otimização personalizada e a experimentação com diferentes técnicas e modelos em cada etapa do processo.

- **Modularidade na recuperação:** a recuperação pode ser realizada por um ou mais módulos especializados. Por exemplo, um módulo pode ser responsável pela recuperação de documentos baseados em *embeddings* semânticos, enquanto outros podem utilizar métodos de recuperação tradicionais ou até mesmo uma abordagem supervisionada. Esses módulos podem ser executados em paralelo ou em sequência, oferecendo uma flexibilidade sem precedentes na construção do *pipeline* de recuperação.
- **Integração dinâmica e *re-ranking*:** após a recuperação pelos diferentes módulos, uma etapa de integração dinâmica ocorre, onde os resultados são combinados e reordenados. Essa integração pode ser realizada usando técnicas de aprendizado que levam em conta a confiabilidade dos diferentes módulos e a relevância do contexto.
- **Geração modular:** a etapa de geração também é modular. Diferentes LMs podem ser utilizados dependendo do tipo de consulta ou da natureza das informações recuperadas. Por exemplo, um modelo mais simples pode ser utilizado para perguntas factuais diretas, enquanto um modelo mais avançado é empregado para consultas que exigem respostas mais elaboradas e contextualizadas.
- **Ajuste fino:** um dos aspectos mais importantes da arquitetura modular é a capacidade de realizar ajustes finos em tempo real. Isso significa que o sistema pode aprender com seus próprios



erros e melhorar continuamente seu desempenho, ajustando os módulos de recuperação e geração conforme necessário.

A arquitetura modular, embora mais complexa, oferece uma solução escalável e adaptável para sistemas RAG. Sua flexibilidade permite não apenas a otimização do desempenho, mas também a adaptação a diferentes domínios e tipos de consultas.

4.3 RETRIEVAL AUGMENTED GENERATION VS. FINE-TUNING

O *fine-tuning*, ou ajuste fino, é uma abordagem comum em LMs onde o modelo pré-treinado é adaptado para uma tarefa específica usando um conjunto de dados de treinamento adicional. Essa técnica tem suas limitações, principalmente em termos de flexibilidade e escalabilidade. Quando um modelo é ajustado para uma tarefa, ele pode perder a capacidade de se adaptar a novos contextos ou informações que surgem após o treinamento.

Em contraste, o RAG oferece uma solução mais dinâmica. Em vez de depender exclusivamente do conhecimento armazenado no modelo, que pode ficar obsoleto ou ser insuficiente, o RAG permite que o usuário accesse uma base de dados externa continuamente atualizada. Isso não só melhora a precisão das respostas, mas também elimina a necessidade de treinar e ajustar o modelo toda vez que novas informações surgem. Dessa forma, o RAG combina o melhor dos dois mundos: a capacidade gerativa dos LLMs com a atualidade e precisão das informações recuperadas em tempo real.

4.4 BENEFÍCIOS DO RETRIEVAL AUGMENTED GENERATION

O uso do RAG oferece vários benefícios. Primeiro, ele permite que os LMs accessem informações que podem estar além de sua capacidade de memória interna, especialmente em domínios que evoluem rapidamente, como ciência, tecnologia e notícias. Isso não só melhora a qualidade das respostas, mas também reduz a necessidade de treinar continuamente o modelo com novos dados, já que o modelo pode buscar informações atualizadas em tempo real.

Além disso, o RAG aumenta a transparência. Como as respostas geradas são baseadas em documentos específicos que podem ser



referenciados, é mais fácil para os usuários entenderem de onde as informações vieram, o que é essencial em contextos onde a veracidade e a fonte das informações são muito importantes.

4.5 DESAFIOS E LIMITAÇÕES

Apesar dos avanços significativos, o RAG ainda enfrenta desafios. A robustez é um dos principais desafios, com sistemas RAG sendo suscetíveis a informações ruidosas ou contraditórias presentes nas bases de conhecimento externas. A presença de informações irrelevantes pode prejudicar a qualidade da resposta final, levando a inconsistências e informações imprecisas.

Outro desafio é a escalabilidade dos sistemas RAG. À medida que as bases de conhecimento externas crescem em tamanho e complexidade, a velocidade da recuperação e a capacidade de processamento do LLM se tornam fatores críticos. Otimizar a indexação, a recuperação e a geração para lidar com grandes volumes de dados é importante para o sucesso do RAG em cenários do mundo real.

As questões éticas também são relevantes no contexto do RAG. A potencial presença de vieses e a necessidade de garantir a privacidade das informações são aspectos que exigem atenção e desenvolvimento de mecanismos de controle e mitigação. Garantir a confiabilidade e a ética dos sistemas RAG é indispensável para sua ampla adoção e impacto positivo na sociedade.



NOTEBOOK COLAB:

<https://colab.research.google.com/drive/1-QpVT3PzGkpHDXItCiK5DWc8UYwFL8bJ>



SAIBA MAIS...

Para aprofundar seu conhecimento sobre RAG, explore os seguintes recursos:

- Artigos e publicações:

- "*Retrieval-augmented generation for large language models: A survey*": apresenta uma visão geral bem detalhada dos principais trabalhos utilizando RAG.
 - Pesquise por artigos sobre "*Retrieval-Augmented Generation*", "RAG", "Knowledge Retrieval" e tópicos relacionados no arXiv.org e em outras plataformas de publicação científica. Mantenha-se atualizado sobre as pesquisas mais recentes e os avanços na área.
- **Bibliotecas e ferramentas:**
- *RAGFlow*: é um *framework* para construção de pipelines RAG de código aberto baseado em compreensão profunda de documentos. Ele oferece um fluxo de trabalho RAG simplificado para empresas de qualquer porte, combinando LLMs para fornecer recursos de resposta a perguntas precisas, com base em citações bem fundamentadas de vários dados formatados.
 - *LangChain*: framework para desenvolvimento de aplicações que utilizam LLMs. O *LangChain* possui diversas funcionalidades que simplificam a implementação de RAG e outras técnicas avançadas.



• **Artigos no Medium:**

- *RAG LLM Best Practices*
- *RAG - Retrieval-Augmented Generation*
- *A first intro to Complex RAG (Retrieval Augmented Generation)*
- *Implementing Retrieval Augmented Generation (RAG): A Hands-On Guide!*



PARA RELEMBRAR...

Nessa Unidade, exploramos o conceito de RAG, uma técnica que permite aos LLMs acessar e utilizar informações externas em

tempo real. Seguem os principais conceitos e termos:

- RAG: técnica que combina a geração de texto dos LLMs com a recuperação de informações de fontes externas, como bancos de dados ou documentos. O RAG permite que os LLMs acessem informações atualizadas e relevantes, tornando-os mais precisos e contextualmente relevantes.
- Indexação: processo de organizar a base de conhecimento externa para facilitar a recuperação de informações. Envolve a divisão dos documentos em *chunks* (pedaços dos documentos) e a criação de representações vetoriais (*embeddings*) para cada *chunk*.
- Recuperação: etapa em que o sistema busca os *chunks* mais relevantes para a consulta do usuário, com base na similaridade entre a representação vetorial da consulta e os *embeddings* dos *chunks*.
- Geração: processo em que o LLM gera uma resposta, utilizando a consulta do usuário e as informações recuperadas como contexto.
- Arquiteturas RAG: diferentes abordagens para implementar RAG, variando em complexidade e desempenho. As principais arquiteturas são: *Naive RAG*, *Advanced RAG* e *Modular RAG*.
- *Naive RAG*: arquitetura mais simples, onde a recuperação e geração são realizadas de forma linear.
- *Advanced RAG*: incorpora técnicas mais avançadas de recuperação e geração, como *embeddings* semânticos e *re-ranking*.
- *Modular RAG*: divide o sistema RAG em módulos independentes e reutilizáveis, oferecendo maior flexibilidade e personalização.
- *Chunks*: pedaços menores de texto em que os documentos são divididos durante a indexação.
- *Embeddings* (representações vetoriais): representações numéricas que capturam o significado semântico do texto. São utilizadas para realizar a busca por similaridade na etapa de recuperação.





Grandes Modelos
de Linguagem

Unidade V
**Integração e
Escalabilidade de
Soluções com LLMs**



UNIDADE V - INTEGRAÇÃO E ESCALABILIDADE DE

SOLUÇÕES COM *LARGE LANGUAGE MODELS*

Para aproveitar as capacidades de LLMs em aplicações do mundo real, é importante saber como integrar esses modelos em sistemas existentes e garantir que eles possam escalar à medida que a demanda cresce. A escalabilidade refere-se à capacidade de um sistema de lidar com um aumento de carga – seja no número de usuários, na quantidade de dados processados ou na complexidade das tarefas realizadas – sem comprometer o desempenho. Já a integração envolve o processo de incorporar o LLM em um sistema ou fluxo de trabalho existente, assegurando que ele funcione de maneira integrada com outros componentes.

Esta Unidade abordará esses conceitos, começando pela definição e importância da escalabilidade, seguida por estratégias práticas de integração e, finalmente, discutindo os desafios para disponibilizar soluções baseadas em LLMs para os usuários finais. O objetivo é fornecer uma compreensão clara e aplicável desses temas, preparando o leitor para implementar soluções robustas e escaláveis utilizando LLMs.

5.1 ESCALABILIDADE DE SOLUÇÕES *LARGE LANGUAGE MODELS*



A escalabilidade é um fator importante quando se trabalha com LLMs, principalmente porque esses modelos exigem uma grande quantidade de recursos computacionais para funcionar de maneira eficaz. Quando falamos em escalar uma solução baseada em LLMs, estamos nos referindo à capacidade de expandir o sistema para atender um número maior de usuários ou processar volumes crescentes de dados.

Para que uma solução seja escalável, é necessário considerar várias questões técnicas desde o início. Primeiro, é importante entender que LLMs são modelos de grande porte que requerem considerável poder de processamento. Por isso, uma das estratégias para escalar esses modelos envolve o uso de Unidades de Processamento Gráfico (GPUs) ou Unidades de Processamento Tensor (TPUs), que são mais adequadas para operações intensivas de cálculos, como as que esses modelos realizam.

Entretanto, a escalabilidade não é apenas uma questão de infraestrutura. Há desafios operacionais que precisam ser enfrentados, como o balanceamento de carga, que garante que nenhuma parte do sistema fique sobrecarregada, e a latência, que deve ser minimizada para proporcionar uma boa experiência ao usuário final. Também é importante considerar os custos associados à escalabilidade, já que o aumento de capacidade muitas vezes vem acompanhado de maiores despesas com hardware, energia e manutenção.

5.2 CAPACIDADE COMPUTACIONAL PARA ADOÇÃO DE *LARGE LANGUAGE MODELS* NAS EMPRESAS

A adoção de LLMs em soluções empresariais exige um planejamento da infraestrutura computacional. Dimensionar incorretamente pode levar a custos excessivos ou performance inadequada, prejudicando o retorno do investimento. Esta seção aborda aspectos fundamentais para essa avaliação, incluindo métricas de capacidade computacional, *benchmarking* de desempenho e custos operacionais, bem como as principais estratégias para otimizar a infraestrutura necessária para adotar LLMs em empresas de diferentes portes e setores.



5.2.1 MÉTRICAS DE CAPACIDADE COMPUTACIONAL PARA *LARGE LANGUAGE MODELS*

A capacidade computacional para suportar LLMs pode ser medida em termos de FLOPs, consumo de memória das GPUs e latência de inferência.

FLOPs são uma métrica central para quantificar a complexidade computacional dos modelos e, consequentemente, o hardware necessário. Estimar o número de FLOPs pode ajudar a dimensionar o número de GPUs necessárias, o que é especialmente útil para empresas que pretendem rodar LLMs *on-premise*. O modelo GPT-3, por exemplo, possui 175 bilhões de parâmetros e necessita de centenas de petaflops para treinamento eficiente, enquanto modelos menores podem ser mais acessíveis em termos computacionais (Shoeybi et al, 2019).

A **latência** é uma métrica importante para os usuários da sua aplicação, especialmente em aplicações de atendimento ao cliente ou e-commerce em que o seu cliente não quer esperar dezenas de segundos ou minutos por uma resposta. Estudos mostram que a latência aumenta significativamente com o tamanho do modelo e a quantidade de dados

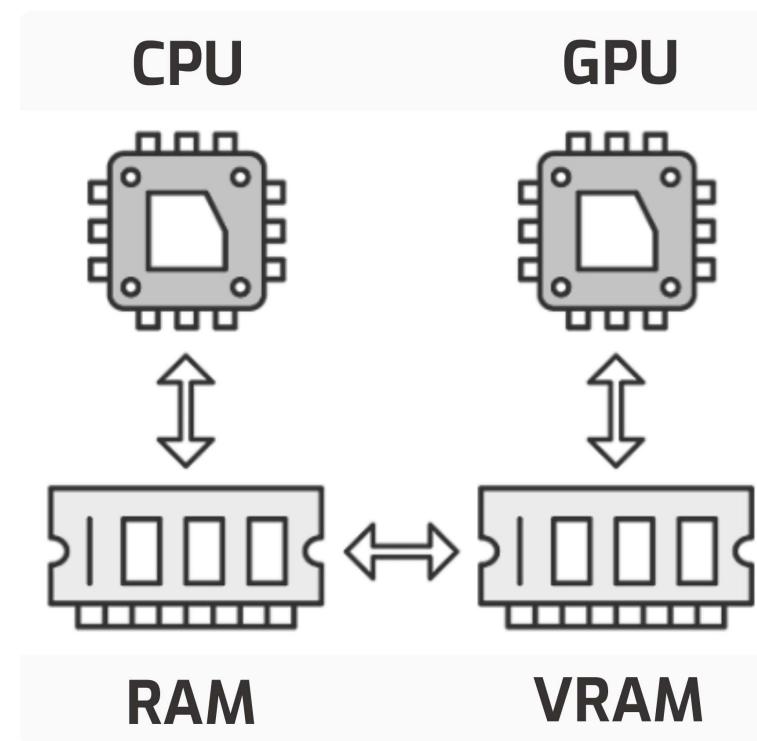
processados (Brown *et al.*, 2020), sugerindo que empresas com alto volume de transações devem considerar arquiteturas otimizadas para inferência. Ou seja, as empresas devem se preocupar com a qualidade da resposta das LLMs (acurácia da inferência) e também com o tempo de resposta das LLMs (latência). Modelos menores tendem a ter uma latência menor e uma acurácia pior do que modelos grandes, mas podem atender bem a cenários de usuários reais sem demandar o mesmo nível de capacidade de memória e processamento.

Outra métrica muito importante é o quanto de **memória de GPUs** você precisa para ter uma LLM rodando na sua organização. Em LLMs, a memória de GPU é utilizada para armazenar tanto os parâmetros do modelo quanto os dados temporários necessários para realizar operações de inferência e treinamento. À medida que o número de parâmetros aumenta — como nos modelos de bilhões de parâmetros, como o GPT-3 —, a demanda por memória cresce, exigindo GPUs com capacidades de VRAM elevadas.

Video Random Access Memory (VRAM) é uma memória de alta velocidade diretamente integrada às GPUs, permitindo que ela acesse dados rapidamente, sem precisar recorrer constantemente à memória da CPU. Na Figura 32, é apresentado um exemplo da integração da memória da GPU e da CPU. Quando a GPU precisa de dados que residem na memória da CPU, esses dados precisam ser copiados para a VRAM por meio de um barramento, representado pelas setas bidirecionais. Esse processo de cópia introduz, geralmente, um atraso no acesso aos dados. Por isso, os modelos são carregados diretamente na VRAM e quanto maior a quantidade de VRAM, mais dados e modelos maiores uma GPU pode processar simultaneamente.

Figura 32 - Arquitetura simplificada de memória da *Central Processing Units (CPU)* e *Graphics Processing Units (GPU)*





Fonte: adaptada de Pickle (2023).

Também é possível dividir um grande LM entre várias GPUs, mas ainda assim, configurações de memória inadequadas podem levar a problemas de **falta de memória**, interrompendo o processamento. Estratégias como quantização são usadas para otimizar o consumo de memória, possibilitando que LLMs sejam executados em GPUs com menores capacidades sem comprometer muito o desempenho.



5.2.2 IMPLANTAÇÃO DO LARGE LANGUAGE MODEL

Uma pergunta comum entre profissionais que estão iniciando a implantação de LLMs nas empresas é: “Como mensurar a capacidade computacional necessária para adotar LLMs em soluções na minha organização?”. Você deve considerar diversos fatores que influenciam diretamente no desempenho, nos custos e na viabilidade dessa tecnologia.

Na Figura 33, é apresentada uma visão geral de quatro abordagens para colocar em produção soluções integradas com LLMs, com uma relação entre custo/dificuldade e qualidade dos resultados.

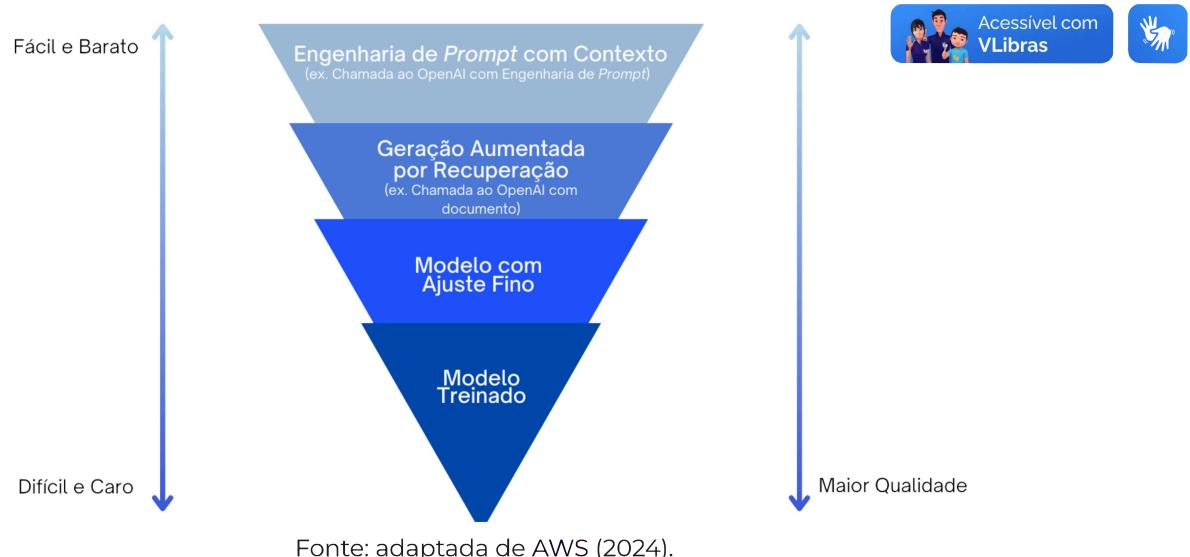
1. **Engenharia de *prompt* com contexto:** essa é a opção mais acessível e simples, onde *prompts* bem estruturados são enviados para APIs (como OpenAI) para melhorar a qualidade das respostas.
2. **RAG:** acrescenta documentos relevantes como contexto ao LLMs para aumentar a precisão das respostas.
3. **Modelos com ajuste fino:** realiza uma customização específica para tarefas específicas, com treinamento em dados próprios,

melhorando a qualidade mas aumentando custos em relação a técnicas sem ajuste fino.

4. **Treinamento de modelos:** é a opção mais cara e complexa, com o modelo treinado inteiramente do zero, oferecendo total controle da qualidade do modelo.

Você pode iniciar sua solução utilizando engenharia de *prompt* e APIs comerciais, como o GPT-4 da OpenAI e o Gemini da Google. O uso de APIs comerciais permite acesso a LLMs de alto desempenho sem a necessidade de infraestrutura própria. Essa solução terceirizada, como as oferecidas pelo OpenAI e Gemini, minimiza custos de manutenção e hardware, garantindo alta escalabilidade. É ideal para empresas que necessitam de facilidade de implementação e não requerem controle total sobre o modelo. No entanto, o custo operacional pode aumentar conforme o volume de uso, e as possibilidades de customização e ajuste fino são limitadas comparado com modelos treinados internamente. De acordo com a Gartner, teremos um aumento de mais de 30% até 2026 na demanda por APIs vindo da IA e de ferramentas que utilizam LLMs.

Figura 33 - Quatro abordagens para colocar *Large Language Models* em produção



Quando a empresa necessita de integração com bases de dados internas da empresa ou de informações que não estão na base de dados do LLM, é normal evoluir para a abordagem do RAG. Aqui, a implantação inclui uma camada de recuperação de contexto. O sistema busca informações relevantes em documentos ou bases de dados e as fornece como contexto ao modelo. Você pode continuar utilizando APIs, como a OpenAI para acesso ao LLM, mas com o RAG, o modelo tem acesso a documentos contextuais adicionais, melhorando a precisão das respostas. Essa configuração requer integração com bases de dados e

um mecanismo de recuperação, elevando o custo, mas ainda sendo mais acessível do que a opção de ajuste fino do modelo.

O **RAG** é mais complexo que a engenharia de prompt porque ele envolve uma etapa extra de recuperação de dados externos para contextualizar as respostas do modelo, garantindo que ele tenha informações atualizadas e específicas ao responder.

Na engenharia de prompt, o modelo gera respostas apenas com base em seu treinamento prévio, sem acesso a informações adicionais. Essa complexidade adicional exige mais infraestrutura e um sistema de indexação e recuperação eficiente, aumentando o custo e o esforço de implementação. Em contrapartida, o RAG melhora a precisão, pois o modelo responde com base em informações contextualizadas e atualizadas, ideal para aplicações que precisam de conteúdo específico e dinâmico.

O ajuste fino (*fine-tuning*) exige treinar o modelo com dados específicos do domínio da empresa, aumentando a qualidade e a relevância das respostas, mas também o custo. Aqui, a organização pode optar por fazer ajuste fino do modelo usando uma GPU própria, serviços de terceiros (como [Together.ai](#)), ou até [*fine-tuning-as-a-service*](#) (como o da OpenAI). Cada opção vem com um custo maior do que o RAG, pois envolve computação adicional e um processo de ajuste do modelo que resulta em um modelo mais adaptado às necessidades específicas da empresa.



O ajuste fino é indicado quando a empresa precisa que o modelo produza respostas muito específicas para seu domínio ou área de atuação. Os treinamentos podem exigir horas de processamento em GPUs, elevando custos com energia e hardware, além de exigir expertise para gerenciar o treinamento e avaliar o desempenho. Para simplificar, empresas podem optar pelo **Fine-Tuning-as-a-Service (FTaaS)** oferecido por plataformas como a OpenAI, que facilitam o processo ao realizar o ajuste fino do modelo na infraestrutura da própria plataforma.

O **FTaaS** reduz a necessidade de manutenção de hardware, mas os custos ainda são relativamente elevados devido ao uso da infraestrutura de terceiros e ao processamento extra necessário para personalizar o modelo com dados específicos do cliente. Esse ajuste envolve infraestrutura avançada, como o uso contínuo de GPUs para calibrar os parâmetros do modelo com os dados fornecidos pela empresa. Na OpenAI, por exemplo, você pode criar seus próprios modelos personalizados fazendo *fine-tuning* nos modelos base com seus dados de treinamento. Após ajustar um modelo, você será cobrado apenas pelos *tokens* usados nas solicitações a esse modelo.

A última opção é utilizar LLMs de código aberto, tal como [Llama](#) da Meta (Meta, 2024). Neste caso, você pode possuir, geralmente, quatro opções:

- **Modelo rodando na infraestrutura local:** a empresa fica responsável por custear GPUs para rodar um LLM internamente. As GPUs podem ser caras de adquirir e manter e essa opção é recomendada para empresas que já possuem recursos computacionais ou que precisam de segurança e controle completos sobre os dados. Por exemplo, é uma opção muito utilizada em órgãos públicos que desejam total controle sobre os dados que estão indo para o modelo.
- **Modelo rodando em plataformas:** você pode utilizar plataformas como Together.ai para acessar modelos de código aberto via API. Essa opção é mais prática para empresas que buscam uma solução escalável e querem evitar os desafios de manutenção e custo de hardware. No entanto, o controle sobre os dados pode ser mais limitado comparado a uma infraestrutura própria.
- **Modelo com ajuste fino rodando na infraestrutura local:** você pode fazer um ajuste fino do modelo de código aberto internamente e rodar a inferência nas GPUs locais dentro da organização. O processo de ajuste fino do modelo exige mais poder computacional e horas de GPU para ajustar o modelo de acordo com os dados específicos da empresa. Isso oferece respostas mais relevantes, mas requer investimento em infraestrutura e equipe técnica para gerenciar o treinamento e monitoramento do modelo.
- **Treinamento do próprio modelo:** treinar um modelo do zero é a opção mais complexa e cara. Exige uma quantidade massiva de dados, alta capacidade computacional (como clusters de GPUs ou TPUs) e expertise técnica. Essa abordagem permite criar um modelo completamente personalizado e controlado pela empresa, mas é viável apenas para organizações com um orçamento elevado e uma necessidade específica que não pode ser atendida por modelos pré-treinados ou ajustados. A OpenAI, por exemplo, treinou o GPT-4 utilizando milhares de GPUs, e estima-se que o custo de computação para um modelo dessa escala seja de **mais de 100 milhões de dólares** (Knight, 2023).



5.2.3 REQUISITOS DE GPU PARA IMPLANTAÇÃO DE UM LARGE LANGUAGE MODEL EM UMA INFRAESTRUTURA LOCAL

O primeiro aspecto a considerar é o tamanho do modelo em termos de número de parâmetros. Modelos maiores, como o Llama-3-70B, com 70 bilhões de parâmetros, exigem aproximadamente 150 GB de memória . De modo geral, para modelos acima de 10 bilhões de parâmetros, são recomendadas GPUs com ao menos 24 GB de VRAM por unidade, como as NVIDIA A100 ou RTX 4090, pois modelos desse porte ultrapassam rapidamente a capacidade de GPUs menores, como as de 8 a 16 GB de VRAM.

A Equação 1 ajuda a prever a quantidade de VRAM necessária para rodar o LLM, levando em conta o tamanho do modelo e as necessidades operacionais adicionais:

1. **Parâmetros (P)**: representa o número total de parâmetros do modelo, que são valores ajustáveis aprendidos durante o treinamento. Quanto mais parâmetros, mais memória é necessária para armazená-los. Por exemplo, o modelo Llama-3-70B possui 70 bilhões de parâmetros.
2. **Bytes por parâmetro (B)**: cada parâmetro ocupa 4 bytes (ou 32 bits) quando carregado na GPU.
3. **Bits para carregamento (Q)**: refere-se à precisão usada ao carregar o modelo na GPU, geralmente 16 bits. Esse valor afeta o tamanho da memória total, pois o modelo pode ser carregado em diferentes níveis de precisão.
4. **Fator de sobrecarga (1.2)**: um fator multiplicador de 1.2 é adicionado para considerar uma sobrecarga de 20%, necessária para gerenciar ativações, cache e outros armazenamentos temporários que surgem durante a execução do modelo.



Equação 1 - Memória necessária para rodar um Large Language Model

$$M = \frac{(P * 4B)}{(32/Q)} * 1.2$$

Fonte: autoria própria.

A equação acima apresenta um exemplo do cálculo do requisito de memória do modelo Llama 70B com precisão de 16 bits. É necessário, aproximadamente, 168 GB de memória de GPU. Uma única GPU A100 de 80 GB não seria suficiente! (Equação 2).

Equação 2 - Exemplo do Llama 70B com precisão de 16 bits

$$\frac{70 * 4\text{bytes}}{32/16} * 1.2 = 168\text{GB}$$

Fonte: autoria própria.

O processo de **quantização** pode ser uma alternativa para reduzir o requisito de memória ao rodar grandes modelos (Lin et al., 2024), como o Llama 70B. Esse processo diminui a precisão dos parâmetros, reduzindo o uso de memória e permitindo uma implantação mais eficiente, especialmente em dispositivos com recursos limitados. Por outro lado, essa redução na precisão dos parâmetros tem um custo: a quantização pode afetar a exatidão e o desempenho do modelo.

Embora a quantização em 8 *bits* mantenha um desempenho próximo ao do modo de 16 *bits*, a quantização em 4 *bits* economiza ainda mais memória, mas pode impactar a precisão do modelo. No exemplo da Equação 3, o Llama 70B com precisão de 4 *bits* poderia rodar em 42 GB, usando duas GPUs L4 de 24 GB, por exemplo.

Equação 3 - Quantização do Llama 70B com precisão de 4 *bits*

$$\frac{70 * 4\text{bytes}}{32/4} * 1.2 = 42\text{GB}$$

Fonte: autoria própria.

5.3 OTIMIZAÇÃO DE LATÊNCIA E ACURÁCIA



Otimizar a latência e a acurácia em LLMs permite equilibrar desempenho e confiabilidade, especialmente em aplicações que exigem respostas rápidas e precisas. A latência afeta diretamente a experiência do usuário e a aplicabilidade em tempo real, enquanto a acurácia determina a qualidade das respostas. Esses dois aspectos, embora distintos, podem ser complementares; ajustes no design de prompts, na recuperação de dados contextuais e no controle de parâmetros são estratégias que ajudam a melhorar tanto a precisão quanto a velocidade de resposta dos modelos.

5.3.1 OTIMIZAÇÃO DE LATÊNCIA DOS *LARGE LANGUAGE MODELS*

Para otimizar a latência na implementação de LLMs dentro das empresas, é necessário adotar estratégias que reduzam o tempo de resposta dos modelos, permitindo uma experiência mais fluida dos usuários. Uma das estratégias é **reduzir o tamanho do modelo** utilizado. Modelos menores podem manter boa precisão ao mesmo tempo que

requerem menos recursos computacionais, o que reduz bastante o tempo de inferência. De acordo com as diretrizes da OpenAI (OpenAi, 2024), a escolha de um modelo adequado ao caso de uso específico permite que empresas alcancem um equilíbrio entre desempenho e velocidade, atendendo a requisitos de latência sem comprometer a qualidade das respostas.

Uma técnica de otimização de latência recomendada é o uso de cache para armazenar resultados de consultas frequentes. Em muitos casos de uso corporativo, certas solicitações ao modelo são recorrentes, e guardar essas respostas em cache evita a necessidade de processamento repetido, diminuindo o tempo de resposta. A OpenAI recomenda configurar um sistema de cache para armazenar respostas de consultas que sejam repetitivas (OpenAi, 2024), particularmente em ambientes de alta demanda. Este método ajuda em interfaces de perguntas e respostas e *chatbots*, onde perguntas semelhantes são feitas com frequência.

Outra técnica importante para otimizar a latência ao utilizar LLMs nas empresas é reduzir o número de *tokens* de entrada e saída. Como o processamento de cada *token* consome tempo e recursos computacionais das GPUs, limitar o número de *tokens* permite que o modelo retorne respostas de forma mais rápida (menor latência). Isso, as empresas podem otimizar as consultas enviadas ao modelo, simplificando as entradas para apenas o necessário, sem informações redundantes ou desnecessárias.

A OpenAI recomenda ajustar as configurações para limitar a quantidade de *tokens* gerados na resposta também (OpenAi, 2024), pois respostas mais curtas não só reduzem a latência, mas também ajudam a economizar recursos computacionais e custos associados ao processamento do modelo. Como regra geral, cortar 50% dos *tokens* de saída pode reduzir a latência em até 50% (OpenAi, 2024). Construa um *prompt* para pedir ao modelo para ser mais conciso, como por exemplo: "em menos de 20 palavras" ou "seja direto". Já a redução de *tokens* de entrada tem menor impacto na latência, com ganhos de **1%** a **5%** em média (OpenAi, 2024).

Cada solicitação ao modelo LLM envolve uma latência de ida e volta. Você pode **consolidar múltiplas etapas em uma única solicitação** à LLM — tal como agrupando tarefas em uma lista numerada e solicitando que o modelo responda em campos JSON —, reduzindo a latência acumulada e simplificando o processamento dos resultados. Para etapas que não dependem de uma ordem específica, é possível também realizar **chamadas paralelas** ao LLM, economizando tempo.



A percepção de latência do usuário é tão importante quanto a latência em si. Existem técnicas, como **streaming** (exibindo a resposta conforme é gerada), que tornam a espera mais interativa. Complementos visuais, como barras de carregamento e indicadores de progresso, melhoraram a experiência, enquanto o **streaming** reduz o tempo de espera do usuário ao permitir que ele comece a ler antes que a resposta esteja completa.

As LLMs são muitas vezes utilizadas em casos em que um **método clássico mais rápido e barato** seria mais adequado. Com o surgimento das LLMs, houve uma grande corrida por utilizá-las em todos os cenários, mas identificar casos em que técnicas tradicionais podem ser utilizadas pode permitir-lhe reduzir significativamente a sua latência. Considere os seguintes exemplos:

- **Codificação tradicional:** se o **resultado** for muito restrito, pode não precisar de um LLM para o gerar. Confirmações de ações, mensagens de recusa e pedidos de entrada padrão são todos grandes candidatos a serem codificados.
- **Pré-computação:** se a sua **entrada** for limitada (por exemplo, seleção de categoria), pode gerar várias respostas antecipadamente e certificar-se de que nunca mostra a mesma resposta a um usuário duas vezes.
- **Aproveitamento da interface do usuário:** métricas resumidas, relatórios ou resultados de pesquisa são, por vezes, melhor transmitidos com componentes de IU clássicos e personalizados, em vez de texto gerado pelo LLM.
- **Técnicas de otimização tradicionais:** princípios de otimização, como pesquisa binária e análise da complexidade de tempo de execução continuam relevantes e eficazes em aplicações que utilizam LLMs e devem ser aplicados para melhorar o desempenho.



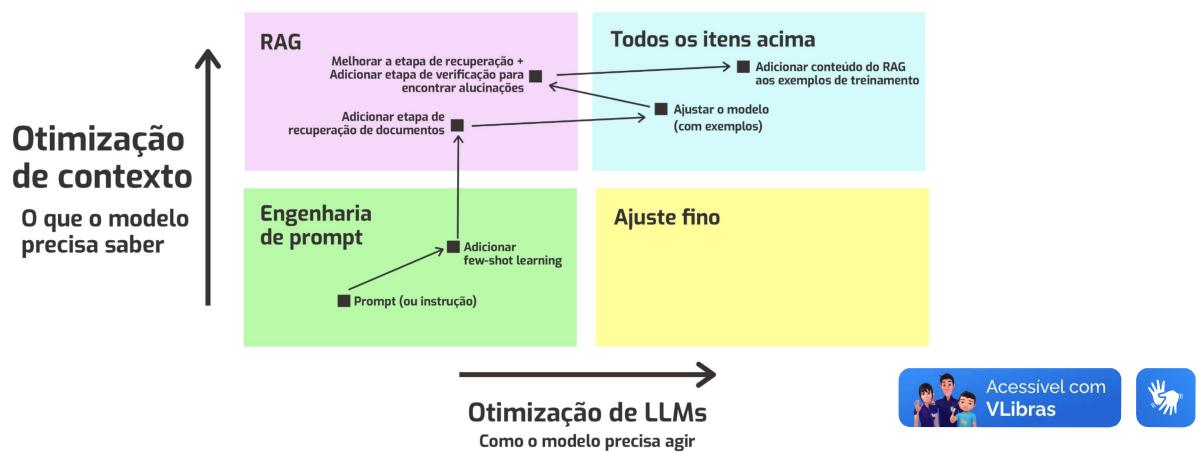
5.3.2 OTIMIZAÇÃO DA ACURÁCIA DOS *LARGE LANGUAGE MODELS*

A acurácia determina a relevância e a confiabilidade das respostas geradas pelos LLMs. Para atingir altos níveis de precisão, você deve aplicar métodos que vão desde o ajuste dos *prompts* até a adaptação dos parâmetros e a implementação de ciclos de verificação e treinamento. É importante ficar ciente durante a leitura dessa seção que qualquer discussão sobre a acurácia de LLMs deve começar com uma estimativa de custo-benefício: quanto uma resposta incorreta do modelo

Ihe custaria e quanto um acerto lhe economizaria ou geraria em receita. Essa análise de custo-benefício vai determinar o nível de precisão necessário para que o LLM seja considerado "bom o suficiente" para uso em produção na sua organização.

Na Figura 34, é apresentada uma matriz que facilita o entendimento dos passos para otimização da acurácia das LLMs. Ela divide a matriz em duas dimensões principais: otimização de contexto (o que o modelo precisa saber) e otimização do próprio LLM (como o modelo precisa agir). A otimização de contexto foca em fornecer as informações necessárias ao modelo, enquanto a otimização do LLM se concentra em ajustar seu comportamento.

Figura 34 - Matriz de otimização da acurácia de *Large Language Models*



Fonte: adaptada de OpenAI (2024).

Você pode seguir diversos caminhos de otimização de acurácia, levando em conta o problema que deseja resolver. O exemplo, na Figura 34, apresenta uma ideia geral que você pode seguir na otimização da acurácia:

1. Comece com um *prompt* e avalie seu desempenho.
2. Adicione exemplos estáticos (*few-shot learning*), que devem melhorar a consistência dos resultados.
3. Inclua uma etapa de recuperação dinâmica com RAG, trazendo documentos baseados na pergunta, para garantir contexto relevante de entrada para o LLM e melhorar a acurácia da resposta.
4. Agora você pode otimizar o seu LLM para a tarefa específica que você deseja realizar (por exemplo, a classificação de documentos). Para tal, prepare um conjunto de dados com 50 ou mais exemplos e faça ajuste fino do modelo para aumentar a acurácia dele na tarefa específica.
5. Ajuste a etapa de recuperação de documentos do RAG e adicione verificação da saída do LLM para evitar alucinações e alcançar

maior acurácia.

6. Re-treine o modelo que você otimizou no ajuste fino usando novos exemplos de treinamento, incluindo as entradas otimizadas do RAG.

Perceba que estas técnicas foram discutidas ao longo deste livro. A engenharia de *prompt* é o melhor lugar para se começar na otimização da acurácia (OpenAi, 2024). Para atingir esse objetivo, você deve sempre começar com um *prompt* simples e uma saída esperada em mente e, em seguida, otimizar o *prompt* adicionando contexto, instruções ou exemplos até que ele forneça o que você deseja. Para obter respostas mais precisas, os *prompts* devem ser claros, específicos e, idealmente, contextualizados. Técnicas de **engenharia de *prompt*** incluem a definição de perguntas diretas, o uso de exemplos e o refinamento iterativo do *prompt* para resolver mal-entendidos ou ambiguidades.

Além de um bom *prompt* é importante possuir um bom conjunto de avaliação dos resultados. Por exemplo, se tivermos um conjunto de mais de 20 perguntas e respostas, e tivermos examinado os detalhes das falhas e tivermos uma hipótese do porquê de elas estarem ocorrendo, então temos hipótese correta para adotar métodos de otimização mais avançados. Antes de passar para métodos de otimização sofisticados, também vale a pena considerar como automatizar essa avaliação para acelerar suas iterações. Você pode, por exemplo, utilizar o melhor modelo disponível (como o modelo **01** da OpenAI) como avaliador, onde você pede ao LLM para avaliar a saída da maneira mais objetiva possível.

Em aplicações onde a precisão de dados atualizados é essencial, utilizar técnicas de RAG e integrar LLMs com bases de dados ou APIs pode complementar a capacidade do modelo de fornecer respostas precisas. Em vez de confiar exclusivamente na "memória" do modelo, as informações de uma base de conhecimento externa atualizada garantem maior acurácia e reduzem o risco de respostas desatualizadas.

Fine-tuning permite o ajuste de LLMs para tarefas ou indústrias específicas, treinando o modelo com dados mais alinhados ao domínio de aplicação. Essa técnica melhora a acurácia, pois o modelo se torna mais adaptado a reconhecer contextos e termos técnicos específicos. Por exemplo, modelos ajustados para o setor jurídico ou médico podem interpretar melhor consultas especializadas e gerar respostas mais precisas e relevantes para esses contextos.

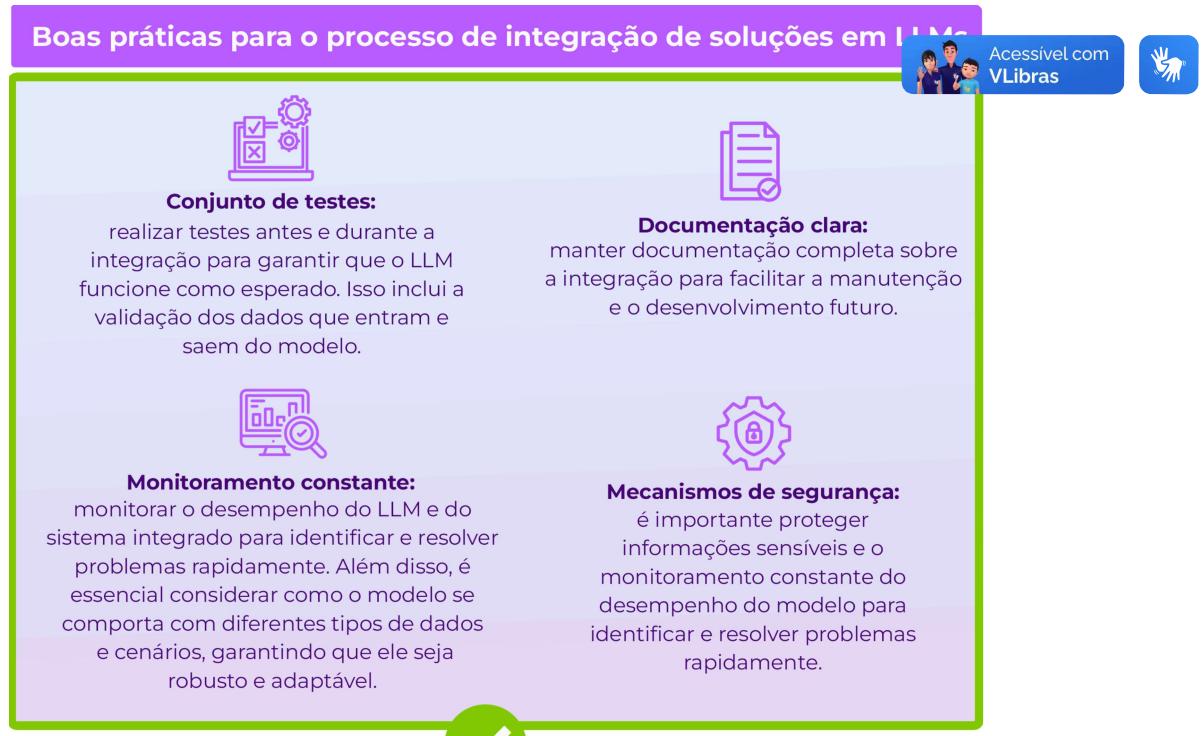


5.4 INTEGRAÇÃO DE SOLUÇÕES *LARGE LANGUAGE MODELS*

Integrar um LLM em sistemas ou aplicações existentes é importante para garantir que o modelo possa ser utilizado de forma prática. Essa integração envolve várias etapas, começando pelo planejamento arquitetural, que define como o LLM será inserido na infraestrutura já existente.

Existem diferentes abordagens para a integração de LLMs, dependendo da arquitetura do sistema. Uma das maneiras mais comuns de integrar LLMs é por meio da criação de APIs. Essas APIs permitem que o LM seja acessado por outros componentes do sistema de forma padronizada. Isso facilita a comunicação entre o LLM e outras partes do sistema. Durante o processo de integração, é importante seguir boas práticas para garantir que o LLM funcione de maneira eficaz. Observe o quadro na Figura 35, a seguir.

Figura 35 - Boas práticas para garantir que o *Large Language Model* funcione de maneira eficaz



Fonte: adaptada deGao et al. (2023).

5.5 DISPONIBILIZAR SOLUÇÕES *LARGE LANGUAGE MODELS* PARA USUÁRIOS FINAIS

Depois de integrar um LLM ao sistema, o próximo passo é disponibilizar o modelo em produção para que possa ser utilizado pelos usuários finais. Este processo requer cuidado e planejamento, pois envolve a configuração da infraestrutura necessária para suportar o modelo, bem como a implementação de estratégias para garantir sua segurança.

Antes de disponibilizar em produção, temos que otimizar o modelo para o ambiente de produção e configurar a infraestrutura adequadamente. Isso pode incluir a otimização do modelo para reduzir seu tamanho ou melhorar sua velocidade, bem como a configuração de servidores ou serviços em nuvem que suportem o volume esperado de tráfego.

Existem várias ferramentas e plataformas disponíveis para auxiliar na disponibilização de modelos em produção. Plataformas de computação em nuvem, como AWS (*Amazon Web Services*), Google Cloud, e Azure, oferecem serviços que podem hospedar e gerenciar modelos de inteligência artificial de maneira escalável e segura. Essas plataformas também fornecem recursos para manter a solução funcionando de forma otimizada e sempre disponível ao usuário final.

O monitoramento e manutenção do ambiente funcionando em produção requer estratégias para facilitar atualizações com segurança. Existe um grande risco de novas versões da aplicação gerarem erros no sistema e o monitoramento permite que novas versões sejam lançadas ao usuário e sejam monitoradas para validar se está atendendo às demandas dos usuários. A manutenção contínua assegura que a solução LLM permaneça atualizada com as últimas evoluções tecnológicas. Além disso, é importante planejar atualizações regulares do modelo para incorporar melhorias e corrigir possíveis vieses dos modelos.

5.6 ESTUDOS DE CASO E EXEMPLOS PRÁTICOS

Vamos explorar alguns estudos de caso que ilustram como diferentes empresas e organizações têm integrado e escalado soluções baseadas em LLMs com sucesso. Esses exemplos práticos ajudarão a contextualizar os conceitos discutidos nas seções anteriores, oferecendo percepções sobre como essas estratégias são aplicadas na prática.



5.6.1 CHATBOTS PARA ATENDIMENTO AO CLIENTE

Imagine a seguinte cena: milhares de clientes acessam um site de comércio eletrônico simultaneamente, com dúvidas sobre produtos, entregas e pagamentos. Atender a todos individualmente seria um grande desafio. Empresas como a Amazon e a Magazine Luiza utilizam chatbots baseados em LLMs para aprimorar a experiência do cliente e otimizar o suporte oferecido.

Essas empresas treinam LLMs com milhões de interações com clientes, permitindo que os chatbots compreendam a linguagem natural e respondam a perguntas frequentes de forma rápida e precisa. Fornecendo respostas instantâneas a dúvidas comuns, os chatbots reduzem o tempo de espera dos clientes e liberam os agentes humanos para questões mais complexas.

Além disso, os chatbots possuem disponibilidade constante e garantem atendimento ininterrupto aos clientes, independentemente do horário ou volume de solicitações. Por meio da análise do histórico de compras e preferências do cliente, os chatbots oferecem recomendações e soluções personalizadas. A implementação de chatbots baseados em LLMs pode resultar em:

- **Aumento da satisfação do cliente:** a velocidade no atendimento contribuem para uma experiência positiva do cliente.
- **Redução de custos operacionais:** a automação de tarefas repetitivas no atendimento libera os agentes para atividades mais estratégicas, otimizando o uso de recursos humanos.
- **Disponibilização de suporte escalável:** a capacidade de atender a um grande volume de solicitações simultâneas garante a satisfação do atendimento ao usuário mesmo em períodos de pico.



5.6.2 PERSONALIZAÇÃO DE CONTEÚDO COM LARGE LANGUAGE MODELS

Em um mundo sedento por informações, oferecer conteúdo relevante para cada usuário é a chave para o sucesso. A Netflix, por exemplo, utiliza LLMs para analisar o histórico de visualizações, avaliações e preferências de cada usuário, gerando recomendações personalizadas de filmes e séries. Essa personalização aumenta a

probabilidade do usuário encontrar conteúdo de seu interesse, otimizando a experiência na plataforma.

Já o Spotify utiliza LLMs para criar *playlists* personalizadas com base no histórico de reproduções, curtidas e artistas favoritos de cada usuário. Essa funcionalidade permite que o usuário descubra novas músicas e artistas que se encaixam em seu perfil, enriquecendo a experiência musical.

A personalização de conteúdo com LLMs permite:

- **Aumento do engajamento:** conteúdo relevante e personalizado mantém o usuário conectado à plataforma por mais tempo.
- **Fortalecimento da fidelização:** a experiência personalizada aumenta a percepção de valor da plataforma, incentivando a fidelização do usuário.
- **Geração de percepções:** os dados coletados sobre as preferências dos usuários podem ser utilizados para aprimorar a oferta de conteúdo e serviços.

5.6.3 TRADUÇÃO AUTOMÁTICA

A globalização exige comunicação rápida em diversos idiomas. O Google Tradutor, um dos pioneiros nesse campo, utiliza LLMs para traduzir textos, sites e documentos em tempo real, oferecendo suporte a diversos pares de idiomas e tornando a informação acessível a um público global.

A Microsoft integrou LLMs em sua plataforma Skype, permitindo que usuários conversem em tempo real com tradução simultânea, facilitando a comunicação entre pessoas que falam diferentes idiomas.

A tradução de conteúdo em tempo real permite que empresas e criadores alcancem um público global. A tradução automática torna o conhecimento e a informação acessíveis a um público mais amplo, independentemente da língua materna, além de estimular a colaboração internacional em diversas áreas.



SAIBA MAIS...

Para se aprofundar nos tópicos de integração e escalabilidade de LLMs, explore os seguintes recursos:

- Arquiteturas de Microsserviços e APIs:
 - "Building Microservices" por Sam Newman: o livro oferece um guia completo sobre arquitetura de microsserviços, incluindo *design*, desenvolvimento, *deploy* e manutenção.
 - Documentação de APIs (ex: REST, GraphQL): familiarize-se com diferentes estilos de arquitetura de APIs e como eles podem ser utilizados para integrar LLMs em sistemas existentes.
- Plataformas de Computação em Nuvem:
 - Documentação da AWS, Google Cloud e Azure: explore os serviços oferecidos por essas plataformas para *deploy* e gerenciamento de modelos de IA, como balanceamento de carga, escalonamento automático e monitoramento.
 - Kubernetes: aprenda sobre Kubernetes, uma plataforma open-source para automatizar o *deploy*, o escalonamento e o gerenciamento de aplicação contêineres.
- MLOps:
 - "Introducing MLOps" por Mark Treveil: este livro introduz os conceitos de MLOps e como eles podem ser aplicados para gerenciar o ciclo de vida de modelos de *machine learning*, incluindo LLMs.
 - Ferramentas de MLOps (ex: MLflow, Kubeflow): estude diferentes ferramentas que podem auxiliar no desenvolvimento, treinamento, *deploy* e monitoramento de modelos de IA.



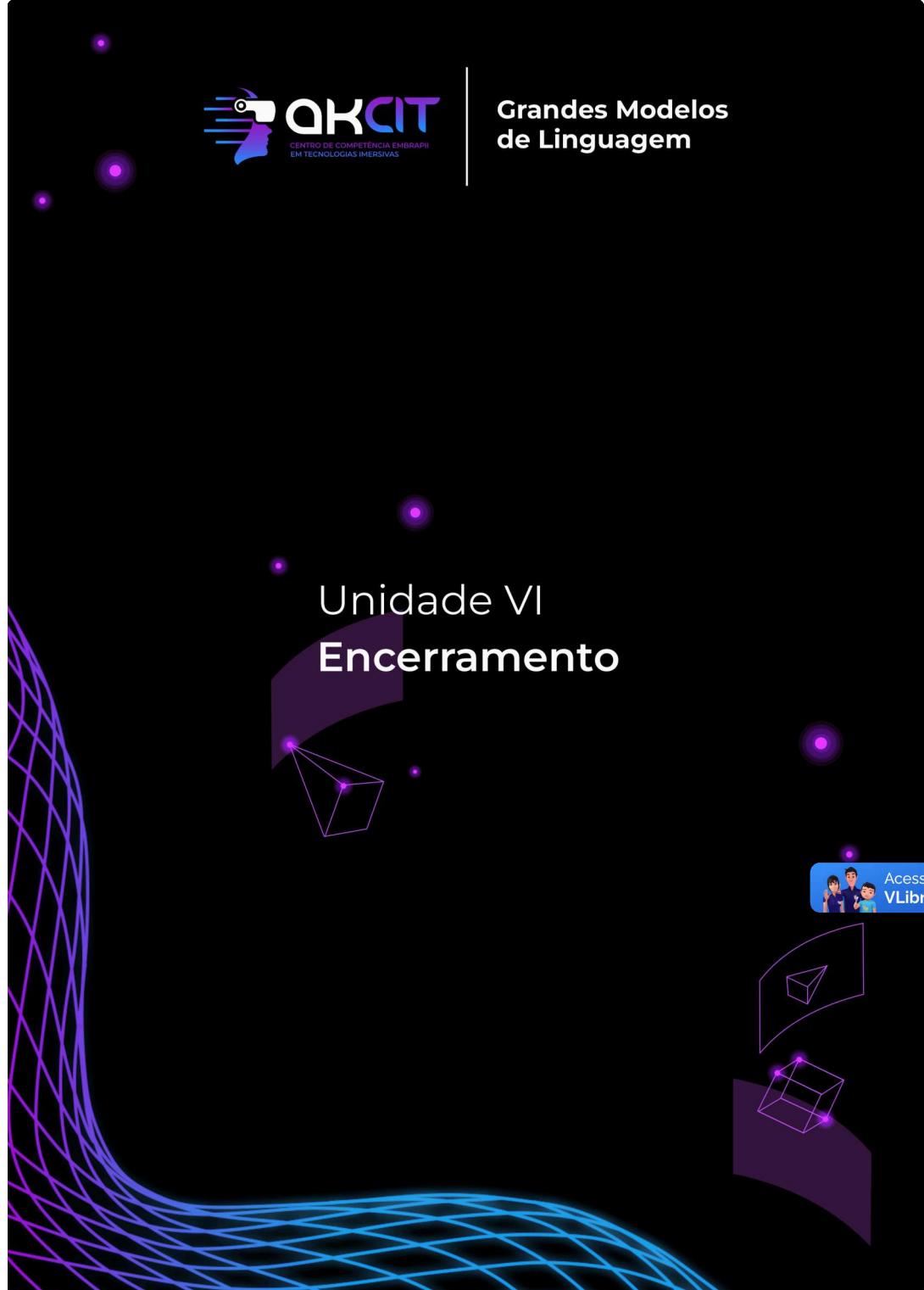
PARA RELEMBRAR...

Nessa Unidade, exploramos os conceitos de integração e escalabilidade de soluções LLMs. Vamos relembrar os principais termos:

- Escalabilidade: capacidade de um sistema de lidar com o aumento da demanda (usuários, dados, complexidade) sem comprometer o desempenho.
- Integração: processo de incorporar um LLM em sistemas ou fluxos de trabalho existentes, assegurando que funcione de forma coesa com outros componentes. Permite que os LLMs sejam utilizados em aplicações do mundo real.
- Paralelismo de modelo: divisão do modelo em partes menores que podem ser processadas simultaneamente, acelerando o processamento.
- GPUs: processadores especializados que oferecem maior poder computacional para tarefas intensivas como a execução de LLMs.
- APIs: permitem que o LLM seja acessado por outros componentes do sistema de forma padronizada, facilitando a comunicação e a integração.
- *Deploy*: processo de disponibilizar o modelo em produção para que possa ser utilizado pelos usuários finais. Envolve a configuração da infraestrutura e a implementação estratégias para garantir a eficiência e a segurança do modelo.
- Plataformas de computação em nuvem (ex: AWS, Google Cloud, Azure): oferecem serviços para hospedar e gerenciar modelos de IA de forma escalável e segura.



Unidade VI **Encerramento**



UNIDADE VI - **ENCERRAMENTO**

Ao longo deste livro, exploramos os fundamentos dos LLMs, uma das tecnologias mais importantes no campo do PLN.

Os LLMs demonstram capacidades que se assemelham ao raciocínio humano, em tarefas como geração de texto criativo, tradução automática, análise de sentimentos e criação de chatbots. Neste livro detalhamos como funcionam os LLMs e como são treinados utilizando a arquitetura Transformer. É importante notar que os LLMs exigem grande volume de dados e poder computacional disponível para conseguir treinar os modelos a partir do zero. Por isso, grandes empresas têm se destacado no lançamento dos LLMs mais utilizados na atualidade.

Discutimos como a utilização de técnicas de engenharia de *prompts* com a estruturação adequada de entradas podem melhorar os resultados obtidos, adaptando os modelos a uma variedade de tarefas, como geração de texto, tradução e respostas a perguntas. Falamos sobre o conceito de *in-context learning*, onde os modelos aprendem e ajustam suas respostas baseando-se em exemplos fornecidos durante a interação.

Quando precisamos de LLMs especializados em áreas como tradução, geração de código ou escrita criativa, o *fine-tuning* é uma opção. Esse ajuste fino molda as capacidades dos LLMs para atender necessidades específicas, aprimorando a precisão e a qualidade das saídas. Outro tema importante foi o RAG, que combina a geração de texto com a recuperação de informações, tornando os LLMs mais práticos e relevantes. Ao conectar os LLMs a vastas bases de dados externas, o RAG permite que eles busquem informações atualizadas em tempo real, ampliando seu conhecimento e capacidade de gerar respostas precisas e relevantes.

Por fim, abordamos os desafios que ainda precisam ser enfrentados na evolução dos LLMs, incluindo questões de escalabilidade e integração com as aplicações. Destacamos a importância de conectar os modelos a sistemas existentes e garantir seu bom funcionamento com o aumento da demanda.

6.1 SUGESTÕES PARA AÇÃO FUTURA E LEITURA ADICIONAL

A pesquisa em LLMs está em constante evolução, com novas descobertas e avanços sendo feitos a todo momento. O ICL, o *fine-tuning* com técnicas como LoRA, LongLoRA e QLoRA, e o aprimoramento das arquiteturas RAG são apenas alguns exemplos das áreas em constante desenvolvimento.

A jornada pelo mundo dos LLMs não termina aqui! Após absorver os conceitos deste livro, você está pronto(a) para ir além e se aprofundar



nesse universo em constante expansão. Aqui estão algumas sugestões para continuar sua jornada de aprendizado:

- **Prática com ferramentas e ambientes de teste:** envolva-se ativamente na experimentação com LLMs usando plataformas como *Google Colab* ou *Hugging Face*, que oferecem ambientes acessíveis para testar e ajustar modelos. Essas plataformas oferecem acesso a LLMs pré-treinados e recursos para você experimentar diferentes técnicas de *prompt engineering*, *fine-tuning* e RAG.
- **Estudo de modelos específicos:** dedique tempo ao estudo de arquiteturas específicas, como GPT, Llama e Gemini. Cada uma dessas arquiteturas tem suas peculiaridades e pontos fortes, e entender essas diferenças pode ser um diferencial na escolha da melhor abordagem para diferentes projetos.
- **Leitura de literatura acadêmica e técnica:** explore artigos científicos e publicações que discutem os avanços mais recentes no campo. Conferências como NeurIPS, ACL e publicações como o *Journal of Machine Learning Research* são fontes ricas de conhecimento avançado.
- **Crie seus próprios projetos:** aplique seus conhecimentos em projetos pessoais que explorem as capacidades dos LLMs. Que tal construir um *chatbot*, um sistema de tradução automática gerador de conteúdo criativo?
- **Junte-se à comunidade:** explore os conteúdos de fóruns, grupos online e comunidades dedicadas a LLMs, como o fórum do *Hugging Face* e o *OpenAI Community Forum*. Compartilhe suas dúvidas, aprenda com as experiências de outros e colabore em projetos open-source.



6.2 LEITURA ADICIONAL

- "*Prompt Engineering for Generative AI*", por James Phoenix e Mike Taylor;
- "*Natural Language Processing with Transformers*", por Lewis Tunstall, Leandro von Werra e Thomas Wolf;
- "*Deep Learning for Natural Language Processing*", por Yoav Goldberg.

6.3 REFLEXÕES FINAIS

Os LLMs representam uma evolução no campo do PLN. Sua capacidade de compreender e gerar linguagem abre um universo de possibilidades para diversas áreas, desde a automação de tarefas repetitivas até a criação de novas formas de arte e interação. No entanto, à medida que continuamos a explorar suas capacidades, é importante lembrar que esses modelos são, ao mesmo tempo, potentes e limitados. A responsabilidade no uso dos LLMs envolve não apenas a busca por resultados precisos, mas também a consideração dos impactos éticos e sociais que eles podem gerar.

Concluímos aqui nossa jornada introdutória pelos LLMs. O campo continua a evoluir rapidamente e incentivamos você a seguir explorando, aprendendo e contribuindo para o desenvolvimento responsável dessa área tão promissora.

REFERÊNCIAS

AWS. Transform your business with generative. **AWS**, [online], 2024.

Disponível em: <https://aws.amazon.com/ai/generative-ai/>. Acesso em: 29 nov. 2024.



BENGIO, Y.; DUCHARME, R.; VINCENT, P.. A neural probabilistic language model. **Advances in Neural Information Processing Systems**, v. 13, 2000.

BROWN, T. et al. Language models are few-shot learners. **Advances in Neural Information Processing Systems**, v. 33, p. 1877-1901, 2020.

CHEN, Y. et al. Longlora: Efficient fine-tuning of long-context large language models. **ICLR 2024**, 20203. arXiv preprint arXiv:2309.12307.

CHENG, M.; DURMUS, E.; JURAFSKY, D.. Marked personas: Using natural language prompts to measure stereotypes in language models. In: The 61st Annual Meeting Of The Association For Computational Linguistics. 2023. **arXiv preprint** arXiv:2305.18189.

COLLOBERT, R. et al. Natural language processing (almost) from scratch. **Journal of Machine Learning Research**, v. 12, p. 2493– 2537, 2011.

COMMON CRAWL. **Common Crawl**, 2007. Disponível em: <https://commoncrawl.org/>. Acesso em: 29 nov. 2024.

DETTMERS, T. et al. Qlora: Efficient finetuning of quantized LLMs. **Advances in Neural Information Processing Systems**, v. 36, 2024.

DEVLIN, J.; CHANG, M-W.; LEE, K.; TOUTANOVA, K.. BERT: **Pre-training of Deep Bidirectional Transformers for Language Understanding**. In:

BURSTEIN, Jill; DORAN, C.; SOLORIO, T. (Ed.). Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, v. 1 (Long and Short Papers). Minneapolis, Minnesota: Association for Computational Linguistics, 2019. p. 4171–4186. DOI: 10.18653/v1/N19-1423.

DODGE, J. et al. Documenting large webtext corpora: a case study on the colossal clean crawled corpus. **arXiv preprint**, arXiv:2104.08758, 2021.

DONG, Q. et al. A survey on in-context learning. **arXiv preprint**, arXiv:2301.00234, 2022.

EDMONDS, J.. Matroids and the greedy algorithm. **Mathematical programming**, v. 1, p. 127-136, 1971.

FEDUS, W.; ZOPH, B.; SHAZER, N.. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. **Journal of Machine Learning Research**, v. 23, n. 120, p. 1-39, 2022.

GARTNER. Gartner prevê que mais de 30% do aumento na demanda por APIs virá de Inteligência Artificial e de ferramentas que usam Modelos de Linguagem até 2026. **Gartner Newsroom**, 20 mar. 2024. Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2024-03-20-gartner-predicts-more-than-30-percent-of-the-increase-in-demand-for-apis-will-come-from-ai-and-tools-using-langs-by-2026>. Acesso em 2024.



GAO, Y. et al. Retrieval-augmented generation for large language models: a survey. **arXiv preprint**, arXiv:2312.10997, 2023.

GEHMAN, S. et al. Real toxicity prompts: evaluating neural toxic degeneration in language models. **arXiv preprint**, arXiv:2009.11462, 2020.

GERO, K. I.; LIU, V.; CHILTON, L.. Sparks: inspiration for science writing using language models. **Proceedings of the 2022 ACM Designing Interactive Systems Conference**. 2022. p. 1002-1019.

GIRAY, L.. Prompt engineering with ChatGPT: a guide for academic writers. **Annals of Biomedical Engineering**, v. 51, n. 12, p. 2629-2633, 2023.

HADI, M. U. et al. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. **Authorea Preprints**, 2024.

HOFFMANN, J. et al. Training compute-optimal large language models. **arXiv preprint**, arXiv:2203.15556, 2022.

HOLTZMAN, A. et al. The curious case of neural text degeneration. In: International Conference on Learning Representations. **arXiv preprint**, arXiv:1904.09751, 2019.

HU, E. J. et al. Lora: low-rank adaptation of large language models. In: **Proceedings of the 2022 International Conference on Learning Representations (ICLR)**, 2022. arXiv preprint arXiv:2106.09685.

JURAFSKY, D.; MARTIN, J. H.. Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. **Pearson/Prentice Hall**, 3. ed., 2024.

KAPLAN, J. et al. Scaling laws for neural language models. **arXiv preprint**, arXiv:2001.08361, 2020.

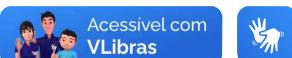
KNIGHT, W.. OpenAI's CEO says the age of Giant AI Models is already over. **WIRED**, 17 abr. 2023. Disponível em: <https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/>. Acesso em: 24 nov. 2024.

KOMBRINK, S. et al. Recurrent neural network based language modeling in meeting recognition. In: **Interspeech**. 2011. p. 2877-2880.

LEWIS, M. Bart: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. **arXiv preprint**, arXiv:1910.13461, 2019.

LI, P. et al. Making ai less "thirsty": uncovering and addressing the secret water footprint of ai models. **arXiv preprint**, arXiv:2304.03271, 2023.

LIN, C-Y. Rouge: a package for automatic evaluation of summarization branches out. 2004. p. 74-81.



LIN, J. et al. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. **Proceedings of Machine Learning and Systems**, v. 6, p. 87-100, 2024.

LIU, Y. et al. RoBERTa: a robustly optimized BERT pretraining approach. **CoRR**, v. abs/1907.11692, 2019.

MANNING, C. D.; SCHÜTZE, H. Foundations of statistical natural language processing. **MIT Press**, 1999.

MCGUFFIE, K.; NEWHOUSE, A.. The radicalization risks of GPT-3 and advanced neural language models. **arXiv preprint**, arXiv:2009.06807, 2020.

MIKOLOV, T. et al. Distributed representations of words and phrases and their compositionality. **Advances in Neural Information Processing Systems**, v. 26, 2013.

MIKOLOV, T. et al. Recurrent neural network based language model. In: **Interspeech 2010, 11th Annual Conference of the International Speech Communication Association**, Makuhari, Chiba, Japan, September 26-30,

2010, Kobayashi, T.; Hirose, K.; Nakamura, K. (Eds.). ISCA, 2010, p. 1045–1048.

MIKOLOV, T. et al.. Efficient estimation of word representations in vector space. **Cornell University**, v. 3781, 2013. arXiv preprint arXiv:1301.3781.

OPENAI. Latency Optimization. **OpenAI Documentation**. Disponível em: <https://platform.openai.com/docs/guides/latency-optimization>. Acesso em: 24 out. 2024.

PAPINENI, K. et al. Bleu: a method for automatic evaluation of machine translation. **Proceedings of the 40th annual meeting of the Association for Computational Linguistics**. 2002. p. 311-318.

PETERS, M. E. et al. Deep contextualized word representations. In: **Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies**, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers), M. A. Walker, H. Ji, & A. Stent, Eds. Association for Computational Linguistics, 2018, pp. 2227–2237.

PICKLE, B. Unified memory architecture. **Techterm.com**, [online], 2023. Disponível em: https://techterms.com/definition/unified_memory_architecture. Acesso em: 29 nov. 2024.

PHOENIX, J.; TAYLOR, M. **Prompt engineering for generative AI**. Acessível com VLibras Media, Incorporated, 2024.



RADFORD, A. et al. Language models are unsupervised multitask learners. **OpenAI blog**, v. 1, n. 8, p. 9, 2019.

RAFFEL, C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer. **Journal of Machine Learning Research**, v. 21, p. 140:1-140:67, 2020.

SANH, V. et al. Multitask prompted training enables zero-shot task generalization. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. **arXiv preprint**, arXiv:2110.08207.

SHOEYBI, M. et al. Megatron-LM: training multi-billion parameter language models using model parallelism. **arXiv preprint**, arXiv:1909.08053, 2019.

SINGH, S. All about deep learning models that you should know. **Labeller**, [online], 2014. Disponível em: <https://www.labellerr.com/blog/all-about-deep-learning-models-that-you-should-know/>. Acesso em: 29 nov. 2024.

SHANAHAN, M. Talking about large language models. **Communications of the Association for Computing Machinery**, v. 67, n. 2, p. 68-79, 2024.

arXiv:2212.03551.

SHENG, E. et al. The Woman Worked as a Babysitter: On Biases in Language Generation. **Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)**. 2019. p. 3407-3412.

TANG, R. et al. Large language models can be lazy learners: analyze shortcuts in in-context learning. **arXiv preprint**, arXiv:2305.17256, 2023.

TAYLOR, R. et al. Galactica: a large language model for science. **arXiv preprint**, arXiv:2211.09085, 2022.

TURNEY, P. D.; PANTEL, P. From frequency to meaning: vector space models of semantics. **Journal of Artificial Intelligence Research**, v. 37, p. 141-188, 2010.

VASWANI, A. Attention is all you need. In: **Advances in Neural Information Processing Systems 30**: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 2017, pp. 5998–6008.

VIEJO, D. P. The science of control: how temperature, Top_P, and Top_K shape large language models. **Medium [online]**, 2013. Disponível em: <https://medium.com/@daniel.puenteviejo/the-science-of-control-how-temperature-top-p-and-top-k-shape-large-language-models-853cb0480dae>. Acesso em: 24 nov. 2023.



WANG, Y. et al. Generalizing from a few examples: A survey on few-shot learning. **Association for Computing Machinery Computing Surveys (CSUR)**, v. 53, n. 3, p. 1-34, 2020.

WANG, T. et al. What language model architecture and pretraining objective works best for zero-shot generalization?. **International Conference on Machine Learning**. PMLR, 2022. p. 22964-22984.

WEI, J. et al. Chain-of-thought prompting elicits reasoning in large language models. **Advances in Neural Information Processing Systems**, v. 35, p. 24824-24837, 2022.

WEI, J. et al. Emergent abilities of large language models. **arXiv preprint**, arXiv:2206.07682, 2022.

WHITE, J. et al. A prompt pattern catalog to enhance prompt engineering with ChatGPT. **arXiv preprint**, arXiv:2302.11382, 2023.

ZHAO, W. X. et al. A survey of large language models. **arXiv preprint**, arXiv:2303.18223, 2023.

