Pós-Graduação em Inteligência Artificial Generativa e LLMs

PUC-Rio

Introdução ao Processamento de Linguagem Natural

Notas de Aula

Vítor Wilher

NLP · 2025.2

Versão 1.0

Resumo. Do pré-processamento de texto aos grandes modelos de linguagem: representações vetoriais, embeddings, RAG e aplicações text-to-text.

Palavras-chave: NLP; embeddings; RAG; LLM

Notas de estudo elaboradas a partir do material das aulas.

Índice

1 Visão Geral da Disciplina

A disciplina **NLP — Introdução ao Processamento de Linguagem Natural** percorre, em doze encontros, o caminho que vai das representações mais elementares de texto até a construção de sistemas de IA generativa capazes de recuperar conhecimento externo e agir de forma autônoma. O ponto de partida é uma pergunta simples e fundamental: como transformar palavras, que são símbolos, em números que algoritmos matemáticos consigam processar? A partir dessa pergunta, o curso constrói uma escada conceitual em que cada degrau nasce da limitação do degrau anterior.

O primeiro bloco (Aulas 1 e 2) cobre os fundamentos: o pipeline canônico de PLN, as representações baseadas em frequência (**Bag-of-Words** e **TF-IDF**), a **similaridade do cosseno** como métrica de comparação entre documentos, e a transição para os **word embeddings** densos (Word2Vec, com as arquiteturas CBOW e Skip-gram). Nessa mesma etapa é apresentado o ecossistema **Hugging Face** e o catálogo de tarefas clássicas de PLN — classificação, NER, tradução, geração, sumarização, QA e reconhecimento de fala.

O segundo bloco (Aulas 3 a 6) trata do **RAG (Retrieval-Augmented Generation)** e da **engenharia de prompt**. Aqui, o curso conecta a base teórica de Recuperação de Informação (modelos booleano, vetorial e probabilístico; métricas como precisão, recall, MAP e NDCG) com a engenharia prática de pipelines: chunking, embeddings, bancos vetoriais, ranking e re-ranking, janela de contexto, e finalmente extração de dados de PDFs via **OCR**. Em paralelo, estuda-se como escrever prompts eficazes, com técnicas que vão do zero-shot ao Chain-of-Thought e ao Tree of Thoughts, apoiadas nos artigos originais dessas linhas de pesquisa.

O terceiro bloco (Aulas 7 a 12) fecha o ciclo com **agentes de IA**, **multimodalidade**, **Transformers** e o paradigma **text-to-text**. Discute-se por que a arquitetura Transformer venceu as redes recorrentes, como o mecanismo de atenção funciona matematicamente, como a escala transformou Transformers em LLMs, e como o paradigma unificado do T5 reformulou todas as tarefas de PLN como geração de texto. A Aula 9, de revisão, costura os quatro pilares do curso em uma síntese prática: NLP para entender, LLM para gerar, RAG para saber e Agentes para agir.

2 Introdução a NLP — Representação e Embeddings

2.1 O que é Processamento de Linguagem Natural

O **Processamento de Linguagem Natural (PLN)** é o ramo da inteligência artificial que se concentra na interação entre computadores e linguagem humana. Ele permite que máquinas entendam, interpretem, gerem e manipulem linguagem natural de forma significativa, combinando técnicas de linguística, ciência da computação e aprendizado de máquina.

Dentro do panorama da IA, o PLN se distingue por trabalhar com **dados não-estruturados** de texto e fala. A visão computacional processa imagens, a robótica controla ações físicas, e o PLN cuida da linguagem — mas todas essas áreas compartilham o mesmo alicerce de machine learning. Exemplos práticos permeiam o cotidiano digital: corretores ortográficos, assistentes virtuais (Siri, Alexa, Google Assistant), tradução automática (Google Translate, DeepL) e análise de sentimentos em redes sociais.

2.2 Evolução histórica

O material apresenta uma linha do tempo que ajuda a situar a área:

- **Anos 1950**: Teste de Turing; primeiras tentativas de máquinas compreenderem linguagem.
- **1960s-80s**: era das regras; ELIZA (1966); sistemas baseados em regras linguísticas.
- **1990s-2000s**: métodos estatísticos, machine learning e crescimento dos corpora.
- **2010s**: deep learning; Word2Vec (2013), RNNs, Transformers (2017).
- **2018-2020**: BERT (2018), GPT-2 (2019), GPT-3 (2020) — a era dos modelos pré-treinados.
- **2020s**: era dos LLMs; ChatGPT (2022), GPT-4 (2023).

A justificativa para a relevância atual é direta: bilhões de textos são gerados diariamente, e máquinas precisam compreendê-los automaticamente para criar valor em saúde, direito, educação e negócios.

2.3 O pipeline de PLN

Todo sistema de PLN segue um fluxo estruturado de seis etapas:

1. **Entrada de texto bruto** — documentos, e-mails ou posts não processados.
2. **Tokenização** — divisão em unidades menores. `"Olá! Como você está?"` torna-se `["Olá", "!", "Como", "você", "está", "?"]`.
3. **Normalização** — lowercase, remoção de pontuação, stemming e lematização.
4. **Extração de características** — conversão do texto em vetores numéricos (BoW, TF-IDF, embeddings).
5. **Treinamento do modelo** — redes neurais, SVM, Random Forest, Transformers.
6. **Predição/saída** — classificação, tradução, sumarização.

2.4 Representações por frequência: BoW e TF-IDF

Bag-of-Words (BoW) representa um documento como um vetor em que cada dimensão corresponde a uma palavra do vocabulário e o valor é a frequência daquela palavra. A frase "O gato está no sofá" vira algo como `{gato: 1, está: 1, sofá: 1, o: 1, no: 1}`. As vantagens são a simplicidade, a eficiência computacional e a interpretabilidade. As desvantagens são estruturais: perde-se a ordem das palavras, a dimensionalidade é altíssima, a semântica não é capturada, sinônimos são tratados como termos distintos e os vetores são muito esparsos.

O **TF-IDF (Term Frequency-Inverse Document Frequency)** corrige parcialmente o problema de que palavras muito comuns dominam a contagem. O termo TF mede quantas vezes o termo aparece no documento; se "machine" aparece 5 vezes em um documento de 100 palavras, então TF = 0,05. O termo IDF penaliza palavras presentes em muitos documentos, sendo definido como $IDF(t) = \log(N/df(t))$, onde N é o total de documentos e $df(t)$ o número de documentos que contêm o termo. O produto final é $TFIDF(t, d) = TF(t, d) \times IDF(t)$.

A intuição é que o TF-IDF atribui mais peso a palavras raras e menos peso a ocorrências comuns. Como consequência, dois documentos são considerados semelhantes quando compartilham palavras **raras** em comum — precisamente o que sistemas de recuperação de informação, classificação e mineração de texto precisam.

2.5 Similaridade do cosseno

A **similaridade do cosseno** mede o ângulo entre dois vetores no espaço vetorial:

$$sim(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

onde o numerador é o produto escalar e o denominador é o produto das normas (magnitudes) dos vetores. O resultado fica entre 0 e 1. Um valor de 1,0 (ângulo de 0 grau) indica vetores idênticos; 0,5 (60 graus) indica similaridade moderada, como entre "Gato preto" e "Gato branco"; e 0,0 (90 graus) indica ausência de similaridade, como entre "Gato preto" e "Carro vermelho".

A superioridade da métrica vem de quatro propriedades: é insensível ao comprimento do documento, é interpretável (valores entre 0 e 1), é computacionalmente eficiente e preserva a orientação semântica dos vetores. Suas aplicações incluem recuperação de informação, clustering de documentos e sistemas de recomendação.

2.6 Limitações da frequência e a solução dos embeddings

Três limitações justificam o salto para embeddings. Primeiro, BoW e TF-IDF **não capturam significado**: "Gato" e "Felino" produzem vetores completamente diferentes apesar de significarem quase o mesmo. Segundo, **não reconhecem sinonímia**: "Carro", "Automóvel" e "Veículo" ocupam três dimensões distintas. Terceiro, **não exploram contexto**: "Amo PLN" e "PLN amo" geram o mesmo vetor BoW apesar de sentidos diferentes.

Embeddings são vetores densos de baixa dimensionalidade (tipicamente 50 a 300 dimensões) que capturam significado semântico e sintático por meio de redes neurais treinadas em grandes corpora. Palavras similares ficam próximas no espaço, relações são preservadas (a analogia clássica é `rei - homem + mulher ~ rainha`), e as representações são transferíveis entre tarefas.

2.7 Word2Vec: CBOW e Skip-gram

O **Word2Vec** é treinado em um grande corpus usando uma rede neural rasa de duas camadas. Sua fundamentação é a **hipótese distribucional**: palavras que ocorrem em contextos semelhantes tendem a ter significados semelhantes. Duas arquiteturas principais implementam essa ideia:

CBOW (Continuous Bag of Words) prevê a palavra alvo a partir do contexto. Em "O gato [_____] no tapete", o modelo tenta prever "dormiu" usando as vizinhas. Treina mais rápido que o Skip-gram, é eficiente com grandes volumes e funciona bem para palavras frequentes, mas é menos eficaz para palavras raras e menos preciso em tarefas de similaridade.

Skip-gram faz o inverso: prevê as palavras do contexto a partir da palavra alvo. Dada a palavra "dormiu", tenta prever "O", "gato", "no", "tapete". Funciona bem com pouca quantidade de dados de treinamento, representa melhor palavras raras e captura relações semânticas complexas — ao custo de treinamento mais lento e maior demanda computacional.

Característica	CBOW	Skip-gram
Velocidade	Mais rápido	Mais lento

Característica	CBOW	Skip-gram
Palavras raras	Menos eficaz	Mais eficaz
Mecanismo	Contexto para alvo	Alvo para contexto

As aplicações práticas incluem similaridade semântica, sistemas de recomendação, tradução automática e análise de sentimento.

3 Tarefas de NLP

3.1 O catálogo de tarefas

O material lista as principais tarefas que buscam permitir que máquinas compreendam, interpretem, manipulem e gerem linguagem humana: classificação de texto, extração de informações, resposta a perguntas, tradução automática, geração de texto, resumo automático, conversação natural e chatbot.

Classificação de texto atribui uma ou mais categorias a um texto. Exemplos típicos são a classificação de e-mails como spam ou não-spam e a classificação de sentimentos (positivo, negativo, neutro). Técnicas modernas empregam redes convolucionais, LSTM, Transformers como o BERT e modelos ajustados por fine-tuning.

Extração de informações (IE) identifica e estrutura informações de textos não estruturados. Envolve o **Reconhecimento de Entidades Nomeadas (NER)**, que identifica pessoas, locais e organizações, e a **Extração de Relações**, que determina como essas entidades se conectam. Em "Barack Obama nasceu no Havaí", o NER identifica "Barack Obama" como pessoa e "Havaí" como localização, enquanto a extração de relações reconhece a relação "nasceu em".

Question Answering (QA) constrói sistemas capazes de responder perguntas em linguagem natural, seja com respostas curtas extraídas diretamente de textos (como no dataset SQuAD), seja com respostas geradas livremente. Sistemas avançados como o RAG combinam busca em bases de dados com geração neural para produzir respostas fundamentadas.

Tradução automática (MT) evoluiu de métodos estatísticos para redes recorrentes (NMT) e hoje é dominada por Transformers como T5, mBART e o modelo massivamente multilíngue da Meta.

Geração de texto produz automaticamente novo conteúdo textual. Modelos como GPT, T5 e PaLM utilizam decoders auto-regressivos ou arquiteturas encoder-decoder. A geração controlada por tópico, estilo ou emoção é um subcampo emergente.

Sumarização cria uma versão condensada preservando a informação central. Há dois tipos: **resumo extrativo**, que seleciona sentenças do texto original, e **resumo abstrativo**, que gera novas frases. Pegasus, BART e FLAN-T5 são citados como altamente eficazes no caso abstrativo.

Análise de sentimentos identifica a emoção ou atitude expressa. Pode ser binária, ternária ou baseada em múltiplas emoções (raiva, alegria, tristeza).

3.2 O ecossistema Hugging Face

A Hugging Face é uma empresa de tecnologia fundada em 2016 que inicialmente trabalhava com chatbots e se tornou líder na democratização de modelos de machine learning para PLN. Sua filosofia declarada é "democratizar o bom Machine Learning". O ecossistema tem quatro pilares:

- **Hub de Modelos**: milhares de modelos pré-treinados prontos para uso (BERT, GPT, T5, DistilBERT, RoBERTa, mT5).
- **Datasets**: coleção vasta de conjuntos de dados para treinamento e avaliação.
- **Bibliotecas**: `transformers`, `datasets` e `accelerate`.
- **Inference API e Spaces**: uso de modelos sem configuração local e deploy de aplicações interativas com Gradio ou Streamlit.

O ponto conceitual mais importante é que o Hugging Face é **baseado em tarefas**. A função `pipeline` é uma abstração de alto nível que permite usar um modelo pré-treinado em uma tarefa comum com poucas linhas de código, sem lidar manualmente com tokenização, pré-processamento ou pós-processamento:

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
result = classifier("Eu amo NLP!")
# [{'label': 'POSITIVE', 'score': 0.99}]
```

Material de slides não disponível especificamente para esta aula; o conteúdo acima é reconstruído a partir do material de tarefas de PLN apresentado no conjunto de slides da disciplina.

3.3 Introdução ao RAG

O mesmo conjunto de slides introduz o **RAG (Retrieval-Augmented Generation)** como resposta a três limitações intrínsecas dos LLMs: atualização de conhecimento (acesso a informação posterior ao treinamento), redução de alucinações (fundamentação em fontes autoritativas) e transparência (citação das fontes usadas). O fluxo básico é: consulta, recuperação (retrieval) e geração (generation).

4 Introdução a RAG

4.1 Fundamentos de Recuperação de Informação

Information Retrieval (IR) é o processo de encontrar documentos ou dados relevantes em uma coleção grande e não estruturada, a partir de uma consulta do usuário. A área nasceu nos anos 1960 com sistemas de busca em bibliotecas digitais e evoluiu de modelos booleanos simples para modelos vetoriais (1975), probabilísticos, e finalmente para os modelos de embedding baseados em deep learning (2019 em diante).

Três famílias de modelos de recuperação são apresentadas:

- **Booleano**: usa operadores lógicos (AND, OR, NOT). Recuperação exata, sem ranking; o resultado é binário — relevante ou não.
- **Vetorial**: representa documentos e consultas como vetores em espaço multidimensional e calcula similaridade (por exemplo, cosseno). Produz ranking contínuo e captura relevância parcial.
- **Probabilístico**: estima a probabilidade de relevância com teoria bayesiana. Modelos como o **BM25** são muito usados; considera frequência de termos e tem base teórica sólida.

As métricas de avaliação são **precisão** (percentual de resultados recuperados que são relevantes), **recall** (percentual de documentos relevantes efetivamente recuperados), **MAP (Mean Average Precision)**, que considera a ordem dos resultados, e **NDCG (Normalized Discounted Cumulative Gain)**, considerado o padrão ouro em IR moderno por penalizar documentos relevantes em posições baixas.

4.2 Fundamentos de aprendizado de máquina

Como material de apoio, a aula utiliza um texto de fundamentos de aprendizado de máquina. Ele define IA como a automação de atividades normalmente associadas a raciocínio — resolver problemas, tomar decisões e aprender — e situa o Aprendizado de Máquina como uma técnica de programação baseada na detecção automática de padrões nos dados e na indução de conclusões a seu respeito.

Um ponto conceitual relevante é a distinção entre **dedução** e **indução**. Na dedução, conclusões logicamente verdadeiras são derivadas de premissas conhecidas e verdadeiras, partindo do geral para o específico; a conclusão é um fato comprovado logicamente. Na indução, as conclusões derivam de premissas particulares não necessariamente demonstradas, partindo do específico para o geral; não há garantia de validade, apenas uma probabilidade que cresce com o número de casos analisados. Aprendizado de Máquina é, nesse enquadramento, inferência por indução: dada uma amostra $\{(x_i, y_i)\}$, induz-se a expressão de uma função geral $y = f(x)$.

Os tipos de aprendizado são quatro: **Supervised Learning** (atributos e resultados associados), **Unsupervised Learning** (apenas atributos, descoberta de padrões), **Semi-Supervised** (quantidade limitada de resultados) e **Reinforcement Learning** (treino por tentativa e erro a partir de um objetivo). Entre as técnicas listadas estão regressão linear, redes neurais densas, SVM, Naive Bayes, árvores de decisão, redes convolucionais, redes recorrentes, redes de atenção e Transformers bidirecionais.

O texto também percorre a revisão matemática necessária — vetores, matrizes, tensores, **Mean Squared Error**, regra da cadeia, gradiente e convolução — e detalha o treinamento de redes neurais densas em seis etapas: definição da arquitetura da rede (L, n, w, b), preparação da amostra, dedução da função de custo, seleção do método de minimização do erro, formulação dos mecanismos de propagação de sinais e ajuste da rede para garantir custo mínimo. Conceitos de validação, **underfitting**, **overfitting** e **regularização** fecham o material.

4.3 O pipeline do RAG

A aula estabelece o princípio central do RAG com uma frase que sintetiza toda a disciplina de recuperação: “A qualidade da resposta final é diretamente proporcional à qualidade do contexto

recuperado. Garbage in, garbage out."

O processo tem três fases:

1. **Preparação do conhecimento**: chunking, embeddings, indexação.
2. **Busca inicial**: similaridade vetorial, recuperação de top-K, filtragem.
3. **Refinamento**: ranking, re-ranking, seleção final.

Chunking é o processo de dividir documentos extensos em pedaços menores, coerentes e otimizados para busca. O impacto na relevância vem da granularidade: chunks muito grandes diluem a informação, chunks muito pequenos perdem o contexto. As melhores práticas são o chunking estrutural (dividir por parágrafos ou seções lógicas, não apenas por contagem de tokens) e a sobreposição (**overlap**), incluindo frases do chunk anterior ou posterior para manter continuidade semântica.

A **similaridade vetorial** compara a representação vetorial da consulta com a dos chunks no banco vetorial. A métrica-chave é a similaridade de cosseno; o score é usado para selecionar os K chunks mais promissores, e a definição de um **limiar (threshold)** mínimo é crucial para evitar injeção de contexto irrelevante.

O **re-ranking** existe porque a similaridade vetorial, embora rápida, nem sempre captura a relevância contextual completa. O processo tem três passos: seleção inicial dos top-K da busca vetorial, reavaliação com um modelo mais sofisticado (**cross-encoder**) e reordenação. A vantagem principal é que o re-ranker avalia a interação direta entre consulta e chunk.

A **janela de contexto** é o limite final e inegociável: o número máximo de tokens que o LLM processa em uma chamada, incluindo consulta, contexto recuperado e resposta gerada. O fenômeno **"Lost in the Middle"** descreve a tendência de LLMs ignorarem informação relevante posicionada no meio de contextos longos — motivo pelo qual o re-ranking deve colocar o essencial no início.

4.4 Técnicas avançadas e otimização

Quatro técnicas avançadas são apresentadas:

- **HyDE (Hypothetical Document Embeddings)**: usa o LLM para gerar documentos hipotéticos que responderiam à consulta e então busca documentos similares a esses hipotéticos. Melhora o matching semântico e é eficaz em consultas vagas.
- **Multi-hop Retrieval**: recupera documentos iterativamente, refinando a consulta a cada etapa. Permite raciocínio multi-passo e conecta informações dispersas.
- **RAG Adaptativo**: implementa feedback do usuário e aprendizado contínuo, personalizando por usuário e domínio.
- **LLM-as-a-Retriever**: usa o próprio LLM para decidir quais documentos recuperar, combinando raciocínio com recuperação.

Na dimensão de performance, quatro estratégias são destacadas: redução de latência (processamento paralelo de chunks, batch processing, menos round-trips de rede), indexação eficiente (índices hierárquicos **HNSW**, quantização de vetores, busca aproximada **ANN**), caching inteligente (de embeddings, de resultados de busca e de respostas geradas) e compressão de embeddings (quantização em int8 ou bfloat16, redução de dimensionalidade).

4.5 Desafios e ferramentas

Quatro desafios recorrentes em produção são mapeados junto de suas soluções: **alucinações residuais** (mitigadas por validação de respostas, citações obrigatórias e confidence scores), **contexto irrelevante** (re-ranking robusto e limiares mais altos), **drift semântico** (re-embedding periódico e monitoramento contínuo) e **qualidade de dados** (limpeza, validação e governança).

O ecossistema de ferramentas cobre modelos de embeddings (OpenAI `text-embedding-3`, Hugging Face BERT e MPNet, Cohere Embed, Mistral Embed), bancos vetoriais (Pinecone, Weaviate, Milvus, Qdrant) e frameworks (LangChain, LlamaIndex, Haystack, Verba). As melhores práticas de implementação são: começar simples, monitorar qualidade desde o início com NDCG, MRR e Precision@K, testar iterativamente e otimizar incrementalmente.

5 RAG

Material de slides expositivos não disponível para esta aula: os arquivos anexados são documentos-fonte (textos técnicos e um guia de turismo científico do Rio de Janeiro) usados como base de conhecimento para os exercícios práticos de RAG, além de notebooks e um conjunto de dados eleitorais em CSV.

5.1 O papel dos documentos-fonte

Esta aula é essencialmente prática e consolida, com corpora reais, o pipeline estudado na Aula 3. A escolha dos documentos é pedagogicamente relevante porque expõe os alunos a três perfis distintos de texto, cada um com desafios próprios de ingestão:

- **Texto técnico-científico** (capítulo de livro acadêmico sobre oncologia e sinalização purinérgica), com terminologia densa, citações bibliográficas embutidas no corpo do texto e numeração de páginas que polui a extração.
- **Material de divulgação estruturado** (cartilha de saúde), com sumário, seções curtas e forte formatação visual.
- **Guia de turismo**, com listas, endereços e informação factual pontual.
- **Dados tabulares** em CSV (resultados eleitorais por município e candidato), que exigem uma estratégia de recuperação diferente da busca semântica pura.

A lição implícita é que o mesmo pipeline — carregar, dividir, vetorizar, armazenar, recuperar, gerar — precisa ser calibrado de forma diferente conforme a natureza do documento. Textos com estrutura hierárquica clara favorecem chunking recursivo por parágrafos; corpora com terminologia rara favorecem estratégias híbridas que combinem busca vetorial com busca por palavra-chave; e dados tabulares frequentemente pedem uma ferramenta separada, e não indexação vetorial.

6 Introdução à Prompt Engineering

6.1 O que são prompts e por que a engenharia importa

Um **prompt** é a entrada fornecida ao modelo de linguagem — a frase ou pergunta que inicia a conversa e orienta o modelo para a saída desejada. **Engenharia de prompt** designa o conjunto de técnicas que ajudam a criar prompts claros e detalhados, por meio de exemplos práticos e instruções melhores. O material é enfático: não se pode mais escrever como se escreve uma busca no Google, porque semântica e contexto importam, e um prompt mal escrito gera saída inútil.

6.2 Alucinações

Alucinações de LLM ocorrem quando o modelo gera informações incorretas ou sem base factual. As causas apontadas são dados de treinamento insuficientes ou enviesados e limitações na arquitetura do modelo. O alerta central é que a resposta frequentemente **parece real** — o modelo chega a criar referências e links que não existem. O material cita o caso, noticiado em 2023, de um advogado que usou casos inventados pelo ChatGPT em um processo judicial.

6.3 Componentes de um bom prompt

Cinco componentes são elencados:

1. **Clareza**: ser específico sobre o que se quer saber.
2. **Contexto**: fornecer informação suficiente para uma resposta relevante.
3. **Objetividade**: perguntas diretas e concisas, sem ambiguidade.
4. **Detalhamento**: incluir especificidades que direcionem a resposta.
5. **Propósito**: definir claramente o objetivo do prompt.

O exemplo dado contrasta "Como funciona uma âncora?" com "Como funciona uma âncora em uma plataforma de extração de petróleo e quais são os principais desafios de manutenção desses sistemas?".

Três dimensões complementares são o **tom** (ajustado à formalidade do contexto), o **idioma** (adequado ao público-alvo) e o reconhecimento das **limitações da IA** (evitar solicitar tarefas fora do escopo ou que exijam julgamento humano complexo). O material observa que escrever em inglês historicamente fazia diferença porque os modelos GPT foram treinados em conjuntos com mais dados em inglês, mas registra que no GPT-5 há treino efetivamente multilíngue, melhor alinhamento semântico e menor dependência de um idioma dominante.

6.4 Hiperparâmetros

Os hiperparâmetros que moldam a saída são:

- `temperature`: entre 0 e 1, define a aleatoriedade. Quanto mais próximo de 1, mais aleatório e criativo; valores baixos produzem respostas conservadoras e consistentes.
- `frequency_penalty`: entre -2 e 2. Valores positivos reduzem a probabilidade de palavras repetidas. Padrão 0.

- `presence_penalty`: entre -2 e 2. Valores positivos aumentam a probabilidade de abordar novos assuntos. Padrão 0.
- `max_tokens`: número máximo de palavras da resposta.
- `n`: quantidade de respostas geradas. Padrão 1.
- `stop`: lista de textos que interrompem a geração. Padrão nulo.

Sobre **tokens**, o material reproduz as referências da própria OpenAI: 1 token equivale a aproximadamente 4 caracteres ou cerca de 1 palavra; 100 tokens correspondem a cerca de 75 palavras; 1 parágrafo tem cerca de 100 tokens; e 1.500 palavras equivalem a aproximadamente 2.048 tokens.

6.5 Técnicas de engenharia de prompt

Iterative Prompting consiste em repetir e ajustar o prompt várias vezes com base no feedback da resposta anterior até alcançar o resultado desejado. É a técnica mais comum.

Chain of Thought (CoT) induz o modelo a seguir um raciocínio passo a passo. Em vez de gerar uma resposta imediata, o modelo reflete sobre cada etapa antes de avançar, o que melhora significativamente a qualidade em tarefas de lógica, cálculo e estatística, e ajuda a reduzir alucinações. O artigo de referência da aula é "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" (Wei et al., Google Research), que mostra que essas habilidades de raciocínio emergem naturalmente em modelos suficientemente grandes quando poucas demonstrações de cadeia de pensamento são fornecidas como exemplares. O trabalho reporta ganhos expressivos em raciocínio aritmético, de senso comum e simbólico — com apenas oito exemplares de cadeia de pensamento, o PaLM 540B atinge acurácia estado da arte no benchmark GSM8K de problemas matemáticos, superando inclusive um GPT-3 ajustado com verificador. As avaliações fora de domínio (**OOD**) mostram que o CoT também facilita a generalização para cadeias mais longas do que as vistas nos exemplares, embora apenas na escala de cerca de 100 bilhões de parâmetros.

Tree of Thoughts (ToT) é a técnica avançada em que o raciocínio deixa de ser linear e passa a ser hierárquico, ramificando-se em múltiplas direções exploradas simultaneamente. O artigo de referência é "Tree of Thoughts: Deliberate Problem Solving with Large Language Models" (Yao et al., NeurIPS 2023). O paper parte da observação de que LLMs ainda estão confinados a decisões em nível de token, da esquerda para a direita, e propõe uma inspiração na teoria de "processo dual" da cognição humana: um modo rápido e automático (Sistema 1) e um modo lento e deliberado (Sistema 2). O ToT mantém uma árvore de **pensamentos** — sequências linguísticas coerentes que servem como passos intermediários — e combina geração, autoavaliação e algoritmos de busca (busca em largura, BFS, ou em profundidade, DFS) com lookahead e backtracking. O resultado mais citado do artigo é o **Game of 24**: enquanto o GPT-4 com prompting de cadeia de pensamento resolveu 4% das tarefas, o ToT alcançou 74% de sucesso. Outras tarefas avaliadas foram Creative Writing e Mini Crosswords.

Em sala, o ToT é ensinado de forma operacional: sempre dar mais de uma alternativa e pedir ao modelo que raciocine sobre elas. Um exemplo é o prompt sobre escolha de notebook para edição de vídeo, que pede três fatores, ao menos duas opções por fator, prós e contras, e uma escolha final — construindo a árvore fatores, opções, análise, decisão.

Prompt Agentic é a forma de escrever prompts pensando em agentes autônomos: sistemas baseados em LLMs que planejam, decidem, executam ações, avaliam resultados e iteram até cumprir um objetivo. A diferença prática é nítida: o prompt tradicional pede "Responda X"; o prompt agentic

diz “Você é um agente que deve analisar X, decidir Y, executar Z, validar o resultado e repetir se necessário”. Os componentes clássicos são papel do agente (role), objetivo claro, plano ou etapas, critérios de decisão, critérios de sucesso, possibilidade de iteração e limites (o que **não** fazer). O fluxo típico é: entender o objetivo, planejar ações, executar, avaliar resultado e decidir continuar, corrigir ou encerrar.

O material observa uma mudança de foco: em modelos mais antigos, a engenharia de prompt servia para compensar limitações — forçar raciocínio passo a passo, repetir instruções, usar truques como “think step by step”. No GPT-5, o foco deixa de ser “fazer o modelo pensar” e passa a ser “fazer o modelo agir do jeito certo”.

Zero-shot, One-shot e Few-shot organizam a técnica pelo número de exemplos: zero-shot não dá exemplos, apenas uma direção; one-shot fornece um exemplo que serve de modelo; few-shot fornece vários exemplos, tornando a resposta mais consistente e a estrutura mais definida.

6.6 Roles e o uso de API

Na API, as mensagens são organizadas em três papéis: `user` (o prompt convencional), `system` (instruções para o modelo, como “você é um poeta”) e `assistant` (resposta do modelo, que serve como contexto). Os papéis `system` e `assistant` não são sempre obrigatórios.

Uma **API (Application Programming Interface)** é a interface que permite que sistemas conversem entre si de forma padronizada, definindo como pedir informações e receber respostas sem conhecer o funcionamento interno. A analogia usada é a do garçom em um restaurante: o cliente faz o pedido, o garçom leva à cozinha e traz o prato pronto.

6.7 Template de prompt no estilo cookbook

O material fecha com um template único, inspirado no guia oficial da OpenAI sobre prompting no GPT-5, com sete blocos: **contexto** (cenário onde a resposta será usada), **papel** (quem o modelo deve ser), **objetivo** (o que constitui uma boa resposta), **critérios de qualidade** (nível técnico, clareza, profundidade, restrições), **regras operacionais** (proatividade, necessidade de confirmação, uso de exemplos), **formato de saída** (tópicos, tabela, código, número de itens, tamanho máximo) e um bloco final de **verificação antes de responder** (refletir sobre o problema, verificar consistência, evitar contradições).

7 Recuperação Avançada com RAG - Usando OCR

7.1 Por que os LLMs sozinhos não bastam

A aula retoma três limitações estruturais: **alucinações** (respostas plausíveis mas incorretas), **conhecimento desatualizado** (o modelo só conhece dados até seu treinamento) e **ausência de domínio específico** (não acessa documentos privados). O RAG conecta o LLM a uma base externa verificada, garantindo precisão, transparência e acesso a dados atualizados.

O ponto de foco desta aula é a etapa de **extração** — o primeiro elo do pipeline e frequentemente o mais frágil.

7.2 O desafio do formato PDF

O PDF foi criado para apresentação visual e impressão, não para leitura lógica por máquinas. Três problemas se destacam:

- **Nativo versus escaneado**: PDFs nativos contêm texto selecionável e codificado; PDFs escaneados são apenas imagens de documentos e exigem OCR.
- **Layout visual versus lógico**: colunas múltiplas, tabelas complexas e caixas de texto flutuantes dificultam a ordem lógica de leitura — o texto pode ser lido da esquerda para a direita quando deveria ser lido de cima para baixo.
- **Elementos não-estruturados**: cabeçalhos, rodapés e números de página são extraídos erroneamente como parte do conteúdo principal.

7.3 Bibliotecas de extração

Três ferramentas são comparadas:

PyPDFLoader é excelente para PDFs nativos, rápido e com boa qualidade de texto, mas depende da estrutura interna do PDF — se o arquivo for apenas imagem, não funciona.

pytesseract (Tesseract OCR) é um motor tradicional e muito maduro, que funciona bem para documentos bem escaneados em alta resolução e com pouco ruído. Depende da instalação do binário Tesseract no sistema e pode ter dificuldade com layouts muito complexos ou fontes diferentes.

EasyOCR é baseado em deep learning, com suporte a muitos idiomas, e se sai muito bem com textos "no mundo real" como placas e fotos. O tempo de inicialização do modelo é mais alto, e a qualidade, embora possa ser superior em certos cenários, precisa ser testada caso a caso.

```
# Estratégia híbrida: tentar extração nativa, com OCR como fallback
from langchain_community.document_loaders import PyPDFLoader

docs = PyPDFLoader("documento.pdf").load()

if not "".join(d.page_content for d in docs).strip():
    # PDF sem texto selecionavel: converter paginas em imagem e aplicar OCR
    import easyocr
    leitor = easyocr.Reader(["pt"])
    texto = "\n".join(leitor.readtext(pagina_img, detail=0))
```

7.4 OCR: conceito e boas práticas

OCR (Optical Character Recognition) é a tecnologia que converte imagens de texto (pixels) em dados de texto editáveis e pesquisáveis. É essencial em PDFs escaneados, fotografias de documentos físicos e sempre que a extração direta retorna vazio.

Quatro boas práticas determinam a qualidade do resultado:

- **Resolução**: usar alta densidade, com mínimo de 300 DPI, para preservar detalhes dos caracteres.

- **Binarização**: converter para preto e branco puro, aumentando o contraste entre texto e fundo.
- **Remoção de ruído**: eliminar manchas e artefatos que geram falsas detecções de pontuação.
- **Alinhamento (deskewing)**: corrigir rotações, pois texto alinhado melhora substancialmente a precisão.

O princípio que resume tudo é o **GIGO — Garbage In, Garbage Out**.

Em produção, a recomendação é misturar estratégias: usar PyPDFLoader sempre que possível, porque é rápido, barato e limpo, e aplicar OCR apenas como fallback ou em PDFs sabidamente escaneados. A decisão pode ser tomada página a página, com base na pergunta "há texto selecionável?".

7.5 Tipos de chunking

Fixed-Size Chunking divide o texto em blocos de tamanho fixo (por exemplo, 500 caracteres), ignorando estrutura semântica ou gramatical. É extremamente rápido, barato e fácil de implementar, mas quebra frases e parágrafos ao meio, com perda severa de contexto. O exemplo clássico é "O rato roeu a roup" seguido de "a do rei de Roma".

Chunking Recursivo é a abordagem recomendada e considerada o padrão ouro para cerca de 90% dos casos de RAG. Ele tenta preservar a estrutura semântica dividindo o conteúdo por uma lista hierárquica de separadores: primeiro parágrafos (`\n\n`), depois linhas (`\n`), depois palavras (espaço) e, em último caso, caracteres. Se um pedaço resultante ainda exceder o `chunk_size`, o algoritmo desce ao próximo separador.

Semantic Chunking usa embeddings para identificar mudanças de tópico. O algoritmo tem quatro passos: dividir o texto em sentenças, gerar vetores para cada sentença, calcular a similaridade de cosseno entre sentenças adjacentes, e criar um novo chunk quando a similaridade cai abaixo de um limiar. O exemplo simulado mostra "O gato está dormindo no sofá" e "Os felinos gostam de lugares macios" com similaridade 0,89 (mantém no mesmo chunk), seguidos de "A taxa de juros do mercado subiu hoje" com similaridade 0,21 (abaixo do limiar, provoca divisão).

Dois parâmetros críticos governam o processo. O `chunk_size` define a granularidade: pequeno é mais preciso mas perde contexto amplo; grande traz mais contexto mas dilui a relevância. O `chunk_overlap` é a quantidade de caracteres repetidos entre o final de um chunk e o início do próximo; evita cortar frases ao meio e mantém o contexto nas fronteiras. A recomendação é de 10% a 20% do tamanho do chunk.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=150,
    separators=["\n\n", "\n", " ", ""],
)
chunks = splitter.split_documents(docs)
```

7.6 Gerenciamento de memória

O desafio é o **"token budget"**: contexto infinito não existe, e enviar documentos inteiros aumenta custo, latência e pode degradar a qualidade da resposta pelo efeito "Lost in the Middle". Três estratégias são apresentadas: **re-ranking** (recuperar muitos chunks, por exemplo 50, e usar um cross-encoder para reordená-los, enviando apenas os cinco melhores), **compressão** (usar um LLM intermediário para resumir ou extrair fatos essenciais dos chunks recuperados) e **janela deslizante** (manter apenas as últimas K interações do histórico ou usar um summary buffer).

No LangChain, dois tipos de memória são distinguidos. A **memória conversacional** mantém o histórico do chat e não tem relação com os documentos do RAG. A **memória baseada em vetor (vectorstore)** não é uma memória "viva": o LangChain apenas envia consultas ao vectorstore, que decide o que recuperar por busca semântica baseada em similaridade de embeddings.

As implementações de memória conversacional são quatro:

- `ConversationBufferWindowMemory` — janela deslizante que mantém apenas as últimas N mensagens. É um recorte literal do histórico; ideal quando memória longa não é necessária.
- `ConversationBufferMemory` — buffer completo, guardando todas as mensagens sem compressão. O contexto explode em custo de tokens e os LLMs começam a alucinar com contexto muito grande. Serve para demonstrações simples.
- `ConversationSummaryMemory` e `ConversationSummaryBufferMemory` — a cada novo bloco, o LangChain pede ao LLM um resumo condensado que substitui parte do histórico. É a compressão inteligente: mantém o sentido da conversa com tamanho fixo.
- `TokenBufferMemory` e `TokenBufferWindowMemory` — controlam a memória pelo número de tokens, não de mensagens ("preserve até 2.000 tokens; se passar, resuma"). É a abordagem descrita como mais moderna e prática.

7.7 Otimizações avançadas de recuperação

Quatro otimizações fecham a aula:

- **Hybrid Search** combina busca semântica por vetores com busca por palavras-chave (**BM25**), capturando termos específicos como nomes e códigos que os embeddings podem perder. Implementada com Ensemble Retriever.
- **Re-ranking** recupera um conjunto amplo e usa um cross-encoder, mais lento mas preciso, para reordenar. Exemplos citados: Cohere Rerank e BGE-Reranker.
- **Query Expansion** usa um LLM para gerar variações da pergunta ou decompor uma pergunta complexa em sub-perguntas. Implementada com MultiQueryRetriever.
- **Context Compression** extrai apenas as frases relevantes antes de enviar ao LLM, economizando tokens. Implementada com ContextualCompressionRetriever.

Os desafios recorrentes e suas soluções recomendadas são: baixa precisão (hybrid search mais reranking), contexto fragmentado (aumentar o overlap ou usar Parent Document Retriever) e alucinação (refinar o system prompt para restringir respostas ao contexto e exigir citações).

O futuro apontado tem três direções: **Agentic RAG** (evolução de pipelines lineares para agentes autônomos que planejam e decidem quando buscar mais), **Self-Correction** (sistemas como CRAG e Self-RAG que avaliam a própria resposta e reiniciam a busca ao detectar alucinações) e **Multimodal**

RAG (indexar e raciocinar sobre imagens, áudio e vídeo). Entre os papers essenciais listados estão o RAG Paper de 2020 (Lewis et al.), o estudo "Lost in the Middle" de 2023 e o Self-RAG de 2023.

8 Introdução a Agentes

Material de slides não disponível para esta aula: o arquivo de apresentação está listado, porém sem texto extraível no contexto fornecido.

A aula é apoiada por notebooks práticos de agentes com a API da OpenAI, de agentes com LangChain, de um agente com uma única ferramenta de cálculo e de orquestração de agentes com LangGraph. O conteúdo conceitual correspondente foi consolidado na aula de revisão e é apresentado ali: a distinção entre chatbot e agente, o loop ReAct, o uso de ferramentas via function calling, memória de curto e longo prazo, arquiteturas multi-agente, frameworks de orquestração e os modos típicos de falha (loop infinito, alucinação de ferramenta e custo explosivo), com **guardrails** como `max_iterations` e `early_stopping_method` como mitigação.

9 Introdução a Modelos Multimodais

9.1 O paradigma text-to-text

Os slides desta aula apresentam **Modelos Text-to-Text: Uma Abordagem Unificada para NLP**, ministrados pelo Prof. Dr. Leonardo Alfredo Forero Mendoza. A agenda tem quatro blocos: o paradigma unificado, a arquitetura T5, tarefas e prompts, e a evolução rumo ao instruction tuning.

A **abordagem tradicional**, exemplificada pelo BERT, exige uma cabeça (head) diferente para cada tarefa: uma cabeça de classificação que produz um rótulo, uma cabeça de span prediction que produz índices. A **abordagem unificada** do T5 e do GPT usa o mesmo modelo para tudo: a entrada `classify sentiment: I love this!` produz a saída `positive`; a entrada `translate English to German: Hello` produz `Hallo`.

O segredo são os **prefixos textuais**. Como a arquitetura é idêntica, adiciona-se um prefixo à entrada que instrui explicitamente o modelo sobre o objetivo: `translate English to German:`, `summarize:`, `cola sentence:`.

9.2 O modelo T5

O **T5 (Text-to-Text Transfer Transformer)** foi introduzido pelo Google Research em 2020 e propõe refatorar todas as tarefas de PLN em um formato unificado de texto para texto, permitindo usar o mesmo modelo, a mesma função de perda e os mesmos hiperparâmetros. Três características o definem:

- **Arquitetura encoder-decoder**: diferente do BERT (apenas encoder) ou do GPT (apenas decoder), o T5 usa a arquitetura Transformer original completa, ideal para tarefas de sequência para sequência.

- **Transfer learning massivo**: pré-treinado no dataset C4 (Colossal Clean Crawled Corpus), aprendendo padrões linguísticos gerais antes do fine-tuning.
- **Prefixos de tarefa**: a tarefa é indicada por um prefixo textual natural.

O pré-treinamento usa **span corruption**, um objetivo de "denoising". Diferente do BERT, que mascara palavras isoladas, o T5 mascara **spans** — sequências de palavras — substituindo cada um por um token sentinela único. Da frase "O processamento de linguagem natural é fascinante", a entrada torna-se "O [X] linguagem natural [Y]" e o alvo, "[X] processamento de [Y] é fascinante [Z]".

9.3 As tarefas na prática

Tradução: a entrada `translate English to German: That is good.` produz `Das ist gut.` O ponto conceitual é que o modelo não possui um módulo de tradução separado — a instrução em linguagem natural condiciona a atenção do modelo a gerar tokens na língua alvo. O mesmo modelo traduz para francês ou romeno apenas mudando o prefixo.

Sumarização: o prefixo `summarize:` seguido de um documento longo produz um resumo. Modelos text-to-text realizam **sumarização abstrativa** — não apenas recortam frases importantes do original (extrativo), mas geram novas sentenças que capturam a essência, parafraseando e condensando de forma humana.

Classificação: a entrada `sst2 sentence: The movie was absolutely wonderful!` gera a string `positive`. A mudança de paradigma é que, em vez de um ID de classe, a saída é literalmente a palavra. Na inferência de linguagem natural, em vez do ID 0, gera-se a string `entailment`.

Question Answering: a entrada compõe pergunta e contexto — `question: Who founded SpaceX? context: SpaceX is an American aerospace manufacturer founded in 2002 by Elon Musk...` — e a saída é `Elon Musk`. Na abordagem BERT, a tarefa é de classificação de tokens: o modelo prevê os índices de início e fim da resposta dentro do contexto, ou seja, "aponta" a resposta. No T5, a resposta é **gerada token por token**, o que permite responder mesmo quando a resposta exata não está escrita explicitamente (QA generativo) ou reformulá-la para soar mais natural.

Aceitabilidade linguística (dataset CoLA): `cola sentence: The course is jumping well.` gera `acceptable`; `cola sentence: The boy is eating the.` gera `unacceptable`. Mesmo classificação binária vira geração de texto: o modelo aprende a gerar as strings literais token por token, em vez de ativar um neurônio de saída específico.

9.4 Vantagens, limitações e evolução

As **vantagens** da abordagem unificada são três. A **simplicidade arquitetural** elimina a necessidade de projetar cabeças de classificação específicas por problema. O **transfer learning positivo** faz com que o conhecimento adquirido em uma tarefa (como tradução) seja transferido para outra (como resumo) por compartilhamento total de parâmetros, melhorando o desempenho especialmente em tarefas com poucos dados. A **flexibilidade de dados** permite misturar datasets de naturezas diferentes no mesmo batch de treinamento, facilitando treinamento multitarefa em escala massiva.

Os **trade-offs** são igualmente claros. A **latência de inferência** é maior porque gerar texto token por token (decodificação auto-regressiva) é significativamente mais lento que uma classificação direta, o que pode ser proibitivo em aplicações de tempo real. A **alucinação de formato** ocorre quando o modelo gera rótulos fora do conjunto esperado — "Positivo" ou "Very Good" em vez de estritamente "positive" — exigindo pós-processamento robusto. O **custo computacional** de manter encoder e decoder completos é maior que o de arquiteturas encoder-only para tarefas que não exigem geração longa.

A evolução tem três marcos. Em **2020**, a era T5 unifica tarefas mas ainda depende de fine-tuning massivo para cada nova tarefa, com prefixos rígidos e curtos. Em **2021**, o Google introduz o **FLAN (Finetuned Language Net)** e o **instruction tuning**: em vez de treinar em dados brutos, treina-se o modelo para **seguir instruções** em milhares de tarefas diferentes. Hoje, graças a essa linhagem, modelos modernos realizam tarefas nunca vistas apenas recebendo uma descrição em linguagem natural, sem nenhum treino extra — a **generalização zero-shot**.

A conclusão da aula é que o T5 demonstrou que um modelo encoder-decoder pré-treinado em escala massiva pode generalizar para quase qualquer tarefa linguística, lançando as bases dos LLMs modernos, nos quais **a instrução (o prompt) define a tarefa, não a arquitetura**.

10 Aula Extra - Revisão

10.1 Estrutura da revisão

A aula de revisão, ministrada pelo Dr. Leonardo Forero Mendoza, é organizada em quatro horas temáticas: fundamentos (NLP clássico versus moderno, Transformers, tokenização), prompt engineering (zero-shot, few-shot, CoT, ToT), RAG e dados (embeddings, vector DBs, chunking, recuperação híbrida) e agentes e prática (arquiteturas, ferramentas, diagramas e ReAct).

10.2 Pré-processamento e tokenização

A revisão detalha a etapa de preparação do texto. A **tokenização** divide o texto em unidades menores: "O gato correu" torna-se `["O", "gato", "correu"]`. A **normalização** padroniza o texto por lowercase, remoção de pontuação e remoção de **stop words** (palavras de baixo valor informativo como "o", "a", "de", "para").

A distinção entre **stemming** e **lematização** é conceitualmente importante. O stemming corta sufixos heuristicamente — é rápido e menos preciso, transformando "correndo" em "corr", que nem sequer é uma palavra. A lematização reduz à forma base (lema) — é mais lenta e precisa, transformando "correndo" em "correr".

10.3 As três gerações de representação

A revisão organiza a evolução das representações em três camadas:

1. **Frequência (BoW, TF-IDF)**: conta ocorrências. Simples, mas perde contexto.
2. **Embeddings estáticos (Word2Vec)**: vetores densos que capturam significado, com a analogia `rei - homem + mulher ~ rainha`.

3. **Embeddings contextuais (Transformers)**: o significado muda com o contexto. "Banco" de dinheiro e "banco" de praça deixam de compartilhar o mesmo vetor.

A ilustração é direta. Com Word2Vec, `Vetor(banco_A) == Vetor(banco_B)` nas frases "Sentei no banco da praça" e "Paguei no banco central" — o que é um erro. Com Transformers, os vetores diferem, porque o **mecanismo de atenção** faz o modelo olhar para todas as outras palavras da frase simultaneamente ao decidir o significado de cada uma.

10.4 LLMs e suas limitações

LLMs são redes neurais profundas treinadas em escala massiva com um objetivo simples: prever a próxima palavra. A escala mencionada é da ordem de 175 bilhões de parâmetros e petabytes de dados. A **propriedade emergente** é o fenômeno em que "mais é diferente": capacidades complexas como raciocínio lógico, codificação e tradução surgem espontaneamente, sem terem sido explicitamente programadas.

O funcionamento da geração é a **next token prediction**: dado o prompt "O céu está", o modelo atribui probabilidades — "azul" 75%, "nublado" 15%, "caindo" 5%.

Três limitações justificam todo o resto do curso: **alucinações** (o modelo prioriza fluência sobre veracidade), **conhecimento estático** (o cérebro do modelo é congelado na data de corte do treinamento) e **ausência de dados privados** (ele não conhece e-mails, contratos ou bancos de dados da empresa).

A causa raiz da alucinação é explicitada: o modelo não busca fatos em um banco de dados, ele prevê a próxima palavra mais provável com base em padrões estatísticos — **plausibilidade acima de verdade**. O exemplo dado é a resposta confiante e falsa sobre um vencedor de Copa do Mundo em 2023, torneio que não ocorreu na categoria masculina.

10.5 Anatomia do prompt e técnicas

A revisão estrutura a anatomia de um prompt em cinco componentes: **persona** ("Aja como um Senior Developer"), **contexto** ("Analise este código Python de RAG"), **tarefa** ("Identifique erros de segurança e otimize a performance"), **formato** ("Responda em lista de bullet points") e **restrição** ("Não use jargão excessivo").

O contraste entre prompt ruim e prompt bom é ilustrativo. Ruim: "Explique o que é RAG." Bom: "Atue como um Engenheiro de ML Sênior. Explique o conceito de RAG para uma audiência de executivos. Use uma analogia simples (como uma prova com consulta), evite jargões técnicos e foque nos benefícios de negócio."

O ciclo de refinamento iterativo tem quatro passos — escrever, testar, analisar, refinar — e é demonstrado com a evolução de "Escreva um email" para "Escreva um email de vendas persuasivo para CFOs, focado em redução de custos, tom profissional e conciso".

No **Chain of Thought**, o exemplo canônico é o problema de Roger com bolas de tênis: sem CoT, o modelo frequentemente erra; com CoT, ele decompõe — Roger começa com 5, ganha 2 latas de 3 bolas cada, o que dá 6, e 5 mais 6 são 11.

No **Tree of Thoughts**, o modelo gera três soluções possíveis, avalia cada uma com um score (por exemplo, A com 4/10, B com 9/10, C com 6/10) e expande a melhor opção.

10.6 Tokens, API e custo

Tokens são pedaços de palavras ou caracteres. "Apple" é 1 token; "Hamburger" é 3 tokens ("Ham", "bur", "ger"). A regra de ouro apresentada é que 1.000 tokens correspondem a aproximadamente 750 palavras. Tokens determinam três coisas: custo, limite de contexto e latência.

A distinção entre **interface** e **API** é operacional. A interface (ChatGPT) é projetada para interação humana direta, é fácil de usar, tem histórico visual, mas é difícil de automatizar. A API é projetada para integração com software, é totalmente programável, escalável e permite controle fino dos parâmetros:

```
client.chat.completions.create(
    model="gpt-4",
    messages=[...],
)
```

10.7 O pipeline RAG consolidado

O RAG é apresentado com a analogia mais didática do curso: é a diferença entre fazer uma prova de memória (LLM puro) e uma prova com consulta ao livro (RAG). Três passos: **retrieve** (buscar documentos relevantes), **augment** (inserir os documentos no contexto do prompt) e **generate** (o LLM responde usando os fatos fornecidos).

O pipeline completo tem duas fases e seis etapas. Na **fase de indexação**: load (carregar PDFs, TXT, HTML), split (dividir em chunks) e embed (converter em vetores), store (salvar em vector database). Na **fase de recuperação**: retrieve (buscar chunks similares) e generate (responder usando o contexto).

Sobre **bancos vetoriais**, o contraste com SQL tradicional é esclarecedor. Uma consulta `SELECT * WHERE text LIKE '%gato%'` faz busca exata e falha se o texto contiver "felino". Uma consulta `search(query_vector, k=3)` faz busca semântica e encontra "felino", "bichano" e "pet" porque o significado é próximo. Pinecone, Chroma e Weaviate são citados.

A **Hybrid Search** resolve a fraqueza da busca vetorial em termos exatos (códigos, nomes próprios) combinando-a com BM25. O algoritmo de fusão citado é o **RRF (Reciprocal Rank Fusion)**, que combina os rankings das duas buscas.

O **re-ranking** usa um **cross-encoder**, que lê pergunta e documento juntos para entender a relevância real — recupera-se o top 50 e envia-se apenas o top 5 ao LLM.

10.8 Agentes de IA

Um **agente** é um sistema que usa LLMs não apenas para falar, mas para raciocinar, planejar e usar ferramentas para agir. A diferença-chave é sintética: o chatbot responde perguntas; o agente executa tarefas.

O **ReAct (Reasoning + Acting)** é o loop cognitivo fundamental, com três passos que se repetem: **Thought** (analisa o estado atual), **Action** (usa uma ferramenta) e **Observation** (lê o resultado). O exemplo transcrito é:

```
Thought: O usuário perguntou a idade do Brad Pitt. Preciso buscar essa informação.
Action: GoogleSearch("Brad Pitt age")
Observation: Brad Pitt is 60 years old (born 1963).
Thought: Tenho a resposta. Vou responder ao usuário.
Final Answer: Brad Pitt tem 60 anos.
```

Em pseudocódigo, o loop é:

```
while not final_answer:
    thought = llm(prompt)
    action = parse(thought)
    obs = tool(action)
    prompt += obs
```

As **ferramentas** não são clicadas pelo LLM: ele gera **chamadas de função** (function calling) que o sistema executa, no formato de um objeto com nome da ferramenta e argumentos. Quatro categorias são listadas: web search (Google, Bing, Tavily) para dados em tempo real; Python REPL para matemática e lógica; bases de dados (SQL, Vector DB) para memória de longo prazo; e APIs externas (Gmail, Slack, CRM) para ações no mundo real.

Criar uma ferramenta customizada é simples, e o material destaca que **a docstring é o manual que o LLM lê** para entender quando e como usar a ferramenta:

```
from langchain.tools import tool

@tool
def consultar_estoque(produto: str) -> str:
    """Retorna a quantidade de itens em estoque para um dado produto."""
    return "50 unidades"
```

A **memória** precisa ser gerenciada explicitamente porque LLMs são stateless. A memória de curto prazo é o histórico da conversa atual — rápida mas volátil e limitada pela janela de contexto. A memória de longo prazo é um banco vetorial — mais lenta, porém persistente e ilimitada.

```
from langchain.memory import ConversationBufferMemory

memory = ConversationBufferMemory(memory_key="chat_history")
agent = initialize_agent(tools, llm, memory=memory)
```

Sistemas **multi-agente** resolvem a limitação de um agente generalista: cria-se um time de especialistas coordenados por um gerente. O padrão apresentado tem um Manager que planeja e delega para Researcher (busca), Writer (escreve), Coder (programa) e Reviewer (verifica qualidade). As vantagens são a especialização (prompts focados) e o paralelismo.

Os **frameworks** se distinguem pelo nível de controle. **LangChain/LangGraph** oferece controle granular de estado e fluxo — blocos de construção do tipo Lego, com máxima flexibilidade. **CrewAI** foca em papéis e times, com abstração de alto nível e máxima facilidade.

```
from crewai import Agent, Task, Crew

researcher = Agent(
    role="Pesquisador Sênior",
    goal="Descobrir tendências de IA",
    backstory="Você é especialista em ...",
    tools=[search_tool],
)
task1 = Task(description="Pesquise sobre LLMs em 2025", agent=researcher)
crew = Crew(agents=[researcher], tasks=[task1])
crew.kickoff()
```

```
from langgraph.graph import StateGraph, END

workflow = StateGraph(AgentState)
workflow.add_node("search", search_node)
workflow.add_node("writer", writer_node)
workflow.set_entry_point("search")
workflow.add_edge("search", "writer")
workflow.add_edge("writer", END)
app = workflow.compile()
```

10.9 Onde os agentes falham

Três modos de falha são identificados: **loop infinito** (o agente tenta a mesma ação repetidamente sem sucesso), **alucinação de ferramenta** (inventa parâmetros ou resultados falsos) e **custo explosivo** (cada passo do loop consome tokens caros). A solução são **guardrails**:

```
agent = initialize_agent(
    tools, llm,
    max_iterations=5,
    early_stopping_method="force",
)
```

10.10 Avaliação, deploy, custo e segurança

A **avaliação** não deve confiar cegamente no modelo. Frameworks citados são RAGAS, LangSmith e Arize Phoenix, com métricas como faithfulness, answer relevance, context precision e latência, frequentemente calculadas com um **LLM juiz (LLM-as-a-Judge)**.

O **deploy** move o agente do notebook para a nuvem: frontend em Streamlit ou React, backend com LangServe sobre FastAPI, containerização com Docker e execução em AWS, Azure ou GCP.

A decisão entre **RAG e fine-tuning** é resumida em uma regra de ouro: use RAG para adicionar **conhecimento** (fatos novos, documentos privados, dados que mudam sempre) e fine-tuning para mudar o **comportamento** (tom de voz específico, formato JSON complexo, linguagem médica). A recomendação prática é começar quase sempre com RAG.

Em **segurança**, o ataque central é o **prompt injection**: o agente pode ser manipulado para revelar segredos ou agir de forma maliciosa. As defesas são delimitadores claros, validação de entrada e saída, e guardrails dedicados como o NeMo Guardrails. Ataques comuns citados: DAN (Do Anything Now), jailbreak e ofuscação.

Em **custo**, três estratégias: **model routing** (usar o modelo grande apenas quando necessário), **caching** e **prompt compression**. A dica prática é usar modelos pequenos para tarefas simples como resumir e classificar, reservando o modelo grande para raciocínio complexo.

10.11 Casos de uso e conclusão

Três casos são detalhados: agente de suporte nível 1 (consulta CRM e base de pedidos, escalando para humano apenas em casos complexos), agente de pesquisa de mercado (descoberta, deep dive em landing pages, síntese comparativa e relatório) e agente de codificação com auto-correção, em que o verdadeiro poder não é gerar código mas consertá-lo, por meio de um loop de escrever, executar testes, ler o erro e reescrever.

A conclusão organiza os quatro pilares do curso em uma progressão: **NLP** para entender, **LLM** para gerar, **RAG** para saber e **Agentes** para agir. A frase que sintetiza a mensagem é: "O valor não está apenas no modelo, mas no sistema inteligente que você constrói ao redor dele."

11 Transformers

11.1 O problema da serialização

A aula abre com uma analogia sobre paralelismo: uma mulher leva nove meses para gerar um bebê, porque o processo tem etapas sequenciais obrigatórias; nove mulheres não conseguem gerar um bebê em um mês. Adicionar recursos não acelera processos intrinsecamente seriais. **Redes Neurais Recorrentes (RNNs)** sofrem exatamente desse problema: para calcular o estado da palavra 100, é preciso ter calculado a palavra 99.

Isso conecta-se diretamente ao hardware. A **CPU** tem poucos núcleos (de 4 a 64), extremamente potentes, focada em terminar uma tarefa complexa o mais rápido possível — baixa latência — e ideal para tarefas sequenciais e lógica condicional. A **GPU** tem milhares de núcleos simples e especializados, focada em processar muitas tarefas simples ao mesmo tempo — alto throughput — e perfeita para álgebra linear. Como deep learning é essencialmente multiplicação de matrizes gigantes, GPUs são muito mais rápidas que CPUs para treinar redes neurais.

A consequência é decisiva. RNNs e LSTMs processam palavra por palavra, e a GPU precisa esperar o cálculo anterior terminar; a maior parte dos núcleos fica ociosa, com uso da ordem de 15%. O Transformer processa a frase inteira de uma vez como uma única matriz gigante, e todos os núcleos trabalham simultaneamente, com uso próximo de 98%. O material chama isso de "revolução silenciosa": o sucesso do Transformer não foi apenas algorítmico, foi a primeira arquitetura de PLN desenhada especificamente para explorar o hardware massivamente paralelo que já existia.

11.2 Linha do tempo

- **1966**: ELIZA, simulação simples de terapeuta baseada em regras.
- **1997**: LSTMs; Hochreiter e Schmidhuber resolvem o vanishing gradient.
- **2014**: Attention; Bahdanau et al. introduzem o mecanismo de atenção.
- **2017**: Transformer; "Attention Is All You Need" muda tudo.
- **2022**: ChatGPT; LLMs se tornam acessíveis ao público geral.

O princípio pedagógico é que cada nova arquitetura nasceu de uma limitação específica da anterior. A RNN resolve o processamento de sequências de tamanho variável mas cria o gargalo de contexto. A LSTM resolve o vanishing gradient mas mantém o processamento sequencial lento. A Attention dá acesso seletivo à informação mas ainda dependia de recorrência. O Transformer traz paralelização total mas cria contexto limitado por complexidade $O(n^2)$. O LLM traz capacidades emergentes via escala mas com custo computacional massivo.

11.3 Por que texto é diferente de imagem

Em visão computacional, os pixels têm posições fixas e a estrutura é estática — a entrada típica é uma matriz fixa. Em linguagem, a **ordem define o significado** e o contexto muda dinamicamente conforme a frase avança. "O banco aprovou o empréstimo" e "Sentei no banco do parque" usam a mesma palavra com sentidos incompatíveis. Redes feedforward convencionais não têm memória e processam cada entrada de forma independente; foi preciso uma arquitetura capaz de manter um estado ao longo do tempo, o que levou às RNNs.

11.4 RNNs e LSTMs

Uma **RNN** processa a sequência elemento por elemento mantendo um **estado oculto** que funciona como memória. Para cada token, combina o token atual com o estado oculto do passo anterior:

$$h_t = \tanh(W_{hh} \times h_{t-1} + W_{xh} \times x_t)$$

A inovação-chave são os **pesos compartilhados**: a mesma matriz W é reutilizada em cada passo de tempo, permitindo processar sequências de qualquer comprimento. O vetor h_t atua como um resumo comprimido de todo o histórico passado.

O **gargalo de contexto** é a consequência: o estado oculto final deve conter toda a informação da sequência. A analogia usada é tentar comprimir um livro inteiro em uma única frase — inevitavelmente detalhes cruciais são perdidos. Dois efeitos se somam: **perda de informação** (quanto mais longa a sequência, mais a informação é diluída pelas transformações sucessivas) e **vanishing gradient** (durante o treinamento, os gradientes diminuem exponencialmente ao retropropagar, de modo que as primeiras palavras têm influência quase nula no aprendizado). O resultado prático é que RNNs falham em reter informação após 10 a 20 passos.

As **LSTMs** (Hochreiter e Schmidhuber, 1997) introduzem o **estado da célula (cell state)**, uma "estrada expressa" para a informação fluir inalterada através da rede, resolvendo o vanishing gradient. Três portões controlam o fluxo:

1. **Forget gate**: decide qual informação do estado anterior deve ser descartada ("o sujeito mudou, esqueça o gênero antigo").
2. **Input gate**: decide qual nova informação deve ser armazenada no estado da célula.
3. **Output gate**: decide o que será enviado à próxima camada e ao próximo passo de tempo, com base no estado filtrado.

Ainda assim, LSTMs permanecem limitadas: uma sequência de 1.000 palavras requer 1.000 passos sequenciais, o que impede o processamento paralelo e desperdiça hardware. A comparação apresentada é eloquente: para 1.000 tokens, a LSTM leva cerca de 1 segundo com 20% de utilização de GPU, enquanto o Transformer leva cerca de 0,01 segundo com 95% de utilização — cem vezes mais rápido. Dependências longas são difíceis para a LSTM (cerca de 30 tokens) e fáceis para o Transformer (qualquer distância).

11.5 O mecanismo de Attention

A ideia revolucionária de Bahdanau, Cho e Bengio (2014) é formulada como pergunta: em vez de comprimir toda a sequência em um único vetor de tamanho fixo, por que não deixar o decodificador olhar para qualquer parte da sequência de entrada quando precisar?

A analogia é a leitura de um livro técnico: ao encontrar uma referência complexa, você não tenta lembrar de tudo de memória — você volta e relê especificamente a parte relevante. A mudança de paradigma é que o modelo deixa de depender de uma memória comprimida (estado oculto) e ganha **acesso aleatório** a todo o histórico, decidindo para cada palavra gerada onde prestar atenção e com que intensidade.

Query, Key e Value são explicados com uma analogia com busca de vídeos:

- **Query (Q)**: o que estamos procurando no momento. No YouTube, é o texto digitado na busca.
- **Key (K)**: o que usamos para comparar com a consulta. No YouTube, são títulos e tags dos vídeos.
- **Value (V)**: a informação real que queremos recuperar. No YouTube, é o conteúdo do vídeo em si.

O **Scaled Dot-Product Attention** tem quatro passos:

1. **Calcular compatibilidade**: multiplica-se a Query pelas Keys transpostas para encontrar a similaridade bruta, $Q \cdot K^T$.
2. **Escalar**: divide-se pela raiz da dimensão, $\sqrt{d_k}$, para evitar valores muito grandes que saturem o gradiente.
3. **Normalizar**: aplica-se softmax, convertendo scores em probabilidades que somam 1.
4. **Soma ponderada**: multiplicam-se os pesos pelos Values, obtendo o vetor de contexto final.

A divisão por $\sqrt{d_k}$ merece atenção especial. Quando as dimensões são grandes (por exemplo, 512), o produto escalar pode gerar números muito altos. Sem escala, scores como (100, 50, 80) produzem um softmax de (1,0; 0,0; 0,0) — saturação, com gradiente aproximadamente zero. Com o divisor $\sqrt{512} \approx 22{,}6$, os scores viram (4,4; 2,2; 3,5) e o softmax vira (0,6; 0,1; 0,3) — distribuição suave e gradientes saudáveis. A regra prática é que a divisão mantém a variância dos scores próxima de 1.

11.6 Self-Attention, Multi-Head e Cross-Attention

Self-Attention é o caso especial em que Query, Key e Value vêm todos da mesma sequência de entrada, permitindo que cada palavra "olhe" para todas as outras da frase para entender seu próprio contexto. A diferença-chave é entre o **embedding original** (significado da palavra isolada, como num dicionário) e o **context vector** (significado enriquecido com o contexto de toda a frase).

Um único mecanismo de atenção cria apenas uma distribuição de probabilidade, forçando o modelo a escolher um único foco principal por palavra. Isso é insuficiente por duas razões: **ambiguidade semântica** (em "O banco fechou cedo", um foco único pode confundir instituição e assento) e **conflito de funções** (em "O gato comeu a ração porque estava com fome", é preciso conectar "estava" a "gato" e a "fome" simultaneamente).

O **Multi-Head Attention** resolve isso dividindo os vetores em h partes (cabeças) e executando attention em paralelo, com cada cabeça aprendendo a focar em um aspecto diferente da linguagem. A analogia é a de um comitê de especialistas analisando a mesma frase: um gramático, um historiador e um poeta. Exemplos de especialização de cabeças: sintaxe (estrutura gramatical, sujeito-verbo), semântica (significado e relação entre entidades) e correferência (resolver a quem "ele" ou "ela" se refere).

Cross-Attention é a ponte entre entendimento (encoder) e geração (decoder). O encoder fornece Key e Value; o decoder fornece Query. A analogia é a do tradutor humano que, enquanto escreve a tradução, olha periodicamente para a frase original em busca de informação específica: a Query é a pergunta do tradutor ("estou traduzindo um verbo, qual é o sujeito na frase original?"), e Key e Value são a resposta do texto original.

11.7 A arquitetura Transformer

Em "Attention Is All You Need" (Vaswani et al., 2017), o **encoder** é uma pilha de seis camadas idênticas, cada uma com duas subcamadas — Multi-Head Self-Attention e uma rede Feed-Forward — cujo objetivo é criar uma representação contextualizada da entrada. O **decoder** também é uma pilha de seis camadas, mas insere uma terceira subcamada: a Cross-Attention, que olha para a saída do encoder. A grande ruptura foi dispensar completamente recorrência e convoluções, baseando-se inteiramente em mecanismos de atenção.

Ao redor de cada subcamada existem uma **conexão residual** (atalho) e uma **normalização de camada (LayerNorm)**. A conexão residual permite que informação e gradiente "pulem" camadas, criando uma superestrada para o gradiente fluir de volta durante o treinamento e resolvendo o vanishing gradient em redes profundas. O LayerNorm normaliza os vetores resultantes para média zero e variância unitária, estabilizando a dinâmica do treinamento e permitindo taxas de aprendizado maiores. A composição é: saída igual a LayerNorm de (x mais Sublayer(x)).

O **masking** preserva a causalidade. Durante o treinamento, alimenta-se o decoder com a frase inteira de uma vez para paralelizar; sem proteção, o modelo poderia simplesmente "olhar para a frente" e copiar a próxima palavra em vez de aprender a prevê-la. A solução é uma máscara triangular (look-ahead mask) que cobre as posições futuras: o score bruto de uma posição futura recebe menos infinito, e o softmax de menos infinito é zero — atenção nula.

O **Positional Encoding** resolve o problema da invariância. Como o Transformer processa todos os tokens simultaneamente, ele não tem noção inerente de ordem: "O gato comeu o rato" e "O rato

comeu o gato" seriam conjuntos idênticos de palavras. A solução é somar ao embedding de cada palavra um vetor especial construído com funções seno e cosseno de diferentes frequências:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d}) \quad \text{e} \quad PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d})$$

A **Feed-Forward Network** processa cada posição individualmente e de forma idêntica após a camada de atenção ter misturado informações entre palavras. Ela projeta o vetor para uma dimensão maior (por exemplo, de 512 para 2048), aplica uma não-linearidade (ReLU) e projeta de volta: $FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$. A divisão de trabalho é clara: enquanto a atenção foca em **roteamento** (buscar informações de outros lugares), a FFN foca em **processamento** — é ali que o modelo "pensa" sobre a informação que acabou de coletar.

11.8 Paralelização e dependências de longo alcance

Em RNN e LSTM, para conectar a palavra "A" à palavra "Z", a informação precisa atravessar todos os passos intermediários — um caminho de complexidade $O(N)$ que degrada o sinal como em uma brincadeira de telefone sem fio. No Transformer, a atenção cria uma estrada direta entre quaisquer duas palavras, com caminho $O(1)$: a palavra "Z" acessa "A" com a mesma facilidade com que acessa sua vizinha. Isso permitiu que modelos entendessem parágrafos inteiros, artigos longos e até livros sem esquecer o início.

O resultado prático é que o que antes levaria anos com LSTMs passou a levar dias ou semanas com Transformers, viabilizando o treinamento em datasets massivos.

11.9 Do Transformer aos LLMs

As LSTMs perderam em PLN por treinamento lento (a natureza sequencial impede paralelização) e por esquecimento catastrófico (a informação se dilui após algumas centenas de tokens). Mas sobrevivem em nichos: séries temporais simples com poucos dados, onde são mais leves e menos propensas a overfitting, e edge computing e IoT, onde o custo de memória constante $O(1)$ da inferência LSTM é vantajoso frente ao custo quadrático $O(N^2)$ do Transformer com o tamanho do contexto. A frase que resume: "O Transformer não matou a recorrência, ele apenas a expulsou do trono do Processamento de Linguagem Natural."

A equação do salto é: **Transformer (arquitetura) mais Big Data (a internet inteira) mais Compute (milhares de GPUs) igual a LLM**. O ponto surpreendente é a estabilidade da arquitetura: o GPT-4 não é radicalmente diferente do Transformer original de 2017. O que mudou não foi o algoritmo, mas a **escala** — descobriu-se que o Transformer escala previsivelmente. O paper original focava em tradução; ao treinar o modelo para simplesmente prever a próxima palavra em todo o texto da internet, ele foi forçado a aprender lógica, programação, história e senso comum apenas para completar as frases corretamente.

O **pré-treinamento** é **self-supervised**: não é preciso humanos rotularem dados, porque o próprio texto é o rótulo. Esconde-se a próxima palavra e pede-se ao modelo que a adivinhe; se errar, ajustam-se os pesos. O material organiza isso em três níveis crescentes: para acertar "O gato subiu no telhado", o modelo precisa aprender **sintaxe**; para acertar "A capital da França é Paris", precisa memorizar **conhecimento** sobre o mundo; para acertar "João é mais alto que Maria. Logo, Maria

é mais baixa", precisa entender **lógica e relações**. A conclusão é que "prever a próxima palavra com precisão exige entender o mundo que gerou aquela palavra".

A **emergência de capacidades** é descrita como uma transição de fase: assim como a água vira vapor a 100 graus, LLMs mostram novas habilidades abruptamente ao atingir certa escala de parâmetros e dados — habilidades que não foram programadas explicitamente. Exemplos citados: aritmética multi-dígito, programação (a partir de treino com código do GitHub) e teoria da mente.

11.10 Decoder-only, tokenização e geração

A arquitetura **decoder-only** é a escolha dominante para GPT e Llama. O Transformer original foi desenhado para tradução (ler A, gerar B), mas para criar inteligência geral percebeu-se que a tarefa fundamental é a **continuação de texto** (ler A, gerar mais A). Se não há uma frase de origem separada, o encoder é desnecessário: remove-se o encoder e as camadas de cross-attention, e o que resta é uma pilha pura de camadas de Self-Attention Mascarado e Feed-Forward.

A **tokenização** é a ponte entre linguagem humana e matemática: texto bruto vira tokens, que viram IDs numéricos. Não se usa palavras inteiras porque o vocabulário seria praticamente infinito e o modelo, gigantesco e ineficiente. O algoritmo **Byte Pair Encoding (BPE)** encontra o equilíbrio: palavras comuns viram um único token, palavras raras ou complexas são quebradas em pedaços menores. Isso permite representar qualquer texto com um vocabulário fixo da ordem de 50 mil tokens.

A **geração auto-regressiva** tem quatro passos em loop: o modelo recebe o contexto atual ("A capital do Brasil é"), calcula probabilidades e escolhe a próxima palavra ("Brasília"), adiciona a palavra escolhida ao final da entrada, e reinicia com o novo contexto. O gargalo de inferência é que, diferente do treinamento (paralelo), a geração é estritamente sequencial — o modelo não pode gerar a décima palavra antes da nona. É por isso que o ChatGPT "digita" a resposta na tela.

A **janela de contexto** é a memória de curto prazo do modelo: a quantidade máxima de texto (prompt mais resposta) que ele consegue ver simultaneamente. A complexidade é $O(N^2)$: como cada token presta atenção em todos os outros, dobrar o tamanho do texto quadruplica o custo computacional e de memória.

11.11 Fine-tuning, RLHF e evolução

Transformar um completador de texto em um assistente útil exige três etapas:

1. **Supervised Fine-Tuning (SFT)**: humanos criam pares pergunta-resposta e o modelo aprende o formato e como seguir instruções.
2. **Reward Modeling (RM)**: geram-se várias respostas, humanos as comparam, e treina-se um Reward Model que prevê preferências.
3. **Reinforcement Learning (PPO)**: o modelo se ajusta por aprendizado por reforço para maximizar a recompensa.

O objetivo é o **alinhamento**: o RLHF alinha o modelo aos três H — Helpful, Honest e Harmless.

A evolução dos LLMs segue as **Scaling Laws**, segundo as quais a performance escala como uma lei de potência com o aumento de dados e computação:

- **GPT-1 (2018)**: 117 milhões de parâmetros; prova de conceito do pré-treinamento não supervisionado.
- **GPT-2 (2019)**: 1,5 bilhão; gerou textos coerentes e chamou atenção pela capacidade.
- **GPT-3 (2020)**: 175 bilhões; grande salto, introduziu o few-shot learning.
- **GPT-4 (2023)**: cerca de 1,8 trilhão (estimado); melhorias em raciocínio e multimodalidade.

11.12 Exemplos práticos e limitações

O exemplo canônico de **Self-Attention em ação** é a resolução de ambiguidade: em "O troféu não cabe na mala porque ela é muito grande", "ela" pode se referir à mala ou ao troféu, e ambos são gramaticalmente válidos. O modelo olha para o adjetivo "grande" e, tendo aprendido que coisas grandes não cabem em coisas pequenas, atribui peso alto (0,85) à conexão entre "ela" e "troféu".

Em **tradução neural**, ao gerar a palavra "gato", o decoder usa cross-attention para focar intensamente na representação vetorial de "cat" produzida pelo encoder, ignorando palavras irrelevantes naquele momento. O alinhamento é **suave** e opera sobre **conceitos**, não palavra a palavra: em "The black cat", o modelo focaria em "black" e "cat" simultaneamente para gerar "O gato preto".

Quatro limitações dos LLMs atuais fecham a aula: **alucinações** (máquinas probabilísticas, não bancos de fatos; priorizam plausibilidade sobre verdade), **viés e estereótipos** (o modelo é um espelho da internet e reproduz preconceitos históricos a menos que fortemente alinhado via RLHF), **raciocínio lógico** (ainda lutam com lógica formal, matemática complexa e planejamento de longo prazo — são ótimos em imitar raciocínio, mas falham na execução rigorosa) e **conhecimento estático** (o conhecimento termina na data de corte do treinamento).

11.13 O futuro além dos Transformers

Três direções são apontadas. **State Space Models (Mamba)** combinam o treinamento paralelo dos Transformers com a inferência rápida das RNNs, prometendo janelas de contexto de milhões de tokens com baixo custo de memória, em complexidade $O(N)$. **Mixture of Experts (MoE)** ativa apenas os especialistas relevantes em vez de todo o modelo para cada palavra, permitindo modelos com trilhões de parâmetros que rodam tão rápido quanto modelos menores. **Multimodalidade nativa** significa modelos que não apenas leem texto, mas ouvem áudio e veem vídeo nativamente por tokenização direta, sem conversores externos.

A metáfora de fechamento: "O Transformer pode não ser o fim da história, mas sim o transistor da era da IA: o bloco fundamental sobre o qual construiremos sistemas ainda mais complexos."

12 Text-to-Text

Material de slides não disponível para esta aula. Os arquivos de apoio listados são a gravação da aula, uma imagem da camada do decoder, uma apresentação sobre "Text-to-Text em LLMs: Do Conceito à Aplicação Empresarial" e três notebooks: um de exercício de text-to-text com a API da OpenAI voltado a geração de SQL, um notebook principal de aula sobre text-to-text com OpenAI, e um notebook de decodificação de tokens.

12.1 O que o material permite consolidar

O fundamento conceitual desta aula foi apresentado nos slides sobre modelos text-to-text (Aula 8) e nos slides sobre Transformers (Aula 10), e a prática se apoia sobre eles.

O paradigma **text-to-text** trata todo problema de PLN como geração de texto: a entrada é uma string que contém, ela própria, a especificação da tarefa; a saída é uma string. Esse enquadramento significa que a fronteira entre "tarefa de classificação", "tarefa de extração" e "tarefa de geração" desaparece do ponto de vista da engenharia — o que muda é apenas o prefixo ou a instrução.

O notebook de **text-to-SQL** é um caso de uso empresarial direto desse paradigma: a entrada é uma pergunta em linguagem natural mais o esquema do banco; a saída é uma consulta SQL válida. O padrão de prompt segue os componentes já estudados — papel, contexto (o esquema das tabelas), tarefa e restrições de formato (retornar apenas SQL, sem explicação).

```
from openai import OpenAI

client = OpenAI()

schema = """
tabela vendas(id, produto, data_venda, valor, regiao)
"""

resposta = client.chat.completions.create(
    model="gpt-4",
    temperature=0,
    messages=[
        {"role": "system",
         "content": "Você converte perguntas em SQL. Retorne apenas a consulta."},
        {"role": "user",
         "content": f"Esquema:\n{schema}\nPergunta: total de vendas por região em 2025"},
    ],
)
```

O uso de `temperature=0` é coerente com o princípio estudado na aula de prompt engineering: tarefas factuais e de geração de código pedem baixa aleatoriedade e alta consistência.

O notebook de **decodificação de tokens** conecta-se diretamente ao conteúdo da Aula 10 sobre tokenização e geração auto-regressiva: observar quais tokens o modelo produz, e com que probabilidades, torna concreto o mecanismo de next token prediction e o funcionamento do BPE.

13 Modelos Text-to-Text II

Material de slides não disponível para esta aula. Os arquivos de apoio são os mesmos da Aula 11 — gravação, imagem da camada de decoder, apresentação sobre text-to-text em LLMs no contexto empresarial e os notebooks de text-to-SQL, de aula e de decodificação de tokens.

13.1 Continuidade e aprofundamento

Tratando-se da continuação direta da Aula 11, esta sessão aprofunda a aplicação empresarial do paradigma text-to-text com os mesmos materiais. Com base no que o conjunto de slides da disciplina permite afirmar, os pontos de atenção prática que fecham essa dupla de aulas são três.

O primeiro é a **latência de inferência**. Como visto nos slides de text-to-text, gerar a saída token por token é significativamente mais lento que uma classificação direta. Em aplicação empresarial de tempo real, isso é um requisito de projeto, não um detalhe.

O segundo é a **alucinação de formato**. Modelos text-to-text podem gerar rótulos ou estruturas fora do conjunto esperado, o que exige pós-processamento robusto. Em text-to-SQL, isso se traduz em validar sintaticamente a consulta gerada antes de executá-la e em jamais executar SQL gerado por LLM diretamente contra um banco de produção sem camada de verificação.

O terceiro é o **custo computacional**. Manter encoder e decoder completos exige mais memória e processamento do que arquiteturas encoder-only para tarefas que não demandam geração longa. A recomendação prática, coerente com a estratégia de model routing apresentada na revisão, é usar modelos menores para tarefas simples e reservar os maiores para raciocínio complexo.

14 Síntese da Disciplina

A disciplina se organiza em torno de uma tese central, formulada explicitamente na aula de revisão: **o valor não está apenas no modelo, mas no sistema inteligente construído ao redor dele**. Os doze encontros podem ser lidos como a construção progressiva desse sistema, em quatro camadas que a própria revisão nomeia: NLP para entender, LLM para gerar, RAG para saber e Agentes para agir.

A primeira camada estabelece que máquinas não entendem texto — entendem números — e que a qualidade da representação determina a qualidade de tudo o que vem depois. A trajetória de BoW e TF-IDF até os embeddings densos do Word2Vec e, daí, aos embeddings contextuais dos Transformers, é a trajetória de uma pergunta que se refina: primeiro contamos palavras, depois pesamos sua raridade, depois aprendemos seu significado a partir do contexto em que aparecem, e finalmente aceitamos que o significado **é** o contexto. A similaridade do cosseno, introduzida logo na primeira aula, atravessa todo o curso — é a mesma métrica que compara documentos com TF-IDF, que ordena chunks em um pipeline RAG e que decide onde cortar um texto no semantic chunking.

A segunda camada explica **por que** os Transformers venceram, e a resposta é dupla: algorítmica e material. Algoritmicamente, o mecanismo de atenção substituiu a memória comprimida do estado oculto por acesso direto a qualquer posição da sequência, transformando um caminho $O(N)$ em $O(1)$ e eliminando o esquecimento catastrófico. Materialmente, ao abolir a recorrência, a arquitetura tornou-se compatível com o paralelismo massivo das GPUs, elevando a utilização de hardware de cerca de 15% para cerca de 98%. As duas coisas juntas viabilizaram treinar em escala de internet, e a escala, por sua vez, produziu capacidades emergentes que ninguém programou explicitamente. Descobriu-se que prever a próxima palavra com precisão exige, como subproduto, aprender sintaxe, conhecimento factual e raciocínio.

A terceira camada confronta os limites do que a escala sozinha entrega. LLMs alucinam porque priorizam plausibilidade sobre verdade; têm conhecimento congelado na data de corte; e não conhecem

dados privados. O **RAG** responde às três limitações com um pipeline de seis etapas — load, split, embed, store, retrieve, generate — mas o curso é insistente em que o elo mais frágil é a recuperação, não a geração. Daí a atenção detalhada ao chunking (fixed-size, recursivo, semântico), à calibração de `chunk_size` e `chunk_overlap`, à busca híbrida que combina vetores com BM25, ao re-ranking com cross-encoders, e ao efeito "Lost in the Middle" que torna a **ordem** do contexto tão importante quanto seu conteúdo. A aula de OCR acrescenta uma lição frequentemente subestimada: antes de qualquer sofisticação de recuperação, é preciso extrair o texto corretamente do PDF — e o princípio GIGO governa todo o resto.

A quarta camada transforma o sistema de reativo em ativo. A **engenharia de prompt** deixa de ser um conjunto de truques para compensar limitações do modelo e passa a ser a especificação de comportamento: papel, objetivo, plano, critérios de decisão, critérios de sucesso e limites. O prompt agentic é a ponte natural para os **agentes**, que operam no loop ReAct — pensar, agir, observar — e usam ferramentas por function calling. Aqui o curso é honesto sobre os riscos: loops infinitos, alucinação de ferramenta, custo explosivo e prompt injection, todos exigindo guardrails explícitos. E é honesto também sobre as escolhas de projeto: RAG para adicionar conhecimento, fine-tuning para mudar comportamento; modelos pequenos para tarefas simples, modelos grandes reservados ao raciocínio complexo.

O paradigma **text-to-text** do T5, apresentado nas aulas finais, é a síntese conceitual mais elegante do curso: se toda tarefa de PLN pode ser expressa como uma transformação de texto em texto, então a arquitetura deixa de definir a tarefa e a **instrução** passa a defini-la. Essa mudança, iniciada com prefixos rígidos em 2020 e generalizada com o instruction tuning do FLAN em 2021, é exatamente o que torna possível a engenharia de prompt como disciplina — e fecha o círculo do curso, ligando a arquitetura Transformer estudada na Aula 10 às técnicas de prompting estudadas na Aula 5.

O que fica, ao final, é menos um catálogo de técnicas e mais um método de raciocínio: cada arquitetura nasceu da limitação da anterior, cada camada de sistema existe para compensar uma fraqueza específica do modelo, e projetar bem significa saber exatamente qual fraqueza se está compensando. Como resume o material sobre Transformers, a arquitetura pode não ser o fim da história, mas é o transistor da era da IA — o bloco fundamental sobre o qual sistemas ainda mais complexos serão construídos.