Pós-Graduação em Inteligência Artificial Generativa e LLMs

PUC-Rio

Desenvolvimento de Aplicações de IA

Notas de Aula

Vítor Wilher

DAI · 2025.2

Versão 1.0

Resumo. Construção de aplicações de IA ponta a ponta: arquitetura, integração de modelos, avaliação e deploy.

Palavras-chave: aplicações de IA; arquitetura; deploy

Notas de estudo elaboradas a partir do material das aulas.

Índice

1 Visão Geral da Disciplina

A disciplina **DAI — Desenvolvimento de Aplicações de IA** trata da construção de sistemas de inteligência artificial que operam sobre mais de uma modalidade de dado: texto, imagem, áudio, vídeo, tabelas e diagramas. O fio condutor é a passagem do modelo isolado para o sistema em produção. Não basta saber que existe um modelo capaz de transcrever fala ou descrever uma imagem; é preciso decidir como esse modelo se encaixa num pipeline, que representação intermediária ele produz, como essa representação é armazenada e recuperada, e como o conjunto é avaliado, explicado e implantado.

O percurso começa pelo sinal acústico. A primeira aula trata do processamento de áudio multimodal e a segunda desdobra esse tema em duas frentes: o reconhecimento de locutor (quem falou, quando e o quê) e o Visual Question Answering, isto é, a resposta a perguntas em linguagem natural sobre conteúdo visual. Em seguida, a disciplina apresenta a peça conceitual que unifica todas as modalidades: os **embeddings multimodais**, vetores densos que projetam texto, imagem e áudio em um mesmo espaço latente, permitindo que a similaridade geométrica represente similaridade semântica independentemente da origem do dado.

A partir dessa base, o curso avança para a camada de extração e de aplicação. A aula de OCR e YOLO cobre a leitura de texto em imagens e a detecção de objetos, complementando a visão computacional com técnicas de reconhecimento estruturado. A aula de RAG Multimodal é o ponto de convergência da disciplina: reúne parsing de documentos, sumarização por modelos de visão-linguagem, indexação multi-vetor, bancos vetoriais e orquestração agêntica em um único sistema capaz de responder perguntas sobre PDFs que combinam texto, tabelas, imagens e diagramas.

As três últimas frentes fecham o ciclo de engenharia: **métricas para LLMs**, que fornece o vocabulário de avaliação; **Explainable AI**, que trata da interpretabilidade de modelos e da atribuição de decisões a evidências; e **Deploy**, que discute a transformação de protótipos em serviços reutilizáveis. Essa sequência corresponde ao arco natural de um projeto real: representar, recuperar, gerar, medir, explicar e publicar.

2 Processamento de Áudio Multimodal

Material de slides não disponível para esta aula.

Esta aula abre a disciplina com o tratamento do sinal de fala e sua relação com os modelos generativos. O material de apoio da disciplina permite situar o tema: o áudio é a modalidade em que a distância entre o dado bruto e a semântica é maior. Um sinal acústico é uma série temporal contínua de amplitudes; para que um modelo de linguagem possa raciocinar sobre ele, é necessário atravessar uma cadeia de representações.

2.1 Do sinal à representação

A cadeia canônica de representação de áudio, apresentada em detalhe na aula de embeddings multimodais, segue quatro etapas:

1. O **sinal de áudio bruto** é convertido em **Mel-Spectrogram**, uma representação tempo-frequência que aproxima a percepção auditiva humana.

2. Uma **CNN** extrai características locais desse espectrograma.
3. Um **Transformer** captura as dependências temporais de longo alcance.
4. O **embedding final** representa o conteúdo semântico do trecho.

Modelos como **Wav2Vec 2.0** e **HuBERT** aprendem essas representações por **aprendizado auto-supervisionado**, ou seja, sem rótulos manuais, explorando a estrutura interna do próprio sinal. Uma consequência prática relevante é que tipos distintos de áudio (fala, música, sons ambientes) formam **clusters distintos** no espaço de embedding, o que permite classificação e busca sem treinamento supervisionado adicional.

2.2 Direções para o restante da disciplina

O material aponta que a evolução dos sistemas de fala caminha para **Audio LLMs**, modelos nativos de áudio com melhor compreensão acústica, e para a **multimodalidade** propriamente dita, com integração de pistas de áudio e visuais. Essas duas direções são exatamente os temas das aulas seguintes.

3 Vision QA

Esta aula reúne dois blocos: o reconhecimento de locutor aplicado a LLMs, com material completo de slides, e os modelos multimodais aplicados a **Visual Question Answering (VQA)**, cujo material de slides não está disponível no contexto.

3.1 O desafio fundamental do reconhecimento de locutor

Em cenários com múltiplos participantes não basta converter fala em texto. É preciso atribuir cada trecho a quem o produziu e estruturar a informação para uso posterior. A pergunta central é **"quem falou, quando e o quê?"**. Em reuniões, atendimento e assistentes, identificar o autor da fala é essencial para resumos, análise e contexto.

O problema se decompõe em duas tarefas complementares:

- **Speaker Diarization** — segmentação do áudio por locutor.
- **Speaker Verification** — verificação da identidade de um locutor.

3.2 Arquitetura em cascata

Sistemas tradicionais operam por um **pipeline modular**, com componentes independentes, o que oferece flexibilidade para lidar com um número variável de locutores. O pipeline tem três estágios:

1. **VAD (Voice Activity Detection)**: classificação binária que identifica quais segmentos contêm fala.
2. **Embedding Extractor**: codifica características acústicas em vetores no espaço latente.
3. **Clustering e Diarizer**: agrupa embeddings por similaridade e refina os rótulos finais.

A modularidade tem custo. As limitações documentadas da cascata são três:

- **Propagação de erros**: falhas na diarização se propagam para o módulo de ASR (reconhecimento automático de fala), degradando a transcrição.
- **Fala sobreposta**: o VAD tradicional assume segmentos com um único locutor, o que torna a sobreposição problemática.
- **Falta de otimização conjunta**: os módulos de diarização e ASR são treinados de forma independente, impedindo sinergia entre eles.

3.3 Arquitetura end-to-end

A alternativa é um **modelo neural único**. Modelos como o **Sortformer** processam áudio bruto e geram rótulos de locutor por frame. As vantagens críticas são a **otimização conjunta** de todos os componentes e a **menor propagação de erro** entre módulos. O trade-off é claro: qualidade superior em cenários complexos, porém menos flexibilidade para número dinâmico de locutores.

Nas soluções end-to-end, a informação de contexto de locutor informa decisões de ASR, e vice-versa. Não há suposição de segmentos single-speaker, o que produz manejo natural da sobreposição.

3.4 Speaker Verification: autenticação biométrica

Ao contrário da diarização, que segmenta, a verificação extrai representações numéricas (**embeddings**) que capturam características únicas da voz de um indivíduo. O processo 1-para-1 tem três passos:

1. **Enrollment**: um modelo neural processa uma amostra e gera um *voiceprint* de referência.
2. **Verificação**: uma nova amostra gera um novo embedding.
3. **Scoring**: comparação usando **similaridade cosseno**.

A métrica de operação é o **Equal Error Rate (EER)**, o ponto em que a taxa de falsos positivos iguala a de falsos negativos. Modelos modernos alcançam EER abaixo de 2%.

3.5 Benchmarking: acurácia versus latência

Em datasets de áudio curto (até 10 s), o modelo **ECAPA-TDNN** alcança o menor EER, de 1,71%, demonstrando capacidade discriminativa superior, com tempo de inferência de 69,43 ms — um balanço favorável entre performance e eficiência. O **PyAnnote** é mais rápido, sendo indicado para cenários com restrições computacionais severas.

3.6 Comparação de tecnologias de diarização

No espaço open-source, **PyAnnote** e **NVIDIA NeMo** dominam. Entre as APIs comerciais, a **AssemblyAI** destaca-se por acurácia em ambientes ruidosos, com 30% de melhoria documentada, enquanto a **Deepgram** prioriza velocidade.

Há também a distinção entre modos de operação. O modo **batch** aguarda o áudio completo antes de processar, sendo o padrão do PyAnnote. O modo **streaming** atribui rótulos em tempo real e é essencial para *voice agents* e legendagem ao vivo, disponível em APIs comerciais e no NeMo.

3.7 O paradigma SpeakerLM

O **SpeakerLM** é um MLLM (modelo de linguagem multimodal) que responde "who spoke when and what" em um único fluxo, integrando **Audio Encoder**, **Projector** e **LLM**. Seu mecanismo de registro de locutor é flexível e adapta-se a cenários reais: locutores desconhecidos, locutores conhecidos representados por embeddings e casos com redundância de registro. O material registra que o SpeakerLM supera baselines em cascata do estado da arte em benchmarks públicos quando há dados suficientes.

3.8 Integração com LLM para pós-processamento

Uma estratégia intermediária, e de excelente custo-benefício, é reaproveitar sistemas em cascata existentes e refinar sua saída com um LLM. O fluxo é:

```
saída da cascata -> LLM (GPT-4 / Claude) -> transcrição refinada
```

Sistemas em cascata geram pares (transcrição, `rotulo_locutor`) que frequentemente contêm inconsistências. LLMs aplicam raciocínio de alto nível para corrigi-las. As correções típicas são:

- **Desalinhamentos**: erros temporais entre diarização e ASR.
- **Rótulos ambíguos**: consolidação de rótulos inconsistentes, por exemplo o mesmo locutor rotulado ora como A, ora como B.
- **Contexto**: correção de erros de transcrição que violam o contexto conversacional.

3.9 Arquitetura de sistema completo

Um sistema em produção integra múltiplas camadas: pré-processamento, diarização (VAD, embeddings, clustering), transcrição por ASR, alinhamento temporal e pós-processamento com LLM. A saída estruturada alimenta análises *downstream*, como sentimento por locutor e resumos contextualizados.

3.10 Desafios persistentes e soluções emergentes

Três desafios se mantêm. A **fala sobreposta** confunde sistemas tradicionais de VAD. **Ambientes ruidosos** degradam severamente a extração de embeddings. A **escalabilidade** de processar centenas de horas de áudio em tempo real exige alta capacidade computacional.

As soluções apontadas são a **detecção neural** de atividade por locutor em modelos end-to-end, o uso de **modelos pré-treinados** como o **WavLM**, que reduzem a necessidade de dados massivos e melhoram a resiliência a ruído, e a **otimização** por quantização e destilação de conhecimento para viabilizar inferência rápida em streaming.

3.11 Aplicações práticas

O material apresenta quatro domínios de aplicação: **call centers**, onde bancos implementam biometria de voz para autenticação sem senha, reduzindo atrito e aumentando segurança; **assistentes de voz**, que fornecem experiências personalizadas com controles de acesso por membro da família; **telessaúde**, onde plataformas verificam provedores de saúde ao acessar registros eletrônicos (EHR), garantindo conformidade; e **pagamentos e segurança**, com autorização de transações por voz e aplicações forenses de identificação de locutor.

3.12 Recomendações de adoção

O material fecha com um roteiro pragmático de escolha tecnológica:

- **Para validação rápida**: comece com APIs comerciais para prototipar sem complexidade de infraestrutura.
- **Para controle e custo**: migre para frameworks open-source (PyAnnote, NVIDIA NeMo) quando o volume de áudio justificar GPUs dedicadas.
- **Para cenários complexos**: explore modelos end-to-end e MLLMs se o domínio tiver alta sobreposição de fala.
- **Toque final**: implemente sempre uma camada de pós-processamento com LLM para refinar a saída da diarização com base no contexto.

Como recursos de referência, o material aponta os frameworks **PyAnnote Audio**, **NVIDIA NeMo** e **SpeechBrain**, e os artigos fundamentais *SpeakerLM* (2025), *ECAPA-TDNN* (2020) e *WavLM* (2022).

3.13 Modelos multimodais e VQA

Material de slides não disponível para esta parte da aula.

O bloco de Visual Question Answering trata de modelos que recebem uma imagem e uma pergunta em linguagem natural e produzem uma resposta textual. Os fundamentos necessários para compreendê-lo aparecem nas aulas seguintes: o alinhamento entre espaços visual e textual (aula de embeddings multimodais) e o uso de **VLMs** (Vision Language Models) como GPT-4o, Claude e LLaVA para descrever, ler e raciocinar sobre conteúdo visual (aula de RAG multimodal).

4 Embeddings multimodais

Esta é a aula conceitualmente central da disciplina. Ela estabelece a representação que torna possível tudo o que vem depois: busca semântica, busca cross-modal, RAG multimodal e sistemas agênticos de recuperação.

4.1 O que são embeddings

Um **embedding** mapeia dados de alta dimensão — texto, imagem, áudio — para um espaço vetorial contínuo de dimensão reduzida, onde a **proximidade geométrica reflete similaridade semântica**. Como sintetiza a citação usada no material, atribuída a Andrej Karpathy, "embeddings are the lingua franca of modern AI systems".

As propriedades fundamentais são quatro:

- capturam relações semânticas latentes;
- permitem operações algébricas com significado (o exemplo clássico: *king* menos *man* mais *woman* aproxima *queen*);
- são transferíveis entre tarefas (*transfer learning*);
- são eficientes para busca por similaridade.

4.2 Embeddings unimodais

Texto. Modelos como BERT, GPT e T5 produzem embeddings contextualizados que capturam significado semântico das palavras, contexto sintático e pragmático, relações entre entidades e conceitos, e intenção e tom. As dimensões típicas são 768 para BERT-base e 1536 para o `text-embedding-3-small`.

Imagem. A arquitetura **Vision Transformer (ViT)** divide a imagem em *patches* e os processa como sequências de tokens, extraindo representações hierárquicas: camadas iniciais capturam bordas, texturas e gradientes; camadas intermediárias, formas e partes de objetos; camadas finais, conceitos semânticos e cenas.

Áudio. Conforme detalhado na primeira aula, Wav2Vec 2.0 e HuBERT produzem embeddings via aprendizado auto-supervisionado.

4.3 O desafio do alinhamento multimodal

O problema fundamental é que dados de naturezas diferentes não se misturam nativamente. As modalidades diferem em natureza, dimensionalidade e características:

Modalidade	Natureza	Características
Texto	Discreto, sequencial (1D, tokens)	Semântica densa, gramática
Imagem	Contínuo, espacial (2D/3D, pixels)	Textura, forma, cor
Áudio	Temporal, contínuo (1D, ondas)	Frequência, ritmo

O objetivo é projetar todas as modalidades em um **espaço latente compartilhado**, onde semântica equivalente corresponde a proximidade vetorial.

4.4 Aprendizado contrastivo

O **aprendizado contrastivo** treina modelos maximizando a similaridade entre **pares positivos** — dados semanticamente relacionados de modalidades diferentes — e minimizando a similaridade entre **pares negativos**. A intuição é direta: "uma foto de um cachorro" e uma imagem de cachorro devem ter embeddings próximos; uma foto de cachorro e "uma receita de bolo" devem ter embeddings distantes.

A grande vantagem é não exigir anotações detalhadas: bastam pares de dados relacionados, abundantes na internet. A função de perda associada é a **NT-Xent**, também referida como **InfoNCE Loss**, perda comum para treinamento contrastivo em minibatches.

4.5 CLIP

O **CLIP (Contrastive Language-Image Pre-training)**, publicado pela OpenAI em 2021, treinou em **400 milhões de pares imagem-texto** coletados da internet, aprendendo um espaço de embedding compartilhado para texto e imagem. Suas capacidades emergentes são a classificação zero-shot em qualquer categoria, a busca de imagens por texto e vice-versa, a compreensão visual baseada em linguagem natural e a transferência para domínios não vistos no treinamento.

A arquitetura é **dual-encoder**, com dois encoders independentes:

- **Image Encoder**: Vision Transformer (ViT-B/32, ViT-L/14) ou ResNet; processa a imagem em patches de 32 por 32 pixels e produz embedding de 512 ou 768 dimensões.
- **Text Encoder**: Transformer de estilo GPT; tokeniza o texto com **BPE (Byte Pair Encoding)** e produz embedding de mesma dimensão.

Ambos os embeddings são normalizados a norma unitária antes de calcular a similaridade via produto escalar.

A classificação **zero-shot** com CLIP segue quatro passos:

1. construir prompts textuais no formato "a photo of a {classe}";
2. calcular os embeddings de todos os prompts;
3. calcular o embedding da imagem de consulta;
4. retornar a classe com maior similaridade cosseno.

O resultado documentado é que o CLIP atinge performance comparável a modelos supervisionados no ImageNet sem ver nenhuma imagem de treinamento do dataset.

4.6 Além do CLIP

Após o CLIP, uma família de modelos multimodais foi desenvolvida, cada um expandindo capacidades: **ALIGN** (Google, 2021), com escala ainda maior, de 1,8 bilhão de pares; **Florence** (Microsoft, 2021), voltado a múltiplas tarefas visuais; **BLIP-2** (Salesforce, 2023), ponte entre imagem e LLM; **ImageBind** (Meta, 2023), com seis modalidades em um espaço; e **Gemini** (Google, 2023), cobrindo texto, imagem, áudio e vídeo.

O **ImageBind** merece atenção especial. Ele alinha seis modalidades em um único espaço usando **imagens como âncora**: texto e imagem via treinamento estilo CLIP; áudio e imagem via pares

vídeo-áudio; vídeo e imagem via frames; profundidade e imagem via datasets RGB-D; e IMU e imagem via dados de acelerômetro. O fenômeno emergente é que modalidades **nunca treinadas juntas** — áudio e texto, por exemplo — ficam alinhadas indiretamente pela âncora comum.

4.7 Similaridade vetorial

Depois de gerar embeddings é preciso medir quão similares dois vetores são. As métricas principais são:

Métrica	Intervalo	Propriedade
Cosseno	de -1 a 1	Invariante à magnitude
Euclidiana	de 0 a infinito	Sensível à magnitude
Produto escalar	irrestrito	Magnitude mais ângulo
Manhattan	de 0 a infinito	Robusto a outliers

A **similaridade cosseno** mede o ângulo entre dois vetores, ignorando magnitude. Formalmente, para vetores u e v de dimensão d, é o produto escalar dividido pelo produto das normas. As razões para preferi-la em embeddings são quatro: em embeddings normalizados o cosseno equivale ao produto escalar; o produto escalar é altamente otimizado em hardware (SIMD, GPU); a invariância à magnitude evita viés por comprimento de texto; e o espaço de embedding é tipicamente hiperesférico.

4.8 Busca semântica

A busca semântica recupera documentos com base em **significado**, não em correspondência lexical. As diferenças em relação à busca por palavras-chave são sistemáticas:

Aspecto	Palavras-chave	Busca semântica
Mecanismo	Matching de strings	Similaridade vetorial
Sinonímia	Não captura	Captura naturalmente
Intenção	Ignora contexto	Considera contexto
Escalabilidade	O(n) com índice invertido	O(log n) com ANN

O exemplo dado é elucidativo: a consulta “cachorro latindo” encontra documentos sobre “cão fazendo barulho” mesmo sem palavras em comum.

O pipeline completo opera em duas fases. Na **indexação (offline)**: coletar o corpus de documentos, imagens ou áudio; gerar embeddings com modelo pré-treinado; construir o índice vetorial no banco de dados; armazenar metadados associados. Na **busca (online)**: receber a query do usuário em qualquer modalidade; gerar o embedding da query; buscar os K vizinhos mais próximos no índice; retornar documentos ranqueados por similaridade.

4.9 Bancos de dados vetoriais

Bancos vetoriais são sistemas especializados para armazenar e recuperar embeddings. Diferem de bancos tradicionais em quatro dimensões: a **query** é de similaridade aproximada e não de igualdade exata; o **índice** usa estruturas para ANN e não B-trees; a **escala** envolve bilhões de vetores de alta dimensão; e as **operações** são busca k-NN, busca por faixa e filtragem.

FAISS (Facebook AI Similarity Search), desenvolvido pelo Meta, é uma biblioteca C++/Python para busca eficiente de similaridade em vetores densos. Suporta CPU e GPU (CUDA), oferece múltiplos algoritmos de indexação, busca exata e aproximada, escala a bilhões de vetores e tem licença MIT. Sua limitação é ser uma biblioteca, não um banco completo: não tem persistência nativa, API de rede nem gerenciamento de metadados.

Os algoritmos de indexação do FAISS oferecem trade-offs distintos:

- **IndexFlat**: busca exata por força bruta, máxima precisão, custo O(n).
- **IndexIVF**: particiona em clusters de Voronoi e busca apenas nos k clusters mais próximos, custo O(n/k).
- **IndexPQ**: comprime vetores via quantização de produto, reduzindo memória de 8 a 64 vezes.
- **IndexHNSW**: grafo navegável hierárquico, com alta velocidade e recall.

ChromaDB é um banco vetorial open-source projetado para aplicações de IA, com foco em simplicidade. Seus diferenciais são a API Python extremamente simples, o modo embedded sem servidor para prototipagem, a integração nativa com LangChain e LlamaIndex, o suporte a múltiplas funções de embedding e a filtragem por metadados com sintaxe simples. É ideal para RAG, chatbots, prototipagem rápida e desenvolvimento.

Qdrant é escrito em **Rust** e projetado para produção com requisitos rigorosos de performance e disponibilidade. Sua arquitetura inclui engine HNSW otimizado para ANN, sharding e replicação automáticos, WAL (Write-Ahead Log) para durabilidade, APIs REST e gRPC, e suporte a vetores densos, esparsos e multi-vetores. Alcança até 47.700 QPS em benchmarks, superando concorrentes por fatores de 2 a 4 vezes.

A recomendação prática do material é direta: começar com ChromaDB para validar a ideia e o pipeline de RAG, e migrar para Qdrant quando for necessário escalar para produção com alta disponibilidade.

4.10 Busca aproximada por vizinhos (ANN)

A busca exata (k-NN) compara a query com **todos** os vetores do banco, o que é O(N) e inviável para milhões de vetores de alta dimensão. A solução é **ANN (Approximate Nearest Neighbor)**, que estrutura os dados em grafos, árvores ou clusters para buscar apenas em uma vizinhança promissora. Os algoritmos comuns são **HNSW** (Hierarchical Navigable Small World), **IVF** (Inverted File) e **PQ** (Product Quantization). O trade-off fundamental: ANN troca uma pequena perda de precisão (recall) por um ganho massivo de velocidade.

4.11 Busca híbrida

A busca semântica é poderosa mas pode falhar em buscas exatas, como SKUs e nomes próprios. A solução moderna é a **busca híbrida**, que combina:

- **Busca vetorial (densa)**: captura significado, contexto e sinônimos.
- **Busca lexical (esparsa, BM25)**: garante correspondência exata de palavras-chave.
- **Filtros de metadados**: restringem resultados por preço, categoria e outros atributos.

O **Reciprocal Rank Fusion (RRF)** é o algoritmo padrão para combinar e re-ranquear os resultados vindos dos diferentes métodos em uma lista final única.

4.12 Busca cross-modal

Com embeddings alinhados num espaço compartilhado, a query e o documento recuperado não precisam ser da mesma modalidade. Os quatro padrões citados são: **texto para imagem** (buscar fotos usando descrições naturais, como em Google Photos e Pinterest); **imagem para texto** (encontrar artigos ou receitas a partir de uma foto); **áudio para imagem** (usar o som de um motor para encontrar fotos do modelo do carro); e **imagem para imagem** (busca visual por produtos similares). O ponto essencial é que a matemática da busca, a similaridade cosseno, permanece exatamente a mesma.

4.13 Aplicações

Busca visual de produtos. No varejo online, o usuário faz upload de uma foto e o sistema retorna produtos visualmente idênticos ou similares; queries abstratas como "vestido floral de verão para casamento" encontram correspondências visuais precisas; e recomendações visuais sugerem itens complementares com base no estilo do produto atual. O impacto de negócio é a redução do atrito na jornada de compra e o aumento da taxa de conversão.

RAG Multimodal. O RAG tradicional enriquece LLMs com documentos textuais. O RAG multimodal expande o paradigma para incluir imagens, gráficos e áudio no contexto. Opera em três etapas: **indexação** de documentos, imagens e áudios como embeddings em banco vetorial; **recuperação** dos fragmentos multimodais mais relevantes a partir de uma query textual ou visual; e **geração** por um LLM multimodal que recebe query e contexto. Permite responder perguntas sobre manuais técnicos com diagramas, relatórios financeiros com gráficos ou vídeos de treinamento.

Fine-tuning para domínios específicos. Modelos como CLIP são treinados em dados gerais da internet e precisam de adaptação para domínios especializados. Em **medicina**, alinham-se laudos radiológicos com imagens de raio-X ou ressonância magnética, como no BioCLIP. Em **e-commerce**, adapta-se a catálogos com jargões de moda ou peças industriais. Métodos eficientes como **LoRA (Low-Rank Adaptation)** ou **Adapters** treinam apenas uma pequena fração dos pesos, obtendo ganhos de Recall@K com custo computacional baixo.

4.14 Desafios

Modality Gap. Mesmo com aprendizado contrastivo, embeddings de modalidades diferentes tendem a se agrupar em regiões separadas do espaço vetorial. A causa está nas arquiteturas distintas de encoders e nas propriedades estatísticas diferentes dos dados brutos. A consequência é que uma query de texto pode estar mais próxima de outros textos do que da imagem correspondente. As mitigações são técnicas adicionais de alinhamento, temperatura ajustável na função de perda e projeções lineares pós-treinamento. O insight relevante: o alinhamento perfeito é difícil, mas a **similaridade relativa** (o ranking) ainda é preservada.

Escalabilidade e custo. Vetores densos de alta dimensão ocupam muito espaço: um embedding de 1024 dimensões em float32 consome 4 KB, de modo que 1 bilhão de vetores equivale a 4 TB apenas de dados brutos, sem contar a sobrecarga do índice. Além disso, algoritmos como HNSW exigem que o índice resida inteiramente em RAM para garantir latência de milissegundos, o que encarece a infraestrutura. Gerar embeddings para grandes volumes requer GPUs potentes, e a manutenção de índices (inserir, atualizar, deletar) é custosa em estruturas otimizadas.

Viés e interpretabilidade. Modelos treinados em dados massivos da internet absorvem e amplificam **estereótipos históricos, raciais e de gênero**. Além disso, um embedding é um vetor denso de números reais cujas dimensões individuais **não possuem significado semântico** diretamente interpretável por humanos. Quando o sistema retorna resultado incorreto ou ofensivo, rastrear a causa raiz é tarefa extremamente complexa e não determinística. O material formula isso como o **paradoxo da representação**: quanto mais expressivo e multimodal o modelo, mais difícil auditar seu comportamento interno.

4.15 Perspectivas

As tendências de arquitetura apontadas são: **scaling laws**, com ganhos contínuos de performance ao aumentar parâmetros e dados; **resolução dinâmica**, processando imagens em resoluções nativas variadas em vez de forçar redimensionamento para 224 por 224; e **contexto longo**, com embeddings capazes de capturar vídeos inteiros ou documentos extensos em representações coesas.

Na evolução das representações: transição de **early-fusion versus late-fusion**, ou seja, de modelos dual-encoder onde as modalidades interagem apenas no fim para arquiteturas unificadas que processam texto e imagem simultaneamente; **representações multi-vetor**, com múltiplos vetores por documento (à la ColBERT multimodal) para preservar nuances finas; e o **paradigma da unificação**, com um único modelo fundacional capaz de processar, alinhar e gerar qualquer combinação de modalidades.

5 OCR YOLO

Material de slides não disponível para esta aula.

Esta aula trata da extração estruturada de informação a partir de imagens, com dois eixos indicados pelo título e pelos materiais de apoio: **OCR** (Optical Character Recognition), com a biblioteca **pytesseract**, e **YOLO** (You Only Look Once), família de modelos de detecção de objetos em imagens e vídeo. Os arquivos de apoio incluem um notebook de treinamento, um notebook de

YOLO, um notebook de pytesseract, uma revisão de CNN e Vision Transformers, além de exemplos concretos de documento (uma conta de luz) e de vídeos para detecção.

O conteúdo se articula com o restante da disciplina em dois pontos. Primeiro, a revisão de CNN e ViT retoma exatamente a arquitetura descrita na aula de embeddings multimodais: camadas iniciais capturando bordas e texturas, camadas finais capturando conceitos semânticos. Segundo, o OCR reaparece de forma crítica na aula de RAG multimodal, onde o material discute suas limitações — erros de OCR quebram tabelas — e apresenta o **ColPali** como alternativa de retrieval visual que dispensa OCR, além de comparar a qualidade de OCR de diferentes VLMs, com destaque para o Phi-3-Vision e o Qwen-VL.

6 RAG Multimodais

Esta aula é o ponto de integração da disciplina. Ela parte de um problema concreto — o que acontece quando se joga um PDF técnico num RAG tradicional — e constrói, camada a camada, um sistema completo que recupera e raciocina sobre texto, tabela, imagem e diagrama juntos, preservando a semântica de cada tipo de conteúdo.

6.1 O problema real

Considere um datasheet contendo uma tabela de especificações, um gráfico de curva e um diagrama de pinagem. O resultado no RAG tradicional é catastrófico: o parser converte tudo em texto corrido, as tabelas perdem a estrutura de linha e coluna, as imagens desaparecem completamente e os diagramas técnicos simplesmente não existem para o modelo.

6.2 As três gerações de RAG

1ª geração — o pipeline ingênuo. O fluxo é linear: documento, chunk, embed, banco vetorial, recuperação Top-K, inserção no prompt, LLM, resposta. Suas três falhas críticas são: **chunk mal cortado** (tabelas divididas ao meio, imagens ignoradas, contexto estrutural perdido); **recuperação irrelevante** (o Top-K por similaridade semântica não captura relações estruturais nem contexto visual); e **múltiplos saltos** (perguntas que exigem raciocínio multi-hop falham porque o sistema não itera a busca).

2ª geração — RAG avançado. Introduz otimizações em três fases. No **pré-retrieval**: chunking semântico em vez de por tamanho fixo, extração de metadados (autor, data, categoria) e query rewriting com expansão. No **retrieval**: busca híbrida combinando BM25 e embeddings densos, reranking com cross-encoders e fusão de múltiplos rankings. No **pós-retrieval**: compressão de contexto baseada em LLM, reordenação de chunks por relevância e deduplicação. A limitação persiste: ainda é um **pipeline fixo**, sem orquestração dinâmica.

3ª geração — RAG agêntico. O LLM passa de gerador final a orquestrador. As quatro capacidades novas são:

- **Orquestração**: o LLM decide qual ferramenta chamar — busca de texto, de tabela, de imagem, SQL. "Qual é a tensão máxima?" aciona a busca de tabela.
- **Decisão dinâmica**: não há pipeline fixo; o agente escolhe a estratégia a cada passo.

- **Iteração**: se a primeira busca não for suficiente, o agente reescreve a query e busca novamente.
- **Auto-crítica**: o agente avalia se o contexto é suficiente antes de gerar a resposta.

Os padrões arquiteturais mencionados são **ReAct**, **roteamento** e **multi-agente**.

6.3 Estratégias por tipo de conteúdo

Cada tipo de conteúdo quebra o RAG tradicional de forma diferente e exige uma estratégia própria:

- **Texto**: funciona bem no RAG tradicional. Mantém-se chunking semântico normal e embeddings de texto.
- **Tabela**: vira linha corrida e perde a relação linha/coluna. A estratégia é preservar como Markdown ou HTML, ou sumarizar a tabela com LLM e indexar o resumo.
- **Imagem**: completamente ignorada pelo parser de texto. A estratégia é gerar descrição detalhada com um VLM e indexá-la, ou usar embedding multimodal tipo CLIP.
- **Diagrama**: ignorado, mas a semântica estrutural é crítica. O VLM descreve estrutura e componentes, guardando referência para recuperar a imagem original.

6.4 As duas estratégias arquiteturais

Estratégia 1 — multimodal por sumarização (texto-centrada). Cada tabela ou imagem vira um resumo textual gerado por VLM; os resumos são indexados num único vector store; na recuperação, o agente busca a imagem ou tabela original por referência. As vantagens são: **barato** (sem embeddings multimodais caros), **robusto** (funciona com qualquer LLM de texto), **compatível** (integra facilmente com LangChain e LlamaIndex) e **preciso** (evita fragilidade em diagramas técnicos densos). Use-a em documentos técnicos com tabelas e diagramas complexos, onde a semântica estrutural importa mais que a similaridade visual.

Estratégia 2 — embeddings multimodais (CLIP-like). Imagem e texto são mapeados no mesmo espaço vetorial, com recuperação direta e bidirecional. As vantagens são a elegância conceitual e a recuperação cruzada natural. As desvantagens são a **fragilidade em diagramas densos** (modelos CLIP perdem detalhes finos e texto embutido) e o **custo elevado**. Use-a em documentos com imagens naturais ou menos estruturados.

6.5 Unstructured: o parser multimodal

A ferramenta central para o parsing é a função `partition_pdf` da biblioteca **Unstructured**, com estratégia `hi_res`:

```
from unstructured.partition.pdf import partition_pdf

elements = partition_pdf(
    "datasheet.pdf",
    strategy="hi_res",
    extract_images_in_pdf=True,
    infer_table_structure=True,
)
```

Suas capacidades são: **layout detection**, separando o documento em blocos lógicos como `Title`, `NarrativeText`, `Table` e `Image`; **tabelas preservadas**, extraídas com `text_as_html`, mantendo a estrutura de linhas e colunas intacta; **extração de mídia**, com imagens e figuras salvas diretamente em disco; e **metadados** estruturais e de página para cada elemento. O resultado, em vez de "sopa de texto", são textos estruturados, tabelas em HTML e imagens em arquivos separados.

6.6 Frameworks de orquestração

LangChain. Seus componentes-chave são o **MultiVectorRetriever** (indexa o resumo no vector store e devolve o documento original do docstore), o **Agent** (o orquestrador que decide qual ferramenta chamar), as **Tools** (`buscar_texto`, `buscar_tabela`, `buscar_imagem`) e as **Chains** (fluxos que conectam prompts, LLMs e parsers de saída). Use quando o foco estiver no fluxo agêntico e nas decisões dinâmicas durante a recuperação.

LlamaIndex. Seus componentes são os **Nodes** (unidades estruturadas que preservam relações e metadados), o **MultiModalVectorStoreIndex** (indexação multimodal nativa para texto e imagens), o **QueryEngine** e os **Document Loaders**. Use quando o foco estiver na camada de dados e na ingestão complexa de múltiplas fontes. A grande vantagem é oferecer abstrações mais altas que o LangChain para lidar com dados, resultando em menos código repetitivo.

Haystack. Framework alternativo focado em pipelines declarativos e componentes reutilizáveis, com Document Stores, Retrievers (BM25, denso, híbrido), Readers e Pipelines. Comparado ao LangChain, tem foco em RAG puro em vez de LLM em geral, pipelines declarativos em vez de imperativos, curva de aprendizado média e comunidade menor, mas integração nativa com Elasticsearch e busca híbrida excelente.

6.7 O ecossistema em três pilares

Embeddings: CLIP (OpenAI, multimodal clássico), SigLIP (Google, alternativa open-source) e ColPali (retrieval visual sem OCR).

VLMs: GPT-4o (proprietário, estado da arte), Claude 3.5 (excelente em análise de imagens) e LLaVA (open-source, rápido e eficiente).

Vector DBs: Qdrant (rápido, escalável, filtros avançados), Milvus (open-source, alta performance em escala) e Weaviate (GraphQL, buscas híbridas nativas).

6.7.1 CLIP versus SigLIP

O **CLIP** é proprietário e acessado via API, treinado em 400 milhões de pares texto-imagem, excelente em classificação zero-shot, com latência em torno de 200 ms por imagem e custo aproximado de US$ 0,02 por imagem. É ideal para produção com orçamento e exigência de máxima qualidade.

O **SigLIP** é open-source com deployment local, treinado com aprendizado contrastivo, disponível em vários tamanhos, com latência em torno de 50 ms em GPU local. Não tem custo por chamada, oferece controle total, latência muito menor e privacidade garantida, ao custo de qualidade ligeiramente inferior e necessidade de GPU. É ideal para escalabilidade, privacidade e alto volume.

6.7.2 ColPali

O **ColPali (Contextual Language and Page Layout)** é um modelo de retrieval visual que entende imagens de documentos **sem precisar de OCR**, trabalhando diretamente com a imagem como um todo. Recebe a imagem de página inteira em vez de chunks de texto, gera um embedding que captura layout, estrutura visual e conteúdo, e a busca por similaridade retorna páginas inteiras relevantes, preservando o contexto visual.

Comparado ao CLIP, o ColPali dispensa OCR (que o CLIP recomenda), é excelente em tabelas (contra "bom" do CLIP), entende layout (que o CLIP ignora), é mais rápido e tem custo baixo por rodar localmente. Use ColPali em PDFs técnicos, tabelas complexas e documentos onde diagramas são importantes.

6.7.3 Bancos vetoriais em produção

O **Qdrant**, escrito em Rust, suporta vetores de alta dimensão, filtros de payload avançados com combinações lógicas (AND, OR, NOT), range queries e queries geoespaciais, além de clustering, replicação, backup e recovery. Um exemplo mínimo de uso:

```
from qdrant_client import QdrantClient
from qdrant_client.models import Distance, VectorParams

client = QdrantClient("localhost", port=6333)

client.create_collection(
    collection_name="documents",
    vectors_config=VectorParams(size=1536, distance=Distance.COSINE),
)
```

O **Milvus**, escrito em C++, suporta múltiplos índices (HNSW, IVF, FLAT), escala horizontalmente por sharding e integra com Kubernetes. Use em escala massiva com deployment em K8s e orçamento zero, ao custo de curva de aprendizado mais alta.

O **Weaviate** oferece API GraphQL nativa, busca híbrida vetorial mais BM25 pronta e RAG integrado (generative search). É menos escalável que o Milvus em volume e requer mais memória.

6.8 Parsers avançados

LlamaParse é um serviço de parsing que usa LLMs para entender estrutura complexa. Envia o PDF para a API, o LLM analisa tabelas, listas e hierarquia, e retorna JSON estruturado com metadados, preservando relacionamentos. Comparado ao Unstructured, é uma API SaaS (contra local ou API), tem estrutura excelente e tabelas perfeitas (contra "bom"), latência de 2 a 5 s (contra 500 ms), custo de US$ 0,003 por página (contra gratuito) e privacidade menor por depender de API.

Marker, da Datalab, é baseado em modelos de visão e retorna Markdown estruturado. É completamente gratuito, rápido em GPU local e sem dependências externas, mas tem qualidade variável em tabelas. Use com orçamento zero e quando Markdown for importante.

MinerU, da Opendatalab, foca em documentos técnicos e retorna JSON estruturado, com OCR melhor que o Marker e excelência em tabelas. É mais lento e exige mais recursos. Use quando tabelas forem críticas e JSON for importante.

6.9 Small Language Models multimodais

SLMs são modelos de linguagem com menos de 8 bilhões de parâmetros, otimizados para rodar localmente com baixo consumo de recursos, e multimodais quando suportam texto mais imagem. Rodam em GPU local com até 4 GB de VRAM, têm latência abaixo de 100 ms, garantem privacidade total, têm custo zero e permitem deployment em dispositivos de borda. Os casos de uso são RAG local em documentos sensíveis, aplicações offline-first, processamento em tempo real e dispositivos móveis e IoT.

O contraste com LLMs grandes é sistemático: SLMs têm menos de 8 B de parâmetros contra mais de 70 B; usam de 4 a 8 GB de memória contra mais de 40 GB; latência de 50 a 100 ms contra 200 a 500 ms; são gratuitos contra custo por token; têm raciocínio "bom" contra "excelente"; privacidade total contra parcial; e deployment local contra nuvem. O contexto também difere fortemente: de 2 a 8 K tokens nos SLMs contra 128 K ou mais nos LLMs. SLMs são também mais propensos a alucinações por terem visto menos dados.

Os principais SLMs multimodais citados são o **Phi-3-Vision** (Microsoft, 4,2 B parâmetros, 4 GB de VRAM, cerca de 50 ms, excelente OCR, melhor custo-benefício), o **Qwen2-VL-2B** (Alibaba, 2 B parâmetros, 2 a 4 GB, cerca de 30 ms) e o **SmolVLM** (Hugging Face, de 256 M a 2 B parâmetros, 1 a 2 GB, cerca de 20 ms, ultracompacto, ideal para edge e mobile).

Entre os VLMs open-source de porte maior, o material cita o **LLaVA** 1.5/1.6 (7 B a 34 B, cerca de 200 ms em GPU, ideal para MVP), o **Pixtral 12B** da Mistral (cerca de 150 ms, qualidade superior) e o **Qwen-VL Max** da Alibaba (32 B, cerca de 300 ms, qualidade máxima e OCR excelente). Entre os proprietários, o GPT-4o oferece 128 K de contexto e cerca de 500 ms de latência, e o Claude 3.5 oferece 200 K de contexto e cerca de 600 ms, ambos com capacidades de OCR em imagens, análise de tabelas, descrição de diagramas e raciocínio visual complexo.

6.10 RAG local e edge computing

A arquitetura totalmente local encadeia: PDF local, parser local (Marker ou MinerU, sem API), embedding local (SigLIP ou Nomic Embed em GPU), banco vetorial local (Qdrant ou SQLite) e SLM local (Phi-3-Vision ou Qwen2-VL). Os benefícios são **privacidade** (dados nunca saem do computador, conformidade com GDPR e LGPD), **performance** (latência abaixo de 100 ms sem rede, funcionamento offline, sem limites de API) e **custo** (modelos open-source gratuitos, investimento único em hardware).

As ferramentas de inferência para rodar SLMs localmente são o **Ollama** (servidor HTTP local, setup ultrassimples de um comando, API compatível com OpenAI, melhor para iniciantes), o **llama.cpp** (runtime C++ puro, extremamente rápido, com quantização nativa de 4 e 8 bits) e o **MLX** da Apple (framework otimizado para Apple Silicon, com API Python completa, melhor opção em Macs da série M).

6.11 Arquitetura recomendada

As ferramentas não competem, se combinam. O fluxo integrado é:

```
PDF -> Unstructured (parsing)
    -> LangChain / LlamaIndex (indexação)
    -> MultiVectorRetriever
    -> Agent (orquestração)
```

A divisão de responsabilidades é clara: **Unstructured** extrai a estrutura do PDF, separando textos, tabelas e imagens; **LangChain ou LlamaIndex** (escolhe-se um dos dois) gerencia armazenamento de vetores e lógica do agente; o **VLM** descreve o conteúdo visual.

6.12 Laboratório passo a passo

Etapa 0 — setup. A instalação de dependências e a estrutura do projeto:

```
pip install unstructured langchain llamaindex openai pillow pdf2image
```

```
rag-multimodal/
-- data/
|   -- pdfs/
|   -- extracted/
-- src/
|   -- parser.py
|   -- indexer.py
|   -- retriever.py
|   -- agent.py
-- main.py
```

Etapa 1 — parsing multimodal. O `partition_pdf` produz o resultado em três baldes: textos (`NarrativeText`, `Title`), tabelas com HTML preservado e imagens extraídas para disco. O momento decisivo, como diz o material, é que "a tabela sai estruturada e as imagens vão para o disco; não é mais uma sopa de texto".

Etapa 2 — sumarização multimodal. Para texto e tabelas, usa-se o LLM para sumarizar; para imagens e diagramas, o VLM gera descrição:

```
resumo = llm.summarize(tabela_html)
descricao = vlm.describe(imagem)
```

A estrutura de dados resultante contém `id`, `tipo`, `resumo`, `conteudo_original` (referência) e `embedding`. O princípio-chave é: **indexa o resumo, guarda o original**. Esse é o coração da arquitetura multi-vetor, que combina busca semântica rápida com recuperação do contexto visual completo.

Etapa 3 — indexação multi-vetor.

```
from langchain.retrievers.multi_vector import MultiVectorRetriever
```

```
retriever = MultiVectorRetriever(
    vectorstore=vector_store,
    docstore=docstore,
    id_key="id",
)
```

O fluxo tem três momentos: o embedding vai no resumo (rápido, denso, semanticamente rico); o docstore guarda o original (tabela HTML completa ou imagem); e a recuperação traz de volta a tabela ou imagem **completa**, não o resumo truncado. O ganho é melhor relevância na busca somada a contexto estrutural e visual completo para o LLM.

Etapa 4a — chain simples. O fluxo é pergunta, retriever, contexto (texto mais tabela mais descrição de imagem), LLM, resposta com citação de fonte. No exemplo, a pergunta "qual é a tensão máxima de operação?" faz o retriever trazer a tabela de especificações e a descrição da imagem, e o LLM responde citando a linha da tabela.

Etapa 4b — o pulo agêntico. Aqui o agente decide:

```
tools = [
    Tool(name="buscar_texto", func=buscar_texto),
    Tool(name="buscar_tabela", func=buscar_tabela),
    Tool(name="buscar_imagem", func=buscar_imagem),
]

agent = initialize_agent(tools, llm, agent="react-docstore")
```

O teste crítico é uma pergunta como "quantos pinos tem o conector no diagrama X e qual a função do pino 3?". O agente percebe que precisa de informação visual e chama `buscar_imagem`; o VLM lê o diagrama recuperado e extrai a informação estrutural. O agente responde com precisão usando uma fonte que seria **invisível** para um RAG tradicional.

6.13 A entrega: do notebook ao serviço

A entrega da aula tem quatro passos. O **passo 1** (ingestão e parsing) exige carregar o PDF e validar integridade, extrair texto, extrair tabelas preservando HTML sem truncar, extrair imagens salvando em disco com descritores únicos, e validar que nenhum elemento foi perdido. O **passo 2** (indexação e sumarização) tem três etapas: sumarizar (LLM resume parágrafos longos e descreve tabelas, VLM descreve imagens), gerar embeddings mantendo ID único para rastreamento, e indexar separando vector store (resumos mais vetores) de docstore (conteúdo original). O **passo 3** (recuperação e orquestração) implementa o ciclo completo, com o agente ReAct decidindo a ferramenta e iterando automaticamente se o contexto for insuficiente. O **passo 4** (interface) expõe quatro endpoints em FastAPI:

```
POST /ingest        -> recebe PDF, faz parsing, indexa
POST /query         -> recebe pergunta, retorna resposta + contexto + fonte
GET  /source/{id}   -> retorna elemento original (tabela HTML ou imagem)
GET  /health        -> status do sistema
```

Os critérios de avaliação são explícitos: 20% para o parsing separar os quatro tipos de conteúdo, 25% para a recuperação trazer o elemento certo, 25% para responder ao menos uma pergunta sobre

imagem ou diagrama, 15% para citar fonte e devolver o original, e 15% para estrutura limpa, ou seja, não um notebook monolítico.

6.14 Casos de uso reais

O material argumenta que ignorar imagens, gráficos e tabelas significa perder até 50% da informação vital em setores técnicos. Os três domínios apresentados são: **suporte técnico e manutenção** na indústria, com manuais de equipamentos pesados, diagramas de circuitos e guias de reparo; **análise financeira e legal** em bancos e seguradoras, com relatórios anuais repletos de gráficos, balanços em tabelas complexas e contratos escaneados com assinaturas e carimbos; e **conformidade e auditoria**, com plantas baixas arquitetônicas, laudos técnicos com evidências fotográficas e manuais de segurança do trabalho.

6.15 Case Pré-Sal: do notebook ao FastAPI

O estudo de caso final é a transformação de um protótipo em produto. O **problema do notebook** é que a ingestão é pesada (roda a cada execução, reprocessa o PDF inteiro, chama APIs a cada boot, sem cache de embeddings), o código é monolítico (tudo em um `.ipynb`, difícil de testar por partes, sem separação de responsabilidades, impossível reutilizar módulos) e não há API (não é acessível via HTTP, sem integração com outros sistemas).

A **solução em FastAPI** resolve os três pontos: a ingestão roda uma única vez via `build_index.py`, com embeddings salvos em `data/index_cache.npz` e a API carregando o cache no startup em cerca de 100 ms; o código é modular, com `ingest/`, `index/` e `agent/` isolados e testáveis independentemente; e a API REST expõe Swagger em `/docs`, com endpoint de pergunta com roteamento agêntico, pronta para deployment em Docker ou nuvem.

A **ingestão multimodal** do caso combina duas frentes. Na extração estruturada, o PDF é renderizado em imagens (uma por página), OCR mais regex capturam campos estruturados, e 101 campos são organizados em JSON com metadados, localização e operador. Na visão multimodal, o GPT-4o Vision analisa mapas embutidos no PDF, descreve localização geográfica, identifica limites e zonas, e extrai informações visuais. A vantagem decisiva é que as descrições ficam **em cache**, de modo que perguntas visuais são respondidas sem chamar o modelo de visão novamente, com custo reduzido a uma única passada por ingestão.

O **cache de embeddings** em arquivo `.npz` armazena os embeddings dos campos, os embeddings das descrições de mapas e os metadados (ID, tipo, fonte). No boot, a API carrega o `.npz` em memória em cerca de 100 ms, sem reprocessar o PDF e sem chamadas ao provedor de embeddings.

O **agente roteador** decide entre três caminhos, conforme o tipo de pergunta:

- "Qual é a profundidade do campo X?" aciona `buscar_campos` (índice textual, busca por cosseno).
- "Quais campos estão no litoral norte?" aciona `consultar_mapa` (descrição visual).
- "Quantos campos a operadora opera?" aciona agregação sobre múltiplas fontes.

O ciclo é: o agente recebe a pergunta, escolhe a ferramenta (busca textual ou visual), executa e retorna o resultado, e a resposta inclui a fonte usada. O resultado é roteamento transparente e respostas rastreáveis.

A execução end-to-end cabe em quatro comandos:

```
pip install -r requirements.txt
export OPENAI_API_KEY="sk-..."
python build_index.py      # gera índice e embeddings
uvicorn app:app --reload   # Swagger em /docs
```

6.16 Documento técnico de referência

Um relatório técnico de apoio consolida o argumento com uma avaliação de recuperação. Compararam-se três abordagens em um conjunto de perguntas cuja resposta depende de conteúdo visual embutido. O RAG somente-texto ingênuo alcança recall baixo, pois as figuras são invisíveis ao índice. A adição de reranking melhora marginalmente. A abordagem multimodal, que indexa descrições geradas por VLM, eleva o recall de forma expressiva, atingindo **86% de recall@5** em perguntas visuais. A conclusão é que indexar descrições geradas por VLM e permitir que um agente roteie entre fontes textuais e visuais produz respostas mais completas e fiéis ao documento original.

6.17 Conclusões da aula

Três lições se destacam. Primeira, o RAG deixou de ser apenas busca de texto para se tornar um sistema multimodal e agêntico. Segunda, tabelas e imagens contêm informações críticas que não podem ser ignoradas no parsing. Terceira, indexar resumos e recuperar originais é a chave para precisão e contexto completo. Os próximos passos indicados são implementar o laboratório com PDFs próprios, escalar para múltiplos PDFs adicionando metadados robustos para filtragem, e monitorar custos e latência, já que o uso de VLMs para sumarizar imagens aumenta o custo.

7 Metricas para LLMs

Material de slides não disponível para esta aula.

Esta aula trata da avaliação de sistemas baseados em LLMs. O material de apoio da disciplina registra apenas a gravação da aula, sem slides no contexto. Ainda assim, o restante do material fornece o vocabulário de avaliação que a disciplina efetivamente utiliza, e vale sistematizá-lo aqui, uma vez que aparece de forma dispersa nas outras aulas.

7.1 Métricas de recuperação

Em sistemas de RAG, a métrica de referência utilizada no material é o **recall@K**, isto é, a fração de perguntas para as quais o elemento correto aparece entre os K primeiros resultados recuperados. É essa métrica que sustenta a comparação entre RAG somente-texto, RAG com reranking e RAG multimodal, na qual a abordagem multimodal atinge 86% de recall@5 em perguntas visuais.

Em fine-tuning de modelos multimodais para domínios específicos, o material menciona ganhos de **Recall@K** como o indicador de sucesso.

7.2 Métricas de verificação e classificação

Na aula de reconhecimento de locutor, a métrica central é o **Equal Error Rate (EER)**, o ponto de operação em que a taxa de falsos positivos iguala a taxa de falsos negativos. É uma métrica de limiar único, adequada a tarefas biométricas em que se quer um número comparável entre modelos: modelos modernos alcançam EER abaixo de 2%, e o ECAPA-TDNN atinge 1,71% em áudio curto.

7.3 Métricas de qualidade de reconstrução

O relatório técnico anexo à aula de RAG multimodal apresenta o conjunto de métricas usado em tarefas de reconstrução visual, útil como referência conceitual sobre famílias de métricas:

- **PSNR (Peak Signal-to-Noise Ratio)**: razão entre a potência máxima possível de um sinal e a potência do ruído introduzido; quanto maior, mais próxima a reconstrução está do original. É calculado a partir do **MSE (Mean Squared Error)** entre imagem original e reconstruída.
- **SSIM (Structural Similarity Index Measure)**: avalia aspectos estruturais, luminância e contraste, alinhados à percepção visual humana, produzindo valores entre 0 e 1, onde 1 indica similaridade perfeita.
- **LPIPS (Learned Perceptual Image Patch Similarity)**: métrica perceptual que compara ativações de múltiplas camadas de uma rede convolucional profunda pré-treinada. Diferentemente de PSNR e SSIM, que operam pixel a pixel, o LPIPS mede diferenças no espaço de características, aproximando-se mais da percepção humana. Valores menores indicam maior similaridade.

A lição metodológica desse conjunto é importante e generalizável para LLMs: métricas que medem correspondência exata (pixel a pixel, ou token a token) podem divergir de métricas que medem qualidade percebida. No material, a interpolação bicúbica atinge PSNR competitivo (24,72 dB) mas SSIM e LPIPS bem inferiores (0,79 e 0,32), mostrando que **uma única métrica não é suficiente** para decidir sobre qualidade.

7.4 Métricas de custo e latência

O material trata custo e latência como métricas de primeira classe, não como detalhes de implementação. Aparecem sistematicamente: latência por imagem em modelos de embedding (200 ms no CLIP via API contra 50 ms no SigLIP local); latência média de VLMs (cerca de 500 ms no GPT-4o, 600 ms no Claude 3.5, 20 a 300 ms nos SLMs); tempo de inferência em verificação de locutor (69,43 ms no ECAPA-TDNN); custo por chamada (US$ 0,02 por imagem no CLIP, US$ 0,005 por 1K tokens de entrada no GPT-4o, US$ 0,003 no Claude 3.5, US$ 0,003 por página no LlamaParse); e janela de contexto como restrição prática (2 a 8 K tokens em SLMs contra 128 K ou 200 K em LLMs proprietários).

O relatório técnico formaliza um ponto sutil sobre latência: o tempo de inferência bruto pode enganar. Em sistemas que precisam acumular múltiplos quadros antes de processar, a latência total é a soma do tempo de buffer com o tempo de inferência, e o tempo de buffer pode dominar. É o

mesmo tipo de raciocínio que se aplica a pipelines de RAG com múltiplas chamadas em sequência: a métrica relevante é a latência fim-a-fim percebida pelo usuário, não a de um componente isolado.

8 Explainable AI

Material de slides não disponível para esta aula.

Esta aula trata da interpretabilidade de modelos de IA. Os materiais de apoio indicam o escopo prático: notebooks de **Captum** (biblioteca de interpretabilidade do PyTorch), de interpretabilidade em **TorchVision**, de explicabilidade do **CLIP**, um material sobre **backpropagation**, e um estudo de caso sobre detecção de **deepfakes** em imagens.

8.1 Por que a explicabilidade importa nesta disciplina

O material da aula de embeddings multimodais formula o problema com precisão. Um embedding é um vetor denso de números reais cujas **dimensões individuais não possuem significado semântico** diretamente interpretável por humanos, ao contrário de features construídas manualmente. Quando o sistema retorna um resultado incorreto, irrelevante ou ofensivo, rastrear a causa raiz num espaço vetorial de bilhões de parâmetros é uma tarefa extremamente complexa e não determinística.

A isso se soma o problema do **viés herdado**: modelos treinados em dados massivos da internet absorvem e amplificam estereótipos históricos, raciais e de gênero, e o espaço vetorial pode, inadvertidamente, associar certas profissões ou sentimentos a grupos demográficos específicos.

O material sintetiza a tensão como **paradoxo da representação**: quanto mais expressivo e multimodal o modelo se torna, mais difícil é auditar seu comportamento interno e garantir que suas representações sejam justas, seguras e alinhadas. Essa é precisamente a motivação da aula de Explainable AI.

8.2 Conexões com as demais aulas

A explicabilidade aparece de forma operacional em vários pontos do curso. No RAG multimodal, a **citação de fonte** e o endpoint `GET /source/{id}` que devolve o elemento original são mecanismos concretos de explicabilidade: o usuário pode verificar de onde veio a resposta. No agente roteador do Case Pré-Sal, o material descreve o objetivo como "roteamento transparente, respostas rastreáveis" — a resposta inclui qual ferramenta foi usada. Em reconhecimento de locutor, a inspeção de embeddings e de similaridade cosseno oferece uma forma direta de justificar decisões de verificação.

A explicabilidade do CLIP, presente nos materiais de apoio, aborda o problema de identificar quais regiões da imagem contribuíram para o alinhamento com um texto dado — um caso particularmente relevante dado que o CLIP é a base de boa parte da busca multimodal apresentada na disciplina.

9 Deploy

Material de slides não disponível para esta aula.

O tópico de deploy encerra a ementa, e o material da disciplina permite reconstruir seu conteúdo essencial a partir do Case Pré-Sal e das discussões de arquitetura da aula de RAG multimodal. O tema pode ser formulado como uma pergunta: o que separa um notebook que funciona de um sistema que serve requisições?

9.1 Do protótipo ao serviço

O diagnóstico do material é claro. Um protótipo em notebook falha em três eixos:

- **Ingestão pesada**: roda a cada execução, reprocessa o documento inteiro, chama APIs externas a cada boot, sem cache.
- **Código monolítico**: tudo num arquivo, difícil de testar por partes, sem separação de responsabilidades, impossível reutilizar módulos.
- **Sem API**: não acessível via HTTP, sem integração com outros sistemas.

As três correções correspondentes são o **cache persistente**, a **modularização** e a **API REST**.

9.2 Cache persistente

O padrão apresentado separa a geração do índice do serviço que o consome. Um script `build_index.py` roda uma única vez, processa o documento completo, gera os embeddings e salva em `data/index_cache.npz`. A API carrega esse cache na inicialização em cerca de 100 ms, sem reprocessar o documento nem chamar o provedor de embeddings. O custo de ingestão passa a ser pago uma única vez.

O mesmo princípio se aplica às descrições visuais: as descrições geradas por VLM ficam em cache, de modo que perguntas visuais são respondidas sem chamar o modelo de visão novamente. O material registra o efeito: descrição em cache significa respostas rápidas e custo reduzido a uma passada por ingestão.

9.3 Modularização

A estrutura de diretórios recomendada isola responsabilidades:

```
rag_presal/
-- ingest/          # parsing e descrição de mapas
-- index/           # embeddings e cache .npz
-- agent/           # roteamento inteligente (ReAct)
-- build_index.py
-- app.py
-- README.md
```

Cada módulo é independente e testável, o que torna fácil corrigir, avaliar e reutilizar. Não é coincidência que 15% da nota da entrega da aula de RAG multimodal dependa de "estrutura limpa, não notebook monolítico": a modularidade é tratada como requisito de entrega, não como refinamento opcional.

9.4 API REST com FastAPI

A camada de serviço expõe o sistema via HTTP. O padrão apresentado usa **FastAPI** com documentação Swagger automática em `/docs`, e a execução completa cabe em quatro comandos: instalar dependências, configurar a chave de API, gerar o índice uma vez e subir o servidor com `uvicorn`. O resultado é uma API pronta para deployment em Docker ou nuvem.

O conjunto mínimo de endpoints, apresentado na entrega da aula de RAG multimodal, cobre ingestão, consulta, recuperação do elemento original e verificação de saúde do sistema. Esse último, `GET /health`, é um detalhe pequeno mas revelador: sistemas em produção precisam ser monitoráveis.

9.5 Estratégias de deployment: nuvem versus local

O material apresenta duas filosofias de deployment como escolha arquitetural consciente, não como preferência.

A abordagem via **APIs comerciais e modelos proprietários** oferece validação rápida e máxima qualidade sem complexidade de infraestrutura. É a recomendação explícita para prototipagem tanto em reconhecimento de locutor quanto em RAG. Seus custos são a dependência de API externa, latência de rede e custo por chamada.

A abordagem **local e edge** — parser local (Marker, MinerU), embedding local (SigLIP, Nomic Embed), banco vetorial local (Qdrant, SQLite) e SLM local (Phi-3-Vision, Qwen2-VL) — entrega privacidade total com conformidade a GDPR e LGPD, latência abaixo de 100 ms sem rede, funcionamento offline e custo zero de API com investimento único em hardware. É a escolha indicada para dados sensíveis, compliance e aplicações offline. Suas ferramentas de execução são Ollama, llama.cpp e MLX.

Os desafios do caminho local, também registrados no material, são a qualidade inferior de respostas, a dificuldade com instruções complexas, o custo inicial de GPU, a configuração mais complexa, a dificuldade de escalar para múltiplos usuários simultâneos e o conhecimento desatualizado dos modelos por data de corte.

9.6 Otimização de inferência

O relatório técnico anexo à disciplina documenta um padrão de otimização relevante para deploy: a conversão do modelo treinado em um framework de alto nível (PyTorch) para um formato mais leve e adequado ao hardware alvo. O modelo é primeiro exportado para **ONNX (Open Neural Network Exchange)**, formato intermediário que garante interoperabilidade entre frameworks, e em seguida convertido para **TensorRT**, plataforma de inferência da NVIDIA que aplica fusão de camadas, quantização e seleção automática de kernels.

Os ganhos documentados são substanciais: um modelo baseado em Transformer passou de 618,06 ms para cerca de 176 ms por quadro, um speedup de 3,5 vezes; um modelo generativo passou de 257,57 ms para 128,46 ms, speedup de 2,0 vezes. Ambos passaram a caber no orçamento de 200 ms por quadro estabelecido pela aplicação, sem degradação mensurável de qualidade.

Duas lições generalizáveis emergem. Primeira, os speedups são **assimétricos**: arquiteturas Transformer se beneficiam mais porque suas operações dominantes (atenção, multiplicações de matrizes grandes, softmax, reorganização de tensores) são exatamente as priorizadas pela fusão de camadas e kernels em motores otimizados. Segunda, e mais importante metodologicamente: a otimização de hardware deve ser tratada como **parte integrante do processo de seleção de modelos**, e não como etapa posterior e opcional. Um modelo descartado por latência na implementação de referência pode ser perfeitamente viável após otimização.

9.7 Restrições operacionais como critério de projeto

O material fecha com um princípio que atravessa toda a disciplina: o orçamento de latência e o orçamento de custo são requisitos de projeto. Escolhas de modelo, de arquitetura e de banco vetorial se subordinam a eles. A recomendação final da aula de RAG multimodal é explícita: monitorar custos e latência, já que o uso de VLMs para sumarizar imagens aumenta o custo, e avaliar o retorno sobre investimento para cada caso de uso.

10 Síntese da Disciplina

A disciplina DAI constrói um argumento único ao longo de suas aulas: **sistemas de IA multimodais são problemas de representação e de arquitetura antes de serem problemas de modelo**.

O argumento começa pela **representação**. As aulas de processamento de áudio e de embeddings multimodais estabelecem que toda modalidade — sinal acústico, imagem, texto — pode ser projetada em um espaço vetorial onde proximidade geométrica corresponde a similaridade semântica. O aprendizado contrastivo é o mecanismo que produz esse alinhamento sem exigir anotação detalhada, e o CLIP é seu exemplar canônico. Uma vez alinhadas as modalidades, a matemática da busca — a similaridade cosseno — permanece a mesma independentemente de a query ser texto, imagem ou áudio. É esse fato aparentemente simples que sustenta a busca cross-modal, os assistentes visuais e o RAG multimodal.

Do espaço vetorial decorre a **infraestrutura**. Embeddings em escala exigem bancos vetoriais, e bancos vetoriais exigem busca aproximada, porque a busca exata é O(N) e inviável. FAISS, ChromaDB e Qdrant representam três pontos distintos no espectro entre biblioteca de pesquisa e sistema de produção, e a recomendação de começar em ChromaDB e migrar para Qdrant é um exemplo do pragmatismo que a disciplina cultiva. A busca híbrida — vetorial mais lexical mais filtros, fundida por RRF — reconhece que a busca semântica sozinha falha em correspondências exatas.

Da representação e da infraestrutura decorre a **aplicação**. A aula de RAG multimodal é o ponto onde tudo converge, e sua contribuição conceitual mais duradoura é o princípio da arquitetura multi-vetor: **indexa o resumo, guarda o original**. Esse princípio resolve simultaneamente dois problemas que pareciam antagônicos — a busca precisa ser rápida e densa, enquanto o contexto

entregue ao LLM precisa ser completo e estruturado. A evolução das três gerações de RAG, do pipeline ingênuo ao agente orquestrador, mostra que a inteligência do sistema migrou do pipeline fixo para a decisão dinâmica: o LLM deixa de ser o gerador final e passa a escolher ferramentas, iterar buscas e criticar seu próprio contexto.

O reconhecimento de locutor, tratado na segunda aula, é um microcosmo do mesmo padrão. A arquitetura em cascata (VAD, embeddings, clustering, ASR) sofre de propagação de erro e falta de otimização conjunta; a arquitetura end-to-end resolve esses problemas ao custo de flexibilidade; e a solução mais pragmática, que a disciplina recomenda explicitamente, é intermediária: reaproveitar a cascata e adicionar uma camada de pós-processamento com LLM que corrige desalinhamentos, consolida rótulos ambíguos e resolve inconsistências de contexto. É o mesmo LLM-como-camada-de-raciocínio que reaparece no agente do RAG.

Finalmente, o argumento se fecha em **engenharia**. As três últimas aulas — métricas, explicabilidade e deploy — tratam do que separa um experimento de um sistema. As métricas ensinam que uma única medida nunca basta: recall@K para recuperação, EER para verificação, e sempre latência e custo como métricas de primeira classe. A explicabilidade responde ao paradoxo da representação — quanto mais expressivo o modelo, mais difícil auditá-lo — com mecanismos concretos: citação de fonte, devolução do elemento original, roteamento transparente. E o deploy transforma o notebook em serviço por três movimentos: cache persistente, modularização e API REST.

O aluno que percorre a disciplina sai com três capacidades articuladas. Sabe **escolher** uma representação e uma ferramenta com base em critérios explícitos de qualidade, latência, custo e privacidade — API comercial para validar, open-source para controlar custo, local para privacidade. Sabe **construir** um pipeline multimodal completo, do parsing de um PDF caótico à resposta agêntica com citação de fonte. E sabe **avaliar e publicar** esse pipeline, medindo o que importa e entregando um serviço modular em vez de um notebook monolítico. Esse é, em última análise, o significado de "desenvolvimento de aplicações de IA".