Inteligência Artificial 2

Aula 11: Bancos de Dados Vetoriais
De $O(N)$ ao HNSW em $O(\log N)$ com ChromaDB

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Setembro - 2026/2

Onde Paramos?

Na aula anterior, vocês construíram um **buscador semântico** completo:

- **Busca Léxica** falha: sinonímia e polissemia destroem relevância.
- **Busca Semântica**: textos são vetores, e a busca vira geometria.
- **Cosseno** é a métrica padrão: ignora magnitude, foca no ângulo.
- **RAG**: Indexação $\rightarrow$ Busca $\rightarrow$ Injeção $\rightarrow$ Geração.
- Implementaram o cosseno `na mão` com NumPy e um laço `for`.

A Pergunta de Hoje

O laço `for` varre N documentos: $O(N \times D)$.

E se $N = 500$ milhões? Como **escalar**?

Objetivos da Sessão

- **O Gargalo k-NN:** Por que busca exata é inviável em escala industrial.
- **Vector Databases:** O que são, por que existem, e o ecossistema corporativo.
- **ANN (Approximate Nearest Neighbors):** O trade-off velocidade vs. precisão.
- **Técnicas de Indexação:** IVF (Voronoi), LSH (Hashing), e o estado da arte.
- **HNSW:** O algoritmo que domina o mercado — hierarquia de grafos navegáveis.
- **Parâmetros Críticos:** M, `ef_construction` e Recall@K.
- **ChromaDB:** Implementação prática — criar, inserir e buscar em Python.

O Gargalo Computacional do k-NN Exato

- Na aula anterior, o buscador calculava o cosseno contra **todos** os documentos.
- **Complexidade:** $O(N \times D)$ por query.
- Para $N = 10M$ e $D = 1536$ (OpenAI ada-002): **15,3 bilhões** de multiplicações... para responder **uma** pergunta.

Docs (N)	Multiplicações
1.000	1,5M
100.000	153M
1.000.000	1,5B
10.000.000	15,3B
500.000.000	768B

O Colapso

Se 100 usuários perguntarem ao mesmo tempo, são 1,53 **trilhões** de operações. Servidores entram em colapso. Latência real: $>$ 200ms por query — inaceitável para chat em tempo real.

Por que Bancos de Dados Clássicos Falham?

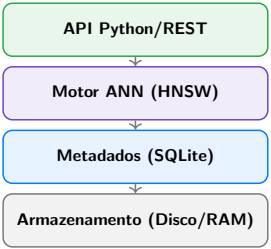
Tipo	Indexação	Por que Falha
SQL (PostgreSQL)	B-Tree	Otimizado para ordens lineares; inútil para distâncias em 1536 dimensões.
NoSQL (MongoDB)	Documentos JSON	Não entende topologia vetorial.
ElasticSearch	Índice Invertido	Busca por *palavras*, não por *vetores*.
Vector DB	**ANN / HNSW**	**Projetado para busca de vizinhos em alta dimensionalidade.**

A Necessidade

A indústria precisou criar sistemas que, no seu *core*, entendem **topologia vetorial** — os Vector Databases.

O que é um Banco de Dados Vetorial?

- Sistema projetado para armazenar, indexar e buscar **bilhões de Embeddings** junto com metadados.
- O diferencial: **algoritmos de indexação na RAM** que substituem busca exata por busca aproximada ultra-rápida.
- Abandona a garantia de resultado "perfeito" por resultados "bons o suficiente" em milissegundos.



O Ecossistema Corporativo Atual

Solução	Linguagem	Tipo	Diferencial
Pinecone	Cloud	SaaS	Zero-ops, popular em enterprise.
Milvus	C++/Go	Open-source	Distribuído (Kubernetes), escala massiva.
ChromaDB	Python	Open-source	Dev-friendly, embedded, ideal para RAG local.
FAISS	C++	Biblioteca	Meta (Facebook), motor matemático puro.
Qdrant	Rust	Open-source	Alto desempenho, filtros avançados.
Weaviate	Go	Open-source	GraphQL, módulos de ML integrados.

Bancos Nativos vs. Plugins Vetoriais

1. Nativos Vetoriais

Pinecone, ChromaDB, Weaviate, Milvus.

Construídos **do zero** para tensores.

Vantagem: Mais rápidos, flexíveis e otimizados para matrizes.

2. Plugins em Bancos Clássicos

`pgvector` (PostgreSQL), `RedisVector`.

Vetores "acoplados" a bancos tradicionais.

Vantagem: ACID, transações, e dados tabulares + vetores no mesmo banco.

Trade-off Arquitetural

Bancos nativos são mais rápidos para ANN. Plugins são mais simples se a empresa já tem PostgreSQL. A escolha depende do cenário de produção.

O Paradigma ANN (Approximate Nearest Neighbors)

- Ao invés de garantir o vizinho **perfeito** (k-NN exato), o ANN garante o vizinho "**bom o suficiente**" em tempo drasticamente menor.

Sacrificamos 2% de precisão em troca de 1000x mais velocidade

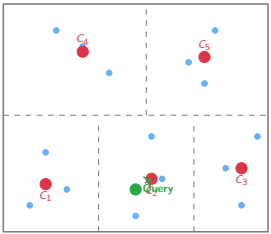
Técnicas de Indexação ANN

1. **IVF (Inverted File Index):** Divide o espaço em **clusters de Voronoi**. Na busca, localiza o centróide mais próximo e varre apenas aquela célula.
2. **LSH (Locality-Sensitive Hashing):** Projeta vetores em "baldes" usando funções hash sensíveis à localidade. Vetores similares caem no mesmo balde.
3. **HNSW (Hierarchical Navigable Small World):** Organiza vetores em um **grafo hierárquico** de múltiplas camadas. Estado da arte.

Técnica	**Complexidade**	**Recall**	**RAM**
IVF	$O(\sqrt{N})$	Médio	Baixa
LSH	$O(1)$ amortizado	Baixo	Média
HNSW	$O(\log N)$	**Alto**	**Alta**

Analogia: Voronoi — O Mapa dos Territórios

- O IVF divide o espaço vetorial em **regiões de Voronoi**.
- Cada região tem um **centróide** representativo.
- Na busca, o algoritmo acha o centróide mais próximo e varre **apenas** os vetores daquela região.
- **Analogia:** Procurar um restaurante? Primeiro descubra o *bairro* certo, depois vasculhe as ruas desse bairro.



HNSW: Hierarchical Navigable Small World

- Baseado na teoria dos **Seis Graus de Separação** (Redes de Pequeno Mundo): qualquer pessoa está a ≤ 6 conexões de qualquer outra no planeta.
- Organiza vetores num **grafo multi-camadas** com conexões entre vizinhos próximos.
- Aplica a lógica de **Skip List** (Listas de Salto) para acelerar a navegação:

Camada	**Analogia**	**Função**
Superior	"Voo intercontinental"	Poucos nós, saltos enormes
Intermediária	"Rodovia interestadual"	Densidade média, saltos moderados
Base (Layer 0)	"Ruas do bairro"	100% dos vetores, vizinhança exata

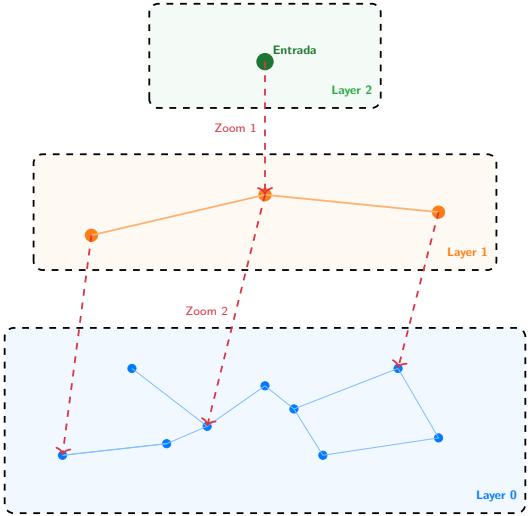
A Hierarquia de Camadas do HNSW

- **Camadas Superiores**: Poucos nós aleatórios, conexões longas ("voos internacionais").
- **Camadas Intermediárias**: Maior densidade, distâncias moderadas ("rodovias").
- **Camada Base (0)**: 100% dos documentos, conexões com vizinhos exatos ("ruas").

O Algoritmo de Busca

1. Entra no nível mais alto.
2. Localiza rapidamente a região geral mais próxima.
3. Desce para o nível abaixo ("zoom").
4. Repete até chegar à Layer 0.
5. Na Layer 0, faz busca fina no "bairro" correto.

Representação Visual do HNSW



Small World Networks: O Segredo de Milgram

- O HNSW é baseado na teoria de **Pequenos Mundos** (Stanley Milgram, 1967).
- A ideia: "Em média, duas pessoas estão separadas por não mais de 6 conexões".
- No HNSW, nós fortemente agrupados (mesmo assunto) recebem **arestas longas** aleatórias que cruzam o espaço.

O Atalho Espacial

Sem arestas longas, viajar do ponto A ao B no espaço vetorial exigiria percorrer centenas de pequenos nós vizinhos ($O(N)$). Com o "pequeno mundo", saltamos do A para o outro lado do grafo em apenas 1 salto, depois refinamos a busca.

Como o M afeta o Grafo (Densidade)

- Se o parâmetro M (arestas máximas por nó) for muito baixo ($M = 4$):
  - Grafo esparso. Rápido de construir e pouca RAM.
  - Risco de "becos sem saída" na navegação. Busca presa num mínimo local.
- Se M for muito alto ($M = 64$):
  - Grafo denso. Quase certeza de achar o vizinho exato (Recall $> 99\%$).
  - Custo brutal de RAM e inserção lenta.

O Sweet Spot

A indústria padronizou $M = 16$ ou $M = 32$ para embeddings de IA (como OpenAI de 1536 dimensões), garantindo recall quase perfeito com uso aceitável de memória.

Complexidade Temporal: A Mágica do $\log N$

- Os "saltos" nas camadas superiores **ignoram 99,99%** dos vetores.
- A busca que demoraria horas cai para $O(\log N)$.
- **100 milhões de PDFs**: tempo de resposta entre 10ms e 50ms.

Docs	k-NN	HNSW
10^3	1,5M ops	$\sim$30 ops
10^6	1,5B ops	$\sim$60 ops
10^8	150B ops	$\sim$80 ops
10^9	1,5T ops	$\sim$90 ops

O Milagre Estatístico

É esse $O(\log N)$ que viabiliza o ChatGPT Enterprise, Copilot, e toda a arquitetura RAG corporativa moderna. Sem HNSW, a IA generativa não escalaria.

Os Parâmetros Críticos do HNSW

M (Conexões por Nó)

Quantas arestas cada ponto pode ter no grafo.

M alto $\Rightarrow$ Recall alto, mas mais RAM.
M baixo $\Rightarrow$ Menos RAM, mas pior recall.

Padrão: $M = 16$

ef_construction

Profundidade de varredura durante a **inserção**.

Alto $\Rightarrow$ Grafo hiper-otimizado, INSERT lento.
Baixo $\Rightarrow$ INSERT rápido, busca menos precisa.

Padrão: $ef = 200$

Analogia

M é o "número de amigos" de cada nó no grafo. `ef_construction` é o "quanto tempo você gasta conhecendo pessoas no primeiro dia" — mais tempo = rede social melhor, mas o cadastro demora mais.

Métricas de Avaliação: Recall@K

- Como provar que a busca aproximada (ANN) não perde resultados críticos?
- **Recall@K**: "Dos K vizinhos absolutamente perfeitos (força bruta), quantos o ANN encontrou?"

$$\text{Recall@K} = \frac{|\text{Resultados ANN} \cap \text{Resultados Exatos}|}{K}$$

Configuração	**Recall@10**	**Latência**
k-NN Exato (baseline)	100%	2.500ms
HNSW ($M = 16$, $ef = 100$)	95%	3ms
HNSW ($M = 32$, $ef = 200$)	**99%**	8ms

Na Prática

Um Vector DB bem tunado atinge Recall@10 $> 98\%$, rodando **300x mais rápido** que a busca exata.

Latência de I/O vs. RAM

- HNSW é **Memory-Bound** — o grafo inteiro precisa caber na RAM.
- Ler grafos do disco (SSD/NVMe) a cada salto de nó introduz latência de I/O catastrófica.
- Se o índice exceder a RAM, ocorre **swapping** (paginação) e a latência explode.

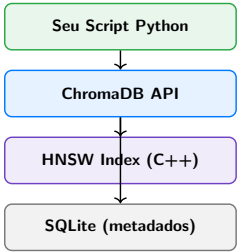
Armazenamento	**Latência/Acesso**	**Viável para HNSW?**
RAM (DDR5)	$\sim$80ns	✓ Ideal
NVMe SSD	$\sim$100μs	$\sim$ Tolerável
HDD	$\sim$10ms	✗ Inviável

Regra Prática

Para cada 1M de vetores de 768D (float32), reserve $\sim$3GB de RAM apenas para o índice HNSW.

O Ecossistema ChromaDB

- No mundo RAG, o **ChromaDB** se consagrou pela **fricção-zero** e integração nativa com Python.
- Usa SQLite para metadados + **HNSW** para vetores.
- Dois modos de operação:
  - **In-Memory:** Volátil, ideal para testes.
  - **Persistent:** Salva em disco (`.sqlite3` + binários).



ChromaDB: Inicialização e Coleções

No ChromaDB, dados ficam em *Collections* (análogo a tabelas SQL):

```python
import chromadb

# Cliente persistente - dados salvos em disco
client = chromadb.PersistentClient(path="./meu_banco")

# Cria uma "tabela" vetorial com metrica cosseno
collection = client.create_collection(
    name="documentos_empresa",
    metadata={"hnsw:space": "cosine"}  # Usa Similaridade de Cosseno
        !
)
```

O que Acontece por Baixo

O parâmetro `hnsw:space: cosine` diz ao ChromaDB para normalizar os vetores (L2) e construir o grafo HNSW otimizado para busca angular.

ChromaDB: Inserção de Documentos

Podemos fornecer vetores prontos ou deixar o Chroma gerar via API:

```
# Inserindo documentos com vetores, metadados e IDs
collection.add(
    embeddings=[
        [0.1, 0.4, -0.2, ...],  # Vetor do documento 1
        [-0.5, 0.2, 0.9, ...]   # Vetor do documento 2
    ],
    documents=[
        "Ferias sao pedidas no portal RH.",
        "Servidores precisam de update semanal."
    ],
    metadatas=[
        {"departamento": "RH"},
        {"departamento": "TI"}
    ],
    ids=["doc_1", "doc_2"]
)
```

ChromaDB: Busca Semântica Otimizada

Com HNSW rodando sob o capô, buscamos Top-K com filtros de metadados:

```
# Busca semantica com filtro de departamento
resultados = collection.query(
    query_embeddings=[query_vetor],  # Vetor da pergunta do usuario
    n_results=2,                     # Top-K (os 2 mais proximos)
    where={"departamento": "RH"}     # Filtro SQL-like no metadado!
)

print(resultados['documents'][0])
# Saida em milissegundos: ['Ferias sao pedidas no portal RH.']
print(resultados['distances'][0])
# Saida: [0.12]  (distancia cosseno)
```

O Diferencial

A busca combina **ANN vetorial** (HNSW) com **filtros SQL** (WHERE) — semântica + metadados numa única query.

Recapitulação — O que Levar Desta Aula

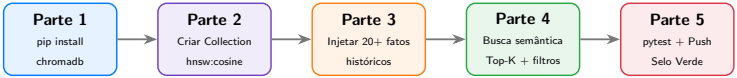
1. **k-NN Exato** é $O(N \times D)$ — inviável para milhões de vetores.
2. **Bancos SQL/NoSQL** não entendem topologia vetorial. Vector DBs sim.
3. **ANN**: sacrifica $\sim$2% de recall por 100–1000x de velocidade.
4. **HNSW**: grafo hierárquico com Skip List — $O(\log N)$.
5. **Parâmetros**: M (conexões) e `ef_construction` (profundidade no INSERT).
6. **Recall@K**: métrica para validar a qualidade da busca aproximada.
7. **ChromaDB**: Python-friendly, HNSW + SQLite, ideal para RAG.

Próxima Aula

Temos o banco vetorial pronto. Mas como transformar um **PDF de 200 páginas** em vetores? A arte do **Chunking** e do pipeline de ingestão documental.

Roteiro do Laboratório 11

Missão: Implementar um banco vetorial real com ChromaDB — ingestão, indexação HNSW e busca semântica.



O Salto

Vocês saem da força bruta (Lab 10) para a infraestrutura profissional (Lab 11). Entender como injetar e consultar dados em Vector DBs é **80%** do trabalho de um AI Engineer moderno.

Lab Passo 1: O Cliente ChromaDB

- Primeiro passo: instanciar o banco localmente.
- O ChromaDB salvará todos os índices numa pasta `./chroma_data`.

```
import chromadb

# Cria um banco persistente na pasta indicada
client = chromadb.PersistentClient(path="./chroma_data")
print("Bancos disponiveis:", client.list_collections())
```

Lab Passo 2: A Collection de Fatos

- Criaremos uma `Collection` (análoga a tabela).
- Importante: usaremos `hnsw:space = "cosine"`.

```
collection = client.create_collection(
    name="fatos_historicos",
    metadata={"hnsw:space": "cosine"}
)
print("Collection criada!")
```

Lab Passo 3: Injeção de Documentos

- ChromaDB usará um modelo de embedding embutido (MiniLM) para vetorizar em background!
- Não precisamos chamar o spaCy manualmente.

```
collection.add(
    documents=["O Brasil foi descoberto em 1500", "A Revolucao
        Francesa ocorreu em 1789"],
    metadatas=[{"pais": "Brasil", "sec": 16}, {"pais": "Franca", "
        sec": 18}],
    ids=["id1", "id2"]
)
```

Lab Passo 4: Busca Semântica Híbrida

- Buscaremos usando linguagem natural E filtro de metadado.

```
resultados = collection.query(
    query_texts=["Quando o pais foi achado?"],
    n_results=1,
    where={"pais": "Brasil"} # O filtro SQL entra aqui
)

print(resultados['documents'][0])
# Resultado: ['O Brasil foi descoberto em 1500']
```

Lab Passo 5: Testes e Avaliação

- O pytest vai verificar se a pasta `./chroma_data` existe.
- Ele fará buscas complexas que falhariam em busca léxica.
- Verificará se vocês aplicaram o filtro `where` corretamente.
- Se sim, Selo Verde no GitLab!

Dúvidas?

Email: alessio@cefetmg.br
Aula: Segunda e Quarta-Feira, 7h as 8:40h (Sala 39)
WhatsApp: Dúvidas rápidas

Lembrete

Confira sua Issue no GitLab!

Próxima aula: PDF Chunking & Ingestão

100% da Aula 11 Concluída!