

Inteligência Artificial 2

Aula 11 (Parte 2): Filtros e Trade-offs de Recuperação Referência exata, ANN e evidência mensurável

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timóteo
Dep. Engenharia de Computação

Setembro - 2026/2

Onde Paramos na Parte 1?

Na primeira sessão, construímos uma infraestrutura vetorial profissional:

- **Contrato de Embeddings:** modelo multilíngue, $D = 384$, métrica cosseno e normalização.
- **Persistência com ChromaDB:** catálogo SQLite + arquivos binários do índice em disco.
- **Ingestão Idempotente:** IDs determinísticos e `upsert()` evitando duplicações.
- **Busca Top-K Multilíngue:** consultas em inglês encontrando documentos em português.

O Dilema do Mundo Real

Similaridade semântica não basta: usuários exigem restrições de categoria, data e permissões.
Como filtrar metadados sem quebrar a busca? E como provar matematicamente que o índice aproximado não perdeu os documentos certos?

Objetivos da Sessão (Parte 2)

- **Semântica vs. Política:** compreender por que embeddings sozinhos falham em regras de negócio.
- **Arquiteturas de Filtragem:** analisar pós-filtro, pré-filtro e travessia com predicado em grafo.
- **Filtros Compostos no ChromaDB:** aplicar operadores lógicos (`$and`, `$or`) e relacionais (`$gte`).
- **O Baseline Exato como Ground Truth:** construir o gabarito matricial em NumPy.
- **Métrica Formal Recall@K:** calcular a fidelidade algorítmica com exemplo numérico passo a passo.
- **O Algoritmo HNSW:** entender Small Worlds, Skip Lists e os parâmetros M , $efConstruction$ e $efSearch$.
- **Triângulo de Trade-offs:** equilibrar latência, memória RAM e qualidade de recuperação.

O que o Embedding Responde

"Quais textos compartilham vizinhança semântica com a consulta?"

- Opera em espaço contínuo ($\mathbb{R}^D$).
- Não entende restrições booleanas estritas.
- Ignora permissões e limites temporais.

O que a Regra de Negócio Exige

"Quais documentos são permitidos pela política do produto?"

- Apenas categoria "ciencia".
- Ano de publicação ≥ 1900 .
- Usuário com autorização no departamento.

O Exemplo Histórico do Laboratório

Consulta: **"descobertas médicas do século XX"**.

- **Edward Jenner (1796)**: vacina da varíola. Similaridade máxima, mas **INELEGÍVEL** (século XVIII!).
- **Alexander Fleming (1928)**: penicilina. Similaridade alta e **ELEGÍVEL** (século XX).

As 3 Arquiteturas de Filtragem Vetorial

1. Post-Filtering

Busca ANN Top-K Global



Aplica filtro SQL no Top-K



Risco Crítico:
Retornar lista vazia se todos
forem inelegíveis!

2. Pre-Filtering

Filtra IDs elegíveis no SQL



Busca ANN no subconjunto



Desafio em Grafos:
Quebra conexões do HNSW;
pode degradar para $O(N)$.

3. Single-Stage (Iterativo)

Navega no grafo HNSW completo



Mascara nós em tempo real



Solução Moderna:
Usa nós inválidos como pontes,
mas só entrega elegíveis!

Estado da Arte

Motores modernos (Chroma, Qdrant) adotam a abordagem **Single-Stage**, navegando no grafo com filtros avaliados durante o salto entre nós.

A Armadilha do Post-Filtering (Pós-Filtro)

O Cenário de Falha:

- Usuário pede $K = 3$ resultados:
“*descobertas médicas do século XX*”.
- O índice vetorial calcula os 3 mais próximos globalmente:
 1. Jenner (1796) — dist: 0,25
 2. Pasteur (1885) — dist: 0,30
 3. Lister (1865) — dist: 0,32
- Em seguida, o sistema aplica o filtro: ano ≥ 1900 .
- **Resultado entregue: 0 documentos!**

Lição: Filtros devem condicionar o espaço de busca, nunca agir como tesoura cega sobre o Top- K .

O Desastre de Experiência

- O banco possuía Fleming (1928) e Salk (1953) perfeitamente indexados!
- Mas como estavam em 4º e 5º lugar no ranking global, foram ignorados pelo corte prévio de $K = 3$.
- O usuário recebe uma mensagem de “nenhum resultado encontrado”, minando a confiança no sistema.

O Dilema do Pre-Filtering Tradicional

Por que não apenas pré-filtrar no SQL?

- No pré-filtro clássico, executamos primeiro:
`SELECT id FROM docs WHERE ano >= 1900.`
- Suponha que de 10M de vetores, restem apenas 5.000 IDs elegíveis.
- **O Problema em Grafos HNSW:**
 - O grafo HNSW conecta nós através de rotas hiperdimensionais de longo alcance.
 - Remover 99,9% dos nós fragmenta o grafo em ilhas desconexas.
 - O algoritmo de busca gulosa fica preso em mínimos locais ou falha ao navegar.

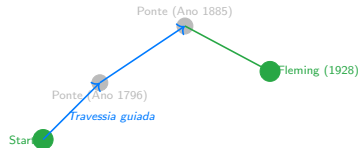


Consequência

O pré-filtro em grafos frequentemente força o banco a abandonar o HNSW e rodar busca exata nos IDs restantes.

Como o HNSW resolve o dilema:

- O grafo HNSW permanece **100% intacto**, preservando todas as arestas e pontes de navegação.
- Cada documento recebe uma **máscara de bits de elegibilidade** gerada pelo filtro relacional.
- **Durante a busca gulosa:**
 1. Nós não elegíveis podem ser visitados para avançar em direção à query (atuam como pontes).
 2. Porém, apenas nós com bit ativo (1) entram na lista final de candidatos ao Top- K .



Eficiência

Garante que sempre retornaremos K itens elegíveis (se existirem), sem quebrar a geometria do grafo.

Sintaxe de Filtros Estruturados no ChromaDB

O ChromaDB adota a convenção de operadores do MongoDB no parâmetro where:

```
resultados = colecao.query(  
    query_texts=["descoberta medica salvando vidas"],  
    n_results=3,  
    where={  
        "$and": [  
            {"categoria": {"$eq": "ciencia"}},  
            {"ano": {"$gte": 1900}}  
        ]  
    },  
    include=["documents", "metadatas", "distances"]  
)
```

Operador	Significado	Exemplo de Uso
\$eq / \$ne	Igual / Diferente	{"categoria": {"\$eq": "ciencia"}}
\$gt / \$gte	Maior / Maior ou igual	{"ano": {"\$gte": 1900}}
\$lt / \$lte	Menor / Menor ou igual	{"ano": {"\$lt": 2000}}
\$in / \$nin	Pertence / Não pertence a lista	{"categoria": {"\$in": ["ciencia", "saude"]}}
\$and / \$or	Conjunção / Disjunção lógica	{"\$and": [{cond1}, {cond2}]}

A Pegadinha da Tipagem Dinâmica

Em Python, podemos acidentalmente passar números como texto:

```
"ano": "1900" vs. "ano": 1900
```

Consequência no SQLite:

- Se o metadado for string, o filtro `$gte: 1900` aplicará **ordem alfabética (lexicográfica)**!
- Na ordem lexicográfica, "200" é maior que "1928".
- Metadados numéricos corrompidos invalidam os filtros silenciosamente!

Armazenamento Interno no ChromaDB

Dentro de `chroma.sqlite3`, o catálogo mantém tabelas normalizadas:

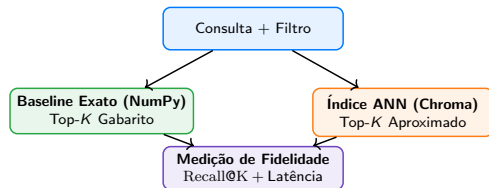
- `string_value` para textos e categorias.
- `int_value` para inteiros (datas, contadores).
- `float_value` para métricas contínuas.

O Chroma traduz o dicionário `where` em cláusulas SQL nativas indexadas no SQLite.

Boas Práticas: Valide sempre tipos no código com dataclasses ou Pydantic antes do `upsert`.

A Ilusão da Velocidade: Por que Precisamos do Baseline?

- Um índice ANN retorna resultados em 1,5 ms.
- **Mas rápido não significa correto!**
- Como um engenheiro de IA prova que o HNSW não ignorou o documento mais relevante?
- Em algoritmos aproximados, **precisamos de um Ground Truth de engenharia**:
 - Uma busca exata de força bruta ($O(N_{\text{eleg}} \times D)$) rodando sobre os mesmos vetores.
 - Ela nos dá a **resposta perfeita matemática** para calibrar o índice.



Construindo o Baseline Exato com NumPy

Implementação da referência exata com álgebra linear vetorizada em BLAS/C:

```
import numpy as np

def top_k_exato(query_vec: np.ndarray, doc_vecs: np.ndarray,
               ids: list[str], k: int) -> list[str]:
    """Calcula o Top-K exato por produto escalar normalizado (cosseno)."""
    # 1. Normalizacao L2 da query e da matriz de documentos
    q_norm = query_vec / (np.linalg.norm(query_vec) + 1e-9)
    docs_norm = doc_vecs / (np.linalg.norm(doc_vecs, axis=1, keepdims=True) + 1e-9)

    # 2. Produto matricial batch: similaridade cosseno exata O(N_elegiveis x D)
    similaridades = docs_norm @ q_norm

    # 3. Ordenacao decrescente dos indices
    ranking_idx = np.argsort(similaridades)[::-1][:k]

    # 4. Retorna a lista ordenada dos IDs exatos
    return [ids[i] for i in ranking_idx]
```

Propósito: Essa função é nosso **padrão-ouro**. O ANN é avaliado por quão próximo chega desse gabarito.

Definição matemática da fidelidade de um algoritmo aproximado em relação ao exato:

$$\text{Recall@K}_{\text{ANN}} = \frac{|L_K^{\text{exato}} \cap L_K^{\text{ANN}}|}{\min(K, |L_K^{\text{exato}}|)} \quad (1)$$

1. O Numerador

Cardinalidade da interseção entre os IDs retornados pelo ANN e pelo Baseline Exato no Top- K . Mede quantos vizinhos verdadeiros o ANN conseguiu capturar.

2. O Denominador: $\min(K, |L_K^{\text{exato}}|)$

Se o filtro for tão restrito que existam apenas 2 documentos elegíveis no corpus, $|L_K^{\text{exato}}| = 2$. Mesmo pedindo $K = 5$, o ANN só poderá acertar 2. Usar K fixo geraria $2/5 = 40\%$, penalizando injustamente o algoritmo!

Se $|L_K^{\text{exato}}| = 0$ (nenhum documento elegível), definimos formalmente $\text{Recall@K} = 0,0$.

Diferença Crucial: Fidelidade do ANN vs. Relevância Humana

Dimensão	Fidelidade do ANN (Recall@K)	Qualidade de IR (Precision / nDCG)
O que avalia?	O índice HNSW achou os mesmos vetores que o cálculo matricial exato?	O texto recuperado atende à necessidade real do usuário humano?
Depende de:	Algoritmo de busca em grafo, <i>M</i> , <i>efSearch</i> .	Modelo de embedding, chunking, curadoria de texto.
Exige Gabarito?	Não: gerado automaticamente pelo baseline exato em NumPy.	Sim: exige anotação e julgamento humano de relevância.
Interpretação	Se Recall@K = 1,0, o algoritmo ANN foi perfeito.	Mesmo com Recall@K = 1,0, a resposta pode ser semanticamente inútil!

Conclusão Arquitetural

Recall@K do ANN afere **engenharia de indexação**. Relevância afere **ciência de dados**.

Exemplo Numérico: Calculando Recall@K na Ponta do Lápis

Considere uma consulta com $K = 3$ sobre um corpus filtrado com 5 documentos elegíveis:

1. Gabarito Exato (NumPy)

Vetorização completa $O(N_{\text{eleg}} \times D)$:

$$L_3^{\text{exato}} = [\text{"doc-002"}, \text{"doc-005"}, \text{"doc-008"}]$$

2. Retorno do ANN (HNSW)

Busca rápida com aproximação:

$$L_3^{\text{ANN}} = [\text{"doc-002"}, \text{"doc-008"}, \text{"doc-011"}]$$

Passo a Passo do Cálculo:

1. Interseção dos conjuntos:

$$\{\text{doc-002}, \text{doc-005}, \text{doc-008}\} \cap \{\text{doc-002}, \text{doc-008}, \text{doc-011}\} = \{\text{doc-002}, \text{doc-008}\}.$$

2. Contagem de acertos: $|\text{interseção}| = 2$ documentos.

3. Denominador: $\min(K, |L_K^{\text{exato}}|) = \min(3, 3) = 3$.

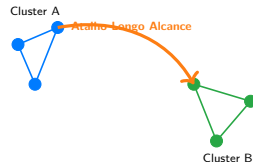
4. Resultado Final:

$$\text{Recall@3} = \frac{2}{3} \approx 0,667 \quad (\mathbf{66,7\%})$$

O documento "doc-005" foi perdido pelo HNSW em favor do "doc-011".

Como a Aproximação Funciona: Redes Small World

- **Experimento de Stanley Milgram (1967):** qualquer pessoa na Terra está conectada a outra por cerca de 6 laços sociais (*Seis Graus de Separação*).
- Grafos do tipo **Small World** possuem:
 1. **Alto Agrupamento Local:** vizinhos de um nó tendem a ser vizinhos entre si (clusters densos).
 2. **Caminho Médio Curto:** existem arestas de longo alcance que cruzam o grafo inteiro rapidamente.
- Na busca vetorial, navegamos de forma gulosa (**Greedy Search**): saltamos para o vizinho mais próximo da query até convergir.



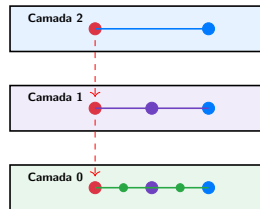
O Desafio do Grafo Plano (NSW)

Em grafos planos, o algoritmo pode ficar preso em mínimos locais se não houver hierarquia de camadas.

A Estrutura do HNSW: Skip Lists em Grafos

O **HNSW** (Hierarchical Navigable Small World) resolve mínimos locais empilhando grafos em camadas, inspirado nas **Skip Lists**:

- **Camada 2 (Top Layer):** Pouquíssimos nós e arestas longas $\Rightarrow$ saltos continentais (zoom out).
- **Camada 1 (Mid Layer):** Densidade intermediária $\Rightarrow$ rodovias estaduais.
- **Camada 0 (Base Layer):** Contém **100% dos documentos** $\Rightarrow$ navegação fina rua a rua.
- **Complexidade média de busca:** reduzida de $O(N)$ para $O(\log N)$.



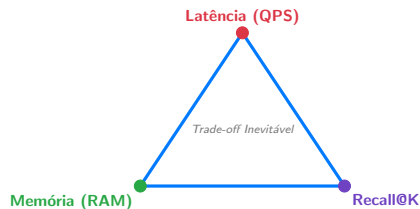
Os Parâmetros Críticos do HNSW

Parâmetro	Valor Típico	Impacto no Sistema
M	16 a 64	Número máximo de conexões bidirecionais por nó. Maior M melhora Recall em alta dimensão, mas consome mais memória RAM e eleva tempo de inserção.
$efConstruction$	100 a 400	Profundidade de exploração durante a indexação. Maior valor constrói conexões mais precisas. Custo pago apenas uma vez (na ingestão).
$efSearch$	10 a 200	Tamanho da fila prioritária na consulta. Parâmetro dinâmico da query: dobrar $efSearch$ aumenta o Recall@K ao custo de mais saltos e maior latência.

O Truque de Produção

Podemos manter M e $efConstruction$ fixos na indexação e ajustar apenas o $efSearch$ em tempo de execução para balancear latência e recall conforme a demanda de pico do produto!

O Triângulo de Trade-offs do HNSW



A Lei da Conservação

Você pode escolher dois vértices; o terceiro pagará o custo:

- **Alto Recall + Baixa Latência:** exige M alto e grafo 100% em RAM DDR5 (alto custo de servidor).
- **Baixa Memória + Baixa Latência:** exige quantização agressiva (PQ), derrubando o Recall@K.
- **HNSW é Memory-Bound:** se o grafo fizer swap em disco, a latência salta de microssegundos para dezenas de ms!

Tabela Comparativa do Experimento (Corpus Histórico)

Resultados medidos no Laboratório 11 (Parte 2) para a consulta “*descobertas médicas do século XX*”:

Modo	Filtro Aplicado	Top-1 Retornado	Recall@3	Status de Negócio
Sem Filtro (Puro)	Nenhum	Jenner (1796)	N/A	Violou Regra (séc XVIII)
Com Filtro ANN	Ciência, ≥ 1900	Fleming (1928)	1,00	Elegível e Rápido
Baseline Exato	Ciência, ≥ 1900	Fleming (1928)	1,00	Gabarito de Engenharia

Interpretação dos Resultados

- O filtro eliminou a vacina da varíola (1796) e promoveu a penicilina (1928) e a estrutura do DNA (1953).
- O HNSW alcançou Recall@3 = 1,00 (100% de fidelidade) preservando exatamente a ordem do baseline exato.

1. **Semântica não é Política:** Similaridade vetorial aponta proximidade; filtros definem elegibilidade.
2. **Single-Stage Traversal:** Motores modernos avaliam predicados em grafo sem truncar resultados.
3. **Tipagem Estrita:** Mantenha anos como inteiros e categorias como strings para evitar falhas no SQLite.
4. **Baseline Exato como Gabarito:** Força bruta em NumPy fornece a referência de calibração do ANN.
5. **Recall@K Formal:** Mede a fração de vizinhos verdadeiros capturados, com denominador $\min(K, |L_K^{\text{exato}}|)$.
6. **HNSW:** Grafo multicamadas navegável em $O(\log N)$; balanceie M , $efConstruction$ e $efSearch$ conforme o SLO.

Objetivo Prático: Construir o módulo `filtros_recuperacao.py` no projeto

ia2-m2-alessio.



Critério de Conclusão

Executar `pytest tests/test_filtros_recuperacao.py` e validar a tabela de fidelidade com 6/6 testes aprovados.

O que dominamos na Aula 11:

- Coleções persistentes, contrato vetorial e busca multilíngue.
- Filtros estruturados combinados com similaridade contínua.
- Auditoria formal de recuperação com Baseline Exato e Recall@K.

Mas há uma premissa oculta...

- Até aqui, injetamos frases curtas e prontas (fatos históricos resumidos).
- No mundo corporativo real, os dados são **manuals técnicos de 200 páginas, PDFs escaneados e planilhas!**

Próxima Aula (Aula 12)

Chunking e Ingestão de Documentos Longos

Como fatiar PDFs reais sem perder contexto?
Overlap, delimitadores e preservação de proveniência em escala.

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: **Chunking e Ingestão de Documentos Longos**

100% da Aula 12 Concluída!