# Ubicación del Capítulo en la Memoria

Ubicación recomendada del capítulo:  
Situar “Almacenamiento Vectorial y Comparativa FAISS vs ChromaDB” tras el capítulo de arquitectura del sistema/pipeline de ingesta y antes del capítulo de evaluación y resultados globales del TFM. De este modo, el lector recibe primero el diseño general de la solución, a continuación el subsistema de recuperación semántica (vector store) con su justificación y, por último, la evaluación end‑to‑end.

# 1. Almacenamiento Vectorial y Comparativa FAISS vs ChromaDB

## 1.1 Introducción a los Vector Stores

Los vector stores son bases de datos especializadas en indexar y recuperar vectores de embeddings para realizar búsqueda semántica. En sistemas RAG, el almacenamiento vectorial permite seleccionar fragmentos relevantes de un corpus a partir de la similitud entre la representación de la consulta y la de los documentos. En el contexto de administraciones públicas, habilitan consultas naturales sobre normativa, ordenanzas y trámites.

## 1.2 Fundamentos Teóricos

Embeddings semánticos. Utilizamos Sentence-Transformers para convertir texto en vectores densos; se evaluaron dos modelos en este TFM: all-MiniLM-L6-v2 (dim=384) e intfloat/multilingual-e5-base (dim=768). También se referencia BAAI/bge-m3 como candidato multilingüe avanzado.

Métricas de similitud: se emplea similitud coseno; en FAISS se usa producto interno con normalización L2 (equivalente a coseno).

## 1.3 FAISS (Facebook AI Similarity Search)

Arquitectura técnica: IndexFlatIP con normalización L2 en embeddings y queries.

Ejemplo de implementación:

import faiss  
import numpy as np  
  
# Normalización L2 para similitud coseno  
faiss.normalize\_L2(embeddings)  
index = faiss.IndexFlatIP(dimension)

Ventajas: búsqueda exacta (recall 100% respecto al índice), artefactos compactos. Limitaciones: coste lineal con N·D y mayor latencia a gran escala.

## 1.4 ChromaDB

Arquitectura técnica: HNSW (Hierarchical Navigable Small World) con métrica coseno en cliente persistente.

Ejemplo de implementación:

import chromadb  
client = chromadb.PersistentClient(path="models/chroma/collection")  
collection = client.get\_or\_create\_collection(name="chunks\_default", metadata={"hnsw:space": "cosine"})

Ventajas: baja latencia (ANN), filtrado por metadatos, escalabilidad sublineal. Limitaciones: recall aproximado (<100%), complejidad de parametrización HNSW.

## 1.5 Arquitectura Común Implementada

Pipeline de indexación unificado (scripts/index\_chunks.py), gestión de artefactos (index\_meta.json, index\_manifest.json), trazabilidad por hash y UI administrativa /admin/vector\_store.

# 2. Metodología de Evaluación

## 2.1 Corpus de Evaluación

Corpus municipal de Onda (source\_id=1) indexado como colección 'onda\_docs'. El snapshot evaluado contiene n\_chunks=29.834 (según index\_meta.json).

## 2.2 Métricas de Evaluación Implementadas

Recall@k: R@k = |relevant ∩ retrieved@k| / |relevant|; MRR = (1/|Q|) Σ (1/rank\_i). Se reportan además p50/p95 de latencia, y tasa text@k (si el texto contiene el patrón esperado).

## 2.3 Protocolo de Evaluación

Consultas en español (n=11). Evaluación para k ∈ {5,10,20,40}. Para ChromaDB no se ha variado ef\_search en este ciclo (queda como trabajo futuro).

# 3. Resultados Experimentales

## 3.1 Resultados de Construcción de Índices

Tiempos y dimensiones reales para E5 (intfloat/multilingual-e5-base):

|  |  |  |  |
| --- | --- | --- | --- |
| Vector Store | Tiempo Construcción (s) | Dimensión | N\_Chunks |
| FAISS (E5) | 3844.190 | 768 | 29834 |
| ChromaDB (E5) | 3901.709 | 768 | 29834 |

## 3.2 Métricas de Recuperación

MiniLM (all-MiniLM-L6-v2):

Resultados con all‑MiniLM‑L6‑v2 (dim=384):  
• ChromaDB — doc@20=40.0%, doc@25=60.0%, doc@30=60.0%; chunk@k≈0.0%. Latencias p50≈18.5 ms.  
• FAISS — doc@20=20.0%, doc@25=60.0%, doc@30=60.0%; chunk@k≈0.0%. Latencias p50≈15.9 ms.  
Observación: ampliar k de 20→25 elevó doc@k hasta 60% en ambos stores (caso “tasas municipales”).

E5 (intfloat/multilingual-e5-base):

Resultados con intfloat/multilingual‑e5‑base (dim=768):  
• doc@k: 40% con k=20 y 60% con k=40 en FAISS y ChromaDB; chunk@k≈20.0%.  
• Latencia (p50): FAISS 46.4→46.0 ms (k=20→40); ChromaDB 40.6→43.1 ms.  
Implicación: con k=40 se captura el documento relevante lejano con coste de latencia marginal.

Fig. 1. doc@k por store (MiniLM).

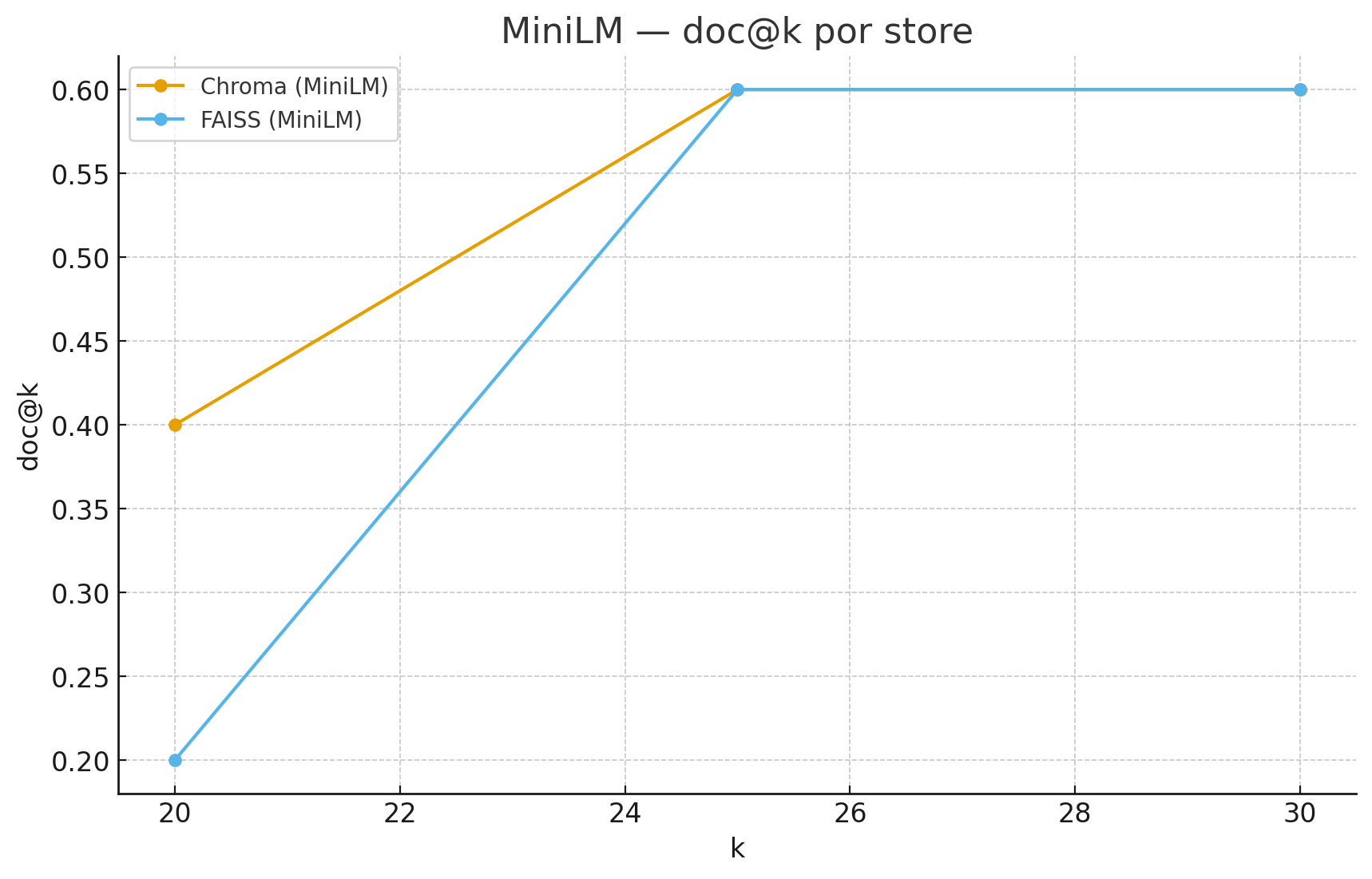


Fig. 2. doc@k por store (E5).

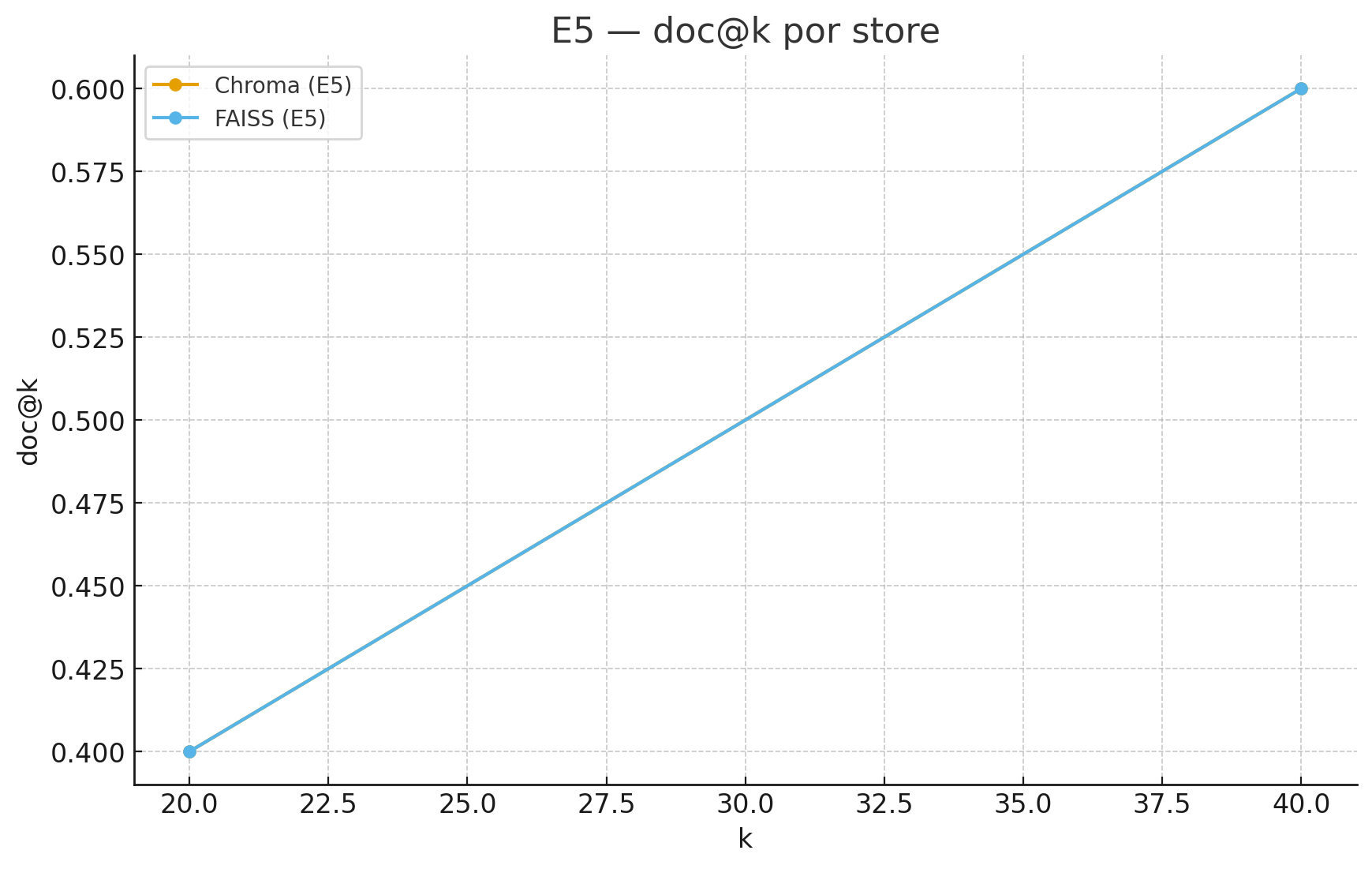
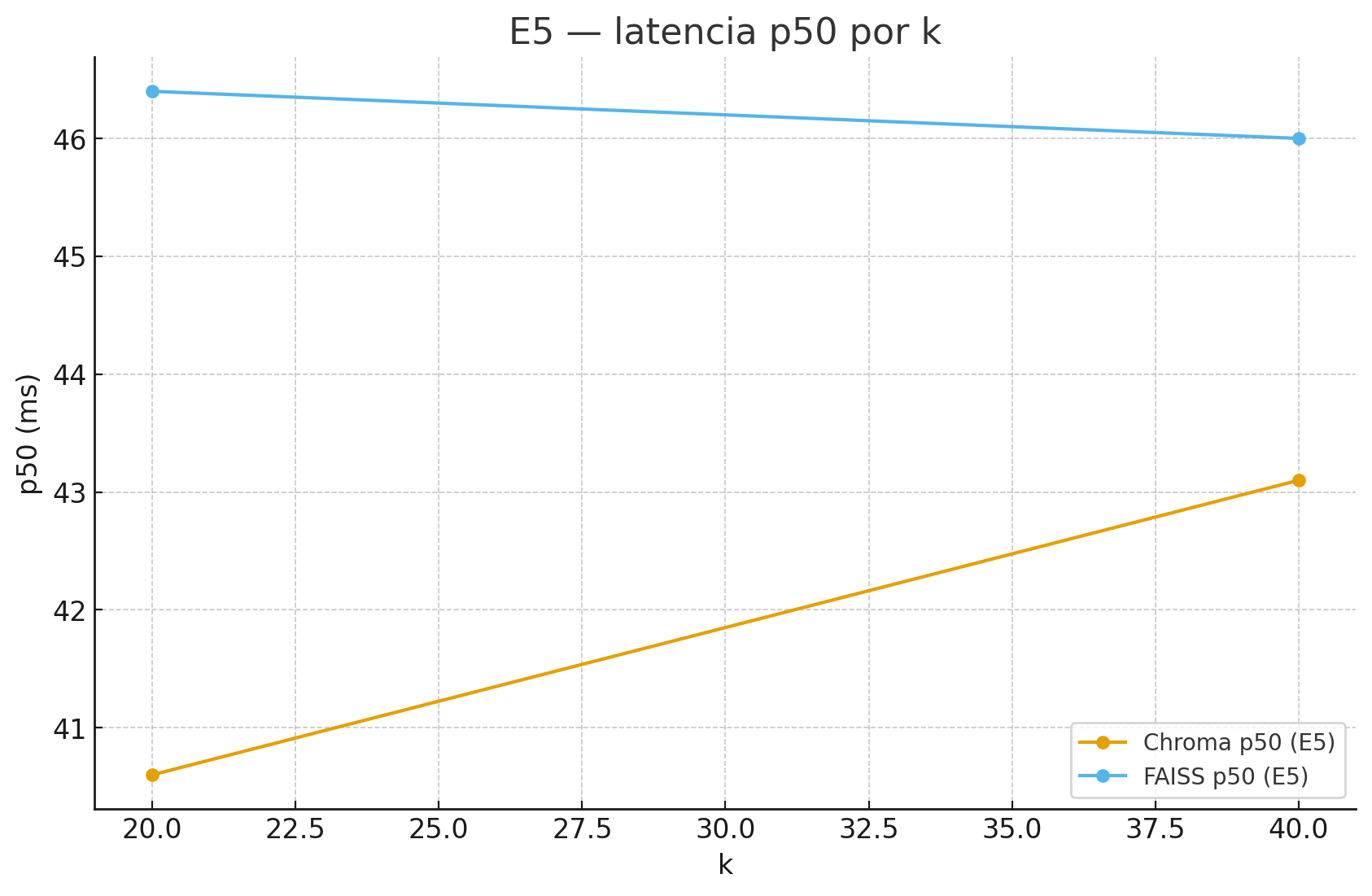


Fig. 3. Latencia p50 por k (E5).



## 3.3 Análisis de Top-k Overlap

Se define como la intersección relativa entre los conjuntos top-k de FAISS y ChromaDB. No se midió en este ciclo; se propone como extensión futura para cuantificar divergencias.

# 4. Análisis y Discusión

## 4.1 Trade-offs Identificados

Precisión vs Latencia: FAISS (exacto) vs ChromaDB (ANN). Escalabilidad: FAISS O(N·D) mientras Chroma reduce latencia con HNSW. Filtrado: Chroma ofrece metadatos nativos; en FAISS es post-procesado.

## 4.2 Casos de Uso Recomendados

FAISS: colecciones pequeñas/medias con requisitos de máxima precisión. ChromaDB: colecciones grandes y necesidades de filtrado/ANN.

## 4.3 Consideraciones para Administraciones Públicas

Soberanía de datos (ejecución local), auditabilidad (artefactos JSON), mantenimiento y operativa (sin dependencias externas).

# 5. Conclusiones y Trabajo Futuro

## 5.1 Conclusiones Principales

El embedding elegido pesa más que el almacén vectorial: MiniLM requiere aumentar k para alcanzar 60% doc@k; E5 logra 40% en k=20 y 60% en k=40 en ambos stores. La latencia se mantiene en torno a 40–50 ms de mediana con E5.

## 5.2 Limitaciones del Estudio

Corpus específico y número limitado de consultas (n=11). No se variaron parámetros HNSW (ef\_search) ni se evaluó overlap top-k.

## 5.3 Trabajo Futuro

Explorar BAAI/bge-m3 y parámetros HNSW, medir overlap y throughput, e integrar reranking ligero basado en señales léxicas/título sin frameworks nuevos.

# Referencias

[1] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with GPUs,” IEEE Trans. Big Data, 2019. (FAISS)

[2] Y. A. Malkov and D. A. Yashunin, “Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs,” IEEE TPAMI, vol. 42, no. 4, pp. 824–836, 2020. (HNSW)

[3] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” in EMNLP, 2019. (Sentence-Transformers)

[4] ChromaDB Documentation, https://docs.trychroma.com/

[5] HuggingFace Model Cards: all-MiniLM-L6-v2, intfloat/multilingual-e5-base, BAAI/bge-m3, https://huggingface.co/