Meeting 11: Vector Database & Semantic Search

AI-40X: Generative AI & Large Language Models

Hendri Karisma, M.T.
Dosen Teknik Informatika STMIK Tazkia
VP Engineering at jejakin.com, 2026

Semester 4 — 2025/2026

Outline

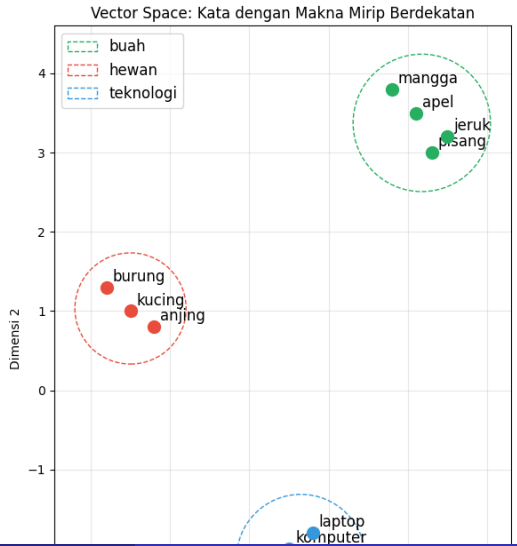
1. Vector Embeddings
2. Metrik Jarak & Similarity
3. Search Algorithms
4. Vector Database
5. Hybrid Search
6. Lab Preview
7. Kesimpulan & Referensi

Dari Kata ke Vektor: Evolusi Embedding

- Komputer tidak memahami makna kata — hanya angka.
- Kita perlu cara untuk merepresentasikan teks sebagai angka (vektor).
- **Evolusi model embedding:**
  1. **Word2Vec** (2013): Satu kata → satu vektor. Pelopor semantic embedding.
  2. **GloVe** (2014): Global co-occurrence statistics. Lebih stabil untuk kata langka.
  3. **Sentence-BERT** (2019): Satu kalimat → satu vektor. Cocok untuk semantic search.
  4. **Modern** (2023+): `text-embedding-3-small` (OpenAI), `all-MiniLM-L6-v2` (HuggingFace), `nomic-embed-text` (lokal, gratis).

Makna Semantik dalam Ruang Vektor

- Setiap kata/kalimat dipetakan ke titik di ruang berdimensi tinggi (768–3072 dimensi).
- **Sifat kunci:** Teks yang maknanya mirip $\rightarrow$ vektornya berdekatan.
- Contoh vektor (disederhanakan menjadi 4 dimensi):
  - "kucing" $\approx [0.8, 0.2, -0.1, 0.5]$
  - "anjing" $\approx [0.7, 0.3, -0.1, 0.4]$
  - "mobil" $\approx [-0.3, 0.9, 0.6, -0.2]$
- Perhatikan: "kucing" dan "anjing" mirip, jauh



Analogi Klasik: King – Man + Woman ≈ Queen

- Embedding menangkap **hubungan relasional** antar konsep:

$$\vec{v}(\text{King}) - \vec{v}(\text{Man}) + \vec{v}(\text{Woman}) \approx \vec{v}(\text{Queen})$$

- Artinya: selisih antara "King" dan "Man" ≈ selisih antara "Queen" dan "Woman".
- Contoh lain yang bekerja:
  - $\vec{v}(\text{Jakarta}) - \vec{v}(\text{Indonesia}) + \vec{v}(\text{Jepang}) \approx \vec{v}(\text{Tokyo})$
  - $\vec{v}(\text{terbang}) - \vec{v}(\text{burung}) + \vec{v}(\text{ikan}) \approx \vec{v}(\text{berenang})$
- **Implikasi:** Vektor bukan hanya "label angka" — vektor menangkap *struktur makna*.

Cosine Similarity

- Mengukur **sudut** antara dua vektor (mengabaikan panjang vektor).
- **Formula:**

$$\text{cosine\_sim}(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{||\vec{A}|| \times ||\vec{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

- **Rentang nilai:**
  - $+1$ = Arah sama (sangat mirip)
  - 0 = Tegak lurus (tidak berhubungan)
  - -1 = Berlawanan arah (bertolak belakang)
- **Paling populer** untuk text embeddings karena tidak terpengaruh panjang dokumen.

Euclidean Distance & Dot Product

Euclidean Distance (L2)

$$d(\vec{A}, \vec{B}) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

- Mengukur "jarak lurus" antar titik.
- Nilai 0 = identik; makin besar = makin jauh.
- Sensitif terhadap skala/magnitudo vektor.

Dot Product (Inner Product)

$$\vec{A} \cdot \vec{B} = \sum_{i=1}^{n} A_i B_i$$

- Gabungan arah + magnitudo.
- Lebih cepat komputasinya (tanpa pembagian/akar).
- Cocok jika vektor sudah **dinormalisasi** (panjang = 1).

Kapan Menggunakan Metrik Apa?

Metrik	Kapan Digunakan	Contoh
Cosine Similarity	Teks, embedding default	Semantic search
Euclidean (L2)	Ruang geometri, clustering	Deteksi anomali
Dot Product	Vektor ternormalisasi, kecepatan	Rekomendasi

Tips Praktis

Jika ragu, gunakan **Cosine Similarity**. Sebagian besar model embedding modern (OpenAI, Sentence-BERT) sudah dioptimasi untuk metrik ini. Jika vektor sudah dinormalisasi, cosine similarity $\equiv$ dot product.

Brute-Force: Akurat tapi Lambat

- **Cara paling sederhana:** Bandingkan query dengan *semua* vektor di database.
- Kompleksitas: $O(N \times D)$ — N = jumlah dokumen, D = dimensi vektor.
- Untuk $N = 100$ dokumen? Tidak masalah.
- Untuk $N = 10.000.000$ dokumen di 768 dimensi? **Sangat lambat.**
- Kita butuh algoritma yang lebih cerdas: **Approximate Nearest Neighbor (ANN)**.

HNSW & IVF: Pencarian Cepat

HNSW (Hierarchical Navigable Small World)

- Membangun graf berlapis seperti "jalan tol berjenjang".
- Layer atas: lompatan jauh (navigasi kasar).
- Layer bawah: lompatan dekat (pencarian presisi).
- **Sangat cepat**: $O(\log N)$.
- Digunakan oleh: Qdrant, ChromaDB, pgvector.

IVF (Inverted File Index)

- Membagi ruang vektor menjadi *cluster* (Voronoi cells).
- Saat search: hanya periksa cluster terdekat, bukan semua data.
- Lebih hemat memori dari HNSW.
- Digunakan oleh: FAISS.

Trade-off Utama

ANN mengorbankan sedikit akurasi untuk kecepatan drastis. Biasanya recall 95–99% sudah cukup baik untuk aplikasi nyata.

k-NN vs ANN: Perbandingan

	Exact k-NN	**ANN (HNSW/IVF)**
Akurasi	100% (exact)	95–99% (approximate)
Kecepatan (N=1jt)	Detik–menit	Milidetik
Kompleksitas	$O(N \times D)$	$O(\log N)$
Cocok untuk	Dataset kecil, evaluasi	Produksi, real-time

- **k-NN** = cari k tetangga terdekat secara pasti (brute-force).
- **ANN** = cari k tetangga terdekat secara *approximate* (cepat).
- Di dunia nyata, hampir selalu gunakan **ANN**.

Mengapa Butuh Database Khusus?

- **Database tradisional** (PostgreSQL, MySQL) dirancang untuk query SQL: `WHERE nama = 'Budi'`.
- **Vector Database** dirancang untuk query: "Cari 5 dokumen paling mirip dengan vektor ini."
- Fitur yang disediakan Vector DB:
  1. Penyimpanan vektor berdimensi tinggi secara efisien
  2. Indexing otomatis (HNSW/IVF) untuk pencarian cepat
  3. Metadata filtering (filter berdasarkan kategori, tanggal, dll.)
  4. CRUD operations (Create, Read, Update, Delete) untuk vektor
- *Analogi:* Anda *bisa* menyimpan gambar di spreadsheet Excel, tapi lebih baik pakai Google Photos.

ChromaDB: Ringan & Python-Native

- **Kelebihan:** Install cukup `pip install chromadb`. Tidak perlu Docker atau server terpisah!
- Ideal untuk belajar, prototyping, dan dataset kecil–menengah (<1 juta vektor).
- Data disimpan di file lokal (SQLite + Parquet di belakang layar).

Contoh ChromaDB

```
import chromadb

client = chromadb.Client()  # In-memory, tanpa setup
collection = client.create_collection("dokumen_saya")

collection.add(
    documents=["Kucing suka ikan", "Anjing suka tulang", "Mobil butuh bensin"],
    ids=["doc1", "doc2", "doc3"]
)

results = collection.query(query_texts=["hewan peliharaan"], n_results=2)
print(results["documents"])
# [['Kucing suka ikan', 'Anjing suka tulang']]  <-- Semantik!
```

FAISS: Cepat untuk Skala Besar

- Dikembangkan oleh **Meta AI (Facebook Research)**.
- Bukan database lengkap — lebih tepat disebut *similarity search library*.
- **Kelebihan:** Sangat cepat, mendukung GPU, bisa menangani miliaran vektor.
- **Kekurangan:** Tidak ada metadata filtering bawaan, perlu kode tambahan.
- Install: `pip install faiss-cpu` (atau `faiss-gpu`).

Contoh FAISS

```
import faiss
import numpy as np

d = 384  # dimensi vektor
vectors = np.random.rand(1000, d).astype('float32')

index = faiss.IndexFlatL2(d)      # Brute-force L2
index.add(vectors)                # Tambahkan 1000 vektor

query = np.random.rand(1, d).astype('float32')
distances, indices = index.search(query, k=5)  # Cari 5 terdekat
```

Perbandingan Vector Database

Fitur	ChromaDB	FAISS	Qdrant	Pinecone
Tipe	Database	Library	Database	Cloud SaaS
Install	`pip install`	`pip install`	Docker	Tidak perlu
Metadata Filter	Ya	Tidak	Ya	Ya
Skala	Kecil–Menengah	Besar (miliaran)	Besar	Besar
Harga	Gratis	Gratis	Gratis/Paid	Berbayar
GPU Support	Tidak	Ya	Tidak	N/A
Cocok untuk	Belajar, RAG	Riset, skala	Produksi	Enterprise

Untuk Kuliah Ini

Kita akan menggunakan **ChromaDB** karena: (1) tidak perlu Docker, (2) cukup `pip install`, (3) API sederhana, (4) gratis. Cocok untuk laptop mahasiswa!

BM25: Pencarian Keyword Klasik

- **BM25** (Best Match 25) adalah algoritma ranking untuk pencarian berbasis kata kunci.
- Prinsip: Kata yang *jarang* muncul di semua dokumen tapi *sering* di satu dokumen → dokumen itu relevan.
- **Kelebihan:** Sangat baik untuk exact match (nama, kode, istilah teknis).
- **Kekurangan:** Tidak mengerti sinonim.
  - Query: "cara mengobati demam" → tidak menemukan dokumen berisi "penanganan suhu tubuh tinggi"
- Digunakan oleh: Elasticsearch, Lucene, Google (sebagian).

Hybrid Search: Gabungkan Keduanya!

Keyword Search (BM25)

- Cocok: nama orang, kode produk, istilah spesifik
- "paracetamol 500mg" → exact match

Vector Search (Semantic)

- Cocok: pertanyaan umum, sinonim, konteks
- "obat penurun panas" → paracetamol

Hybrid Search

- Jalankan **keduanya** secara paralel.
- Gabungkan hasilnya.
- Teknik penggabungan: **Reciprocal Rank Fusion (RRF)**.

$$\text{RRF}(d) = \sum_{r \in R} \frac{1}{k + \text{rank}_r(d)}$$

k = konstanta (biasanya 60), R = set hasil dari masing-masing pencarian.

Kapan Menggunakan Hybrid vs Pure Semantic?

Situasi	**Rekomendasi**
Dokumen teknis, kode, ID produk	Hybrid Search
FAQ, customer support	Pure Semantic
Dataset multibahasa	Pure Semantic
Dokumen hukum, regulasi	Hybrid Search
Pencarian umum (catch-all)	Hybrid Search

Aturan Praktis

Jika ragu, gunakan **Hybrid Search**. Dalam benchmark, hybrid search hampir selalu mengungguli pure semantic atau pure keyword saja.

Lab: Semantic Search di atas Dokumen PDF

Yang akan kita bangun:

1. Membaca dokumen PDF menggunakan `PyMuPDF` / `pdfplumber`
2. Memecah teks menjadi chunk (potongan) berukuran 512 token
3. Mengubah setiap chunk menjadi vektor menggunakan Sentence-BERT
4. Menyimpan vektor ke ChromaDB
5. Menerima pertanyaan user, mengubahnya jadi vektor
6. Mencari 5 chunk paling relevan (nearest neighbor)

Setup Environment

```
pip install chromadb sentence-transformers pymupdf
```

Tidak Perlu GPU!

Model `all-MiniLM-L6-v2` cukup kecil (80MB) dan berjalan lancar di CPU laptop.

Kesimpulan

1. **Embedding** mengubah teks menjadi vektor yang menangkap makna semantik.
2. **Cosine Similarity** adalah metrik paling umum untuk mengukur kemiripan teks.
3. **HNSW** dan **IVF** memungkinkan pencarian tetangga terdekat secara cepat ($O(\log N)$).
4. **ChromaDB** adalah pilihan ideal untuk belajar dan prototyping — cukup `pip install`.
5. **Hybrid Search** (keyword + semantic) memberikan hasil terbaik di sebagian besar kasus.
6. **Minggu depan:** Kita akan menggunakan vector DB ini sebagai fondasi untuk membangun sistem **RAG (Retrieval-Augmented Generation)**.

Referensi

- Mikolov, T., et al. (2013). “Efficient Estimation of Word Representations in Vector Space” (Word2Vec).
- Pennington, J., et al. (2014). “GloVe: Global Vectors for Word Representation.”
- Reimers, N. & Gurevych, I. (2019). “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.”
- Malkov, Y. & Yashunin, D. (2020). “Efficient and Robust Approximate Nearest Neighbor using Hierarchical Navigable Small World Graphs” (HNSW).
- Johnson, J., et al. (2019). “Billion-scale Similarity Search with GPUs” (FAISS).
- ChromaDB Documentation: `https://docs.trychroma.com`