Vector databases + AI

Problems with classical search in DB

Keyword / full text search (traditional)

- matches exact words / word parts
- fails with:
  - synonyms ("car" vs "automobile")
  - paraphrasing
  - different languages

Example:

- Query: "How to learn programming„
- Document: "Best way to study coding„
  - Keyword search returns: **no match**

AI search - embeddings

What is an embedding?

- a vector representing meaning of a text
- Similar meanings → vectors close together

How embeddings are created?

- An **AI model** converts text → vector
- Same model is used for:
  - documents
  - search queries
- AI returns vector with dimension somewhere around 384, 768 or 1536 -> Numerical representation of meaning

Vector similarity

- To compare two texts:
  - Convert both to embeddings
  - Compute **similarity**

Most common metric:

  - Cosine similarity or cosine distance (i.e. 1 – cosine similarity, sometimes normalized to 0-1)

Cosine similarity returns:

- 1.0 → identical meaning
- 0.0 → unrelated
- -1.0 → opposite meaning

Vector database

- To implement smart AI search, we need:
  1. create embedding vector for each document
     - store the vector alongside the document in the database
  2. Create embedding vector for a user‘s search query
  3. Search for the nearest embedding vectors in the database

- The database must be able to:
  - Store vectors
  - Compute cosine similarity / cosine distance of vectors
  - Quickly search for the nearest vectors (vector index)

AI search steps

- Create embeddings for documents
- Store them
- Embed user query
- Find most similar vectors
- Return matching documents

- Tools that handle vector databases:
  - E.g. FAISS, PG_VECTOR, MariaDB, Qdrant, ...

Vector indexes

- DB must be able to quickly give answer to
  "Which vectors are closest to my query vector?"
- If you have:
  - 100 vectors → easy (just compare all)
  - 1,000,000 vectors → too slow
- Analogy: finding nearest cafés
  - Each café = a point on a map
  - Query = "my current location"
- **Option A: Check all cafés**
  - Accurate
  - Slow if city is huge

Option B: City map with districts

- First find nearby districts
- Then search cafés only there
- This is what vector indexes do.

Exact vs Approximate search

- **Exact search**
  - Checks all vectors
  - Always finds true nearest neighbors
  - Slow for large datasets
- **Approximate Nearest Neighbor (ANN)**
  - Checks only *promising* candidates
  - 99% correct
  - 100–1000× faster

  - Tree-based indexes,
  - Hash-based indexes (LSH),
  - **Graph-based indexes (HNSW – Hierarchical Navigable Small World**)

HNSW – Hierarchical Navigable Small World

- **Idea**
  - Vectors are nodes in a graph
  - Each node connects to a few nearest neighbors
  - Graph has multiple layers
- **Layers**
  - Top layers: few nodes, long jumps
  - Bottom layer: all nodes, fine detail

Search process (HNSW)

- Start at the top layer
- Jump closer to query vector
- Go down layer by layer
- At bottom, refine search
- Return top K neighbors