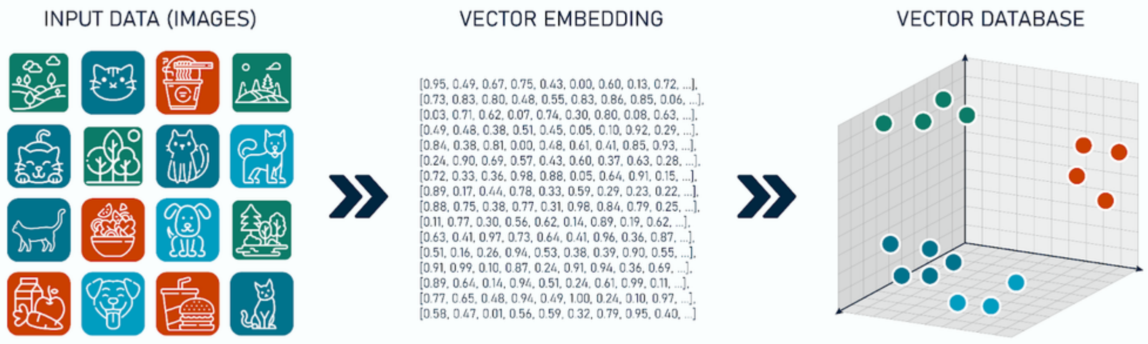
What is a vector database and vector embedding?

A **vector database** is a specialized type of database that stores, manages, and queries mathematical representations of unstructured data – texts, images, audio, etc. These representations are called *vector embeddings*.



Vector database containing image embeddings

A **vector embedding** is a sequence of numbers like [0.4, 0.8, -0.1, 0.6, 1.1, ...] that captures the original meaning of a data point (a sentence, an image, an audio signal, etc.) in relation to other points.

Think of each number in the sequence as a coordinate in a separate dimension that represents some aspect or feature of the original data. For example, in movie descriptions, dimensions can represent the genre, the release year, the rating, and more.

The number of dimensions depends on the complexity of the data. Basic geolocation might require only a few dimensions, while high-resolution images or large texts need thousands of dimensions to represent all its features.

An embedding doesn't have standalone significance. What matters is how it aligns with other vectors in the multidimensional space. For instance, the embedding for the movie *Tenet* would be close to *Inception*, as they share similarities – themes of science fiction and mind-bending plots.

Vector embedding types

Vector embeddings are generated from unstructured data using various machine learning techniques. Here are some examples.

Word embedding. A vector might represent a word or sentence. Words with similar meanings (like *polite* and *nice*) will have vectors that are close to each other in the database space.
Techniques: Word2Vec, TF-IDF, FastText, Global Vectors for Word Representation (GLoVe).

Image embedding. A vector can capture visual features like color, texture, or shapes in an image. For instance, pictures of cats will have similar vectors because they share common features—fur, paws, and whiskers.
Techniques: convolutional neural network (CNN), Vision Transformer (ViT), color histograms, bag of visual words (BoVW).

Audio embedding. A vector might represent patterns in sound, such as pitch, tone, or rhythm.
Techniques: Recurrent Neural Networks (RNNs), Contrastive Language-Audio Pretraining (CLAP), Pretrained Audio Neural Networks (PANNs).

Video embedding. A vector can combine features from both images (frames) and sound to represent motion or scenes in a video.

Vector database use cases

The key feature of a vector database is its ability to quickly find and retrieve data based on the similarity of the vectors. This makes it especially useful for applications that rely on searches based on meaning or context.

Recommendation engines. Vector databases store user profiles (preferences, past behaviour, purchases, etc.) and product features as vectors, enabling personalized recommendations.

When a customer makes a purchase, the recommender system may suggest items with similar characteristics or items previously chosen by users with similar profiles.

Fraud detection. When thinking of vector databases, the first application that comes to mind is searching for similarities. However, they also help spot outliers (or anomalies). This capability is particularly important in fields like finance and cybersecurity when you need to detect deviations that indicate something unusual.

A vector database makes it simpler to flag suspicious transactions, monitor unusual network activity, or spot irregular behaviours in real time, ensuring that potential risks are identified and addressed as soon as possible.

Retrieval-augmented generation (RAG). RAG is a means of enhancing the performance and accuracy of LLMs by linking them to data sources relevant to a particular use case. Conversational AI, using RAG, produces more precise outputs. Think of a corporate virtual assistant based on enterprise documents or a technical support chatbot. Vector databases play a crucial role in supporting RAG due to their ability to effectively store and quickly retrieve textual information.

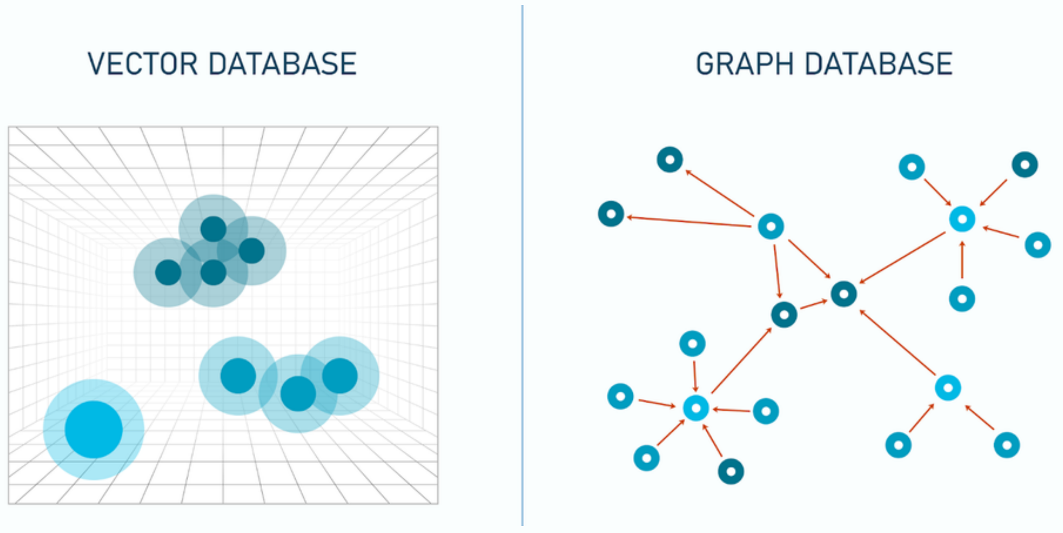
Computer vision. In facial recognition, images of faces are transformed into vectors, and the database can then match a new face to stored images based on similarity. Other examples include object detection, image search, medical scans, and surveillance footage

Vector database vs. graph database

Vector databases and graph databases can get confused because they

- handle nontabular data;
- work with complex, interconnected data;
- are used in AI applications; and
- may look conceptually similar.

Both databases manage connections in data, but they do so using different structures and techniques suited for different purposes.



Vector database vs. graph database

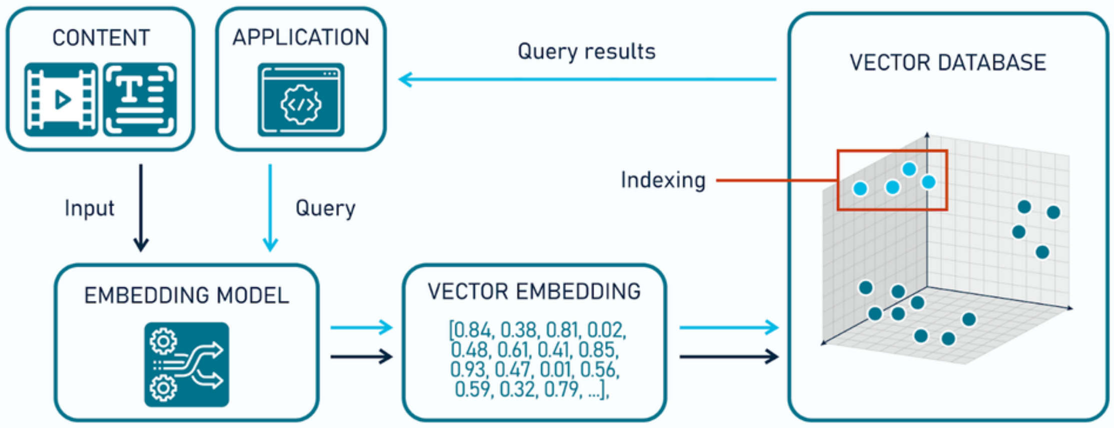
Vector databases arrange data in a multidimensional space based on similarities: Data points with much in common will be relatively close.

Graphs store data as nodes (entities) and edges (relationships) between them. They enable efficient traversal and querying of complex relationships, making them ideal for applications where connections and interactions between entities are central.

Both vector and graph databases are powerful tools for recommendation engines, but the former would be better for movie recommendations as they capture attributes like genre, cast, and plot. If a user watches a comedy, the system can offer other comedies simply relying on this genre.
Graph databases, on the other hand, would be more suited for social recommendations, where connections and networks (e.g., a friend of a friend) play a crucial role in suggesting relevant accounts.

How a vector database works

Let's break down an example of a recommendation workflow on a movie streaming service.



How a vector database works

Input. Say the user watches *Inception*, a sci-fi movie directed by Christopher Nolan. The system tracks their interaction and updates their preferences. It prepares the input data and groups relevant attributes (e.g., movie genres, cast, crew, director, etc.).

Embedding. An embedding model generates a vector embedding for *Inception* and combines it with the user's previous history. **Storage.** The generated embeddings are now saved in a vector database.

Query. When a streaming platform looks for movies to recommend, the system converts their query into a vector using the same embedding model. The system compares the query vector with other movie vectors to find and return the most similar results.

Indexing. To speed up querying by proximity, the vector database uses indexing that narrows down the search space to a subset of nearby vectors. This ensures low latency even with millions of embeddings.

Output. The nearest vectors retrieved are ranked based on their similarity scores, ensuring the most relevant movies appear first. As a result, movies like *Interstellar* and *Tenet* are

ranked and shown as recommendations, with a focus on the sci-fi genre and pictures directed by Christopher Nolan.
Now that you’ve grasped the overall concept of the way a vector database works, let’s delve into its primary functions.

Storing vectors

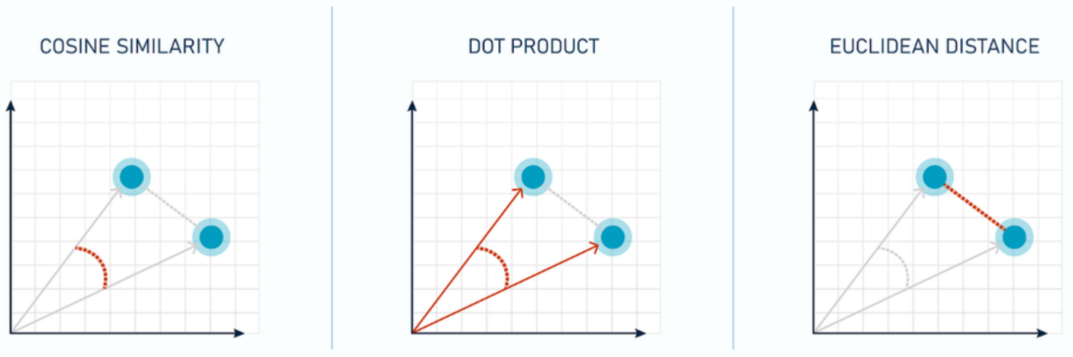
The storage function of a vector database is pretty simple and self-explanatory. Databases keep vector embeddings and each vector’s metadata (information that provides more context to the data represented by the vector).

Metadata filters are used to narrow down search results by specifying certain criteria. For example, if you only want to search for vectors created after a particular date or that belong to a particular category, you apply metadata filters to exclude vectors that don't meet these conditions.

Similarity search

When you ask an AI chatbot a question, the system converts that search or prompt into a vector, known as the "query vector."

The database compares the query vector to the vectors it stores and calculates the "distance" between them using different metrics. The closer the vectors are, the more they have in common. Here are the three most popular approaches to the task.



Cosine similarity vs. Dot product vs. Euclidean distance

Cosine similarity measures how similar two data points are in terms of their direction. Instead of focusing on the distance between them, it checks their alignment. Think of it as comparing the angle between two arrows pointing out from the same origin.

This is particularly useful for text analysis, where you're comparing the meaning of documents or words rather than their size. For instance, two sentences with similar topics but different word counts can still have high cosine similarity.

Dot product incorporates not only the alignment of vectors but also their magnitude. It is useful when you want to account for both alignment and magnitude. For example, in a recommendation system, you may want to consider both the similarity of preferences and the strength of those preferences.

This approach is used with non-normalized vectors.

Normalization means scaling a vector so that magnitude is 1. When vectors aren't normalized, they can have varying magnitudes that influence calculations. Cosine similarity normalizes the vectors, which eliminates the effect of magnitude. That's why it only considers the alignment of vectors.

Euclidean distance focuses on how far the vectors are from each other. It is like measuring the shortest path between two points. Imagine standing at one corner of a room and walking diagonally to the opposite corner—that's the Euclidean distance. It's great for situations where the absolute difference between data points matters, such as comparing two images based on pixel values.

After calculating the distances, the system ranks all the stored vectors based on their similarity to the query vector. It then returns the top results (the vectors with the smallest distance or highest alignment) to the user.

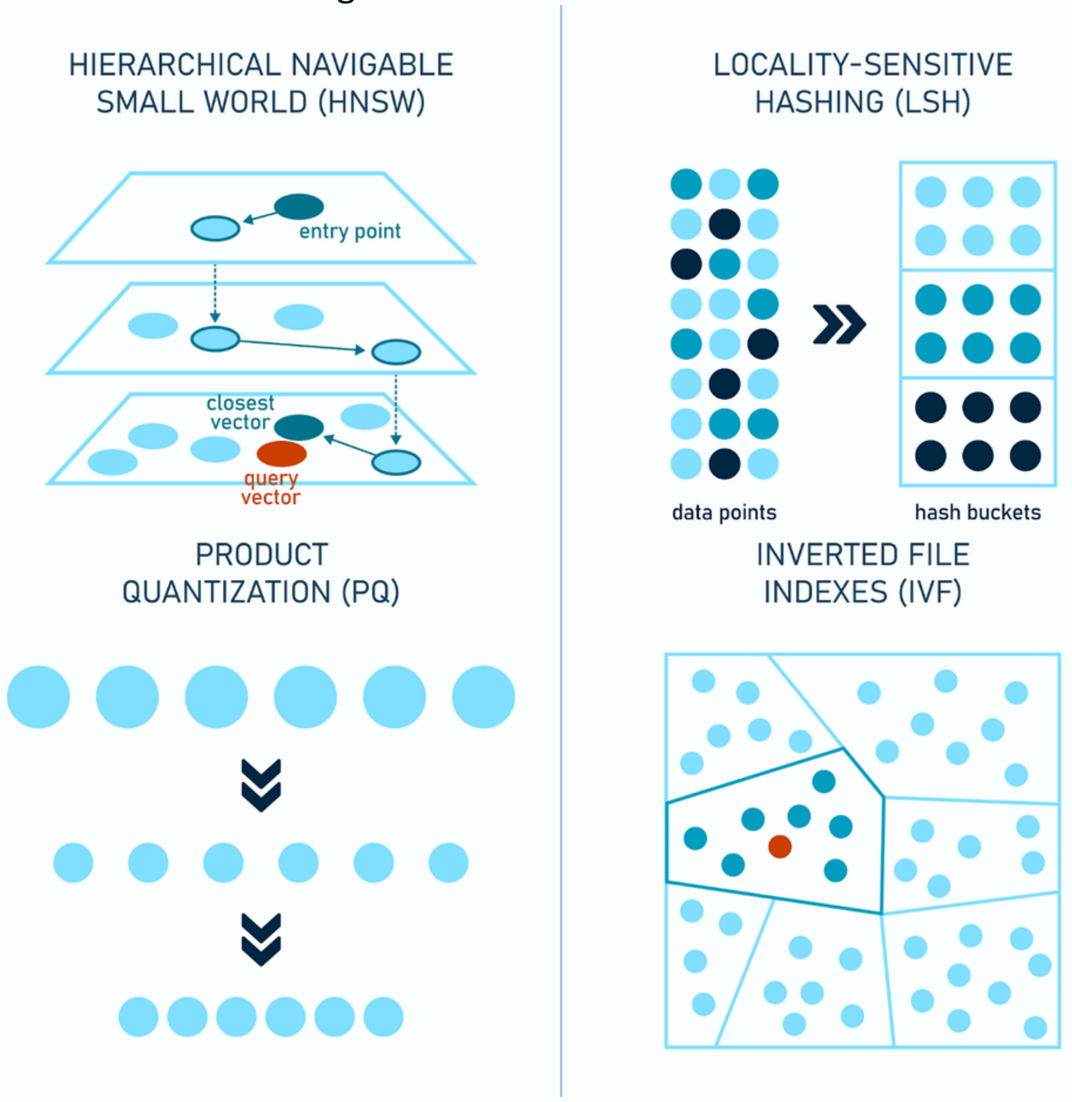
However, comparing a query vector with every vector stored in a database is time-consuming and computationally expensive, especially when dealing with large datasets or high-dimensional data. To address this, vector databases use indexing and an Approximate Nearest Neighbor (ANN) search.

ANN search and indexing

Approximate Nearest Neighbour (ANN) is a technique used to efficiently find points in a dataset that are close to a given query point in terms of a specific distance metric (e.g., Euclidean distance or cosine similarity). It narrows down the search space to a smaller subset of potential matches.

Unlike nearest neighbour search, which exhaustively compares the query to all points in the dataset, ANN trades some accuracy for significant improvements in speed, making it practical for large-scale and high-dimensional data.

Indexing is the method of organizing the data for efficient searching. Various approaches, each with its own strengths, can be used for indexing.


HIERARCHICAL NAVIGABLE SMALL WORLD (HNSW)
entry point
closest vector
query vector
LOCALITY-SENSITIVE HASHING (LSH)
data points
hash buckets
PRODUCT QUANTIZATION (PQ)
INVERTED FILE INDEXES (IVF)

Indexing methods

A **hierarchical navigable small world (HNSW)** uses a multilayered graph structure for an ANN search. The top contains fewer vectors, while the lower layers have more vectors and provide finer detail.

The search starts at the top and moves downward until it reaches the bottom layer. This setup allows for quick navigation and finding vectors similar to a query.

Locality-sensitive hashing (LSH) is an ANN search technique that uses *hash functions*. They are mathematical functions that transform high-dimensional data points into lower-dimensional representations called *buckets*. Points that are close together in the original space are more likely to be hashed into the same bucket.

LSH uses multiple hash tables, each with its own hash function, so that similar points can still be found even if they fall into different buckets in one table. The algorithm hashes the query point, identifies relevant buckets across all tables, and only compares the query to points in those buckets.

The **inverted file indexes (IVF)** is a technique used to speed up similarity searches in large datasets by clustering vectors into smaller groups (cells). Instead of comparing a query vector to every single vector in the entire dataset, IVF reduces the search scope by only comparing the query vector to vectors within the same cluster (or nearby clusters).

Product quantization (PQ) focuses on reducing memory usage and can be combined with other indexing methods. It compresses the data by converting each vector into a shorter, more compact representation. This approach saves memory and allows for faster searches.