**TODO**: Replace this documentation with blog link to medium articles

# Vector Databases

**What are vector embeddings and why do they matter?**

In short:

Vector Embeddings are arrays of numbers that codify real world objects and concepts. They matter because they are convenient ways to help computers grasp the difference and similarities between real world objects (and concepts) – for instance vector embeddings can help codify the difference and similarity between dogs and cats.

[Optional]Long form description:

* What are they? Vector embeddings are essentially numerical representations of data. Each piece of data (like a word or an image) is mapped to a point in a high-dimensional space. These points are like addresses, and their positions capture the semantic relationships between the data points.
* Why do they matter? This numerical encoding brings numerous benefits:
  + Machine-friendly information: Numbers are much easier for computers to work with than raw text or images. Embeddings allow machines to compare and process data efficiently, paving the way for complex tasks like:
    - Semantic search: Finding things based on meaning, not just keywords. Imagine searching for "funny cat videos" and getting results that capture the humor, even if the exact words aren't present.
    - Recommendation systems: Suggesting things you might like based on your past preferences. Embeddings help these systems understand your tastes and recommend similar items, like suggesting books similar to ones you previously enjoyed.
    - Machine translation: Understanding the nuances of language and translating meaning across languages. Embeddings allow translation systems to go beyond literal word-for-word conversions and convey the true intent.

**What are vector databases and why do they matter?**

In short:

Production ML systems need huge corpus of embeddings (order of billions) for accurate training and sometimes inference. Traditional data stores are not optimized for efficient storage and retrieval of embeddings. Vector Databases are custom built for this purpose - to support efficient storage, retrieval and also operations (like similarity search) on embeddings, so production ML models can use embeddings at scale.

[Optional]Long form description:

Vector databases are important for several reasons:

**1. Efficient storage and retrieval of vector embeddings**: Traditional databases struggle with storing and retrieving vector data, specifically high-dimensional vectors, effectively. Vector databases are optimized for this purpose, enabling efficient indexing, searching, and querying based on similarity.

**2. Powerful similarity search**: Traditional keyword-based searches often miss the mark when dealing with data like text, images, or music. Vector databases excel at finding data points similar to a given query based on their underlying vector representations. This is crucial for applications like:

Recommendation systems: Suggesting products, movies, or music similar to what users have enjoyed.

Image and video search: Finding visually similar images or videos based on content, not just captions.

Natural language processing: Understanding the meaning of text beyond keywords, enabling tasks like chatbots and question answering.

**3. Scalability and performance:** As the volume and dimensionality of vector data grow, traditional databases can become overwhelmed. Vector databases are designed to handle large datasets efficiently, maintaining fast search and retrieval times even with millions of data points.

**4. Flexibility and integration:** Vector databases often offer diverse APIs and SDKs, making them easy to integrate with various applications and workflows. This flexibility allows developers to build powerful applications built around vector search and similarity analysis.

**5. Operationalization of AI models:** Many AI models create vector embeddings as outputs. Vector databases provide a robust and organized way to store and manage these embeddings, making them easily accessible for downstream applications.

Overall, vector databases are becoming increasingly important as the use of vector data and AI continues to grow. They provide a specialized platform for optimizing the storage, retrieval, and utilization of vector information, powering innovative applications across various industries.

**What is the fundamental operation supported by Vector DBs?**

**Approximate Nearest Neighbor (ANN) Search**: Given a query vector ‘q’, identify and return K vectors that are closest to ‘q’.

Closeness/proximity can be determined through several distance measures (Cosine and Euclidean similarities being popular measures)

**Why *approximate* search**? Because exact matches are not always available and also users may prefer some variety in their query results (For instance – If I have already watched Naruto, I now want to watch animations like Naruto but not exactly Naruto!)

**How is ANN implemented?**

ANN algorithms are a fast-evolving space but there are 3 predominant family of methods at present:

**1. Quantization methods**: These methods divide the data space into smaller regions (cells) and assign each data point to a cell by quantizing its vector. Searching for nearest neighbors then involves exploring only the cells close to the query point. Popular examples include **LSH (Locality-Sensitive Hashing)**: Projects data points into multiple random low-dimensional spaces, where points close in the original space collide with high probability.

**2. Space-partitioning methods**: These methods divide the data space into regions using various geometric structures like k-d trees or ball trees. Searching for nearest neighbors involves traversing these structures efficiently, focusing on potentially relevant regions. Examples include: **k-d trees** Divide the space by hyperplanes perpendicular to dimensions, recursively creating axis-aligned partitions and **Ball trees** Partition the space with hyperspheres, where each sphere contains a data point and all its neighbors within a given radius.

**3. Graph-based methods**: These methods represent the data points as a graph, where edges connect similar points. Searching for nearest neighbors involves exploring this graph, iteratively visiting nodes likely to contain closer neighbors. Examples include **Navigating Small Worlds on Graphs** Explores the local structure of the graph by randomly jumping between neighbors, focusing on dense regions with multiple close points.

Choosing the right ANN algorithm depends on several factors:

* Dataset size and dimensionality: Some algorithms scale better than others with increasing data size or dimensionality.
* Accuracy vs. speed trade-off: Different algorithms offer varying levels of accuracy and efficiency.
* Distance metric: Some algorithms work best with specific distance metrics (e.g., Euclidean distance).

**Real talk – do we really need vector databases?**

Almost everything Vector DBs do can in theory (and in practice) be done by existing range of databases, with some augmentations of course.

Such attempts at augmenting existing operational databases to support vector search have also kickstarted and a trend to watch out for (I have no opinions at present as to whether this trend will become mainstream):

https://stackoverflow.blog/2023/09/20/do-you-need-a-specialized-vector-database-to-implement-vector-search-well/

Rather, I will let the market and the industry decide how relevant Vector DBs are going to be in the long term. Meanwhile, I will focus on feeding the nerd in me, learning and sharing more about this fascinating intersection of algorithms, graphs, ML and database theory which is Vector Databases.

Follow me on Twitter: [@SrishiC](https://twitter.com/SrishiC) and LinkedIn: [Saptarshi](https://www.linkedin.com/in/saptarshi-chaudhuri-70430a5b/)