# What is a Vector Database?

Milvus · Follow
Published in Vector Database for AI
10 min read · Jan 13, 2022



> *An introduction to the concepts related to vector database.*

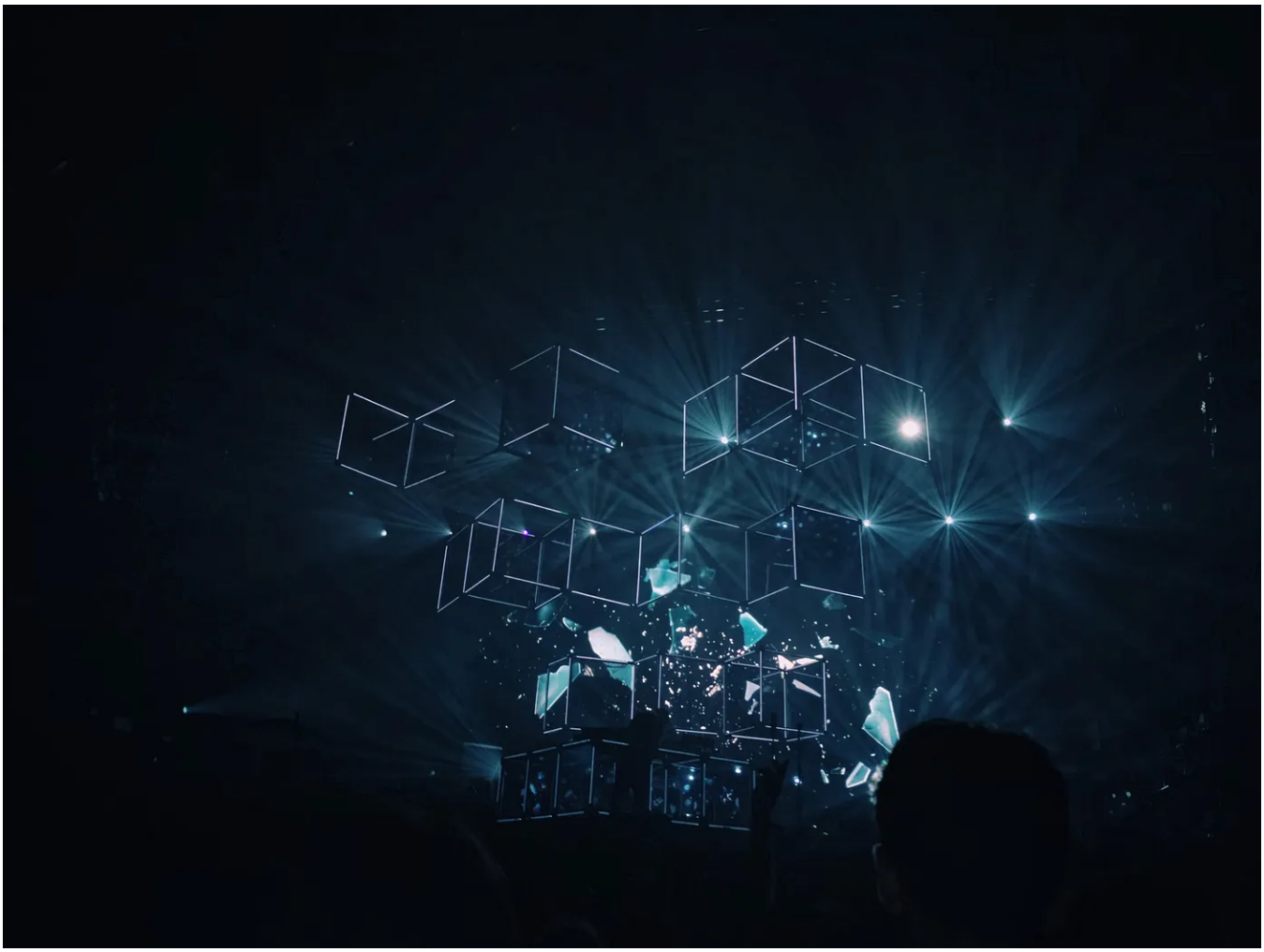


Photo by fabio on Unsplash

*By Frank Liu (@frankzliu)*

*This article has been updated since it was first published and is now part of a blog series that serves as a coursebook for concepts and guides related to vector databases.*

Jump to:



In the previous tutorial, we took a quick look at the ever-increasing amount of data that is being generated on a daily basis. We then covered how these bits of data can be split into structured/semi-structured and unstructured data, the differences between them, and how modern machine learning can be used to understand unstructured data through embeddings. Finally, we briefly touched upon unstructured data processing via ANN search.

Through all of this information, it's now clear that the ever-increasing amount of unstructured data requires a paradigm shift and a new category of database management system — the vector database.

> A vector database is a fully managed, no-frills solution for storing, indexing, and searching across a massive dataset of unstructured data that leverages the power of embeddings from machine learning models.

## Vector databases from 1000 feet

Guess how many curators it took to label the now-famous ImageNet dataset. Ready for the answer?

25000 people (that's a lot).

Being able to search across images, video, text, audio, and other forms of unstructured data via their content rather than human-generated labels or tags is exactly what vector databases were meant to solve. When combined with powerful

machine learning models, vector databases such as Milvus have the ability to revolutionize e-commerce solutions, recommendation systems, computer security, pharmaceuticals, and many other industries.

As mentioned in the introduction, **a vector database is a fully managed, no-frills solution for storing, indexing, and searching across a massive dataset of unstructured data that leverages the power of embeddings from machine learning models.**

But let's think about it from a user perspective. What good is a piece of technology without strong usability and a good user API? In concert with the underlying technology, multi-tenancy and usability are also incredibly important attributes for vector databases. Let's list out all of the features a mature vector database should have (many of these features overlap with those of databases for structured/semi-structured data):

- **Scalability and tunability**: As the number of unstructured data elements stored in a vector database grows into the hundreds of millions or billions, horizontal scaling across multiple nodes becomes paramount (scaling up by manually inserting sticks of RAM into a server rack every 3 months is no fun). Furthermore, differences in insert rate, query rate, and underlying hardware may result in different application needs, making overall system tunability a mandatory feature for vector databases.
- **Multi-tenancy and data isolation**: Supporting multiple users is an obvious feature for all database systems. However, going guns blazing and creating a new vector database for every new user will probably turn out poorly for everyone. Parallel to this notion is data isolation — the idea that any inserts, deletes, or queries made to one collection in a database should be invisible to the rest of the system unless the collection owner explicitly wishes to share the information.
- **A complete suite of APIs**: A database without a full suite of APIs and SDKs is, frankly speaking, not a real database. For example, Milvus maintains Python, Node, Go, and Java SDKs for communicating with and administering a Milvus database.
- **An intuitive user interface/administrative console**: User interfaces can help significantly reduce the learning curve associated with vector databases. These

interfaces also expose new vector database features and tools that would otherwise be inaccessible.

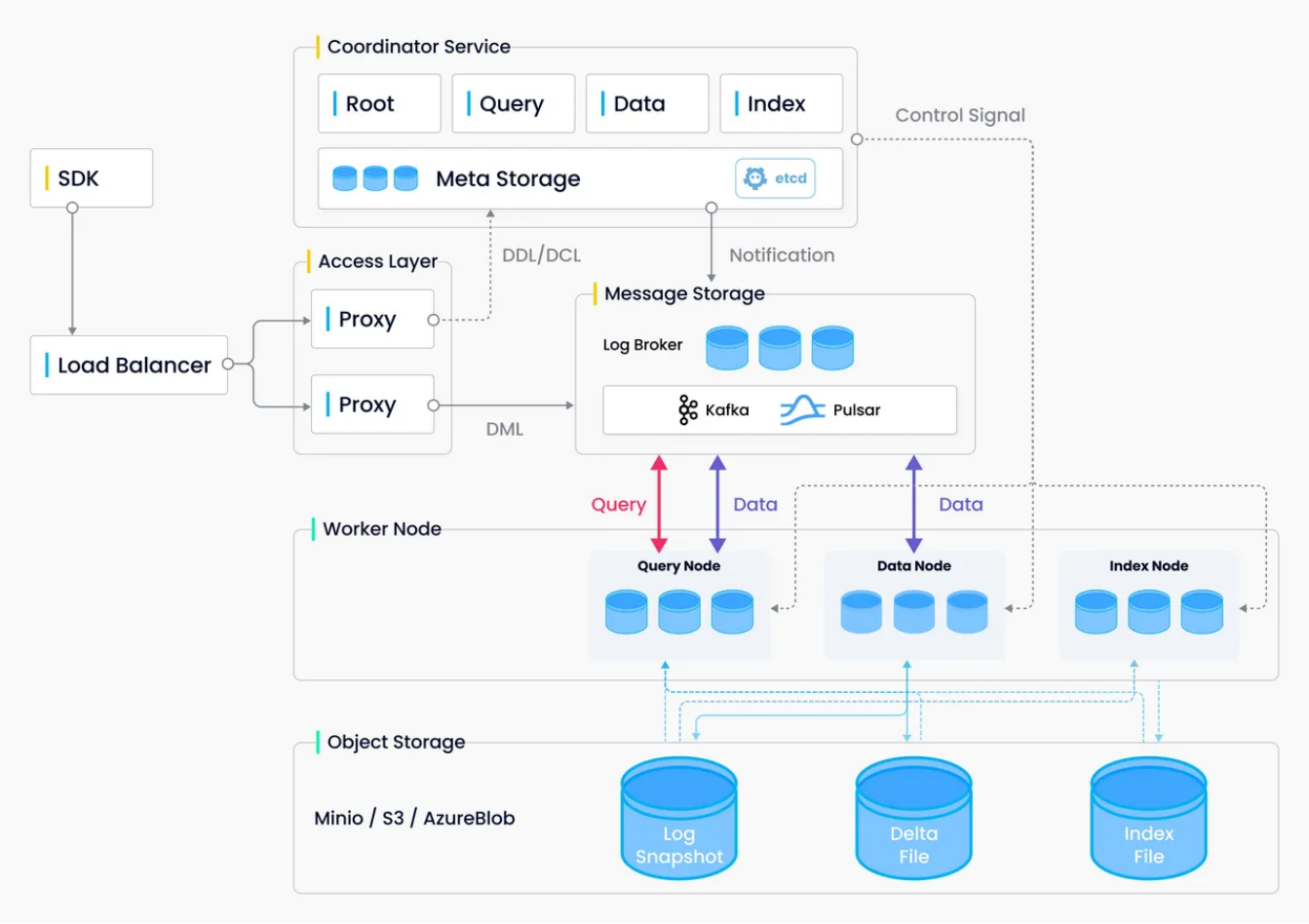
Phew. That was quite a bit of info, so we'll summarize it right here: a vector database should have the following features: 1) scalability and tunability, 2) multi-tenancy and data isolation, 3) a complete suite of APIs, and 4) an intuitive user interface/administrative console. In the next two sections, we'll follow up on this concept by comparing vector databases versus vector search libraries and vector search plugins, respectively.

## Vector databases vs. vector search libraries

A common misconception that I hear around the industry is that vector databases are merely wrappers around ANN search algorithms. This could not be further from the truth!

**A vector database is, at its core, a full-fledged solution for unstructured data.** As we've already seen in the previous section, this means that user-friendly features present in today's database management systems for structured/semi-structured data — cloud-nativity, multi-tenancy, scalability, etc — should also be attributes for a mature vector database as well. All of these features will become clear as we dive deeper into this tutorial.

On the other hand, projects such as FAISS, ScaNN, and HNSW are lightweight ANN libraries rather than managed solutions. The intention of these libraries is to aid in the construction of vector indices — data structures designed to significantly speed up nearest neighbor search for multi-dimensional vectors[1]. If your dataset is small and limited, these libraries can prove to be sufficient for unstructured data processing, even for systems running in production. However, as dataset sizes increase and more users are onboarded, the problem of scale becomes increasingly difficult to solve.




High-level overview of Milvus's architecture. It looks confusing, I know, but don't worry, we'll dive into each component in the next tutorial.

Vector databases also operate in a totally different layer of abstraction from vector search libraries — vector databases are full-fledged services, while ANN libraries are meant to be integrated into the application that you're developing. In this sense, ANN libraries are one of the many components that vector databases are built on top of, similar to how Elasticsearch is built on top of Apache Lucene.

To give an example of why this abstraction is so important, let's take a look at inserting a new unstructured data element into a vector database. This is super easy in Milvus:

```
from pymilvus import Collection
collection = Collection('book')
mr = collection.insert(data)
```

It's really as easy as that — 3 lines of code. With a library such as FAISS or ScaNN, there is unfortunately no easy way of doing this without manually re-creating the

entire index at certain checkpoints. Even if you could, vector search libraries still lack scalability and multi-tenancy, two of the most important vector database features.

### Vector search plugins for traditional databases

Great, now that we've established the difference between vector search libraries and vector databases, let's take a look at how vector databases differ from vector search plugins.

An increasing number of traditional databases and search systems such as Clickhouse and Elasticsearch are including built-in vector search plugins. Elasticsearch 8.0, for example, includes vector insertion and ANN search functionality that can be called via restful API endpoints. The problem with vector search plugins should be clear as night and day — **these solutions do not take a full-stack approach to embedding management and vector search.** Instead, these plugins are meant to be enhancements on top of existing architectures, thereby making them limited and unoptimized. Developing an unstructured data application atop a traditional database would be like trying to fit lithium batteries and electric motors inside a the frame of a gas-powered car — not a great idea!

To illustrate why this is, let's go back to the list of features that a vector database should implement (from the first section). Vector search plugins are missing two of these features — tunability and user-friendly APIs/SDKs. I'll continue to use Elasticsearch's ANN engine as an example; other vector search plugins operate very similarly so I won't go too much further into detail. Elasticsearch supports vector storage via the `dense_vector` data field type and allows for querying via the `_knn_search` endpoint:

```
PUT index
{
"mappings": {
"properties": {
"image-vector": {
"type": "dense_vector",
"dims": 128,
"index": true,
"similarity": "l2_norm"
}
}
}
```

```
}
PUT index/_doc
{
"image-vector": [0.12, 1.34, …]
}
```

```
GET index/_knn_search
{
"knn": {
"field": "image-vector",
"query_vector": [-0.5, 9.4, …],
"k": 10,
"num_candidates": 100
}
}
```

Elasticsearch's ANN plugin supports only one indexing algorithm: Hierarchical Navigable Small Worlds, also known as HNSW (I like to think that the creator was ahead of Marvel when it came to popularizing the multiverse). On top of that, only L2/Euclidean distance is supported as a distance metric. This is an okay start, but let's compare it to Milvus, a full-fledged vector database. Using `pymilvus`:

```
>>> field1 = FieldSchema(name='id', dtype=DataType.INT64, description='int64',
>>> field2 = FieldSchema(name='embedding', dtype=DataType.FLOAT_VECTOR, descrip
>>> schema = CollectionSchema(fields=[field1, field2], description='hello worlc
>>> collection = Collection(name='my_collection', data=None, schema=schema)
>>> index_params = {
'index_type': 'IVF_FLAT',
'params': {'nlist': 1024},
"metric_type": 'L2'}
>>> collection.create_index('embedding', index_params)
```

```
>>> search_param = {
'data': vector,
'anns_field': 'embedding',
'param': {'metric_type': 'L2', 'params': {'nprobe': 16}},
'limit': 10,
'expr': 'id_field > 0'
```

```
    }
    >>> results = collection.search(**search_param)
```

While both Elasticsearch and Milvus have methods for creating indexes, inserting embedding vectors, and performing nearest neighbor search, it's clear from these examples that Milvus has a more intuitive vector search API (better user-facing API) and broader vector index + distance metric support (better tunability). Milvus also plans to support more vector indices and allow for querying via SQL-like statements in the future, further improving both tunability and usability.

We just blew through quite a bit of content. This section was admittedly fairly long, so for those of you who skimmed it, here's a quick tl;dr: Milvus is better than vector search plugins because Milvus was built from the ground-up as a vector database, allowing for a richer set of features and an architecture more suited towards unstructured data.

## Technical challenges

Earlier in this tutorial, I listed the desired features a vector database should implement, before comparing vector databases to vector search libraries and vector search plugins. Now, let's briefly go over some high-level technical challenges associated with modern vector databases. In future tutorials, we'll provide an overview of how Milvus tackles each of these, in addition to how these technical decisions improve Milvus' performance over other open-source vector databases.

Picture an airplane. The airplane itself contains a number of interconnected mechanical, electrical, and embedded systems, all working in harmony to provide us with a smooth and pleasurable in-flight experience. Likewise, a vector database is composed of a number of evolving software components. Roughly speaking, these can be broken down into the storage, the index, and the service. Although these three components are tightly integrated[2], companies such as Snowflake have shown the broader storage industry that "shared nothing" database architectures are arguably superior to the traditional "shared storage" cloud database models. Thus, the first technical challenge associated with vector databases is designing a flexible and scalable data model.

Great, so we have a data model. What's next?

With data already stored in a vector database, being able to search across that data, i.e. querying and indexing, is the next important component. The compute-heavy nature of machine learning and multi-layer neural networks has allowed GPUs, NPUs/TPUs, FPGAs, and other general purpose compute hardware to flourish. Vector indexing and querying is also compute-heavy, operating at maximum speed and efficiency when run on accelerators. This diverse set of compute resources gives way to the second main technical challenge, developing a heterogeneous computing architecture.

With a data model and architecture in place, the last step is making sure your application can, well, read from the database — this ties closely into the API and user interface bullet points mentioned in the first section. While a new category of database necessitates a new architecture in order to extract maximal performance at minimal cost, the majority of vector database users are still acclimated to traditional CRUD operations (e.g. `INSERT`, `SELECT`, `UPDATE`, and `DELETE` in SQL). Therefore, the final primary challenge is developing a set of APIs and GUIs that leverage existing user interface conventions while maintaining compatibility with the underlying architecture.

Note how each of the three components corresponds to a primary technical challenge. With that being said, there is no one-size-fits-all architecture for vector databases. The best vector databases will fulfill all of these technical challenges by focusing on delivering the features mentioned in the first section.

## Wrapping up

In this tutorial, we took a quick tour of vector databases. Specifically, we looked at 1) what features go into a mature vector database, 2) how a vector database differs from vector search libraries, 3) how a vector database differs from vector search plugins in traditional databases or search systems, and 4) the key challenges associated with building a vector database.

This tutorial is not meant to be a deep dive into vector databases, nor is it meant to show how vector databases can be used in applications. Rather, the goal is to provide an overview of vector databases. This is where your journey truly begins!

In the next tutorial, we'll provide an introduction to Milvus, the world's most popular open-source vector database:

- We'll provide a brief history of Milvus, including the most important question — where does the name come from!
- We'll cover how Milvus 1.0 differs from Milvus 2.0 and where the future of Milvus lies.
- We'll discuss the differences between Milvus and other vector databases such as Google Vertex AI's Matching Engine.
- We'll briefly go over some common vector database applications.

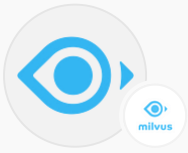
See you in the next tutorial.

. . .

[1]: We'll go over vector indices in more detail in an upcoming tutorial, so stay tuned.

[2]: Updating the storage component, for example, will impact how the vector indices are built in addition to how the user-facing services implement reads, writes, and deletes.

Data Science | Similarity Search | Machine Learning | Artificial Intelligence | Tech



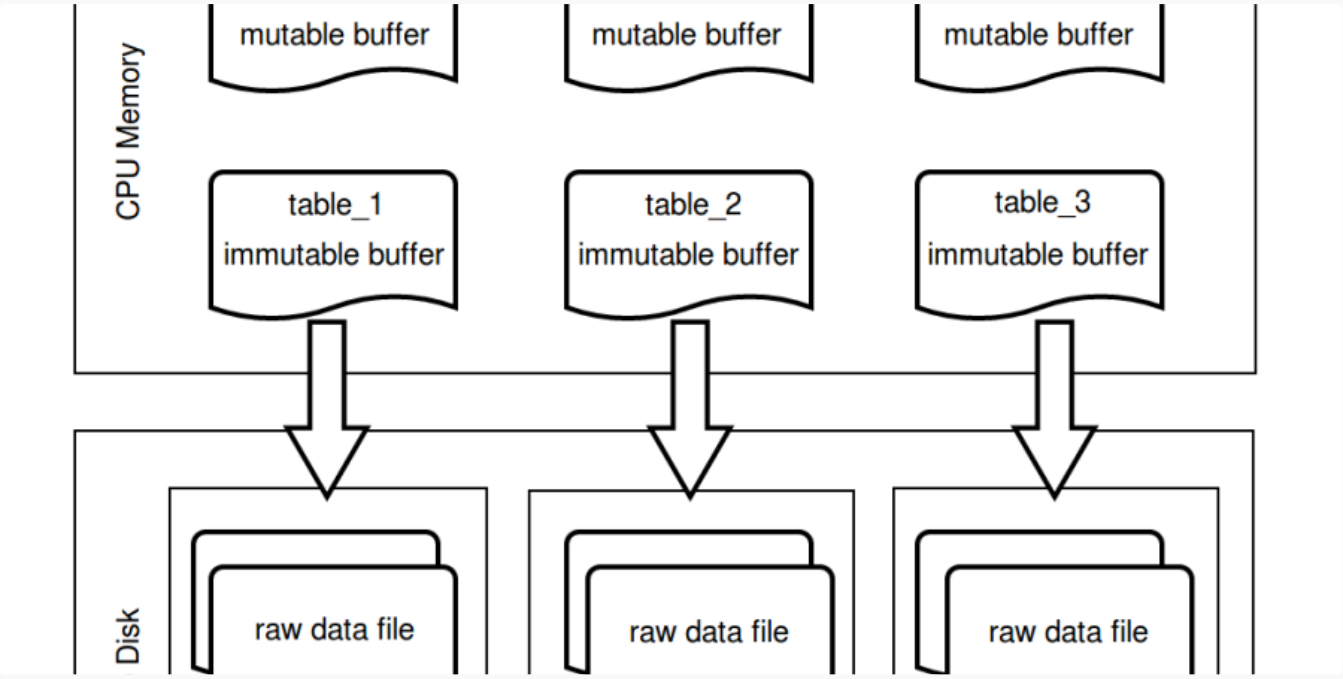
Follow

## Written by Milvus

989 Followers · Editor for Vector Database for AI

Open-source Vector Database Powering AI Applications. #SimilaritySearch #Embeddings #MachineLearning

## More from Milvus and Vector Database for AI




Milvus in Vector Database for AI

### Managing Data in Massive-Scale Vector Search Engine

How data management is done in Milvus

10 min read · Nov 13, 2019

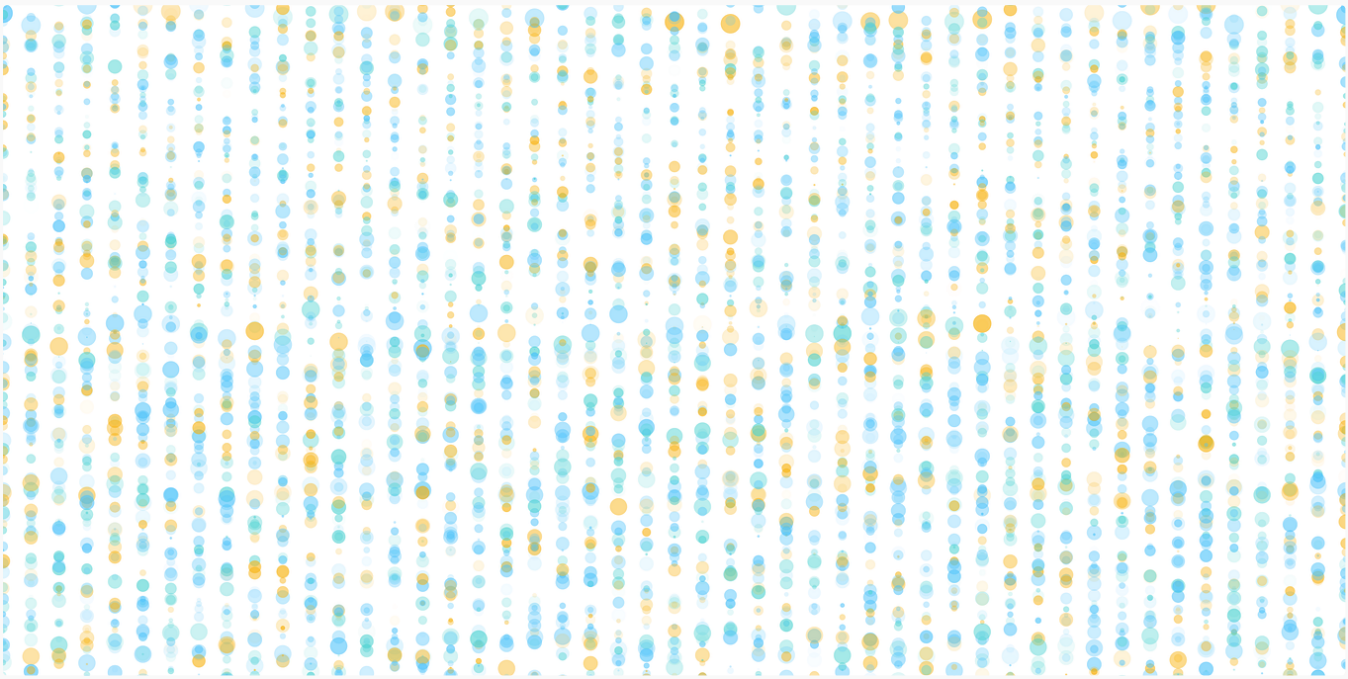
99

Rishav Ray in Vector Database for AI

## Building a real-time recommendation system

8 min read · Jun 16, 2022

146 4



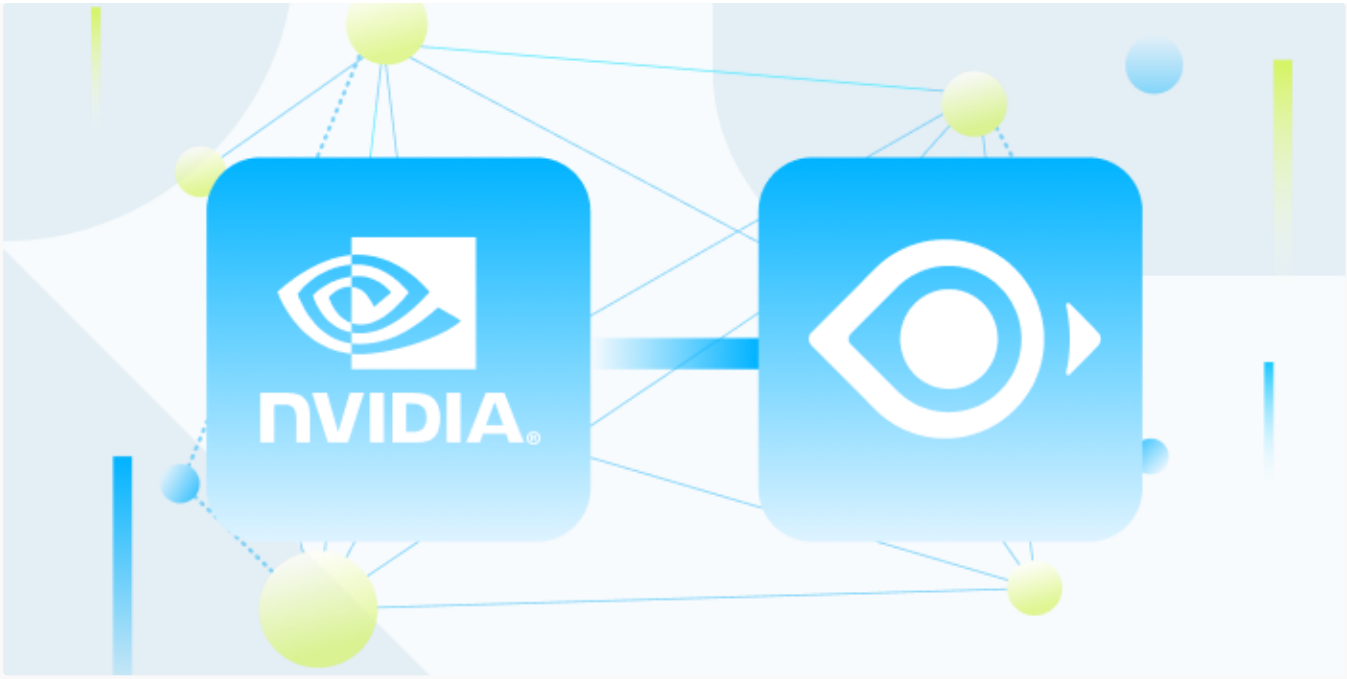
Milvus in Vector Database for AI

## Accelerating Similarity Search on Really Big Data with Vector Indexing

Without vector indexing, many modern applications of AI would be impossibly slow. Learn how to select the right index for your next ML app.

8 min read · Dec 5, 2019

102 1




Milvus in Vector Database for AI

# Efficient Vector Similarity Search in Recommender Workflows Using Milvus with NVIDIA Merlin

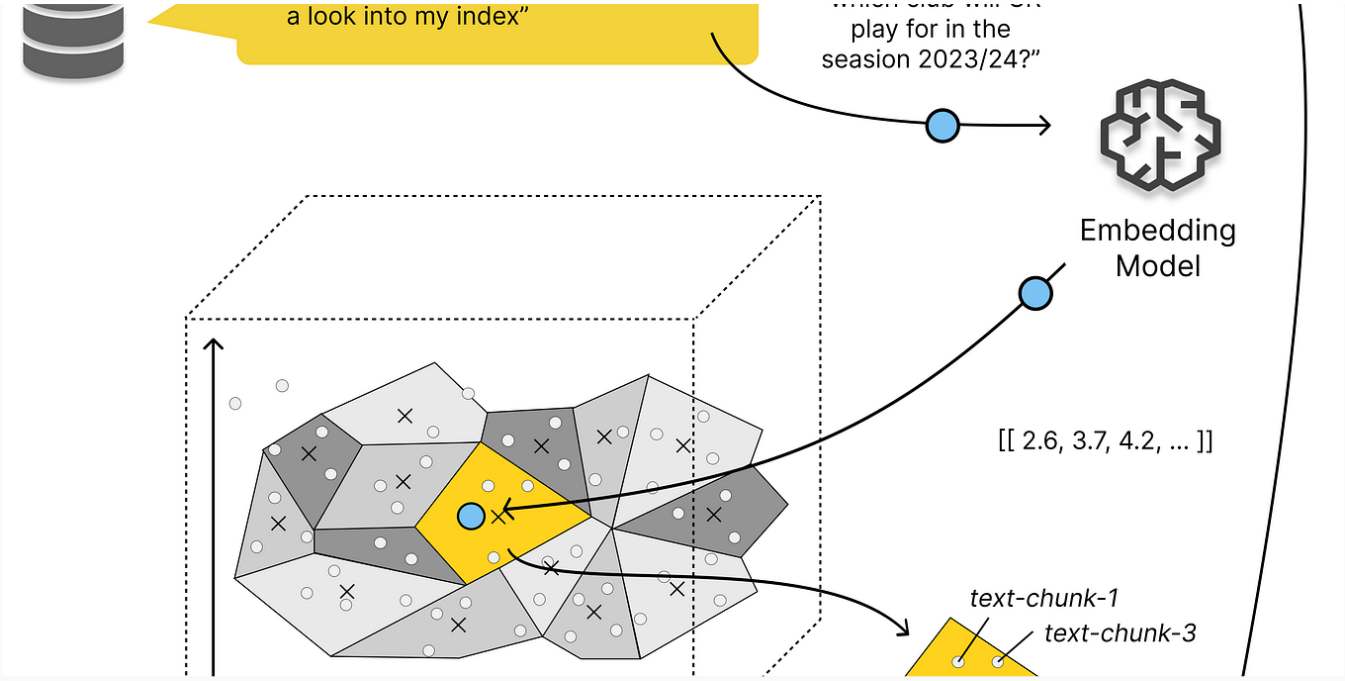
13 min read · Dec 16, 2023

5

See all from Milvus

See all from Vector Database for AI

## Recommended from Medium




Dominik Polzer in Towards Data Science

## All You Need to Know about Vector Databases and How to Use Them to Augment Your LLM Apps

A Step-by-Step Guide to Discover and Harness the Power of Vector Databases

✦ · 24 min read · Sep 18, 2023

1.3K 12


Mutahar Ali

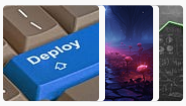
## Optimizing RAG: A Guide to Choosing the Right Vector Database

Introduction

7 min read · Dec 2, 2023

22

## Lists

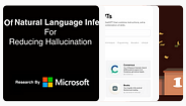


**Predictive Modeling w/ Python**

20 stories · 788 saves

**Practical Guides to Machine Learning**

10 stories · 912 saves

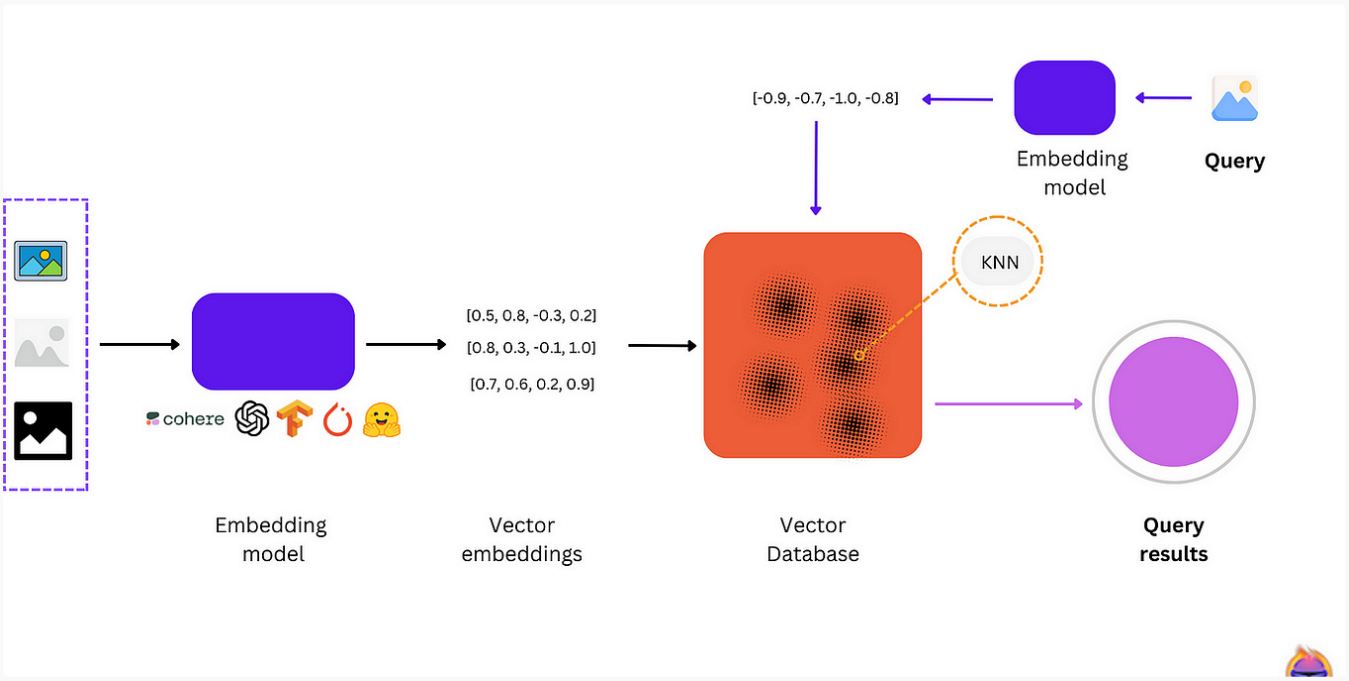



**Natural Language Processing**

1097 stories · 562 saves

**ChatGPT prompts**

34 stories · 972 saves



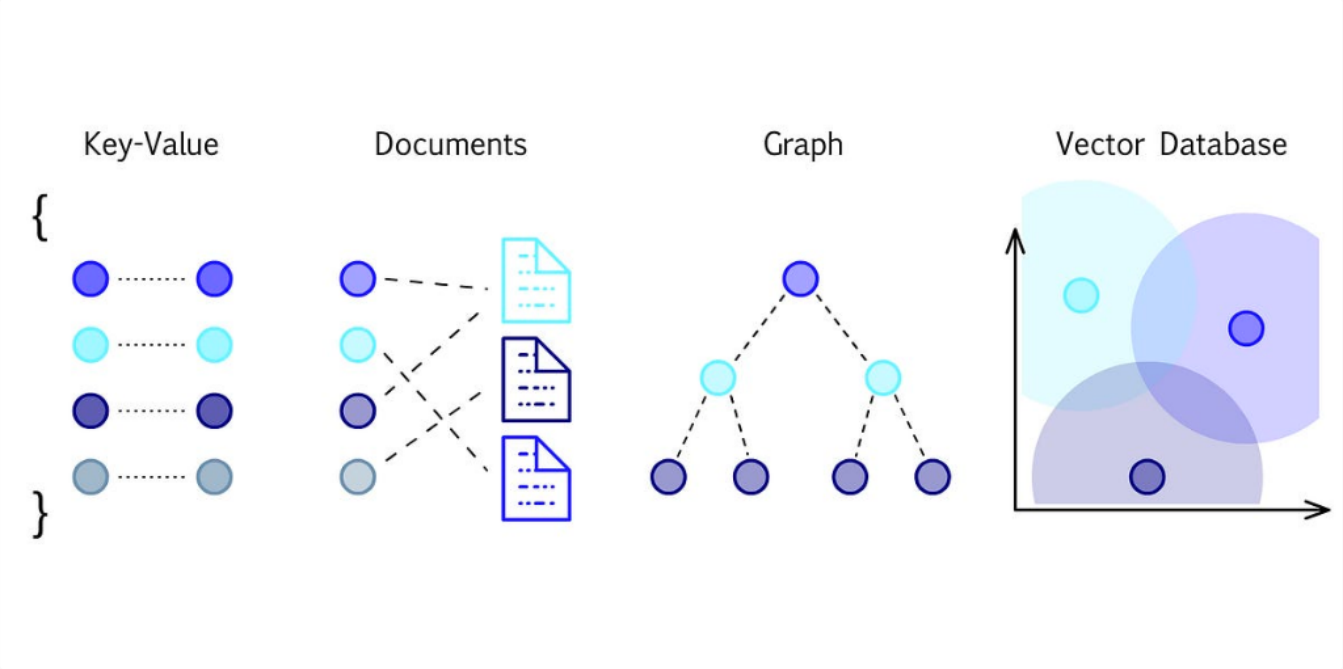

Apurva Kumar

# Vector Databases | Which one to Choose?

What is a vector database?

6 min read · Aug 19, 2023

9 1



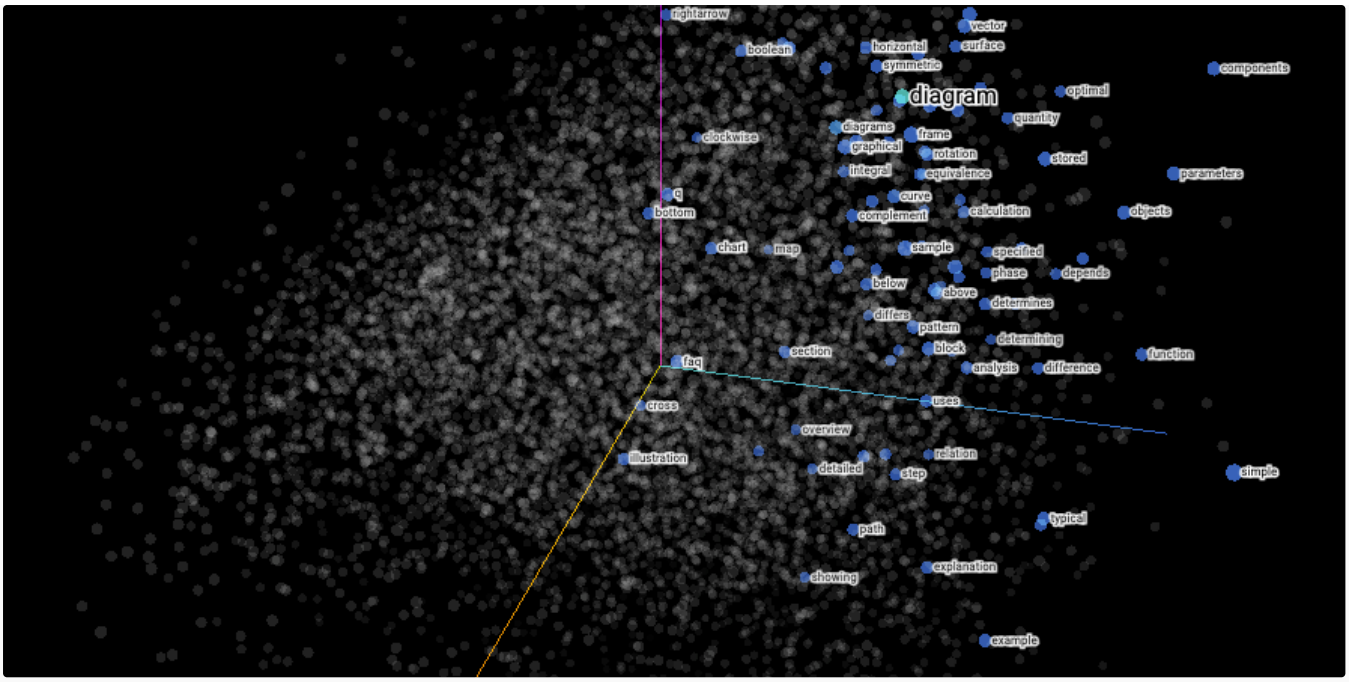

Pavan Belagatti in Data And Beyond

## Vector Databases: A Beginner's Guide!

In the age of burgeoning data complexity and high-dimensional information, traditional databases often fall short when it comes to...

10 min read · Aug 25, 2023

429 3




Navid Rezaei

## Which Vector Database Should I Use? A Comparison Cheatsheet

Semantic search and retrieval-augmented generation (RAG) applications require systems to be able to save lots of embedding vectors...

3 min read · Jul 30, 2023

218 8

Neil Kanungo in KX Systems

## Vector Indexing: A Roadmap for Vector Databases

Road maps are useful when traveling because they take dense geographical information from the real world and condense it into an easily...

7 min read · Sep 8, 2023

7

See more recommendations