Vector Databases (are All The Rage)

No kidding — A first attempt of a rationalization

Christoph Bussler

Published in Google Cloud - Community · 16 min read · Just now

Since everybody else wrote already about AI, I have to do it, too. I went to check out the landscape of vector database systems, a database technology variant that receives a lot of attention right now.

Usually you expect a few players in an established technology space, maybe a few more in a new, upcoming or newly relevant space.

Boom! Boy was I surprised when I started to try to list the existing vector database systems for review. And currently on an almost a daily basis new vector database systems and technologies appear — it'll slow down eventually.

Here the current list of database systems and technologies that I collected so far (and it is not a complete list). I plan to update this list as new database systems and technologies pop up on my radar — if you know of additional systems, please send it to me to incorporate.

The list is followed by a rationalization as well as additional discussions on benchmarks, database system comparisons, standardization, applications and more.

As a nugget, I included a brief discussion about AI models generating embeddings, including a huge heads up when embarking on a vector database systems deployment journey (there is a significant rub to be aware of).

Vector database systems and related technologies

The database systems and technologies in the list are not vetted by me regarding quality, correctness or any other features; I included all database systems, technologies, extensions that claim to be vector database systems or to support the vector data type (including its semantics).

- **ActiveLoop**: https://www.activeloop.ai/

- **Amazon Aurora**: https://aws.amazon.com/blogs/database/leverage-pgvector-and-amazon-aurora-postgresql-for-natural-language-processing-chatbots-and-sentiment-analysis/
- **AnalyticDB for PostgreSQL:** https://www.alibabacloud.com/help/en/analyticdb-for-postgresql/latest/4d5b34
- **AnnDB:** https://anndb.com/
- **ArcadeDB**: https://github.com/ArcadeData/arcadedb
- **Atlas**: https://atlas.nomic.ai/
- **AwaDB**: https://github.com/awa-ai/awadb
- **Azure Cognitive Search:** https://azure.microsoft.com/en-us/products/ai-services/cognitive-search
- **BagelDB**: https://www.bageldb.ai/
- **Cassandra**: https://cwiki.apache.org/confluence/display/CASSANDRA/CEP-30%3A+Approximate+Nearest+Neighbor(ANN)+Vector+Search+via+Storage-Attached+Indexes
- **Chroma:** https://www.trychroma.com/
- **Clarifai**: https://www.clarifai.com/blog/finding-what-you-need-a-comprehensive-guide-to-vector-search
- **ClickHouse:** https://clickhouse.com/
- **CockroachDB**: https://thenewstack.io/cockroach-labs-chief-targets-llms-with-vector-encoding/
- **DashVector**: https://help.aliyun.com/document_detail/2510225.html
- **Databricks:** https://www.databricks.com/company/newsroom/press-releases/databricks-introduces-new-generative-ai-tools-investing-lakehouse
- **DataStax Astra Vector Search:** https://docs.datastax.com/en/astra-serverless/docs/vector-search/overview.html
- **DB5**: https://vector5.ai/vector-database/
- **dingo**: https://github.com/dingodb/dingo
- **deeplake**: https://github.com/activeloopai/deeplake
- **DocArray Hsnwlib:** https://docs.docarray.org/user_guide/storing/index_hnswlib/
- **DocArray In-Memory**: https://docs.docarray.org/user_guide/storing/index_in_memory/
- **Elastic Search Relevance Engine (ESRE):** https://www.elastic.co/enterprise-search/generative-ai
- **embeddinghub**: https://github.com/featureform/featureform/tree/main/embeddinghub

- **Epsilla**: https://github.com/epsilla-cloud/vectordb
- **Google Cloud AI: Vertex AI Matching Engine:** https://cloud.google.com/vertex-ai/docs/matching-engine/overview
- **Google Cloud database systems:** AlloyDB for PostgreSQL, Cloud SQL for PostgreSQL (based on pgvector)
- **JaguarDB**: http://www.jaguardb.com/windex.html
- **KDB.AI**: https://kx.com/products/kdb-ai/
- **LanceDB**: https://github.com/lancedb/lancedb
- **Lantern**: https://github.com/lanterndata/lanterndb
- **Marqo**: https://www.marqo.ai/
- **Meilisearch**: https://www.meilisearch.com/
- **Metal**: https://getmetal.io/ (it is noteworthy that their search in documents is based on prompting: https://docs.getmetal.io/introduction)
- **Milvus**: https://milvus.io/
- **Milvus Lite**: https://github.com/milvus-io/milvus-lite
- **MongoDB Atlas**: https://www.mongodb.com/docs/atlas/atlas-search/field-types/knn-vector/
- **MyScale**: https://myscale.com/
- **Neo4j**: https://neo4j.com/generativeai/
- **NucliaDB**: https://github.com/nuclia/nucliadb
- **OpenSearch**: https://opensearch.org/platform/search/vector-database.html
- **Orama:** https://github.com/oramasearch/orama
- **Pinecone**: https://www.pinecone.io/
- **pg_embedding:** https://github.com/neondatabase/pg_embedding
- **pgvecto.rs:** https://github.com/tensorchord/pgvecto.rs
- **Qdrant:** https://qdrant.tech/
- **Redis**: https://redis.io/docs/interact/search-and-query/search/vectors/
- **redisvl**: https://github.com/RedisVentures/redisvl
- **RelevanceAI**: https://documentation.relevanceai.com/datasets/introduction
- **Rockset**: https://rockset.com/

- **ScaNN**: https://github.com/google-research/google-research/tree/master/scann
- **scikit-learn**: https://scikit-learn.org/stable/
- **SingleStore**: https://www.singlestore.com/
- **sqlite-vss**: https://github.com/asg017/sqlite-vss
- **StarRocks**: https://www.starrocks.io/
- **supabase**: https://supabase.com/
- **SuperDuperDB**: https://github.com/SuperDuperDB/superduperdb
- **SWIFT Vector Database**: https://github.com/Dripfarm/SVDB
- **Tair**: https://www.alibabacloud.com/help/en/tair/product-overview/what-is-tair
- **Tencent Cloud VectorDB**: https://technode.com/2023/07/05/tencent-cloud-unveils-ai-native-vector-database/
- **TerminusDB**: https://terminusdb.com/vectorlink/
- **Tigris**: https://www.tigrisdata.com/docs/quickstarts/quickstart-vector-search/
- **TileDB**: https://tiledb.com/blog/why-tiledb-as-a-vector-database
- **tinyvector**: https://github.com/m1guelpf/tinyvector
- **typesense**: https://typesense.org/
- **txtai**: https://github.com/neuml/txtai
- **Usearch**: https://unum-cloud.github.io/usearch/
- **Vald**: https://vald.vdaas.org/
- **vearch**: https://github.com/vearch/vearch
- **vectara**: https://vectara.com/
- **VectorDB:** https://github.com/jina-ai/vectordb
- **VectorLake**: https://github.com/msoedov/vector_lake
- **vector-storage**: https://github.com/nitaiaharoni1/vector-storage
- **vercel**: https://vercel.com/ (via pgvector)
- **Vespa**: https://vespa.ai/
- **vlite**: https://github.com/sdan/vlite
- **Weaviate:** https://weaviate.io/
- **Xata**: https://xata.io/

- **Zep**: https://www.getzep.com/
- **Zilliz**: https://zilliz.com/

There is a funny introduction to vector database systems using a dating app: Explain like I'm 5 — Vector Database Hype. More fundamental articles explain vector database systems like What is a Vector Database?

The Google query for vector database systems returns a seemingly endless list of results; quite amazing. This article looks at some of the funding events: The Fast-Emerging World of Vector Databases.

Reports are appearing for the specific segment of vector database systems, like New Trends of Vector Database Market To Receive Overwhelming Hike In Revenue That Will Boost Overall Industry Growth,Forecast 2030 | Shanghai Yirui Information Technology.

Other reports compare services and products from different cloud providers, like Generative AI Cloud Platforms: AWS, Azure, or Google?, Top vector database choices in 2023, or Vector databases (Part 1): What makes each one different?

Rationalization

Data type "vector"

A vector as defined by vector database systems is a data type with data type-specific properties and semantics.

A vector is a ordered set of scalar data types, mostly the primitive type float, and a vector does not have additional internal structure. In some systems a vector has an upper limit of elements, or a different performance behavior once a certain threshold of elements in the vector is exceeded.

For example in ClickHouse a vector is represented as array (ClickHouse Data Types), and in Zilliz as well (Create a collection).

Operations implementing "vector distance"

There is the very important key function of distance between vectors. Vector database systems implement different distance metrics. Not all database systems implement all alternatives of distance metrics and the implementation of the same distance metrics might not necessarily return the same result in all vector database systems. Some implement several distance metrics that can be selected at query time. Example metrics are (not a complete list):

- **Euclidean distance**: https://en.wikipedia.org/wiki/Euclidean_distance

- **Squared Euclidean distance**: https://en.wikipedia.org/wiki/Euclidean_distance#Squared_Euclidean_distance
- **Cosine similarity**: https://en.wikipedia.org/wiki/Cosine_similarity
- **Angular distance**: https://en.wikipedia.org/wiki/Angular_distance
- **Inner product space**: https://en.wikipedia.org/wiki/Inner_product_space
- **Dot product**: https://en.wikipedia.org/wiki/Dot_product
- **Manhattan distance**: https://en.wikipedia.org/wiki/Taxicab_geometry
- **Minkowski distance**: https://en.wikipedia.org/wiki/Minkowski_distance
- **Hamming distance:** https://en.wikipedia.org/wiki/Hamming_distance
- **Jaccard distance**: https://en.wikipedia.org/wiki/Jaccard_index

The blog What are Distance Metrics in Vector Search? discusses some of the distance metrics and has some comparisons between those as well.

It might be that over time vector database systems support the custom definition of distance metrics so that domain specific distances can be implemented by clients. I tried to find a vector database system that supports it already, but was not successful.

Index types for data type vector

To make retrieval efficient in a vector database system indexes can be specified. An index is specified on a set of vectors (table or collection or other set structure depending on the database system) and implemented on the vector data type so that it takes the vector specific structure into consideration.

Several vector database systems use Facebook AI Similarity Search (FAISS), for example, Milvus (Vector Index Basics and the Inverted File Index). FAISS implements several index types: Faiss indexes. Pinecone provides a discussion on several indexes: Nearest Neighbor Indexes for Similarity Search. A blog post on indexes: Vector databases (Part 3): Not all indexes are created equal.

Prominent indexes are the following, which each having different functional and non-functional properties (the list is not complete). Many of the vector database systems provide explanations for indexes in general or those that they implement:

- **Flat indexing**: https://www.pinecone.io/learn/series/faiss/vector-indexes/
- **IVF (inverted file index)**: https://thedataquarry.com/posts/vector-db-3/#inverted-file-index

- **IVFFlat**: https://www.timescale.com/blog/nearest-neighbor-indexes-what-are-ivfflat-indexes-in-pgvector-and-how-do-they-work/
- **Annoy**: https://en.wikipedia.org/wiki/Nearest_neighbor_search#Approximate_nearest_neighbor
- **PQ (product quantization)**: https://thesequence.substack.com/p/guest-post-choosing-the-right-vector
- **HNSW (Hierarchical Navigable Small-World)**: https://thedataquarry.com/posts/vector-db-3/#hnsw
- **Scalar quantization (SQ)**: https://thesequence.substack.com/p/guest-post-choosing-the-right-vector
- **Vamana/DiskANN**: https://thedataquarry.com/posts/vector-db-3/#vamana

There are libraries like FAISS. For example,

- **ScaNN**: https://ai.googleblog.com/2020/07/announcing-scann-efficient-vector.html
- **FalcoNN**: https://github.com/FALCONN-LIB/FALCONN
- **NMSLIB (Non-Metric Space Library)**: https://github.com/nmslib/nmslib

A blog discussing indexes and picking the right one is on substack: Choosing the Right Vector Index For Your Project.

In Pincone, when creating an index, the distance metric is an optional input parameter: create_index. Most likely the index will behave differently depending on the distance metric.

The question arises if at some point some of the vector database systems strive for implementing all distance metrics and all indexes within one system. Given the variety in both cases it is a tall order for a database system's software engineering team. From a customer perspective this would be great, though.

Query languages for data type vector

While storing vectors is a key functionality, a database system needs to provide a query language as well. Different forms of query languages are implemented by the different vector database systems:

- **SQL**. ClickHouse shows a SQL query for vector similarity search: Putting it all together.
- **API**. Zilliz shows searching White House speeches by an API: Using a Vector Database to Search White House Speeches. Pinecone provides an API as well: Introduction to Vector Search for Developers.

While the result of a similarity search by SQL or by API is most likely not vastly different (ideally not at all), the differences start to become more apparent when search results have to be further combined with other (possibly non-vector-based) data. Chances are that in SQL this will result in joins, projection or set operations (in case data are managed in the same schema or database), whereas in the case of APIs the required logic will be implemented as additional code, possibly invoking APIs of additional database systems or other non-database systems.

Classical implementation approaches of a specific data type

As in other cases in the past, like object-relational, XML, JSON, just to mention a few, there are a few basic alternatives for implementation of the vector data type in database systems (There is an interesting paper summarizing 35 years of data model development in database systems: What Goes Around Comes Around). These approaches explain to large extend the features and the functionality of the vector database systems:

- **Extend existing database systems**. Systems like Oracle, PostgreSQL, SQL Server and many more implement new data types over time into their existing database architecture making them multi-model database systems. The benefit is that a new data type can utilize the existing infrastructure and features, including transactions, backup, SQL support and many more avoiding implementing operational and semantic complexity just because of a new data type. On a basic level, a vector is "just" another data type available in an existing database system with its own semantics like distance metric and indexes.
- **Build data type specific database systems**. An alternative approach is to build a database system that implements the new data type as its core data type. Systems like MongoDB, Codasyl, IMS, Neo4J, MarkLogic, are examples of database systems that have a specific data type as their core data model.

There is a finer point when extending existing database systems with additional data types or functionality: it might be accomplished by extensions, or true additional functionality in the core database code. This distinction might not necessarily be relevant for your use case, but important to keep in mind.

To understand the scope of specific database systems and the extension of existing database systems, review Database of Databases and db-engines.com. Both are listing an ever expanding array of database systems and try to characterize the database systems by supported data types as well. The vector database system specific section on db-engines.com is this: DB-Engines Ranking von Vektor DBMS (notice the spelling with "k").

As a side note, existing database systems do not only include additional data types or other functionality over time, but also features like columnar processing (Oracle, AlloyDB), consistent global distribution based on partitioning (Yugabyte, CockroachDB), main memory caching, and many more.

In comparison, specialized database systems around a data type like vector or JSON have to add the “standard” or “regular” database system features like transactions, backup, or query languages in order to be address the expected database system functionality that is independent of a particular data type.

Articles are devoted to this topic of how to implement a new data type like for example Why Your Vector Database Should Not be a Vector Database or Do we really need a specialized vector database? Another article in this area is Five Ways Vector Databases Are Created Equal (and Not).

Historically, data type specific database systems, because they are specific to a data type, are “faster” in providing data type specific features, and have a “better” implementation (latency, performance, scale) as the data type specific properties do not need general functionality that applies to other data types as well. However, at the same time, historically, existing database systems catch up “quickly” implementing new data types, and providing the standard database features (like backup, transactions) as they have done before for the already implemented data types. And in scale, performance and latency they catch up as well. It is in the eye of the beholder based on requirements what type of database system to select.

The vector database system technology is expanding to additional expressiveness already

So far the focus of vector database systems is to implement the vector data type with similarity metrics and index types, and query syntax to some extent.

A shift starts taking place with for example Marqo (https://github.com/marqo-ai/marqo) — I just came across this database system. Instead of providing a pure vector data type implementation, Marqo raises the interface abstraction to documents; see Marqo Getting started for a brief first impression.

Benchmarks, comparisons, standardization

Benchmarks and comparisons between database systems are being specified, implemented and created. This list provides some of the performance or functionality benchmarks:

- Vector Database Benchmarks, vector-db-benchmark
- VectorDBBench — A Vector Database Benchmark Tool
- ANN Benchmarks

On the standardization front it will probably take some time until a standardization activity on the data type vector and its semantics takes place. That in turn will then cause standard set of operations that most compliant database systems will implement.

A standardization effort might be already in discussion, however, I have not come across it at this point. If you know about one, please let me know, I’d be curious to know about any activity. As example, here the standard incorporating JSON into SQL: ISO/IEC TR 19075–6:2017.

Comparisons of vector database systems are appearing, for example

- Which Vector Database Should I Use? A Comparison Cheatsheet
- This tweet (this “x”? “tweets” — “xs”?) has a two-dimensional categorization: https://twitter.com/YingjunWu/status/1667232357953466369

I guess that there will be published more over time as vector databases become a standard technology in context of AI architectures.

Use cases, applications and products

There are a huge number of use cases when implemented can benefit from a vector database system. The following is a random list of use cases based on different vector database systems that I came across over time; this list is truly random, and by no means complete:

- Explaining LLMs, Tools and Agents!
- LangChain101: Question A 300 Page Book (w/ OpenAI + Pinecone)
- How to Build LLM Applications With pgvector Vector Store in LangChain
- Using a Vector Database to Search White House Speeches
- The LangChain Python Package has integrations with many vector database systems: Vector stores.

Not only use cases and architectures are discussed, but products that provide a complete application system (“stack”) are being announced as well: VMware, Nvidia team on enterprise-grade AI platform.

Embeddings — Caveat Emptor

And here is the rub!

Embeddings

As the use cases have shown, embeddings are a key aspect in many cases. Embeddings are explained in many references, for example:

- Meet AI’s multitool: Vector embeddings
- New and improved embedding model
- What are embeddings?

One of the key aspects to be aware of is that embeddings have a direct relationship to models as they are generated from models. A model change might cause changes in embeddings as stated here: "**Smaller embedding size.** The new embeddings have only 1536 dimensions, one-eighth the size of `davinci-001` embeddings, making the new embeddings more cost effective in working with vector databases."

In addition, a new model might not be "better" in all aspects: "The new `text-embedding-ada-002` model is not outperforming `text-similarity-davinci-001` on the SentEval linear probing classification benchmark."

Furthermore, there are specific "risks and limitations" to be aware of as well. Llama 2 has an Acceptable Use Policy.

Consequences for vector database system deployment

In principle, there are two dimensions that need to be clearly understood and addressed when storing and managing embeddings in a vector database system:

- Use of different models
- Use of different versions of the same model
- or both (of course)

When using different models, embeddings derived from them cannot be mixed and matched, meaning, storing those in a vector database system must ensure that the embeddings are partitioned by model. And each partition needs to be aware of which model they came from — additional metadata must be managed so that at any point in time it is possible for each model to associate the derived embeddings and vice versa.

When embeddings come from different models, they cannot be directly compared. A result as a combination of the embeddings of different models must be derived from the individual results (e.g., returned document identifiers based on similarity).

In case a (current) model releases a new version (like in the example above) the key question arises how to manage the existing embeddings of the current version of the model. If embeddings of additional data are created based on the new model, those cannot necessarily combined with the embeddings of the previous version (unless explicitly stated in the model release notes). There are two choices:

- Recompute the embeddings of the current model to use the new version so that all embeddings in the database are of the same model version. This requires (and that's the key issue in this case) that any data for which embeddings were computed have to be kept and the same approach to embedding generation has to take place for the new version of the model.

- Or, keep the existing embeddings of the current model, and only generate embeddings for new data with the new version of the model. In this case each similarity search has to be done across two sets of embeddings, and since these are two sets (from two different versions of a model), the results might be different compared to having only one set of embeddings (how is similarity defined on two versions of a model, and how would search deal with that if it were to execute at all?).

The fact that multiple models are available, and each model over time might release new versions might cause a significant data management requirement to keep data from the very beginning in order to be able to either recompute embeddings of a new model or use the data to generate embeddings in multiple models concurrently.

Think, for example, about the implications managing your first billion documents that need to be kept so that embeddings for those can be recomputed on a new version of a model at any time.

Multi-modal models are available as well (e.g., Meta-Transformer: A Unified Framework for Multimodal Learning), however, from a data management perspective those are a single model that can process different input modalities. A blog describing a history perspective is The Multimodal Evolution of Vector Embeddings.

(As a side note, the following discusses comparing models: Comparing Different Vector Embeddings.)

Testing, testing, testing upon context changes

As soon as a model publishes a new version, or a database system publishes a new version, a regression test is in order to establish that the change (in model or in database implementation or both) does not degrade the similarity search.

If a regression is found, clients of the database system need to become aware of it in order to decide from a set of options, like rolling back the change, changing the client logic, or other approaches.

This is not specific to vector data type implementations, or vector database systems. This is standard procedure in any type of use cases.

Generating embeddings

There are many APIs and examples how to generate embeddings from your data (text, documents, images, etc.) so that you can store them in a vector database system; I won't list them here.

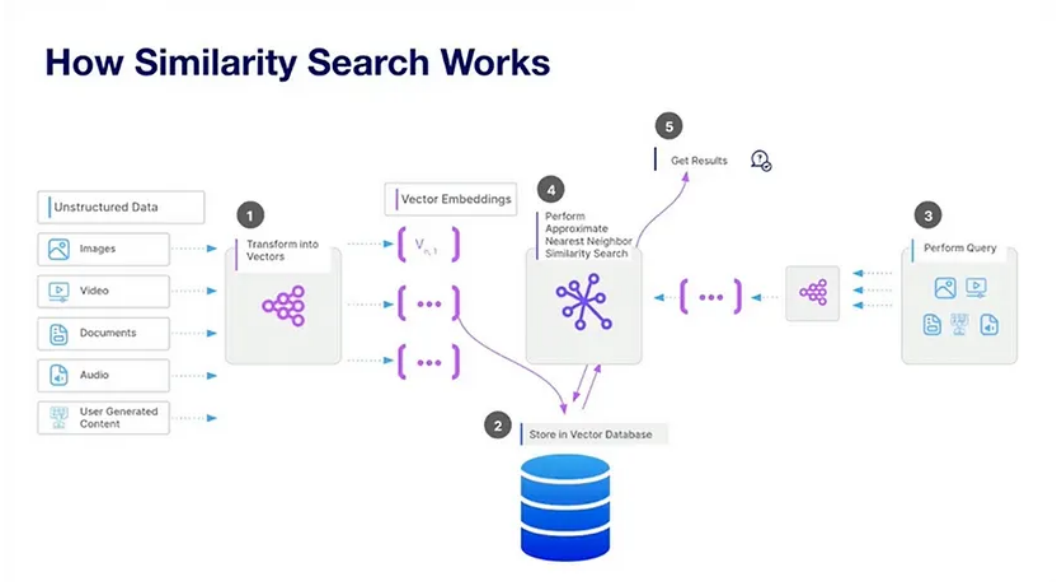
However, there is a tool called towhee that provides a collection of already implemented operators for many models in ETL style. If you have to generate many embeddings from many models, and possibly repeatedly, such a tool might be useful in your overall architecture.

Airbyte discusses connectors to vector database systems as part of data migration: AI shouldn’t waste time reinventing ETL with a concrete example here: How to Chat With Your Data Using OpenAI, Pinecone, Airbyte and Langchain: A Guide.

Architectures

Architectures are emerging that contain vector database systems as building blocks. Examples are

- Emerging Architectures for LLM Applications
- Microsoft, TikTok give generative AI a sort of memory
- Understanding the Fundamental Limitations of Vector-Based Retrieval for Building LLM-Powered Chatbots — (Part 1/3)
- The following diagram is cited from Vector Search Best Practices



From Vector Search Best Practices

Additional architectures will appear over time, with additional refinements (like for example several versions of the same model, or the data management aspect as discussed earlier).

Opinionated advice for selecting a vector database system

Here is my opinion of how to select a vector database system.

Whatever you implement might be so successful that scale, throughput, latency become major factors in addition to the core distinguishing functionality like similarity search and indexing.

Performance: benchmarks are your friends

Most likely your application based on a vector database system has the following characteristics:

- Interactive supporting a user interface with ideally a low latency
- Based on a growing data set that in turn will have a growing number of vectors
- High throughput due to access concurrency

These key characteristics require a vector database system that can provide low latency, an ever increasing data set and concurrency without significant degradation.

Once you have settled on an initial metrics for your application, use benchmarks to select the best vector database system.

Additional requirements like customer references, managed (cloud) offering or global footprint might be relevant for your situation as well.

Functionality: clear requirements and expectations are your friends, too

Not specific to vector database systems, but worth repeating is:

- Know what you are looking for when it comes to distance metrics, indexes, query expressiveness, managing of different versions of a model, and so on. Best is to have a clearly written document that is the basis for database system selection decisions.
- If index types or distance metrics that you require are not available in any database system, can you find a database system that allows you to contribute the missing features or functionality?
- If one system cannot satisfy all requirements, are you prepared to install and to manage more than one database system providing vector data type support?

During the selection process additional requirements might become apparent, and documenting those along the way is a good approach, as always.

Performance vs. functionality

Using the correct distance metrics is important for implementing the desired semantics of your use case. To make it perform requires a database system implementing indexes for fast retrieval and a database system that can scale. Both might not be available in one database system at this point in time.

Is one vector database system sufficient?

If you have more than one use case that require different distance metrics, or indexes, you might have to consider more than one vector database system if a single database system’s implementation does not support all use cases at the same time.

Be wary of convenience

In your company you might have database systems already deployed that also already provide the vector data type and operations (or plan to implement the vector type soon).

While this is great as the database systems are already deployed, they might not provide the performance (latency, throughput, storage capacity) that you eventually need for your vector processing. Or they might or might not provide the distance metrics or index that you need.

Even in these situations I suggest to clearly articulate your requirements. Use benchmarks to verify that already deployed database systems are able to satisfy your requirements as migrating to a different vector database system down the road is additional effort that increases the farther out the migration is going to be in the future.

Summary

This was my (in the end long) AI blog-focusing on vector database systems: I find the space super interesting coming from a database system background.

If you have any input, please send it: I'll strive for making this blog better over time and am happy to include more valuable information.

And, of course, other comments you have for me are welcome as well, please contact me anytime.

PS: Vector clocks are a thing as well: Vector Clocks: So what time is it?

Acknowledgements

Thanks to
Kushal Juneja for his encouragement to get this blog published.