Member-only story

Vector Databases and Vector Libraries (Part 4 of RAG Series)

Storing Embeddings in an efficient manner

Chandan Durgia · Follow
9 min read · Feb 8, 2024

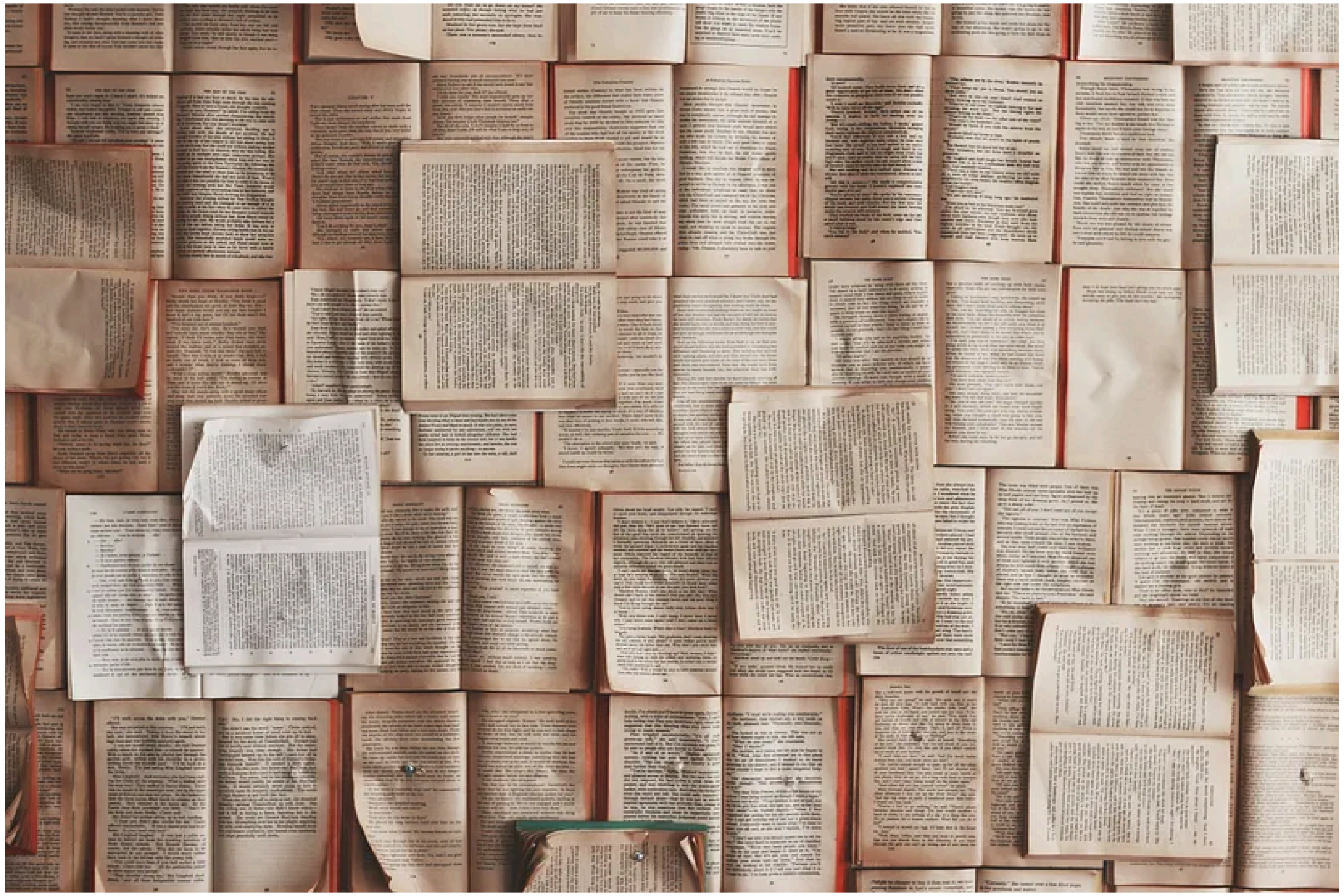
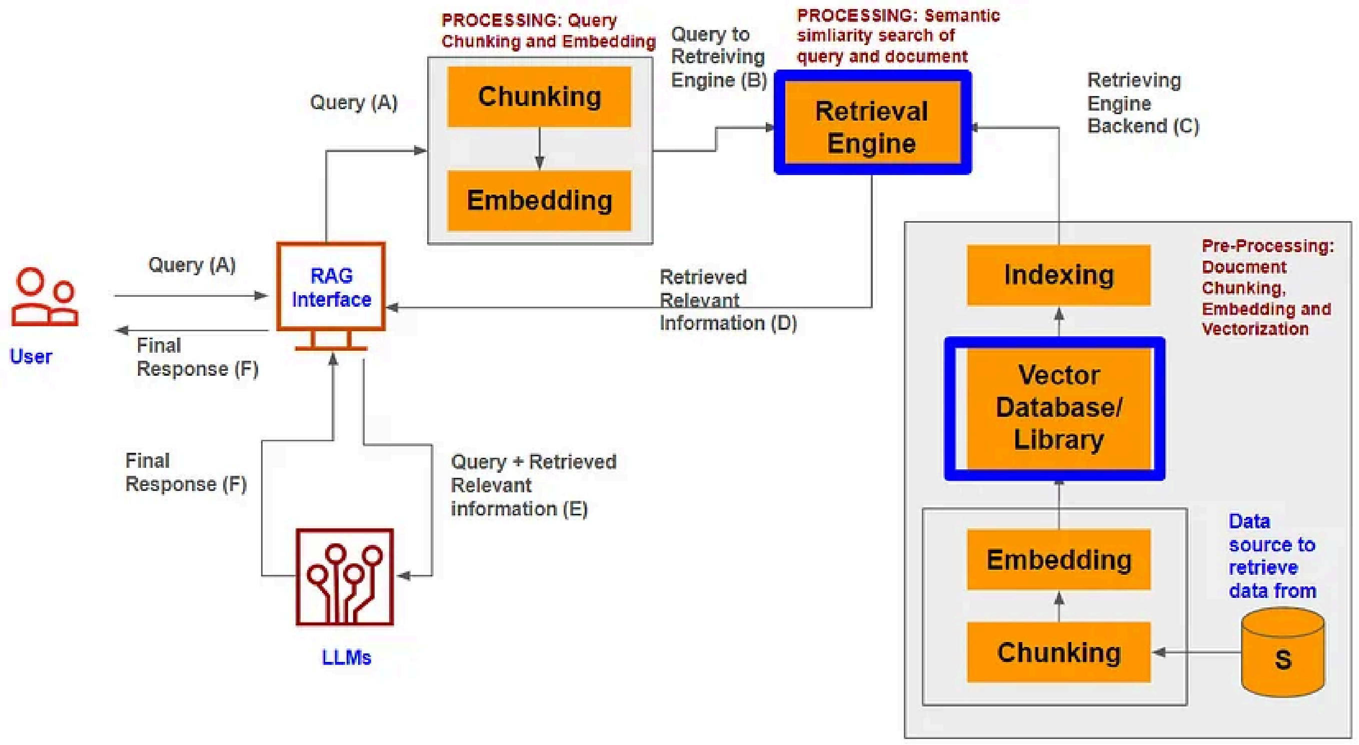


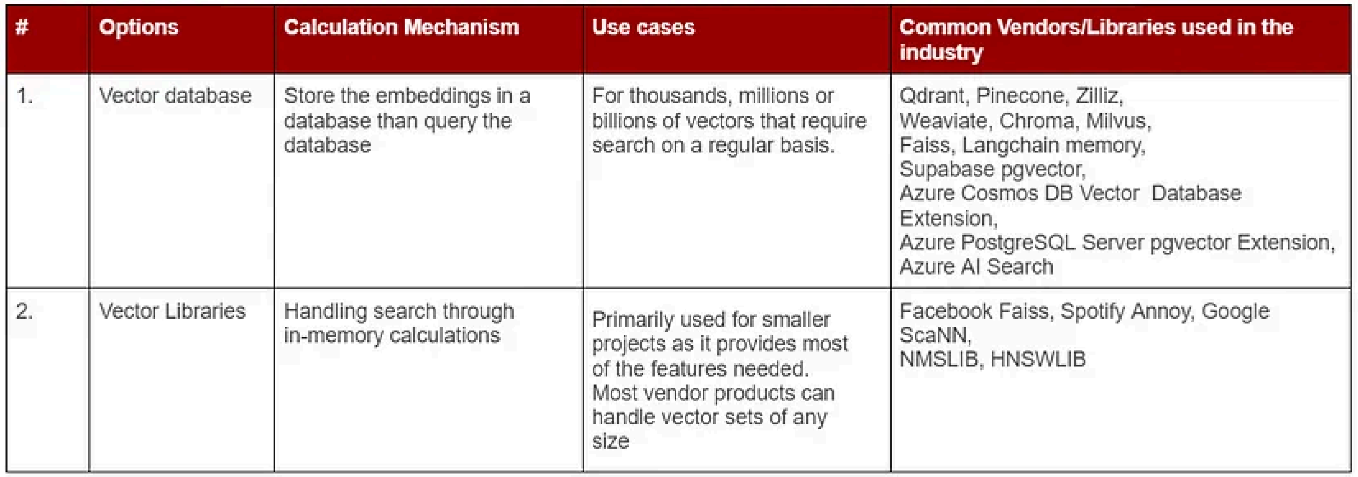
Photo by Patrick Tomasso on Unsplash

This is part 4 of the “Retrieval-Augmented Generation (RAG) — Basics to Advanced Series”. Links to other blogs in the series are at the bottom of this blog. Taking forward from part 1 (RAG Basics), part 2 (Chunking) and part 3 (Embedding) in this blog we will focus on the “Vector Databases and Vector libraries” component in addition we will also include the details around the Retrieval Engine (highlighted in Blue).


PROCESSING: Query Chunking and Embedding
Query to Retreiving Engine (B)
PROCESSING: Semantic simliarity search of query and document
Query (A)
Chunking
Embedding
Retrieval Engine
Retrieving Engine Backend (C)
Query (A)
User
RAG Interface
Final Response (F)
Retrieved Relevant Information (D)
Indexing
Pre-Processing: Doucment Chunking, Embedding and Vectorization
Vector Database/ Library
Final Response (F)
Query + Retrieved Relevant information (E)
LLMs
Embedding
Chunking
Data source to retrieve data from
S

RAG Architecture (image by Author)

In the last few blogs, we have discussed mechanisms to use the text data, break it into chunks and create vectors (Embeddings) of these chunks. Once the (source and query) chunks are converted to vectors, the next step involves comparing the query vectors with the source vectors to identify the closest similarity and extract the relevant chunks from the source. This can be done in two different ways:



#	Options	Calculation Mechanism	Use cases	Common Vendors/Libraries used in the industry
1.	Vector database	Store the embeddings in a database than query the database	For thousands, millions or billions of vectors that require search on a regular basis.	Qdrant, Pinecone, Zilliz, Weaviate, Chroma, Milvus, Faiss, Langchain memory, Supabase pgvector, Azure Cosmos DB Vector Database Extension, Azure PostgreSQL Server pgvector Extension, Azure AI Search
2.	Vector Libraries	Handling search through in-memory calculations	Primarily used for smaller projects as it provides most of the features needed. Most vendor products can handle vector sets of any size	Facebook Faiss, Spotify Annoy, Google ScaNN, NMSLIB, HNSWLIB

Image by Author

In this blog, we will first deepdive into the details and workings of Vector Databases. Then we will provide details of Vector Libraries whilst contrasting it with Vector Databases.

Vector Databases

Introduction to Vector database

Vector databases are designed specifically for storing and retrieving high-dimensional vectors/Embeddings. These databases can index embeddings and utilize models such as Approximate Nearest Neighbor (ANN) (details below) to facilitate quick retrieval of most similar/relevant data based on their semantic or contextual meaning (using functions like Cosine similarity or Euclidean).

Before getting into further details, its worthwhile to note a couple of considerations around scale and coverage of vector databases:

1. Vector dimensions could vary from tens to thousands (depending on the complexity and granularity of the underlying data)
2. Vectors can be derived from diverse types of raw data, including text, images, audio, video, and more. This versatility allows vectors to be applied in a wide range of applications, such as natural language processing (NLP), image/audio/video recognition (matching images/audio/video), and recommendation systems (product recommendations).

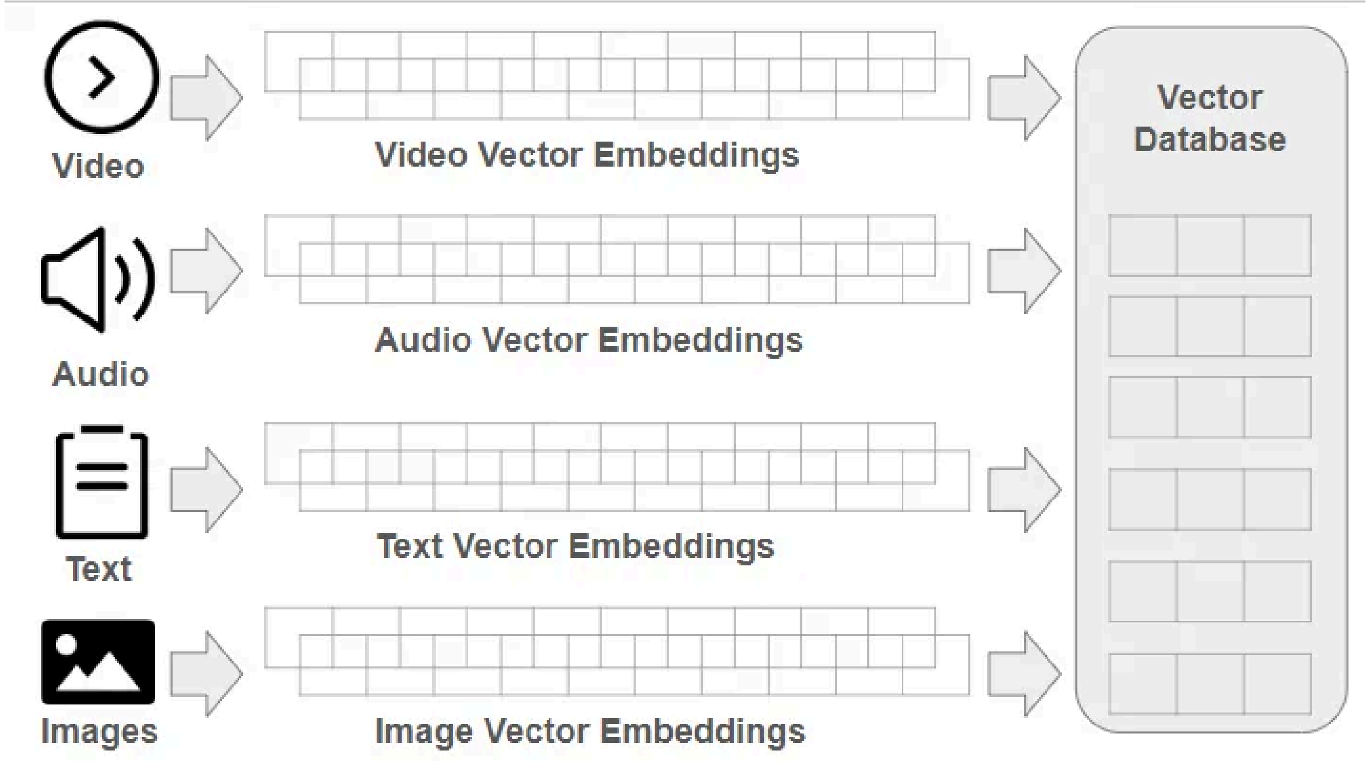


Image by Author

Vector databases are specifically designed to efficiently manage vector data. In the mentioned scenarios, these databases enhance the performance, scalability, and flexibility of the data, thereby unleashing data's complete potential. Vector databases support multitude of Embedding and their underlying use cases. Please refer to the section "Embedding — key use cases" in the following blog.

Note that Vector databases can store metadata associated with each vector entry. Users can query the database using additional metadata filters for finer-grained queries.

How vector Databases are different from traditional databases

There are many differences between the vector databases and traditional databases.

Imp

1. **Data Storage:** Traditional databases use rows and columns to store data which is usually numeric or string. This is usually structured data. In contrast, the vector database stores the vectors for varying categories of data like text, images, video etc. (unstructured data)
2. **Retrieval:** In traditional databases, SQL query enables retrieval of specific rows or columns based on **matches on specific keywords**. In contrast, vector databases work with vectors and apply a **similarity metric mechanism** to find the most similar vector. The underlying query mechanism is called Approximate Nearest Neighbor (ANN) search which uses different approaches for indexing and calculating similarities (like hashing, graph-based search, or quantization in a pipeline to retrieve neighbors of the queried vector). The rationale for better performance of Vector databases is because these databases pre-calculates the similarity between the vectors using ANN.

✓ 3. **Programming language support:** Unlike SQL which is the most common language for traditional databases, Vector databases support popular data science languages such as Python, Tensorflow, Java and SQL.

✓ 4. **Performance:** Though the relational databases querying performances have been improved considerably, for vector data, vector database support: fast data ingestion, advanced storage, sharding (dividing data across multiple nodes), indexing, query processing techniques and replication (creating redundant copies of data).

Note that these benefits are not just important but a necessity for vector databases as they are expected to efficiently handle diverse queries and algorithmic patterns across similarity search, anomaly detection, observability, fraud detection etc.

How does querying mechanism works in a Vector database: ANN search

As one can envision, performing similarity calculations between a query and numerous embedded objects using a basic k-nearest neighbors (kNN) algorithm can be time-intensive, particularly when dealing with millions of embeddings. Therefore, vector databases employ ANN which uses diverse indexing techniques and similarity calculations. This sacrifices a certain level of precision in favor of swiftness, thereby retrieving the closest approximate matches to a given query. Note that Vector databases excel at rapidly retrieving comparable items from a query due to their pre-computed nature.

When a vector database is queried it goes through the following three stages:

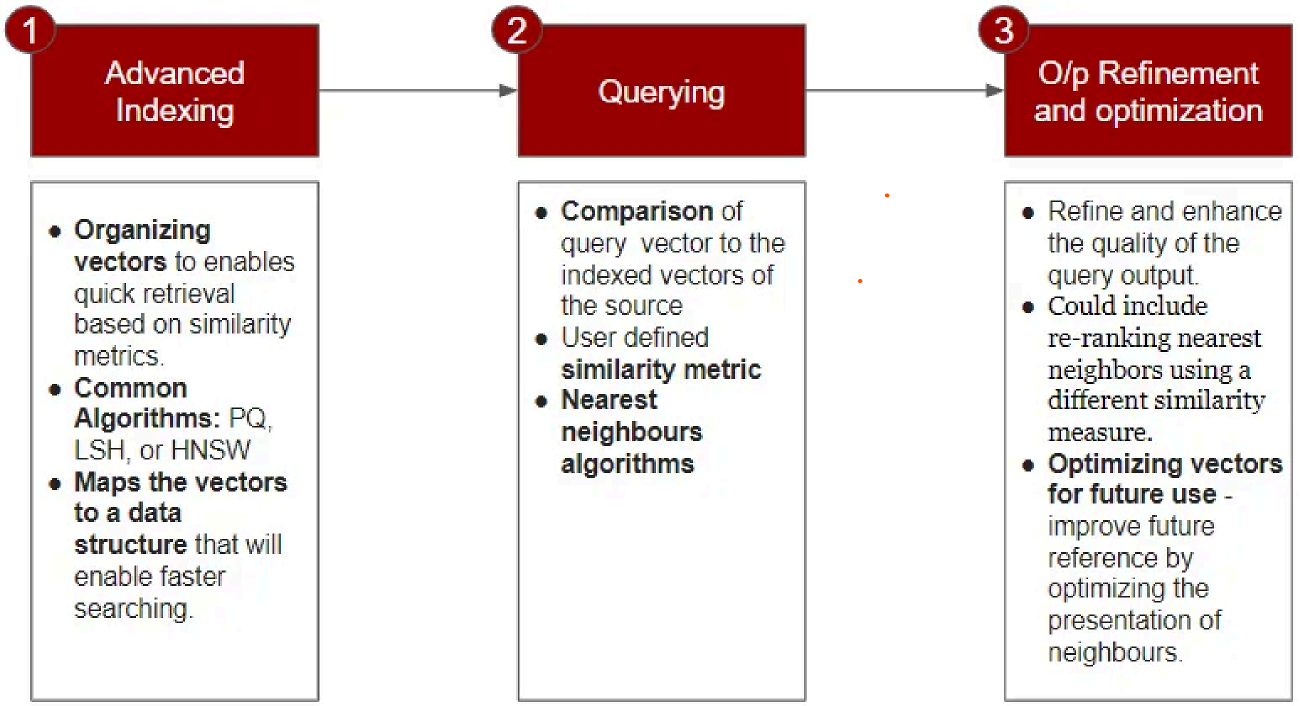

1
Advanced Indexing
2
Querying
3
O/p Refinement and optimization
• Organizing vectors to enables quick retrieval based on similarity metrics.
• Common Algorithms: PQ, LSH, or HNSW
• Maps the vectors to a data structure that will enable faster searching.
• Comparison of query vector to the indexed vectors of the source
• User defined similarity metric
• Nearest neighbours algorithms
• Refine and enhance the quality of the query output.
• Could include re-ranking nearest neighbors using a different similarity measure.
• Optimizing vectors for future use - improve future reference by optimizing the presentation of neighbours.

Image by Author

We will discuss the details of these three stages below:

1. **Indexing**

Indexing stage uses a variety of algorithms to index the vectors — with a key objective of enabling efficient and fast querying. The key idea behind all algorithms is how to represent the data in a compressed format so that it can be quickly traversed to get the output. Though there are multitude of algorithms, the key ones are discussed below:

a. Random Projection (RP): Simplistically key idea behind this algorithm is:

i. Convert high dimensional vectors into low dimensional vectors using encoding (uses Random numbers matrix) such that it preserves the similarity

ii. For querying use the same encoding to convert the query to lower dimensional vectors.

iii. Comparison of #1 and #2 would be quick and efficient as both are in lower dimensional space.

Note that the accuracy of the output would be a function of the encoding (through Random numbers)

b. Product quantization (PQ): Again, simplistically the idea behind the algorithm is as follows:

i. Break the original large vector into smaller segments.

ii. Assign "code(s)" to each segment using some encoding mechanism — so that the chunks can be joined together again. An intuitive way of thinking about these codes is as a latent new vector which is the center of the clustering of the segments. The way to visualize these codes is a person carrying a placard mentioning which all segments are there with the code.

iii. For querying, quantize the query vector using the same "code" mechanism. Then, use this "code" based query to quickly find the nearest code and then the nearest vectors.

Note that the accuracy of the output would be a function of the size of the "code(s)", higher the size more the accuracy but higher the cost.

c. Locality-sensitive hashing: Intuitively this algorithm is very similar to the PQ algorithm, with a key difference being instead of using "code", this algorithm uses a Hashing mechanism to assign the segments into various buckets. The query uses the same Hashing mechanism and gets assigned to one of the buckets where the closest vectors are further searched. An intuitive way to think about the buckets are the “pin codes” of a place.

d. Hierarchical Navigable Small World (HNSW): In HNSW algorithm, the vectors are tied to buckets called nodes here but the nodes are also structured in a hierarchical manner so that the traversing can happen quickly.

Note that there are other algorithms as well which are commonly used but not covered here, some examples being faiss-ivf, scann.

2. Querying: Once the indexing of the Source and the Query is done, the similarity between the query and the source index can be performed through a number of Similarity Measures. These measures help find the nearest neighbors to the query from the indexed vectors. Common similarity measures include cosine similarity, dot product, Manhattan distance, Hamming distance, Euclidean distance etc.

3. O/p refinement and optimization: There are 2 key processes which are performed in this stage:

1. The output could be re-ranked by changing the similarity measure from what was chosen in Querying stage
2. The initial indexing of the source could get optimized on the basis of the query and similarity search.

This concludes the section and details of Vector database, the next part of the blog would focus on Vector Libraries and the comparison between Vector databases and Vector Libraries

Vector Libraries (and comparison with Vector databases)

In simple terms, Vector libraries are stand alone libraries which store vector embeddings in "in-memory indexes" (instead of the database). A relatable analogy is RAM (Vector Libraries) vs Hard Disk (Vector Database).

Having deep dived considerably on Vector Databases above, I have covered Vector Libraries here through comparison of its characteristics with Vector Databases.

1. **How are vectors stored:** Unlike Vector database which stores both the vector Embeddings and the associated objects. Vector libraries only store vector embeddings and object IDs but not the associated objects. The

objects have to be stored in a separate storage and looked up basis the object IDs

2. **Ongoing Optimization:** As explained in the phase 3 of the query mechanism above, the vector database keeps on optimizing the indexing on the basis of queries. However, for Vector Libraries, the index is not optimized on every run. It is immutable.

3. **Lock for querying:** Vector databases allow for querying on the amount of data it contains and the user can keep on adding the data in a regular manner. However, for Vector Libraries, user cannot query unless whole data is imported.

4. **Data Management:** Creating, editing and removing data is easier and can be done in real-time in Vector databases but not in Vector Libraries

5. **Metadata Storage:** As mentioned above, Vector databases can store metadata associated with each vector entry. Users can then query the database using additional metadata filters for finer-grained queries. Vector Libraries do not support this functionality.

6. **Others:** Expectedly, Vector databases being a database are easier for managing backups, integrating with the technical ecosystem, scaling, security, crash recovery, persistence, multi-processing etc.

Conclusion

In conclusion, vector databases and vector libraries are both powerful tools in the realm of data management and analysis. While vector databases excel in providing efficient storage and retrieval of vector data, vector libraries offer a wide range of pre-trained models and algorithms for vector-based tasks.

Vector databases, with their ability to handle large-scale vector datasets and perform complex spatial queries, are ideal for applications such as geospatial analysis, recommendation systems, and similarity search. They provide a robust infrastructure for storing and querying vector data, enabling faster and more accurate results.

On the other hand, vector libraries offer a vast collection of pre-trained models and algorithms that can be readily used for various vector-based tasks. These libraries provide a convenient way to leverage state-of-the-art techniques without the need for extensive training or expertise. They are particularly useful for tasks like natural language processing, image recognition, and sentiment analysis.

When it comes to choosing between a vector database and a vector library, it ultimately depends on the specific requirements of your project. If you need efficient storage and retrieval of vector data, along with the ability to perform complex spatial queries, a vector database would be the ideal choice. However, if you are looking for ready-to-use models and algorithms for vector-based tasks, a vector library would be more suitable.

Some indexing methods
- Random Projection (RP)
- Product Quantization
- Locality - Sensitive Hashing
- Hierarchical Navigable Small World

Vector database
↳ Index, Query, Optimize → Continuously improve
↳ Use ANN or other algorithm for arranging and fetching indexed data.

Here is a view of what's in the series, feel free to let me know if you would like me to cover any specific aspect.

1. Retrieval-Augmented Generation (RAG) — Basics to Advanced Series (Part 1)

2. Pre-processing block — Chunking (part 2): strategies, considerations and optimization

3. Pre-processing block — Embedding (part 3): Types, Use cases and Evaluation.

4. Pre-processing block — Vector Databases and Vector Libraries (part 4) — this blog