Vector Search
Definition: Vector search is a way to find related objects that have similar characteristics using machine learning models that detect semantic relationships between objects in an index.

What are vector embeddings?
Vectorization is the process of converting words into vectors (numbers) which allows their meaning to be encoded and processed mathematically.

How vector embeddings are created?

- For words, use word embedings like word2vec, GloVe, BERT word embeddings, ELMo and so on.
- For Sentence, there are two ways:
    1) Get word embeddings and do average
    2) Get embeddings via sentence transformers like BERT sentence embedding, Universal Sentence Encoder, etc.(You can find in hugging face repositories)
- For Image, use CLIP model(Contrastive Language-Image Pre-training)

Once you get embeddings do any of similarity
1. Jaccard Similarity
2. Cosine Similarity
3. Earth Mover Distance
4. Jensen-Shannon distance

To do:
Now, There are two types of deduplication application

- One application is the cleaning of databases to make sure there is only one record per subject.
- A second application is to prevent that a new sample is entered in the database as a new entry(Basically check similarity between pair of sentences). Here all study materials focused on this application

QUICK POC

- Get database according to application(segregate similar/duplicates and non-similar/non-duplicates pairs)
- Get embeddings
- Do similarity
- Get results and performance

Similarity via Vector Database:
There are also various algorithms which can be used to search a vector database to find similarity. These include:

- ANN (approximate nearest neighbor): an algorithm that uses distance algorithms to locate nearby vectors.
- kNN: an algorithm that uses proximity to make predictions about grouping.
- (SPTAG) Space partition tree and graph: a library for large scale approximate nearest neighbors.
- Faiss: Facebook's similarity search algorithm.
- HNSW (hierarchical navigable small world): a multilayered graph approach for determining similarity.

Types of Vector Database available:

- Milvus - open source
- Weaviate - open source
- Pinecone
- Chroma - open source
- Rektor
- Qdrant
- Vespa

Vector Similarity readily available in below systems:

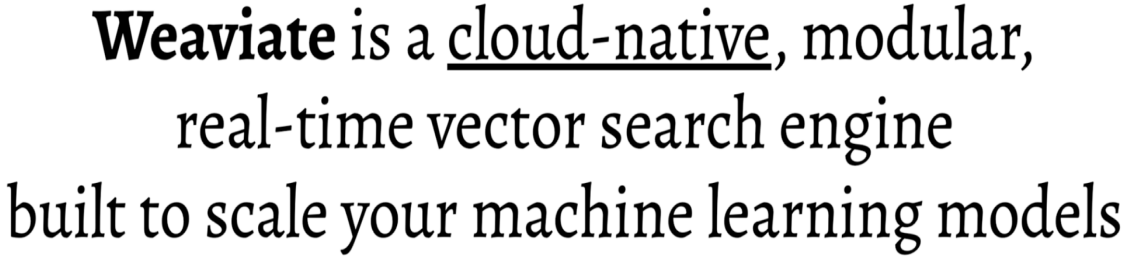
- Pgvector for postgres (Database system)
- Redisearch for Redis(Database system)

Advantages of Vector Database:

- Can Extend LLM to Long term memory
- Search based on meaning of context
- Similarity Search for text/images/audio/video
- Recommendation engine

Weaviate

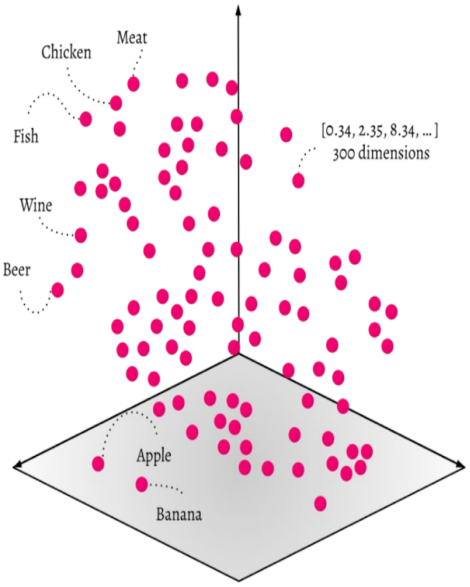
Weaviate is a vector database which enables fast and scalable vector storage, search and retrieval.



Weaviate - How does it work?

- Data is stored as high dimensional vectors
- Vector positions capture data meaning and context
- Pre-trained NLP module for automatic
  - Vectorization
  - Classification
  - Nearest neighbor search

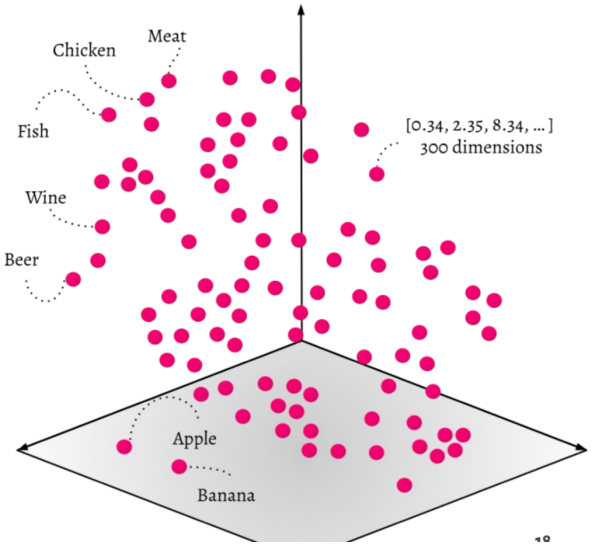
⇒ Weaviate *understands* the data



Weaviate - How does it work?

1. Weaviate Machine Learning Modules

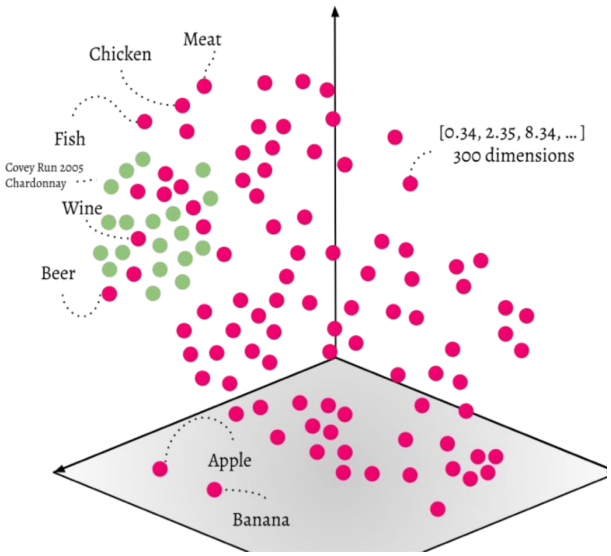
- E.g. NLP module trained with *fasttext*
- This language model represents all words and concepts in the hyperspace.



Weaviate - How does it work?

2. Automatically vectorize and index your data

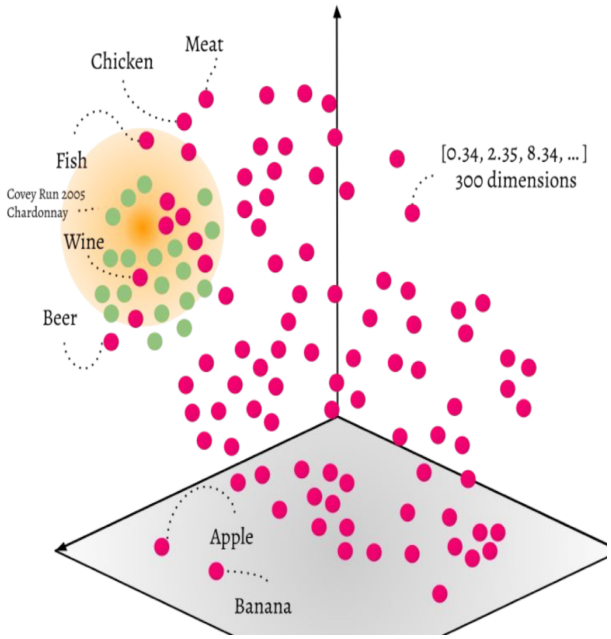
- Weaviate *understands* your data
- When you import data: Weaviate looks at the language in your data object
- E.g. a *Chardonnay* is closely related to *Wine*, *White* and the *food* it fits with



Weaviate - How does it work?

3. Search query

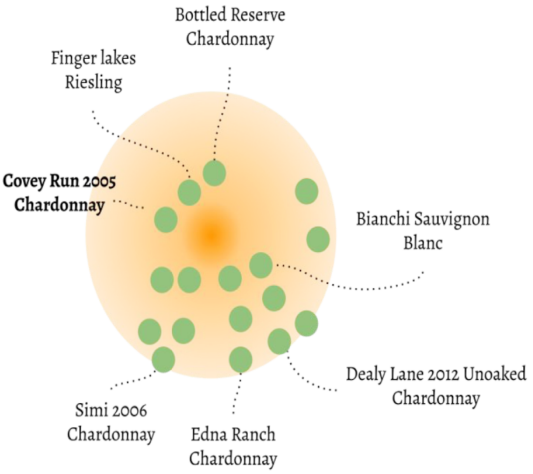
- Your search queries in natural language will also be vectorized and understood by the machine learning module of Weaviate
- It is places close to the words and data object that are semantically related to the query
- E.g. "*Wine that fits with a seafood dish*"



Weaviate - How does it work?

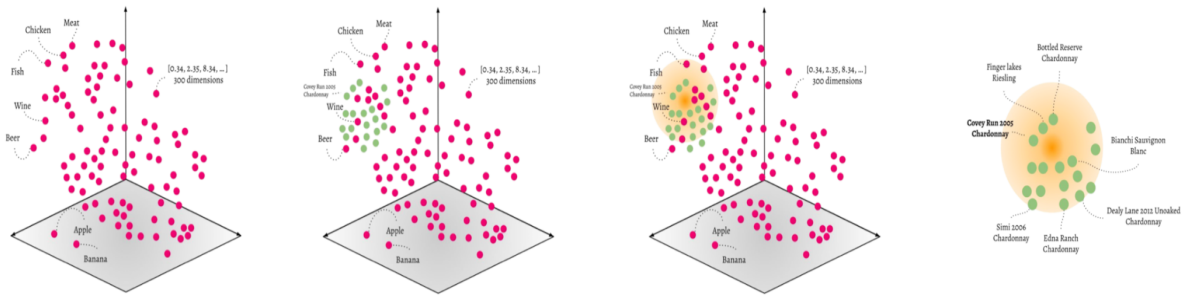
4. Results

- The data objects that are closest to the search query are retrieved from the dataset



Weaviate - How does it work?

Pretrained ML model + Your own data + Your search query = Results



(e.g. NLP) *(e.g. Wines)* *(e.g. Wine for seafood)* *(e.g. Chardonnay)*

Weaviate Vectorization Modules

Machine learning models

- **Weaviate's NLP module: trained with *fasttext***
  - Contextionary, in multiple languages
- **Transformer modules: general-purpose NLP architectures**
  - e.g. BERT, GPT-2, RoBERTa, XLM, DistilBert, etc
- **Custom modules**

How to run

- Haystack, a framework for NLP applications, provides a range of Document Store integrations. Weaviate is one of the possible DocumentStore integrations.
- Haystack expects text to come in Document format. So, first we have to prepare the data to write to WeaviateDocumentStore.
- Launch a Weaviate cluster
- Write the documents to a WeaviateDocumentStore
- The Retriever

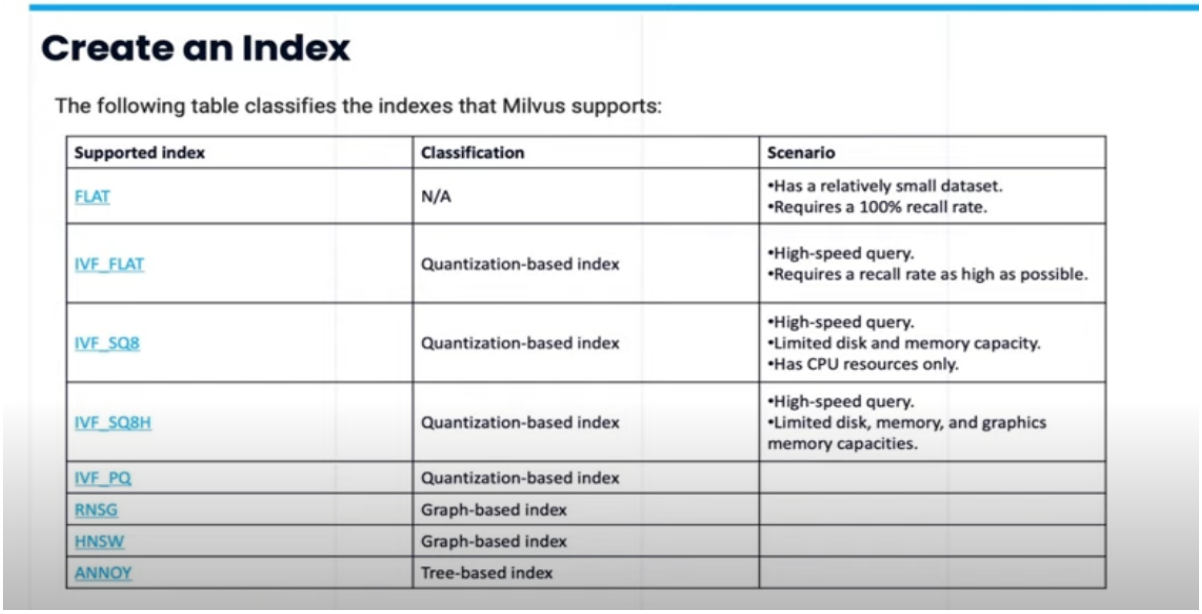
Next, we define our Retriever. This component acts as a filter to retrieve only the relevant document from your DocumentStore, based on your questy. You have a few options, depending on your usecase and desired accuracy outcomes, you can pick and choose which to use:
EmbeddingRetriever
DensePassageRetriever

Milvus:
Working Functionality similar to Weaviate

How to run:

- Create Collection
- Insert Entites to a collection

- Create Index
- Conduct Vector Serach



Create an Index

The following table classifies the indexes that Milvus supports:

Supported index	Classification	Scenario
FLAT	N/A	•Has a relatively small dataset. •Requires a 100% recall rate.
IVF_FLAT	Quantization-based index	•High-speed query. •Requires a recall rate as high as possible.
IVF_SQ8	Quantization-based index	•High-speed query. •Limited disk and memory capacity. •Has CPU resources only.
IVF_SQ8H	Quantization-based index	•High-speed query. •Limited disk, memory, and graphics memory capacities.
IVF_PQ	Quantization-based index	
RNSG	Graph-based index	
HNSW	Graph-based index	
ANNOY	Tree-based index	

Milvus Features & benefits
Comprehensive Similarity Metrics
Leading-Edge Performance
Dynamic Data Management
Near Real Time Search
Rich Data Type & Advanced Search
Cost Efficient
Highly Scalable and Robust
Cloud Native
Ease of Use

FAISS:
Below is for text embedding. Semantic search

- Word Embedings are created
- Use mean pooling to create sentence vectors
- Once we have our embeddings we calculate Cosine similarity

FAISS index used for fast nearest neighbour lookup
Entire in demo is in study material

Working with Vector Challenges:

- Not efficient(Takes time to get results). Needs faster queries
- Can't scale without significant investment in computer processing.
- Need more compute power

Solution to Vector Challenges:
Binary Vectors: Neural hashing makes vector-based search as fast as keyword search and this is done without the need for GPUs or specialized hardware. Neural hashing uses neural networks to hash vectors — compressing the vectors into binary hashes (or binary vectors).

Hybrid search: Hybrid search is a new method to combine a full-text keyword search engine and a vector search engine into a single

Elastic Search:

Advantages with Elastic for image similarity search:

- **Reduce application complexity:** With Elastic you don't need separate services for running kNN search and vectorizing your search input. Vector search and NLP inference endpoints are integrated within a scalable search platform.

- **Scale with speed.** In Elastic, you get scale and speed. Models live alongside nodes running search in the same cluster, which applies to on-premise clusters, and even more so if you deploy to the cloud. Elastic Cloud allows you to easily scale up and down, depending on your current search workload.

ANN algorithm used Elastic Search:

- Elasticsearch 8.0 uses an ANN algorithm called Hierarchical Navigable Small World graphs (HNSW), which organizes vectors into a graph based on their similarity to each other. HNSW shows strong search performance across a variety of ann-benchmarks datasets
- HNSW is widely used in industry, having been implemented in several different systems.

ANN:
Approximate nearest neighbor (ANN) is a search algorithm that returns points whose distance from the query is close to the distance from the query to its nearest points. The ANN algorithm is able to solve multi-class classification tasks. The difference between KNN and ANN is that in the prediction phase, all training points are involved in searching k-nearest neighbors in the KNN algorithm, but in ANN this search starts only on a small subset of candidates points. Approximate nearest neighbor (ANN) search is a technique to find points in a data set that are close to a query point, without necessarily finding the exact closest point 1. It works by using some data

structures or algorithms that can reduce the search space or the number of distance comparisons, such as hashing, trees, or graphs 2 3 4. It trades off some accuracy for speed and scalability, especially in high-dimensional data 1 4

Advantages of ANN:

- It is faster and more scalable than exact nearest neighbor search, especially for large datasets.
- It can handle noisy data and arbitrary decision boundaries.
- It is simple to understand and implement, and requires only a few parameters to be tuned.

Disadvantages of ANN:

- It can be **computationally intensive** as the number and dimension of the data points increase. 1 2
- It can have **low accuracy** or **high error** compared to exact nearest neighbor search, depending on the approximation factor. 3
- It can result in **suboptimal solutions** for some problems, such as the traveling salesman problem 4
- It can be **sensitive to the choice of parameters**, such as the number of neighbors, the distance metric, and the hashing function 5

ANN Application Examples:

- **Pattern recognition** – for example, to recognize handwritten digits or characters 1.
- **Statistical classification** – for example, to classify images or texts based on their features 1.
- **Computer vision** – for example, to register or align point clouds or images 1.
- **Content recommendation** – for example, to recommend similar content to users based on their preferences 2.
- **Machine learning** – for example, to speed up the training or inference of neural networks or other models 3 4.

For text/image deduplication, you can use LSH to map documents into buckets based on their similarity, and then compare the documents within each bucket to find and remove duplicates. This can be useful for applications such as text mining, information retrieval, and plagiarism detection. 1 2 3

List of Algorithms:

Principle	Algorithms
k-NN graph	KGraph (KG) [13], SWGraph (SWG) [29, 8], HNSW [28, 8], PyNNDescent (NND) [30], PANNG [2, 19]
tree-based	*FLANN* [32], *BallTree (BT)* [8], Annoy (A) [7], RPForest (RPF) [27], MRPT [17, 1]
LSH	FALCONN (FAL) [5], *MPLSH* [14, 8]
other	Multi-Index Hashing (MIH) [33] (exact Hamming search), FAISS-IVF (FAI) [20] (inverted file)

Additional info of Algorithm principles that support the approximate nearest neighbor search are:

- **Locality-sensitive hashing (LSH):** It hashes the data points into buckets such that similar points are likely to be in the same bucket.
- **Best bin first (BBF):** It uses a priority queue to search the bins in order of their distance from the query point.
- **Balanced box-decomposition tree (BBT):** It partitions the data space into boxes such that each box contains roughly the same number of points.

List of algorithms in approximate nearest neighbor:

- **Annoy:** It uses random projections to build a forest of binary trees.
- **Faiss:** It offers a suite of algorithms for large dataset similarity search, such as LSH, HNSW, and IVF.
- **Flann:** It uses a combination of randomized kd-trees and hierarchical k-means trees.
- **HNSWlib:** It builds a graph of nearest neighbors using a greedy heuristic.
- **Apache Ignite ANN:** It uses a distributed KMeans clustering algorithm to find centroids.
- **NGT-panng:** Yahoo Japan's Neighborhood Graph and Tree for Indexing High-dimensional Data.
- **Pynndescent:** Python implementation of Nearest Neighbor Descent for k-neighbor-graph construction and ANN search.
- **SW-graph(nmslib):** Small world graph ANN search as part of the non-metric space library.

Limitation of KD-trees(KD-trees is a data structure for kNN on low-dimensional vectors):

- Modern embedding models for text and images typically produce high-dimensional vectors of 100 - 1000 elements, or even more. These vector representations present a unique challenge, as it's very difficult to efficiently find nearest neighbors in high dimensions.
- Faced with this difficulty, nearest neighbor algorithms usually sacrifice perfect accuracy to improve their speed. These approximate nearest neighbor (ANN)

algorithms may not always return the true k nearest vectors. But they run efficiently, scaling to large datasets while maintaining good performance.

Limitations of approximate nearest neighbor algorithms
nearest neighbor algorithms usually sacrifice perfect accuracy to improve their speed. These approximate nearest neighbor (ANN) algorithms may not always return the true k nearest vectors. But they run efficiently, scaling to large datasets while maintaining good performance.

k-Nearest Neighbors algorithm
Advantages:
1. It's pretty intuitive and simple
2. It makes no assumptions
3. There is no training step
4. It immediately adapts to changes
5. It naturally lends itself to multi-class problems
6. It can be used for classification & regression tasks: The natural practice of this method is placing input points into appropriate categories, but we can easily extend this algorithm to regression problems. Instead of combining the discrete labels of k-neighbors we have to combine continuous predictions. These predictions can be obtained in different ways, for example, by averaging the values of the k-most similar instances.
7. Only one hyper parameter to be set
8. Flexible to distance choice: can use multiple numeric distance formulas like (Euclidean, Manhattan, etc.) and categorical features (Hamming, etc.).
9. It can provide density estimation
10. It's easy to implement and tune

Disadvantages:
1. Test stage is slow: every test sample, the model has to run through the entire data set to compute distances and then find the nearest neighbors.
2. It usually needs homogenous features
3. It requires to pick the hyper parameter k: The selection of the number of neighbors is a key point and you need to know how this is going to affect your classifier. Choosing a very small k means low bias but high variance, while many voters are involved variance decreases but bias increases.
4. Imbalanced data causes problems
5. It's sensitive to outliers
6. It can't deal with missing values
7. Distance to the closest neighbors might change a lot for different instances
8. K-NN isn't suited for high dimensional data
9. It isn't robust against irrelevant features
10. Redundancy affects performance: If two features or more are dependent, the algorithm will lay a decompensate weight to them without any reason to do that.

Vector Search
Why vector search?

What's the problem?

- 80% to 90% of data is unstructured, that's about 34 trillion gigabytes!
- We keep storing more and more data but businesses can't get valuable insights from it.
- 95% of businesses have issues getting value from their unstructured data.

Pgvector vs redisearch vs FAISS vs Milvus vs Weavaiate:

pgvector is an open-source PostgreSQL extension that supports exact and approximate nearest neighbor search 1
pgvector does not have a built-in deduplication feature, but it supports indexing and querying over vectors with any number of dimensions 1
You can use dimensionality reduction techniques to reduce the size of the vectors and avoid duplicates 1

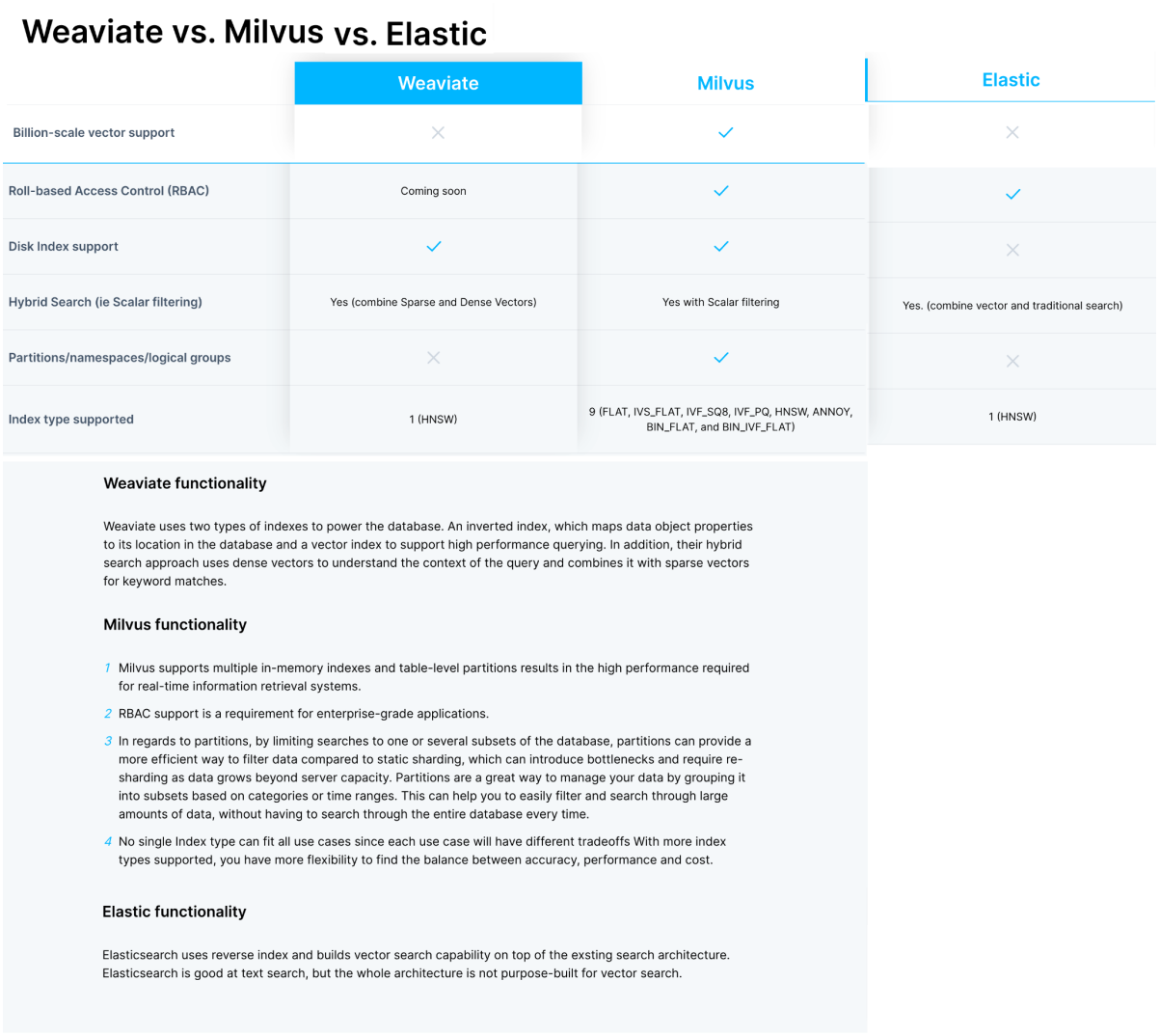
RediSearch is a Redis module that provides full-text search, secondary indexing, and vector similarity search 2
RediSearch supports deduplication by using a unique document ID for each vector. 1
It also provides fast indexing and searching over large datasets 2

FAISS is a library for efficient similarity search and clustering of dense vectors developed by Facebook AI Research 3

FAISS supports various index types and compression methods to reduce the memory footprint of the vectors to support deduplication [2]

Milvus is a distributed vector database that supports multiple index types, partitioning, and scalability [4]
Milvus supports deduplication by using a primary key for each vector [3]
It also supports partitioning, sharding, and replication for scalability and fault tolerance [3]

Weaviate is a cloud-native vector database that supports semantic search, GraphQL queries, and schema inference [5] [6]
Weaviate supports deduplication by using a UUID for each vector [4]
It also supports semantic search, GraphQL queries, and schema inference [4]



Weaviate vs. Milvus vs. Elastic

	Weaviate	Milvus	Elastic
Billion-scale vector support	✗	✓	✗
Roll-based Access Control (RBAC)	Coming soon	✓	✓
Disk Index support	✓	✓	✗
Hybrid Search (ie Scalar filtering)	Yes (combine Sparse and Dense Vectors)	Yes with Scalar filtering	Yes. (combine vector and traditional search)
Partitions/namespaces/logical groups	✗	✓	✗
Index type supported	1 (HNSW)	9 (FLAT, IVS_FLAT, IVF_SQ8, IVF_PQ, HNSW, ANNOY, BIN_FLAT, and BIN_IVF_FLAT)	1 (HNSW)

Weaviate functionality

Weaviate uses two types of indexes to power the database. An inverted index, which maps data object properties to its location in the database and a vector index to support high performance querying. In addition, their hybrid search approach uses dense vectors to understand the context of the query and combines it with sparse vectors for keyword matches.

Milvus functionality

1. Milvus supports multiple in-memory indexes and table-level partitions results in the high performance required for real-time information retrieval systems.
2. RBAC support is a requirement for enterprise-grade applications.
3. In regards to partitions, by limiting searches to one or several subsets of the database, partitions can provide a more efficient way to filter data compared to static sharding, which can introduce bottlenecks and require re-sharding as data grows beyond server capacity. Partitions are a great way to manage your data by grouping it into subsets based on categories or time ranges. This can help you to easily filter and search through large amounts of data, without having to search through the entire database every time.
4. No single Index type can fit all use cases since each use case will have different tradeoffs With more index types supported, you have more flexibility to find the balance between accuracy, performance and cost.

Elastic functionality

Elasticsearch uses reverse index and builds vector search capability on top of the exsting search architecture. Elasticsearch is good at text search, but the whole architecture is not purpose-built for vector search.

Purpose-built

What's your vector database for?

A vector database is a fully managed solution for storing, indexing, and searching across a massive dataset of unstructured data that leverages the power of embeddings from machine learning models. A vector database should have the following features:

1. Scalability and tunability
2. Multi-tenancy and data isolation
3. A complete suite of APIs
4. An intuitive user interface/administrative console

	Weaviate	Milvus	Elastic
Purpose-built for Vectors	✓	✓	✗
Database rollback	✓	✓	✗
Tunable consistency	✓	✓	✗
Support for both stream and batch of vector data	✓	✓	✓
Binary Vector support	✓	✓	✓
Multi-language SDK	Python, Java, Go	Python, Java, Go, C++, Node.js	Python, Java, Go, C++, Node.js, Rust, Ruby, .NET (C#), PHP, Perl

what's right for me?

Weaviate

Weaviate is maintained by a single commercial company offering a cloud version of Weaviate.
License: BSD-3-Clause license

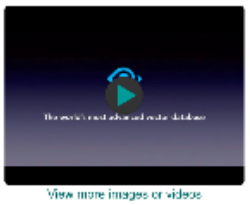
Milvus

Milvus is a fully open source and independent project, maintained by a number of companies and individuals, some of whom also offer commercial services and support. Graduate of LF AI Data.
License: Apache-2.0 license

Elasticsearch

Elasticsearch is built on Apache Lucene and was first released in 2010 by Elastic.
License: Dual-licensed Server Side Public License (SSPL) or the Elastic License

What's the difference between Faiss, Milvus, Pinecone, and pgvector? Compare Faiss vs. Milvus vs. Pinecone vs. pgvector in 2023 by cost, reviews, features, integrations, deployment, target market, support options, trial offers, training options, years in business, region, and more using the chart below.

	Faiss	Milvus	Pinecone	pgvector
	View Product	View Product	View Product	View Product
Average Ratings	0 Ratings	0 Ratings	0 Ratings	0 Ratings
	No User Reviews. Be the first to provide a review:	No User Reviews. Be the first to provide a review:	No User Reviews. Be the first to provide a review:	No User Reviews. Be the first to provide a review:
	Write a Review	Write a Review	Write a Review	Write a Review
API Access				
Has API		✓	✓	
Screenshots	View All	View All	View All	View All
				
	View more images or videos	View more images or videos	View more images or videos	View more images or videos
Integrations				
Haystack	✓		✓	
Amazon S3		✓		
Apache Kafka		✓		
Apache Pulsar		✓		
ChatGPT			✓	
ChatGPT Plus			✓	
Corel			✓	
Databricks Lakehouse			✓	
Docker		✓		
Elasticsearch			✓	
Show More Integrations				
	Explore All 1 Integration	Explore All 8 Integrations	Explore All 14 Integrations	Explore All 2 Integrations
Pricing Details				
	Free	Free	No price information available.	Free
Free Trial				
Free Version	✓	✓		✓
Deployment				
SaaS	✓	✓	✓	✓
iPhone				
iPad				
Android				
Windows				
Mac				
Linux				
Support				
Business Hours				
24/7 Live Support				
Online		✓	✓	
Training				
Documentation	✓	✓	✓	✓
Webinars				
Live Online		✓		
In Person		✓		
Vendor Details				

Locality-sensitive hashing(LSH):

Locality-sensitive hashing (LSH) is an algorithmic technique that hashes similar input items into the same buckets with high probability. It is used for finding near-duplicate or similar data in large datasets[1].

LSH works by applying a set of hash functions to each item, such that the probability of two items having the same hash value is proportional to their similarity. The hash functions are chosen to be **locality-sensitive**, meaning that they preserve the similarity of the items in the hash space[2].

LSH can be used for **approximate nearest neighbor (ANN) search**, which is the problem of finding the most similar items to a given query item in a dataset. LSH can reduce the time and space complexity of ANN search by avoiding pairwise comparisons of all items and using compact hash codes instead[3].

LSH can also be used for **data clustering**, which is the problem of grouping similar items into clusters. LSH can reduce the complexity of clustering by partitioning the data into buckets and then applying a clustering algorithm on each bucket[4].

LSH algorithms for deduplication:

Below are commonly used:

- **MinHash**: This algorithm hashes each document to a vector of integers by applying a set of hash functions to its shingles. The similarity of two documents is estimated by the fraction of hash values that are equal in their vectors[1][2].
- **SimHash**: This algorithm hashes each document to a binary vector by applying a set of hash functions to its features. The similarity of two documents is estimated by the Hamming distance between their vectors[3].
- **Hyperplane Hashing**: This algorithm hashes each document to a binary vector by applying a set of random hyperplanes to its features. The similarity of two documents is estimated by the fraction of bits that are equal in their vectors[3].

Byte-level LSH:

Byte-level LSH is a technique for finding near-duplicate text documents by hashing their byte sequences and comparing the hashes. It is based on the idea that similar documents will have similar hashes with high probability.

One way to implement byte-level LSH is to use **minhash** to generate a fingerprint of each document and then apply a **locality sensitive hash** to the fingerprint. This reduces the dimensionality of the fingerprints and allows for efficient comparison of large sets of documents[1].

Another way to implement byte-level LSH is to use **n-gram shingles** to represent each document as a set of overlapping substrings and then compute the **Jaccard similarity** of the shingle sets. This measures the overlap of the shingle sets and can capture semantic similarity as well as byte-level similarity[2].

FAISS and Milvus Speed Benchmarking (Flat and HNSW)

https://github.com/milvus-io/milvus/discussions/4939

The number below show queries per second given a certain document store, when loaded with a certain number of documents.

	FAISS Flat	FAISS HNSW	Milvus Flat	Milvus HNSW
1000	40.41	44.32	40.48	40.39
10000	28.03	42.12	38.13	38.15
100000	6.62	40.68	28.30	28.08
500000	1.54	39.43	15.30	24.50

While Milvus Flat seems significantly faster than FAISS Flat, Milvus HNSW does not match the near constant speed that FAISS HNSW has. It is in fact only about as fast as Milvus Flat for 1k, 10k and 100k and is only faster at 500k. For reference, here are the mAP scores for the same configurations.

	FAISS Flat	FAISS HNSW	Milvus Flat	Milvus HNSW
1000	92.95	92.95	92.95	92.95
10000	89.87	89.50	89.87	89.87
100000	86.55	84.33	86.55	86.55
500000	80.86	75.73	80.86	74.86

Differences between pgvector vs redisearch:

- pgvector is a PostgreSQL extension that can store and query vectors as columns in relational tables[1][4]. redisearch is a Redis module that can store and query vectors as fields in Redis hashes or JSON documents[2][3].
- pgvector supports two indexing methods: FLAT and IVFFLAT, which are based on brute-force search or inverted file search[4]. redisearch supports three indexing methods: FLAT, HNSW, and PQ, which are based on brute-force search, hierarchical navigable small world graphs, or product quantization[3].
- pgvector can use partitioning or partial indexing to handle nearest neighbor queries with a WHERE clause[4]. redisearch can use tags or filters to handle nearest neighbor queries with additional conditions[3].
- pgvector is a PostgreSQL extension that can store and query vectors, which are numerical representations of data[12345]. It can perform similarity search by using some distance metrics, such as Euclidean distance or cosine similarity, to compare the vectors and return the most similar ones[124].
- To use pgvector, you need to install it on your database server and enable it in your database[125]. Then, you can create a table with a column of type vector and specify the size of the vector[13]. You can insert or update data into the table as usual, and use the operators `<->` or `<=>` to query the vectors by distance or similarity[123]. You can also create an index on the vector column to speed up the search, and adjust some parameters to improve the accuracy or performance[4].
- redisearch is a Redis module that adds **secondary indexing**, **full-text search**, and a **query language** to Redis[12345]. It allows you to run complex queries on your Redis data, such as multi-field queries, aggregation, exact phrase matching, numeric filtering, and geo-filtering[234].
- To use redisearch, you need to create an index on your Redis data, either as hashes or JSON documents[123]. Then, you can use the `FT.SEARCH` command

to query the index by using a query syntax that supports various operators and modifiers[2][3]. You can also use the `FT.AGGREGATE` command to perform aggregations and transformations on the query results[2].

is LSH available in Milvus:

No

LSH really important? if yes why it's not there in Milvus and Weavaiate because they are also intended to be used in LLMs which are meant to give close vector results.

LSH is a technique that can speed up the search for similar items in a large data set by hashing them into buckets[1]. However, it is not the only technique that can do that. There are other methods, such as **HNSW**, **PQ**, **IVFFLAT**, etc., that can also achieve fast and accurate similarity search[2][3].

Milvus and Weaviate are both **vector databases** that can store and query vectors, which are numerical representations of data[2][3]. They do not use LSH, but they use other methods to index and search vectors efficiently. For example, Milvus supports FLAT, IVFFLAT, HNSW, and PQ indexing methods[2]. Weaviate supports HNSW indexing method[3].

A Data Scientist's Guide to Picking an Optimal Approximate Nearest-Neighbor Algorithm | by Braden Riggs | GSI Technology | Medium

Understanding Locality Sensitive Hashing(LSH): A Powerful Technique for Similarity Search. | by Sarthak Joshi | Jul, 2023 | Medium

Top Shortlisted

- FAISS - LSH
- Milvus
- Weaviate
- redisearch

Additional info:
- Microsoft uses ANN

Image Dedupe to remove duplicates using vector embeddings
[Image deduplication using OpenAI's CLIP and Community Detection]

How to run:
- Load the models from the hugging face hub.
- Load the images under our directory and generate our embeddings
- Do Cluster (use dbscan)
- Cluster formed are duplicates

Use `imagededup` Python Library
It is a python package that simplifies the task of finding exact and near duplicates in an image collection.

How to run:
- Import perceptual hashing method
- Generate encodings for all images in an image directory
- Find duplicates using the generated encodings
- Plot duplicates obtained for a given file (eg: 'ukbench00120.jpg') using the duplicates dictionary