Deep Dive on Vector Databases and AI

Tim Spann @ Zilliz

zilliz

milvus

Tim Spann

Principal Developer
Advocate, Zilliz

tim.spann@zilliz.com
https://www.linkedin.com/in/timothyspann/
https://x.com/PaaSDev

 |

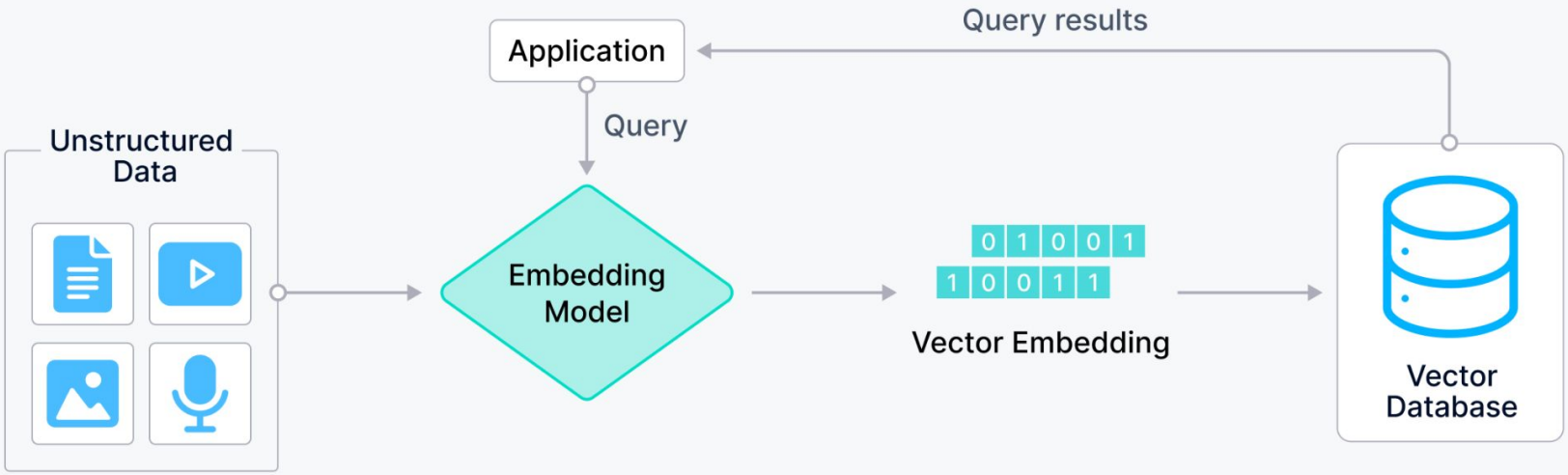
zilliz

It will be held by "Linux Legion Club" of The PES University.
The session will be held in AMA manner that is:

- 05 min : intro
- 30+ mins : presentation
- left over time : Q&A by audience

Vector Database : making sense of unstructured data

A vector database stores embedding vectors and allows for semantic retrieval of various types of unstructured data.



zilliz

Milvus: The most widely-adopted vector database

Milvus is an **Open-Source Vector Database** to **store**, **index**, **manage,** and **use** the massive number of **embedding vectors** generated by deep neural networks and LLMs.

milvus
LF AI &DATA

400+
contributors

29K+
stars

66M+
docker pulls

2.7K+
forks

zilliz

Rich functionality

Dynamic Schema

Float, Binary and
Sparse Vector

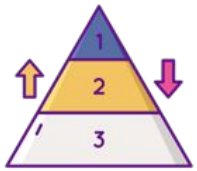
Tag + Vector
Optimized filtering

Hybrid Search
Sparse + Dense

RBAC
TLS, Encryption

Million level
Tenants support

Disk based Index

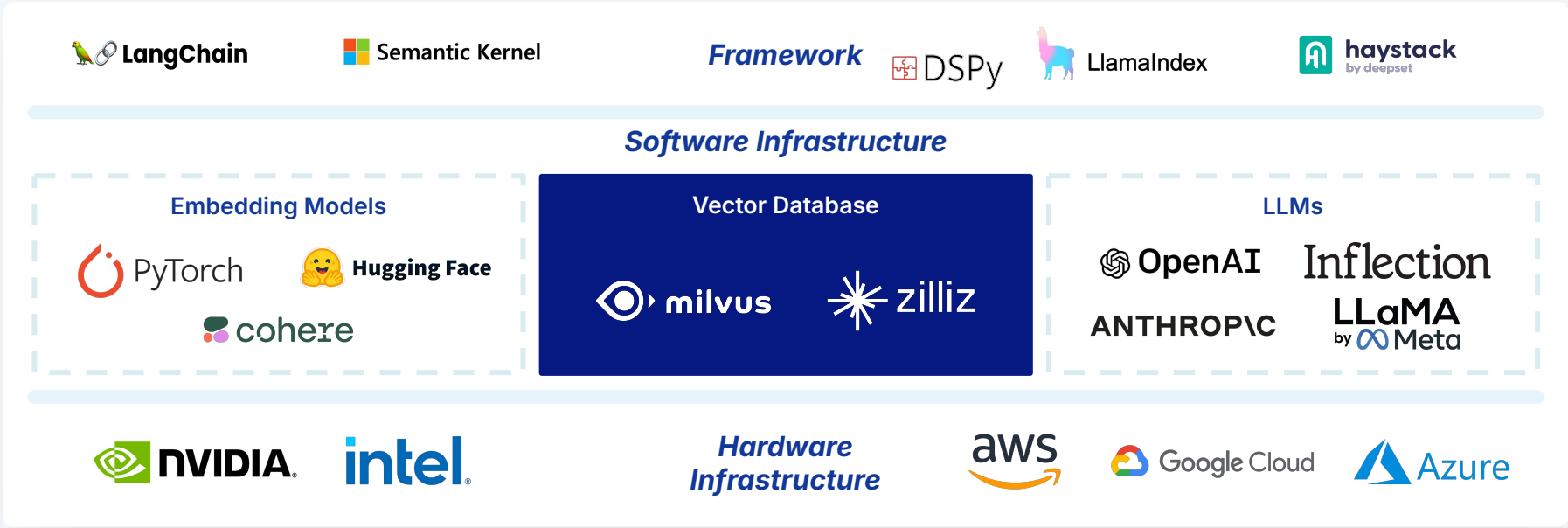


Tiered Storage

Support bulk import

GPU Support
Intel + ARM Cpu Support

Well-connected in LLM infrastructure to enable RAG use cases

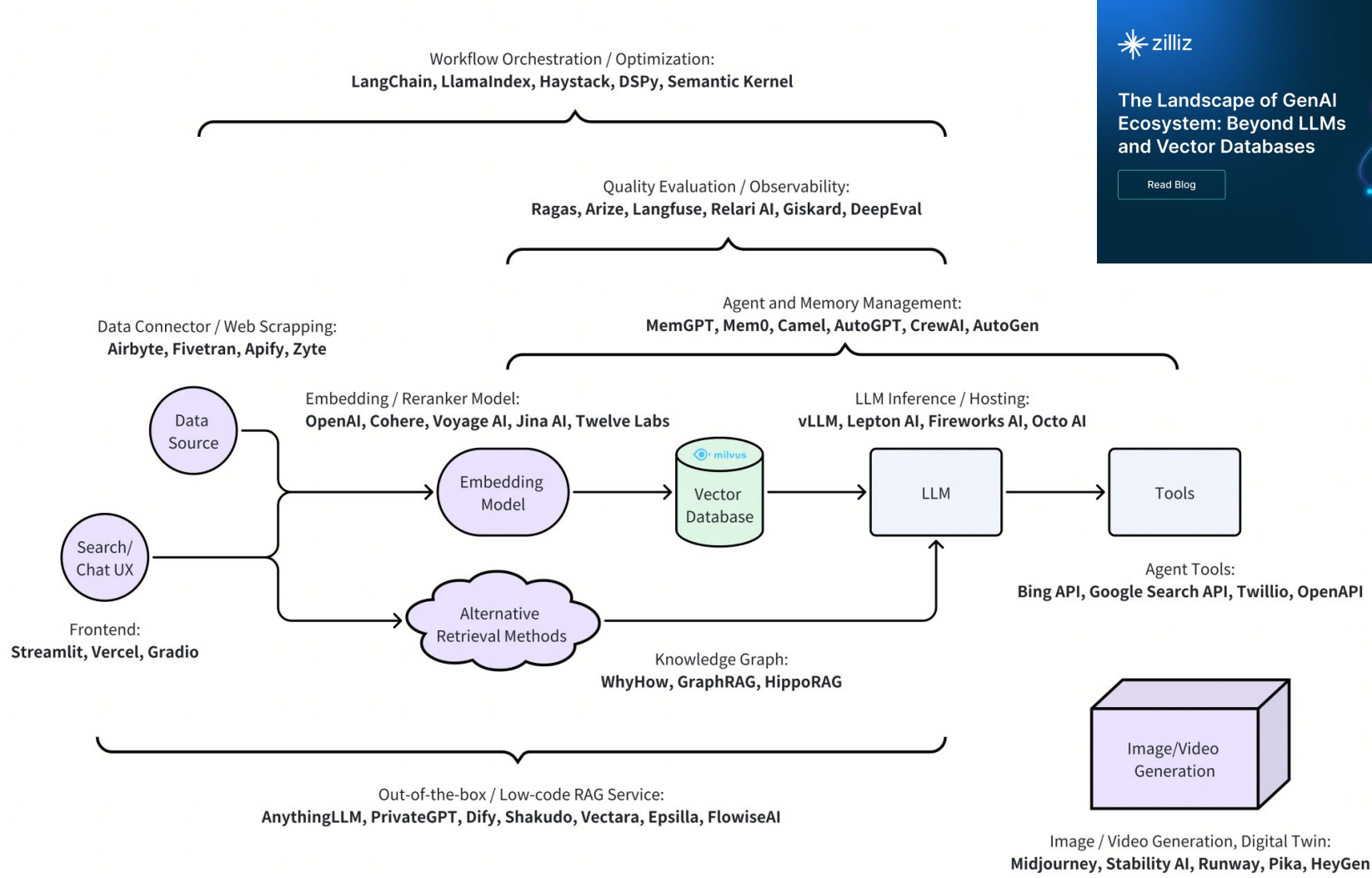


zilliz

zilliz

The Landscape of GenAI Ecosystem: Beyond LLMs and Vector Databases

Read Blog



zilliz

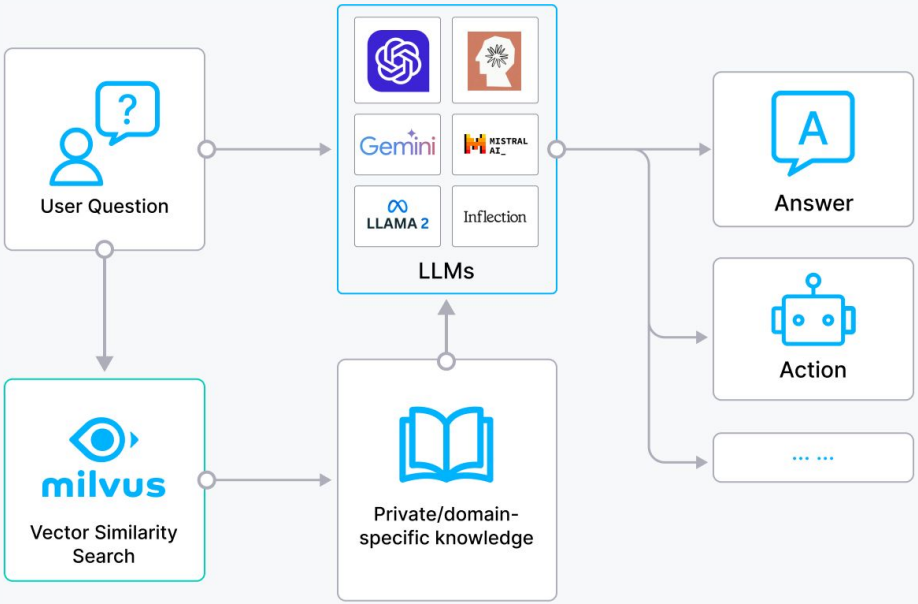
Comparison on functionalities

Feature	Milvus Lite	Milvus Standalone	Milvus Distributed
SDK / Client Lirary	Python gRPC	Python Go Java Node.js C# RESTful	Python Java Go Node.js C# RESTful
Data types	Dense Vector Sparse Vector Binary Vector Boolean Integer Floating Point VarChar Array JSON	Dense Vector Sparse Vector Binary Vector Boolean Integer Floating Point VarChar Array JSON	Dense Vector Sparse Vector Binary Vector Boolean Integer Floating Point VarChar Array JSON
Search capabilities	Vector Search (ANN Search) Metadata Filtering Range Search Scalar Query Get Entities by Primary Key Hybrid Search	Vector Search (ANN Search) Metadata Filtering Range Search Scalar Query Get Entities by Primary Key Hybrid Search	Vector Search (ANN Search) Metadata Filtering Range Search Scalar Query Get Entities by Primary Key Hybrid Search
CRUD operations	✓	✓	✓
Advanced data management	N/A	Access Control Partition Partition Key	Access Control Partition Partition Key Physical Resource Grouping
Consistency Levels	Strong	Strong Bounded Staleness Session Eventual	Strong Bounded Staleness Session Eventual

Retrieval-Augmented Generation (RAG)

A technique that combines the strength of retrieval-based and generative models:

- Improve accuracy and relevance
- Eliminate hallucination
- Provide domain-specific knowledge



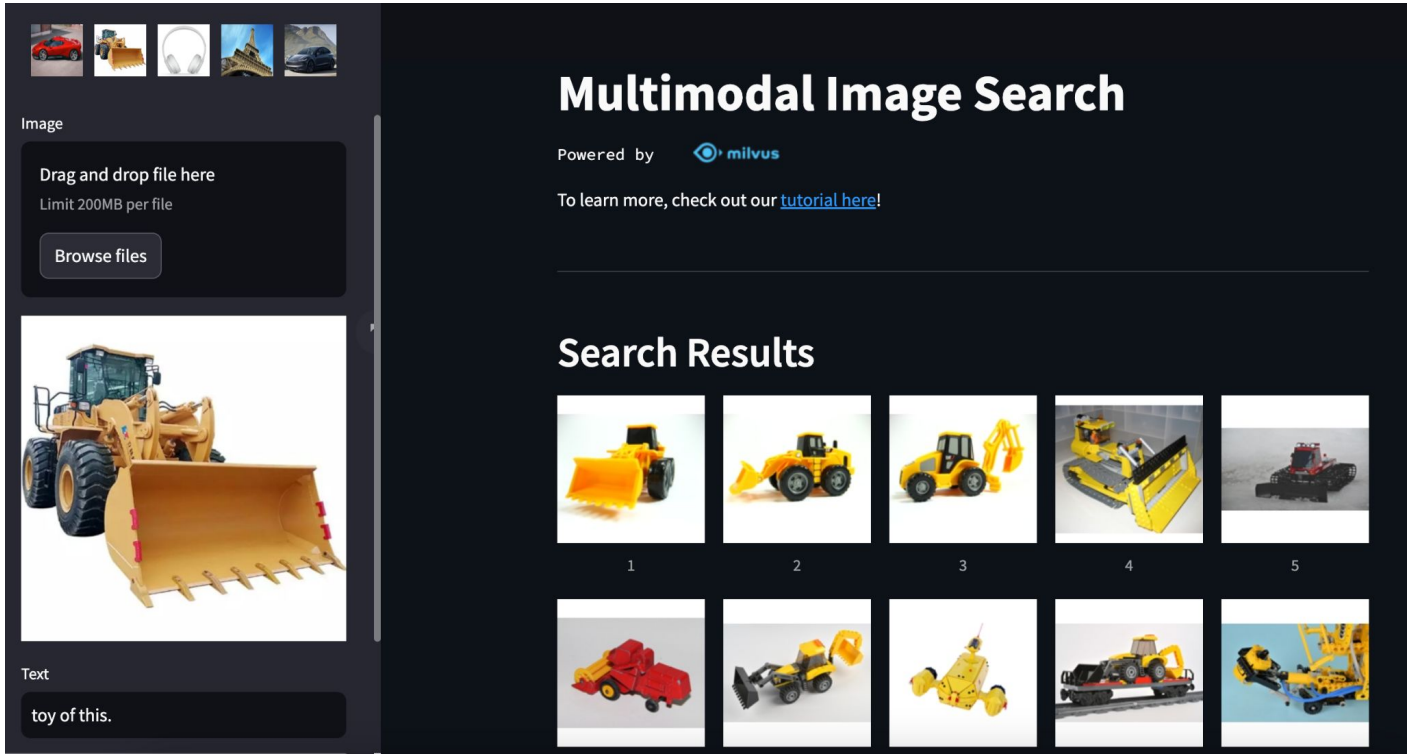
zilliz

D

Demos

zilliz

Multi-Modal Search



multimodal-demo.milvus.io

 |

zilliz

A

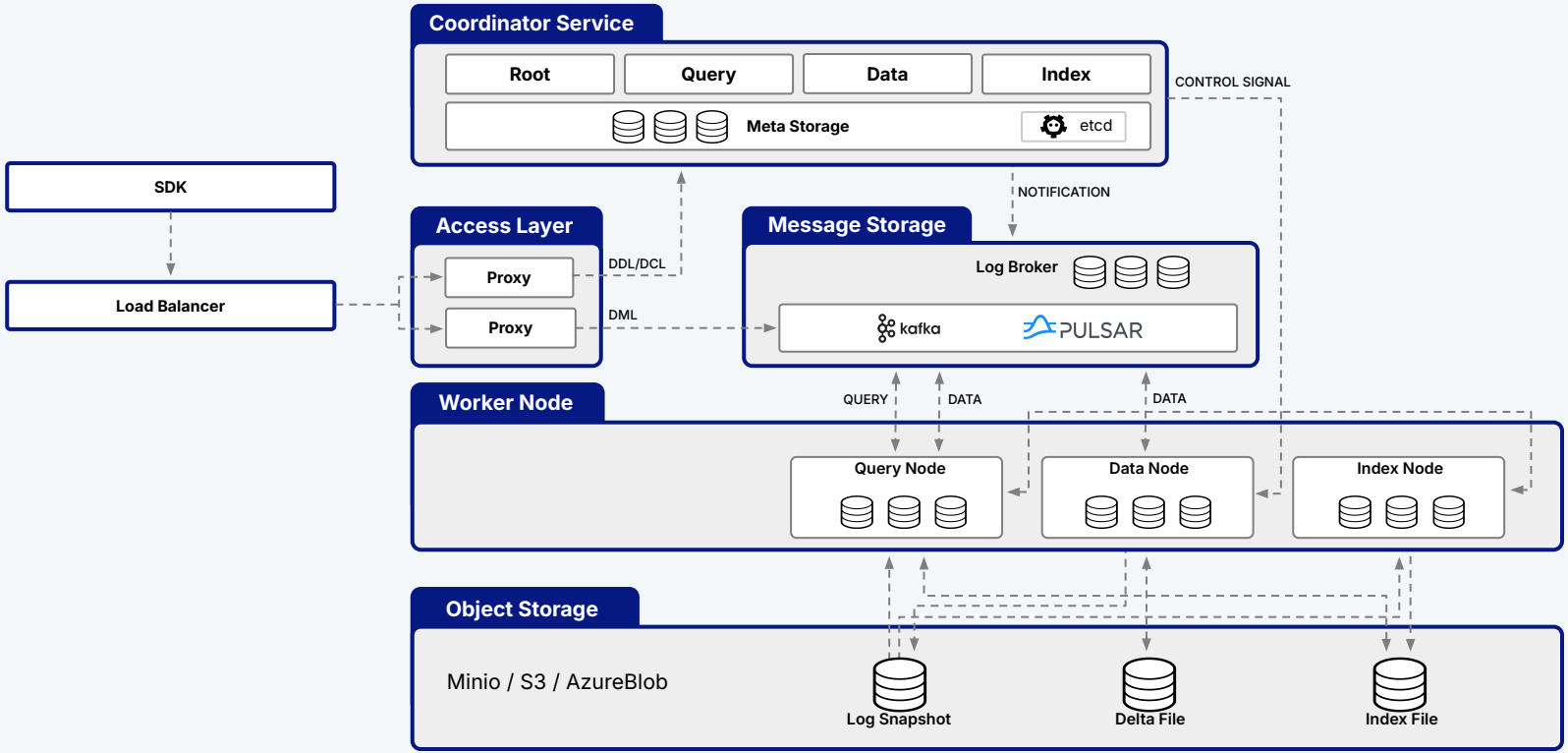
Architecture

Learn

https://zilliz.com/learn/Retrieval-Augmented-Generation

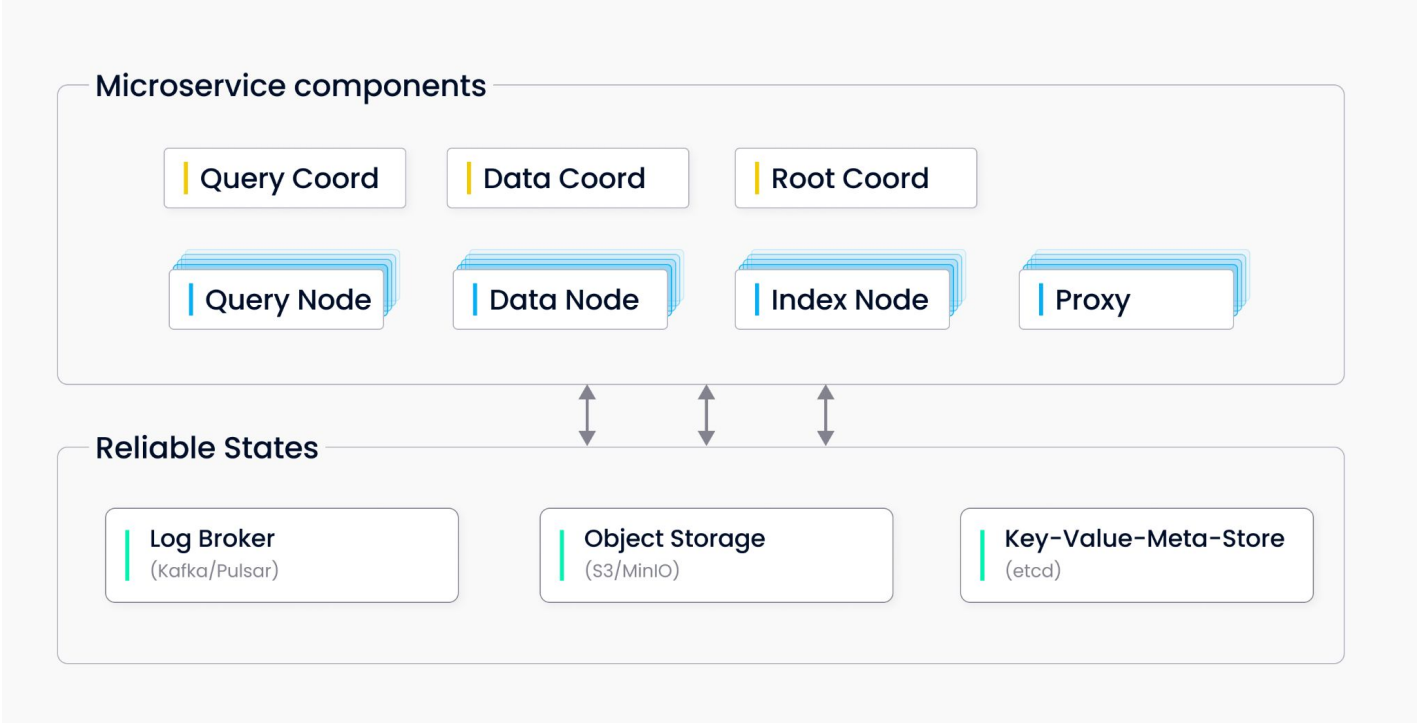
https://zilliz.com/blog/scale-search-with-milvus-handle-massive-datasets-with-ease

Milvus Architecture



 |

Distributed Architecture



zilliz

Dynamic Scaling 🚀

Stateless components for **Easy Scaling**

- **Query**, **Index**, and **Data Nodes** can be scaled **independently**
- Allows for **optimized resource allocation** based on workload characteristics

Data sharding across multiple nodes

- **Distributes** large **datasets** across multiple **Data Nodes**
- Enables **parallel processing** for improved query performance

Horizontal Pod Autoscaler (**HPA**)

- **Automatically scales** up and down
- **Custom metrics** can be used (e.g., query latency, throughput)

zilliz

Stateless Architecture

Stateless Components

All Milvus components are deployed **Stateless**.

Object Storage

Milvus relies on **Object Storage** (MinIO, S3, etc) for data **persistence**.
Vectors are stored in **Object Storage**, **Metadata** is in **etcd.**

Scaling and Failover

Scaling and failover **don't** involve traditional **data rebalancing.** When **new pods** are added or existing ones fail, they can immediately start handling requests by **accessing data** from the **shared object storage.**

zilliz

Different Consistency levels

Ensures every node or replica has the same view of data at a given time.

- **Strong**: Guaranteed up-to-date reads, highest latency
- **Bounded**: Reads may be slightly stale, but within a time bound
- **Session**: Consistent reads within a session, may be stale across sessions
- **Eventually**: Lowest latency, reads may be stale

Trade Offs

- **Strong** consistency for **critical applications** requiring accurate results
- **Eventually** consistency for **high-throughput**, latency-sensitive apps

 |

zilliz

S

Schemas

Schemas

Properties	Description	Note
`name`	Name of the field in the collection to create	Data type: String. Mandatory
`dtype`	Data type of the field	Mandatory
`description`	Description of the field	Data type: String. Optional
`is_primary`	Whether to set the field as the primary key field or not	Data type: Boolean (`true` or `false`). Mandatory for the primary key field

https://milvus.io/docs/schema.md

zilliz

Schemas

Properties	Description	Note
`auto_id` (Mandatory for primary key field)	Switch to enable or disable automatic ID (primary key) allocation.	`True` or `False`
`max_length` (Mandatory for VARCHAR field)	Maximum length of strings allowed to be inserted.	[1, 65,535]
`dim`	Dimension of the vector	Data type: Integer ∈[1, 32768]. Mandatory for a dense vector field. Omit for a sparse vector field.
`is_partition_key`	Whether this field is a partition-key field.	Data type: Boolean (`true` or `false`).

zilliz

Data Types

Primary key field supports:

INT64: numpy.int64
VARCHAR: VARCHAR

Scalar field supports:

BOOL: Boolean (true or false)
INT8: numpy.int8
INT16: numpy.int16
INT32: numpy.int32
INT64: numpy.int64
FLOAT: numpy.float32
DOUBLE: numpy.double
VARCHAR: VARCHAR

JSON: JSON

Array: Array

 |

zilliz

Data Types

JSON as a composite data type is available. A JSON field comprises key-value pairs. Each key is a string, and a value can be a number, string, boolean value, array, or list. For details, refer to JSON: a new data type.

Vector field supports:

BINARY_VECTOR: Stores binary data as a sequence of 0s and 1s, used for compact feature representation in image processing and information retrieval.

FLOAT_VECTOR: Stores 32-bit floating-point numbers, commonly used in scientific computing and machine learning for representing real numbers.

FLOAT16_VECTOR: Stores 16-bit half-precision floating-point numbers, used in deep learning and GPU computations for memory and bandwidth efficiency.

BFLOAT16_VECTOR: Stores 16-bit floating-point numbers with reduced precision but the same exponent range as Float32, popular in deep learning for reducing memory and computational requirements without significantly impacting accuracy.

SPARSE_FLOAT_VECTOR: Stores a list of non-zero elements and their corresponding indices, used for representing sparse vectors. For more information, refer to Sparse Vectors.

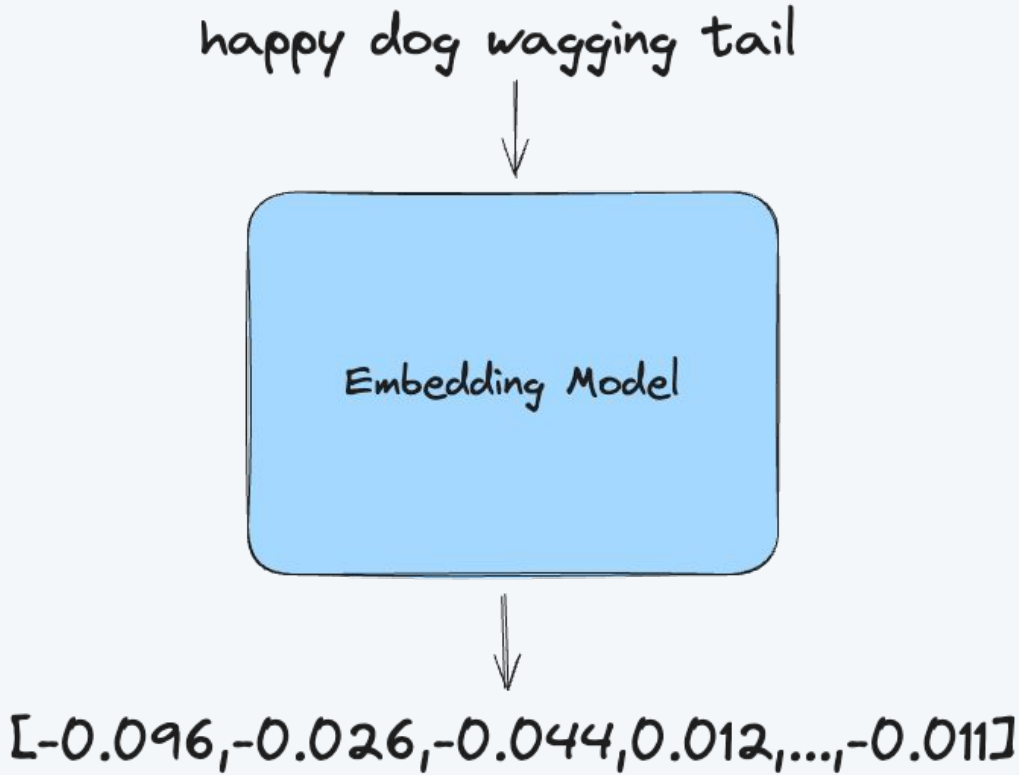
zilliz

E

Embedding Models

zilliz

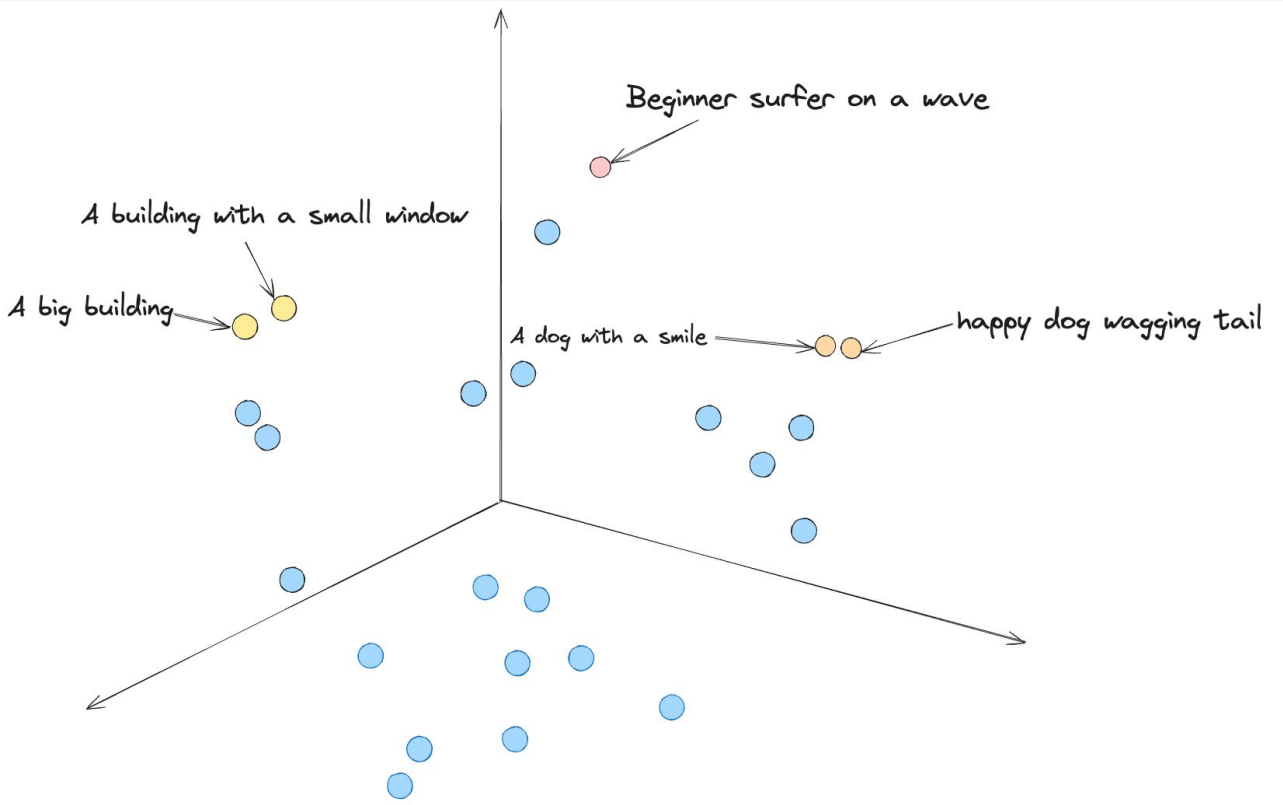
Vector Embedding



 |

zilliz

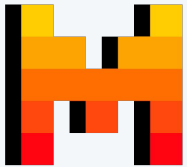
Vector Space



 |

zilliz

Choose Your Embedding Function



Embedding Function	Type	API or Open-sourced
openai	Dense	API
sentence-transformer	Dense	Open-sourced
bm25	Sparse	Open-sourced
Splade	Sparse	Open-sourced
bge-m3	Hybrid	Open-sourced
voyageai	Dense	API
jina	Dense	API
cohere	Dense	API
Instructor	Dense	Open-sourced
Mistral AI	Dense	API
Nomic	Dense	API
mGTE	Hybrid	Open-sourced

zilliz

Further Dive

https://zilliz.com/learn/generative-ai

https://zilliz.com/learn/what-are-binary-vector-embedding

Indexing

zilliz

Choose Your Own Index

Picking a vector index

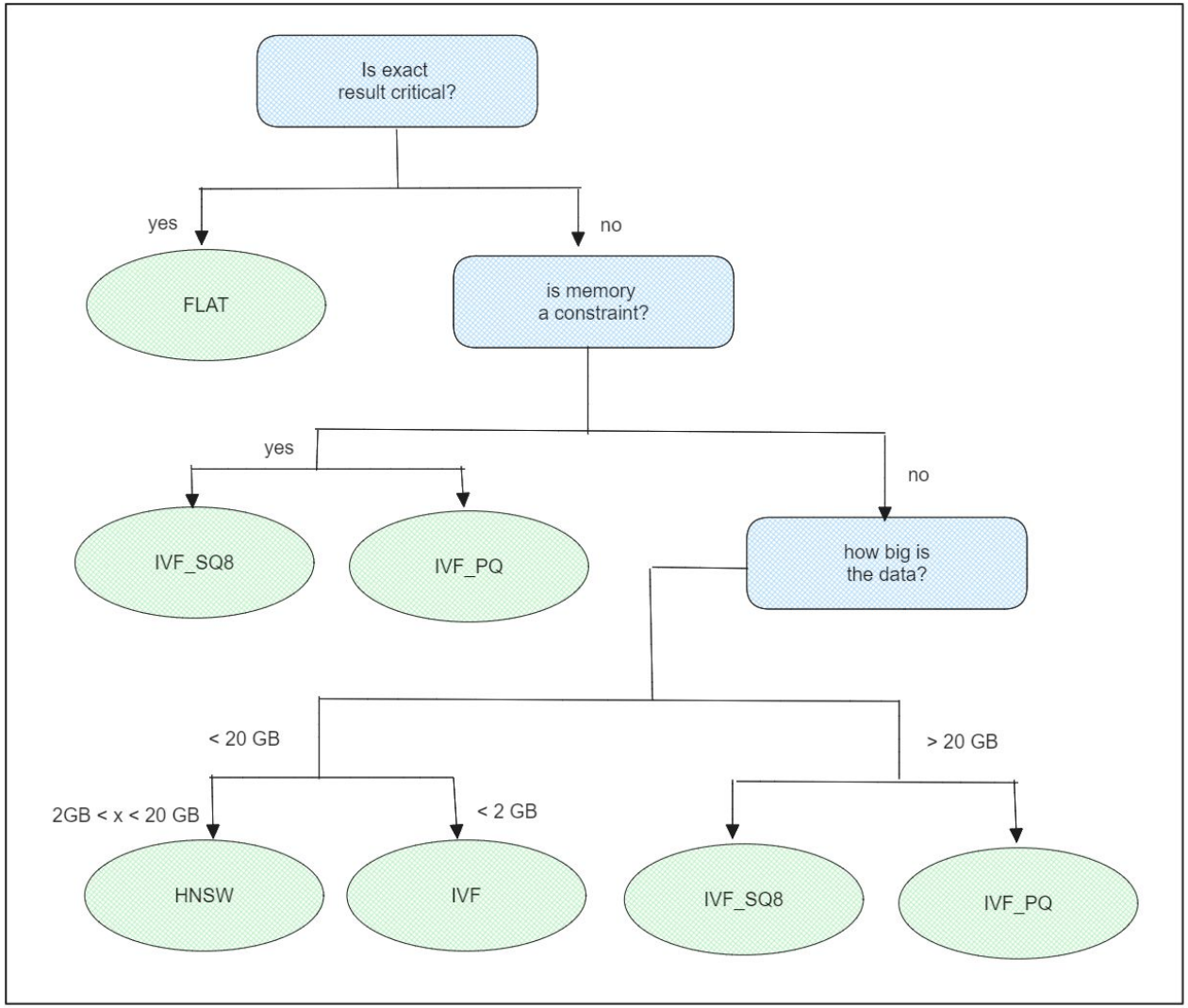
So, how exactly do we choose the right vector index? This is a fairly open-ended question, but one key principle to remember is that the right index will depend on your application requirements. For example: are you primarily interested in query speed (with a static database), or will your application require a lot of inserts and deletes? Do you have any constraints on your machine type, such as limited memory or CPU? Or perhaps the domain of data that you'll be inserting will change over time? All of these factors contribute to the most optimal index type to use.

https://zilliz.com/learn/choosing-right-vector-index-for-your-project

zilliz

Category	Index	Accuracy	Latency	Throughput	Index Time	Cost
Graph-based	Cagra (GPU)	High	Low	Very High	Fast	Very High
	HNSW	High	Low	High	Slow	High
	DiskANN	High	High	Mid	Very Slow	Low
Quantization-based or cluster-based	ScaNN	Mid	Mid	High	Mid	Mid
	IVF_FLAT	Mid	Mid	Low	Fast	Mid
	IVF + Quantization	Low	Mid	Mid	Mid	Low

zilliz



milvus

 |

zilliz

Picking an Index

- 100% Recall – Use `FLAT` search if you need 100% accuracy
- 10MB < `index_size` < 2GB – Standard IVF
- 2GB < `index_size` < 20GB – Consider `PQ` and `HNSW`
- 20GB < `index_size` < 200GB – Composite Index, `IVF_PQ` or `HNSW_SQ`
- Disk-based indexes

 |

zilliz

Indexes

Most of the vector index types supported by Milvus use approximate nearest neighbors search (ANNS),

- **HNSW**: HNSW is a graph-based index and is best suited for scenarios that have a high demand for search efficiency. There is also a GPU version **GPU_CAGRA**, thanks to Nvidia's contribution.
- **FLAT**: FLAT is best suited for scenarios that seek perfectly accurate and exact search results on a small, million-scale dataset. There is also a GPU version **GPU_BRUTE_FORCE**.
- **IVF_FLAT**: IVF_FLAT is a quantization-based index and is best suited for scenarios that seek an ideal balance between accuracy and query speed. There is also a GPU version **GPU_IVF_FLAT**.
- **IVF_SQ8**: IVF_SQ8 is a quantization-based index and is best suited for scenarios that seek a significant reduction on disk, CPU, and GPU memory consumption as these resources are very limited.
- **IVF_PQ**: IVF_PQ is a quantization-based index and is best suited for scenarios that seek high query speed even at the cost of accuracy. There is also a GPU version **GPU_IVF_PQ**.

Indexes Continued.

- **SCANN**: SCANN is similar to IVF_PQ in terms of vector clustering and product quantization. What makes them different lies in the implementation details of product quantization and the use of SIMD (Single-Instruction / Multi-data) for efficient calculation.
- **DiskANN**: Based on Vamana graphs, DiskANN powers efficient searches within large datasets.

Indexing Strategies

- Tree based
- Graph based
- Hash based
- Cluster based

100% recall: This one is fairly simple - use `FLAT` search if you need 100% accuracy. All efficient data structures for vector search perform ***approximate*** nearest neighbor search, meaning that there's going to be a loss of recall once the index size hits a certain threshold.

`index_size` < 10MB: If your total index size is tiny (fewer than 5k 512-dimensional `float32` vectors), just use `FLAT` search. The overhead associated with index building, maintenance, and querying is simply not worth it for a tiny dataset.

10MB < `index_size` **< 2GB**: If your total index size is small (fewer than 1M 512-dimensional `float32` vectors), my recommendation is to go with a standard inverted-file index (e.g. `IVF`). An inverted-file index can reduce the search scope by around an order of magnitude while still maintaining fairly high recall.

2GB < `index_size` **< 20GB**: Once you reach a mid-size index (fewer than 10M 512-dimensional `float32` vectors), you'll want to start considering other `PQ` and `HNSW` index types. Both will give you reasonable query speed and throughput, but `PQ` allows you to use significantly less memory at the expense of low recall, while `HNSW` often gives you 95%+ recall at the expense of high memory usage - around 1.5x the total size of your index. For dataset sizes in this range, composite `IVF` indexes (`IVF_SQ`, `IVF_PQ`) can also work well, but I would use them only if you have limited compute resources.

20GB < `index_size` **< 200GB**: For large datasets (fewer than 100M 512-dimensional `float32` vectors), I recommend the use of ***composite indexes***: `IVF_PQ` for memory-constrained applications and `HNSW_SQ` for applications that require high recall. A composite index is an indexing technique combining multiple vector search strategies into a single index. This technique effectively combines the best of both indexes; `HNSW_SQ`, for example, retains most of `HNSW`'s base query speed and throughput but with a significantly reduced index size. We won't dive too deep into composite indexes here, but [FAISS's documentation](https://github.com/facebookresearch/faiss/wiki/Faiss-indexes-(composite) provides a great overview for those interested.

One last note on Annoy - we don't recommend using it simply because it fits into a similar category as HNSW since, generally speaking, it is less performant. Annoy is the most uniquely named index, so it gets bonus points there.

A word on disk indexes

Another option we haven't dove into explicitly in this blog post is disk-based indexes. In a nutshell, disk-based indexes leverage the architecture of NVMe disks by colocating individual search subspaces into their own NVMe page. In conjunction with zero seek latency, this enables efficient storage of both graph- and tree-based vector indexes.

These index types are becoming increasingly popular since they enable the storage and search of billions of vectors on a single machine while maintaining a reasonable performance level. The downside to disk-based indexes should be obvious as well. Because disk reads are significantly slower than RAM reads, disk-based indexes often experience increased query latencies, sometimes by over 10x! If you are willing to sacrifice latency and throughput for the ability to store billions of vectors at minimal cost, disk-based indexes are the way to go. Conversely, if your application requires high performance (often at the expense of increased compute costs), you'll want to stick with `IVF_PQ` or `HNSW_SQ`.

Wrapping up

In this post, we covered some of the vector indexing strategies available. Given your data size and compute limitations, we provided a simple flowchart to help determine the optimal strategy. Please note that this flowchart is a general guideline, not a hard-and-fast rule. Ultimately, you'll need to understand the strengths and weaknesses of each indexing option, as well as whether a composite index can help you squeeze out the last bit of performance your application needs. All these index types are freely available to you in Milvus, so you can experiment as you see fit. Go out there and experiment!

Quantization Based Indexes

https://medium.com/@tspann/how-good-is-quantization-in-milvus-6d224b5160b0

https://zilliz.com/learn/scalar-quantization-and-product-quantization?source=post_page-----6d224b5160b0----------------------------

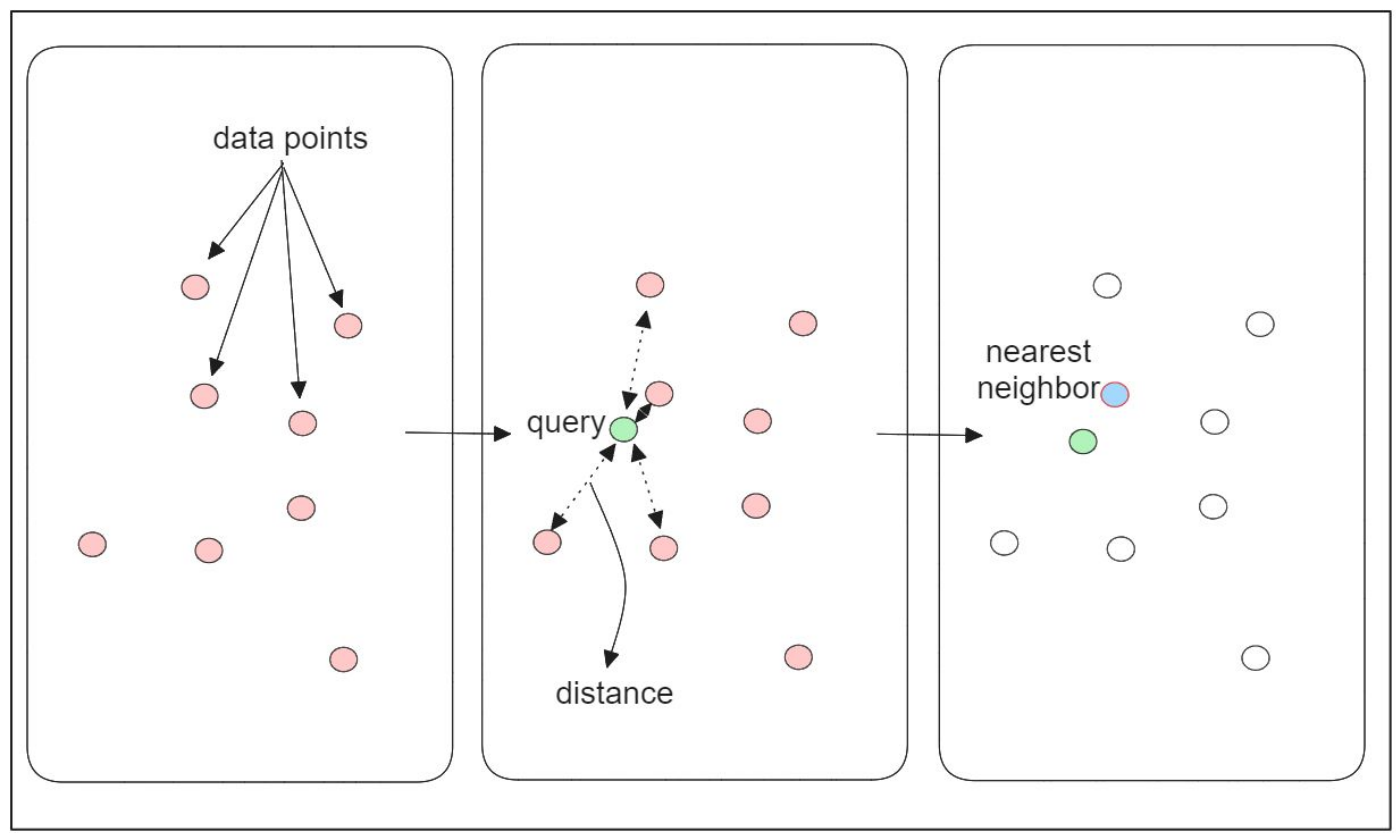
zilliz

FLAT

 |

zilliz

FLAT Index



 |

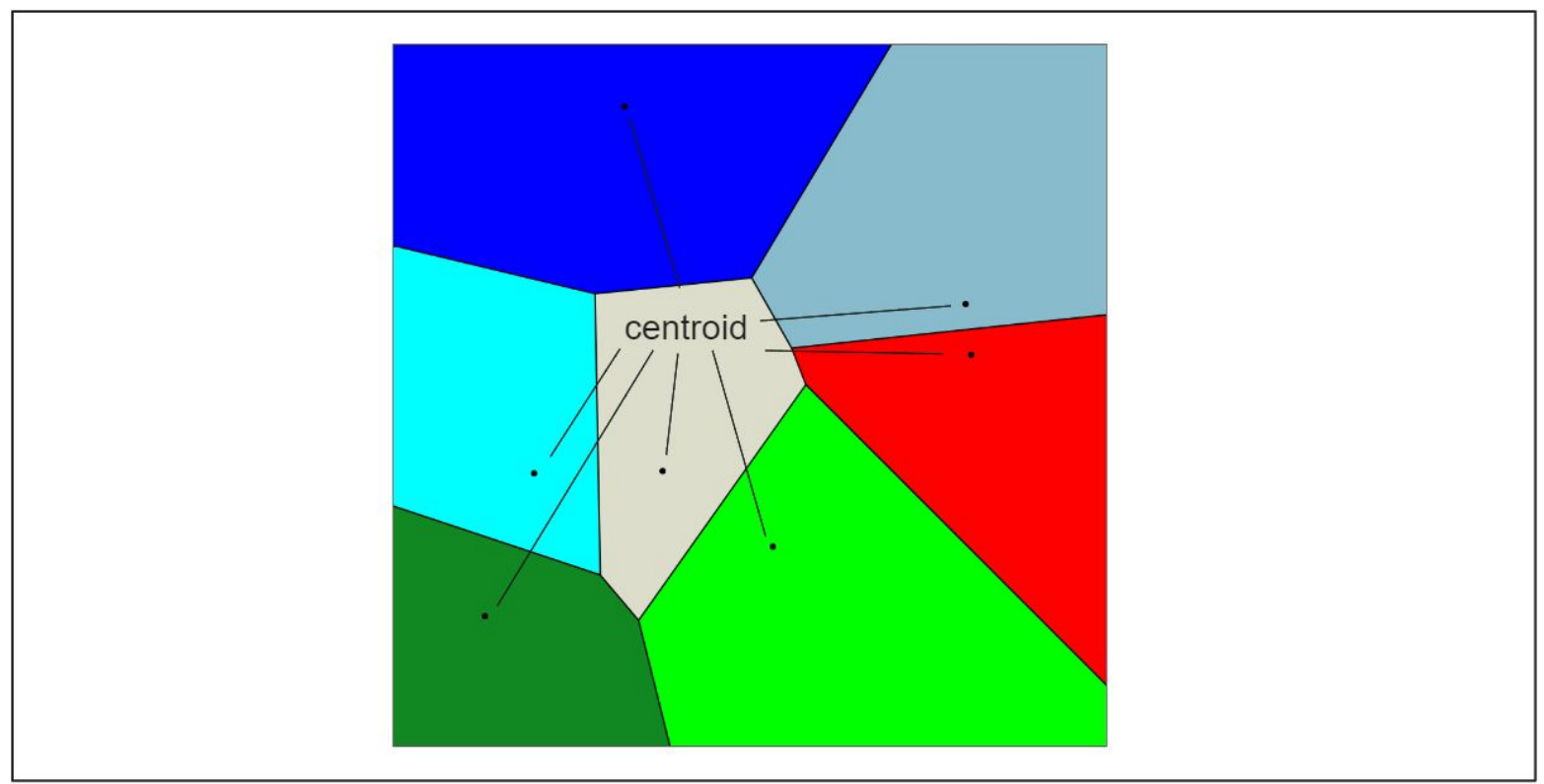
zilliz

Inverted File FLAT
(IVF-FLAT)

 |

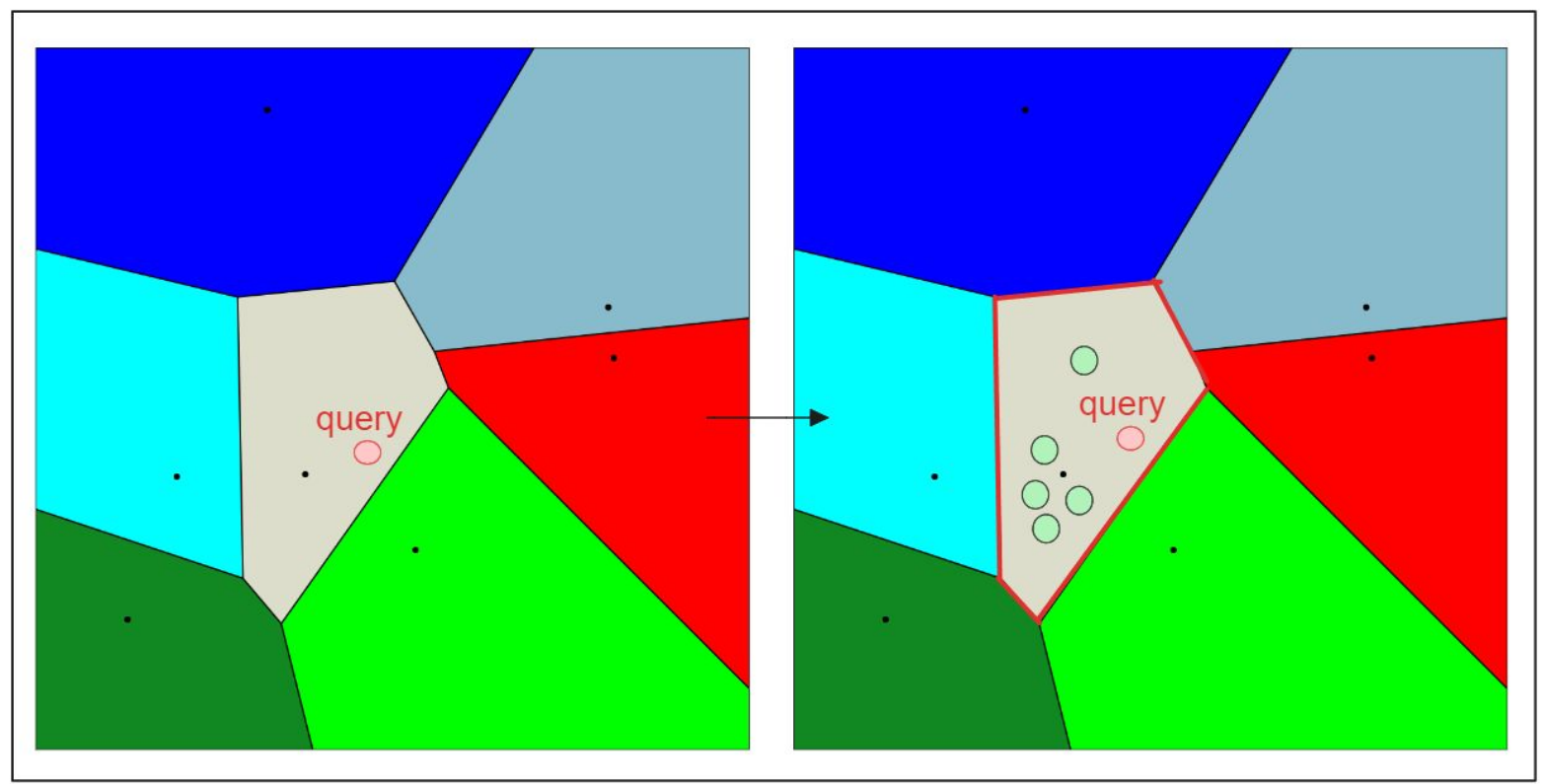
zilliz

IVF-FLAT Index



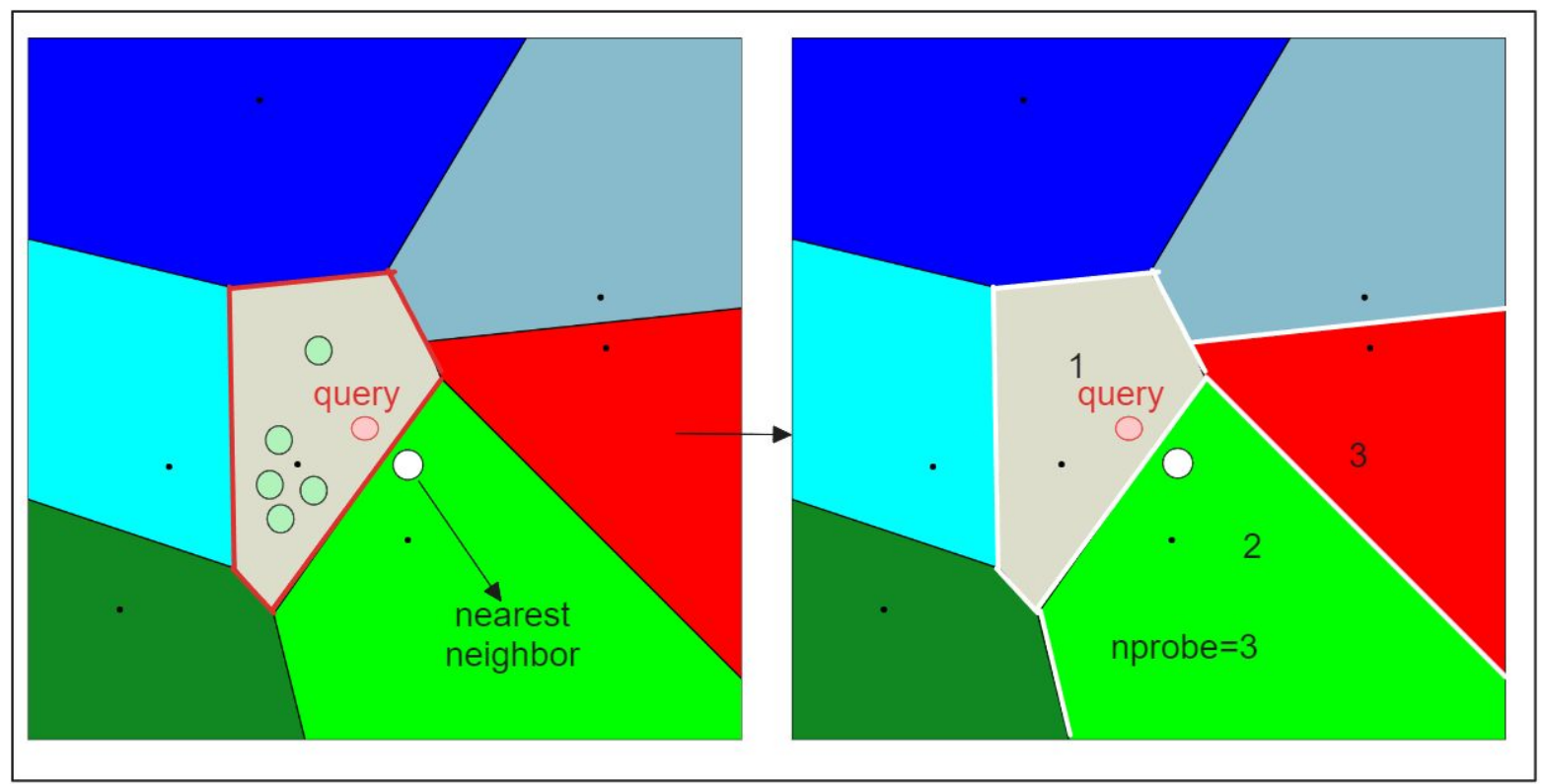
zilliz

IVF-FLAT Index



zilliz

IVF-FLAT Index

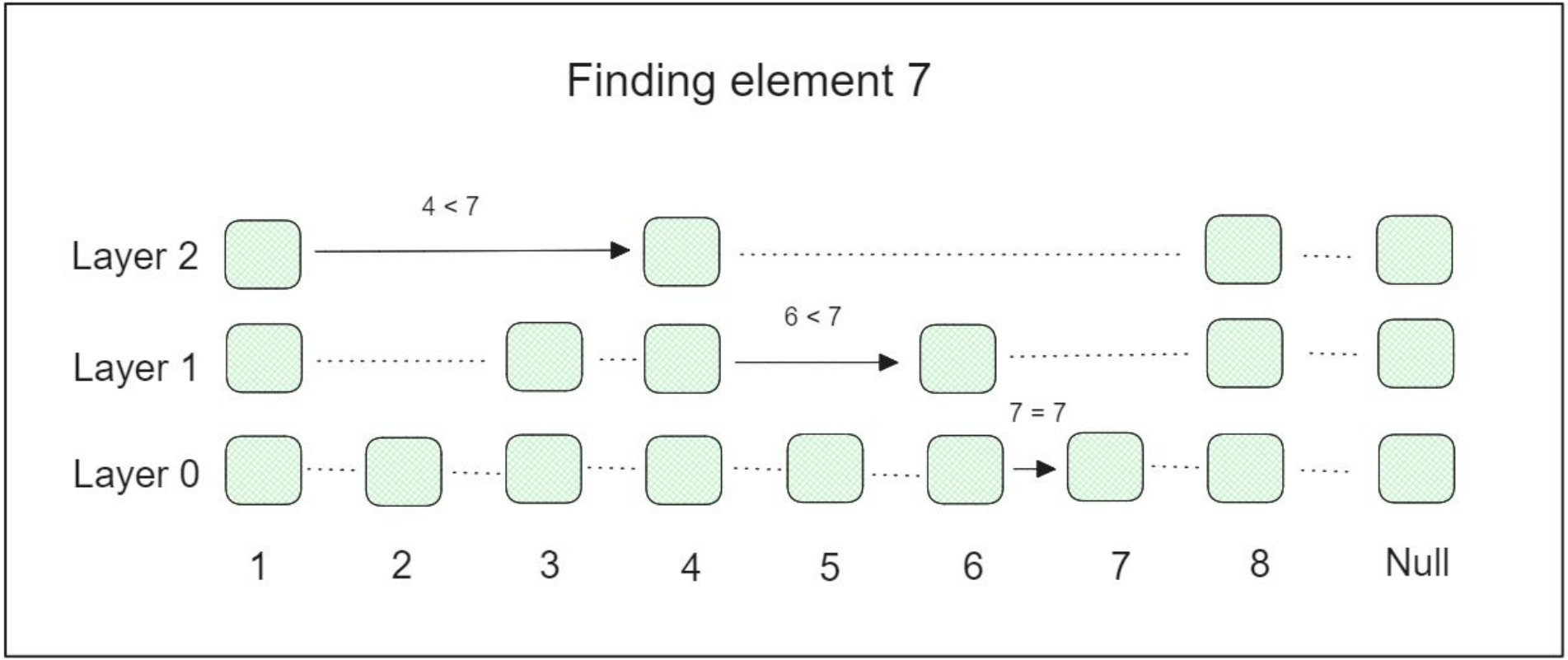


zilliz

Hierarchical Navigable Small World (HNSW)

zilliz

HNSW - Skip List



 |

zilliz

HNSW - NSW Graph

- Built by randomly shuffling data points and inserting them one by one, with each point connected to a predefined number of edges (M).

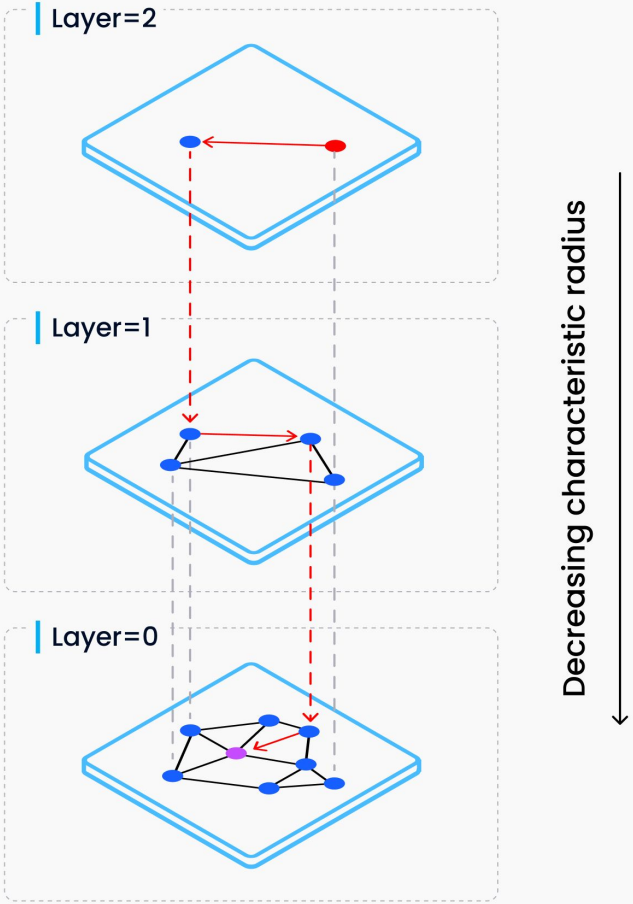
  ⇒ Creates a graph structure that exhibits the "small world".

  ⇒ Any two points are connected through a relatively short path.

 |

zilliz

HNSW



 |

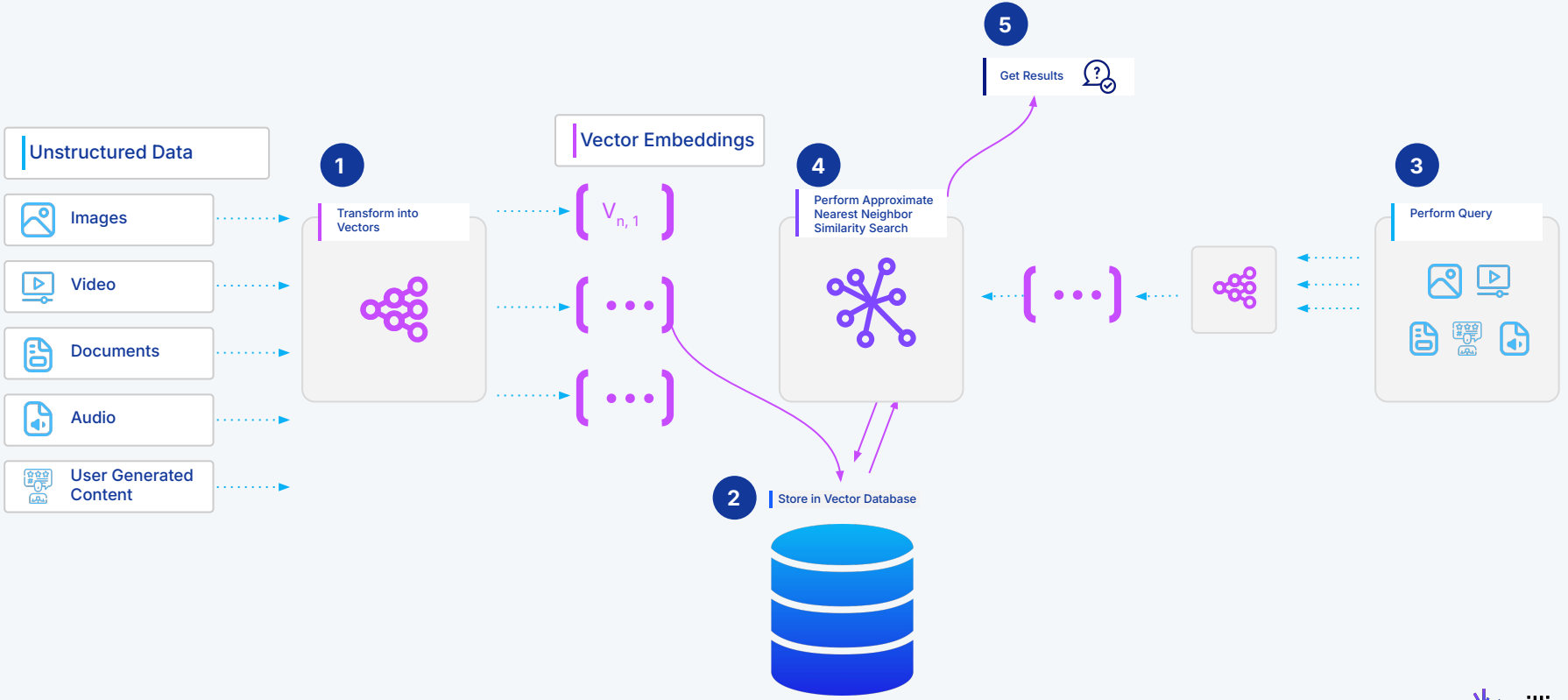
zilliz

S

Search

zilliz

How Similarity Search Works



 |

zilliz

Vector Search Overview

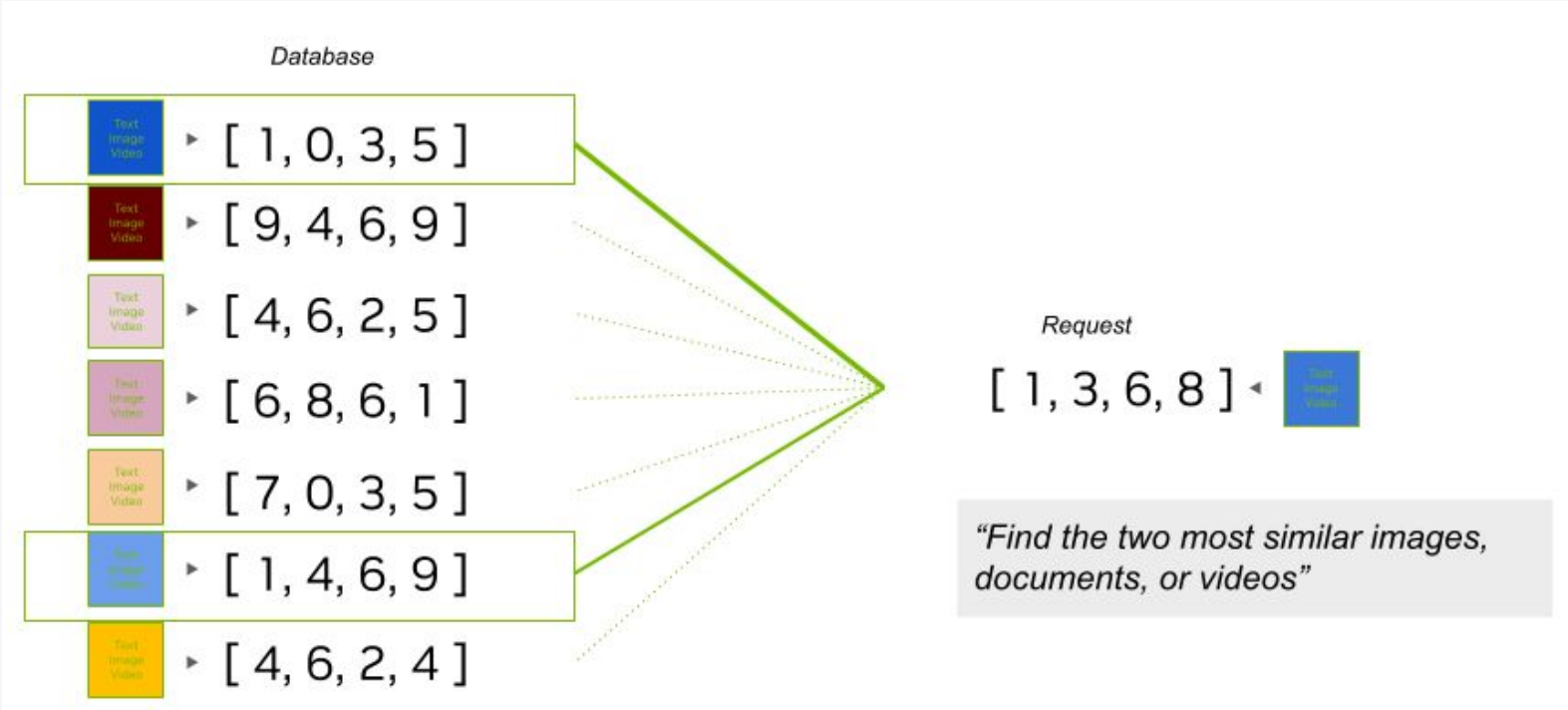


Image from Nvidia

 |

zilliz

Vector Search Overview

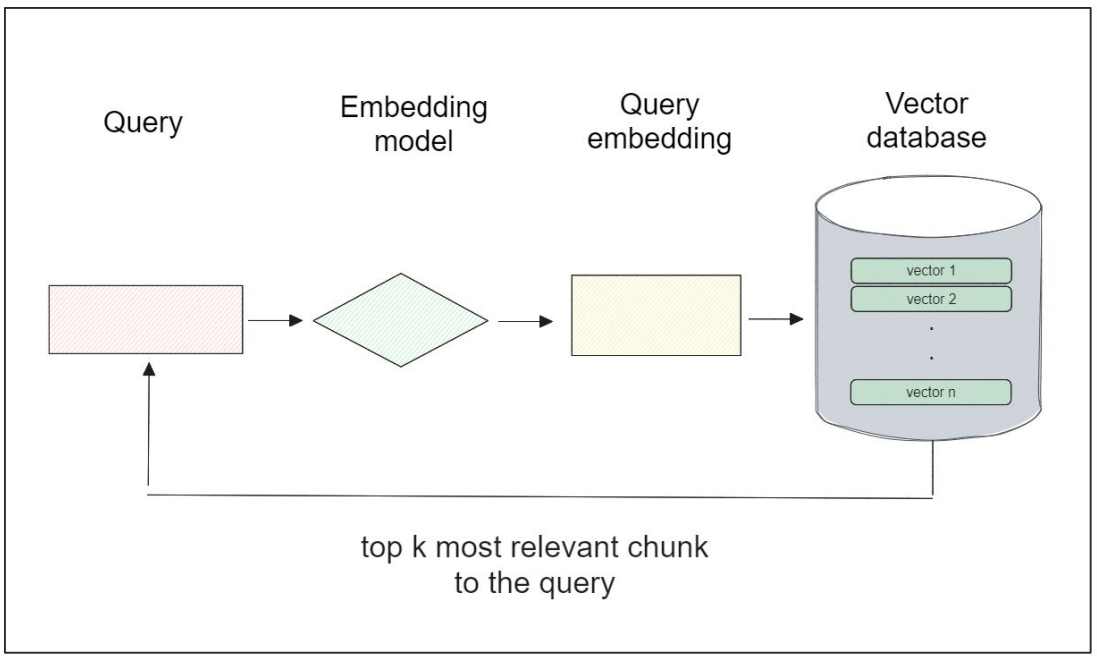


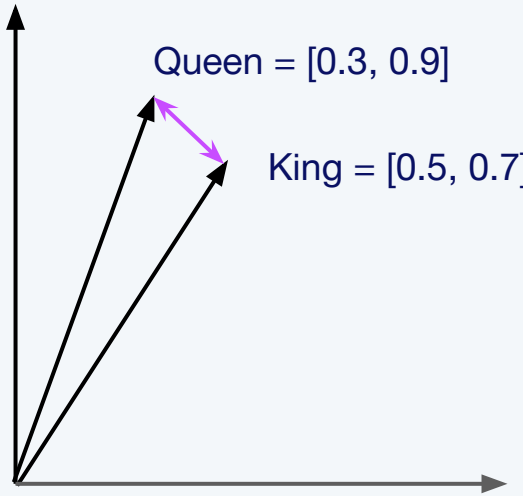
Image from Nvidia

 |

zilliz

Vector Similarity Measures: L2 (Euclidean)

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$



d(Queen, King) = $\sqrt{(0.3-0.5)^2 + (0.9-0.7)^2}$

= $\sqrt{(0.2)^2 + (0.2)^2}$

= $\sqrt{0.04 + 0.04}$

= $\sqrt{0.08} \cong 0.28$

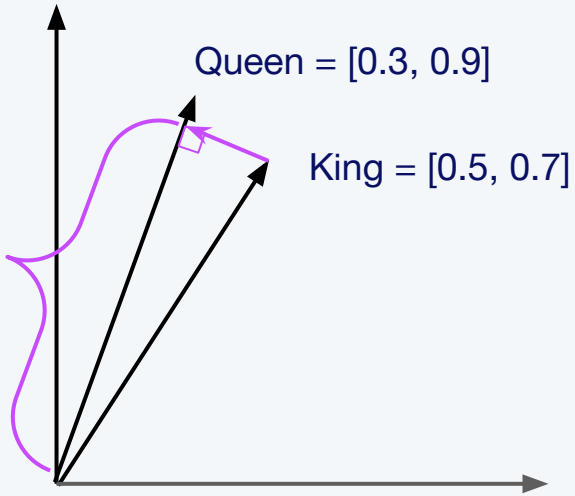
zilliz

Vector Similarity Measures: Inner Product (IP)

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

Queen · King = (0.3*0.5) + (0.9*0.7)

= 0.15 + 0.63 = 0.78

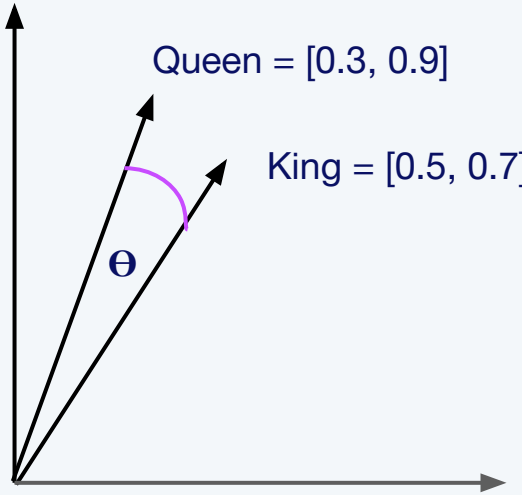


 |

zilliz

Vector Similarity Measures: Cosine

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$



cos(Queen, King) = $\frac{(0.3*0.5)+(0.9*0.7)}{\sqrt{0.3^2+0.9^2} * \sqrt{0.5^2+0.7^2}}$

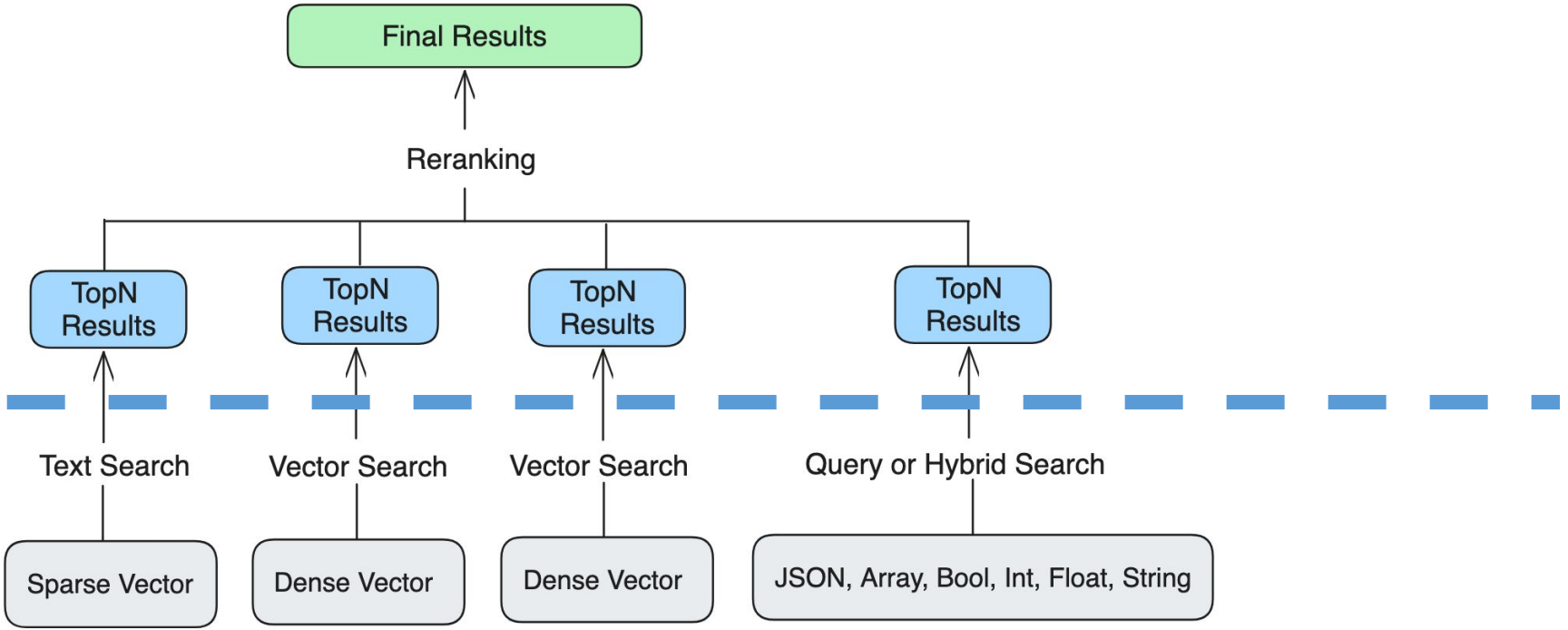
= $\frac{0.15+0.63}{\sqrt{0.9} * \sqrt{0.74}}$

= $\frac{0.78}{\sqrt{0.666}}$

≅ 0.03

zilliz

Hybrid Search



 |

zilliz

M

METADATA FILTERING

zilliz

zilliz

Metadata Filtering in LangChain

with Milvus Vector Database

Step 1: Install the LangChain wrapper for Milvus langchain-milvus and other dependencies.

```
$ pip install --upgrade --quiet  langchain langchain-core langchain-community langchain-text-splitters langchain-milvus langchain-openai bs4
```

Step 2: Ingest data to Milvus through the Milvus.from_documents() abstraction.

```
from langchain_milvus import Milvus from langchain_openai import OpenAIEmbeddings  embeddings = OpenAIEmbeddings()  vectorstore = Milvus.from_documents(  # or Zilliz.from_documents
documents=docs,     embedding=embeddings,     connection_args={
"uri": "./milvus_demo.db",  # or network endpoint for Milvus server.
},     drop_old=True,  # Drop the old Milvus collection if it exists
)
```

Step 3: Perform semantic similarity search and apply filter expr to only select from a certain publishing source.

```
vectorstore.similarity_search(

"What is CoT?",

k=1,

expr="source == 'https://lilianweng.github.io/posts/2023-06-23-agent/'",

)
```

 |

zilliz

Filtering on Metadata

- Search Space Reduction w/ **Pre-Filtering**
- Bitset Wizardry 🧙
  - Use Compact Bitsets to represent Filter Matches
  - Low-level CPU operations for speed
- Scalar Indexing
  - Bloom Filter
  - Hash
  - Tree-based

 |

zilliz

P

Partition by Name or Key

zilliz

How To

https://medium.com/@tspann/partitioning-collections-by-name-395eb48a2238

https://medium.com/@tspann/super-charge-your-ai-applications-with-easy-high-speed-multi-tenancy-with-partition-keys-5fa581127dd6

15

RE-RANKING

zilliz

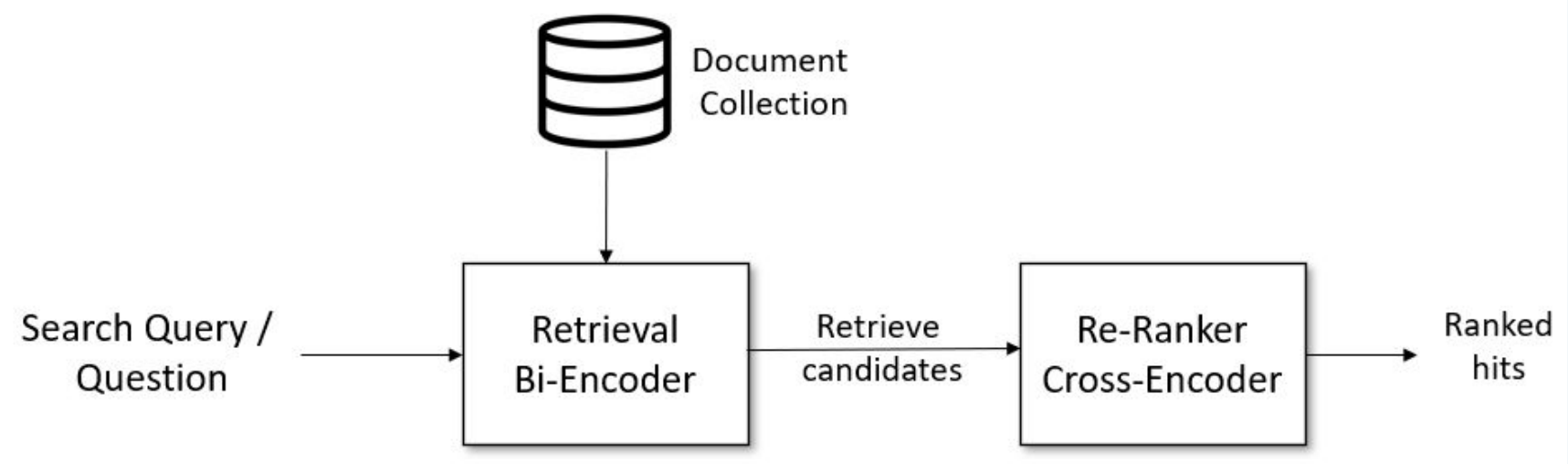
Choose Your Rerank Function

Rerank Function	API or Open-sourced
BGE	Open-sourced
Cross Encoder	Open-sourced
Voyage	API
Cohere	API
Jina AI	API

https://medium.com/@tspann/ranking-for-relevance-with-bm25-b2d9dd62e2f8

 |

Rerankers

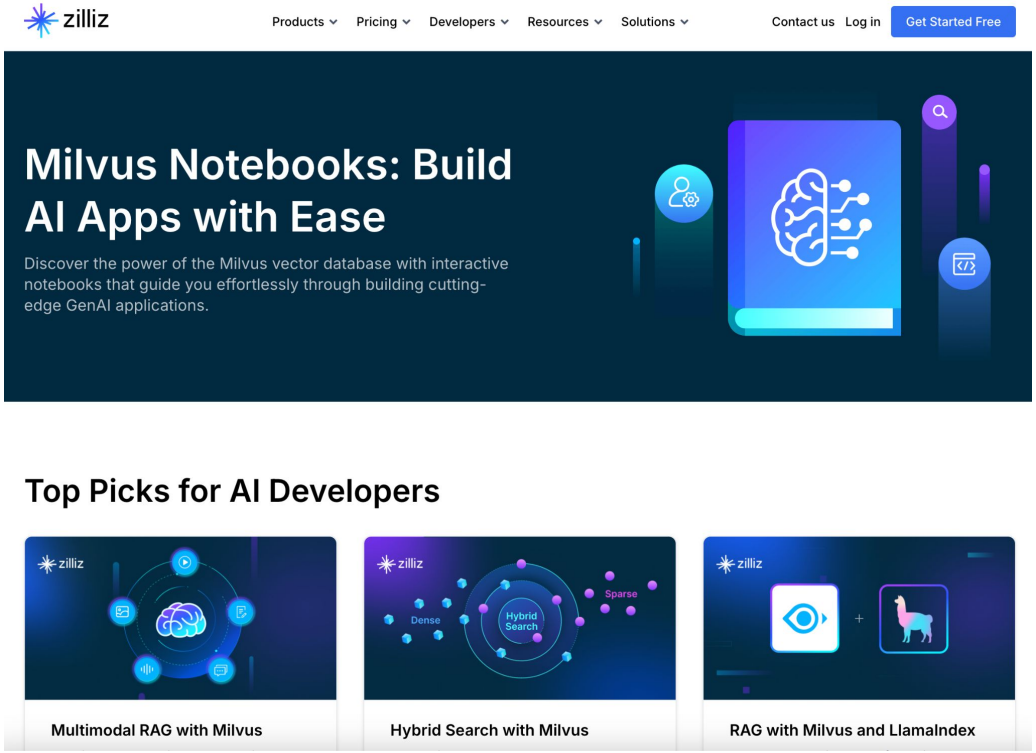


https://zilliz.com/learn/what-are-rerankers-enhance-information-retrieval

zilliz

Q & A

zilliz



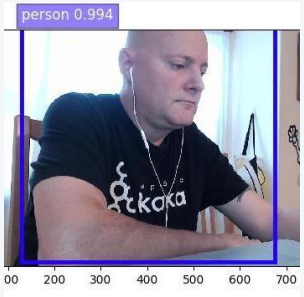
https://zilliz.com/learn/milvus-notebooks

 |

zilliz

Vector Database Resources

Give Milvus a Star!

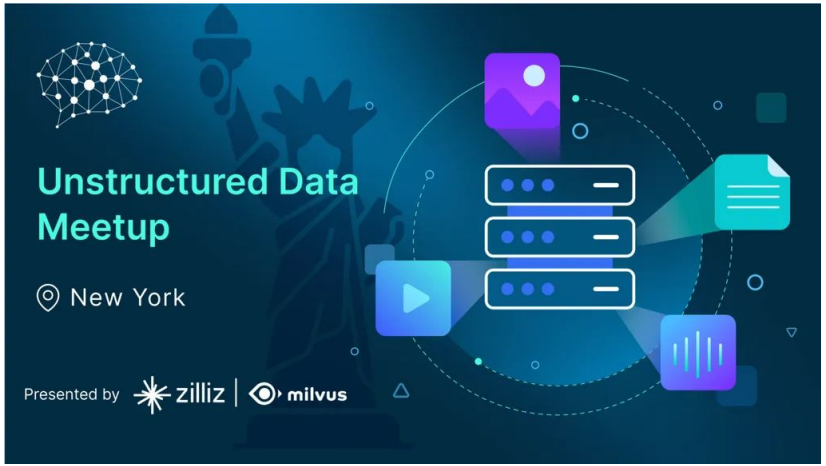


Chat with me on Discord!

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

zilliz

Unstructured Data Meetup



https://www.meetup.com/unstructured-data-meetup-new-york/

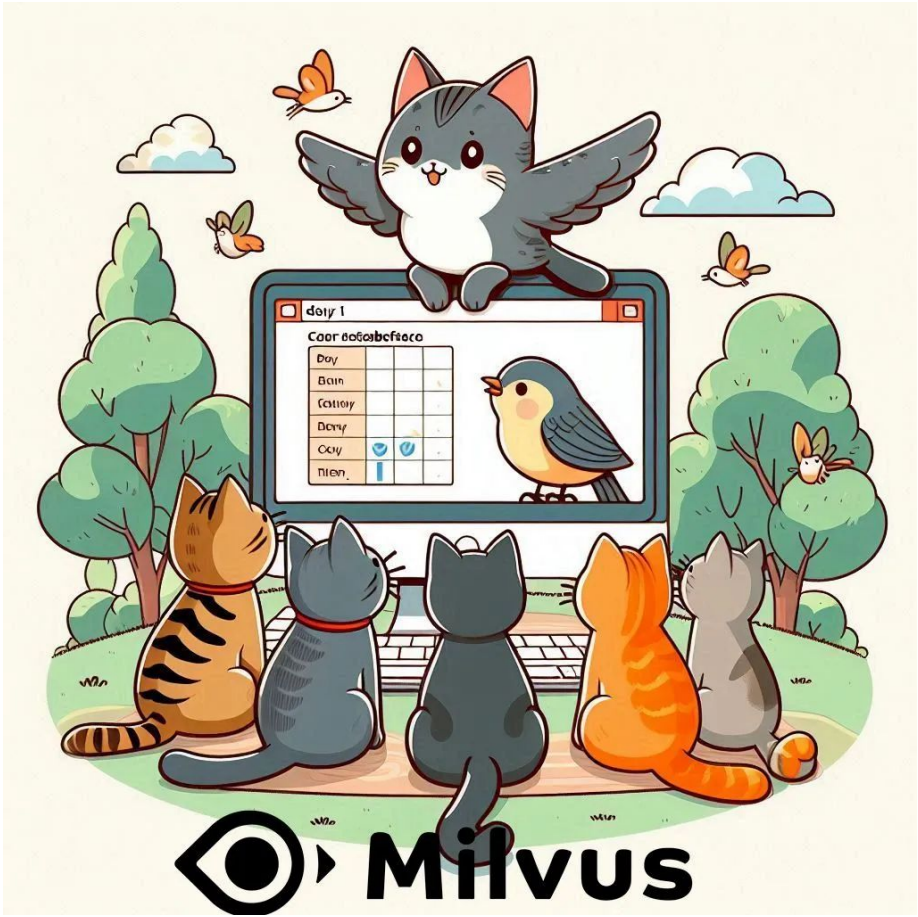
This meetup is for people working in unstructured data. Speakers will come present about related topics such as vector databases, LLMs, and managing data at scale. The intended audience of this group includes roles like machine learning engineers, data scientists, data engineers, software engineers, and PMs.

This meetup was formerly Milvus Meetup, and is sponsored by Zilliz maintainers of Milvus.

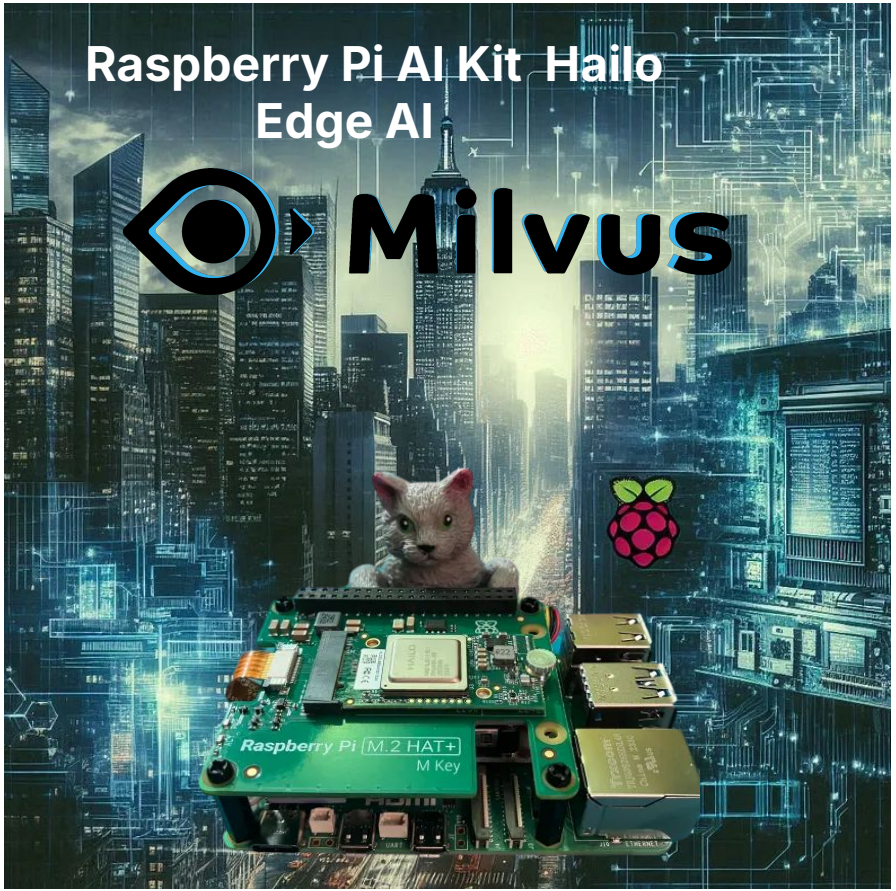

Learning Vector

https://medium.com/@tspann/unstructured-street-data-in-new-york-8d3cde0a1e5b

https://medium.com/@tspann/not-every-field-is-just-text-numbers-or-vectors-976231e90e4d



https://medium.com/@tspann/shining-some-light-on-the-new-milvus-lite-5a0565eb5dd9



https://medium.com/@tspann/unstructured-data-processing-with-a-raspberry-pi-ai-kit-c959dd7fff47

AIM Weekly by Tim Spann

https://bit.ly/32dAJft
https://github.com/milvus-io/milvus

This week in Milvus, Towhee, Attu, GPT Cache, Gen AI, LLM, Apache NiFi, Apache Flink, Apache Kafka, ML, AI, Apache Spark, Apache Iceberg, Python, Java, Vector DB and Open Source friends.

zilliz

Thank you!

milvus.io

github.com/milvus-io/

@milvusio

Connect with me!

@paasDev

/in/timothyspann

 |

zilliz

Getting Started with Vector Databases

Milvus

Open Source Self-Managed

github.com/milvus-io/milvus

29K+ - Star us on GitHub!

Zilliz Cloud

SaaS Fully-Managed

zilliz.com/cloud

 |

zilliz

Zilliz is Hiring!

Join our Team

zilliz.com/careers

- Developer Advocate
- Staff Software Engineer
- Solutions Architect
- and more!

zilliz

zilliz

Get started for free

zilliz.com/cloud