zilliz

The Key to Scaling LLM Applications

Yujian Tang

Speaker

Yujian Tang

Developer Advocate, Zilliz

yujian@zilliz.com
https://www.linkedin.com/in/yujiantang
https://www.twitter.com/yujian_tang

Company

zilliz

@Zilliz_Universe

linkedin.com/in/zilliz

milvusio.slack.com

github.com/milvus-io/milvus

zilliz.com

CONTENTS

01

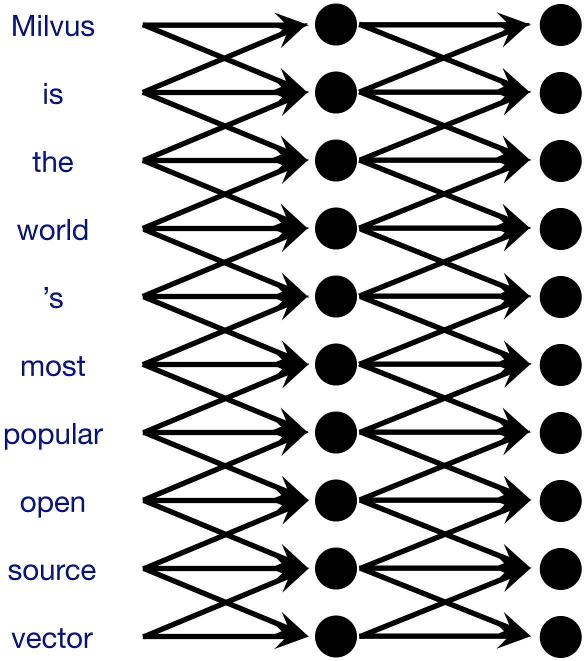
Large Language Models (LLMs)

ChatGPT Craziness

Claude

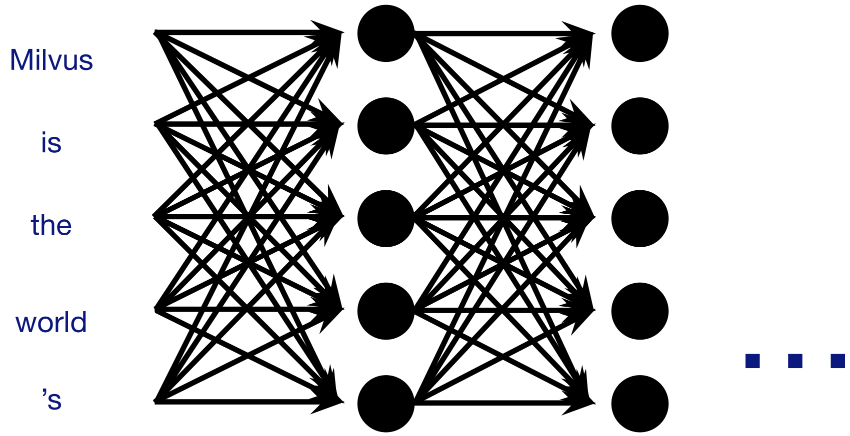
Bard Experiment

Convolution



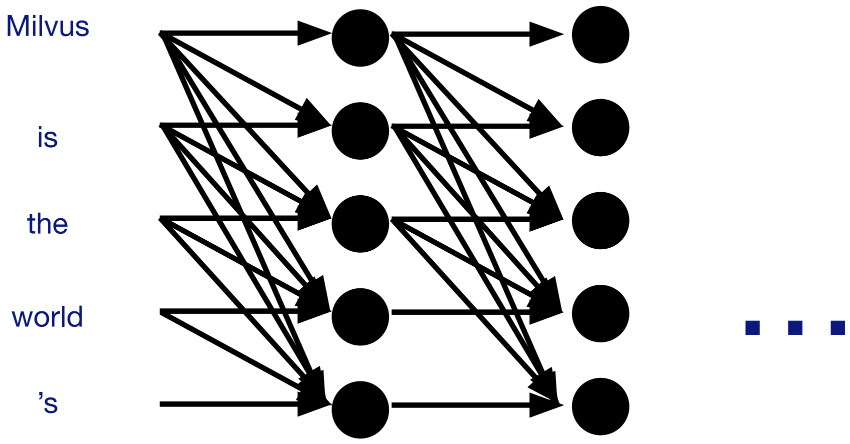
...

Self-Attention



(Global context)

Causal Attention



(Directional global context)

LLMs are *Stochastic*

- LLMs predict future tokens (a-la RNNs)
  - “Milvus is the world ’s most popular vector ___”
  - {“database”: 0.86, “search”: 0.11, “embedding”, 0.01, …}
- Downside: outdated input data could be cause for hallucination
  - Plausible-sounding but factually incorrect responses

Some Obligatory Math

- Goal: given some tokens $t_0, t_1, \ldots, t_n$: predict t_{n+1}
- Model outputs a probability distribution across all possible tokens
  - p(“database” | “milvus”, “is”, “the”…) = 0.86
  - p(“search” | “milvus”, “is”, “the”…) = 0.11
  - p(“embedding” | “milvus”, “is”, “the”…) = 0.01
- $\boldsymbol{\theta} \in \mathbf{R}^N$
  - Some set of N total parameters (weights)
- **D**
  - Input dataset (training data)
- L(**θ**, **D**)
  - Causal language modeling loss

02

Challenges with LLMs

Hallucination Example

- Query: “How do I perform a query using Milvus?”

Hallucination Example

```
import random
from milvus import Milvus, IndexType, MetricType

# Set up a connection to Milvus server
milvus = Milvus(host='localhost', port='19530')

# Create a collection
collection_name = 'my_collection'
milvus.create_collection({
    'collection_name': collection_name,
    'dimension': 512,
    'index_file_size': 1024,
    'metric_type': MetricType.L2
})

# Insert vectors
vectors = [[random.random() for _ in range(512)] for _ in range(1000)]
ids = [i for i in range(1000)]
milvus.insert(collection_name=collection_name, records=vectors, ids=ids)

# Build an index
milvus.create_index(collection_name, IndexType.IVF_FLAT, {"nlist": 128})

# Perform a query
query_vector = [random.random() for _ in range(512)]
results = milvus.search(collection_name, query_records=[query_vector], top_k
```

Hallucination Example

```python
import random
from milvus import Milvus, IndexType, MetricType

# Set up a connection to Milvus server
milvus = Milvus(host='localhost', port='19530')

# Create a collection
collection_name = 'my_collection'
milvus.create_collection({
    'collection_name': collection_name,
    'dimension': 512,
    'index_file_size': 1024,
    'metric_type': MetricType.L2
})

# Insert vectors
vectors = [[random.random() for _ in range(512)] for _ in range(1000)]
ids = [i for i in range(1000)]
milvus.insert(collection_name=collection_name, records=vectors, ids=ids)

# Build an index
milvus.create_index(collection_name, IndexType.IVF_FLAT, {"nlist": 128})

# Perform a query
query_vector = [random.random() for _ in range(512)]
results = milvus.search(collection_name, query_records=[query_vector], top_k
```

Interfacing with a Milvus instance is done via `connections`, not a client

The Solution to Hallucination

- Inject domain knowledge into large language models

03

The CVP Framework

The CVP Framework

- Key idea: we can view LLM apps as a general purpose computer
  - Processor
  - Persistent storage
  - Code

The CVP Framework

C: ChatGPT (or any other LLM)

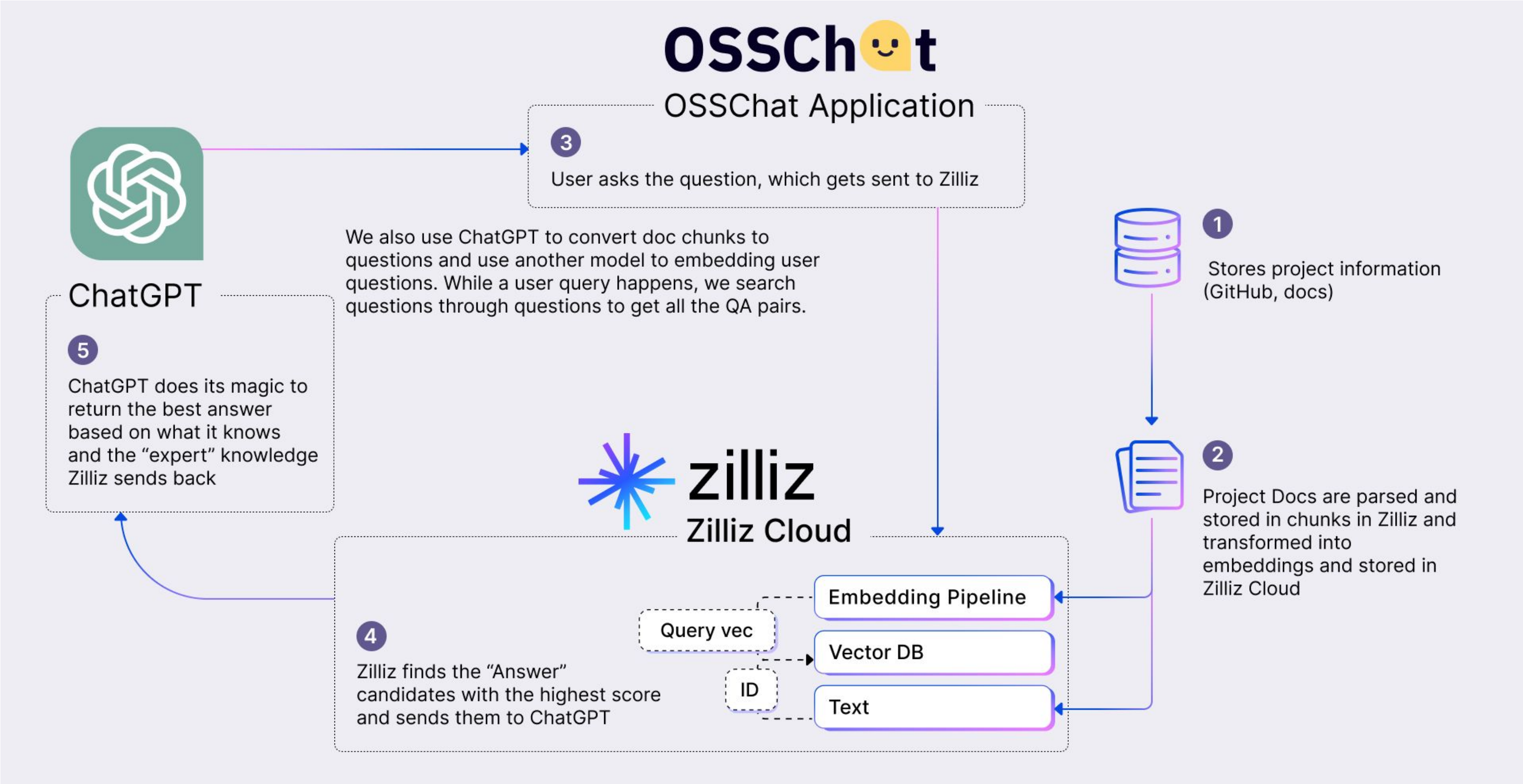
- This can also be interpreted as the “processor” block for CVP

V: Vector database (e.g. Milvus)

- Can also be interpreted as the “storage” block for CVP

P: Prompt-as-code

- Interface between processor and storage blocks



https://zilliz.com/blog/ChatGPT-VectorDB-Prompt-as-code

Hallucination Example

- Query: “How do I perform a query using Milvus?”

The Solution to Hallucination

```python
from pymilvus import connections, FieldSchema, CollectionSchema, DataType, Collection

# Connect to Milvus server
connections.connect(host='localhost', port='19530')

# Define collection schema
dim = 128
default_fields = [
    FieldSchema(name="count", dtype=DataType.INT64, is_primary=True),
    FieldSchema(name="random_value", dtype=DataType.DOUBLE),
    FieldSchema(name="float_vector", dtype=DataType.FLOAT_VECTOR, dim=dim)
]
default_schema = CollectionSchema(fields=default_fields, description="test collection")

# Load collection
collection = Collection(name="hello_milvus", schema=default_schema)

# Define query vector
query_vector = [[0.1, 0.2, 0.3, ..., 0.9, 1.0]]

# Perform query
search_param = {"nprobe": 16}
results = collection.query(query_records=query_vector, top_k=10, params=search_param)

# Print query results
for result in results:
    print(result.id, result.distance)
```

How Does This Solve Hallucinations?

- Access to Domain Knowledge
- Semantic Search on Domain Knowledge via Vector Embeddings

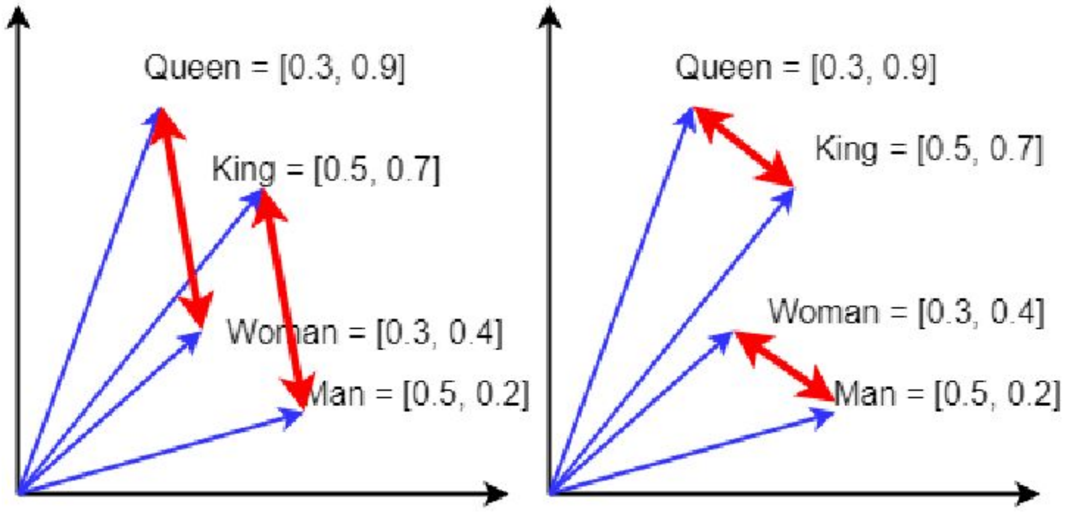
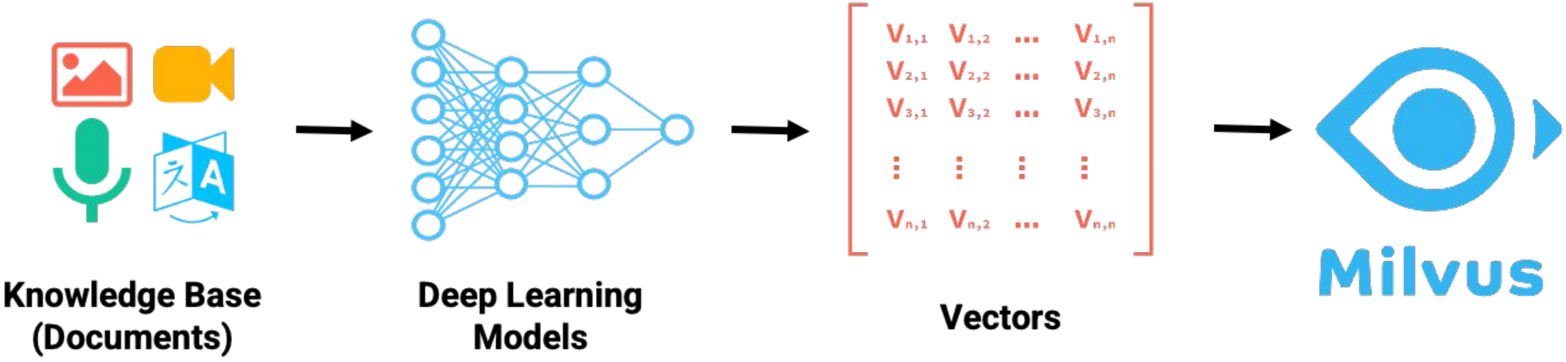


Image from Sutor et al

How Do We Implement This in Practice?



04

What is a Vector Database? Featuring Milvus

Vector Database Overview

zilliz

Milvus

A database purpose-built to store, index, and query large quantities of vector embeddings.

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

Why a Purpose-Built Vector Database?

- Vector search library
  - High-performance vector search
- Vector database
  - High-performance vector search
  - Replication, failover
  - Horizontal/vertical scalability
  - Automatic indexing
  - Backup/recovery

- How do I support different applications?
  - High query load
  - High insertion/deletion
  - Full precision/recall
  - Accelerator support (GPU, FPGA)
  - Billion-scale storage

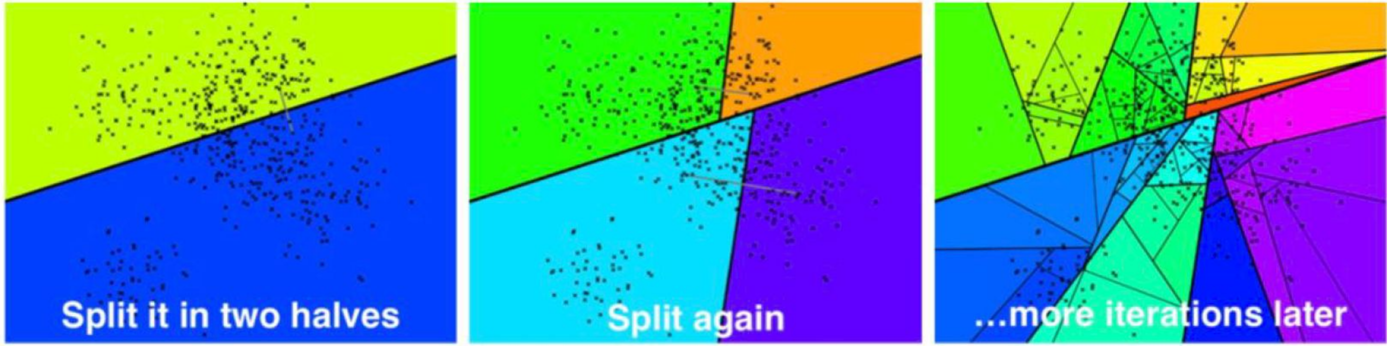
Why a Purpose-Built Vector Database?

- Vector search library
  - High-performance vector search

- Vector database
  - Advanced filtering (filtered vector search, chained filters)
  - Hybrid search (e.g. full text + dense vector)
  - Durability (any write in a db is durable, a library typically only supports snapshotting)
  - Replication / High Availability
  - Sharding
  - Aggregations or faceted search
  - Backups
  - Lifecycle management (CRUD, Batch delete, dropping whole indexes, reindexing)
  - Multi-tenancy

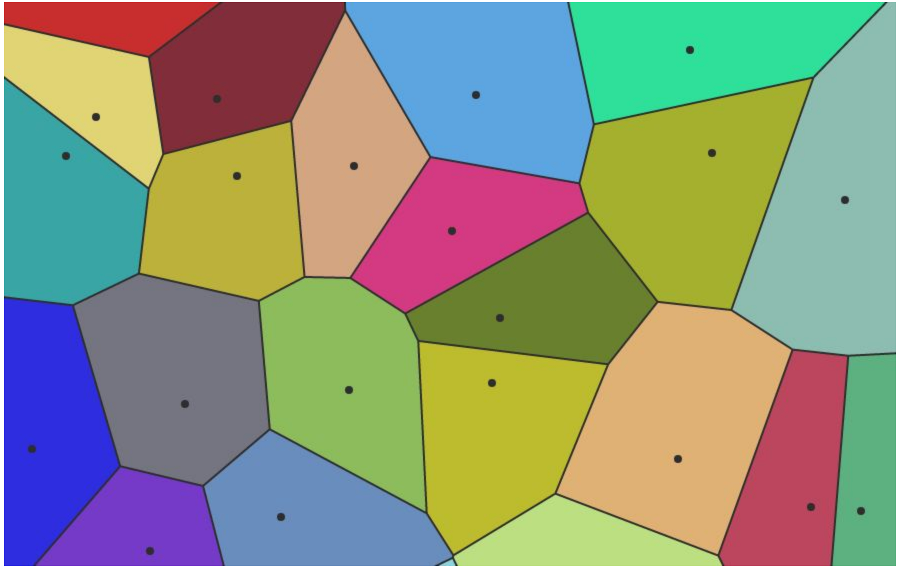
- How do I support different applications?
  - High query load
  - High insertion/deletion
  - Full precision/recall
  - Accelerator support (GPU, FPGA)
  - Billion-scale storage

Approximate Nearest Neighbors Oh Yeah



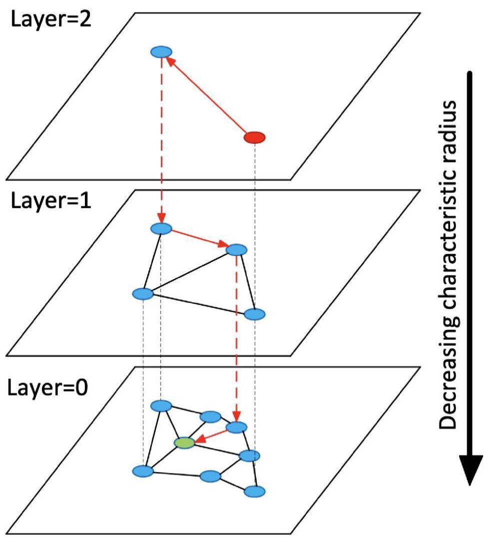
Source:
https://sds-aau.github.io/M3Port19/portfolio/ann/

Inverted File Index



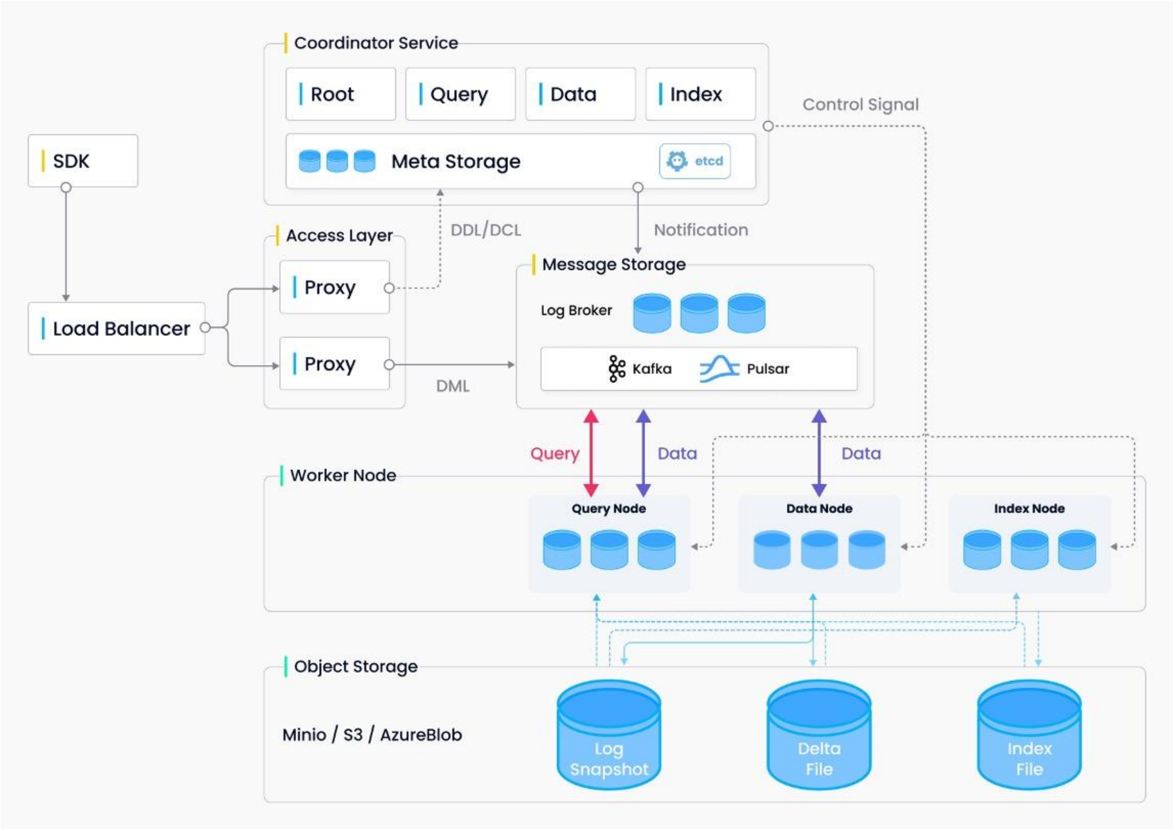
Source:
https://towardsdatascience.com/similarity-search-with-ivfpq-9c6348fd4db3

Hierarchical Navigable Small Worlds (HNSW)



Source:
https://arxiv.org/ftp/arxiv/papers/1603/1603.09320.pdf

Milvus Architecture



05

A Quick Demo

OSSChat

osschat.io

zilliz

THANK YOU FOR LISTENING

github.com/milvus-io/milvus