Cloud Computing (INFS3208)

Lecture 7: Vector Databases

Dr Heming Du

School of Electrical Engineering and Computer Science

Faculty of Engineering, Architecture and Information Technology

The University of Queensland

Re-cap

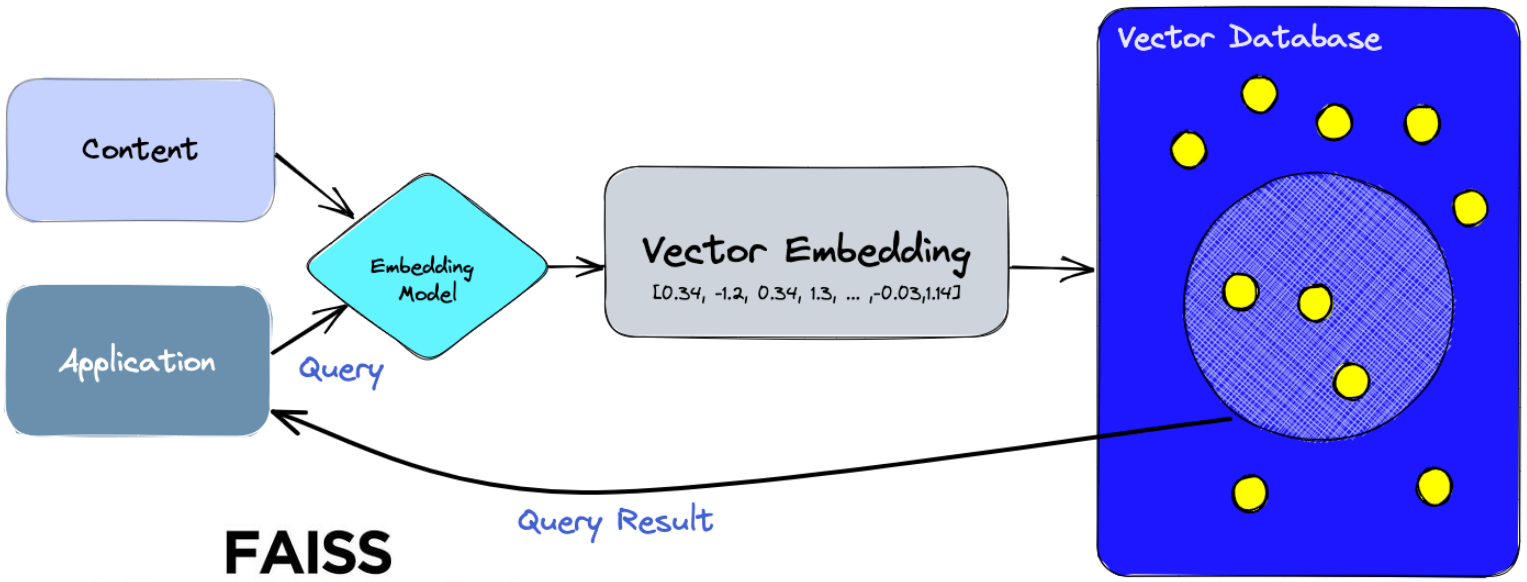
- Database Background
- Relational Data Bases
  - Revisit Relational DBs
  - ACID Properties
  - Clustered RDBMs
- Non-relational Data Bases
  - NoSQL concepts & types
  - Database Partitioning and CAP Theorem
  - MongoDB
  - Cassandra
  - HBase

Outline

- Introduction to Vector Databases
- Fundamentals of Vector Databases
  - What are vectors?
  - Vector Similarity Measures
- Indexing Techniques for Vector Databases
  - Introduction to Indexing
  - Types of Indexing Techniques
- Vector Databases in the Cloud
- Applications:
  - Retrieval Augmented Generation
  - Anomaly Detection

What is a Vector Database?

"A vector database is any database that allows you to store, index, and query vector embeddings, or numerical representations of unstructured data, such as text, images, or audio."



vespa

marqo

drant

FAISS
Scalable Search With Facebook AI

Milvus

LanceDB

https://cloud.google.com/discover/what-is-a-vector-database

Common Vector Database Use Cases

Retrieval Augmented Generation (RAG)

- Enhance the breadth of knowledge in Large Language Models (LLMs) by integrating external data sources, enriching both the LLMs and associated AI-driven applications.

Recommender System

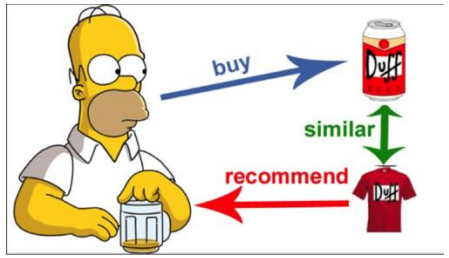
- Utilize vector databases to align user actions or content attributes with similar entities, facilitating tailored and effective recommendations.

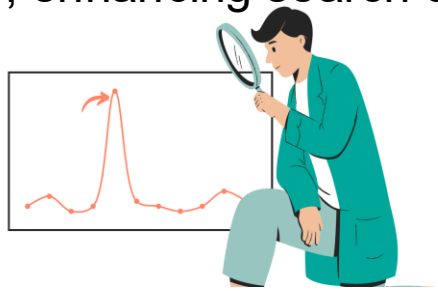
Anomaly Detection

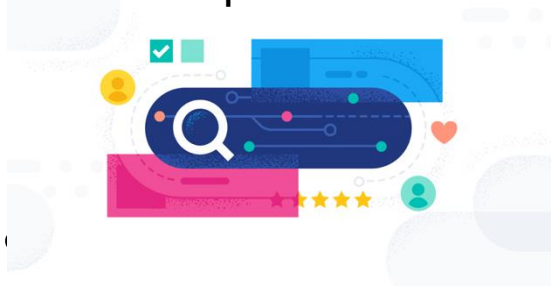
- Identify data points, events, or observations that significantly diverge from established patterns, enabling timely intervention or further analysis.

Image Similarity Search

- Leverage vector databases to efficiently locate and retrieve images or objects that closely resemble those in extensive image repositories, enhancing search capabilities and user experience.







Why Vector Database?

Data perspective

- Support complex data, e.g., *geospatial data*, *genomic data*, *social media likes*
- Embedding unstructured data - 80% of the world's data is unstructured
- High-dimensional data handling

Method perspective

- Efficient and accurate similarity search and retrieval
- Sophisticated query capabilities

Application perspective

- AI and machine learning integration
- Scalability for modern applications, e.g., biology, healthcare, e-commerce, etc.

Outline

- Introduction to Vector Databases
- Fundamentals of Vector Databases
  - What are vectors?
  - Vector Similarity Measures
- Indexing Techniques for Vector Databases
  - Introduction to Indexing
  - Types of Indexing Techniques
- Vector Databases in the Cloud
- Applications:
  - Retrieval Augmented Generation
  - Anomaly Detection

Fundamentals of vector databases (1/8)

Vector Embeddings:

- Dense representation of data items (like text, images, or sounds) in a high-dimensional space.
- Each dimension represents a feature of the data, and the distance between vectors indicates their similarity.
- Also known as dense representations, feature vectors, latent vectors, embedding vectors, etc.

Similarity Search:

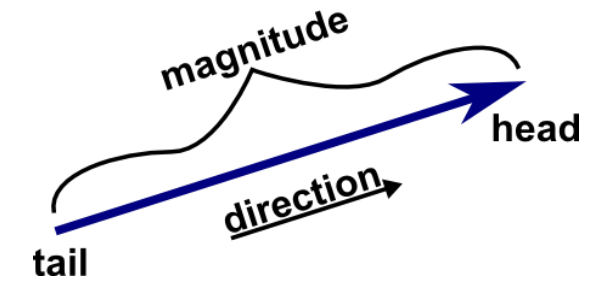
- Search based on the similarity of data points, rather than exact matches
- Common metrics used to determine similarity include Euclidean distance, cosine similarity, and inner product.

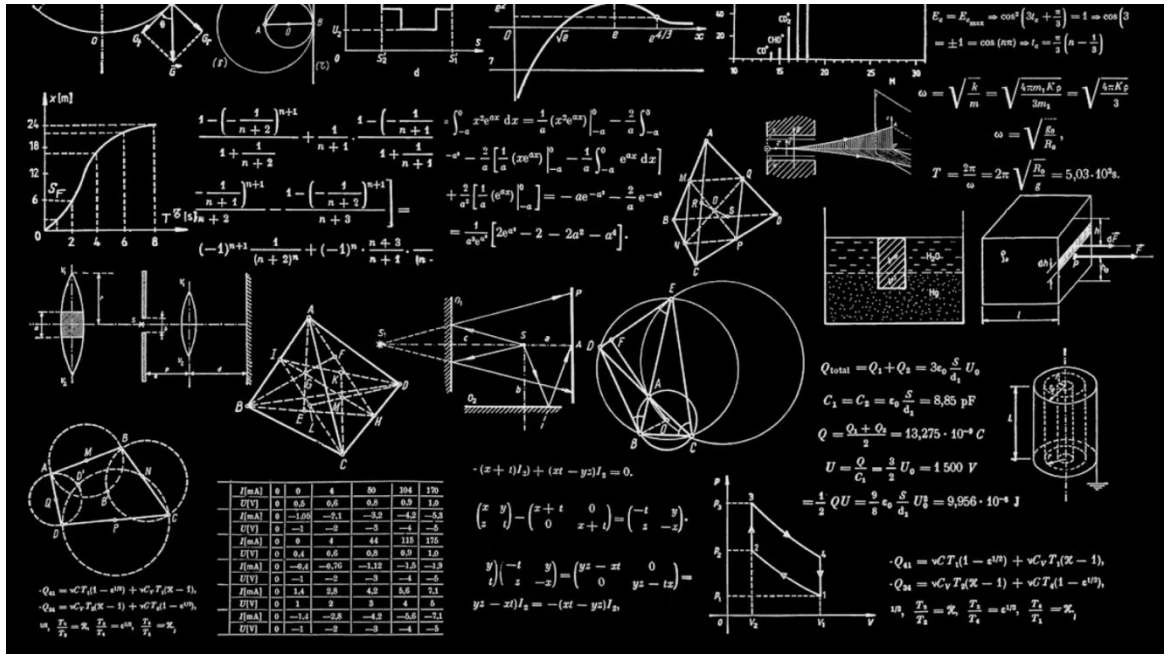
Indexing:

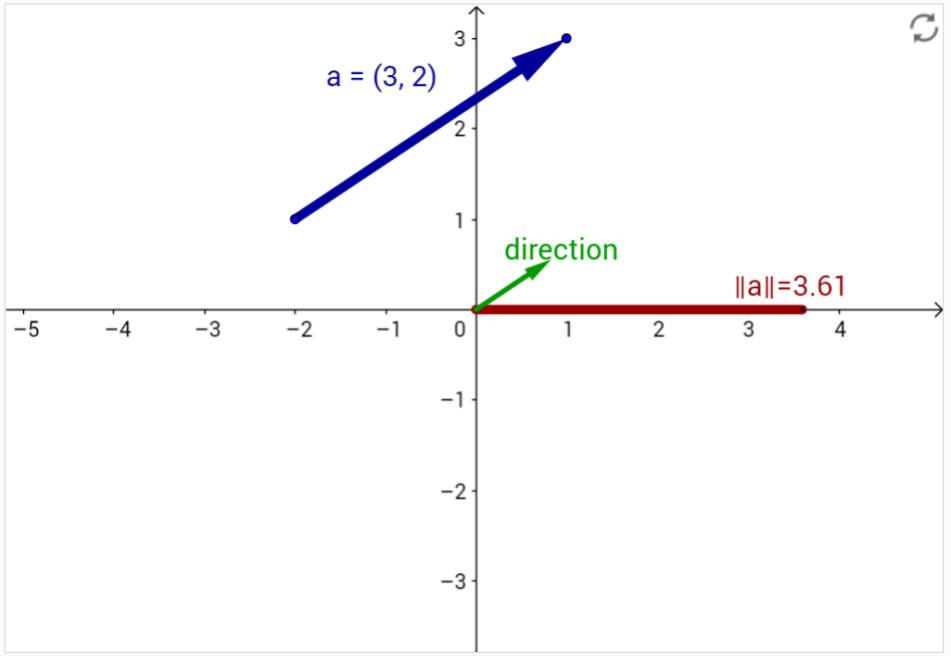
- Critical for vector databases to speed up similarity searches among high-dimensional data
- Techniques such as k-d trees, hierarchical navigable small world graphs (HNSW), and inverted files are used to enable efficient nearest neighbour searches.

Fundamentals of vector databases (2/8)

In mathematics, physics, and engineering, a **VECTOR** is a geometric object that has **magnitude** (or length) and **direction**.



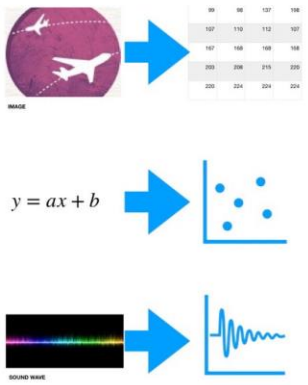




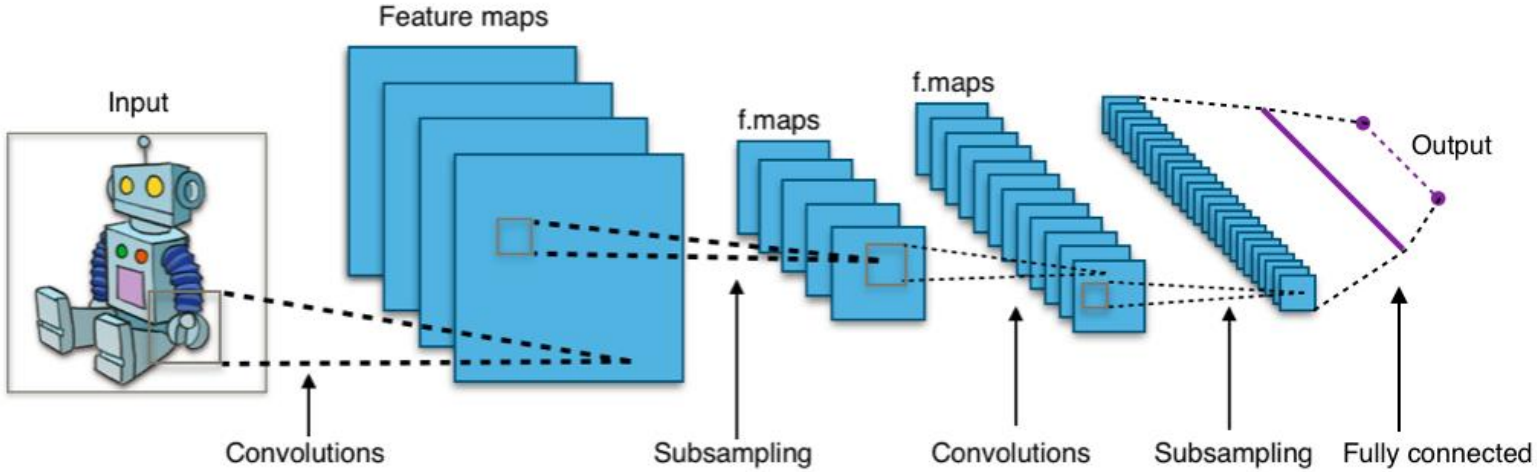
https://en.wikipedia.org/wiki/Euclidean_vector
https://mathinsight.org/vector_introduction
https://myscale.com/blog/key-facts-sparse-vectors/

Fundamentals of vector databases (3/8)

- In machine learning and pattern recognition, **a feature (vector)** is an *individual measurable property* or *characteristic* of a phenomenon.
- **Feature learning or representation learning** is a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data.
- **Dense** vectors are a type of mathematical **objects that represent data in machine learning and artificial intelligence**.



Representations

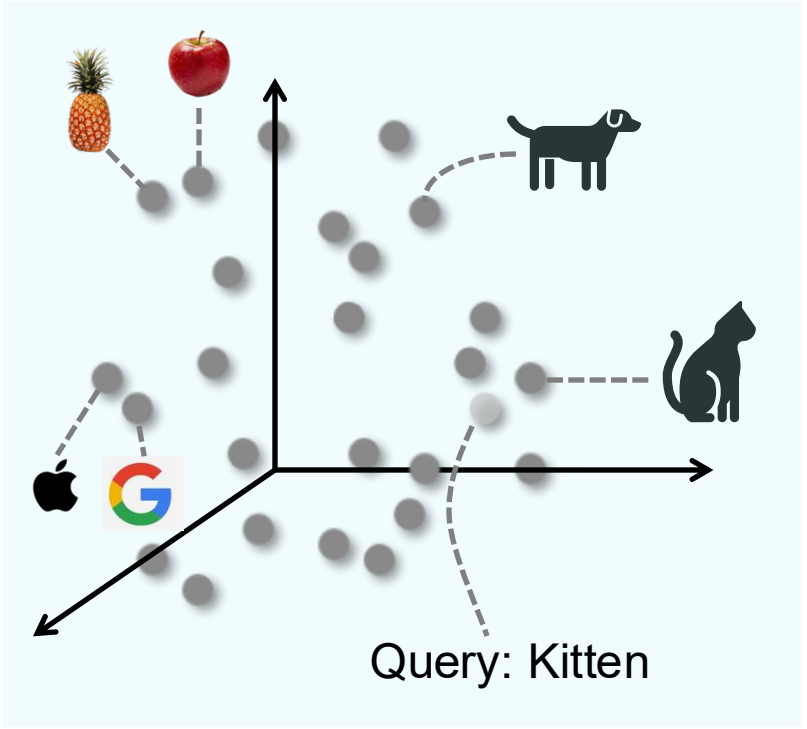


Bishop, Christopher (2006). Pattern recognition and machine learning. Berlin: Springer. ISBN 0-387-31073-8. https://www.pinecone.io/learn/vector-embeddings/

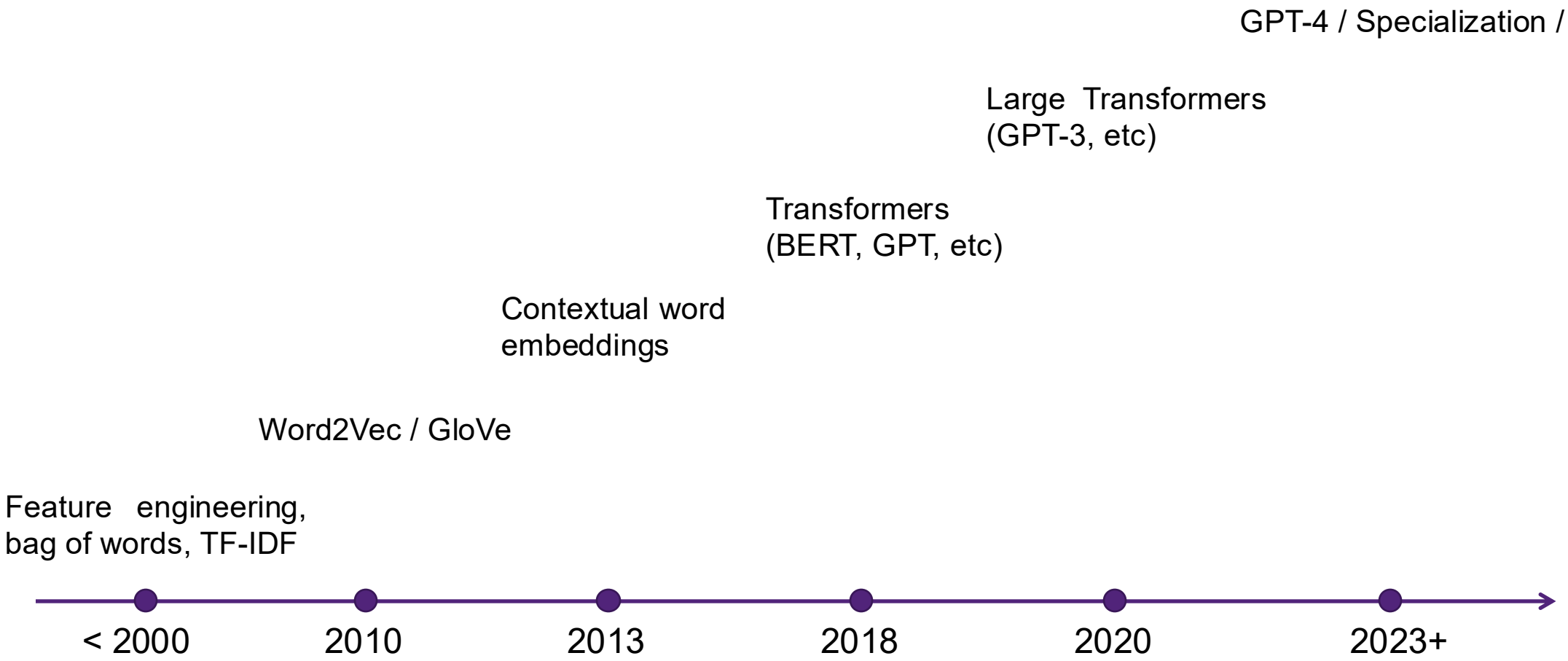
Fundamentals of vector databases (4/8)

A vector in the context of vector database

- Stores data as mathematical objects defined by *size and direction*
- Is a mathematical way to show data points with *more than one dimension*
- *An array* of numerical values of data points
- An array of numerical values or attribute that relate to different features
- Each numerical data point is a dimension

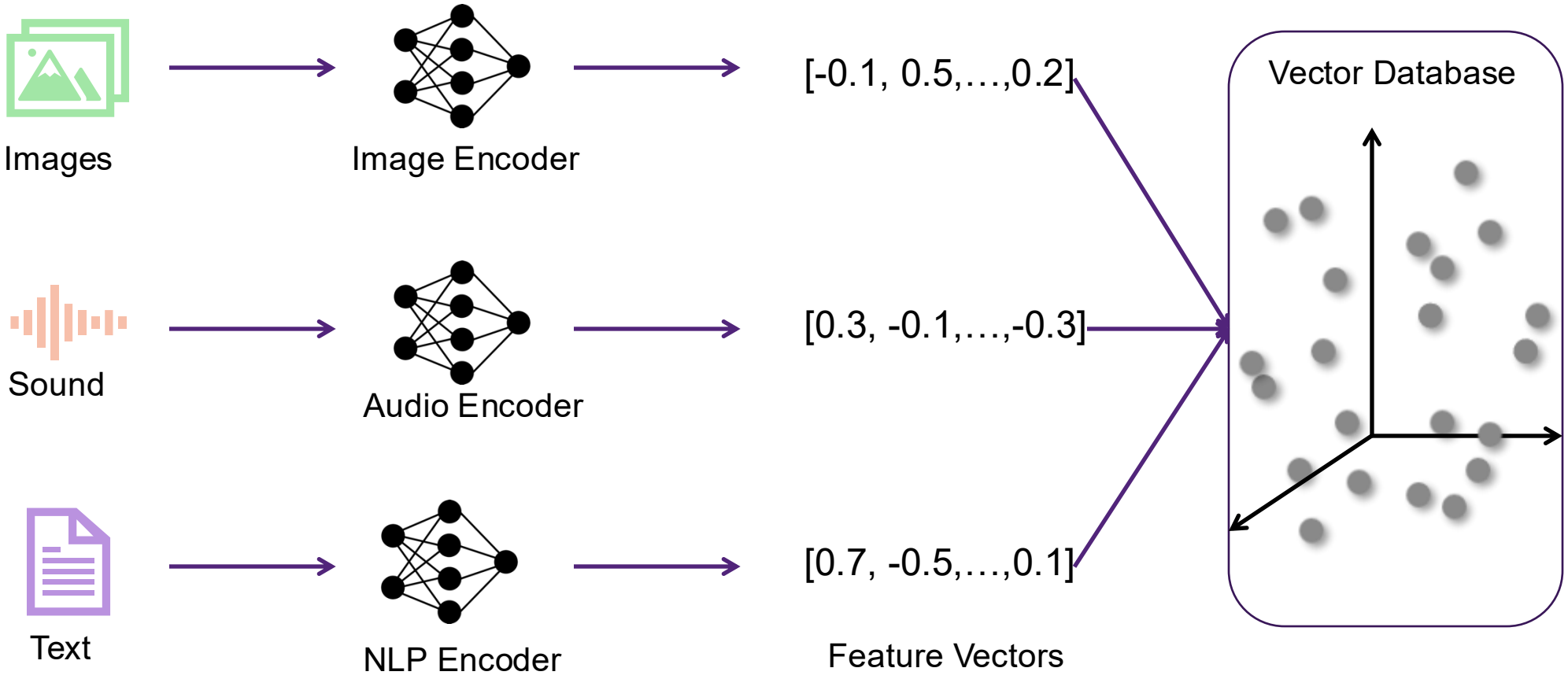


Fundamentals of vector databases (5/8)



Fundamentals of vector databases (6/8)

Support many kinds of data:



Graph, Video, Point Cloud, etc.

Fundamentals of vector databases (7/8)

Data Representation:

- Relational Database: Traditional databases organize data in a structured format using tables, rows, and columns, ideal for relational data.
- Vector Database: Vector databases represent data as multi-dimensional vectors, efficiently encoding complex and unstructured data like images, text, and sensor data.

Data Search and Retrieval:

- Relational Database: SQL queries are suited for traditional databases with structured data.
- Vector Database: Vector databases specialize in similarity searches and retrieving vectorized data, facilitating tasks like image retrieval, recommendation systems, and anomaly detection.

Indexing:

- Relational Database: Traditional databases employ indexing methods like B-trees for efficient data retrieval.
- Vector Database: Vector databases use indexing structures like metric trees and hashing suited for high-dimensional spaces, enhancing nearest-neighbor searches and similarity assessments.

Fundamentals of vector databases (8/8)

Data Representation

Data Search and Retrieval

Indexing

Scalability:

- Relational Database: Scaling traditional databases can be challenging, often requiring resource augmentation or data sharding.
- Vector Database: Vector databases are designed for scalability, especially in handling large datasets and similarity searches, using distributed architectures for horizontal scaling.

Applications:

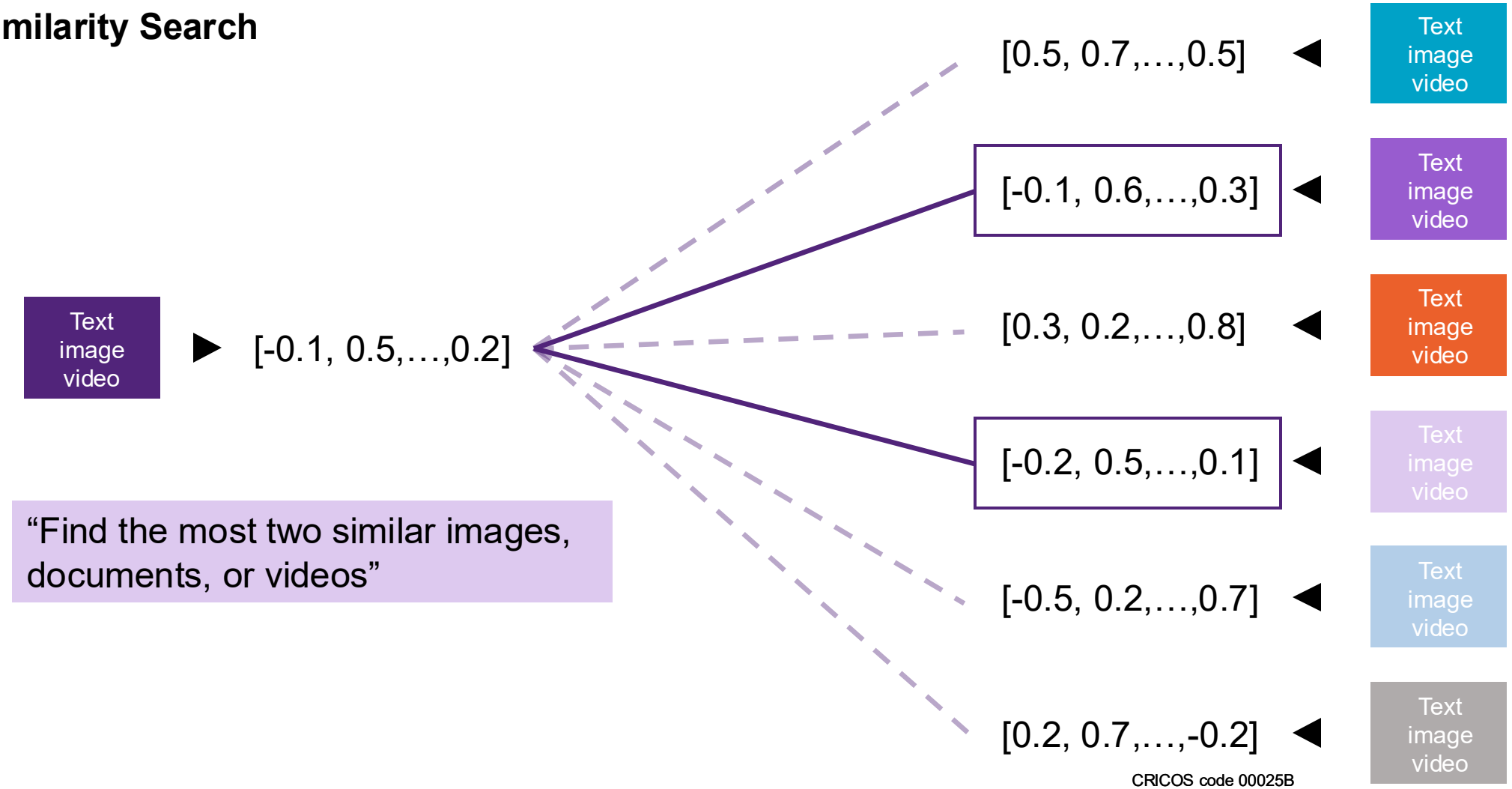
- Relational Database: Traditional databases are pivotal in business applications and transactional systems where structured data is processed.
- Vector Database: Vector databases shine in analyzing vast datasets, supporting fields like scientific research, natural language processing, and multimedia analysis.

Outline

- Introduction to Vector Databases
- Fundamentals of Vector Databases
  - What are vectors?
  - Vector Similarity Measures
- Indexing Techniques for Vector Databases
  - Introduction to Indexing
  - Types of Indexing Techniques
- Vector Databases in the Cloud
- Applications:
  - Retrieval Augmented Generation
  - Anomaly Detection

Vector Similarity Measures

Similarity Search

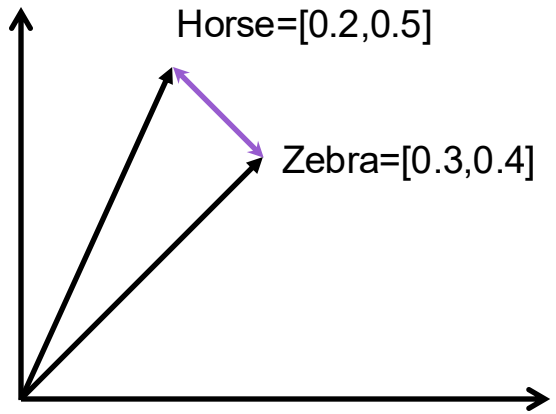


"Find the most two similar images, documents, or videos"

Vector Similarity Measures

Vector Similarity Measure: L2 (Euclidean)

- Ideal for preserving magnitude



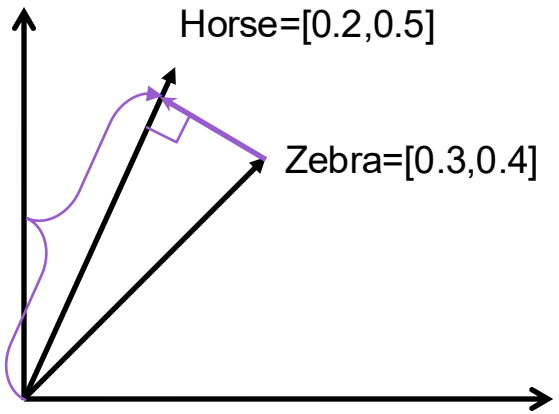
$$\mathrm{d}(\mathrm{p},\mathrm{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

$$\mathrm{d}(\mathrm{Horse},\mathrm{Zebra}) = \sqrt{(0.2-0.3)^2 + (0.5-0.4)^2} = 0.14$$

Vector Similarity Measures

Vector Similarity Measure: Inner Product

- Ideal for preserving both magnitude and orientation



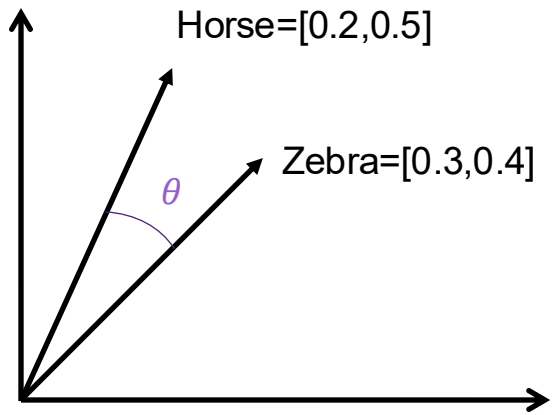
$$\mathrm{a} \cdot \mathrm{b} = \sum_{i=1}^{n} a_i b_i$$

$$\mathrm{Horse} \cdot \mathrm{Zebra} = (0.2 * 0.3) + (0.5 * 0.4) = 0.35$$

Vector Similarity Measures

Vector Similarity Measure: Cosine distance

- Ideal for preserving orientation



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

$$\cos(Horse, Zebra) = \frac{(0.2 * 0.3) + (0.5 * 0.4)}{\sqrt{0.2^2 + 0.5^2} + \sqrt{0.3^2 + 0.4^2}} = 0.966$$

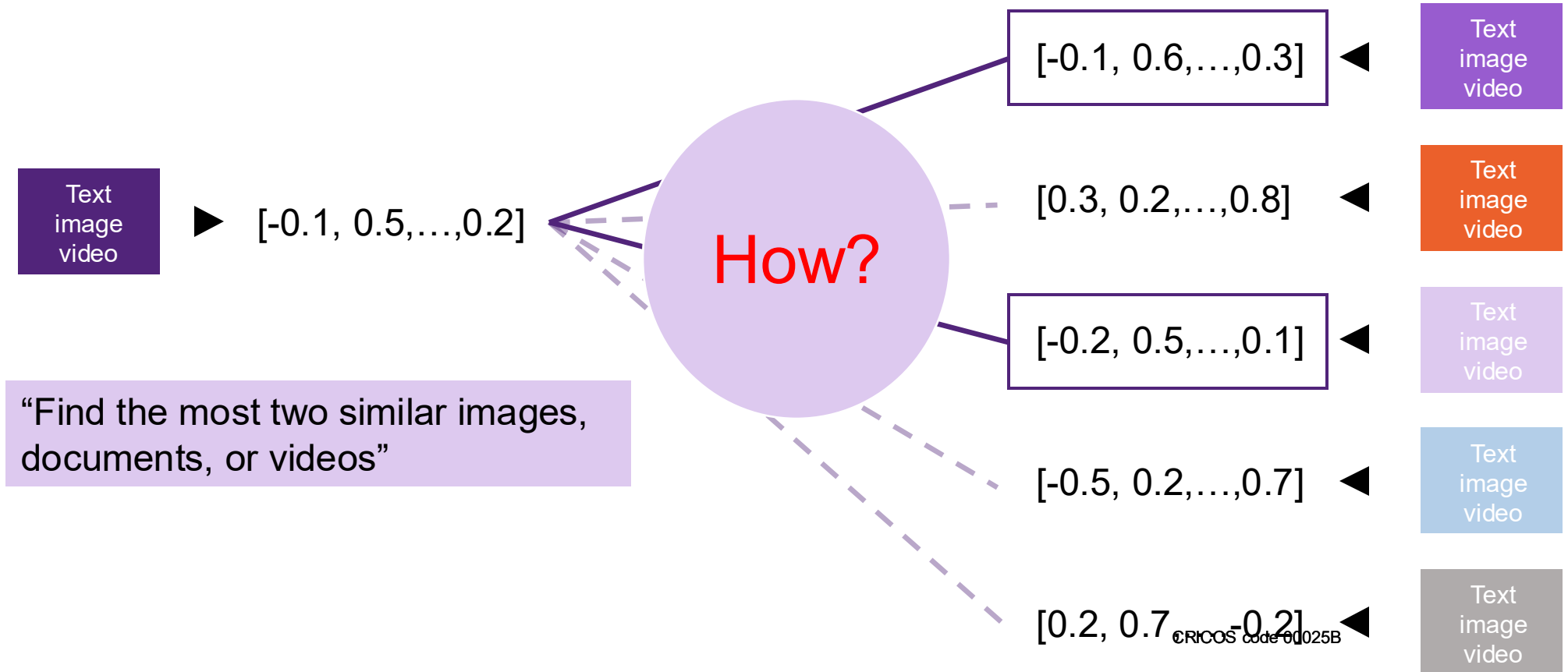
Outline

- Introduction to Vector Databases
- Fundamentals of Vector Databases
  - What are vectors?
  - Vector Similarity Measures
- Indexing Techniques for Vector Databases
  - Introduction to Indexing
  - Types of Indexing Techniques
- Vector Databases in the Cloud
- Applications:
  - Retrieval Augmented Generation
  - Anomaly Detection

Indexing for Vector Databases

Vector Index

- A vector index is a data structure used in computer science and information retrieval to efficiently store and retrieve high-dimensional vector data, enabling fast similarity searches and nearest neighbour queries.

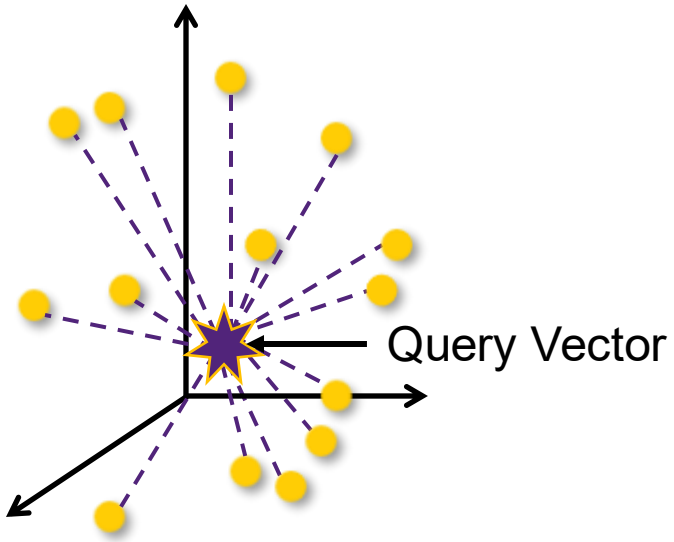


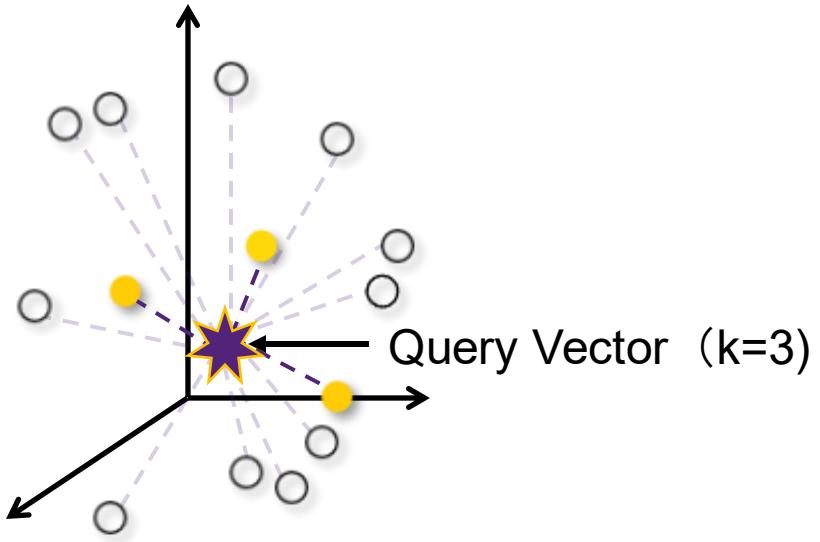
"Find the most two similar images, documents, or videos"

Indexing for Vector Databases

Flat Index (Brute Force**)**

- Flat indices are a *direct representation* of the vector embedding
- Deliver the best accuracy of all indexing methods, but notoriously slow
- Search is *exhaustive:* it's performed from the query vector across *every single vector embedding* and distances are calculated for each pair.





Indexing for Vector Databases

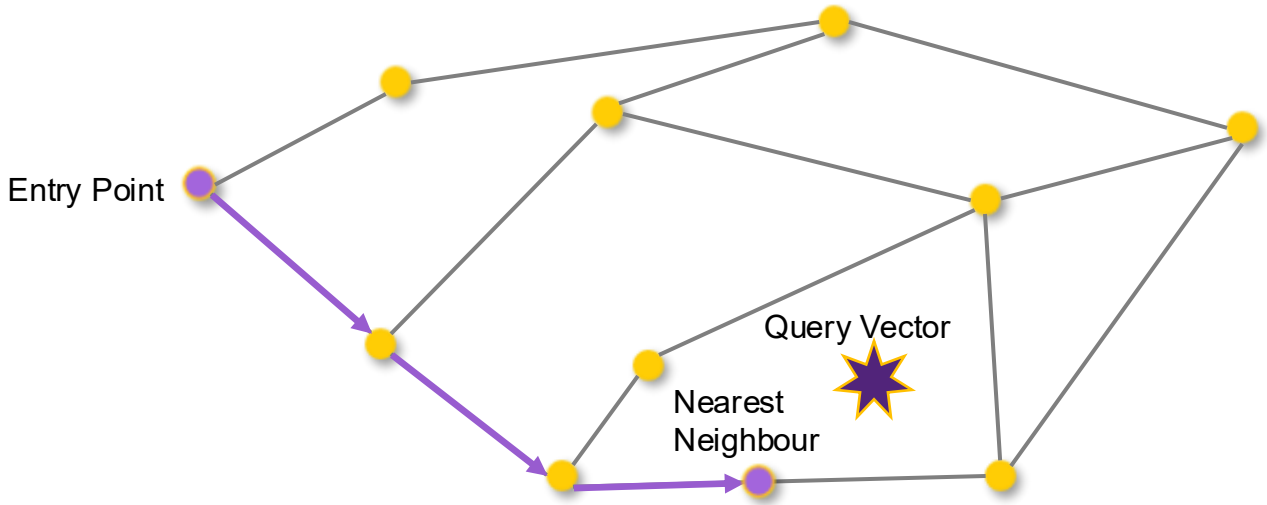
Flat Index (Brute Force**)** is desirable when**:**

- **Low-dimensional data**: With low-dimensional vectors, typically up to a few dozen dimensions, flat indices can be sufficiently fast while maintaining high accuracy.
- **Small-Scale Databases**: When the database contains a relatively small number of vectors, a flat index can be sufficient to provide quick retrieval times.
- **Simple Querying**: If the primary use case involves simple queries, a flat index can offer competitive performance compared to other indices without the complexity
- **Real-time Data Ingestion**: When vectors are continuously added to the database, they must be indexed quickly. Due to the simplicity of flat indexing, minimal computation is needed to generate the new indices.
- **Low Query Volume**: If the rate of queries being performed on the database is low, a flat index can handle these queries effectively
- **Benchmarking Comparisons**: In situations where you want to evaluate the accuracy of other index methods, using the perfectly accurate flat index can be used as a benchmark for comparison purposes.

Indexing for Vector Databases

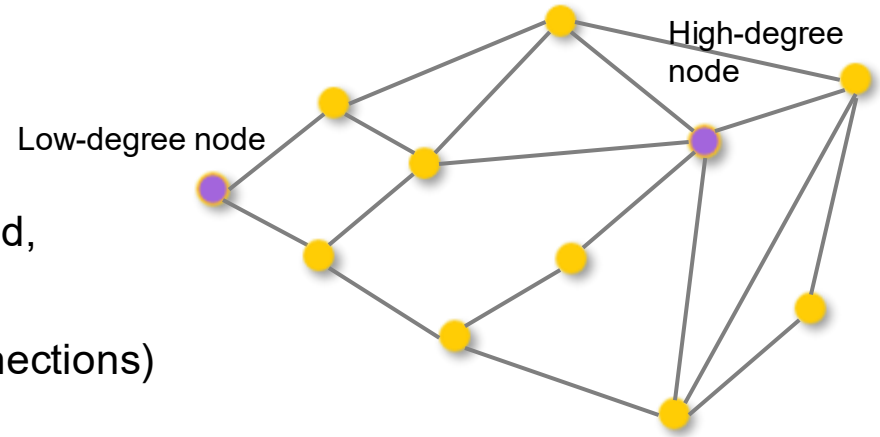
Hierarchical Navigable Small Worlds (HNSW)

- HNSW is a *graph indexing* method, which use nodes and edges to construct a network-like structure
- Nodes represent the vector embeddings, the edges represent the relationships between embeddings
- HNSW is a “proximity graph” where two embedding vertices are linked based on their proximity — often defined by Euclidean Distance
- Starting at a predefined “entry point”, the algorithm traverses connected friend vertices until it finds the nearest neighbour to the query vector



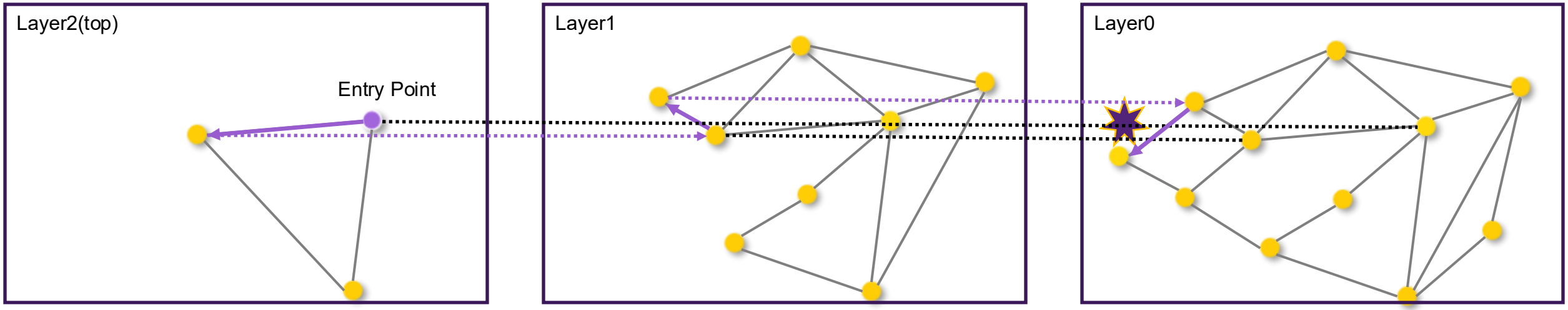
Malkov, Yu A., and Dmitry A. Yashunin. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs." *IEEE transactions on pattern analysis and machine intelligence* 42.4 (2018): 824-836.

Indexing for Vector Databases



Hierarchical Navigable Small Worlds (HNSW)

- HNSW creates a multi-layered graph structure where each layer is a simplified, navigable small world network.
- The entry point is typically on **high-degree vertices** (vertices with many connections) to reduce the chance of stopping early by starting on **low-degree vertices**
- The number of degrees on the entry vertex can be specified and is typically balanced between recall and search speed.



Indexing for Vector Databases

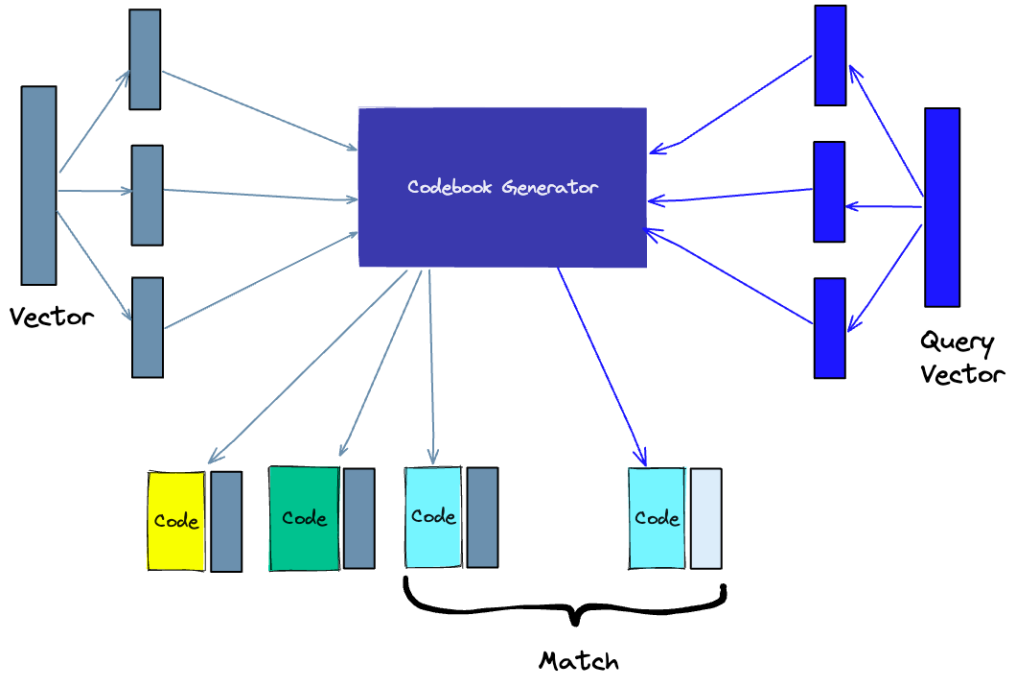
Hierarchical Navigable Small Worlds (HNSW) is desirable when:

- **High-Dimensional Data**: HNSW is specifically designed for high-dimensional data, typically hundreds or thousands of dimensions.
- **Efficient Nearest Neighbour Search:** HNSW is well-suited for applications where quickly finding the most similar data points to a given query is critical, such as recommendation systems, content-based image retrieval, and natural language processing tasks.
- **Approximate Nearest Neighbour Search:** While HNSW is primarily designed for accurate nearest neighbour search, it can also be used for approximate nearest neighbor search tasks where the user seeks to minimize search computation cost.
- **Large-Scale Databases:** HNSW is designed to scale well with large datasets.
- **Real-time and Dynamic Data:** HNSW is capable of accommodating dynamically changing data, such as real-time updates
- **Highly-Resourced Environments:** HNSW's performance is not solely reliant on the memory of a single machine, making it well-suited to distributed and parallel computing environments.

Indexing for Vector Databases

Inverted File Product Quantization Index:

- A ***lossy*** compression technique for high-dimensional vectors
- It takes the original vector, breaks it up into sub-vectors (smaller chunks**)**, simplifies the representation of each chunk by creating a representative “code” for each chunk, and then puts all the chunks back together - without losing information that is vital for similarity operations.



Indexing for Vector Databases

Inverted File Product Quantization Index:

The process of PQ can be broken down into four steps

- ***Splitting*** - The vectors are broken into sub-vectors (smaller **chunks)**.
- ***Training*** - we build a “codebook” for each chunk. The algorithm generates a pool of potential “codes” that could be assigned to a vector. This “codebook” is made up of the center points of clusters created by performing k-means clustering on each of the vector’s segments. We would have the same number of values in the segment codebook as the value we use for the k-means clustering.
- ***Encoding*** - The algorithm assigns a specific code to each segment. The nearest value in the codebook to each vector segment after the training is complete. PQ code for the segment will be the identifier for the corresponding value in the codebook. we can pick multiple values from the codebook to represent each segment.
- ***Querying*** - Breaks down the query vectors into sub-vectors and quantizes them using the same codebook. Then, it uses the indexed codes to find the nearest vectors to the query vector.

Indexing for Vector Databases

Inverted File Index:

- A fundamental data structure
- Used primarily in the field of information retrieval, such as in search engines and database management systems.
- The term "inverted" is derived from the way the index reverses the mapping of data typically seen in traditional databases.

Consider a simple case with three text documents:

Document 1: "apple banana"
Document 2: "apple apple orange"
Document 3: "banana orange apple“

An inverted index for this set of documents might look like this:

apple: [1, 2, 2, 3]
banana: [1, 3]
orange: [2, 3]

Example Scenario for a Vector Database

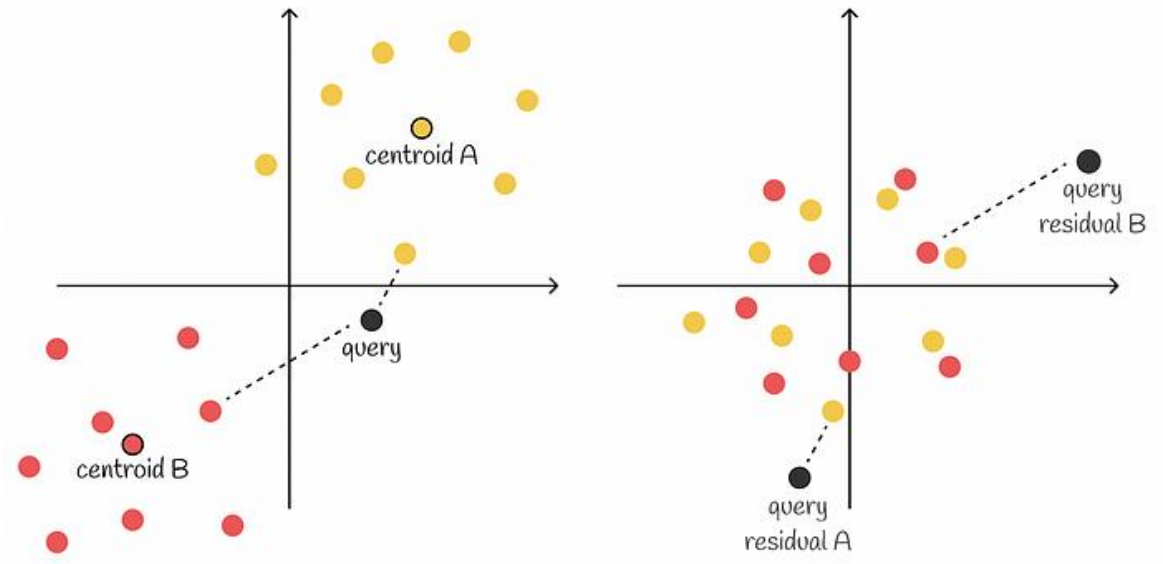
Vector 1: [0.1, 0.2]
Vector 2: [0.1, 0.3]
Vector 3: [0.9, 0.8]
Vector 4: [0.9, 0.7]
Vector 5: [0.1, 0.4]
Vector 6: [0.9, 0.6]

Partition A: [1, 2, 5]
Partition B: [3, 4, 6]

Indexing for Vector Databases

Relative Distances:

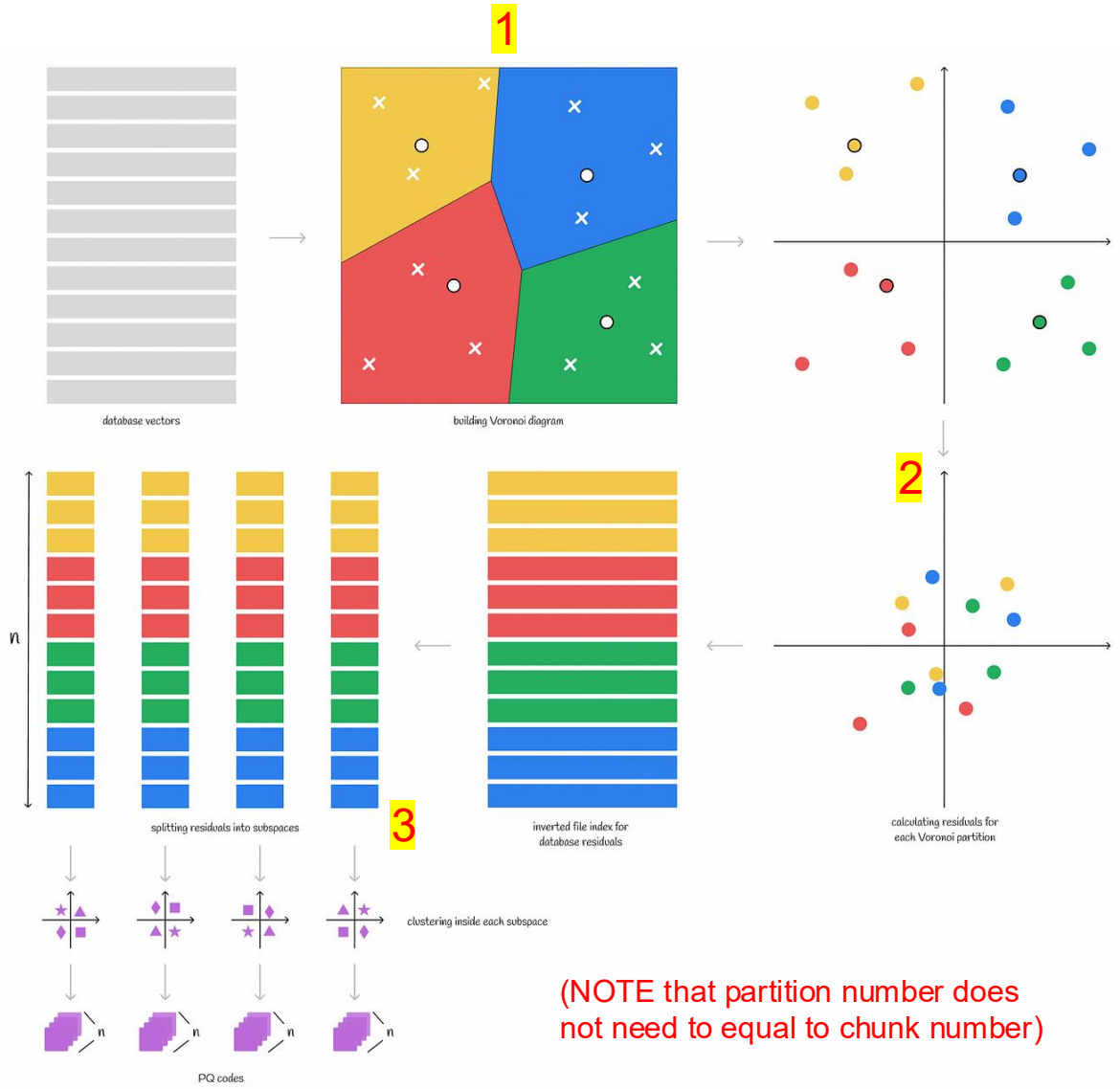
- After subtracting mean values from each cluster, all the points become centered around 0.
- Relative position from query and query residuals to other points of corresponding clusters does not change.



Indexing for Vector Databases

Inverted File Product Quantization Index:

- ***Training*** - we build a "codebook" for each chunk.
  1. An inverted file index is constructed: dividing the set of vectors into k Voronoi partitions. (reduce the search space)
  2. Inside each Voronoi partition, each vector subtracted the coordinates of its centroid vector.
  3. The product quantization algorithm is run on vectors from all the partitions:
     a. Splitting high-dimensional vectors into smaller sub-vectors.
     b. Quantizing these sub-vectors into a finite number of clusters or bins, each represented by a centroid.
     c. Using a codebook for each sub-vector to enable efficient storage and approximate distance calculation.

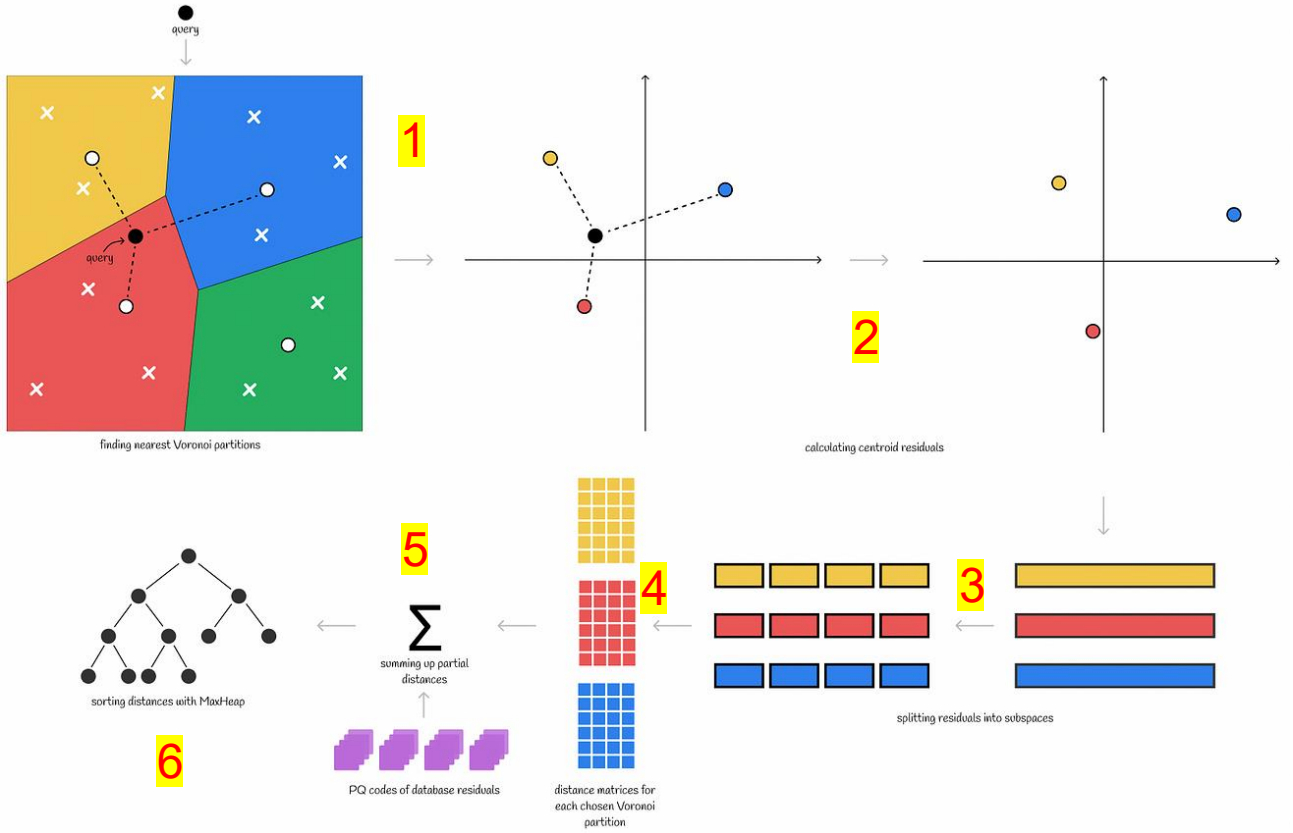


(NOTE that partition number does not need to equal to chunk number)

Indexing for Vector Databases

Inverted File Product Quantization Index:

- ***Inference***
  1. Find k nearest centroids of Voronoi partitions.
  2. Calculate the query residual separately for k Voronoi partitions
  3. The query residual is then split into subvectors.
  4. Calculate the partial distance to each partition
  5. Partial distances are summed up.
  6. Maintaining a MaxHeap data structure for sorting the results. (Store n smallest structure at each step)



Outline

- Introduction to Vector Databases
- Fundamentals of Vector Databases
- Indexing Techniques for Vector Databases
  - Introduction to Indexing
  - Types of Indexing Techniques
- Vector Databases in the Cloud
  - Types of Vector Database
- Applications:
  - Retrieval Augmented Generation
  - Anomaly Detection

Types of Vector Databases

In-memory vector database (RedisAI, Torchserve**)**

- Store vectors directly in memory
- Enable swift read-and-write operations
- Support real-time analytics and recommendation systems

Disk-based vector database (Annoy, Milvus, ScanNN**)**

- Store vectors on disk
- Suitable for large data sets
- Using indexing and compression techniques

Distributed vector database (FAISS, Elastics Search With Vector Plugin, Dask-ML**)**

- Spread vector data across multiple nodes or servers
- Allows for horizontal scalability and fault tolerance
- Suitable for managing massive data sets and high-throughput tasks

Types of Vector Databases

In-memory vector database

Disk-based vector database

Distributed vector database

Graph-based vector database (Neo4j, TigerGraph, Amazon Neptune**)**

- Model data as a graph
- Nodes and edges represent vector attributes or embeddings
- Excel at capturing complex relationships
- Facilitate graph analytics

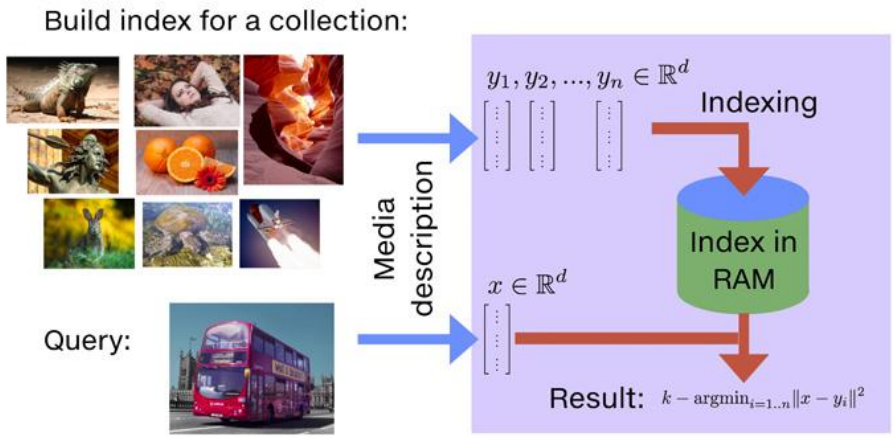
Time-series vector database (InfluxDB, TimescaleDB, Prometheus**)**

- Represent data collected over time as vectors
- Serve as tools for temporal pattern and anomaly analysis

Storage Types of Vector Databases

In-memory data store

- Store vectors directly in RAM
- Enable swift read-and-write operations
- Support real-time analytics and recommendation systems
- Pros and Cons:
  - High performance and low latency; efficient vector operations
  - Limited by memory constraints
- Representative Products:
  - FAISS, TorchServe, RedisVL

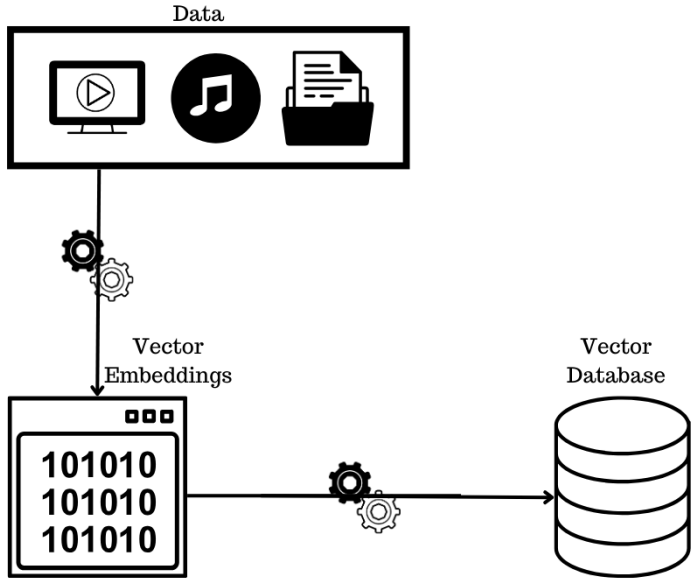


FAISS
Scalable Search With Facebook AI

Storage Types of Vector Databases

Disk-based data store

- Store data on disk
- Slower retrieval times than in-memory solutions
- Suitable for larger datasets
- Pros and Cons:
  - Scalability - Large Data Capacity; Cost-Effective - Storage on disk is typically cheaper than RAM
  - Slower Access Times
- Representative Products:
  - Milvus, ANNOY, Google ScaNN



Milvus

spotify/**annoy**

Approximate Nearest Neighbors in C++/Python optimized for memory usage and loading/saving to disk

65 Contributors | 4k Used by | 13k Stars | 1k Forks

Faiss - a library for efficient similarity search and clustering of dense vectors

- Developed by Facebook AI Research (FAIR)
- **Efficiency at Scale:** It handles vast oceans of data with the grace and efficiency of an expert.
- **Versatility in Search Methods:** It doesn't just stick to one style; instead, it offers a rich palette of search techniques, each tailored to different types of data landscapes.
- **Compatibility and Integration:** It integrates effortlessly with a variety of programming languages and frameworks, inviting both Python enthusiasts and C++ experts to waltz along.
- **GPU Acceleration:** Faiss with GPU acceleration. It takes the raw power of GPUs and channels it into your search tasks, transforming a steady jog into a supersonic flight.

Faiss - Demo

Search within the Oxford Dictionary

Oxford English Dictionary.txt - Notepad

File Edit Format View Help

```
A



A-  prefix (also an- before a vowel sound) not, without (amoral). [greek]

Aa  abbr. 1 automobile association. 2 alcoholics anonymous. 3 anti-aircraft.

Aardvark  n. Mammal with a tubular snout and a long tongue, feeding on termites. [afrikaans]

Ab-  prefix off, away, from (abduct). [latin]

Aback  adv.  take aback surprise, disconcert. [old english: related to *a2]

Abacus  n. (pl. -cuses) 1 frame with wires along which beads are slid for calculating. 2 archit. Fla

Abaft  naut. —adv. In the stern half of a ship. —prep. Nearer the stern than. [from *a2, -baft: see

Abandon  —v. 1 give up. 2 forsake, desert. 3 (often foll. By to; often refl.) Yield to a passion, an

Abandoned  adj. 1 deserted, forsaken. 2 unrestrained, profligate.

Abase  v. (-sing) (also refl.) Humiliate, degrade.  abasement n. [french: related to *ad-, *base2]

Abashed  predic. Adj. Embarrassed, disconcerted. [french es- *ex-1, baïr astound]

Abate  v. (-ting) make or become less strong etc.; diminish.  abatement n. [french abatre from latir

Abattoir  n. Slaughterhouse. [french abatre fell, as *abate]

Abbacy  n. (pl. -ies) office or jurisdiction of an abbot or abbess. [latin: related to *abbot]
```

```python
start_time = time.perf_counter()
with open('Oxford English Dictionary.txt', 'r') as file:
    # Create an empty list to store the lines
    lines = []
    # Iterate over the lines of the file
    for line in file:
        # Remove the newline character at the end of the line
        line = line.strip()
        # Append the line to the list
        lines.append(line)
lines = list(set(lines))
lines = lines[:3000]
print(len(lines))
end_time = time.perf_counter()
elapsed_time = end_time - start_time
print("Elapsed time: ", elapsed_time)
lines
```

```
3000
Elapsed time:  0.03438336099497974

['',
 'Everywhere  adv. 1 in every place. 2 colloq. In many places.',
 'Rom.  Abbr. Roman (type).',
 'Fake  —n. False or counterfeit thing or person. —adj. Counterfeit; not genuine
gn (a feeling, illness, etc.). [german fegen sweep]',
 'Show-off  n. Colloq. Person who shows off.',
 'Fiord  n. (also fjord) long narrow sea inlet, as in norway. [norwegian]',
 'Counselor  n. (brit. Counsellor) 1 adviser. 2 person giving professional guida
 "Parallax  n. 1 apparent difference in the position or direction of an object c
s. [greek, = change]",
```

Faiss - Demo

Preprocessing

```
df = pd.DataFrame(lines, columns=['txt'])
start_time = time.perf_counter()
nlp = spacy.load('en_core_web_sm')
stop_words = nlp.Defaults.stop_words
def preprocess(text):
  doc = nlp(str(text))
  preprocessed_text = []
  for token in doc:
    if token.is_punct or token.like_num or token in stop_words or token.is_space:
      continue
    preprocessed_text.append(token.lemma_.lower().strip())
  return ' '.join(preprocessed_text)
df['Processed Text'] = df['txt'].apply(preprocess)
df
end_time = time.perf_counter()
elapsed_time = end_time - start_time
print("Elapsed time: ", elapsed_time)
```

```
Elapsed time:  15.580718785058707
```

	txt	Processed Text
0		
1	Everywhere adv. 1 in every place. 2 colloq. I...	everywhere adv in every place colloq in many p...
2	Rom. Abbr. Roman (type).	rom abbr roman type
3	Fake —n. False or counterfeit thing or person...	fake n. false or counterfeit thing or person a...
4	Show-off n. Colloq. Person who shows off.	show off n. colloq person who show off
...	...	...
2995	Usage this word is sometimes confused with po...	usage this word be sometimes confuse with policy
2996	Naff adj. Slang 1 unfashionable. 2 rubbishy. ...	naff adj slang unfashionable rubbishy origin u...
2997	Secret ballot n. Ballot in which votes are ca...	secret ballot n. ballot in which vote be cast ...
2998	Vertical —adj. 1 at right angles to a horizon...	vertical adj at right angle to a horizontal pl...
2999	Usage tortuous should not be confused with to...	usage tortuous should not be confuse with tort...

Faiss - Demo

Create embeddings for each sentence

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2', device="cpu")
start_time = time.perf_counter()
df['Embedding'] = df['Processed Text'].apply(model.encode)
vector = model.encode(df['Processed Text'])
end_time = time.perf_counter()
elapsed_time = end_time - start_time
print("Elapsed time: ", elapsed_time)
```

```
Elapsed time:  22.33752644294873
```

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2', device="cuda")
start_time = time.perf_counter()
df['Embedding'] = df['Processed Text'].apply(model.encode)
vector = model.encode(df['Processed Text'])
end_time = time.perf_counter()
elapsed_time = end_time - start_time
print("Elapsed time: ", elapsed_time)
```

```
Elapsed time:  11.104513209080324
```

```
vector
```

```
array([[-0.11883845,  0.04829863, -0.00254813, ...,  0.12640949,
         0.04654901, -0.01571725],
       [ 0.07155535, -0.01467227,  0.01904674, ...,  0.10639305,
        -0.05447157, -0.03420812],
       [-0.12742081,  0.04474863, -0.00671719, ..., -0.07745058,
         0.03997674,  0.03948158],
       ...,
       [-0.09131163,  0.02018466, -0.06345702, ...,  0.02803741,
         0.00759016,  0.0245064 ],
       [-0.05358332,  0.02261099, -0.07105472, ...,  0.07828537,
        -0.07352977, -0.03810928],
       [ 0.00820408,  0.01046418, -0.08309177, ..., -0.0272886 ,
         0.05790045, -0.02062473]], dtype=float32)
```

```
vector.shape
```

```
(3000, 384)
```

Faiss - Demo

Faiss stores index and vector embeddings

```
dim = vector.shape[1]


import faiss
index = faiss.IndexFlatL2(dim)

index.add(vector)
```

```
search_query = 'The University of Queensland'
test_pre = preprocess(search_query)
encode_pre = model.encode(test_pre)
encode_pre.shape

#FAISS expects 2d array, so next step we are
# converting encode_pre to a 2D array
import numpy as np
svec = np.array(encode_pre).reshape(1,-1)

#We will get euclidean distance and index of
# the nearest neighbours
distance,pos = index.search(svec,k=10)
```

```
df.txt.iloc[pos[0]]

2252    Campus  n. (pl. -es) 1 grounds of a university...
2364    Ucca  abbr. Universities central council on ad...
1158                      Bd  abbr. Bachelor of divinity.
715     Aborigine  n. (usu. In pl.) 1 aboriginal inhab...
1863                      D.sc.  Abbr. Doctor of science.
2819      Des  abbr. Department of education and science.
2100    Summer school  n. Course of summer lectures et...
2376    Postgraduate  —n. Person engaged in a course o...
1920      Rugby union  n. Amateur rugby with teams of 15.
86              Doe  abbr. Department of the environment.
Name: txt, dtype: object
```

Outline

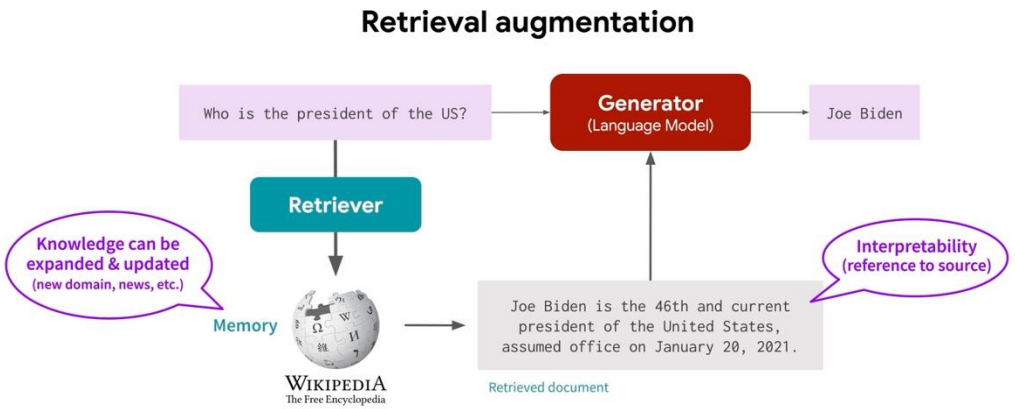
- Introduction to Vector Databases
- Fundamentals of Vector Databases
- Indexing Techniques for Vector Databases
  - Introduction to Indexing
  - Types of Indexing Techniques
- Vector Databases in the Cloud
  - Types of Vector Database
- Applications:
  - Retrieval Augmented Generation
  - Anomaly Detection

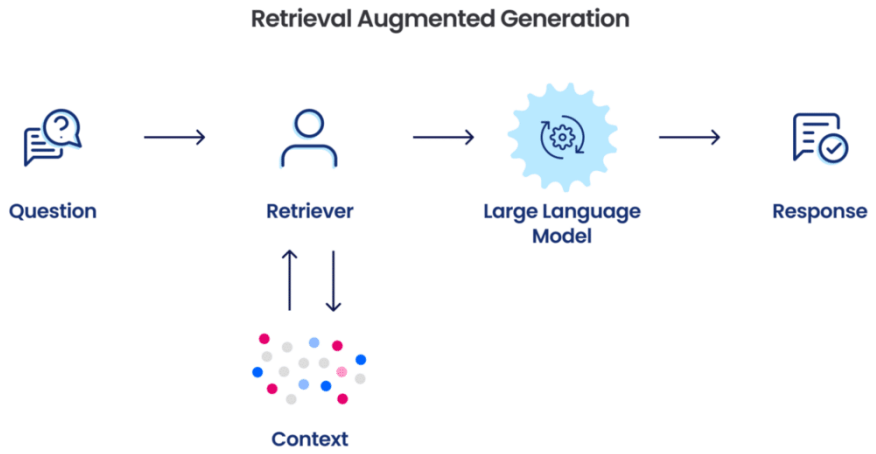
Retrieval Augmented Generation (RAG)

Definition:

- RAG is a method that combines *retrieval-based models* with *generative models* to enhance the generation of *contextually relevant and accurate* responses.

- RAG uses a retrieval mechanism to fetch relevant documents or pieces of information which are then used to condition the generative model.

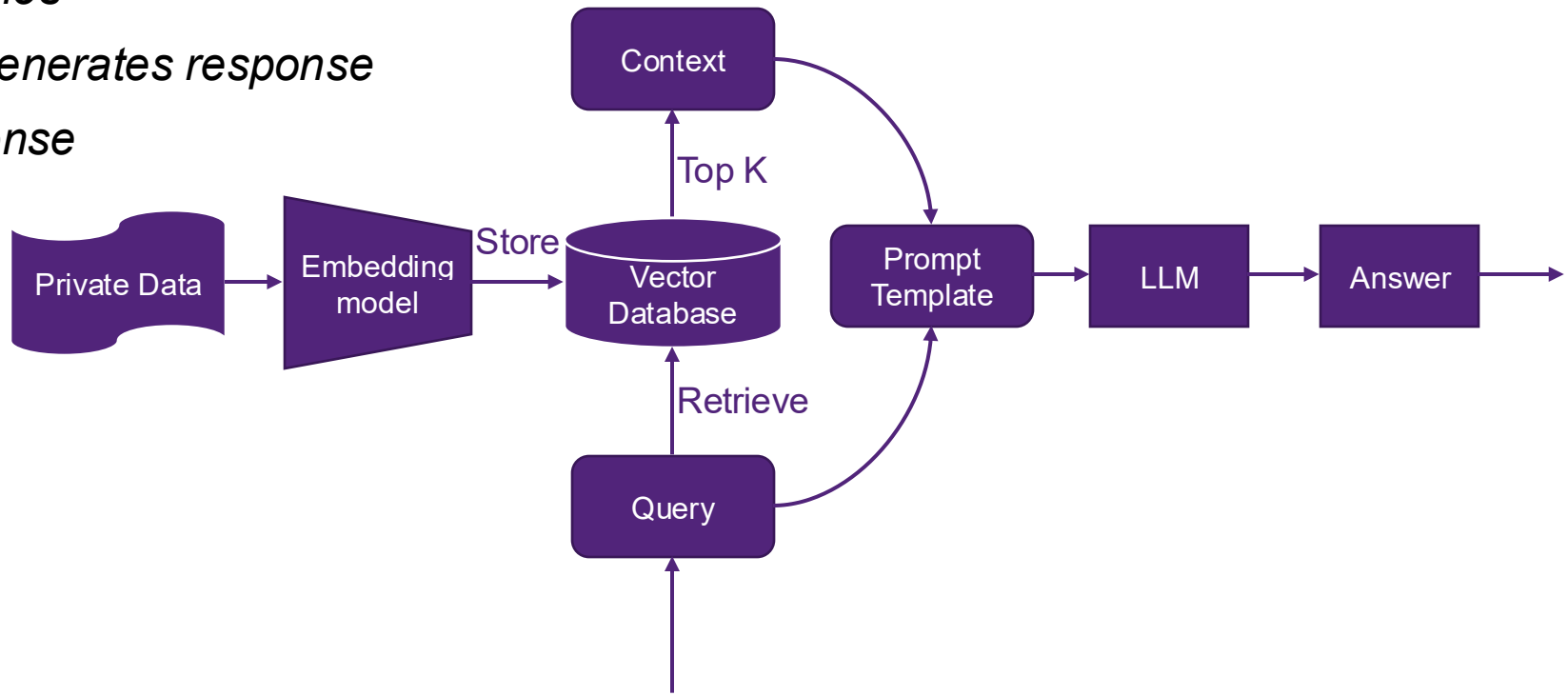




Retrieval Augmented Generation (RAG)

Generalized RAG Approach

- *Step 1 – User sends query*
- *Step 2 – App forwards queries*
- *Step 3 – RAG retrieves + generates response*
- *Step 4 – LLM returns response*



Retrieval Augmented Generation (RAG)

Limitation of Traditional Generative Models:

- **Contextual and Factual Limitations**: Traditional generative models, such as GPT-3, are known for their ability to generate fluent and coherent text based on large-scale training on diverse data sets. However, they often *struggle with factual accuracy and context relevance* because they generate responses based solely on learned patterns and associations rather than real-time, verifiable data.

- **Lack of Specificity**: These models sometimes fail to provide detailed or specific information that is only available in up-to-date or niche documents, as their responses are *constrained by the scope of their training data*.

Retrieval Augmented Generation (RAG)

Addressing Limitations with RAG:

- **Enhancing Factual Accuracy and Relevance**: RAG addresses these issues by incorporating a retrieval step that *fetches relevant and factual information from vector databases*. This ensures that the responses are not only contextually relevant but also factually grounded.

- **Dynamic Content Integration**: By leveraging dynamic and continuously updated databases stored in the cloud, RAG systems can *integrate the latest information into their responses*, which is particularly valuable in fast-changing fields such as news, scientific research, and market trends.

RAG DEMO

Jupyter Notebook provided

https://github.com/Adesoji1/RAG-System/tree/main

Retrieval Augmented Generation (RAG)

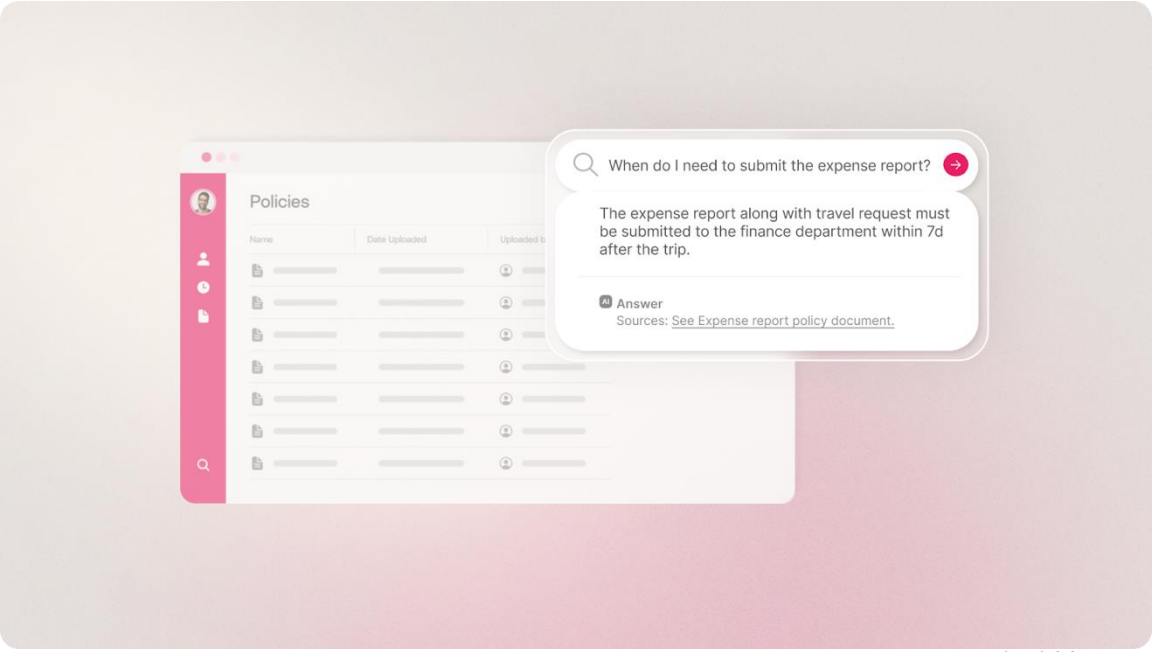
RAG real-world use cases

- ***1. Employee productivity apps -*** *Assist employees in finding and leveraging existing organizational knowledge buried across databases, emails, and documents.*
- ***2. Analysis assistants -*** *Provide insights to employees through document summarization or Q&A by retrieving and synthesizing internal research documentations, such as financial reports, research studies, clinical trials, market trends, competitor analysis, and customer feedback.*
- ***3. Customer support chatbots -*** *Enhance customer support by providing accurate, context-rich responses to customer queries, based on specific user information and organizational documents like help center content & product overviews.*

Retrieval Augmented Generation (RAG)

RAG real-world use cases

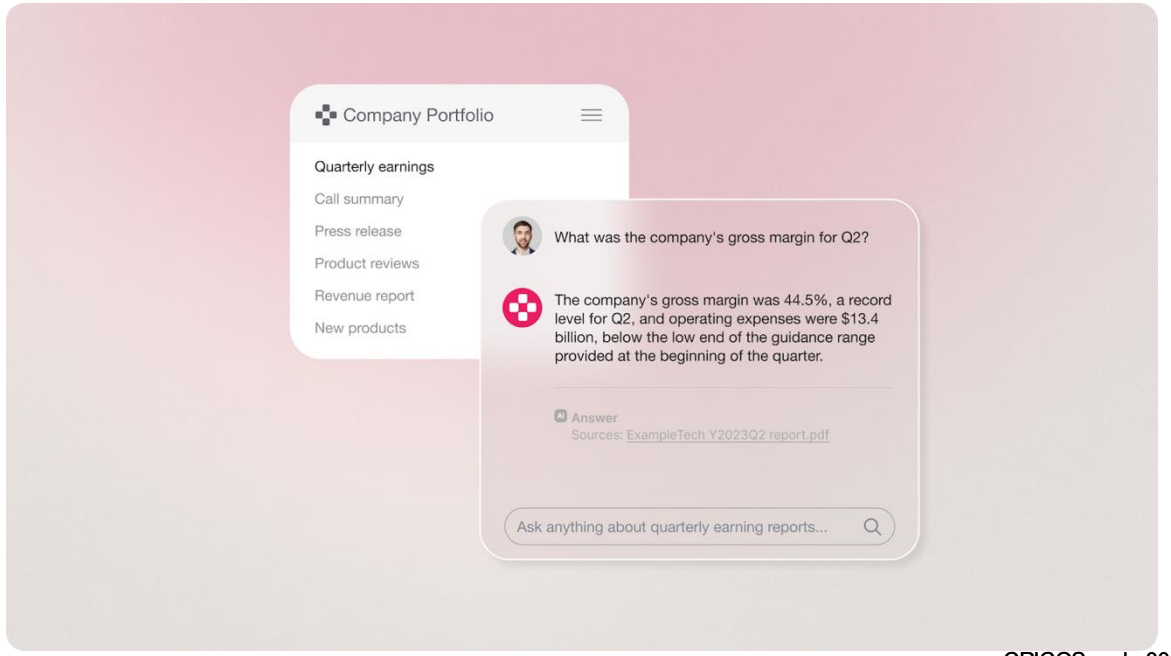
- ***1. Employee productivity apps -*** *Assist employees in finding and leveraging existing organizational knowledge buried across databases, emails, and documents.*



Retrieval Augmented Generation (RAG)

RAG real-world use cases

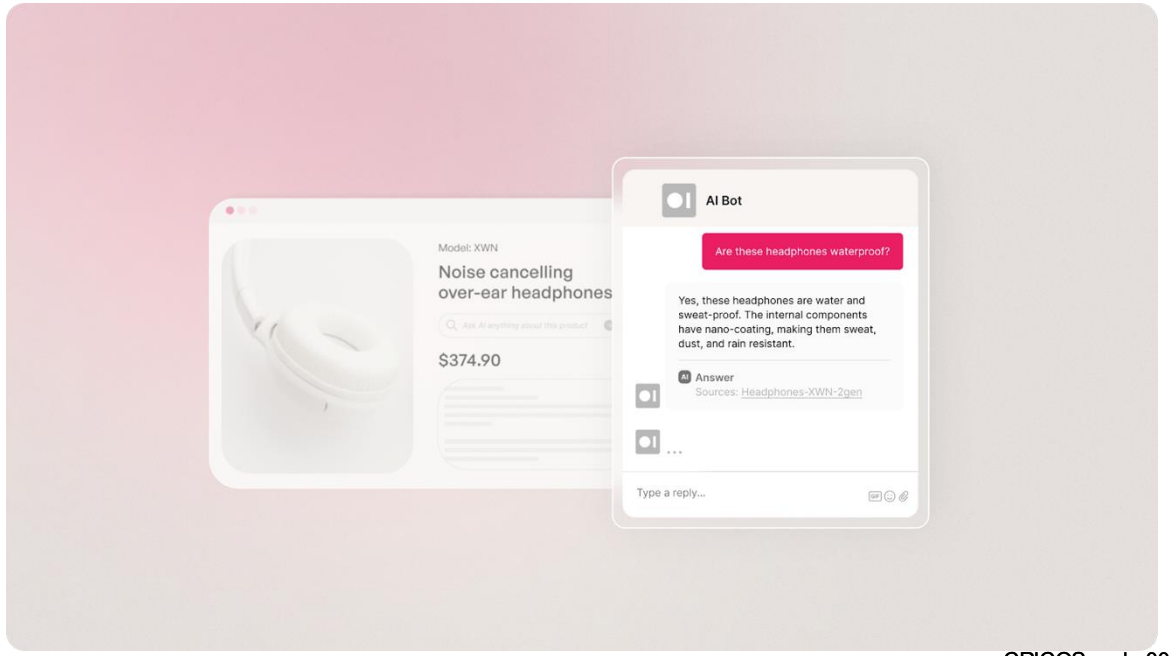
- ***2. Analysis assistants -*** *Provide insights to employees through document summarization or Q&A by retrieving and synthesizing internal research documentations, such as financial reports, research studies, clinical trials, market trends, competitor analysis, and customer feedback.*



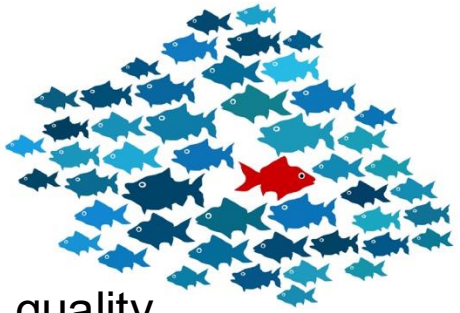
Retrieval Augmented Generation (RAG)

RAG real-world use cases

- ***3. Customer support chatbots -*** *Enhance customer support by providing accurate, context-rich responses to customer queries, based on specific user information and organizational documents like help center content & product overviews.*

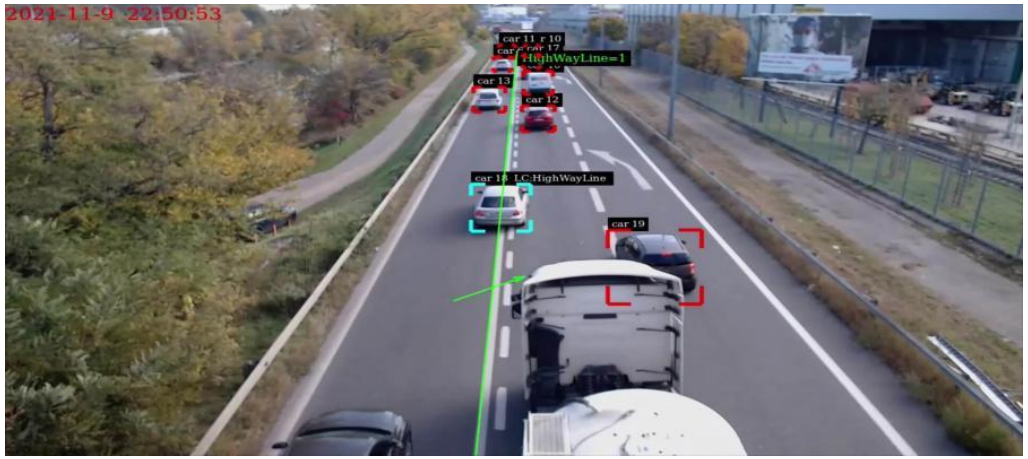


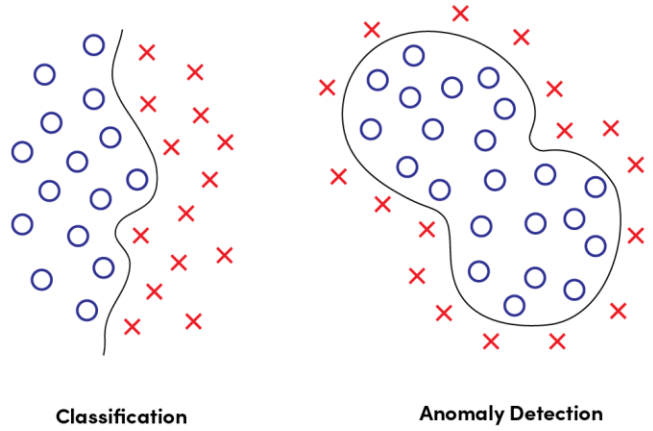
Vector Databases for Anomaly Detection



Why anomaly detection:

- Anomalies can be found in various domains, including fraud detection, network security, quality control and healthcare. This is especially true for data-driven applications and artificial intelligence (AI), where anomaly detection is a critical component.
- However, efficiently identifying these anomalies in high-dimensional data is a complex task. Additionally, anomaly detection poses a significant challenge when it comes to dealing with massive data sets containing complex patterns.





Vector Databases for Anomaly Detection

Challenges of anomaly detection in high-dimensional data:

- **The curse of dimensionality.** High-dimensional data often contains a large number of features, making it more susceptible to the curse of dimensionality. This phenomenon can lead to increased computational complexity and decreased detection accuracy.
- **Sparsity.** High-dimensional data is typically sparse, meaning that most of the dimensions may be empty or contain very little information. This sparsity can make it difficult to distinguish between normal and anomalous behavior.
- **Overfitting.** With a high number of dimensions, models used for anomaly detection may overfit the data, with the possibility of identifying noise as anomalies.

Vector Databases for Anomaly Detection

Why vector Databases for anomaly detection:

- Traditional methods often struggle to efficiently identify anomalies, leaving room for potential risks and missed insights. This is where vector databases can help.
- Vector databases can efficiently handle high-dimensional data and support advanced similarity search algorithms. They're good at representing complex patterns and identifying anomalies in real time, enabling businesses to proactively respond to potential threats.

Role of vector databases in anomaly detection:

- **Efficient storage**. Vector databases optimize storage for high-dimensional vectors, reducing overhead and ensuring that the data remains easily accessible.
- **Scalability for big data**. Anomaly detection often involves processing vast amounts of data. Vector databases can scale horizontally to accommodate growing data sets.
- **Enhanced performance**. The specialized nature of vector databases allows for faster computation of vector-based metrics, accelerating the anomaly detection process.
- **Feature engineering**. Vector databases can support the creation of additional attributes associated with vector data, enhancing the feature set used for anomaly detection models.

References

1. What is vector database? https://cloud.google.com/discover/what-is-a-vector-database

2. Vector databases introduction with chromadb?

3. The Promise of RAG: Bringing Enterprise Generative AI to Life https://www.ai21.com/blog/the-promise-of-rag-bringing-enterprise-generative-ai-to-life

4. https://medium.com/@ypredofficial/faiss-vector-database-be3a9725172f

5. https://lancedb.github.io/lancedb/concepts/index_ivfpq/#putting-it-all-together

6. https://towardsdatascience.com/similarity-search-blending-inverted-file-index-and-product-quantization-a8e508c765fa