RAG

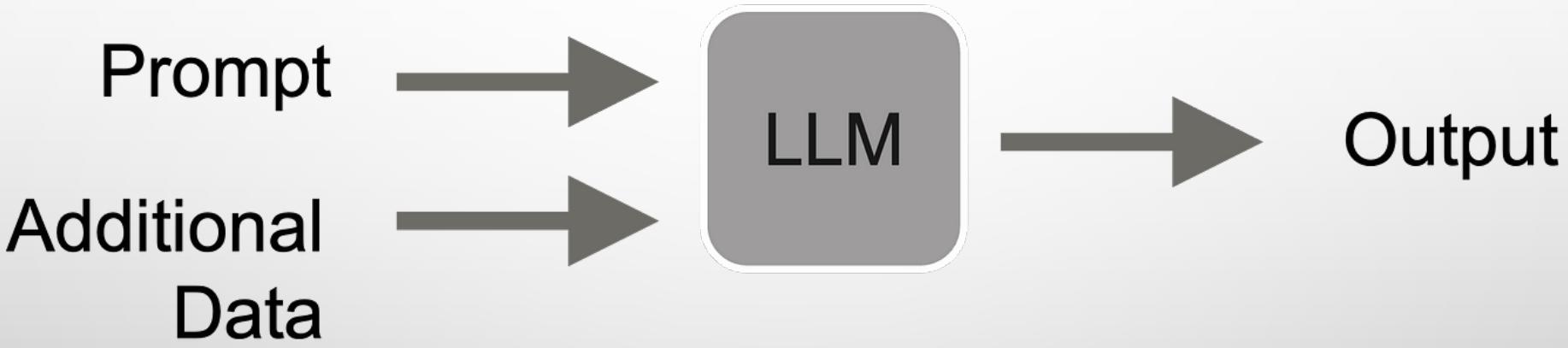
RETRIEVAL AUGMENTED GENERATION

Topics

- What is RAG
- What Are Vector Databases?
- Using ChromaDB

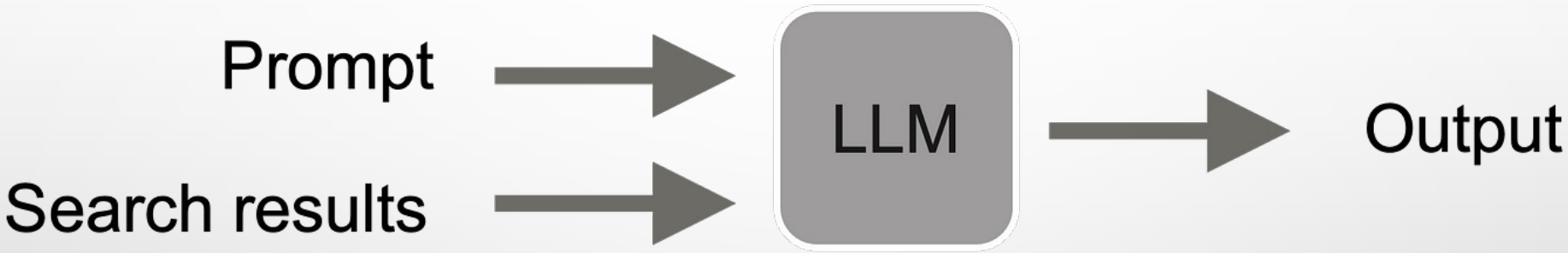
Simple Augmented Generation

You have been doing this!!!



https://sausheong.com/programming-with-ai-rag-27bf5c19daa7

Simple Retrieval Augmented Generation

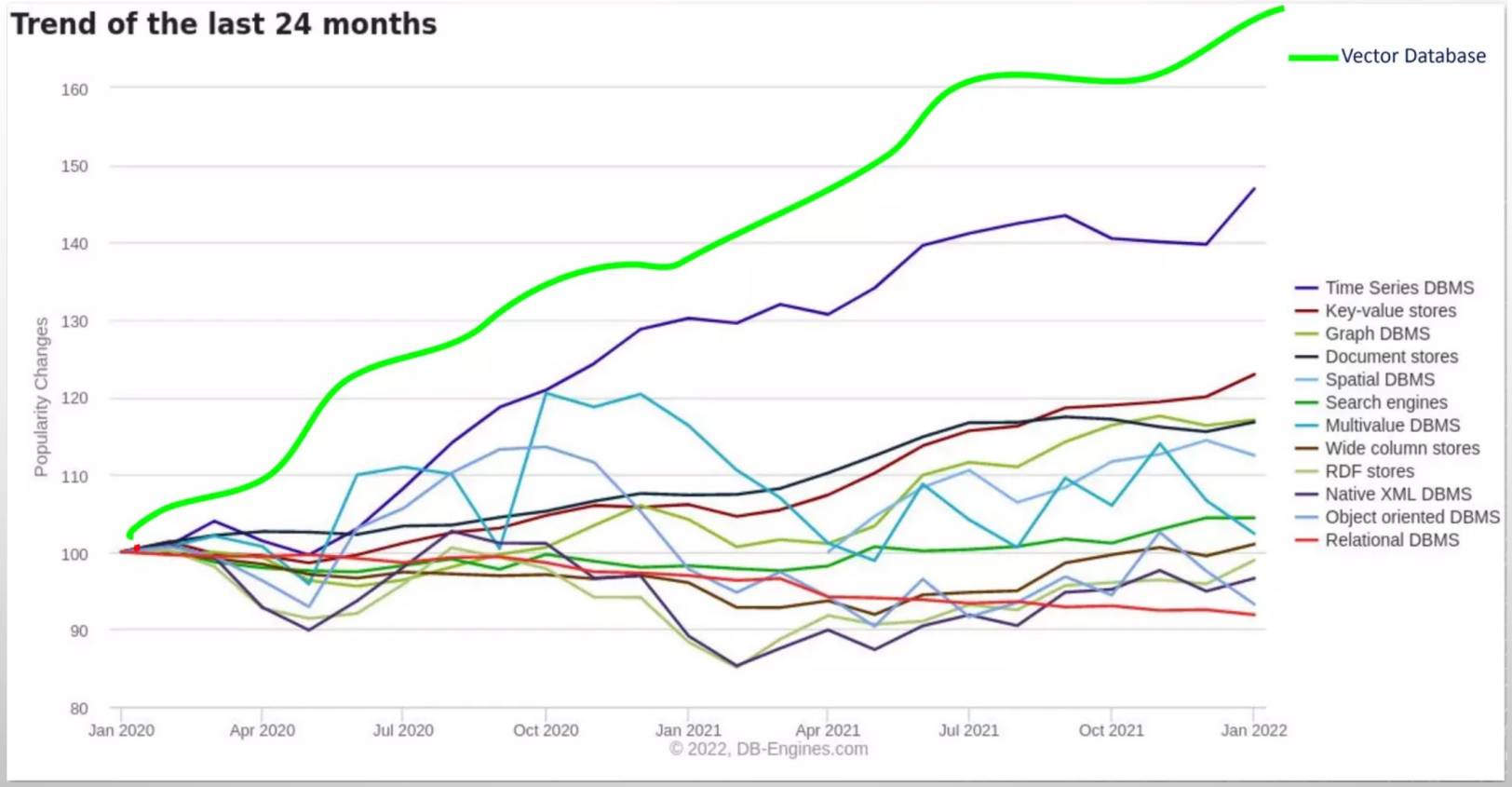


https://sausheong.com/programming-with-ai-rag-27bf5c19daa7

How to find ‘relevant information’ for the LLM

→ Vector Databases

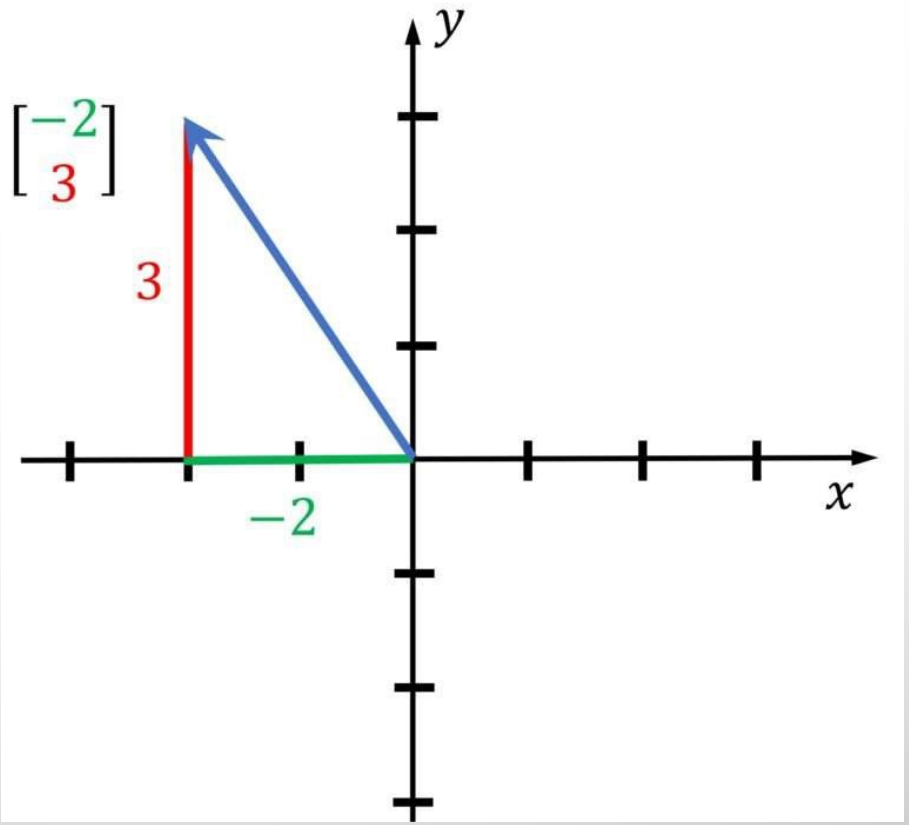
Vector Databases are Newish



https://www.slideshare.net/slideshow/vector-database/251053508

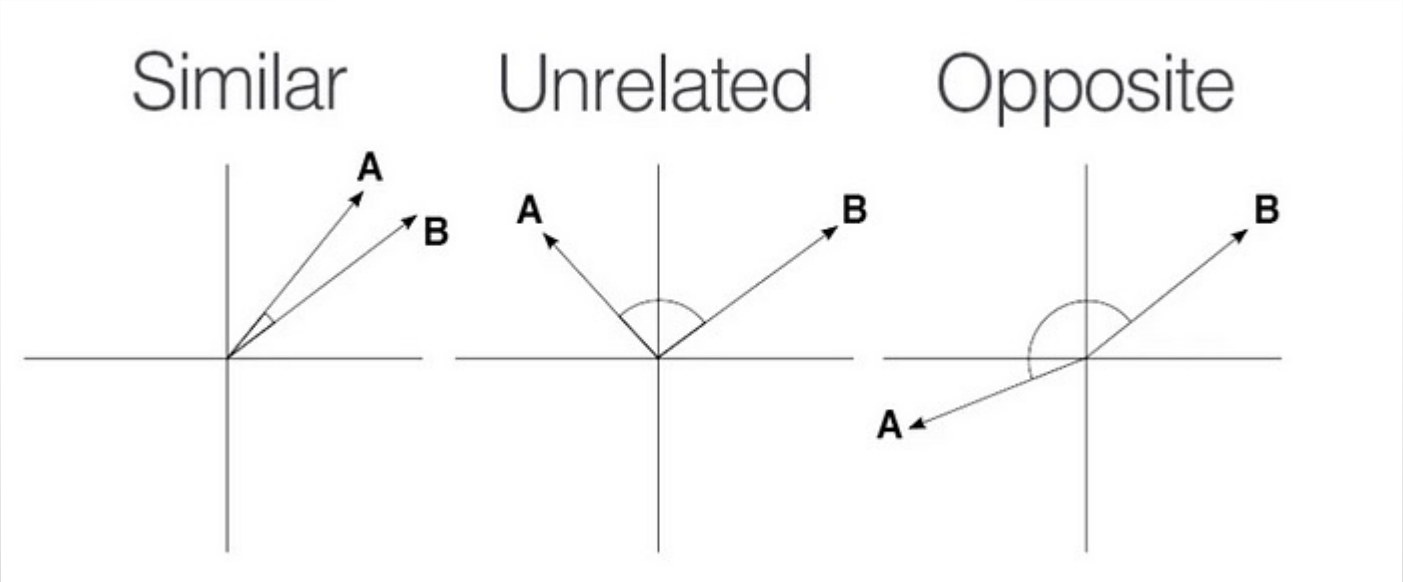
Vectors

Distance in 2D



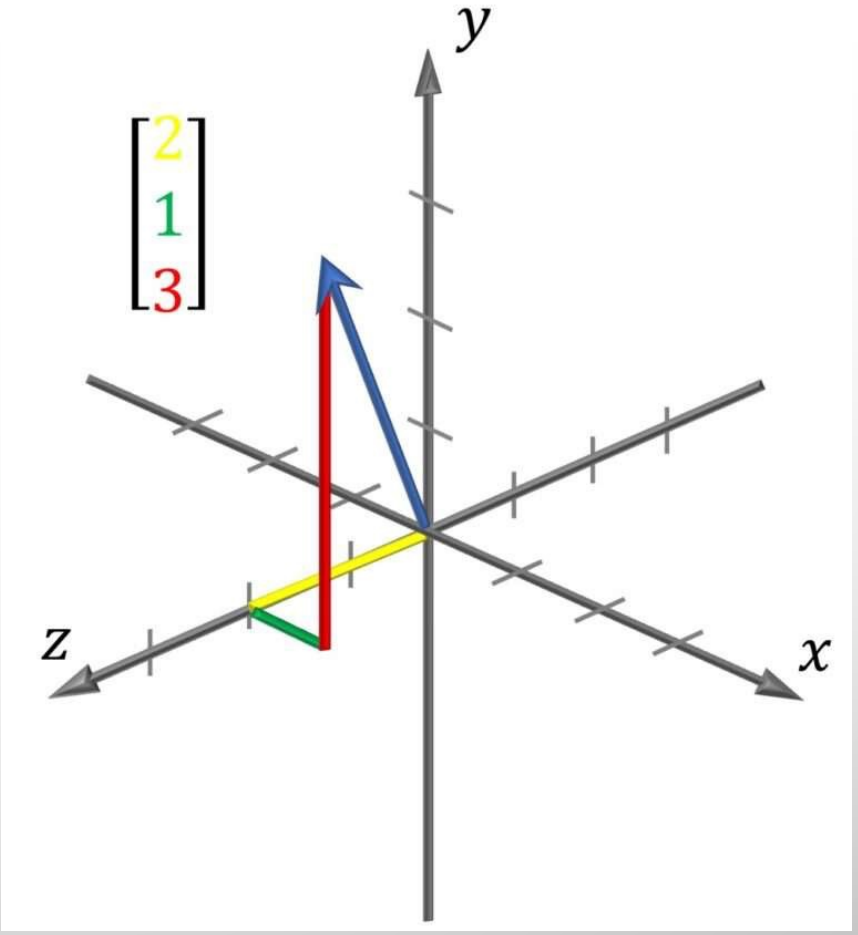
https://www.percona.com/blog/an-introduction-to-vector-databases/

Vector Similarity



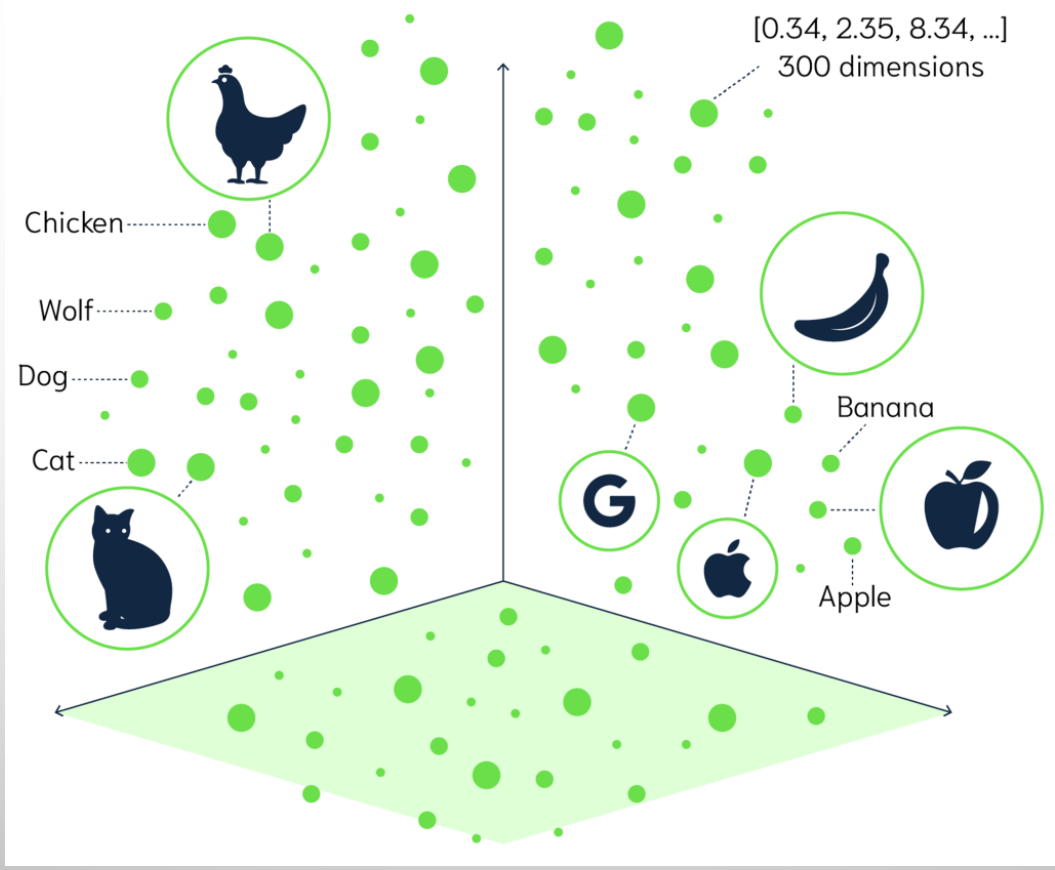
https://sausheong.com/programming-with-ai-rag-27bf5c19daa7

3D Vectors



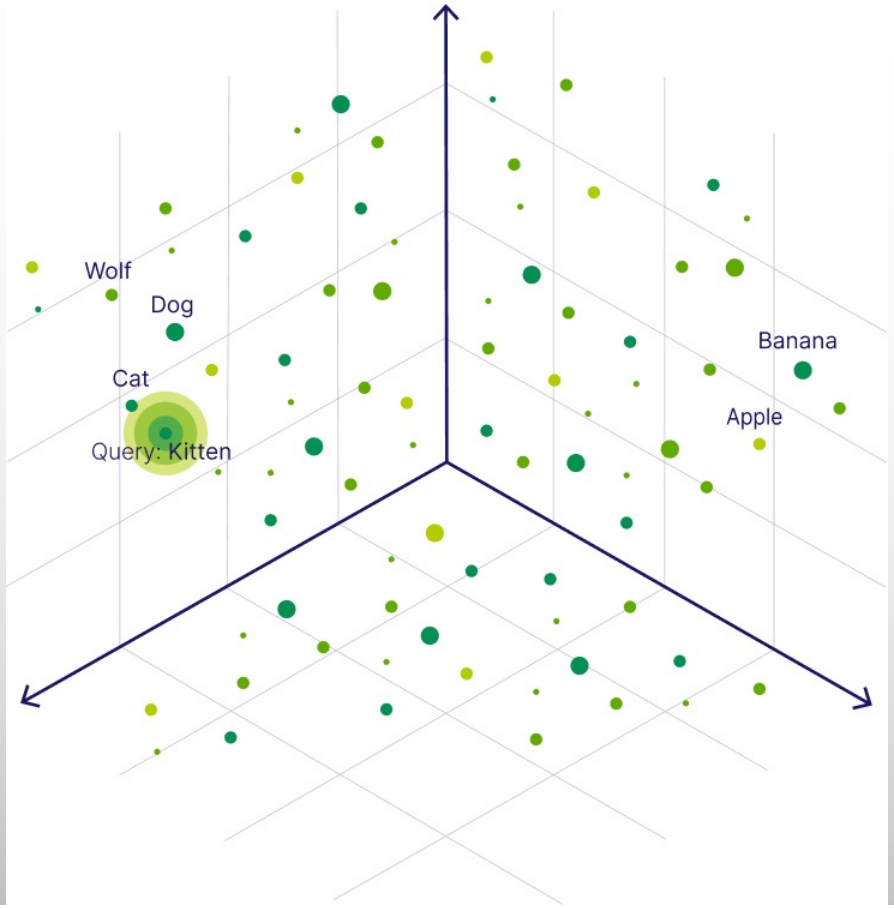
https://www.percona.com/blog/an-introduction-to-vector-databases/

Similarity Search across the vectors



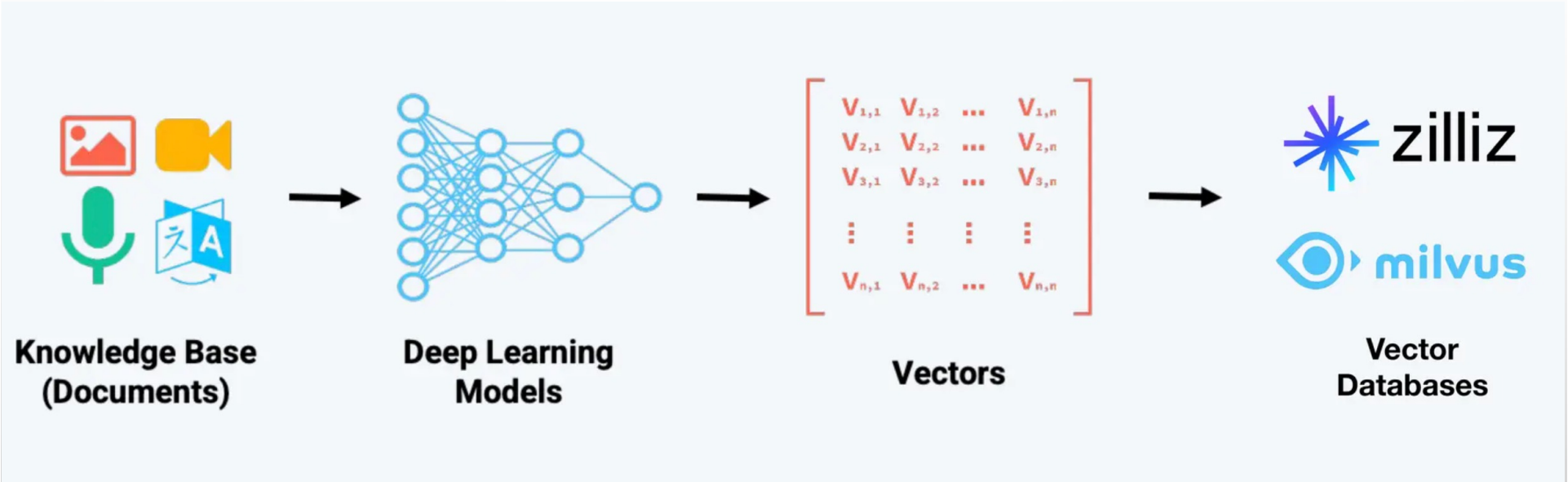
https://weaviate.io/blog/distance-metrics-in-vector-search

Similarity Search across the vectors Vector Database?



https://weaviate.io/blog/what-is-a-vector-database

Where do the Vectors ‘come from’?



https://www.slideshare.net/slideshow/vector-databases-101-an-introduction-to-the-world-of-vector-databases/267310487

Vectors can Represent Complex Information

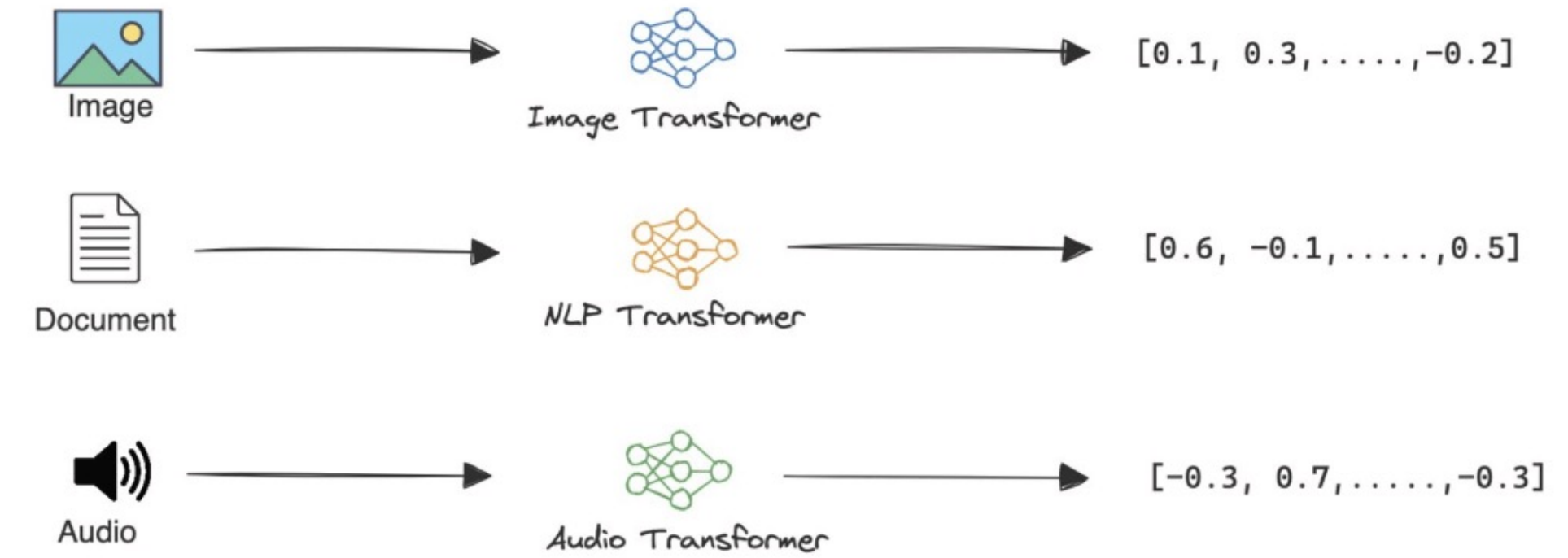
Vectors represent complex data, such as:

- Text, images, sounds, etc.,

The data is converted to a machine understandable format :

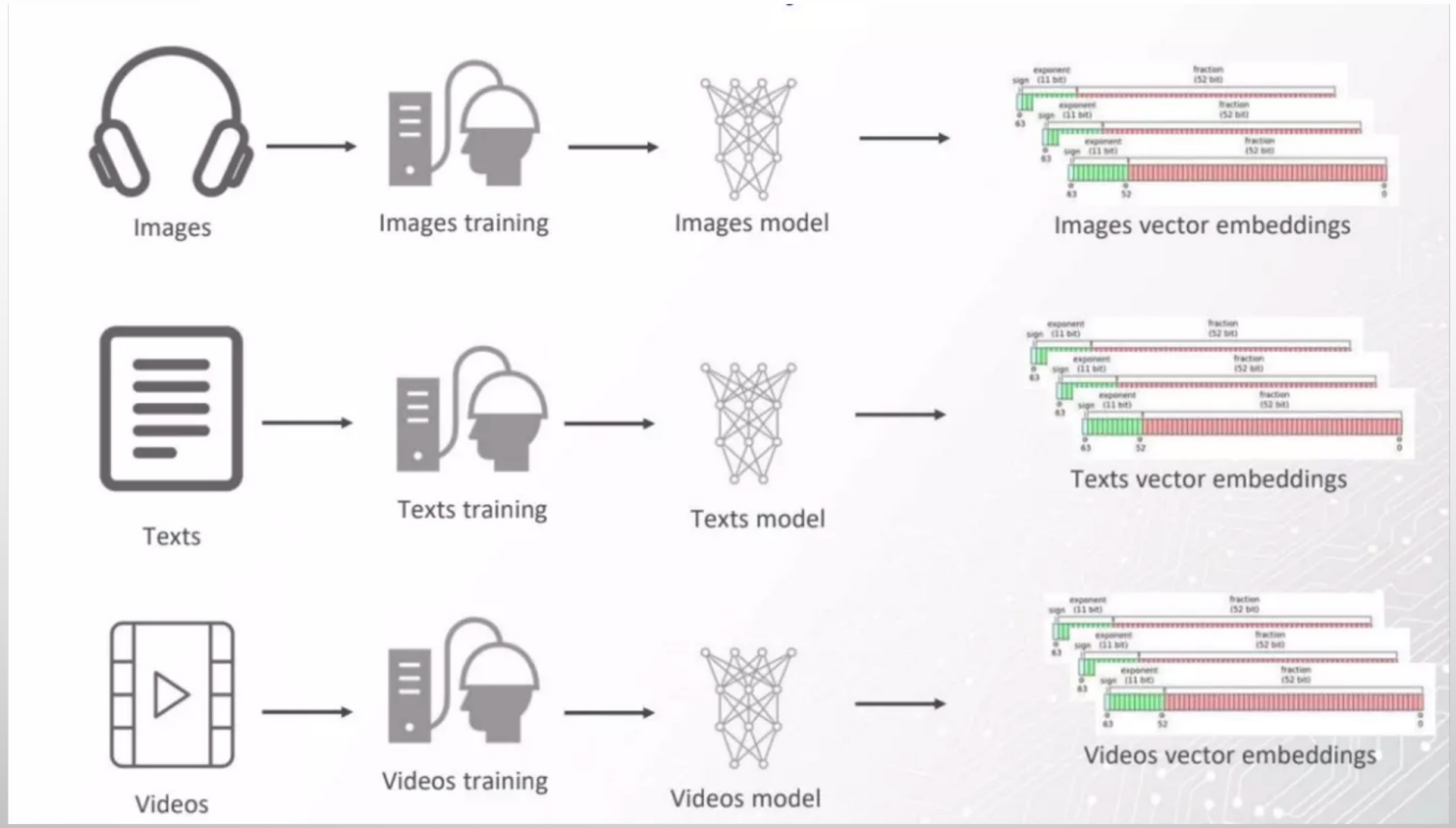
- Transforms qualitative attributes into quantitative features
- Can be easily compared using math operations

Vectors are also known as EMBEDDINGS



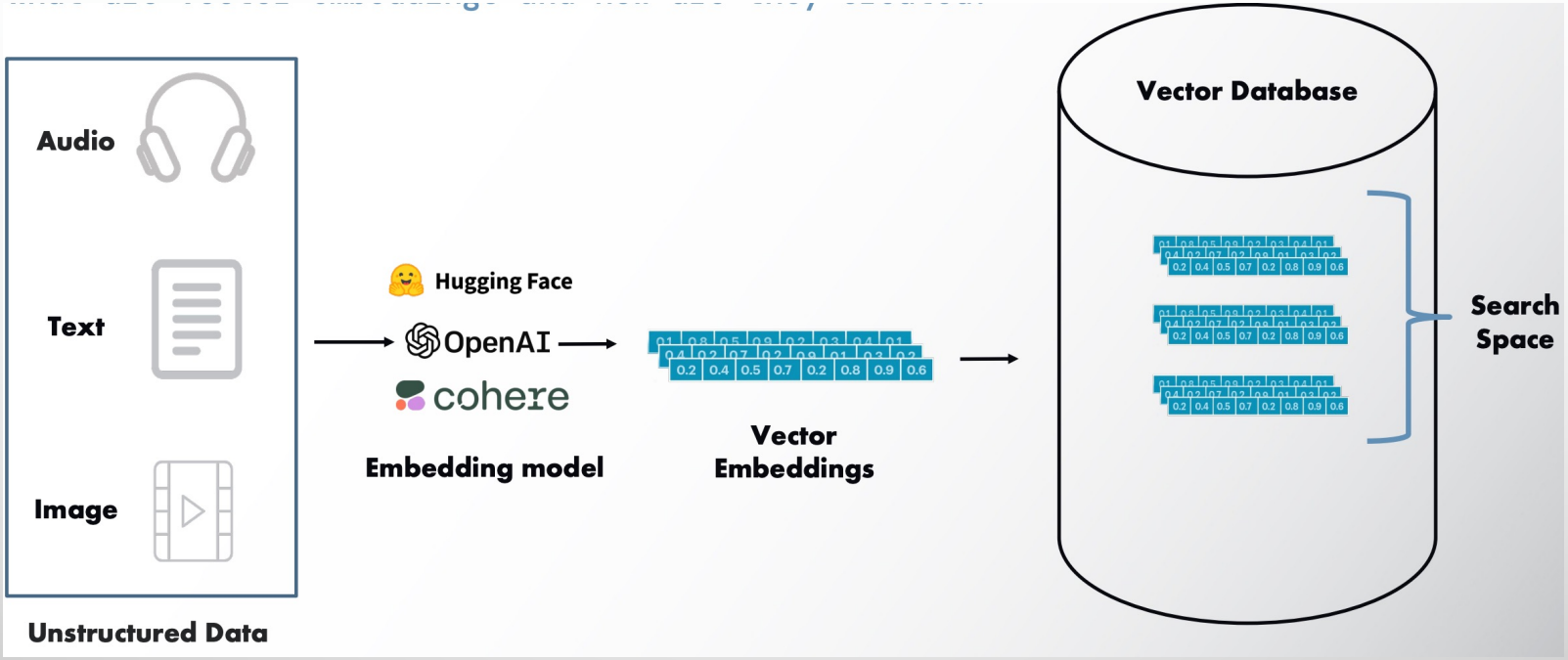
https://tge-data-web.nyc3.digitaloceanspaces.com/docs/Vector%20Databases%20-%20A%20Technical%20Primer.pdf

Generating Embeddings



https://www.slideshare.net/slideshow/vector-database/251053508

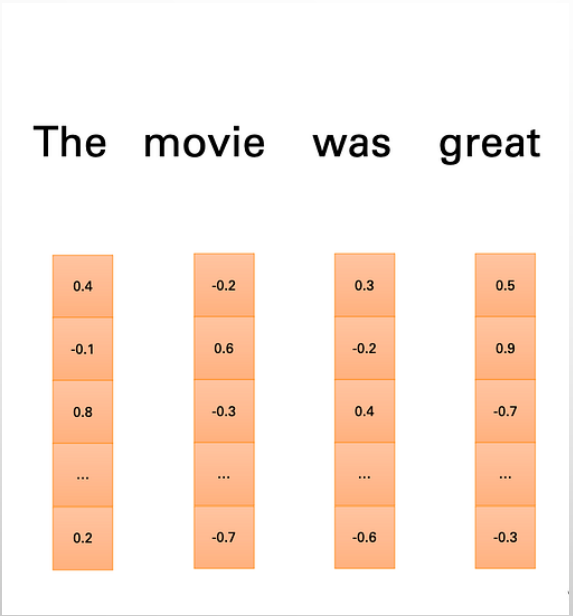
Vector Embeddings and Vector Databases



https://partee.io/download/LLM-in-Prod-talkdeck.pdf

EMBEDDINGS

- Words are converted into numerical vectors.
- These vectors capture semantic meaning.



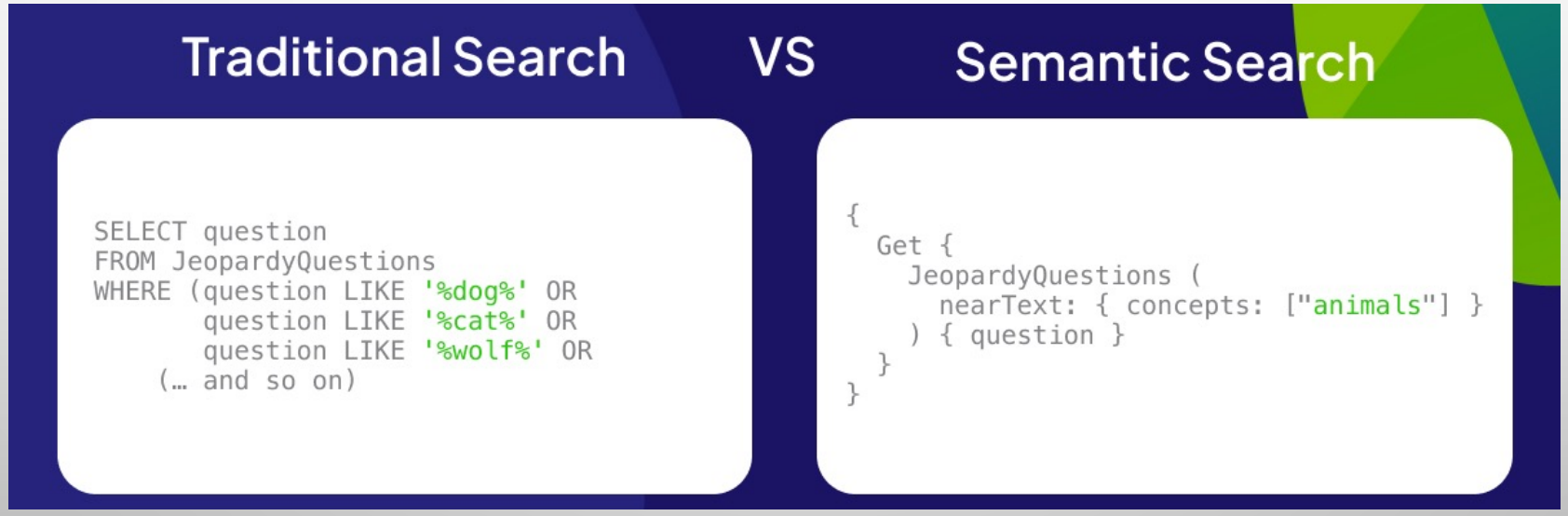
Similar concepts have a vector of numbers that are closer together

https://medium.com/data-science-at-microsoft/how-large-language-models-work-91c362f5b78f

Comparing Vector Databases to SQL databases

Indexes, stores and provides access to structured and unstructured data along with the vector embeddings (vector of digits).

It allows users to find and retrieve similar objects (concepts) quickly



https://weaviate.io/blog/what-is-a-vector-database

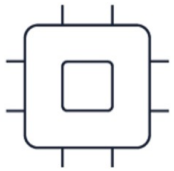
Vector Database Example Use cases

Context Retrieval

- Search for relevant sources of text from the “knowledge base”
- Provide as “context” to LLM

LLM “Memory”

- Persist embedded conversation history
- Search for relevant conversation pieces as context for LLM



LLM Cache

- Search for semantically similar LLM prompts (inputs)
- Return cached responses

https://partee.io/download/LLM-in-Prod-talkdeck.pdf

Common Use Cases

Retrieval Augmented Generation (RAG)
Expand LLMs' knowledge by incorporating external data sources into LLMs and your AI applications.

Recommender System
Match user behavior or content features with other similar ones to make effective recommendations.

Text/ Semantic Search
Search for semantically similar texts across vast amounts of natural language documents.

Image Similarity Search
Identify and search for visually similar images or objects from a vast collection of image libraries.

Video Similarity Search
Search for similar videos, scenes, or objects from extensive collections of video libraries.

Audio Similarity Search
Find similar audios in large datasets for tasks like genre classification or speech recognition

Molecular Similarity Search
Search for similar substructures, superstructures, and other structures for a specific molecule.

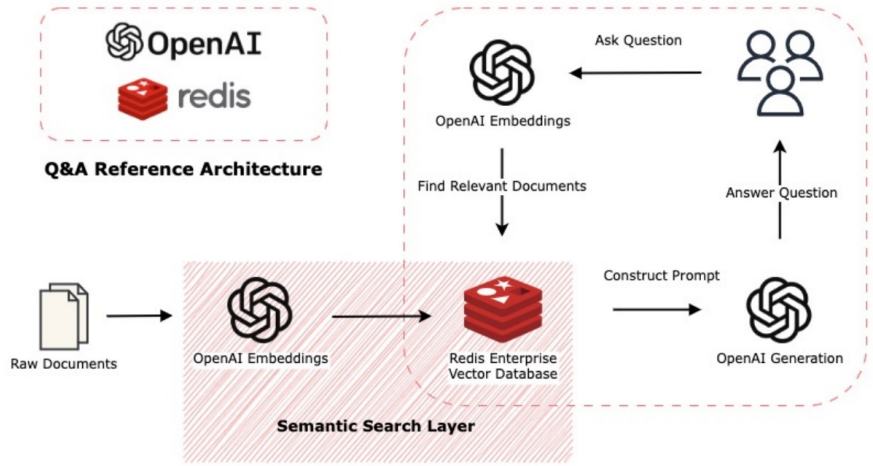
Anomaly Detection
Detect data points, events, and observations that deviate significantly from the usual pattern

Multimodal Similarity Search
Search over multiple types of data simultaneously, e.g. text and images

https://www.slideshare.net/slideshow/vector-databases-101-an-introduction-to-the-world-of-vector-databases/267310487

The RAG Q&A Example

For Question and Answering systems



- Description
  - Vector database is used as an external knowledge base for the large language model.
  - Queries are used to detect similar information (context) within the knowledge base
- Benefits
  - **Cheaper and faster** than fine-tuning
  - **Real-time updates** to knowledge base
  - **Sensitive data** doesn't need to be used in model training or fine tuning
- Use Cases
  - Document discovery and analysis
  - Chatbots

Document QnA Example: https://github.com/RedisVentures/redis-openai-qna

Chatbot Example w/ Langchain: https://github.com/RedisVentures/redis-langchain-chatbot

https://partee.io/download/LLM-in-Prod-talkdeck.pdf

DIFFERENT FORMS OF SIMILARITY SEARCH

Euclidean Distance (L2 Norm)

$$d = \sum (A_i - B_i)^2$$

- It measures the straight-line distance between two points in the embedding space.
- This method is useful when the scale of the data is important.

Inner Product (Dot Product)

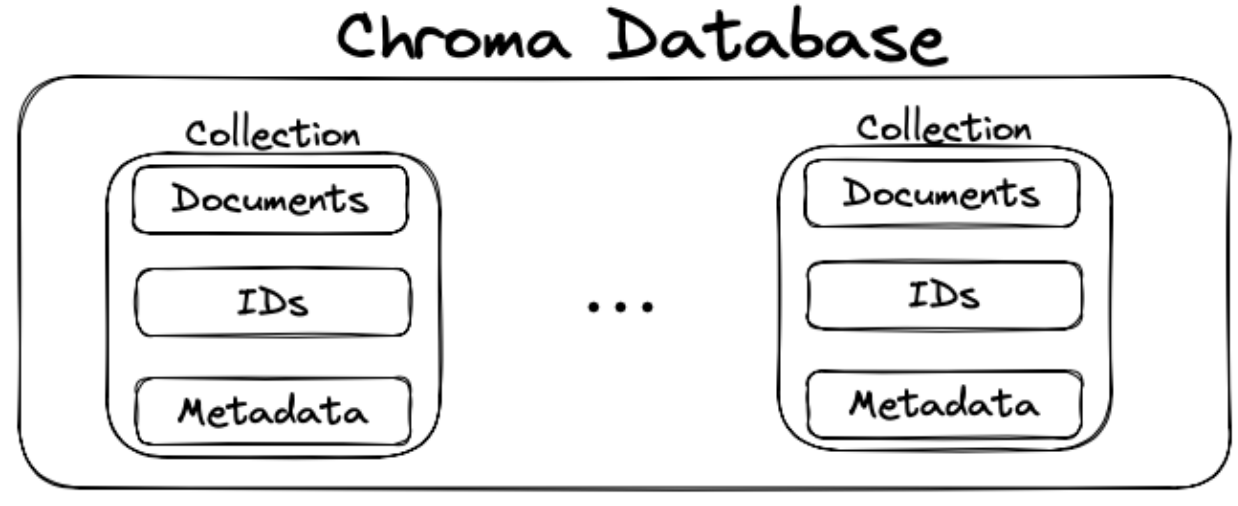
$$d = 1.0 - \sum (A_i - B_i)$$

- It measures the similarity between vectors by calculating the sum of the products of their corresponding entries.
- This method is suitable when you want to highlight the dimensions where both embeddings have high values.

Cosine Similarity

$$d = 1.0 - \frac{\sum (A_i \times B_i)}{\sqrt{\sum (A_i^2)} \cdot \sqrt{\sum (B_i^2)}}$$

- It measures the cosine of the angle between two vectors, focusing on the direction rather than the magnitude.
- This method is useful when the magnitude of the vectors should not affect the similarity measure, such as in text analysis.



- ChromaDB stores a set of documents in a collection.
- Each collection has a name:
  - Think of these as tables in a relational database.
  - Each collection has many IDs (possibly also the actual document)
  - Each ID and has a number of metadata items (attributes) associated to it. (Think of these as the columns of the table)

https://www.devtools.wtf/blog/dissect-vectordb/

OTHER COMPONENTS OF CHROMADB (BEYOND EMBEDDINGS)

Ids: key stores the indexing value for each document (key value pair).

Indexing:
ChromaDB employs indexing algorithms to store similar embeddings together.

Metadata:
Each embedding metadata helps give context to make query results more precise.

Examples:
Document title, publication source, date, etc.

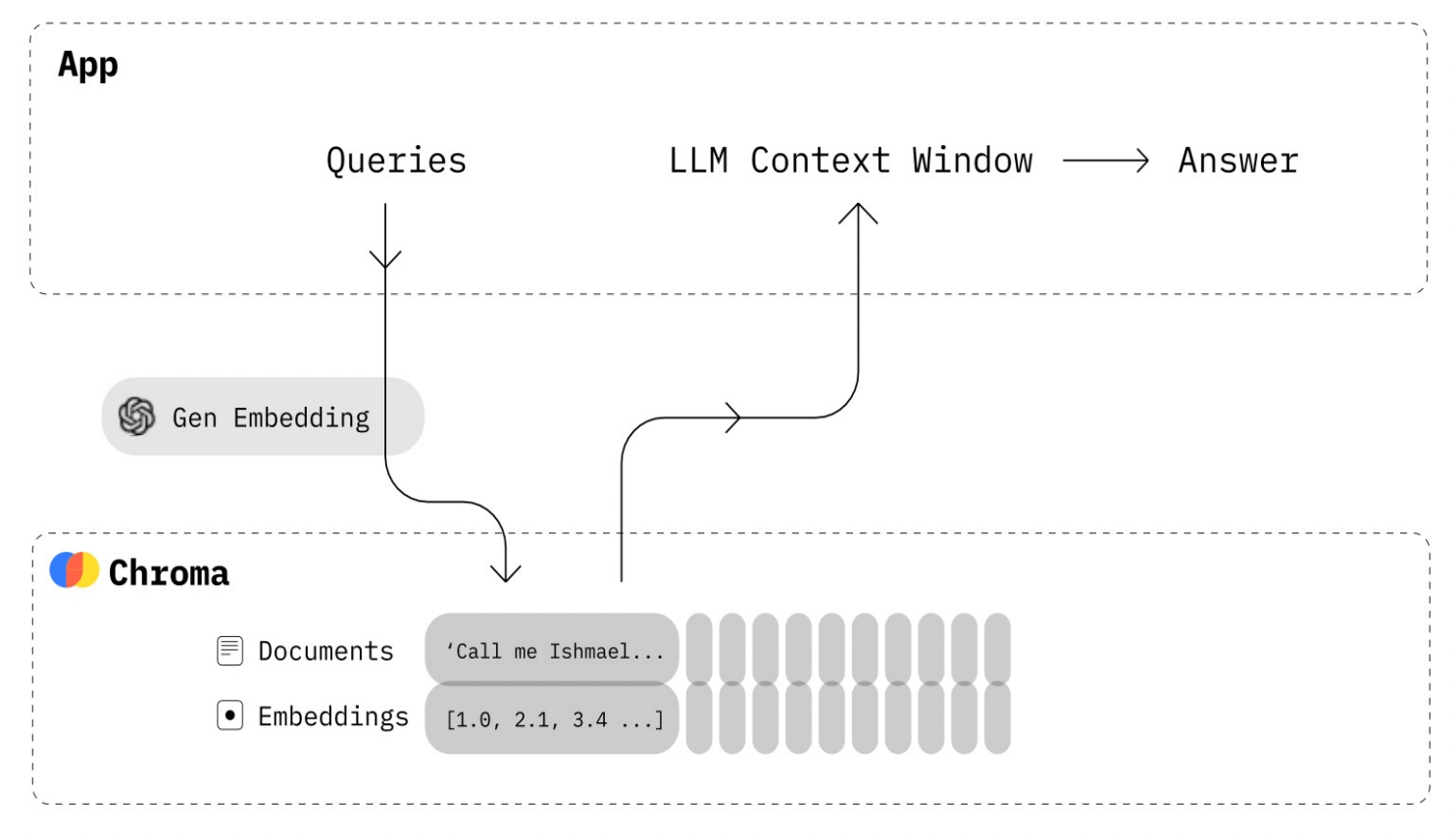
Can have multiple key and value pairs to filter and get faster query results.

CHROMADB

- ChromaDB embeddings:
  - Can use an ChromaDB embedding function to store and query data.
  - Can pass embeddings previously calculated (ex. via OpenAI embedding)

- For ChromaDB embedding:
  - Utilizes sentence transformer **all-MiniLM-L6-v2** as default function for embedding.

CHROMADB



https://docs.trychroma.com

CHROMADB (LOCAL)

- Installation

```
$ pip install chromadb
```

- Initiating a Chroma client

```
client = chromadb.PersistentClient(path="/path/to/save/to")
```

This command will configure Chroma to save and load the database from your local machine.

Data will be persisted automatically and loaded on start (if exists)

CHROMADB (SERVER/CLIENT)

- Start the Chroma server from the terminal.

```
chroma run --path /db_path
```

- Use the HttpClient method to connect to the ChromaDB localhost server.

```
import chromadb
chroma_client = chromadb.HttpClient(host='localhost', port=8000)
```

ADD DATA TO COLLECTION

ChromaDB also accepts documents where ids will be assigned automatically and tokenized based on the embedding function

```
collection.add(
    documents=["lorem ipsum...", "doc2", "doc3", ...],
    metadatas=[{"chapter": "3", "verse": "16"}, {"chapter": "3", "verse": "5"}
    ids=["id1", "id2", "id3", ...]
)
```

You can also add embedding directly to the collection

```
collection.add(
    embeddings=[[1.1, 2.3, 3.2], [4.5, 6.9, 4.4], [1.1, 2.3, 3.2], ...],
    metadatas=[{"chapter": "3", "verse": "16"}, {"chapter": "3", "verse": "5"}
    ids=["id1", "id2", "id3", ...]
)
```

A SIMPLE EXAMPLE

```python
import chromadb
chroma_client = chromadb.Client()

# switch `create_collection` to `get_or_create_collection` to avoid creating a new co
collection = chroma_client.get_or_create_collection(name="my_collection")

# switch `add` to `upsert` to avoid adding the same documents every time
collection.upsert(
    documents=[
        "This is a document about pineapple",
        "This is a document about oranges"
    ],
    ids=["id1", "id2"]
)

results = collection.query(
    query_texts=["This is a query document about florida"], # Chroma will embed this
    n_results=2 # how many results to return
)

print(results)
```

Note:
This will recreate embeddings every time this is run!!!

Create once, query a lot…

QUERY CHROMADB

- User's can query via query_texts. Chroma will first embed each query_text with the collection's embedding function, and then perform the query with the generated embedding.

```
collection.query(
    query_texts=["doc10", "thus spake zarathustra", ...],
    n_results=10,
    where={"metadata_field": "is_equal_to_this"},
    where_document={"$contains":"search_string"}
)
```

- This query shows that the user can query by metadata and document content.
- The output can be stored in a variable and similar text can be fetched using the command:

```
query_results["documents"]
```

CHROMADB QUERY RESULT

- Query the ChromaDB with a textual query → get the top 3 similar documents

```
result = collection.query(query_texts="Microsoft CEO", include=['documents', 'distances', 'metadatas'], n_results=3)
```

- The results contain three list:
  - The documents (from the chromaDB) that are similar to the query

```
result['documents'][0]
```

  - The special distance between the document and query

```
result['distances'][0]
```

```
[0.974673330783844, 1.0633965730667114, 1.084348440170288]
```

  - The metadata of each document

```
result['metadatas'][0]
```

```
[{'label': 'Sci/Tech'}, {'label': 'Business'}, {'label': 'Sci/Tech'}]
```

ADDING DATA TO CHROMADB USING OPENAI

- DATAFRAME CONTAINS NEWS AND NEWS TITLE

	title	description	label_int	label
0	World Briefings	BRITAIN: BLAIR WARNS OF CLIMATE THREAT Prime M...	1	World
1	Nvidia Puts a Firewall on a Motherboard (PC Wo...	PC World - Upcoming chip set will include buil...	4	Sci/Tech

- ADDING THE DATA TO THE CHROMADB

```
batch_size = 100

for i in range(0, len(df), batch_size):
    batch_df = df[i:i+batch_size]
    collection.add(
        ids=[str(j+i) for j in range(len(batch_df))],
        documents=(batch_df['title'] + '. ' + batch_df['description'].apply(lambda x: ''.join(x))).to_list(),
        metadatas=[{"label": label} for label in batch_df['label'].to_list()]
    )
```

CREATE A COLLECTION USING OPENAI EMBEDDING

- Use OpenAI to calculate the embedding for each document

- Create the embeddings and then give those to ChromaDB
  (or you can give ChromaDB the OpenAI embedding function)

USING CHROMADB WITH OPENAI

```python
# Create an OpenAI client.
if 'openai_client' not in st.session_state:
    api_key = st.secrets["OPENAI_API_KEY"]
    st.session_state.openai_client = OpenAI(api_key=api_key)


def add_to_collection(collection, text, filename):

    # Create an embedding
    openai_client = st.session_state.openai_client
    response = openai_client.embeddings.create(
        input=text,
        model="text-embedding-3-small")

    # Get the embedding
    embedding = response.data[0].embedding

    # Add embedding and document to ChromaDB
    collection.add(
        documents=[text],
        ids = [filename],
        embeddings=[embedding]
    )
```

USING CHROMADB WITH OPENAI

```
topic = st.sidebar.selectbox("Topic",
                ( "Text Mining", "GenAI"))

openai_client = st.session_state.openai_client
response = openai_client.embeddings.create(
        input=topic,
        model="text-embedding-3-small")

# Get the embedding
query_embedding = response.data[0].embedding

#get the text relating to this question (this prompt)
results = collection.query(
        query_embeddings=[query_embedding],
        n_results=3  # Number of closest documents to return
)

# Print the results with IDs using an index
for i in range(len(results['documents'][0])):
        doc = results['documents'][0][i]
        doc_id = results['ids'][0][i]
        st.write(f"The following file/syllabus might be helpful: {doc_id} ")
```

USING CHROMADB WITH STREAMLIT CLOUD

```python
import streamlit as st
from openai import OpenAI
import os
from PyPDF2 import PdfReader

#fix for working with  ChromaDB and streamlit
__import__('pysqlite3')
import sys
sys.modules['sqlite3'] = sys.modules.pop('pysqlite3')

import chromadb
```

requirements.txt

```
streamlit
openai
chromadb
pysqlite3-binary
PyPDF2
protobuf==3.20
```

LAB 4 – PART A

- Start with HW3 or Lab3 - Add it to the main Streamlit Lab app as a new "Lab4" page
- Create a function (method) - This function should:
  - Construct a ChromaDB collection named "Lab4Collection"
  - For the embedding vector, use OpenAI - such as the "text-embedding-3-small" model
  - Use the 7 PDF files provided - read the PDF files and convert into text.
  - Have a key for storing / getting the documents (ex. filename) -- Use "metadatas" as needed.
- Store the vector database collection in st.session_state.Lab4_vectorDB
  - Create the chromaDB once (per application run) - This will help in reducing the embedding cost
  - Only call this function if 'Lab4_vectorDB' is not in st.session_state
- Test the vectorDB - Do not yet integrate into a conversation"
  - Use a search string as a test ("Generative AI", "Text Mining", "Data Science Overview") and output (ex. st.write) an ordered list of 3 returned documents (names of files)