Vector Databases for RAG: An Introduction

Importance of vector databases

Use vector databases for analysis tasks that:

Group items

Classify items

Suggest relationships among items

Handle complex data types

Vector databases support complex data, including:

Social likes

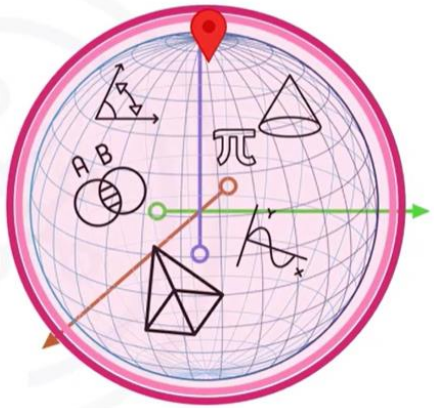
Geospatial data

Genomic data

Handle complex data types

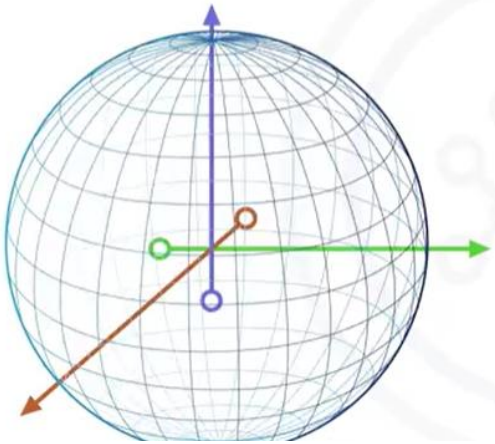
Vector databases enable:

- Easy storage
- Fast data retrieval
- Complex data analysis



Perform similarity searches

Data scientists can locate database items in proximity to each other



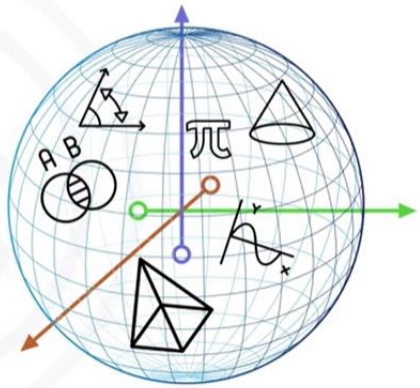
High performance: Speed and scalability

- Use the following techniques:
  - Distributed computing
  - Indexing
  - Parallel processing
- Quickly manage:
  - Big data sets
  - Process lots of queries quickly

Characteristics of vector databases

A vector:

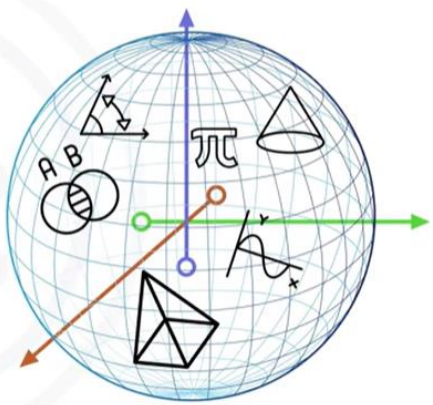
- Stores data as mathematical objects defined by their size and direction
- Is a mathematical way to show data points in a place with more than one dimension



Characteristics of vector databases

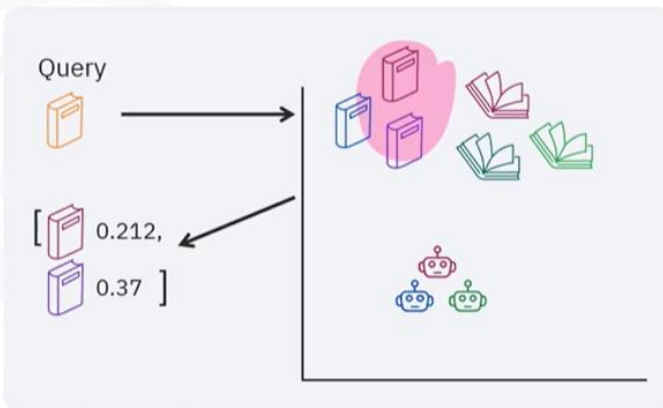
- Can represent many data types, including:
  - Images
  - Sounds
  - Text files
  - Pattern data
  - Mapping data
  - Genomic information

  ... and more



What is a vector?

- A vector is an array of numerical values or data points
- An array of numerical values or attributes that relate to different features
- Each numerical data point is a dimension



Vector data examples

Books as vectors:


A fiction book: [1, 350, 2003, 4.5]


A non-fiction book: [2, 250, 2015, 4.8]


A science fiction book: [3, 400, 1990, 4.2]

Vector data examples

- The first number represents the genre.
- The second number represents the number of pages in the book
- The third number represents the book's publication year
- The fourth number represents the book's average rating.

[1, 350, 2003, 4.5]
[2, 250, 2015, 4.8]
[3, 400, 1990, 4.2]

Vector data examples

Interested in science fiction books:
With approximately 200 pages
With ratings between 4.7 and 5.0

Vector data examples

Available science fiction books:


Book 1 [1, 180, 2010, 4.6]


Book 2 [1, 220, 2005, 4.8]


Book 3 [1, 210, 2015, 4.9]


Book 4 [1, 190, 2018, 4.7]


Book 5 [1, 200, 2012, 4.5]

Recap

In this video, you learned that:

- Vector databases simplify data storage, organization, and retrieval of complex data types, including images, likes, sounds, text files, pattern data, map data, genomic information, and others
- You can use vector databases for analysis tasks that group items, classify items, and suggest relationships among items
- Vector databases, an integral part of machine learning and for data in diverse domains, offer high performance and scalability
- Vector databases store data as mathematical objects defined by size and direction
- Vector databases use distributed computing, indexing, and parallel processing techniques to quickly manage big data sets and process queries
- A vector is an array of numerical values relating to different features or data attributes

Vector and relational databases compared

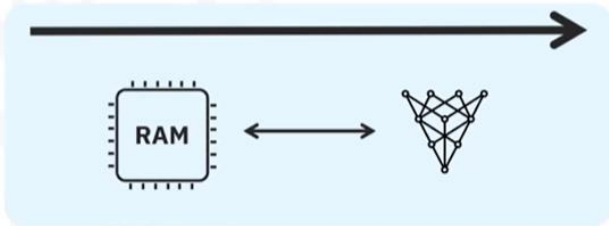
Check out this comparative table to understand the distinctions between vector and relational databases, which is crucial for selecting the appropriate database type for specific needs.

Function	Traditional databases	Vector databases
Data Representation	Traditional databases organize data in a structured format using tables, rows, and columns, ideal for relational data.	Vector databases represent data as multi-dimensional vectors, efficiently encoding complex and unstructured data like images, text, and sensor data.
Data Search and Retrieval	SQL queries are suited for traditional databases with structured data.	Vector databases specialize in similarity searches and retrieving vectorized data, facilitating tasks like image retrieval, recommendation systems, and anomaly detection.
Indexing	Traditional databases employ indexing methods like B-trees for efficient data retrieval.	Vector databases use indexing structures like metric trees and hashing suited for high-dimensional spaces, enhancing nearest-neighbor searches and similarity assessments.
Scalability	Scaling traditional databases can be challenging, often requiring resource augmentation or data sharding.	Vector databases are designed for scalability, especially in handling large datasets and similarity searches, using distributed architectures for horizontal scaling.
Applications	Traditional databases are pivotal in business applications and transactional systems where structured data is processed.	Vector databases shine in analyzing vast datasets, supporting fields like scientific research, natural language processing, and multimedia analysis.

- A vector database is a specialized database representing data numerically as vectors in a multi-dimensional space.
- While traditional databases are adept at managing structured data and transactional operations, vector databases handle high-dimensional data and perform rapid similarity searches.
- Vector libraries read and update data; however, vector databases can perform create, read, update, and delete (CRUD) functions.
- Vector databases store data formatted so that numerical vectors depict each data item.
- Relational databases organize data into tables, utilizing rows and columns.
- Traditional databases use SQL queries, and vector databases use similarity search to retrieve data.

In-memory vector databases

- Store vectors directly in memory
- Enable swift read-and-write operations
- Support real-time analytics and recommendation systems
- Include database vendors such as:
  - RedisAI
  - Torchserve

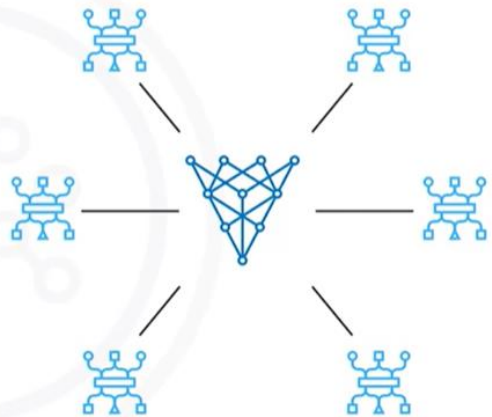


Disk-based vector databases

- Store vectors on disk
- Suitable for large data sets
- Use indexing and compression techniques
- Include vendors such as:
  - Annoy
  - Milvus
  - ScaNN

Distributed vector databases

- 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

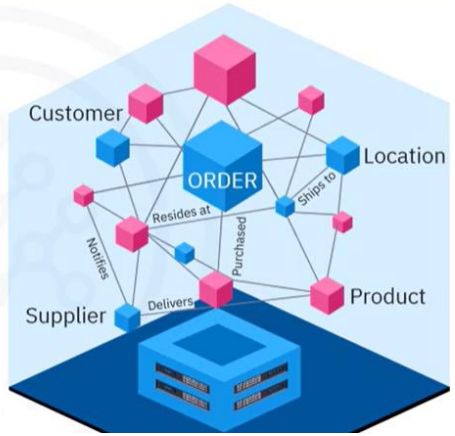


Distributed database vendors

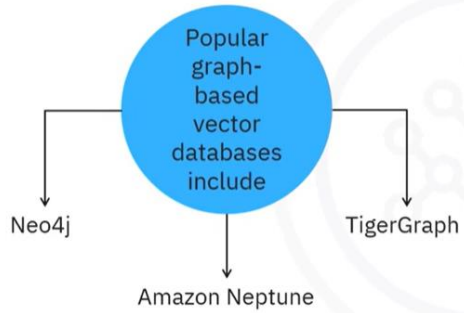
1. FAISS
2. Elasticsearch with Vector Plugin
3. Dask-ML

Graph-based vector databases

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

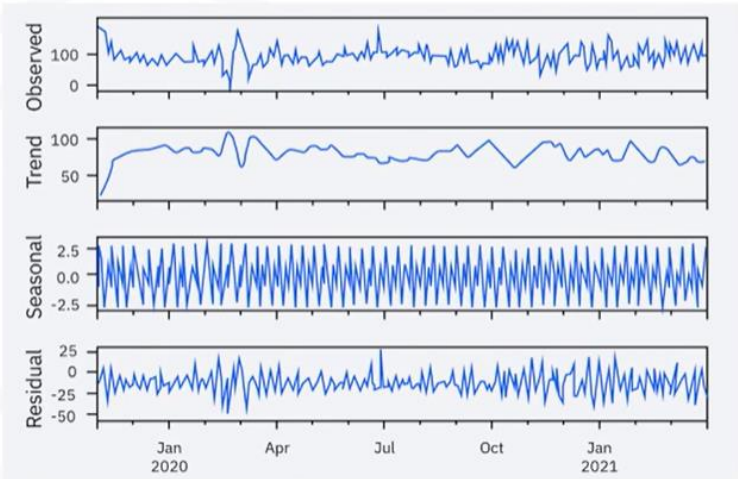


Graph-based vector database vendors

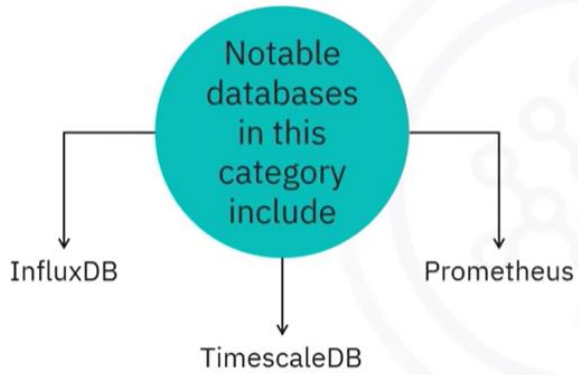


Time-series vector databases

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



Time-series vector databases



Dedicated vector databases

Use special characteristics to store, index, query, and analyze vector data

Provide efficiency for similarity search, clustering, and classification tasks

Dedicated vector database characteristics

Use unique data structures

- Reversed indexes
- Product quantization
- Locality-sensitive hashing (LSH)

Support vector operations

- Nearest neighbor search
- Similarity search
- Distance calculations

Dedicated vector database characteristics

Provide scalability

- Store and query big vector data sets quickly across clusters or distributed system

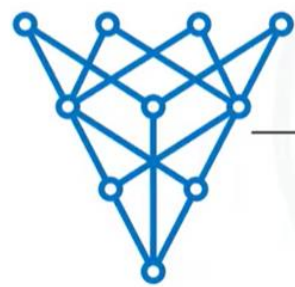
Deliver speed

- Use optimized algorithms and data structures to get quick answers

Provide customization

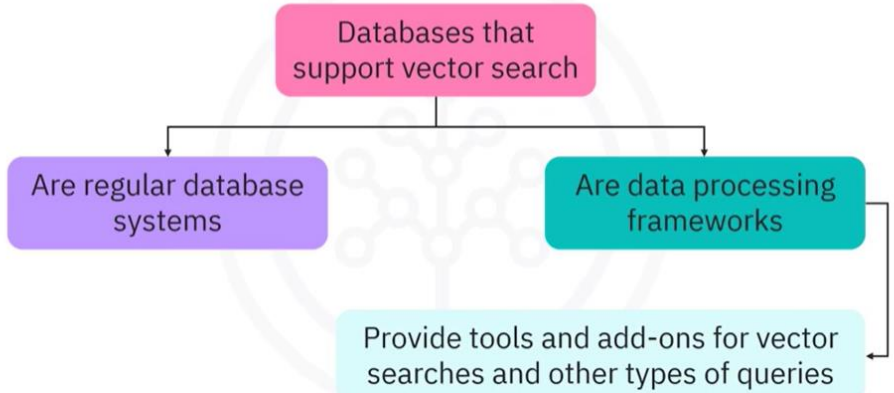
- Change database parameters for indexing and searching

Dedicated vector database examples



- Faiss (Facebook AI Similarity Search)
- Annoy (Approximate Nearest Neighbors Oh Yeah)
- Milvus

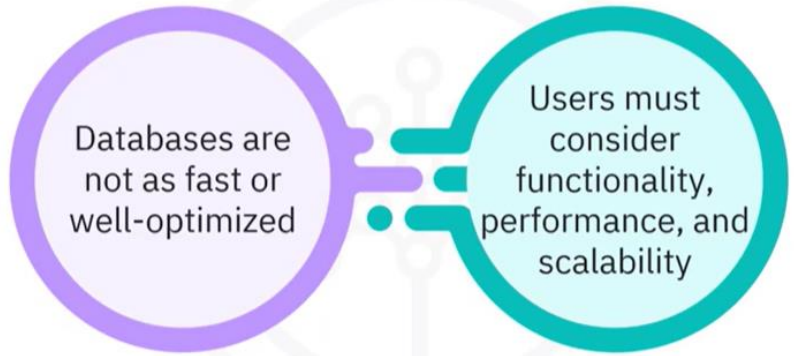
Databases that support vector search



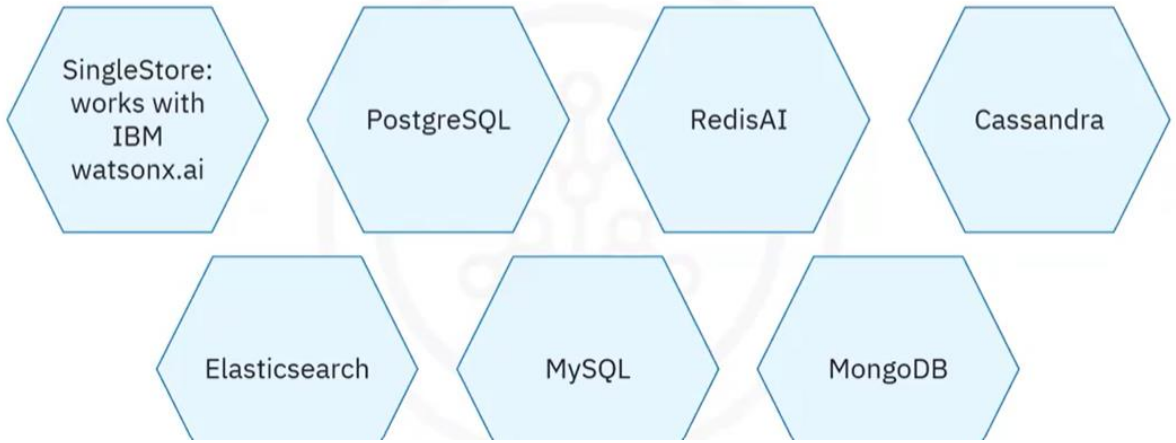
Databases that support vector search characteristics

Databases that support vector search characteristics

Vector search performance considerations



Vector search database vendors



Recap

In this video, you learned that:

- Vector-based database types include in-memory, disk-based, distributed, graph-based, and time-based databases
- Vector-based databases speed performance for recommendation systems, social network analysis, knowledge graphs (graph analysis), and other complex tasks
- Dedicated vector databases use unique data structures, reversed indexes, product quantization, and locality-sensitive hashing (LSH) that provide scalability, deliver speed, and enhance customization
- Popular database vector database vendors include FAISS, Annoy, and Milvus

Recap

In this video, you learned that:

- Databases that support vector search are either regular database systems or data processing frameworks that let you store data as BLOBs, arrays, or user-defined types
- Notable vendors that support vector search include SingleStore with its support for IBM watsonx.ai, Elasticsearch, PostgreSQL, MySQL, RedisAI, Apache MongoDB, and Apache Cassandra

Recommendation systems

Task	Vector database capability	Uses
Embedding storage and nearest neighbor search	Incorporate embeddings or numerical representations of items or entities generated by a recommendation system	Access the vector's likes and traits Locate the vector's closest neighbors for improved personalized suggestions
Deliver performance improvement and scalability	Provide scalability to handle additional searches and vectors Improve query processing and indexing structure	Deliver fast, scalable recommendation services for large numbers of concurrent users
Provide cross-domain suggestions	Stores embeddings and carry out cross-domain suggestions	Enhance the completeness of recommendation systems

Geospatial analysis and location-based services

Task	Vector database capabilities	Uses
Efficiently store and index data	• Use indexing methods like R-tree or quadtree • Store geospatial data like addresses, polygons, GPS locations	Deliver spatial queries, like closeness searches, range queries, and spatial joins, for GPS information and other mapping needs
Provide location-based suggestions	• Combine geospatial data with user preferences and location	Deliver recommendations for nearby events, services, and places of interest
Deliver real-time geospatial analytics	• Process streaming data in real-time • Groups items together spatially • Recognizes spatial patterns	Power apps like tracking vehicles, managing fleets, dynamic routing, finding hotspots

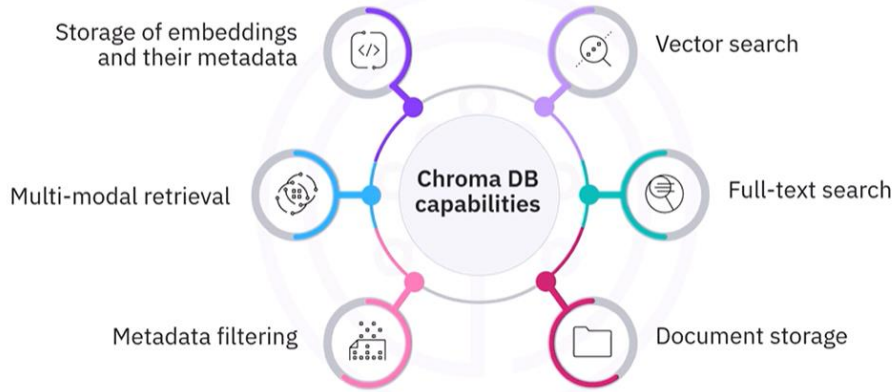
Marketing and social media insights

Task	Vector database capability	Uses
Provide distributed storage and parallel processing for horizontal scalability	Spread data and queries across multiple nodes or groups	Process big data and handle simultaneous queries such as SEO calculations
Reduce latency and boost overall speed	Use optimized caching and query execution plans	Obtain trending analytics faster
Adjust to changing task needs	Support auto-scaling and dynamic resource allocation	Your company can scale hardware and cloud resource usage for the best performance and lower costs

Image and video analysis

Task	Vector database capability	Uses
Perform feature extraction and representation	Store high-dimensional feature vectors	Displays aspects of images, such as color histograms, texture descriptions, or deep learning embeddings
Similarity searches	Store feature vectors	Locate images, summarize videos, and suggest images and videos based on content
Process real-time data	Provide horizontal scalability for real-time data storage	Perform video surveillance, object recognition, and live event analysis

Introduction to Chroma DB



Deploying Chroma DB

Client-server architecture mode

- Client connects to server running in separate process
- Server launched by CLI or in a Docker image
- Clients connect via HTTP

Two ways to deploy Chroma DB

Standalone mode

- For Python
- Both server and client are running in a single process
- Useful for quick testing or single-machine scenarios

Chroma DB architecture

Phases of Chroma DB architecture

- Optional: Obtaining embeddings
- Creating collections
- Storing data
- Performing collection operations
- Querying and grouping data

Chroma DB ecosystem clients and integrations

Chroma DB supports multiple clients, languages, and integrations:

- Official clients:
  - Python, JavaScript
  - Maintained by Chroma Core team
- Community supported clients:
  - Ruby, Java, Go, C#, Rust, PHP

For more information: https://cookbook.chromadb.dev/ecosystem/clients

Chroma DB ecosystem clients and integrations

- Integration with popular frameworks:
  - LangChain, LlamaIndex, Ollama
- Native integration with embedding models:
  - Hugging Face, Google, OpenAI

Practical example walkthrough – creating and querying collections

Step 1: Create a collection

- Give it a logical name

Step 2: Add text chunks and chunk metadata

- Chroma automatically stores text and handles embeddings

Practical example walkthrough – creating and querying collections

Step 3: Query the collection

- Chroma returns most similar results
- Chroma handles embedding of query text
- Uses Euclidean distance to identify similarity by default
- Supports cosine distance and dot product calculations

Performance

Efficient similarity search:

- Optimized for approximate nearest neighbor searches
- Allows users to quickly find similar vectors
- Chroma DB uses an advanced algorithm:
  - Hierarchical Navigable Small World (HNSW)

Performance

Good coding practices:

- Core written in Rust
- 3 to 5 times speed improvement in querying and writing operations
  - Compared to a core in Python

Use cases and applications

Recommender systems

- Based on user preference

Image retrieval

- Based on text queries using multi-modal retrieval

Document search

- Efficient search engines using vector or full text

AI-based chatbots

- Built with semantic search and retrieval capabilities for context augmentation

Recap

- Chroma DB is a powerful vector database designed for various retrieval tasks
- Chroma DB supports multi-modal retrieval, metadata filtering, vector search and full-text search
- Deploying Chroma DB can be done using client-server architecture or standalone mode
- Chroma DB integrates with popular frameworks and supports multiple programming languages

Recap

- Chroma DB uses an advanced algorithm for approximate nearest neighbor search, which efficiently finds the approximately closest chunks to a query within a collection
- Chroma DB is suitable for a wide range of applications, including recommender systems, document search, image retrieval, and AI-based chatbots

Creating collections

Collections organize data

```
import chromadb
from chromadb.utils import embedding_functions

ef = embedding_functions.SentenceTransformerEmbeddingFunction(
    model_name="all-MiniLM-L6-v2"
)
```

Creating collections

Define the Chroma DB client

```
client = chromadb.Client()
collection = client.create_collection(
    name="my_collection",
    metadata={"description": "A collection for storing user data"},
    configuration={
        "embedding_function": ef
    }
)
print(f"Collection created: {collection.name}")
```

Connecting to existing collections

```
collection2 = client.get_collection(name="my_collection")
print(f"collection2 metadata: {collection2.metadata}")
```

Output:

```
collection2 metadata: {'description': 'A collection for storing
user data'}
```

Modifying a collection

```
collection.modify(name="test2", metadata={"key": "value"})
print(f"Collection metadata: {collection.metadata}")

Output:
Collection metadata: {'key': 'value'}
```

- Some changes cannot be made within an existing collection:
  - Embedding model
  - Distance metric

- In these cases, you need to clone the collection

Adding documents to collections

```
collection.add(
    documents=[
        "This is a document about LangChain",
        "This is a document about LlamaIndex"
    ],
    metadatas=[
        {"source":"langchain.com", "version":"0.2"},
        {"source":"llamaindex.ai", "version":"0.12"}
    ],
    ids=["id1", "id2"]
)
```

Getting documents from a collection

```
collection.get()
```

Output:

```
{'ids': ['id1', 'id2'],
 'embeddings': None,
 'documents': ['This is a document about LangChain',
  'This is a document about LlamaIndex'],
 'uris': None,
 'included': ['metadatas', 'documents'],
 'data': None,
 'metadatas': [{'version': '0.2', 'source': 'langchain.com'},
  {'source': 'llamaindex.ai', 'version': '0.12'}]}
```

Getting documents from a collection

Get embeddings information from a collection

```
collection.get(include=['embeddings'])
```

Get individual documents from a collection

```
collection.get('id1')
```

Updating data in collections

```
collection.update(
    ids=["id1"],
    metadatas=[{"source":"langchain.com", "version":"0.3"}],
    documents=["This an updated document about LangChain"],
)
```

Deleting data from collections

```
collection.delete(
    ids=["id1", "id2"],
    where={"source": "llamaindex.ai"}
)
```

Using the HNSW parameter for collections

```
collection = client.create_collection(
    name="my_collection",
    metadata={"description": "A collection for storing user data"},
    configuration={
        "hnsw": {"space": "cosine"},
        "embedding_function": ef
    }
)
```

Using the HNSW space parameter for collections

- HNSW performs approximate nearest neighbor searches
- Use the **space** parameter to define distance function
- Default = **l2** (squared L2 norm)
- Other options:
  - **cosine** (cosine distance)
  - **ip** (inner product or dot product distance)

Recap

- Chroma DB operations form the foundation for building intelligent, vector-based applications
- Collections are a way to organize your data within Chroma DB
- You define an embedding model using Chroma DB's **embedding_functions**
- Create a collection using the **create_collection** method
- Metadata helps you keep track of the purpose and contents of your collections
- Connect to an existing collection by using the **get_collection** method
- Alter an existing collection by using the **modify** method
- Use the **add** method to insert documents into a collection
- Get data from a collection using the **get** method
- Update existing data in a collection by using the **update** method
- Delete data from a collection using the **delete** method
- Use the **hnsw space** configuration parameter to specify the distance function when performing approximate nearest neighbor searches

What is RAG?

RAG is a framework that enhances language models

- Retrieves relevant information from external sources
- Generates accurate, grounded responses
- Reduces hallucinations

The problems RAG solves

LLMs have limited context windows

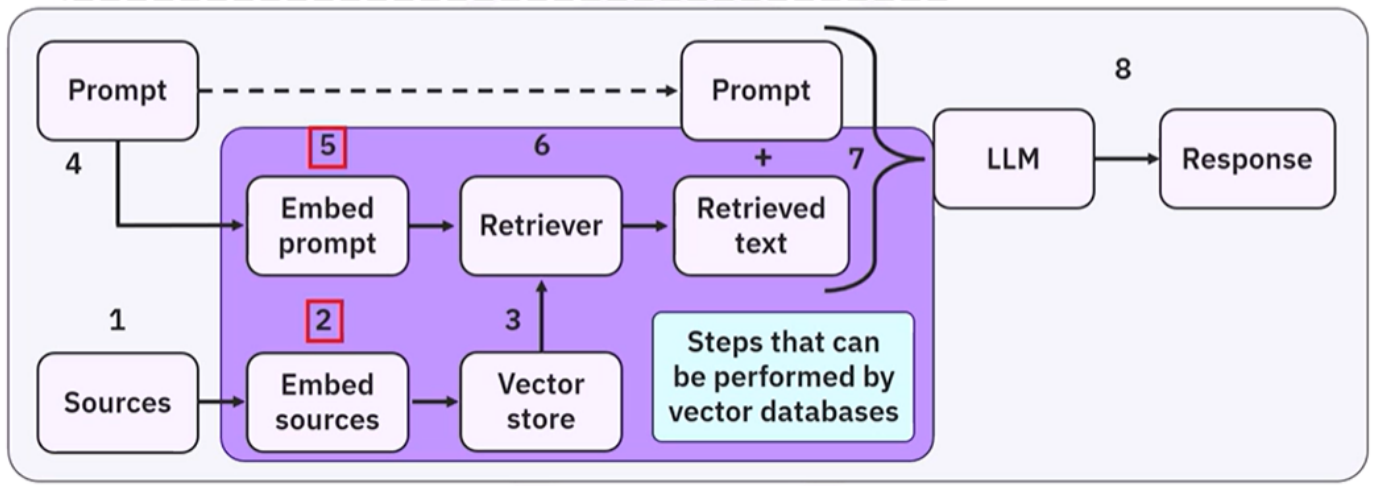
LLMs can hallucinate facts

LLM knowledge is frozen at training time

RAG addresses issues by retrieving and injecting relevant knowledge into the prompt

Where vector databases fit in

Vector databases play a central role in the RAG pipeline



Why you should use vector databases for most RAG steps

Reduces the risk of critical errors, such as:

- Using different embedding models for source documents and user prompts
- Incorrectly linking embeddings to their source documents

Optimizes performance:

- Vector databases are built for fast semantic similarity searches
- Custom-built alternatives are often slower without significant optimization

Simplifies and speeds up development:

- Less custom code to maintain
- Faster implementation and debugging

Pitfalls to avoid and how to do it right

Pitfalls

- Using different embedding models for source documents and prompts
- Chunks that are either too long or too short
- Not re-embedding when the data, metrics, or embedding model changes
- Assuming retrieval is always accurate

Best practices

- Use the same embedding model for everything
- Choose a chunk size that is just long enough to keep the meaning clear
- Re-embed when your data, metrics, or model changes
- Test your results and adjust if needed

Why RAG frameworks add value

Vector key databases handle key parts of RAG

But they don't cover everything:

- Chunking
- Advanced retrieval logic
- Prompt augmentation
- LLM integration

RAG frameworks fill the gaps

- LangChain
- LlamaIndex

Simplify the development and deployment of RAG applications

Recap

- RAG enhances LLM responses by providing relevant information from external sources
- Vector databases are the foundation that makes RAG work
- Vector databases can handle embedding source documents and user prompts, storing embeddings, retrieving most relevant matches, and providing the retrieved content for prompt augmentation
- Using a vector database helps prevent critical mistakes, speeds up development, and optimizes performance
- Some RAG pipeline tasks, such as chunking, advanced retrieval logic, prompt augmentation, and LLM integration, usually happen outside the database
- RAG frameworks, such as LangChain and LlamaIndex can wrap around your vector database and help manage the full RAG pipeline