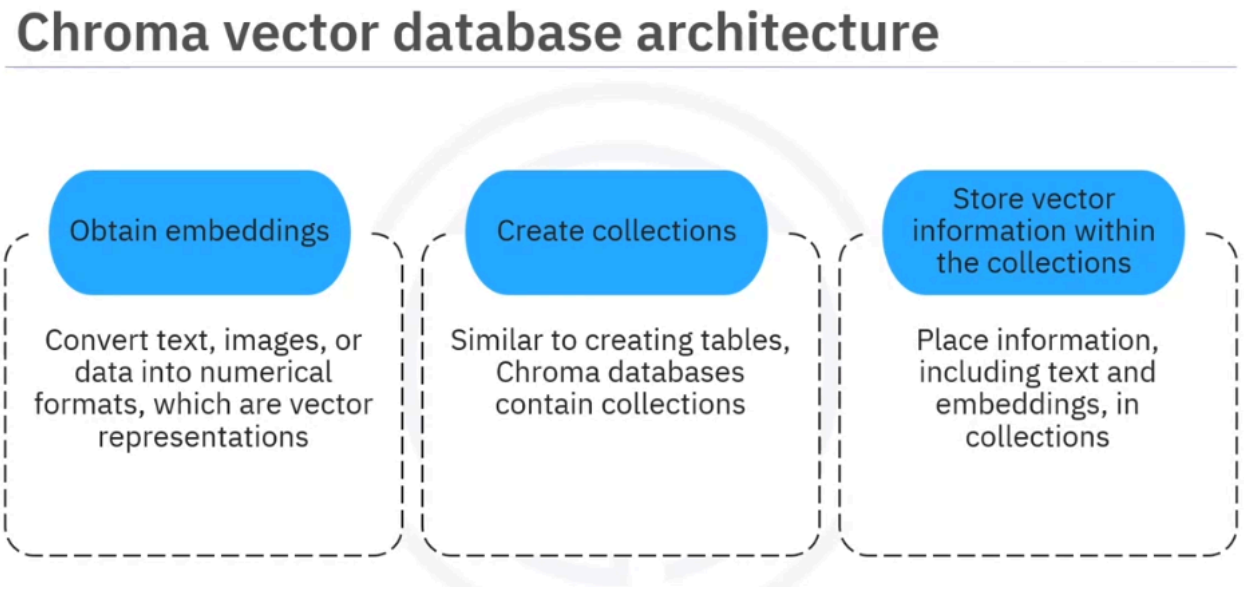
What is Chroma DB?

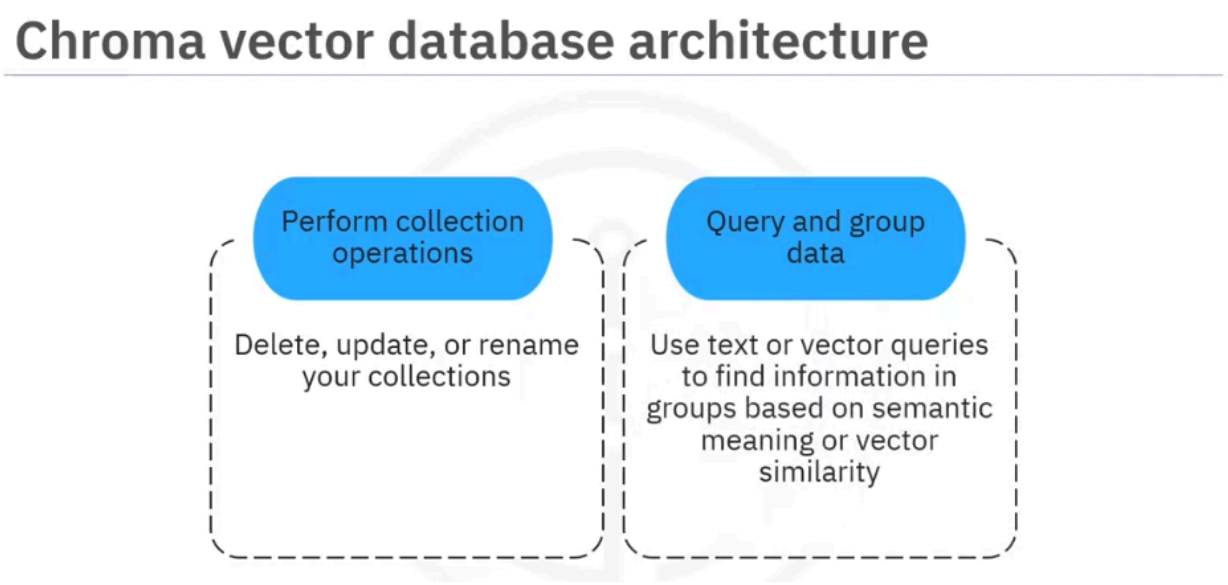
Chroma DB- Introduction. Chroma DB is an open-source vector... | by Nidhiworah | Medium

Chroma DB is an open-source vector store used for storing and retrieving vector embeddings. Its main use is to save embeddings along with metadata to be used later by large language models. Additionally, it can also be used for semantic search engines over text data.

Chroma DB features

- **Simple and powerful:**
  - Install with a simple command: `pip install chromadb`.
  - Quick start with Python SDK, allowing for seamless integration and fast setup.
- **Full-featured:**
  - **Comprehensive retrieval features**: Includes vector search, full-text search, document storage, metadata filtering, and multi-modal retrieval.
  - **Highly scalable**: Supports different storage backends like [DuckDB](DuckDB - Wikipedia) for local use or ClickHouse for scaling larger applications.
- **Multi-language support:**
  - Offers SDKs for popular programming languages, including **Python**, **JavaScript/TypeScript**, **Ruby**, **PHP**, and **Java**.
- **Integrated:**
  - Native integration with embedding models from HuggingFace, OpenAI, Google, and more.
  - Compatible with Langchain and LlamaIndex, with more tool integrations coming soon.
- **Open source**:
  - Licensed under **Apache 2.0**.
- **Speed and simplicity:**
  - Focuses on simplicity and speed, designed to make analysis and retrieval efficient while being intuitive to use.



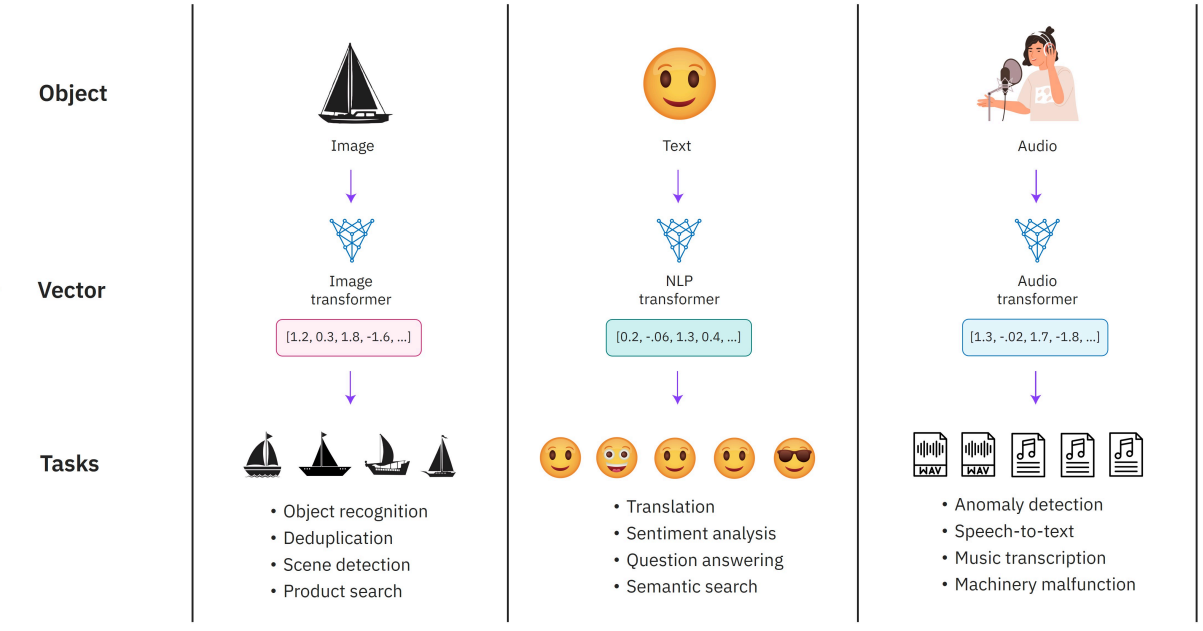


Vector Embeddings and Vector Databases

Traditional databases are great at storing and retrieving data based on exact matches. However, the world of AI necessitates a more nuanced approach to evaluate data based on similar characteristics. Vector embeddings are essential for this data task.

Imagine representing data points, including text, images, and other complex data types, as vectors in a high-dimensional space. Vector points that exist closer together in this space hold greater semantic similarity.

As seen in the following infographic, you can analyze vectorized images for tasks such as object recognition, deduplication, scene detection, and product search. You can analyze text for translation, sentiment analysis, answers to questions, and semantic search. And finally, you can analyze audio files for anomaly detection, speech-to-text, music transcripts, and machinery malfunctions. Summarized, vector data tranformation is available for images, text and audio data analysis.

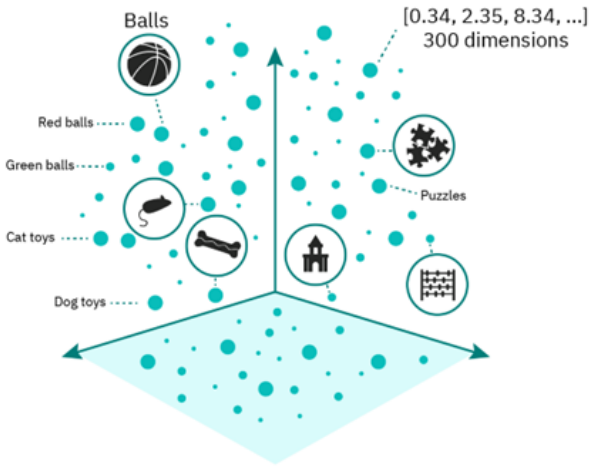


Next, imagine you have a room with a bunch of different toys scattered around. Each toy is represented by a list of qualities, like color, shape, size, and material. Now, imagine each toy as a point

in a space where each quality is in a different direction; you'd have a high-dimensional space. Similar toys, like two different balls, would be closer together in this space because they share many qualities, while something different, like a doll, might be farther away. So, when representing data points as vectors in a high-dimensional space, it's like assigning each toy a unique set of coordinates based on its qualities, and toys with similar coordinates (or qualities) end up closer together.

A Chroma DB vector database could be incredibly useful in the scenario described above. Let's say you have a collection of images of toys, and you want to organize them based on their visual similarities. You could use Chroma DB to convert each image into a high-dimensional vector representation, where each dimension represents different aspects of the image, such as color distribution, texture, and shape.

Once you have these vector representations, you can store them in the Chroma DB database. Now, when you want to find similar images to a given one, you can simply query the database with the vector representation of that image. Chroma DB will then efficiently search through all the vectors in the database and retrieve the ones that are closest to the query vector.



Analyzing the data: Similarity search algorithms

The nearest neighbor search algorithm

What does the nearest neighbor search algorithm indicate about the point of reference or the query?

Depending on the distance or the similarity metric, the nearest neighbor indicates that most features of the query are similar to that of the nearest neighbor. You can also use this algorithm to synthesize data. For instance, Nearest neighbor algorithm is used in image data for pixelating and improving the quality of the image.

This algorithm represents complex data such as text, images, videos, and others as vectors in a highly multidimensional space, considering various data attributes. For instance, text data attributes could include vectors that represent language, tense, tone, and sentiment.

During the machine learning process, developers create and train a model on all the data, considering specific attributes. The training on these features generates the vector output. Since the training involves the same attributes, vectors that are closest to each other or have the highest similarity scores represent similar data.

How far can the nearest neighbor be?

There should be a threshold that needs to be set to consider the validity of the nearest neighbor. If there are datapoints in the set, one of the datapoints in all probabilities will be closer than the others,

making it the nearest neighbor in comparison. But that doesn't have to necessarily indicate that there is any sort of similarity between the two data points.

It is always logical to set a threshold or distance or similarity to consider a point as the nearest neighbor.

For example, if the cosine similarity between two vectors is .5, it makes sense to infer that there are reasonable common features. But if the similarity is closer to 0, but a valid number, then similarity is minimal.

The Chroma DB query engine

The heart of Chroma DB lies in its powerful query engine. This engine is optimized for nearest neighbor search methods. Instead of searching for exact matches, the database can apply the techniques needed to identify data points with the closest vectors in the embedding space, effectively returning the most semantically similar information.

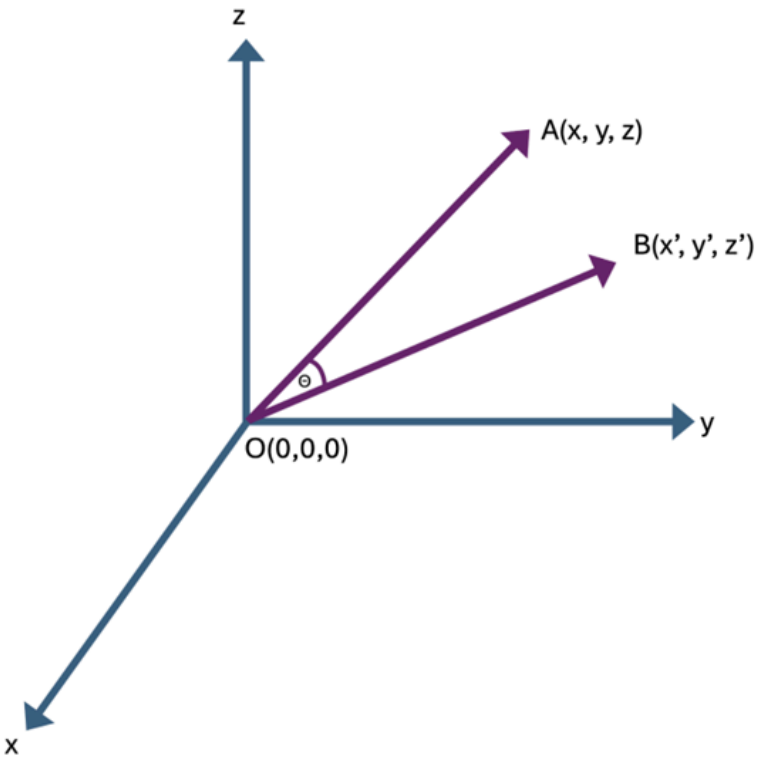
Popular similarity search methods

Developers can employ many methods to determine such distances and similarity scores, including the cosine similarity, Manhatten search and Euclidean distance. Let's explore the details associated with each of these methods.

Cosine similarity

Imagine two vectors as arrows in space. Cosine similarity calculates the angle between these arrows. A smaller angle indicates higher similarity. This metric is ideal for capturing semantic relationships because it focuses on the direction of the vectors, not their magnitude. For example, even if two documents use different words but express similar ideas, cosine similarity will likely identify them as close.

To explain more, cosine similarity is a value that ranges from 0 to 1. The values closer to 0 depict dissimilarity and the values close to 1 depict similarity. It also considers the angle between the two vectors as the vectors also have directions.



Cosine similarity is calculated by using the following formula where A and B are vectors.

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

Now, consider three vectors with three dimensions.

For example, "Hello" is represented by vector A [5,4,2]. "Hi"" is represented by vector B [4,3,2], and "World" is represented by vector C [3,1,4].

Let Θ be the angle between A and B and Θ1 be the angle between A and C.

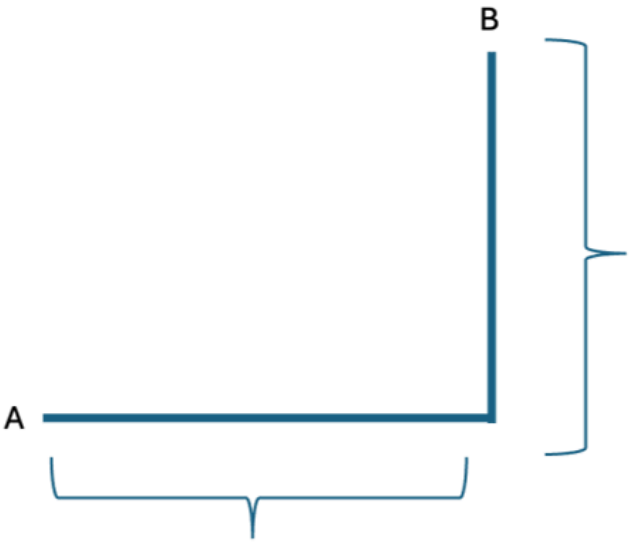
$$Cos(\Theta) = \frac{(5X4)+(4X3)+(2X2)}{\sqrt{5^2+4^2+2^2}\ X\sqrt{4^2+3^2+2^2}} = \frac{36}{36.125} = .996$$

$$Cos(\Theta_1) = \frac{(5X3)+(4X1)+(2X4)}{\sqrt{5^2+4^2+2^2}\ X\sqrt{3^2+1^2+4^2}} = \frac{27}{34.205} = .789$$

The cosine similarity between A and B is higher than A and C, which means B is more similar to A than C. Calculations for vectors obtained from text, image, video, and other data use the same formula.

Manhattan distance

Manhattan distance, known as L1 distance, is understood as the distance you cover as you walk from point A to B as if you were walking in the city of Manhattan. The distance is measured along the axes at right angles.



$$L1 = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|$$

Let's use the same example, which includes the three vectors we used in the previous example. "Hello|" is represented by vector A [5,4,2], "Hi" is represented by vector B [4,3,2], and "World" is represented by vector C [3,1,4].

$$L1(A\ and\ B) = |5 - 4| + |4 - 3| + |2 - 2| = 2$$

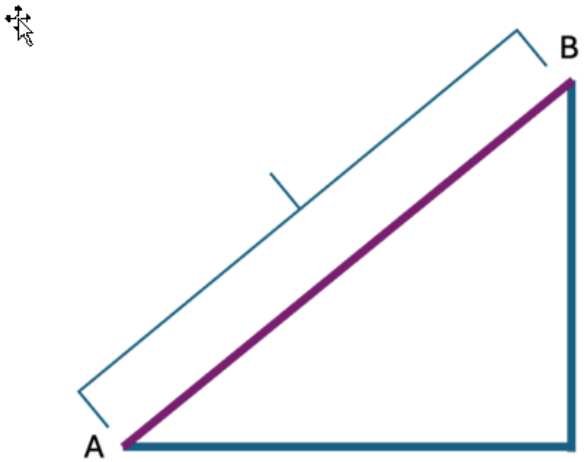
$$L1(A\ and\ C) = |5 - 3| + |4 - 1| + |2 - 4| = 7$$

As you can see, the distance between A and C is greater than the distance between A and B, meaning that A is much closer to B than C. These answers are comparable to the inference from cosine similarity.

Euclidean distance

This method calculates the straight-line distance between two points (vectors) in the embedding space. While simpler to understand, Euclidean distance might not be the best choice for semantic search. It

treats all dimensions equally, and documents with very different word frequencies might be considered close simply because they use some of the same words.
Euclidean distance, also known as L2 distance, is the distance between point A to point B as the bird flies. Euclidean distance is the measure of the straight line from point A to point B. So, visually, Euclidean distance might look like a right-angle triangle.



$$L2 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Here you can see that the Euclidean distance between A and C is greater than the Euclidean distance between A and B, meaning A is a lot closer to B than C. These answers are comparable to the inference from cosine similarity and Manhattan distance.

$$L2(A\ and\ B) = \sqrt{(5-4)^2 + (4-3)^2 + (2-2)^2} = 1.4142$$

$$L2(A\ and\ C) = \sqrt{(5-3)^2 + (4-1)^2 + (2-4)^2} = 4.1231$$

As observed in the illustrated examples, the inferences of cosine similarity, Manhattan distance, and Euclidean distance are similar, as the data points are all represented as vectors that have both dimension and direction; cosine similarity is considered a preferred measure for calculating vector similarities.

Let's review one simpler example of a nearest neighbor search using a text search of the following sentence:

> "What does the Kangaroo do?"

The vector embedding of this sentence, when compared to the following sentences to find cosine similarity, has the following output.

Sentences	Cosine Similarity
"The horse is galloping",	0.3361197784331184
"The owl is hooting",	0.4088818753877638
"The rabbit is hopping"	0.4343758141976563
"The koala is munching",	0.5371295467295504
"The penguin is waddling",	0.33488226610040134
"**The kangaroo is hopping**",	**0.7776482744569905**
"The fox is prowling",	0.33915101907071954
"The parrot is squawking",	0.3771924290381113
"The turtle is crawling",	0.27591877734423753
"The cheetah is sprinting"	0.43455598908399584

The nearest neighbor for the sentence is "The kangaroo is hopping," which has the highest cosine similarity. An ideal threshold for this scenario would be .6. Anything over .6 can be ignored. If the sentence "the Kangaroo is hopping" was not in the set of sentences to be considered, there will be no nearest neighbor when threshold of .6 is set.

Choosing the right distance metric

Developers generally prefer cosine similarity for semantic search tasks in Chroma DB because this method prioritizes the direction (meaning) of the vectors. However, Euclidean distance can be a suitable option for the following conditions:

- Your data has a high degree of uniformity regarding word usage.
- You're primarily interested in finding exact matches or close variations of the query terms.

Additional developer resources

Chroma DB empowers developers with user-friendly Software Development Kits (SDKs). Currently, Chroma offers Python and JavaScript/TypeScript SDKs, which streamline the process of interacting with the database from this application code.

Additional query types available using Chroma DB

Range search and filtering by metadata are two additional query types you can use with Chroma DB.

Range search

This query retrieves embeddings within a specific radius of the query vector. It allows you to find data points that share a certain level of semantic similarity with your query but not necessarily the absolute closest ones.

Example: You might use a range search to recommend similar movies to a user based on a movie they enjoyed. The retrieved embeddings wouldn't necessarily be the most similar movies, but they would share some level of thematic or genre similarity.

Filtering by metadata

While Chroma DB excels at semantic similarity search, you can combine vector searches with filtering based on additional metadata associated with the embeddings.

Example: You could search for similar products (using nearest neighbor search) and then filter the results only to show products within a specific price range.

Next, let's summarize the process of implementing Chroma DB architecture and analysis capabilities.

Implementing Chroma DB: A summarized step-by-step process

1. **Get embeddings**: Take the actions needed to convert words, images, or data into numbers and, specifically, vector representations.
2. **Create collections**: Create collections to group data like tables in a relational database.
3. **Put data into collections**: Save the data, like text and embeddings, in collections after preprocessing the data and generating vector embeddings.
4. **Perform collection operations**: Use Chroma DB to delete, change, or rename collections, among other actions, which gives you more ways to work with your data.
5. **Use text or vector searches to find information** to locate your answers based on semantic meaning or vector similarity, which is a useful way to get the most out of your data.