Run Chroma DB on a local machine and as a Docker container.

Abhishek V Tatachar · Follow
8 min read · Aug 15, 2023

Unlike traditional data, text embeddings are high-dimensional numerical representations that capture the semantic relationships and contextual information of natural text. Traditional databases struggle to handle the complexity and dimensionality of these embeddings effectively. Vector databases, however, are optimised to manage and retrieve high-dimensional data efficiently, making them ideal for storing text embeddings. In vector databases, text embeddings are stored as vectors in a manner that preserves their relative positions in high-dimensional space.

Recently, I have had a chance to explore text embeddings and vector databases. I came across an amazing **open-source** vector database called Chroma DB. As the document suggests, **chromadb** is "the AI-native open-source embedding database". Personally, I find chromadb to be one of the well documented and packaged open-source vector databases. You can find the complete documentation of chromadb here: https://docs.trychroma.com/.

In this article, I have provided a walkthrough of two ways in which Chroma DB can be implemented. First of all, we see how we can implement chroma db to load/save data on the local machine and then we see how chroma db can be run on a docker container.

Save/Load data from local machine

First things first install chromadb using pip

```
pip3 install chromadb
```

Once we have chromadb installed, we can go ahead and create a persistent client for chromadb.

```
import os
import chromadb
from chromadb.config import Settings
DIR = os.path.dirname(os.path.abspath(__file__))
DB_PATH = os.path.join(DIR, 'data')
chroma_client = chromadb.PersistentClient(path=DB_PATH, settings=Settings(allow
sample_collection = chroma_client.get_or_create_collection(name="sample_collect
documents = [
    "Mars, often called the 'Red Planet', has captured the imagination of scier
    "The Hubble Space Telescope has provided us with breathtaking images of dis
    "The concept of a black hole, where gravity is so strong that nothing can e
    "The Renaissance was a pivotal period in history that saw a flourishing of
    "The Industrial Revolution marked a significant shift in human society, lea
    "The ancient city of Rome was once the center of a powerful empire that spa
    "Dolphins are known for their high intelligence and social behavior, often
    "The chameleon is a remarkable creature that can change its skin color to b
    "The migration of monarch butterflies spans thousands of miles and involves
    "Christopher Nolan's 'Inception' is a mind-bending movie that explores the
    "The 'Lord of the Rings' trilogy, directed by Peter Jackson, brought J.R.R.
    "Pixar's 'Toy Story' was the first feature-length film entirely animated us
    "Superman, known for his incredible strength and ability to fly, is one of
    "Black Widow, portrayed by Scarlett Johansson, is a skilled spy and assassi
    "The character of Iron Man, played by Robert Downey Jr., kickstarted the im
]
metadatas = [{'category': "Space"}, {'category': "Space"}, {'category': "Space"
ids = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14

sample_collection.add(documents=documents, metadatas=metadatas, ids=ids)

query_result = sample_collection.query(query_texts="Give me some facts about sp
result_documents = query_result["documents"][0]

for doc in result_documents:
    print(doc)
```

What are we doing in the above code block? First of all, we define where in our local storage we want to store our data (embeddings). The DB_PATH variable points out the local folder that stores the data. Next, we create a persistent client for ChromaDB. ***PersistentClient()*** creates a persistent instance of Chroma that saves to disk. The document suggests that this is useful for testing and development but is not recommended for production use.

We then create a collection for storing the documents, metadatas, embeddings, and ids. Collection is a primitive that Chroma DB uses to store data. The function ***get_or_create_collection()*** fetches an exiting collection if a collection with the given name exists; otherwise, it creates a new one.

The next step is to add our data to the collection. Chroma DB by default uses a sentence transformer model to calculate embeddings; however, this can be overridden by a model of one's choice, and this choice has to be strictly specified at the time of collection creation. (Refer to the documentation; it clearly states how the embedding function can be changed.)

When we do ***collection.add()***, what happens here is that the documents provided are embedded using the specified or default embedding model and stored within the collection in an n-dimensional space.

Finally, we query the document collection with the query text. What exactly happens here is that the provided text is embedded using the same model that was used while creating the collection, and then a similarity is calculated between the query embeddings and the embeddings within the collection to find the best match. The n_results variable specifies the number of results that we are looking for.

Running Chroma DB as a docker-container

We just saw how we could load/save data locally on the disk with Chroma DB. But what if we want to use Chroma DB in production? We might then want to run it in client-server mode, where we have a remote server running and then connect to it using an HTTP request.

We can achieve this by creating a Docker container for the Chroma DB that runs as a server and then creating a Chroma client that connects to the server.

First things first, setting up a chroma client:

```
import chromadb
from chromadb.config import Settings

chroma_client = chromadb.HttpClient(host="chroma", port = 8000, settings=Settir
documents = [
    "Mars, often called the 'Red Planet', has captured the imagination of scier
    "The Hubble Space Telescope has provided us with breathtaking images of dis
    "The concept of a black hole, where gravity is so strong that nothing can e
    "The Renaissance was a pivotal period in history that saw a flourishing of 
    "The Industrial Revolution marked a significant shift in human society, lea
    "The ancient city of Rome was once the center of a powerful empire that spa
    "Dolphins are known for their high intelligence and social behavior, often 
    "The chameleon is a remarkable creature that can change its skin color to b
    "The migration of monarch butterflies spans thousands of miles and involves
    "Christopher Nolan's 'Inception' is a mind-bending movie that explores the 
    "The 'Lord of the Rings' trilogy, directed by Peter Jackson, brought J.R.R.
    "Pixar's 'Toy Story' was the first feature-length film entirely animated us
    "Superman, known for his incredible strength and ability to fly, is one of 
    "Black Widow, portrayed by Scarlett Johansson, is a skilled spy and assassi
    "The character of Iron Man, played by Robert Downey Jr., kickstarted the im
]
metadatas = [{'source': "Space"}, {'source': "Space"}, {'source': "Space"}, {'s
ids = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14

collection_status = False
while collection_status != True:
    try:
        document_collection = chroma_client.get_or_create_collection(name="samp
        collection_status = True
    except Exception as e:
        pass

document_collection.add(documents=documents, metadatas=metadatas, ids=ids)

results = document_collection.query(query_texts="Give me some facts about space
result_documents = results["documents"][0]
for doc in result_documents:
    print(doc)
```

I’ll not go deep into the code here; most of the code is very similar to what we have done earlier. But something to note here is that we have moved from a persistent client to an HTTP client and provided a host name and the port to connect on.

Next is to create a dockerfile for our client; we want our client to also run as a container service, and then make the two services (the Chroma client and the Chroma server) communicate.

```
FROM python:latest
WORKDIR /app
COPY requirements.txt /app/
RUN pip install -r requirements.txt
COPY . /app/
CMD ["python", "chroma_client.py"]
```

What we are doing here:

1. ***FROM python:latest***: Start with a base image of the latest Python version.
2. ***WORKDIR /app***: Set the working directory inside the container to /app. This is where your code will be copied and where commands will be executed.
3. ***COPY requirements.txt /app/***: Copy the requirements.txt file from your local machine into the container's /app directory.
4. ***RUN pip install -r requirements.txt***: Install the Python packages listed in requirements.txt using pip, which is Python's package manager.
5. ***COPY . /app/***: Copy all the files and folders from your local directory (where the Dockerfile is located) into the container's /app directory.
6. ***CMD ["python", "chroma_client.py"]***: Specify the default command that will be run when the container starts. In this case, it runs the chroma_client.py file using the Python interpreter.

The next step is to create a docker-compose.yml that defines the two services. In docker-compose, we create a network bridge that allows the two services to communicate and exchange data.

```
version: "3.8"
services:
  application:
    build:
      context: .
      dockerfile: ./Dockerfile
    image: application
    container_name: application
    volumes:
      - ./:/app/
    networks:
      - net

  chroma:
    image: ghcr.io/chroma-core/chroma:latest
    volumes:
      - index_data:/chroma/.chroma/index
    ports:
      - 8000:8000
    networks:
      - net

volumes:
  index_data:
    driver: local
  backups:
    driver: local

networks:
  net:
    driver: bridge
```

For the application (which is our chroma client), I have used the docker file that we have defined to build an image. For the Chroma DB, There is an image already available. We create containers using this image and expose port 8000, which is used by the application to set up a connection and communicate.

To create containers and run chroma-db and the application, all you have to do in your command line is (PS: Have your docker desktop running).

```
docker-compose up --build
```

You can find the complete repository for the Docker approach here: https://github.com/abhitatachar2000/dockerize-chromadb

As we wrap up our exploration of vector databases, my sincere attempt here is to show how Chroma DB can be used to create and store text embeddings. I've personally found Chroma DB to be quite special; it keeps getting better, and its documentation is super helpful.

I encourage the readers to try this out as well. Consider making your own examples and trying these databases in different situations. I've shared a simple app that talks to Chroma DB, but there's a lot more you can discover. You can find even more interesting stuff in Chroma DB's documentation at https://docs.trychroma.com. It's like a map to explore all the features.

Your thoughts and ideas are really important. I'd love to hear what you think and how we can make things even cooler. If you enjoyed this journey, maybe share this blog with someone who'd also be excited about this technology. Let's spread the word and learn together.

Thanks for being an amazing reader. Until next time, keep exploring and keep learning!

Vector Database Chromadb Docker Large Language Models Text Embedding

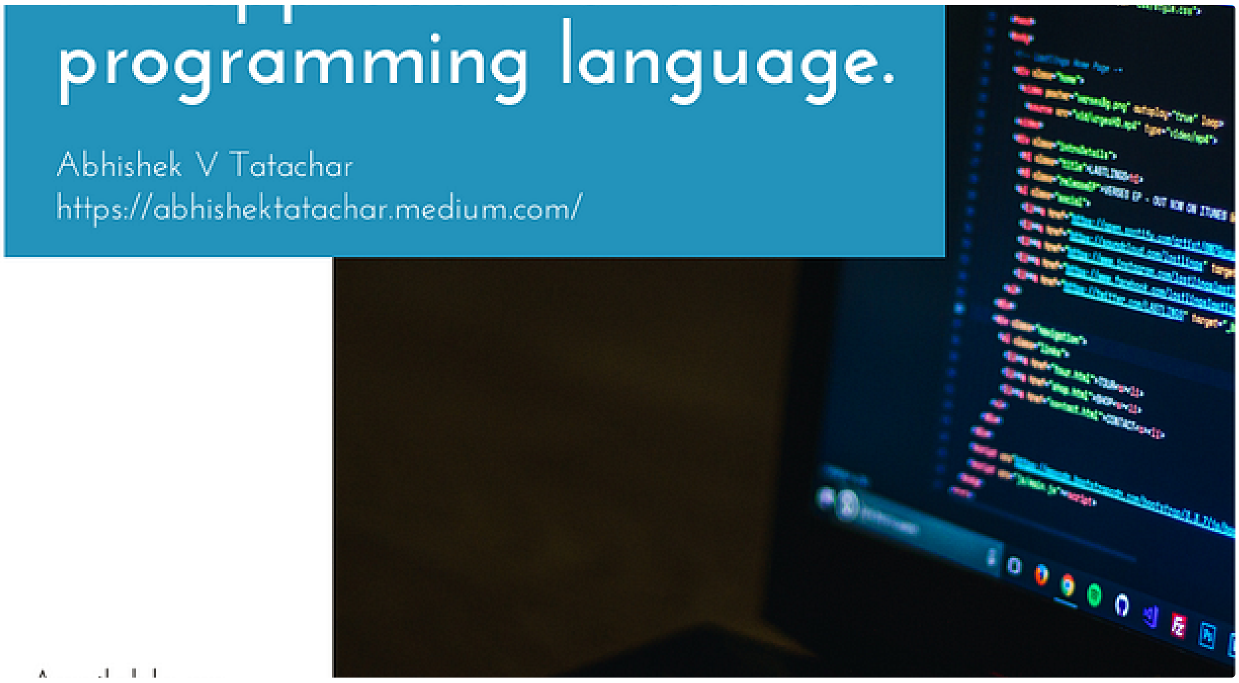
Follow

Written by Abhishek V Tatachar

4 Followers

A Data Scientist specialising in crafting AI-powered solutions for businesses. Speaks about AI Applications, GenerativeAI, Prompt Engineering and Productivity

More from Abhishek V Tatachar



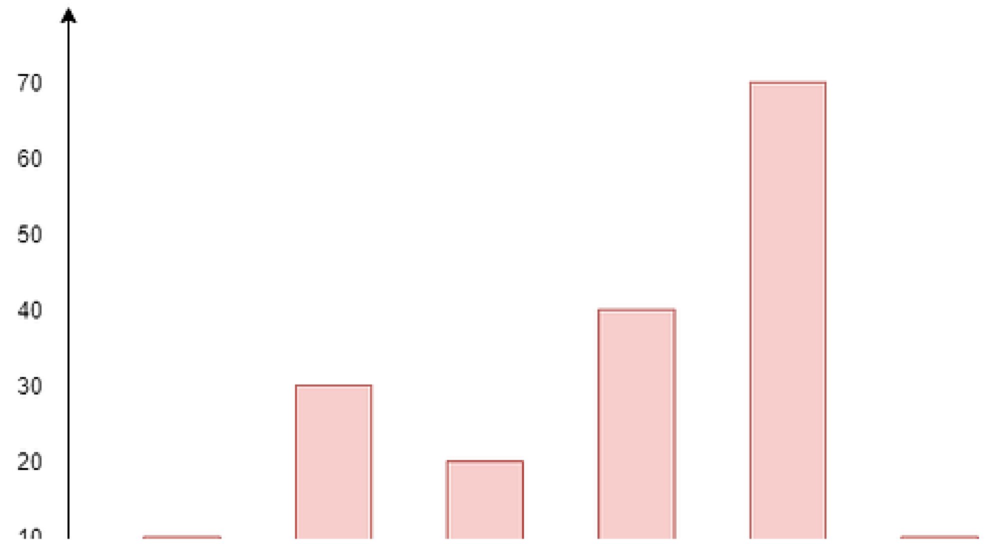
Abhishek V Tatachar

An approach to learn a programming language.

Hey everyone, this is Abhishek V Tatachar, an engineering student at Global Academy of Technology. I have been coding or practically…

3 min read · Apr 19, 2021

56



Abhishek V Tatachar

From the world of data.

“I need a cloth of 1 meter”. “This ice cream is chunky”. “The average height of people in my class is 5 foot and 6 inches”. If you are...

6 min read · May 10, 2021

2

Abhishek V Tatachar

Need help? Then ask for it.

We all keep hearing to this statement “A friend in need is a friend in deed”, and also believe in it. What do we do when we need some help...

2 min read · Apr 6, 2022

53

See all from Abhishek V Tatachar

Recommended from Medium

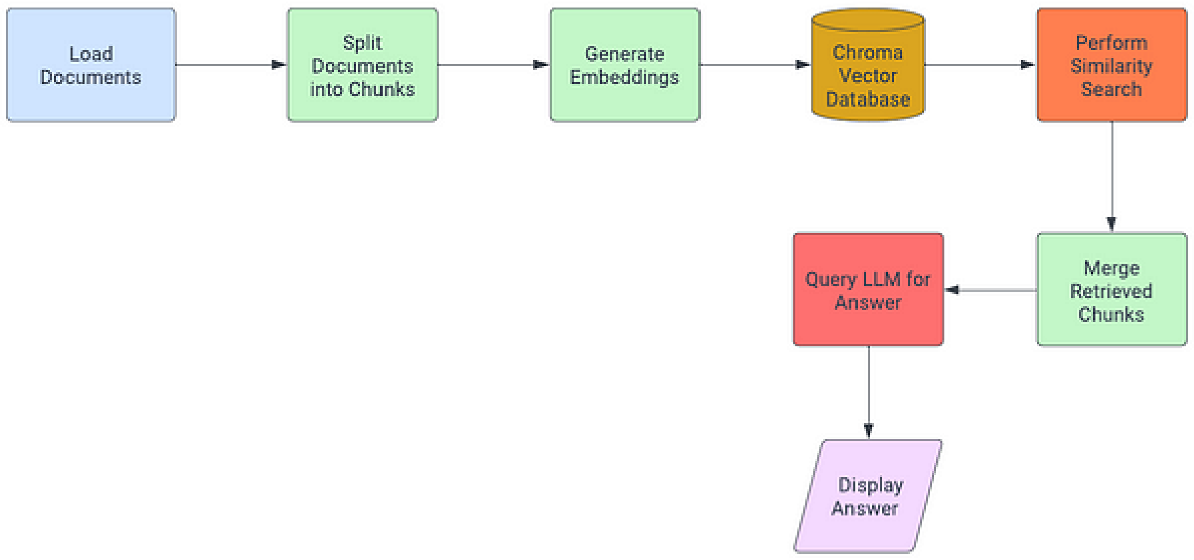
Chris McKenzie

Setting Up Your First ChromaDB Server

Guide for JavaScript Engineers who want to start using vector databases

6 min read · Sep 23, 2023

7



Abdulmajeed Alroumi

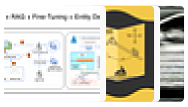
Enhancing Semantic Search with LangChain, Vector Databases, and Llama2–70B-Chat

Introduction:

8 min read · Sep 19, 2023

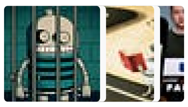
17 1

Lists



Natural Language Processing

1202 stories · 679 saves

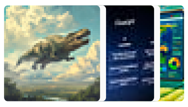


AI Regulation

6 stories · 318 saves

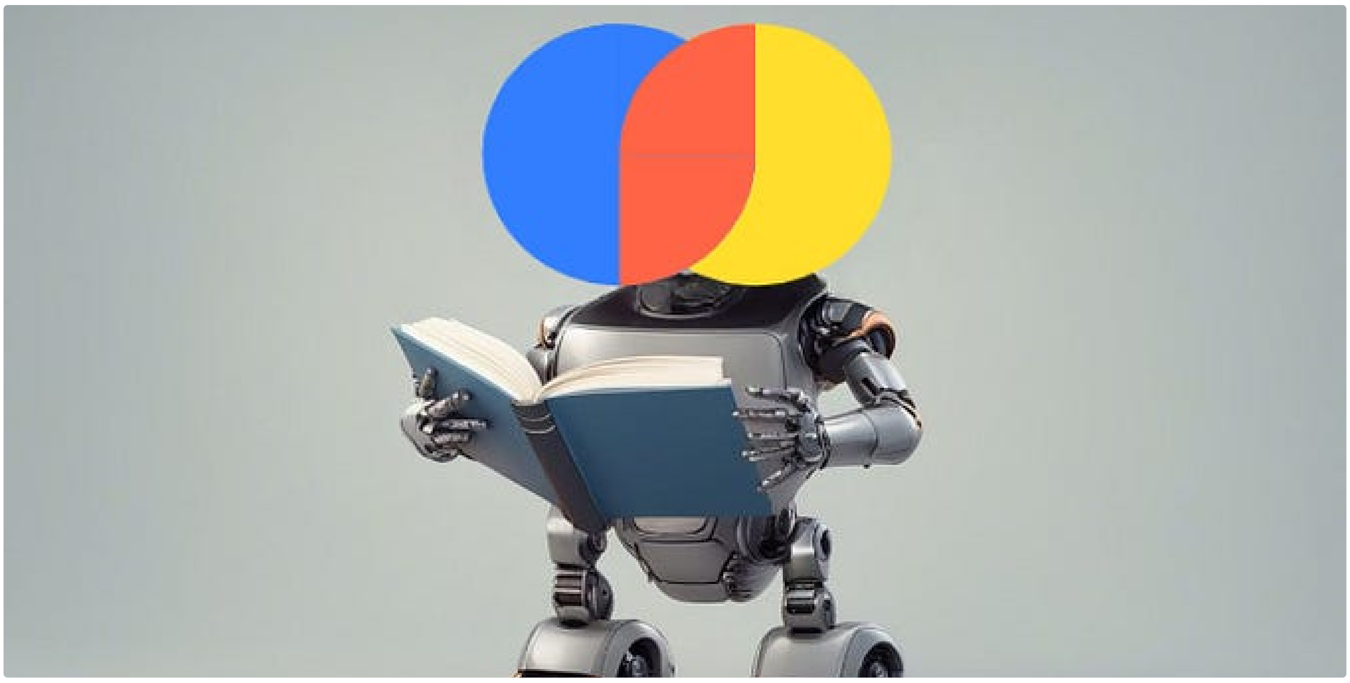
Coding & Development

11 stories · 449 saves



ChatGPT prompts

40 stories · 1127 saves



Stephen Collins

How to use Chroma to store and query vector embeddings

Chroma is an open-source embedding database designed to store and query vector embeddings efficiently, enhancing Large Language Models...

5 min read · Oct 1, 2023

2

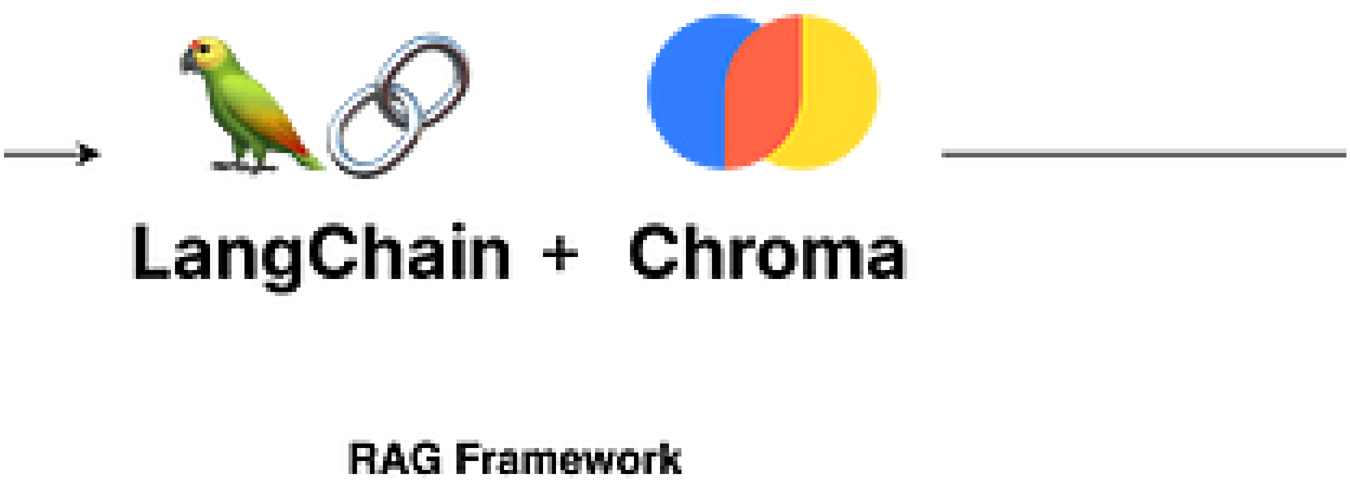
Amikos Tech

Running ChromaDB — Part 1: Local Server

Simple and easy ways to get started with Chroma locally

5 min read · Dec 22, 2023

18



Duy Huynh

Build your own RAG and run it locally: Langchain + Ollama + Streamlit

With the rise of Large Language Models and its impressive capabilities, many fancy applications are being built on top of giant LLM...

6 min read · Dec 5, 2023

403 2

Ahmed Jawed

Unleashing LLaMA’s Power: Crafting a Flask API to Seamlessly Load and Engage with Language Models

Introduction

7 min read · Oct 3, 2023

8

See more recommendations