**Chroma**

**Chroma is the open-source embedding database**. Chroma makes it easy to build LLM apps by making knowledge, facts, and skills pluggable for LLMs.

Chroma gives you the tools to:

* store embeddings and their metadata
* embed documents and queries
* search embeddings

Chroma prioritizes:

* simplicity and developer productivity
* analysis on top of search
* it also happens to be very quick

Chroma consists of a **Python** client SDK, **JavaScript/TypeScript** client SDK and a server application. Check out the **[Colab demo](https://colab.research.google.com/drive/1QEzFyqnoFxq7LUGyP1vzR4iLt9PpCDXv?usp=sharing" \t "_blank)**.(yes, it can run in a notebook 😄)

Chroma is licensed under [**Apache 2.0**](https://github.com/chroma-core/chroma/blob/main/LICENSE)

**Python**[**​**](https://docs.trychroma.com/#python)

In Python, Chroma can run **in-memory** or in **client/server** (in alpha) mode.

pip install chromadb

**JavaScript**[**​**](https://docs.trychroma.com/#javascript)

In JavaScript, Chroma runs in **client/server** mode and talks to a Python backend.

npm install --save chromadb # yarn add chromadb

Continue with the full [**getting started guide**](https://docs.trychroma.com/getting-started).

[**🔑 Getting started**](https://docs.trychroma.com/getting-started)  
[**🧪 Usage Guide**](https://docs.trychroma.com/usage-guide)  
[**🧬 Embeddings**](https://docs.trychroma.com/embeddings)  
[**📄 API Reference**](https://docs.trychroma.com/api-reference)  
[**👽 About**](https://docs.trychroma.com/about)

[**💬 Join Community Discord**](https://discord.gg/MMeYNTmh3x)  
[**@trychroma**](https://twitter.com/trychroma)

**Language Support[​](https://docs.trychroma.com/" \l "language-support" \o "Direct link to Language Support)**

|  | **in-memory** | **client & server** |
| --- | --- | --- |
| Python | ✅ | ✅ |
| Javascript | ➖ | ✅ |
| Ruby | ➖ | [from @mariochavez](https://github.com/mariochavez/chroma) |
| C# | ➖ | *from Microsoft (coming soon)* |
| Other? | ❓ | ❓ |

We welcome contributions for other languages!

Learn more on the [**Community Discord**](https://discord.gg/MMeYNTmh3x).

# 🔑 Getting started

**Select a language**

* **Python**
* **JavaScript**

Chroma is a database for building AI applications with embeddings. It comes with everything you need to get started built in, and runs on your machine. A [**hosted version**](https://airtable.com/shrOAiDUtS2ILy5vZ) is coming soon!

### 1. Install[​](https://docs.trychroma.com/getting-started#1-install)

pip install chromadb

*\* chromadb currently does not support Python 3.11 because of pytorch*

### 2. Get the Chroma Client[​](https://docs.trychroma.com/getting-started#2-get-the-chroma-client)

import chromadb  
chroma\_client = chromadb.Client()

### 3. Create a collection[​](https://docs.trychroma.com/getting-started#3-create-a-collection)

Collections are where you'll store your embeddings, documents, and any additional metadata. You can create a collection with a name:

collection = chroma\_client.create\_collection(name="my\_collection")

### 4. Add some text documents to the collection[​](https://docs.trychroma.com/getting-started#4-add-some-text-documents-to-the-collection)

Chroma will store your text, and handle tokenization, embedding, and indexing automatically.

collection.add(  
 documents=["This is a document", "This is another document"],  
 metadatas=[{"source": "my\_source"}, {"source": "my\_source"}],  
 ids=["id1", "id2"]  
)

If you have already generated embeddings yourself, you can load them directly in:

collection.add(  
 embeddings=[[1.2, 2.3, 4.5], [6.7, 8.2, 9.2]],  
 documents=["This is a document", "This is another document"],  
 metadatas=[{"source": "my\_source"}, {"source": "my\_source"}],  
 ids=["id1", "id2"]  
)

### 5. Query the collection[​](https://docs.trychroma.com/getting-started#5-query-the-collection)

You can query the collection with a list of query texts, and Chroma will return the **n** most similar results. It's that easy!

results = collection.query(  
 query\_texts=["This is a query document"],  
 n\_results=2  
)

By default data stored in Chroma is ephemeral making it easy to prototype scripts. It's easy to make Chroma persistent so you can reuse every collection you create and add more documents to it later. It will load your data automatically when you start the client, and save it automatically when you close it. Check out the [**Usage Guide**](https://docs.trychroma.com/usage-guide) for more info.

Find **[chromadb on PyPI](https://pypi.org/project/chromadb/" \t "_blank)**.

## 📚 Next steps[​](https://docs.trychroma.com/getting-started" \l "-next-steps" \o "Direct link to 📚 Next steps)

* Chroma is designed to be simple enough to get started with quickly and flexible enough to meet many use-cases. You can use your own embedding models, query Chroma with your own embeddings, and filter on metadata. To learn more about Chroma, check out the [Usage Guide](https://docs.trychroma.com/usage-guide) and [API Reference](https://docs.trychroma.com/api-reference).
* Chroma is integrated in [LangChain](https://python.langchain.com/en/latest/modules/indexes/vectorstores.html?highlight=chroma" \l "langchain.vectorstores.Chroma" \t "_blank) (**python** and **js**), making it easy to build AI applications with Chroma. Check out the [integrations](https://docs.trychroma.com/integrations) page to learn more.
* You can [deploy a persistent instance](https://docs.trychroma.com/deployment) of Chroma to an external server, to make it easier to work on larger projects or with a team.

## Coming Soon[​](https://docs.trychroma.com/getting-started" \l "coming-soon" \o "Direct link to Coming Soon)

* A hosted version of Chroma, with an easy to use web UI and API
* Multiple datatypes, including images, audio, video, and more

# 🧪 Usage Guide

**Select a language**

* **Python**
* **JavaScript**

## Initiating a persistent Chroma client[​](https://docs.trychroma.com/usage-guide#initiating-a-persistent-chroma-client)

import chromadb

You can configure Chroma to save and load from your local machine. Data will be persisted automatically and loaded on start (if it exists).

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

The **path** is where Chroma will store its database files on disk, and load them on start.

USE A SINGLE CLIENT AT-A-TIME

Having many clients that are loading and saving to the same path can cause strange behavior including data deletion. As a general practice, create a Chroma client once in your application, and pass it around instead of creating many clients.

The client object has a few useful convenience methods.

client.heartbeat() # returns a nanosecond heartbeat. Useful for making sure the client remains connected.  
client.reset() # Empties and completely resets the database. ⚠️ This is destructive and not reversible.

## Running Chroma in client/server mode[​](https://docs.trychroma.com/usage-guide#running-chroma-in-clientserver-mode)

Chroma can also be configured to use an on-disk database, useful for larger data which doesn't fit in memory. To run Chroma in client server mode, run the docker container:

docker-compose up -d --build

Then update your chroma client to point at the docker container. Default: **localhost:8000**

import chromadb  
chroma\_client = chromadb.HttpClient(host='localhost', port=8000)

That's it! Chroma's API will run in **client-server** mode with just this change.

#### Using the python http-only client[​](https://docs.trychroma.com/usage-guide#using-the-python-http-only-client)

If you are running chroma in client-server mode. You may not require the full Chroma library and instead only the client library on your client machine. In this case, you can install the **chromadb-client** package. This package is a lightweight HTTP client for the server with a minimal dependency footprint.

pip install chromadb-client

import chromadb  
from chromadb.config import Settings  
# Example setup of the client to connect to your chroma server  
client = chromadb.HttpClient(host='localhost', port=8000)

Note that the **chromadb-client** package is a subset of the full Chroma library and does not include all the dependencies. If you want to use the full Chroma library, you can install the **chromadb** package instead. Most importantly, there is no default embedding function. If you add() documents without embeddings, you must have manually specified an embedding function and installed the dependencies for it.

## Using collections[​](https://docs.trychroma.com/usage-guide#using-collections)

Chroma lets you manage collections of embeddings, using the **collection** primitive.

### Creating, inspecting, and deleting Collections[​](https://docs.trychroma.com/usage-guide#creating-inspecting-and-deleting-collections)

Chroma uses collection names in the url, so there are a few restrictions on naming them:

* The length of the name must be between 3 and 63 characters.
* The name must start and end with a lowercase letter or a digit, and it can contain dots, dashes, and underscores in between.
* The name must not contain two consecutive dots.
* The name must not be a valid IP address.

Chroma collections are created with a name and an optional embedding function. If you supply an embedding function, you must supply it every time you get the collection.

collection = client.create\_collection(name="my\_collection", embedding\_function=emb\_fn)  
collection = client.get\_collection(name="my\_collection", embedding\_function=emb\_fn)

CAUTION

If you later wish to **get\_collection**, you MUST do so with the embedding function you supplied while creating the collection

The embedding function takes text as input, and performs tokenization and embedding. If no embedding function is supplied, Chroma will use [**sentence transfomer**](https://www.sbert.net/index.html) as a default.

You can learn more about [**🧬 embedding functions**](https://docs.trychroma.com/embeddings), and how to create your own.

Existing collections can be retrieved by name with **.get\_collection**, and deleted with **.delete\_collection**. You can also use **.get\_or\_create\_collection** to get a collection if it exists, or create it if it doesn't.

collection = client.get\_collection(name="test") # Get a collection object from an existing collection, by name. Will raise an exception if it's not found.  
collection = client.get\_or\_create\_collection(name="test") # Get a collection object from an existing collection, by name. If it doesn't exist, create it.  
client.delete\_collection(name="my\_collection") # Delete a collection and all associated embeddings, documents, and metadata. ⚠️ This is destructive and not reversible

Collections have a few useful convenience methods.

collection.peek() # returns a list of the first 10 items in the collection  
collection.count() # returns the number of items in the collection  
collection.modify(name="new\_name") # Rename the collection

### Changing the distance function[​](https://docs.trychroma.com/usage-guide#changing-the-distance-function)

**create\_collection** also takes an optional **metadata** argument which can be used to customize the distance method of the embedding space by setting the value of **hnsw:space**.

collection = client.create\_collection(  
 name="collection\_name",  
 metadata={"hnsw:space": "cosine"} # l2 is the default  
 )

Valid options for **hnsw:space** are "l2", "ip, "or "cosine". The **default** is "l2". The equations for each can be found in the docs for Hnswlib [**here**](https://github.com/nmslib/hnswlib/tree/master#python-bindings).

### Adding data to a Collection[​](https://docs.trychroma.com/usage-guide#adding-data-to-a-collection)

Add data to Chroma with **.add**.

Raw documents:

collection.add(  
 documents=["lorem ipsum...", "doc2", "doc3", ...],  
 metadatas=[{"chapter": "3", "verse": "16"}, {"chapter": "3", "verse": "5"}, {"chapter": "29", "verse": "11"}, ...],  
 ids=["id1", "id2", "id3", ...]  
)

If Chroma is passed a list of **documents**, it will automatically tokenize and embed them with the collection's embedding function (the default will be used if none was supplied at collection creation). Chroma will also store the **documents** themselves. If the documents are too large to embed using the chosen embedding function, an exception will be raised.

Each document must have a unique associated **id**. Trying to **.add** the same ID twice will result in only the initial value being stored. An optional list of **metadata** dictionaries can be supplied for each document, to store additional information and enable filtering.

Alternatively, you can supply a list of document-associated **embeddings** directly, and Chroma will store the associated documents without embedding them itself.

collection.add(  
 documents=["doc1", "doc2", "doc3", ...],  
 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"}, {"chapter": "29", "verse": "11"}, ...],  
 ids=["id1", "id2", "id3", ...]  
)

If the supplied **embeddings** are not the same dimension as the collection, an exception will be raised.

You can also store documents elsewhere, and just supply a list of **embeddings** and **metadata** to Chroma. You can use the **ids** to associate the embeddings with your documents stored elsewhere.

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"}, {"chapter": "29", "verse": "11"}, ...],  
 ids=["id1", "id2", "id3", ...]  
)

### Querying a Collection[​](https://docs.trychroma.com/usage-guide#querying-a-collection)

Chroma collections can be queried in a variety of ways, using the **.query** method.

You can query by a set of **query\_embeddings**.

collection.query(  
 query\_embeddings=[[11.1, 12.1, 13.1],[1.1, 2.3, 3.2], ...],  
 n\_results=10,  
 where={"metadata\_field": "is\_equal\_to\_this"},  
 where\_document={"$contains":"search\_string"}  
)

The query will return the **n\_results** closest matches to each **query\_embedding**, in order. An optional **where** filter dictionary can be supplied to filter the results by the **metadata** associated with each document. Additionally, an optional **where\_document** filter dictionary can be supplied to filter the results by contents of the document.

If the supplied **query\_embeddings** are not the same dimension as the collection, an exception will be raised.

You can also query by a set of **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"}  
)

You can also retrieve items from a collection by **id** using **.get**.

collection.get(  
 ids=["id1", "id2", "id3", ...],  
 where={"style": "style1"}  
)

**.get** also supports the **where** and **where\_document** filters. If no **ids** are supplied, it will return all items in the collection that match the **where** and **where\_document** filters.

##### Choosing which data is returned[​](https://docs.trychroma.com/usage-guide#choosing-which-data-is-returned)

When using get or query you can use the include parameter to specify which data you want returned - any of **embeddings**, **documents**, **metadatas**, and for query, **distances**. By default, Chroma will return the **documents**, **metadatas** and in the case of query, the **distances** of the results. **embeddings** are excluded by default for performance and the **ids** are always returned. You can specify which of these you want returned by passing an array of included field names to the includes parameter of the query or get method.

# Only get documents and ids  
collection.get(  
 include=["documents"]  
)  
  
collection.query(  
 query\_embeddings=[[11.1, 12.1, 13.1],[1.1, 2.3, 3.2], ...],  
 include=["documents"]  
)

### Using Where filters[​](https://docs.trychroma.com/usage-guide#using-where-filters)

Chroma supports filtering queries by **metadata** and **document** contents. The **where** filter is used to filter by **metadata**, and the **where\_document** filter is used to filter by **document** contents.

##### Filtering by metadata[​](https://docs.trychroma.com/usage-guide#filtering-by-metadata)

In order to filter on metadata, you must supply a **where** filter dictionary to the query. The dictionary must have the following structure:

{  
 "metadata\_field": {  
 <Operator>: <Value>  
 }  
}

Filtering metadata supports the following operators:

* **$eq** - equal to (string, int, float)
* **$ne** - not equal to (string, int, float)
* **$gt** - greater than (int, float)
* **$gte** - greater than or equal to (int, float)
* **$lt** - less than (int, float)
* **$lte** - less than or equal to (int, float)

Using the $eq operator is equivalent to using the **where** filter.

{  
 "metadata\_field": "search\_string"  
}  
  
# is equivalent to  
  
{  
 "metadata\_field": {  
 "$eq": "search\_string"  
 }  
}

##### Filtering by document contents[​](https://docs.trychroma.com/usage-guide#filtering-by-document-contents)

In order to filter on document contents, you must supply a **where\_document** filter dictionary to the query. The dictionary must have the following structure:

# Filtering for a search\_string  
{  
 "$contains": "search\_string"  
}

##### Using logical operators[​](https://docs.trychroma.com/usage-guide#using-logical-operators)

You can also use the logical operators **$and** and **$or** to combine multiple filters.

An **$and** operator will return results that match all of the filters in the list.

{  
 "$and": [  
 {  
 "metadata\_field": {  
 <Operator>: <Value>  
 }  
 },  
 {  
 "metadata\_field": {  
 <Operator>: <Value>  
 }  
 }  
 ]  
}

An **$or** operator will return results that match any of the filters in the list.

{  
 "$or": [  
 {  
 "metadata\_field": {  
 <Operator>: <Value>  
 }  
 },  
 {  
 "metadata\_field": {  
 <Operator>: <Value>  
 }  
 }  
 ]  
}

### Updating data in a collection[​](https://docs.trychroma.com/usage-guide#updating-data-in-a-collection)

Any property of items in a collection can be updated using **.update**.

collection.update(  
 ids=["id1", "id2", "id3", ...],  
 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"}, {"chapter": "29", "verse": "11"}, ...],  
 documents=["doc1", "doc2", "doc3", ...],  
)

If an **id** is not found in the collection, an error will be logged and the update will be ignored. If **documents** are supplied without corresponding **embeddings**, the embeddings will be recomupted with the collection's embedding function.

If the supplied **embeddings** are not the same dimension as the collection, an exception will be raised.

Chroma also supports an **upsert** operation, which updates existing items, or adds them if they don't yet exist.

collection.upsert(  
 ids=["id1", "id2", "id3", ...],  
 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"}, {"chapter": "29", "verse": "11"}, ...],  
 documents=["doc1", "doc2", "doc3", ...],  
)

If an **id** is not present in the collection, the corresponding items will be created as per **add**. Items with existing **id**s will be updated as per **update**.

### Deleting data from a collection[​](https://docs.trychroma.com/usage-guide#deleting-data-from-a-collection)

Chroma supports deleting items from a collection by **id** using **.delete**. The embeddings, documents, and metadata associated with each item will be deleted. ⚠️ Naturally, this is a destructive operation, and cannot be undone.

collection.delete(  
 ids=["id1", "id2", "id3",...],  
 where={"chapter": "20"}  
)

**.delete** also supports the **where** filter. If no **ids** are supplied, it will delete all items in the collection that match the **where** filter.

# 🧬 Embeddings

**Select a language**

* **Python**
* **JavaScript**

Embeddings are the A.I-native way to represent any kind of data, making them the perfect fit for working with all kinds of A.I-powered tools and algorithms. They can represent text, images, and soon audio and video. There are many options for creating embeddings, whether locally using an installed library, or by calling an API.

Chroma provides lightweight wrappers around popular embedding providers, making it easy to use them in your apps. You can set an embedding function when you create a Chroma collection, which will be used automatically, or you can call them directly yourself.

To get Chroma's embedding functions, import the **chromadb.utils.embedding\_functions** module.

from chromadb.utils import embedding\_functions

## Default: all-MiniLM-L6-v2[​](https://docs.trychroma.com/embeddings#default-all-minilm-l6-v2)

By default, Chroma uses the [**Sentence Transformers**](https://www.sbert.net/) **all-MiniLM-L6-v2** model to create embeddings. This embedding model can create sentence and document embeddings that can be used for a wide variety of tasks. This embedding function runs locally on your machine, and may require you download the model files (this will happen automatically).

default\_ef = embedding\_functions.DefaultEmbeddingFunction()

## Sentence Transformers[​](https://docs.trychroma.com/embeddings#sentence-transformers)

Chroma can also use any [**Sentence Transformers**](https://www.sbert.net/) model to create embeddings.

sentence\_transformer\_ef = embedding\_functions.SentenceTransformerEmbeddingFunction(model\_name="all-MiniLM-L6-v2")

You can pass in an optional **model\_name** argument, which lets you choose which Sentence Transformers model to use. By default, Chroma uses **all-MiniLM-L6-v2**. You can see a list of all available models [**here**](https://www.sbert.net/docs/pretrained_models.html).

## OpenAI[​](https://docs.trychroma.com/embeddings#openai)

Chroma provides a convenient wrapper around OpenAI's embedding API. This embedding function runs remotely on OpenAI's servers, and requires an API key. You can get an API key by signing up for an account at [**OpenAI**](https://openai.com/api/).

This embedding function relies on the **openai** python package, which you can install with **pip install openai**.

openai\_ef = embedding\_functions.OpenAIEmbeddingFunction(  
 api\_key="YOUR\_API\_KEY",  
 model\_name="text-embedding-ada-002"  
 )

To use the OpenAI embedding models on other platforms such as Azure, you can use the **api\_base** and **api\_type** parameters:

openai\_ef = embedding\_functions.OpenAIEmbeddingFunction(  
 api\_key="YOUR\_API\_KEY",  
 api\_base="YOUR\_API\_BASE\_PATH",  
 api\_type="azure",  
 model\_name="text-embedding-ada-002"  
 )

You can pass in an optional **model\_name** argument, which lets you choose which OpenAI embeddings model to use. By default, Chroma uses **text-embedding-ada-002**. You can see a list of all available models [**here**](https://platform.openai.com/docs/guides/embeddings/what-are-embeddings).

## Cohere[​](https://docs.trychroma.com/embeddings#cohere)

Chroma also provides a convenient wrapper around Cohere's embedding API. This embedding function runs remotely on Cohere’s servers, and requires an API key. You can get an API key by signing up for an account at [**Cohere**](https://dashboard.cohere.ai/welcome/register).

This embedding function relies on the **cohere** python package, which you can install with **pip install cohere**.

cohere\_ef = embedding\_functions.CohereEmbeddingFunction(api\_key="YOUR\_API\_KEY", model\_name="large")  
cohere\_ef(texts=["document1","document2"])

You can pass in an optional **model\_name** argument, which lets you choose which Cohere embeddings model to use. By default, Chroma uses **large** model. You can see the available models under **Get embeddings** section [**here**](https://docs.cohere.ai/reference/embed).

### Multilingual model example[​](https://docs.trychroma.com/embeddings#multilingual-model-example)

cohere\_ef = embedding\_functions.CohereEmbeddingFunction(  
 api\_key="YOUR\_API\_KEY",   
 model\_name="multilingual-22-12")  
  
multilingual\_texts = [ 'Hello from Cohere!', 'مرحبًا من كوهير!',   
 'Hallo von Cohere!', 'Bonjour de Cohere!',   
 '¡Hola desde Cohere!', 'Olá do Cohere!',   
 'Ciao da Cohere!', '您好，来自 Cohere！',  
 'कोहेरे से नमस्ते!' ]  
  
cohere\_ef(texts=multilingual\_texts)

For more information on multilingual model you can read [**here**](https://docs.cohere.ai/docs/multilingual-language-models).

## Instructor models[​](https://docs.trychroma.com/embeddings#instructor-models)

The [**instructor-embeddings**](https://github.com/HKUNLP/instructor-embedding) library is another option, especially when running on a machine with a cuda-capable GPU. They are a good local alternative to OpenAI (see the [**Massive Text Embedding Benchmark**](https://huggingface.co/blog/mteb) rankings). The embedding function requires the InstructorEmbedding package. To install it, run **pip install InstructorEmbedding**.

There are three models available. The default is **hkunlp/instructor-base**, and for better performance you can use **hkunlp/instructor-large** or **hkunlp/instructor-xl**. You can also specify whether to use **cpu** (default) or **cuda**. For example:

#uses base model and cpu  
ef = embedding\_functions.InstructorEmbeddingFunction()

or

ef = embedding\_functions.InstructorEmbeddingFunction(  
model\_name="hkunlp/instructor-xl", device="cuda")

Keep in mind that the large and xl models are 1.5GB and 5GB respectively, and are best suited to running on a GPU.

## Google PaLM API models[​](https://docs.trychroma.com/embeddings#google-palm-api-models)

[**Google PaLM APIs**](https://developers.googleblog.com/2023/03/announcing-palm-api-and-makersuite.html) are currently in private preview, but if you are part of this preview, you can use them with Chroma via the **GooglePalmEmbeddingFunction**.

To use the PaLM embedding API, you must have **google.generativeai** Python package installed and have the API key. To use:

palm\_embedding = embedding\_functions.GooglePalmEmbeddingFunction(  
 api\_key=api\_key, model=model\_name)

## Custom Embedding Functions[​](https://docs.trychroma.com/embeddings#custom-embedding-functions)

You can create your own embedding function to use with Chroma, it just needs to implement the **EmbeddingFunction** protocol.

from chromadb import Documents, EmbeddingFunction, Embeddings  
  
class MyEmbeddingFunction(EmbeddingFunction):  
 def \_\_call\_\_(self, texts: Documents) -> Embeddings:  
 # embed the documents somehow  
 return embeddings

We welcome contributions! If you create an embedding function that you think would be useful to others, please consider [**submitting a pull request**](https://github.com/chroma-core/chroma) to add it to Chroma's **embedding\_functions** module.

We welcome pull requests to add new Embedding Functions to the community.

# 👽 About

## Who we are[​](https://docs.trychroma.com/about#who-we-are)

WE ARE HIRING

We are hiring software engineers and applied research scientists.

[**➡️ View open roles on Notion**](https://www.notion.so/trychroma/careers-chroma-9d017c3007c7478ebd85bad854101497?pvs=4)

Chroma as a project is coordinated by a small team of full-time employees who work at a company also called Chroma.

We work in the sunny neighborhood of Potrero Hill in San Francisco.

Chroma is co-founded by [**Jeff Huber**](https://twitter.com/jeffreyhuber) (left) and [**Anton Troynikov**](https://twitter.com/atroyn) (right).

Pessoas em uma sala

Descrição gerada automaticamente

## Our commitment to open source[​](https://docs.trychroma.com/about#our-commitment-to-open-source)

Chroma is a company that builds the open-source project also called Chroma.

We are committed to building open source software because we believe in the flourishing of humanity that will be unlocked through the democratization of robust, safe, and aligned AI systems. These tools need to be available to a new developer just starting in ML as well as the organizations that scale ML to millions (and billions) of users. Open source is about expanding the horizon of what’s possible.

Chroma is a commercial open source company. What does that mean? We believe that organizing financially sustainable teams of people to work to manage, push and integrate the project enriches the health of the project and the community.

It is important that our values around this are very clear!

* We are committed to building Chroma as a ubiquitous open source standard
* A successful Chroma-based commercial product is essential for the success of the technology, and is a win-win for everyone. Simply put, many organizations will not adopt Chroma without the option of a commercially hosted solution; and the project must be backed by a company with a viable business model. We want to build an awesome project and an awesome business.
* We will decide what we provide exclusively in the commercial product based on clear, consistent criteria.

What code will be open source? As a general rule, any feature which an individual developer would find useful will be 100% open source forever. This approach, popularized by Gitlab, is called [**buyer-based open source**](https://about.gitlab.com/company/stewardship/). We believe that this is essential to accomplishing our mission.

Currently we don’t have any specific plans to monetize Chroma, we are working on a hosted service that will be launched as a free technical preview to make it easier for developers to get going. We are 100% focused on building valuable open source software with the community and for the community.

## Our investors[​](https://docs.trychroma.com/about#our-investors)

Chroma raised an $18M seed round led by Astasia Myers from Quiet Capital. Joining the round are angels including Naval Ravikant, Max and Jack Altman, Jordan Tigani (Motherduck), Guillermo Rauch (Vercel), Akshay Kothari (Notion), Amjad Masad (Replit), Spencer Kimball (CockroachDB), and other founders and leaders from ScienceIO, Gumroad, MongoDB, Scale, Hugging Face, Jasper and more.

Pessoas lado a lado

Descrição gerada automaticamente

Chroma raised a pre-seed in May 2022, led by Anthony Goldbloom (Kaggle) from AIX Ventures, James Cham from Bloomberg Beta, and Nat Friedman and Daniel Gross (AI Grant).

We're excited to work with a deep set of investors and enterpreneurs who have invested in and built some of the most successful open-source projects in the world.

# 📖 API Cheatsheet

NOTE

This is a quick cheatsheet of the API. For full API docs, refer to the JS and Python docs in the sidebar.

**Select a language**

* **Python**
* **JavaScript**

## Initialize client - Python[​](https://docs.trychroma.com/api-reference#initialize-client---python)

### In-memory chroma[​](https://docs.trychroma.com/api-reference#in-memory-chroma)

import chromadb  
client = chromadb.Client()

### In-memory chroma with saving/loading to disk[​](https://docs.trychroma.com/api-reference#in-memory-chroma-with-savingloading-to-disk)

In this mode, Chroma will persist data between sessions. On load - it will load up the data in the directory you specify. And as you add data - it will save to that directory.

import chromadb  
client = chromadb.PersistentClient(path="/path/to/data")

### Run chroma just as a client to talk to a backend service[​](https://docs.trychroma.com/api-reference#run-chroma-just-as-a-client-to-talk-to-a-backend-service)

For many use cases, an in-memory database will not cut it. Run **docker-compose up -d --build** to run a persistent backend in Docker. Simply update your API initialization and then use the API the same way as before.

import chromadb  
chroma\_client = chromadb.HttpClient(host="localhost", port=8000)

## Methods on Client[​](https://docs.trychroma.com/api-reference#methods-on-client)

### Methods related to Collections[​](https://docs.trychroma.com/api-reference#methods-related-to-collections)

COLLECTION NAMING

Collections are similar to AWS s3 buckets in their naming requirements because they are used in URLs in the REST API. Here's the [**full list**](https://docs.trychroma.com/usage-guide#creating-inspecting-and-deleting-collections).

# list all collections  
client.list\_collections()  
  
# make a new collection  
collection = client.create\_collection("testname")  
  
# get an existing collection  
collection = client.get\_collection("testname")  
  
# get a collection or create if it doesn't exist already  
collection = client.get\_or\_create\_collection("testname")  
  
# delete a collection  
client.delete\_collection("testname")

### Utility methods[​](https://docs.trychroma.com/api-reference#utility-methods)

# resets entire database - this \*cant\* be undone!  
client.reset()  
  
# returns timestamp to check if service is up  
client.heartbeat()

## Methods on Collection[​](https://docs.trychroma.com/api-reference#methods-on-collection)

# change the name or metadata on a collection  
collection.modify(name="testname2")  
  
# get the number of items in a collection  
collection.count()  
  
# add new items to a collection  
# either one at a time  
collection.add(  
 embeddings=[1.5, 2.9, 3.4],  
 metadatas={"uri": "img9.png", "style": "style1"},  
 documents="doc1000101",  
 ids="uri9",  
)  
# or many, up to 100k+!  
collection.add(  
 embeddings=[[1.5, 2.9, 3.4], [9.8, 2.3, 2.9]],  
 metadatas=[{"style": "style1"}, {"style": "style2"}],  
 ids=["uri9", "uri10"],  
)  
collection.add(  
 documents=["doc1000101", "doc288822"],  
 metadatas=[{"style": "style1"}, {"style": "style2"}],  
 ids=["uri9", "uri10"],  
)  
  
# update items in a collection  
collection.update()  
  
# upsert items. new items will be added, existing items will be updated.  
collection.upsert(  
 ids=["id1", "id2", "id3", ...],  
 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"}, {"chapter": "29", "verse": "11"}, ...],  
 documents=["doc1", "doc2", "doc3", ...],  
)  
  
# get items from a collection  
collection.get()  
  
# convenience, get first 5 items from a collection  
collection.peek()  
  
# do nearest neighbor search to find similar embeddings or documents, supports filtering  
collection.query(  
 query\_embeddings=[[1.1, 2.3, 3.2], [5.1, 4.3, 2.2]],  
 n\_results=2,  
 where={"style": "style2"}  
)  
  
# delete items  
collection.delete()

# 🔌 Integrations

**Select a language**

* **Python**
* **JavaScript**

## 🦜️🔗 Langchain[​](https://docs.trychroma.com/integrations" \l "%EF%B8%8F-langchain" \o "Direct link to 🦜️🔗 Langchain)

* [LangChain + Chroma](https://blog.langchain.dev/langchain-chroma/) on the LangChain blog
* [Harrison's **chroma-langchain** demo repo](https://github.com/hwchase17/chroma-langchain)
  + [question answering over documents](https://github.com/hwchase17/chroma-langchain/blob/master/qa.ipynb) - ([Replit version](https://replit.com/@swyx/LangChainChromaStarter" \l "main.py" \t "_blank))
  + [to use Chroma as a persistent database](https://github.com/hwchase17/chroma-langchain/blob/master/persistent-qa.ipynb)
* Tutorials
  + [Chroma and LangChain tutorial](https://github.com/grumpyp/chroma-langchain-tutorial) - The demo showcases how to pull data from the English Wikipedia using their API. The project also demonstrates how to vectorize data in chunks and get embeddings using OpenAI embeddings model.
  + [Create a Voice-based ChatGPT Clone That Can Search on the Internet and local files](https://betterprogramming.pub/how-to-create-a-voice-based-chatgpt-clone-that-can-search-on-the-internet-24d7f570ea8)
* [LangChain's Chroma Documentation](https://python.langchain.com/en/latest/modules/indexes/vectorstores.html?highlight=chroma#langchain.vectorstores.Chroma)

## 🦙 LlamaIndex[​](https://docs.trychroma.com/integrations" \l "-llamaindex" \o "Direct link to 🦙 LlamaIndex)

formerly known as GPT-index

* **LlamaIndex** [Vector Store page](https://gpt-index.readthedocs.io/en/latest/how_to/integrations/vector_stores.html)
* Demo: <https://github.com/jerryjliu/llama_index/blob/main/docs/examples/vector_stores/ChromaIndexDemo.ipynb>
* [Chroma Loader on Llamahub](https://llamahub.ai/l/chroma)

# ✈️ Migration

Schema and data format changes are a necessary evil of evolving software. We take changes seriously and make them infrequently and only when necessary.

Chroma's commitment is whenever schema or data format change, we will provide a seamless and easy-to-use migration tool to move to the new schema/format.

Specifically we will announce schema changes on:

* Discord ([#migrations channel](https://discord.com/channels/1073293645303795742/1129286514845691975))
* Github (here)
* Email listserv [Sign up](https://airtable.com/shrHaErIs1j9F97BE" \t "_blank)

We will aim to provide:

* a description of the change and the rationale for the change.
* a CLI migration tool you can run
* a video walkthrough of using the tool

## Migration Log[​](https://docs.trychroma.com/migration#migration-log)

### Migration from >0.4.0 to 0.4.0 - July 17, 2023[​](https://docs.trychroma.com/migration#migration-from-040-to-040---july-17-2023)

What's new in this version?

* New easy way to create clients
* Changed storage method
* **.persist()** removed, **.reset()** no longer on by default

**New Clients**

### in-memory ephemeral client  
  
# before  
import chromadb  
client = chromadb.Client()  
  
# after  
import chromadb  
client = chromadb.EphemeralClient()  
  
  
### persistent client  
  
# before  
import chromadb  
from chromadb.config import Settings  
client = chromadb.Client(Settings(  
 chroma\_db\_impl="duckdb+parquet",  
 persist\_directory="/path/to/persist/directory" # Optional, defaults to .chromadb/ in the current directory  
))  
  
# after  
import chromadb  
client = chromadb.PersistentClient(path="/path/to/persist/directory")  
  
  
### http client (to talk to server backend)  
  
# before  
import chromadb  
from chromadb.config import Settings  
client = chromadb.Client(Settings(chroma\_api\_impl="rest",  
 chroma\_server\_host="localhost",  
 chroma\_server\_http\_port="8000"  
 ))  
  
# after  
import chromadb  
client = chromadb.HttpClient(host="localhost", port="8000")

You can still also access the underlying **.Client()** method. If you want to turn off telemetry, all clients support custom settings:

import chromadb  
from chromadb.config import Settings  
client = chromadb.PersistentClient(  
 path="/path/to/persist/directory",   
 settings=Settings(anonymized\_telemetry=False))

**New data layout**

This version of Chroma drops **duckdb** and **clickhouse** in favor of **sqlite** for metadata storage. This means migrating data over. We have created a migration CLI utility to do this.

If you upgrade to **0.4.0** and try to access data stored in the old way, you will see this error message

*You are using a deprecated configuration of Chroma. Please pip install chroma-migrate and run****chroma-migrate****to upgrade your configuration. See*[***https://docs.trychroma.com/migration***](https://docs.trychroma.com/migration)*for more information or join our discord at*[***https://discord.gg/8g5FESbj***](https://discord.gg/8g5FESbj)*for help!*

Here is how to install and use the CLI:

pip install chroma-migrate  
chroma-migrate

Texto

Descrição gerada automaticamente

If you need any help with this migration, please reach out! We are on [**Discord**](https://discord.com/channels/1073293645303795742/1129286514845691975) ready to help.

**Persist & Reset**

**.persist()** was in the old version of Chroma because writes were only flushed when forced to. Chroma **0.4.0** saves all writes to disk instantly and so **persist** is no longer needed.

**.reset()**, which resets the entire database, used to by enabled-by-default which felt wrong. **0.4.0** has it disabled-by-default. You can enable it again by passing **allow\_reset=True** to a Settings object. For example:

import chromadb  
from chromadb.config import Settings  
client = chromadb.PersistentClient(path="./path/to/chroma", settings=Settings(allow\_reset=True))

# ☁️ Deployment

ALPHA STATUS

Chroma Server is currently in Alpha. We are working hard to move Chroma from a in-memory single-process oriented library to a distributed production-grade DB!

*  Alpha <- Currently
*  Technical Preview - ~1 month away, powered by a completely new backend
*  Full production
*  GA - General Availability

# ☁️ Deployment

You can also deploy Chroma on a long-running server, and connect to it remotely.

There are many possible configurations, but for convenience we have provided a very simple AWS CloudFormation template to experiment with deploying Chroma to EC2 on AWS.

## Hosted Chroma[​](https://docs.trychroma.com/deployment#hosted-chroma)

We want to offer hosted Chroma and we need your help.

Fill out the survey to jump the waitlist. Coming Q3 2023.

[**📝 30 second survey**](https://airtable.com/shrOAiDUtS2ILy5vZ)

## Simple AWS Deployment[​](https://docs.trychroma.com/deployment#simple-aws-deployment)

⚠️ Chroma and its underlying database need at least 2gb of RAM, which means it won't fit on the 1gb instances provided as part of the AWS Free Tier. This template uses a **t3.small** EC2 instance, which costs about two cents an hour, or $15 for a full month. If you follow these instructions, AWS will bill you accordingly.

⚠️ This basic stack doesn't support any kind of authentication; anyone who knows your server IP will be able to add and query for embeddings. To secure this endpoint, you'll need to put it behind [**AWS API Gateway**](https://aws.amazon.com/api-gateway/) or add your own authenticating proxy.

⚠️ By default, this template saves all data on a single volume. When you delete or replace it, the data will disappear. For serious production use (with high availability, backups, etc) please read and understand the CloudFormation template and use it as a basis for what you need, or reach out to the Chroma team for assistance.

### Step 1: Get an AWS Account[​](https://docs.trychroma.com/deployment#step-1-get-an-aws-account)

You will need an AWS Account. You can use one you already have, or [**create a new one**](https://aws.amazon.com/).

### Step 2: Get credentials[​](https://docs.trychroma.com/deployment#step-2-get-credentials)

For this example, we will be using the AWS command line interface. There are [**several ways**](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-prereqs.html) to configure the AWS CLI, but for the purposes of these examples we will presume that you have [**obtained an AWS access key**](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html) and will be using environment variables to configure AWS.

Export the **AWS\_ACCESS\_KEY\_ID** and **AWS\_SECRET\_ACCESS\_KEY** environment variables in your shell:

export AWS\_ACCESS\_KEY\_ID=\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
export AWS\_SECRET\_ACCESS\_KEY=\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

You can also configure AWS to use a region of your choice using the **AWS\_REGION** environment variable:

export AWS\_REGION=us-east-1

### Step 3: Run CloudFormation[​](https://docs.trychroma.com/deployment" \l "step-3-run-cloudformation" \o "Direct link to Step 3: Run CloudFormation)

Chroma publishes Cloudformation templates to S3 for each release.

To launch the template using AWS CloudFormation, run the following command line invocation:

aws cloudformation create-stack --stack-name my-chroma-stack --template-url https://s3.amazonaws.com/public.trychroma.com/cloudformation/latest/chroma.cf.json

Replace **--stack-name my-chroma-stack** with a different stack name, if you wish.

Wait a few minutes for the server to boot up, and Chroma will be available! You can get the public IP address of your new Chroma server using the AWS console, or using the following command:

aws cloudformation describe-stacks --stack-name my-chroma-stack --query 'Stacks[0].Outputs'

### Step 4: Customize the Stack (optional)[​](https://docs.trychroma.com/deployment#step-4-customize-the-stack-optional)

The CloudFormation template allows you to pass particular key/value pairs to override aspects of the stack. Available keys are:

* **InstanceType** - the AWS instance type to run (default: **t3.small**)
* **KeyName** - the AWS EC2 KeyPair to use, allowing to access the instance via SSH (default: none)

To set a CloudFormation stack's parameters using the AWS CLI, use the **--parameters** command line option. Parameters must be specified using the format **ParameterName={parameter},ParameterValue={value}**.

For example, the following command launches a new stack similar to the above, but on a **m5.4xlarge** EC2 instance, and adding a KeyPair named **mykey** so anyone with the associated private key can SSH into the machine:

aws cloudformation create-stack --stack-name my-chroma-stack --template-url https://s3.amazonaws.com/public.trychroma.com/cloudformation/latest/chroma.cf.json \  
 --parameters ParameterKey=KeyName,ParameterValue=mykey \  
 ParameterKey=InstanceType,ParameterValue=m5.4xlarge

### Step 5: Configure the Chroma Library[​](https://docs.trychroma.com/deployment#step-5-configure-the-chroma-library)

When you launch the Chroma client library to actually use Chroma, all you need to do is configure it to use the server's IP address and port **8000**. You can do this in two ways:

###### Using Environment Variables[​](https://docs.trychroma.com/deployment#using-environment-variables)

export CHROMA\_API\_IMPL=rest  
export CHROMA\_SERVER\_HOST=<server IP address>  
export CHROMA\_SERVER\_HTTP\_PORT=8000

###### In Code[​](https://docs.trychroma.com/deployment#in-code)

import chromadb  
from chromadb.config import Settings  
chroma = chromadb.HttpClient(host=<server IP address>, port=8000)

### Step 6: Clean Up (optional).[​](https://docs.trychroma.com/deployment#step-6-clean-up-optional)

To destroy the stack and remove all AWS resources, use the AWS CLI **delete-stack** command.

aws cloudformation delete-stack --stack-name my-chroma-stack

⚠️ This will destroy all the data in your Chroma database, unless you've taken a snapshot or otherwise backed it up.

### Troubleshooting[​](https://docs.trychroma.com/deployment#troubleshooting)

If you get an error saying **No default VPC for this user** when creating **ChromaInstanceSecurityGroup**, head to [**AWS VPC section**](https://us-east-1.console.aws.amazon.com/vpc/home?region=us-east-1#vpcs) and create deafault VPC for your user

# 📏 Telemetry

Chroma contains a telemetry feature that collects **anonymous** usage information.

## **Why?**[​](https://docs.trychroma.com/telemetry#why)

We use this information to help us understand how Chroma is used, to help us prioritize work on new features and bug fixes, and to help us improve Chroma’s performance and stability.

## **Opting out**[​](https://docs.trychroma.com/telemetry#opting-out)

Set **anonymized\_telemetry** in your clients settings to **false** to opt out of telemetry.

from chromadb.config import Settings  
client = chromadb.Client(Settings(anonymized\_telemetry=False))

## **What do you track?**[​](https://docs.trychroma.com/telemetry#what-do-you-track)

We will only track usage details that help us make product decisions, specifically:

* Chroma version and environment
* The Collection commands **add** or **delete**. We track the anonymized uuid of a collection as well as the number of items added or deleted.

We **do not** collect personally-identifiable or sensitive information, such as: usernames, hostnames, file names, environment variables, or hostnames of systems being tested.

To view the list of events we track, you may reference the [**code**](https://github.com/chroma-core/chroma/blob/main/chromadb/telemetry/events.py)

## **Where is telemetry information stored?**[​](https://docs.trychroma.com/telemetry#where-is-telemetry-information-stored)

We use **[Posthog](https://posthog.com/" \t "_blank)** to store and visualize telemetry data.

*Posthog is an open source platform for product analytics. Learn more about Posthog on*[***posthog.com***](https://posthog.com/)*or*[***github.com/posthog***](https://github.com/posthog/posthog)

# 🛣️ Roadmap

LAST UPDATED

**Jul 23, 2023**

The goal of this doc is to align core and community efforts for the project and to share what's in store for this year!

**Sections**

* What is the core Chroma team working on right now?
* What will Chroma prioritize over the next 6mo?
* What areas are great for community contributions?

## What is the core Chroma team working on right now?[​](https://docs.trychroma.com/roadmap#what-is-the-core-chroma-team-working-on-right-now)

* 🌩️ Standing up that distributed system as a managed service (aka "Hosted Chroma" - [sign up for waitlist](https://airtable.com/shrOAiDUtS2ILy5vZ)!)

## What did the Chroma team just complete?[​](https://docs.trychroma.com/roadmap#what-did-the-chroma-team-just-complete)

Features like:

* New - [Chroma 0.4](https://www.trychroma.com/blog/chroma_0.4.0) - our first production-oriented release
* 🐍 A more minimal python-client only build target
* ✋ Google PaLM embedding support
* 🎣 OpenAI ChatGPT Retrieval Plugin

## What will Chroma prioritize over the next 6mo?[​](https://docs.trychroma.com/roadmap#what-will-chroma-prioritize-over-the-next-6mo)

**Next Milestone: ☁️ Launch Hosted Chroma**

**Areas we will invest in**

Not an exhaustive list, but these are some of the core team’s biggest priorities over the coming few months. Use caution when contributing in these areas and please check-in with the core team first.

* ⏩ **Workflow**: Building tools for answer questions like: what embedding model should I use? And how should I chunk up my documents?
* 🌌 **Visualization**: Building visualization tool to give developers greater intuition embedding spaces
* 🔀 **Query Planner**: Building tools to enable per-query and post-query transforms
* 🔧 **Developer experience**: Extending Chroma into a CLI
* 📦 **Easier Data Sharing**: Working on formats for serialization and easier data sharing of embedding Collections
* 🔍 **Improving recall**: Fine-tuning embedding transforms through human feedback
* 🧠 **Analytical horsepower**: Clustering, deduplication, classification and more

## What areas are great for community contributions?[​](https://docs.trychroma.com/roadmap#what-areas-are-great-for-community-contributions)

This is where you have a lot more free rein to contribute (without having to sync with us first)!

If you're unsure about your contribution idea, feel free to chat with us (@chroma) in the **#general** channel in [**our Discord**](https://discord.gg/rahcMUU5XV)! We'd love to support you however we can.

### ⚙️ Example Templates[​](https://docs.trychroma.com/roadmap#%EF%B8%8F-example-templates)

We can always use [**more integrations**](https://docs.trychroma.com/integrations) with the rest of the AI ecosystem. Please let us know if you're working on one and need help!

Other great starting points for Chroma (please send PRs for more [**here**](https://github.com/chroma-core/docs/tree/swyx/addRoadmap/docs)):

* [Google Colab](https://colab.research.google.com/drive/1QEzFyqnoFxq7LUGyP1vzR4iLt9PpCDXv?usp=sharing)
* [Replit Template](https://replit.com/@swyx/BasicChromaStarter?v=1)

For those integrations we do have, like **LangChain** and **LlamaIndex**, we do always want more tutorials, demos, workshops, videos, and podcasts (we've done some pods [**on our blog**](https://trychroma.com/blog)).

### 📦 Example Datasets[​](https://docs.trychroma.com/roadmap#-example-datasets)

It doesn’t make sense for developers to embed the same information over and over again with the same embedding model.

We'd like suggestions for:

* "small" (<100 rows)
* "medium" (<5MB)
* "large" (>1GB)

datasets for people to stress test Chroma in a variety of scenarios.

### ⚖️ Embeddings Comparison[​](https://docs.trychroma.com/roadmap#%EF%B8%8F-embeddings-comparison)

Chroma does ship with Sentence Transformers by default for embeddings, but we are otherwise unopinionated about what embeddings you use. Having a library of information that has been embedded with many models, alongside example query sets would make it much easier for empirical work to be done on the effectiveness of various models across different domains.

* [Preliminary reading on Embeddings](https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526?gi=ee46baab0d8f)
* [Huggingface Benchmark of a bunch of Embeddings](https://huggingface.co/blog/mteb)
* [notable issues with GPT3 Embeddings](https://twitter.com/Nils_Reimers/status/1487014195568775173) and alternatives to consider

### ⚗️ Experimental Algorithms[​](https://docs.trychroma.com/roadmap#%EF%B8%8F-experimental-algorithms)

If you have a research background, please consider adding to our **ExperimentalAPI**s. For example:

* Projections (t-sne, UMAP, the new hotness, the one you just wrote) and Lightweight visualization
* Clustering (HDBSCAN, PCA)
* Deduplication
* Multimodal (CLIP)
* Fine-tuning manifold with human feedback [eg](https://github.com/openai/openai-cookbook/blob/main/examples/Customizing_embeddings.ipynb" \t "_blank)
* Expanded vector search (MMR, Polytope)
* Your research

### 🧑‍💻️ Additional Client SDKs[​](https://docs.trychroma.com/roadmap" \l "%EF%B8%8F-additional-client-sdks" \o "Direct link to 🧑‍💻️ Additional Client SDKs)

We will be happy to work with people maintaining additional client SDKs as part of the community. Specifically:

* Ruby
* Clojure
* Elixir

You can find the REST OpenAPI spec at **localhost:8000/openapi.json** when the backend is running.

Please [**reach out**](https://discord.gg/MMeYNTmh3x) and talk to us before you get too far in your projects so that we can offer technical guidance/align on roadmap.

# 🔍 Troubleshooting

This page is a list of common gotchas or issues and how to fix them.

If you don't see your problem listed here, please also search the **[Github Issues](https://github.com/chroma-core/chroma/issues" \t "_blank)**.

## Using .get or .query, embeddings say None[​](https://docs.trychroma.com/troubleshooting#using-get-or-query-embeddings-say-none)

This is actually not an error. Embeddings are quite large and heavy to send back. Most application don't use the underlying embeddings and so, by default, chroma does not send them back.

To send them back: add **include=["embeddings", "documents", "metadatas", "distances"]** to your query to return all information.

For example:

results = collection.query(  
 query\_texts="hello",  
 n\_results=1,  
 include=["embeddings", "documents", "metadatas", "distances"],  
)

NOTE

We may change **None** to something else to more clearly communicate why they were not returned.

## Your index resets back to just a few number of records[​](https://docs.trychroma.com/troubleshooting#your-index-resets-back-to-just-a-few-number-of-records)

Users report that they are using Chroma, happily adding data, and then they go to check the **count()** or **query()** and only a single item or a very small fraction of their data is in Chroma.

Chroma has 3 clients: **Ephemeral**, **Persistent**, and **Http** (coming soon). The **Ephemeral** and **Persistent** clients should treated as a **singleton**.

Here is what commonly happens.

1. Create a new Chroma client, #1 saving to **./db**, and add 10 items.
2. Create another new Chroma client, #2 saving to **./db**, and add 10 more items. Call this B.
3. Chroma does not lock the database between clients, each client maintains its own locking structure, so these clients can overwrite each other.

**Solution**: Don't use multiple **Ephemeral** or **Persistent** clients at the same time. Create one client and use it for all your operations.

NOTE

We may add extra logic to warn if multiple in-memory clients are used with the same path.

## Build error when running pip install chromadb[​](https://docs.trychroma.com/troubleshooting" \l "build-error-when-running-pip-install-chromadb" \o "Direct link to build-error-when-running-pip-install-chromadb)

If you encounter an error like this during setup

Failed to build hnswlib  
ERROR: Could not build wheels for hnswlib, which is required to install pyproject.toml-based projects

Try these few tips from the [**community**](https://github.com/chroma-core/chroma/issues/221):

1. If you get the error: **clang: error: the clang compiler does not support '-march=native'**, set this ENV variable, **export HNSWLIB\_NO\_NATIVE=1**
2. If on Mac, install/update xcode dev tools, **xcode-select --install**
3. If on Windows, try [these steps](https://github.com/chroma-core/chroma/issues/250#issuecomment-1540934224)

## SQLite[​](https://docs.trychroma.com/troubleshooting#sqlite)

Chroma requires SQLite > 3.35, if you encounter issues with having too low of a SQLite version please try the following.

1. Install the latest version of Python 3.10, sometimes lower versions of python are bundled with older versions of SQLite.
2. If you are on a Linux system, you can install pysqlite3-binary, **pip install pysqlite3-binary** and then override the default sqlite3 library before running Chroma with the steps [here](https://gist.github.com/defulmere/8b9695e415a44271061cc8e272f3c300). Alternatively you can compile SQLite from scratch and replace the library in your python installation with the latest version as documented [here](https://github.com/coleifer/pysqlite3#building-a-statically-linked-library).
3. If you are on Windows, you can manually download the latest version of SQLite from <https://www.sqlite.org/download.html> and replace the DLL in your python installation's DLLs folder with the latest version. You can find your python installation path by running **os.path.dirname(sys.executable)** in python.
4. If you are using a Debian based Docker container, older Debian versions do not have an up to date SQLite, please use **bookworm** or higher.

# Client

## EphemeralClient[​](https://docs.trychroma.com/reference/Client" \l "ephemeralclient" \o "Direct link to EphemeralClient)

def EphemeralClient(settings: Settings = Settings()) -> API

Creates an in-memory instance of Chroma. This is useful for testing and development, but not recommended for production use.

## PersistentClient[​](https://docs.trychroma.com/reference/Client" \l "persistentclient" \o "Direct link to PersistentClient)

def PersistentClient(path: str = "./chroma",  
 settings: Settings = Settings()) -> API

Creates a persistent instance of Chroma that saves to disk. This is useful for testing and development, but not recommended for production use.

**Arguments**:

* **path** - The directory to save Chroma's data to. Defaults to "./chroma".

## HttpClient[​](https://docs.trychroma.com/reference/Client" \l "httpclient" \o "Direct link to HttpClient)

def HttpClient(  
 host: str = "localhost",  
 port: str = "8000",  
 ssl: bool = False,  
 headers: Dict[str, str] = {},  
 settings: Settings = Settings()) -> API

Creates a client that connects to a remote Chroma server. This supports many clients connecting to the same server, and is the recommended way to use Chroma in production.

**Arguments**:

* **host** - The hostname of the Chroma server. Defaults to "localhost".
* **port** - The port of the Chroma server. Defaults to "8000".
* **ssl** - Whether to use SSL to connect to the Chroma server. Defaults to False.
* **headers** - A dictionary of headers to send to the Chroma server. Defaults to {}.

## Client[​](https://docs.trychroma.com/reference/Client" \l "client" \o "Direct link to Client)

def Client(settings: Settings = \_\_settings) -> API

Return a running chroma.API instance

# Client Methods

class API(Component, ABC)

## heartbeat[​](https://docs.trychroma.com/reference/Client#heartbeat)

def heartbeat() -> int

Get the current time in nanoseconds since epoch. Used to check if the server is alive.

**Returns**:

* **int** - The current time in nanoseconds since epoch

## list\_collections[​](https://docs.trychroma.com/reference/Client" \l "list_collections" \o "Direct link to list_collections)

def list\_collections() -> Sequence[Collection]

List all collections.

**Returns**:

* **Sequence[Collection]** - A list of collections

**Examples**:

client.list\_collections()  
# [collection(name="my\_collection", metadata={})]

## create\_collection[​](https://docs.trychroma.com/reference/Client" \l "create_collection" \o "Direct link to create_collection)

def create\_collection(name: str,  
 metadata: Optional[CollectionMetadata] = None,  
 embedding\_function: Optional[EmbeddingFunction] = ef.  
 DefaultEmbeddingFunction(),  
 get\_or\_create: bool = False) -> Collection

Create a new collection with the given name and metadata.

**Arguments**:

* **name** - The name of the collection to create.
* **metadata** - Optional metadata to associate with the collection.
* **embedding\_function** - Optional function to use to embed documents. Uses the default embedding function if not provided.
* **get\_or\_create** - If True, return the existing collection if it exists.

**Returns**:

* **Collection** - The newly created collection.

**Raises**:

* **ValueError** - If the collection already exists and get\_or\_create is False.
* **ValueError** - If the collection name is invalid.

**Examples**:

client.create\_collection("my\_collection")  
# collection(name="my\_collection", metadata={})  
  
client.create\_collection("my\_collection", metadata={"foo": "bar"})  
# collection(name="my\_collection", metadata={"foo": "bar"})

## get\_collection[​](https://docs.trychroma.com/reference/Client" \l "get_collection" \o "Direct link to get_collection)

def get\_collection(  
 name: str,  
 embedding\_function: Optional[EmbeddingFunction] = ef.  
 DefaultEmbeddingFunction()  
) -> Collection

Get a collection with the given name.

**Arguments**:

* **name** - The name of the collection to get
* **embedding\_function** - Optional function to use to embed documents. Uses the default embedding function if not provided.

**Returns**:

* **Collection** - The collection

**Raises**:

* **ValueError** - If the collection does not exist

**Examples**:

client.get\_collection("my\_collection")  
# collection(name="my\_collection", metadata={})

## get\_or\_create\_collection[​](https://docs.trychroma.com/reference/Client" \l "get_or_create_collection" \o "Direct link to get_or_create_collection)

def get\_or\_create\_collection(  
 name: str,  
 metadata: Optional[CollectionMetadata] = None,  
 embedding\_function: Optional[EmbeddingFunction] = ef.  
 DefaultEmbeddingFunction()  
) -> Collection

Get or create a collection with the given name and metadata.

**Arguments**:

* **name** - The name of the collection to get or create
* **metadata** - Optional metadata to associate with the collection
* **embedding\_function** - Optional function to use to embed documents

**Returns**:

The collection

**Examples**:

client.get\_or\_create\_collection("my\_collection")  
# collection(name="my\_collection", metadata={})

## delete\_collection[​](https://docs.trychroma.com/reference/Client" \l "delete_collection" \o "Direct link to delete_collection)

def delete\_collection(name: str) -> None

Delete a collection with the given name.

**Arguments**:

* **name** - The name of the collection to delete.

**Raises**:

* **ValueError** - If the collection does not exist.

**Examples**:

client.delete\_collection("my\_collection")

## reset[​](https://docs.trychroma.com/reference/Client#reset)

def reset() -> bool

Resets the database. This will delete all collections and entries.

**Returns**:

* **bool** - True if the database was reset successfully.

## get\_version[​](https://docs.trychroma.com/reference/Client" \l "get_version" \o "Direct link to get_version)

def get\_version() -> str

Get the version of Chroma.

**Returns**:

* **str** - The version of Chroma

## get\_settings[​](https://docs.trychroma.com/reference/Client" \l "get_settings" \o "Direct link to get_settings)

def get\_settings() -> Settings

Get the settings used to initialize the client.

**Returns**:

* **Settings** - The settings used to initialize the client.

**Collection Objects**

class Collection(BaseModel)

**count**[**​**](https://docs.trychroma.com/reference/Collection#count)

def count() -> int

The total number of embeddings added to the database

**Returns**:

* **int** - The total number of embeddings added to the database

**add**[**​**](https://docs.trychroma.com/reference/Collection#add)

def add(ids: OneOrMany[ID],  
 embeddings: Optional[OneOrMany[Embedding]] = None,  
 metadatas: Optional[OneOrMany[Metadata]] = None,  
 documents: Optional[OneOrMany[Document]] = None) -> None

Add embeddings to the data store.

**Arguments**:

* **ids** - The ids of the embeddings you wish to add
* **embeddings** - The embeddings to add. If None, embeddings will be computed based on the documents using the embedding\_function set for the Collection. Optional.
* **metadatas** - The metadata to associate with the embeddings. When querying, you can filter on this metadata. Optional.
* **documents** - The documents to associate with the embeddings. Optional.

**Returns**:

None

**Raises**:

* **ValueError** - If you don't provide either embeddings or documents
* **ValueError** - If the length of ids, embeddings, metadatas, or documents don't match
* **ValueError** - If you don't provide an embedding function and don't provide embeddings
* **ValueError** - If you provide both embeddings and documents
* **ValueError** - If you provide an id that already exists

**get**[**​**](https://docs.trychroma.com/reference/Collection#get)

def get(ids: Optional[OneOrMany[ID]] = None,  
 where: Optional[Where] = None,  
 limit: Optional[int] = None,  
 offset: Optional[int] = None,  
 where\_document: Optional[WhereDocument] = None,  
 include: Include = ["metadatas", "documents"]) -> GetResult

Get embeddings and their associate data from the data store. If no ids or where filter is provided returns all embeddings up to limit starting at offset.

**Arguments**:

* **ids** - The ids of the embeddings to get. Optional.
* **where** - A Where type dict used to filter results by. E.g. **{"color" : "red", "price": 4.20}**. Optional.
* **limit** - The number of documents to return. Optional.
* **offset** - The offset to start returning results from. Useful for paging results with limit. Optional.
* **where\_document** - A WhereDocument type dict used to filter by the documents. E.g. **{$contains: {"text": "hello"}}**. Optional.
* **include** - A list of what to include in the results. Can contain **"embeddings"**, **"metadatas"**, **"documents"**. Ids are always included. Defaults to **["metadatas", "documents"]**. Optional.

**Returns**:

* **GetResult** - A GetResult object containing the results.

**peek**[**​**](https://docs.trychroma.com/reference/Collection#peek)

def peek(limit: int = 10) -> GetResult

Get the first few results in the database up to limit

**Arguments**:

* **limit** - The number of results to return.

**Returns**:

* **GetResult** - A GetResult object containing the results.

**query**[**​**](https://docs.trychroma.com/reference/Collection#query)

def query(  
 query\_embeddings: Optional[OneOrMany[Embedding]] = None,  
 query\_texts: Optional[OneOrMany[Document]] = None,  
 n\_results: int = 10,  
 where: Optional[Where] = None,  
 where\_document: Optional[WhereDocument] = None,  
 include: Include = ["metadatas", "documents",  
 "distances"]) -> QueryResult

Get the n\_results nearest neighbor embeddings for provided query\_embeddings or query\_texts.

**Arguments**:

* **query\_embeddings** - The embeddings to get the closes neighbors of. Optional.
* **query\_texts** - The document texts to get the closes neighbors of. Optional.
* **n\_results** - The number of neighbors to return for each query\_embedding or query\_texts. Optional.
* **where** - A Where type dict used to filter results by. E.g. **{"color" : "red", "price": 4.20}**. Optional.
* **where\_document** - A WhereDocument type dict used to filter by the documents. E.g. **{$contains: {"text": "hello"}}**. Optional.
* **include** - A list of what to include in the results. Can contain **"embeddings"**, **"metadatas"**, **"documents"**, **"distances"**. Ids are always included. Defaults to **["metadatas", "documents", "distances"]**. Optional.

**Returns**:

* **QueryResult** - A QueryResult object containing the results.

**Raises**:

* **ValueError** - If you don't provide either query\_embeddings or query\_texts
* **ValueError** - If you provide both query\_embeddings and query\_texts

**modify**[**​**](https://docs.trychroma.com/reference/Collection#modify)

def modify(name: Optional[str] = None,  
 metadata: Optional[CollectionMetadata] = None) -> None

Modify the collection name or metadata

**Arguments**:

* **name** - The updated name for the collection. Optional.
* **metadata** - The updated metadata for the collection. Optional.

**Returns**:

None

**update**[**​**](https://docs.trychroma.com/reference/Collection#update)

def update(ids: OneOrMany[ID],  
 embeddings: Optional[OneOrMany[Embedding]] = None,  
 metadatas: Optional[OneOrMany[Metadata]] = None,  
 documents: Optional[OneOrMany[Document]] = None) -> None

Update the embeddings, metadatas or documents for provided ids.

**Arguments**:

* **ids** - The ids of the embeddings to update
* **embeddings** - The embeddings to add. If None, embeddings will be computed based on the documents using the embedding\_function set for the Collection. Optional.
* **metadatas** - The metadata to associate with the embeddings. When querying, you can filter on this metadata. Optional.
* **documents** - The documents to associate with the embeddings. Optional.

**Returns**:

None

**upsert[​](https://docs.trychroma.com/reference/Collection" \l "upsert" \o "Direct link to upsert)**

def upsert(ids: OneOrMany[ID],  
 embeddings: Optional[OneOrMany[Embedding]] = None,  
 metadatas: Optional[OneOrMany[Metadata]] = None,  
 documents: Optional[OneOrMany[Document]] = None) -> None

Update the embeddings, metadatas or documents for provided ids, or create them if they don't exist.

**Arguments**:

* **ids** - The ids of the embeddings to update
* **embeddings** - The embeddings to add. If None, embeddings will be computed based on the documents using the embedding\_function set for the Collection. Optional.
* **metadatas** - The metadata to associate with the embeddings. When querying, you can filter on this metadata. Optional.
* **documents** - The documents to associate with the embeddings. Optional.

**Returns**:

None

**delete**[**​**](https://docs.trychroma.com/reference/Collection#delete)

def delete(ids: Optional[IDs] = None,  
 where: Optional[Where] = None,  
 where\_document: Optional[WhereDocument] = None) -> None

Delete the embeddings based on ids and/or a where filter

**Arguments**:

* **ids** - The ids of the embeddings to delete
* **where** - A Where type dict used to filter the delection by. E.g. **{"color" : "red", "price": 4.20}**. Optional.
* **where\_document** - A WhereDocument type dict used to filter the deletion by the document content. E.g. **{$contains: {"text": "hello"}}**. Optional.

**Returns**:

None