

ChromaDB

Open-Source-Vektordatenbank

Einführung in ChromaDB

ChromaDB ist eine Open-Source-**Vektordatenbank** zur **Speicherung, Verwaltung und Suche von Embeddings**.

Sie ermöglicht semantische Suche, Filterung nach **Metadaten** und einfache Integration mit Python.

Ideal für **RAG**, Dokumentanalyse oder Chatbots mit Kontextspeicher.



Installation von ChromaDB

ChromaDB lässt sich einfach installieren:

```
pip install chromadb
```

Optional:

```
pip install chromadb[client]
```

für Netzwerkbetrieb.

Nach der Installation kann Chroma lokal im Speicher oder persistent auf der Festplatte laufen.

Erste Schritte mit ChromaDB

Beispiel: Sammlung "data" anlegen und Texte automatisch embedden und speichern.
ChromaDB nutzt den Default-Embedder [all-MiniLM-L6-v2](#), man kann aber auch [externe, bessere Embedder](#) nutzen (zb. OpenAI)

```
import chromadb

client = chromadb.Client()
collection = client.create_collection("data")

collection.add(
    ids=["1", "2"],
    documents=["Hi There", "Tannhauser Gate"],
)
```

<https://docs.trychroma.com/docs/embeddings/embedding-functions>

ChromaDB Clients: verschiedene Betriebsarten

ChromaDB bietet mehrere Client-Typen für unterschiedliche Einsatzzwecke:

PersistentClient: Speichert Daten **dauerhaft auf der Festplatte**. Ideal für lokale Projekte mit Persistenz.

Beispiel: `client = chromadb.PersistentClient(path="data/chroma")`

EphemeralClient: Arbeitet vollständig im **Speicher (RAM)** und verwirft Daten nach Beendigung. Perfekt für Tests oder temporäre Sessions (**default client**)

Beispiel: `client = chromadb.EphemeralClient()`

CloudClient: Verbindung zu einem **gehosteten Chroma-Dienst** (z. B. Chroma Cloud). Nützlich für Team- oder Produktionsumgebungen.

HttpClient: Kommuniziert über **HTTP mit einem laufenden Chroma-Server**.

ChromaDB HTTP-Client – empfohlen für Produktion

Der **HTTP-Client** ist die **beste Wahl für produktive Umgebungen**. Er ermöglicht eine klare Trennung zwischen Anwendung und Datenbank, unterstützt Skalierung, Containerisierung und sichere Netzwerkzugriffe.

Ein **Chroma-Server** läuft als zentraler Dienst (z. B. im **Docker-Container** oder Kubernetes-Pod), während mehrere Clients parallel darauf zugreifen können.

Vorteile:

Stabiler Betrieb ohne Datenverlust

Gemeinsame Nutzung durch mehrere Services

Einfache Integration in CI/CD und Cloud-Infrastruktur

Kompatibel mit Load Balancern, Firewalls und Authentifizierungsschichten

Arbeiten mit persistenter Speicherung

Standardmäßig läuft Chroma im Speicher (In-Memory). Für **persistente Speicherung**:

```
client = chromadb.PersistentClient(path="data/chroma_db")
```

Alle Vektoren, Texte und Metadaten werden dauerhaft gespeichert und stehen beim nächsten Start wieder zur Verfügung.

Metadaten in ChromaDB

Neben Texten und IDs können beliebige **Metadaten** gespeichert werden, z. B. Quelle, Kategorie oder Dokumenttyp.

```
collection.add(  
    ids=["doc1"],  
    documents=["KI verändert die  
Softwareentwicklung."],  
    metadatas=[{"source": "blog", "topic": "AI"}]  
)
```

Diese **Metadaten** können später für **Filter** genutzt werden. In der Wahl der **Schlüssel** ist man völlig frei.

Suchen mit ChromaDB

Semantische Suche basiert auf Embeddings. Beispiel:

```
results = collection.query(  
    query_texts=["Wie beeinflusst KI die Programmierung?"],  
    n_results=2  
)  
print(results["documents"])
```

Das Ergebnis liefert die **2 inhaltlich ähnlichsten Texte**, nicht nur exakte Worttreffer (**Similarity Search**)

Filtern nach Metadaten

Chroma erlaubt kombinierte **Filterung** nach Bedeutung und **Metadaten**:

```
results = collection.query(  
    query_texts=["Softwareentwicklung"],  
    where={"topic": "AI"}  
)
```

So lassen sich nur Dokumente aus einer bestimmten Quelle oder Kategorie durchsuchen. Diese Metadaten wurden beim Anlegen des Dokuments gespeichert (siehe Metadaten in ChromaDB).

Aktualisieren und Löschen von Einträgen

Daten lassen sich mit denselben IDs **updaten** oder **entfernen**:

```
collection.update(  
    ids=["doc1"],  
    documents=["Aktualisierter Text"])
```

```
collection.delete(ids=["doc2"])
```

Chroma verwaltet automatisch Embeddings und Metadaten konsistent.

Verwendung mit externen Embedding-Modellen

Chroma kann mit beliebigen Embedding-Providern kombiniert werden, z. B. OpenAI, Cohere oder SentenceTransformers.

```
from chromadb.utils.embedding_functions import OpenAIEmbeddingFunction

embedder = OpenAIEmbeddingFunction()
collection = client.create_collection("texte", embedding_function=embedder)
```

So werden Embeddings beim Hinzufügen automatisch berechnet.

Alternativen und Integration

ChromaDB ist leichtgewichtig und ideal für lokale Projekte. Alternativen für große Anwendungen sind [Milvus](#), [Weaviate](#), [Qdrant](#) oder [Pinecone](#).

[Qdrant](#) lässt sich auch selbst betreiben:
<https://github.com/qdrant/qdrant>

Chroma lässt sich einfach mit Frameworks wie [LangChain](#), [LlamaIndex](#) oder [FastAPI](#) integrieren , perfekt für semantische Suche und RAG-Systeme.