ChromaDB — Full Documentation (Beginner to Advanced)

---

1. Overview

ChromaDB is an open-source **vector database** designed for: - Semantic search - RAG (Retrieval Augmented Generation) - AI assistants with memory - Embedding storage & similarity search

ChromaDB stores **vectors**, **documents**, and **metadata**, enabling fast information retrieval based on meaning.

---

2. Why ChromaDB? (Problem → Solution)

❌Traditional Search Problem

- Only works with **keywords**
- Misses meaning
- Cannot use embeddings

✅ChromaDB Solution

- Stores embeddings (vectors)
- Performs **semantic search**
- Integrates with LLMs (OpenAI, HuggingFace, LangChain)
- Extremely fast, local, persistent

---

3. Installation

```
pip install chromadb
```

For persistence (local DB storage):

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

---

4. Project Folder Structure

```
my_rag_app/
│
├── data/
```

```
│   └── chroma/            # ChromaDB persistent storage
│
├── src/
│   ├── ingest.py          # Add documents + embed
│   ├── query.py           # Query the DB
│   └── utils.py           # Helpers
│
└── requirements.txt
```

---

5. Core API Flow (Explained)

```
Raw Text → Embedding → Stored in ChromaDB → Semantic Search → Relevant Output
```

5.1 Steps

1. **Initialize ChromaDB client**
2. **Create / Load a collection**
3. **Insert documents**
4. **Query with semantic search**
5. (Optional) **Update/Delete** data

---

6. Basic Usage Examples

6.1 Create / Load Database

```
import chromadb
client = chromadb.PersistentClient(path="data/chroma")
collection = client.get_or_create_collection("my_docs")
```

6.2 Add Documents

```
docs = [
    "Python is a programming language.",
    "Django is a Python web framework."
]

collection.add(
    ids=["1", "2"],
    documents=docs
)
```

6.3 Query Documents

```
result = collection.query(
    query_texts=["what is django"],
    n_results=2
)
print(result)
```

Returns: - Similar documents - Their distances - Their metadata (if present)

---

7. Metadata Support

Store extra information with each document.

```
collection.add(
    ids=["3"],
    documents=["Flask is a micro web framework."],
    metadatas=[{"category": "python"}]
)
```

Query with metadata filter:

```
collection.query(
    query_texts=["framework"],
    where={"category": "python"}
)
```

---

8. Updating and Deleting Data

Update

```
collection.update(
    ids=["1"],
    documents=["Python is a widely used programming language."]
)
```

Delete

```
collection.delete(ids=["2"])
```

---

9. Embedding Models Integration

You can plug in: - OpenAI embedding API - HuggingFace SentenceTransformers - LangChain embedding wrappers

Example (HuggingFace)

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')

vector = model.encode(["Hello world"])
```

---

10. ChromaDB + LangChain Flow

```
Documents → Embeddings → Chroma Vector Store → LangChain Retriever → LLM
```

Example

```
from langchain.vectorstores import Chroma
from langchain.embeddings import OpenAIEmbeddings

emb = OpenAIEmbeddings()
vector_db = Chroma(
    collection_name="my_docs",
    embedding_function=emb,
    persist_directory="data/chroma"
)
```

---

11. RAG Example (Full Flow)

```
query = "Explain Django in short"
retriever = vector_db.as_retriever()
context = retriever.get_relevant_documents(query)

print(context)
```

---

12. Best Practices

- Use **persistent mode** for production
- Use **metadata** for filtering

• Chunk long documents before embedding
• Use batch insertion for speed

---

13. Limitations

Limitation	Notes
Not a full DB	Only vector storage
Not distributed	But can scale locally
Requires embeddings separately	You choose the embedding model

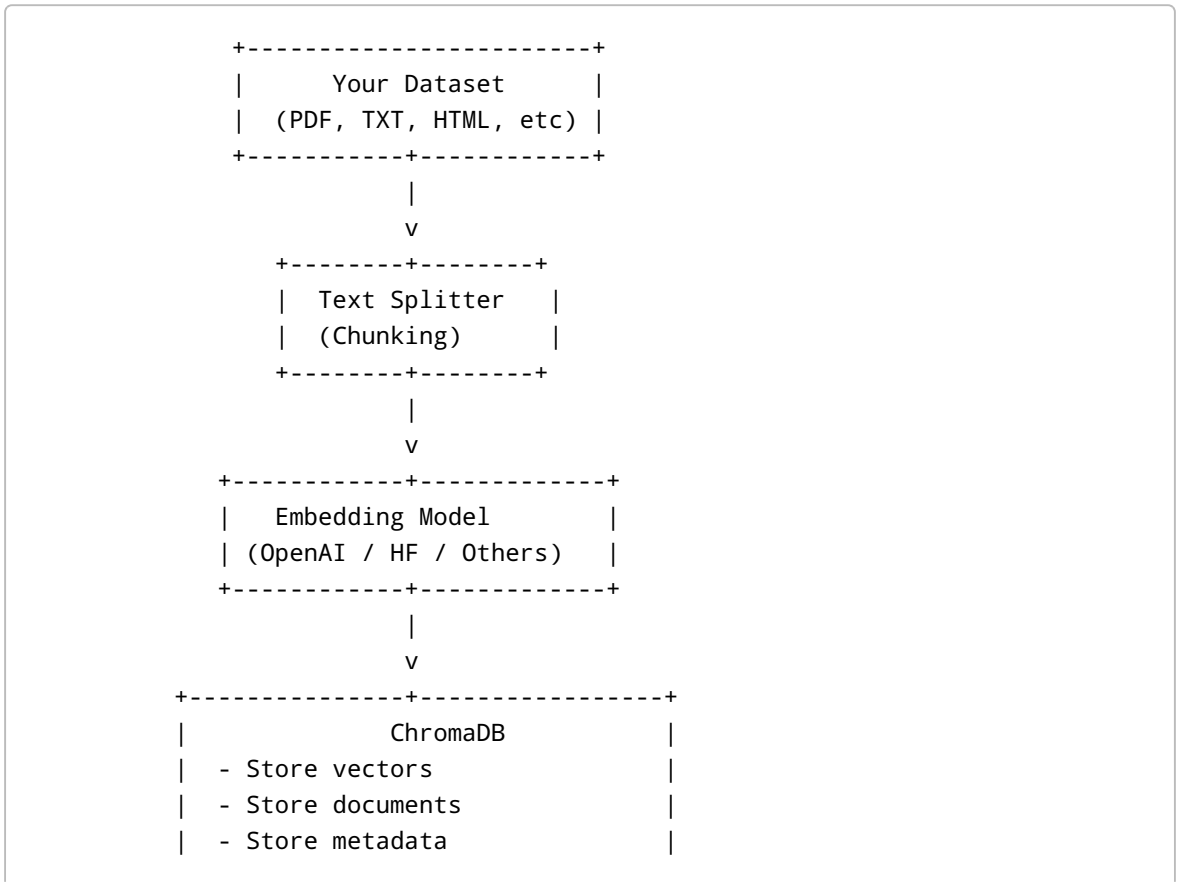
---

14. When to Use ChromaDB

• AI chatbots with memory
• Semantic search engines
• RAG systems
• Document Q&A
• Code search

---

15. Architecture Diagram (Text-Based)



```
|  - Semantic search              |
+---------------+-----------------+
                |
                v
       +--------+--------+
       |   Retriever     |
       | (Top-K Search)  |
       +--------+--------+
                |
                v
  +-------------+--------------+
  |          LLM Model         |
  |   (GPT, Claude, etc.)      |
  |  Combines query + context   |
  +-------------+--------------+
                |
                v
       +--------+--------+
       |  Final Response  |
       | (Answer Output)  |
       +------------------+
```

16. One-Page Summary

ChromaDB = Fast vector database for storing embeddings + semantic search. Perfect for RAG and LLM apps.

If you want, I can also add: - 💡 Architecture diagram - ⚙️Advanced performance tuning - 🗜️A complete RAG mini-project using ChromaDB - 📚PDF export of this documentation