**Quy trình xây dựng kho vector phục vụ cho RAG (Retrieval-Augmented Generation) bằng ChromaDB**

1. Chuyển PDF sang văn bản (text)
2. Chia nhỏ văn bản thành các đoạn (chunk)
3. Tạo embedding từ các đoạn
4. Lưu vào ChromaDB để có thể truy vấn sau này

mongodb+srv://grace:25112003@cluster0.nepogtj.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

my\_rag\_project/

│

├── BigData.pdf # File PDF chứa nội dung cần vector hóa

├── build\_vectorstore.py # Tạo Chroma vector store từ PDF

├── rag\_query.py # Truy vấn câu hỏi + lưu kết quả vào MongoDB

├── requirements.txt # Danh sách thư viện cần cài

└── chroma\_bigdata/ # Thư mục lưu vector store của Chroma

requirements.txt

langchain

chromadb

sentence-transformers

pypdf

transformers

torch

accelerate

langchain-community

pymongo

build\_vectorstore.py

from langchain.document\_loaders import PyPDFLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain.embeddings import SentenceTransformerEmbeddings

from langchain.vectorstores import Chroma

# Load file PDF

loader = PyPDFLoader("BigData.pdf")

pages = loader.load()

# Chia nhỏ văn bản

text\_splitter = RecursiveCharacterTextSplitter(

chunk\_size=500, # Độ dài đoạn văn bản tối đa

chunk\_overlap=50 # Trùng lặp giữa các đoạn văn

)

documents = text\_splitter.split\_documents(pages)

# Khởi tạo mô hình embedding

embedding\_function = SentenceTransformerEmbeddings(model\_name="all-MiniLM-L6-v2")

# Tạo Chroma vector store và lưu trữ

vectorstore = Chroma.from\_documents(

documents=documents,

embedding=embedding\_function,

persist\_directory="chroma\_bigdata"

)

vectorstore.persist()

print("✅ Vector store đã được tạo và lưu tại 'chroma\_bigdata'")

rag\_query.py

from langchain.vectorstores import Chroma

from langchain.embeddings import SentenceTransformerEmbeddings

from pymongo import MongoClient

from datetime import datetime

# Kết nối MongoDB Atlas (không cần bảo mật)

mongo\_uri = "mongodb+srv://grace:25112003@cluster0.nepogtj.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"

client = MongoClient(mongo\_uri)

db = client["rag\_project"]

collection = db["query\_logs"]

# Khởi tạo vector store

embedding\_function = SentenceTransformerEmbeddings(model\_name="all-MiniLM-L6-v2")

vectorstore = Chroma(

persist\_directory="chroma\_bigdata",

embedding\_function=embedding\_function

)

# Nhập câu hỏi và truy vấn vectorstore

query = input("🔍 Câu hỏi của bạn: ")

results = vectorstore.similarity\_search(query, k=3)

# Hiển thị kết quả và lưu vào MongoDB

print("\n📄 Các đoạn văn bản gần nhất:")

result\_texts = []

for i, doc in enumerate(results, 1):

print(f"\n--- Kết quả {i} ---\n{doc.page\_content}")

result\_texts.append(doc.page\_content)

# Lưu kết quả vào MongoDB

collection.insert\_one({

"query": query,

"results": result\_texts,

"timestamp": datetime.now()

})

print("\n✅ Kết quả đã được lưu vào MongoDB Atlas!")

Chạy 3 lệnh này theo thứ tự:

pip install -r requirements.txt  
python build\_vectorstore.py

python rag\_query.py

Cách chạy của tú uyên

**🧪 Cách chạy**

**#1 Tạo môi trường chạy có 2 cách ( ko được cách 1 thì làm cách 2)**

**C1:**

python -m venv rag\_env

source rag\_env/bin/activate # hoặc .\rag\_env\Scripts\activate trên Windows

**C2:**

pip install virtualenv

virtualenv rag\_env

.\rag\_env\Scripts\Activate.ps1

**# 2. Cài thư viện**

pip install -r requirements.txt

**# 3. Tạo kho vector**

python build\_vectorstore.py

**# 4. Truy vấn**

python rag\_query.py

**Lệnh cơ bản để thao tác với MongoDB Atlas từ Python**

**1. Cài đặt MongoDB Python Driver**

pip install pymongo

**2. Kết nối MongoDB Atlas**

from pymongo import MongoClient

# Thay thế bằng URI của bạn (dùng MongoDB URI của Atlas)

mongo\_uri = "mongodb+srv://username:password@cluster0.mongodb.net/?retryWrites=true&w=majority"

# Kết nối tới MongoDB Atlas

client = MongoClient(mongo\_uri)

# Kết nối tới database cụ thể

db = client["your\_database\_name"]

# Kết nối tới collection cụ thể

collection = db["your\_collection\_name"]

**3. Thêm Document (Insert)**

# Thêm một document

collection.insert\_one({"name": "Alice", "age": 25})

# Thêm nhiều document

data = [{"name": "Bob", "age": 30}, {"name": "Charlie", "age": 35}]

collection.insert\_many(data)

**4. Tìm Document (Find)**

# Tìm tất cả document trong collection

all\_docs = collection.find()

# Tìm document theo điều kiện (ví dụ: tìm document có tên là "Alice")

query = {"name": "Alice"}

docs = collection.find(query)

# In ra các kết quả

for doc in docs:

print(doc)

**5. Cập nhật Document (Update)**

# Cập nhật một document (cập nhật tuổi của Alice)

query = {"name": "Alice"}

new\_values = {"$set": {"age": 26}}

collection.update\_one(query, new\_values)

# Cập nhật nhiều document (cập nhật tuổi của tất cả các người có tên "Bob")

query = {"name": "Bob"}

new\_values = {"$set": {"age": 31}}

collection.update\_many(query, new\_values)

**6. Xóa Document (Delete)**

# Xóa một document

query = {"name": "Alice"}

collection.delete\_one(query)

# Xóa nhiều document

query = {"age": {"$gt": 30}} # Xóa tất cả document có tuổi lớn hơn 30

collection.delete\_many(query)

**7. Tìm và giới hạn kết quả (Find with limit)**

# Tìm 3 document đầu tiên

docs = collection.find().limit(3)

for doc in docs:

print(doc)

**8. Sắp xếp kết quả (Sorting)**

# Sắp xếp theo tuổi (ascending)

docs = collection.find().sort("age", 1) # 1: ascending, -1: descending

for doc in docs:

print(doc)

**9. Chỉ định các trường cần truy vấn (Projection)**

# Lấy các trường 'name' và 'age', không lấy các trường khác

docs = collection.find({}, {"\_id": 0, "name": 1, "age": 1})

for doc in docs:

print(doc)

**10. Đếm số lượng document (Count)**

# Đếm tất cả các document

count = collection.count\_documents({})

# Đếm document có điều kiện

query = {"age": {"$gt": 30}}

count = collection.count\_documents(query)

print(f"Số lượng document: {count}")

**11. Tạo chỉ mục (Index)**

# Tạo chỉ mục cho trường 'name'

collection.create\_index([("name", 1)]) # 1: ascending, -1: descending

**12. Kết nối và đóng kết nối**

# Đóng kết nối (tùy chọn, pymongo tự động đóng khi script kết thúc)

client.close()

**Ví dụ đầy đủ về thao tác với MongoDB Atlas:**

from pymongo import MongoClient

# Kết nối MongoDB Atlas

mongo\_uri = "mongodb+srv://username:password@cluster0.mongodb.net/?retryWrites=true&w=majority"

client = MongoClient(mongo\_uri)

# Kết nối tới database và collection

db = client["rag\_project"]

collection = db["query\_logs"]

# Thêm document vào MongoDB

query = {"query": "Ứng dụng của dữ liệu lớn trong thống kê nhà nước là gì?"}

results = ["Câu trả lời 1", "Câu trả lời 2", "Câu trả lời 3"]

collection.insert\_one({

"query": query["query"],

"results": results,

"timestamp": datetime.now()

})

# Tìm và in ra kết quả

docs = collection.find({"query": "Ứng dụng của dữ liệu lớn trong thống kê nhà nước là gì?"})

for doc in docs:

print(doc)

# Đóng kết nối

client.close()

**LÝ THUYẾT MONGODB**

**1. Kiến Thức Cơ Bản**

* **MongoDB**: Là cơ sở dữ liệu NoSQL, lưu trữ dữ liệu dưới dạng tài liệu JSON/BSON. Dễ dàng thay đổi cấu trúc dữ liệu và mở rộng quy mô.
* **MongoDB Atlas**: Dịch vụ quản lý MongoDB trên đám mây, hỗ trợ các nền tảng như AWS, Google Cloud, Azure.
* **Cấu trúc**: Bao gồm Database, Collection và Document.
* **Thao tác cơ bản**:
  + **Insert**: Thêm document.
  + **Find**: Truy vấn dữ liệu.
  + **Update/Delete**: Cập nhật/xóa document.
  + **Indexing**: Tạo chỉ mục để tối ưu truy vấn.
  + **Aggregation**: Phân tích và xử lý dữ liệu phức tạp.

**2. Kiến Thức Nâng Cao**

* **Replication**: Tạo replica sets để đảm bảo khả năng sẵn sàng cao.
* **Sharding**: Phân mảnh dữ liệu trên nhiều server để tăng khả năng mở rộng.
* **Aggregation Pipeline**: Dùng để xử lý dữ liệu phức tạp (tương tự SQL GROUP BY).
* **Indexing**: Tăng tốc truy vấn bằng cách tạo chỉ mục trên các trường thường xuyên truy vấn.
* **Security**:
  + **Authentication**: Xác thực người dùng.
  + **Authorization**: Cấp quyền truy cập.
  + **Encryption**: Mã hóa dữ liệu khi lưu và truyền tải.
* **Backup & Restore**: Sao lưu tự động và phục hồi dữ liệu.
* **Performance Monitoring**: Giám sát hiệu suất cơ sở dữ liệu.
* **Data Lake**: Quản lý và truy vấn dữ liệu từ nhiều nguồn khác nhau.
* **Triggers & Change Streams**: Tự động phản hồi thay đổi trong cơ sở dữ liệu.

**3. Kiến Thức Mở Rộng**

* **MongoDB Charts**: Công cụ phân tích và trực quan hóa dữ liệu trong MongoDB. Bạn có thể tạo các biểu đồ, báo cáo trực tiếp từ cơ sở dữ liệu mà không cần phải dùng công cụ bên ngoài.
* **Data Migration**: MongoDB Atlas hỗ trợ di chuyển dữ liệu từ các hệ quản trị cơ sở dữ liệu khác (MySQL, PostgreSQL, SQL Server) vào MongoDB thông qua các công cụ như **MongoDB Atlas Data Migration Tool** hoặc **MongoDB Compass**.
* **Multi-Region Clusters**: Tạo cụm cơ sở dữ liệu phân phối trên nhiều khu vực địa lý để giảm độ trễ và nâng cao khả năng chịu lỗi.
* **Global Clusters**: Hỗ trợ các ứng dụng toàn cầu với dữ liệu được phân phối qua nhiều khu vực địa lý. Giúp tối ưu hóa độ trễ và tăng tốc truy vấn dữ liệu cho người dùng ở các vùng khác nhau.
* **Serverless Instances**: MongoDB Atlas cung cấp tùy chọn serverless cho các ứng dụng không yêu cầu tài nguyên cố định, giúp giảm chi phí khi tải lên và tải xuống dữ liệu ít thay đổi.

**4. Quản lý Người Dùng và Quyền Truy Cập**

* **Roles**: MongoDB Atlas cho phép tạo các vai trò (roles) tùy chỉnh để kiểm soát quyền truy cập dữ liệu, với các quyền như đọc, ghi, quản lý cấu hình, sao lưu và phục hồi.
* **IP Whitelisting**: Chỉ cho phép các địa chỉ IP cụ thể truy cập vào cơ sở dữ liệu để bảo mật.
* **MFA (Multi-Factor Authentication)**: Bảo mật tài khoản bằng cách yêu cầu xác thực hai yếu tố (2FA) khi đăng nhập vào MongoDB Atlas.

**5. Tối Ưu Hóa Hiệu Suất**

* **Automatic Scaling**: MongoDB Atlas có thể tự động mở rộng hoặc thu hẹp kích thước cơ sở dữ liệu và các tài nguyên máy chủ khi có nhu cầu.
* **Performance Advisor**: Công cụ giúp phát hiện các vấn đề về hiệu suất và đề xuất các chỉ mục tối ưu hóa.
* **Slow Query Profiler**: Công cụ giúp theo dõi và tối ưu các truy vấn chậm, cải thiện hiệu suất của ứng dụng.

**6. Tích Hợp với Các Công Cụ Khác**

* **MongoDB Realm**: Công cụ phát triển ứng dụng di động và web có khả năng đồng bộ hóa dữ liệu giữa các thiết bị và MongoDB Atlas, thích hợp cho các ứng dụng cần dữ liệu đồng bộ.
* **AWS Lambda, Azure Functions**: Tích hợp MongoDB Atlas với các dịch vụ không máy chủ (serverless) như AWS Lambda, Azure Functions để xử lý sự kiện hoặc tác vụ dựa trên cơ sở dữ liệu.
* **Webhooks**: Sử dụng webhooks để tự động gửi thông báo hoặc phản hồi một sự kiện khi có thay đổi trong cơ sở dữ liệu.

**7. Phân Tích Dữ Liệu và Truy Vấn**

* **Aggregation Framework**: MongoDB hỗ trợ bộ công cụ mạnh mẽ để phân tích dữ liệu. Aggregation Pipeline giúp xử lý dữ liệu qua nhiều giai đoạn như lọc, nhóm, sắp xếp và tính toán phức tạp.
* **Full-text Search**: MongoDB Atlas hỗ trợ tìm kiếm toàn văn bản (full-text search) với tính năng tích hợp từ **MongoDB Atlas Search**, giúp tìm kiếm dữ liệu nhanh chóng và hiệu quả trong các trường văn bản.

**8. Quản Lý Backup và Phục Hồi Dữ Liệu**

* **Automated Backups**: MongoDB Atlas tự động sao lưu dữ liệu định kỳ. Sao lưu có thể được lưu trữ trong các đám mây và có thể phục hồi bất kỳ lúc nào.
* **Point-in-time Restore**: Có khả năng phục hồi dữ liệu từ một điểm thời gian cụ thể (tính năng phục hồi theo thời gian thực).
* **Continuous Backup**: Các bản sao lưu liên tục giúp theo dõi mọi thay đổi trong cơ sở dữ liệu và cho phép khôi phục dữ liệu ngay cả khi đã xóa hoặc cập nhật.

**9. Security và Compliance**

* **Encryption at Rest and in Transit**: MongoDB Atlas mã hóa dữ liệu cả khi lưu trữ và khi truyền tải, bảo vệ dữ liệu khỏi các cuộc tấn công.
* **VPC Peering**: Atlas hỗ trợ kết nối với các mạng ảo (VPC) của bạn trong các dịch vụ đám mây như AWS, Azure và Google Cloud, giúp tăng cường bảo mật và giảm độ trễ.
* **Compliance**: MongoDB Atlas tuân thủ nhiều tiêu chuẩn bảo mật và tuân thủ quốc tế như **GDPR**, **HIPAA**, **SOC 2**, **ISO 27001**, và nhiều tiêu chuẩn khác.

**10. Cộng Đồng và Tài Liệu**

* **MongoDB Community**: MongoDB Atlas có một cộng đồng người dùng rộng lớn, với tài liệu, hướng dẫn và ví dụ phong phú từ cả MongoDB và cộng đồng bên ngoài.
* **MongoDB University**: Cung cấp các khóa học miễn phí giúp người dùng học cách sử dụng MongoDB Atlas một cách hiệu quả.

**LÝ THUYẾT MONGODB**

**1. Giới Thiệu về ChromaDB**

* **ChromaDB** là một cơ sở dữ liệu vector được thiết kế để lưu trữ và truy vấn dữ liệu vector trong các ứng dụng học máy và tìm kiếm thông minh, đặc biệt hữu ích trong các ứng dụng xử lý ngôn ngữ tự nhiên (NLP) và trí tuệ nhân tạo (AI).
* ChromaDB lưu trữ dữ liệu dưới dạng các vector, với các tính năng như tìm kiếm tương đồng (similarity search) và tìm kiếm gần đúng (approximate nearest neighbor search - ANN).

**2. Kiến Thức Cơ Bản**

* **Vector Stores**: ChromaDB sử dụng các vector để biểu diễn thông tin. Một vector có thể là sự chuyển đổi của một đoạn văn bản, hình ảnh, hoặc bất kỳ dạng dữ liệu nào có thể chuyển thành số.
* **Similarity Search**: Chroma hỗ trợ truy vấn tương đồng, giúp tìm kiếm các vector có sự gần gũi cao nhất với vector truy vấn, được sử dụng rộng rãi trong các ứng dụng tìm kiếm văn bản, tìm kiếm hình ảnh, v.v.
* **Indexing**: Chroma sử dụng các thuật toán chỉ mục (indexing) hiệu quả như **HNSW** (Hierarchical Navigable Small World) để tối ưu hóa việc tìm kiếm trong các cơ sở dữ liệu vector lớn.

**3. Tính Năng Nổi Bật**

* **Kết Hợp với LangChain**: Chroma tích hợp mạnh mẽ với **LangChain** để xử lý các văn bản và tạo các ứng dụng trí tuệ nhân tạo, chẳng hạn như **RAG** (Retrieval-Augmented Generation).
* **Persistence**: Dữ liệu được lưu trữ trong Chroma có thể được lưu vĩnh viễn vào hệ thống file hoặc cơ sở dữ liệu để duy trì trạng thái qua các lần khởi động lại.
* **Custom Embedding Models**: Chroma hỗ trợ tích hợp các mô hình nhúng (embedding models) tùy chỉnh, chẳng hạn như các mô hình từ **Sentence Transformers** hoặc **OpenAI embeddings**.

**4. Kiến Thức Nâng Cao**

* **Tối Ưu Hóa Truy Vấn**: ChromaDB sử dụng các kỹ thuật tối ưu như **approximate nearest neighbor search (ANN)** để giảm độ phức tạp tính toán và tăng tốc quá trình tìm kiếm tương đồng. Việc sử dụng các thuật toán như **HNSW** hoặc **IVF** giúp tối ưu hóa truy vấn với tập dữ liệu lớn.
* **Scalability**: ChromaDB có khả năng mở rộng để làm việc với các tập dữ liệu rất lớn mà không làm giảm hiệu suất tìm kiếm. Các kỹ thuật như sharding (chia nhỏ dữ liệu) và parallelization (xử lý song song) giúp xử lý lượng lớn dữ liệu.
* **Distributed Computing**: Chroma hỗ trợ việc triển khai phân tán (distributed), cho phép triển khai trên nhiều máy chủ hoặc cụm máy chủ để phục vụ các ứng dụng lớn và yêu cầu cao về khả năng mở rộng.
* **Embedding Optimization**: Bạn có thể sử dụng các mô hình học sâu (deep learning models) hoặc các mô hình nhúng như **Sentence-Transformer** để tối ưu hóa việc tạo ra các vector có chất lượng cao, giúp cải thiện khả năng tìm kiếm và độ chính xác của kết quả.

**5. Tích Hợp và API**

* **LangChain Integration**: ChromaDB tích hợp tốt với **LangChain**, giúp xây dựng các ứng dụng AI, như hệ thống truy vấn ngữ nghĩa (semantic search) hoặc các ứng dụng tổng hợp thông tin tự động.
* **API**: ChromaDB cung cấp API đơn giản để bạn có thể dễ dàng thêm, xóa, hoặc truy vấn dữ liệu vector, giúp tích hợp vào các hệ thống hoặc dịch vụ khác.

**6. Tối Ưu Hóa và Quản Lý**

* **Indexing and Retrieving**: Việc tạo chỉ mục (indexing) dữ liệu là cực kỳ quan trọng trong ChromaDB để tối ưu hóa tốc độ truy vấn. Cần lựa chọn chỉ mục sao cho phù hợp với kích thước dữ liệu và mục đích sử dụng.
* **Metadata**: ChromaDB hỗ trợ thêm các dữ liệu bổ sung (metadata) vào mỗi vector để truy vấn không chỉ dựa trên giá trị của vector mà còn dựa vào các thông tin khác như thời gian, tác giả, loại nội dung, v.v.
* **Vector Persistence**: Chroma hỗ trợ lưu trữ và khôi phục dữ liệu vector để các ứng dụng có thể tiếp tục từ nơi đã dừng lại mà không mất dữ liệu trước đó. Điều này cực kỳ quan trọng trong các hệ thống yêu cầu tính liên tục và đáng tin cậy.

**7. Bảo Mật và Quản Lý Quyền Truy Cập**

* **Access Control**: Các cơ sở dữ liệu vector trong Chroma có thể được bảo vệ và phân quyền truy cập bằng các cơ chế như API keys hoặc các lớp bảo mật tích hợp.
* **Data Privacy**: Khi sử dụng Chroma trong các ứng dụng có yêu cầu bảo mật cao (như hệ thống y tế hoặc tài chính), việc bảo vệ dữ liệu là cực kỳ quan trọng. Chroma hỗ trợ mã hóa dữ liệu khi lưu trữ và truyền tải.

**8. Ứng Dụng Thực Tế**

* **Search Engines**: Chroma được sử dụng rộng rãi trong các công cụ tìm kiếm thông minh, nơi mà các câu truy vấn có thể được so sánh với các văn bản trong cơ sở dữ liệu để tìm ra các kết quả gần đúng.
* **Recommendation Systems**: Chroma có thể được tích hợp vào các hệ thống gợi ý, nơi các vector mô tả hành vi hoặc sở thích của người dùng có thể giúp đề xuất sản phẩm hoặc nội dung tương tự.
* **Question Answering**: Các hệ thống trả lời câu hỏi (QA) sử dụng Chroma để tìm kiếm thông tin liên quan từ các tài liệu lớn và trả lời câu hỏi của người dùng một cách chính xác và nhanh chóng.

**9. Lợi Ích của ChromaDB**

* **Hiệu Suất Cao**: Chroma tối ưu hóa việc tìm kiếm tương đồng và giúp giảm thiểu độ trễ trong các ứng dụng AI và NLP.
* **Dễ Dàng Tích Hợp**: Với sự hỗ trợ API mạnh mẽ, Chroma dễ dàng tích hợp với các framework và hệ thống khác, từ LangChain đến các ứng dụng tùy chỉnh.
* **Mở Rộng và Linh Hoạt**: Chroma cho phép triển khai phân tán và mở rộng quy mô, rất phù hợp với các ứng dụng yêu cầu dung lượng dữ liệu lớn.

**10. Cộng Đồng và Tài Liệu**

* **Documentation**: ChromaDB có tài liệu chi tiết và hướng dẫn sử dụng với các ví dụ cụ thể, giúp người dùng dễ dàng làm quen với cách sử dụng và tích hợp.
* **Community Support**: Cộng đồng sử dụng Chroma đang phát triển và bạn có thể tìm thấy sự hỗ trợ từ các diễn đàn, nhóm thảo luận hoặc trên GitHub.

LỆNH THƯỜNG SỬ DỤNG CHROMADB

**2. Khởi Tạo Chroma Vector Store**

from langchain.vectorstores import Chroma

from langchain.embeddings import SentenceTransformerEmbeddings

from langchain.document\_loaders import PyPDFLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

# Load dữ liệu từ PDF

loader = PyPDFLoader("BigData.pdf")

pages = loader.load()

# Chia nhỏ văn bản thành các đoạn

text\_splitter = RecursiveCharacterTextSplitter(

chunk\_size=500,

chunk\_overlap=50

)

documents = text\_splitter.split\_documents(pages)

# Khởi tạo embedding model

embedding\_function = SentenceTransformerEmbeddings(model\_name="all-MiniLM-L6-v2")

# Tạo vector store và lưu trữ

vectorstore = Chroma.from\_documents(

documents=documents,

embedding=embedding\_function,

persist\_directory="chroma\_bigdata"

)

# Lưu vector store

vectorstore.persist()

**3. Truy Vấn Dữ Liệu Từ Chroma Vector Store**

from langchain.vectorstores import Chroma

from langchain.embeddings import SentenceTransformerEmbeddings

# Load lại vector store từ thư mục đã lưu

embedding\_function = SentenceTransformerEmbeddings(model\_name="all-MiniLM-L6-v2")

vectorstore = Chroma(

persist\_directory="chroma\_bigdata",

embedding\_function=embedding\_function

)

# Truy vấn với câu hỏi

query = "Ứng dụng của dữ liệu lớn trong thống kê nhà nước là gì?"

results = vectorstore.similarity\_search(query, k=3)

# In kết quả

for i, doc in enumerate(results, 1):

print(f"--- Kết quả {i} ---\n{doc.page\_content}")

**4. Cập Nhật Dữ Liệu Vector**

# Thêm tài liệu mới vào vector store

new\_documents = [...] # Cần có dữ liệu mới (có thể là văn bản hoặc các đối tượng Document khác)

vectorstore.add\_documents(new\_documents)

vectorstore.persist() # Lưu lại vector store

**5. Xóa Dữ Liệu**

# Xóa dữ liệu (có thể xóa theo ID hoặc các phương thức liên quan)

vectorstore.delete\_documents([document\_id])

vectorstore.persist()

**6. Khôi Phục Vector Store**

vectorstore = Chroma(

persist\_directory="chroma\_bigdata",

embedding\_function=embedding\_function

)

**7. Tạo Metadata cho Mỗi Vector**

documents\_with\_metadata = [

{"document": doc, "metadata": {"source": "BigData.pdf", "page": 1}}

for doc in documents

]

# Thêm vào vector store

vectorstore.add\_documents(documents\_with\_metadata)

**8. Sử Dụng Approximate Nearest Neighbors (ANN)**

# Cấu hình ANN khi khởi tạo vector store

vectorstore = Chroma(

persist\_directory="chroma\_bigdata",

embedding\_function=embedding\_function,

index\_type="hnsw"

)

**9. Chạy Tìm Kiếm Với Metadata**

# Truy vấn với metadata

results = vectorstore.similarity\_search(query, k=3, filter={"source": "BigData.pdf"})

**10. Lưu Trữ và Quản Lý Dữ Liệu**

# Persist dữ liệu

vectorstore.persist()

# Thay đổi thư mục lưu trữ

vectorstore.persist\_directory = "new\_directory"

**11. Tạo và Quản Lý Chỉ Mục**

# Khởi tạo chỉ mục và chỉ định loại chỉ mục

vectorstore = Chroma(

persist\_directory="chroma\_bigdata",

embedding\_function=embedding\_function,

index\_type="ivf"

)

**12. Sử Dụng Chroma với LangChain**

from langchain.chains import RetrievalQA

from langchain.llms import OpenAI

# Tạo chain hỏi-đáp sử dụng vector store Chroma

qa\_chain = RetrievalQA.from\_chain\_type(

llm=OpenAI(),

chain\_type="map\_reduce",

retriever=vectorstore.as\_retriever()

)

# Câu hỏi và trả lời

query = "Thế nào là dữ liệu lớn?"

answer = qa\_chain.run(query)

print(answer)

**Tóm Tắt Các Lệnh Thao Tác với ChromaDB:**

1. **Khởi tạo Vector Store**: Chroma.from\_documents()
2. **Truy vấn dữ liệu**: vectorstore.similarity\_search()
3. **Cập nhật dữ liệu**: vectorstore.add\_documents()
4. **Lưu trữ**: vectorstore.persist()
5. **Xóa dữ liệu**: vectorstore.delete\_documents()
6. **Khôi phục**: Chroma(persist\_directory="path")
7. **Sử dụng ANN**: Cấu hình chỉ mục như HNSW hoặc IVF khi khởi tạo vector store.