



Search

Write

Sign up

Sign in



Eugene Tan · Follow

6 min read · Feb 13, 2024



+



MongoDB®

Image Credits: [Ollama](#) and [MongoDB Atlas](#)

## The Power of Ollama and MongoDB Atlas Vector Search

Ever wondered how to get started creating your own RAG application, without worrying about initializing and running your LLMs locally?

Ollama is a new open source tool to hit the streets today. It allows you to get started with some of the most popular LLMs out there (i.e. Llama2, Mistral, etc) by abstracting the complexity of managing them away behind a simple library / API.

In this post today, let's walk through how to build a simple RAG application with Ollama and MongoDB Atlas Vector Search using Langchain. So let's get cracking and build our next game changing AI product together!

## **More about MongoDB Atlas Vector Search**

MongoDB Atlas Vector Search offers a compelling solution for developers building Retrieval-Augmented Generation (RAG) applications due to its seamless integration of database management with advanced search capabilities. This feature allows for the efficient storage and retrieval of vectorized data, alongside operational / transactional data with its flexible schema.

This is crucial for applications that rely on understanding and processing large volumes of text or other complex data types. By enabling fast and accurate searches within high-dimensional vector spaces, MongoDB Atlas Vector Search facilitates the quick retrieval of relevant information, enhancing the performance and relevance of RAG applications.

## **More about Langchain**

LangChain is a Python library designed to facilitate the development of applications leveraging large language models (LLMs) such as GPT (Generative Pre-trained Transformer) models and their variants. It provides a suite of tools and abstractions that simplify the integration of language models into various tasks, including text generation, question answering,

and more complex workflows involving language understanding and generation.

## Ollama Synergy with MongoDB Atlas

Integrating Ollama with MongoDB Atlas leverages the strengths of both technologies to create RAG applications that are not just intelligent but also incredibly efficient in data retrieval and management. MongoDB Atlas's robust database solutions complement Ollama's processing power, enabling the development of applications that can manage and analyze large volumes of data with ease. This synergy is crucial for creating RAG applications that can quickly access the necessary data, understand the context of queries, and generate accurate, informative responses.

### Setting the Stage:

Retrieval-Augmented Generation applications combine the prowess of large language models (LLMs) with the precision of database queries, enabling more contextually rich and accurate responses. By leveraging Ollama, a quantized local version of LLM, alongside MongoDB Atlas's scalable database solutions, developers can craft applications that not only understand complex queries but also provide tailored responses based on a vast dataset.

### Prerequisites:

- Basic knowledge of Python and MongoDB.

- An environment to run Python (e.g., local machine, cloud-based IDE).
- MongoDB Atlas account and a cluster set up.
- Access to Ollama and necessary Python packages ( `langchain` , `pymongo` , etc.).

## Step-by-Step Guide:

### Step 1: Environment Setup

Begin by installing the required Python packages. Ensure you have `streamlit`, `pymongo`, `langchain`, and other dependencies installed. If you haven't set up a MongoDB Atlas cluster, now is the time to create one and obtain your connection URI.

*Create a `requirements.txt` like below.*

```
langchain
pymongo
streamlit
sentence_transformers
```

Also, do download Ollama from this [link](#) here to get started with it.

*Once done, run the below command to pull the Llama2 model locally using the Ollama CLI*

```
ollama pull llama2
```

## Step 2: MongoDB Atlas Configuration

Utilize MongoDB Atlas to store and manage your dataset. In our case, a collection of movies will serve as the foundation for our RAG application. Create a MongoDB Atlas cluster with a M0 tier (free-forever).

Ensure your database is populated with relevant data, such as movie titles and plot, which will be used to generate responses. You can do so by following these [steps](#) to load the sample movie data set first into your mongodb cluster.

## Step 3: Initialize Ollama and MongoDB Clients

Integrate Ollama for the language model capabilities and MongoDB client for database interactions. This step involves setting up the database connection using the URI and initializing the Ollama model with the desired configuration.

*Create a config.py like so below:*

```
mongo_uri = "mongodb+srv://user:password@<your-atlas-uri>/?retryWrites=true&w=m  
db_name = "sample_mflix"  
coll_name = "movies"
```

*Create a MongoDB Atlas vector search index (by following these steps [here](#)) after the data has been loaded to your `sample_mflix.movies` collection. The index definition for your vector search is like so:*

```
{
  "fields": [
    {
      "numDimensions": 384,
      "path": "plot_embedding_hf",
      "similarity": "cosine",
      "type": "vector"
    }
  ]
}
```

## Step 4: Embedding and Vector Search Setup

Configure text embeddings and vector search to enable efficient retrieval from your MongoDB collection. By using HuggingFace embeddings and MongoDB Atlas's vector search capabilities, you can transform movie descriptions into searchable vectors, facilitating quick and relevant retrievals based on user queries.

*Create a `encoder.py` file to encode your movie documents 10 at a time and store it in the same document once it has been encoded.*

```
from sentence_transformers import SentenceTransformer
import pymongo
import config
```

```
mongo_uri = config.mongo_uri
db = config.db_name
collection = config.coll_name

# initialize db connection
connection = pymongo.MongoClient(mongo_uri)
collection = connection[db][collection]

# define transofrmer model (from https://huggingface.co/sentence-transformers/a
model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")

for x in collection.find({"plot_embedding_hf": {"$exists": False}}, {}).limit(1):
    # checking if vector already computed for this doc
    if "vector" not in x.keys():
        if "title" in x.keys():
            movieid = x["_id"]
            title = x["title"]
            print("computing vector.. title: " + title)
            text = title
            fullplot = None

            # if fullpplot field present, concat it with title
            if "fullplot" in x.keys():
                fullplot = x["fullplot"]
                text = text + ". " + fullplot

            vector = model.encode(text).tolist()

            collection.update_one(
                {"_id": movieid},
                {
                    "$set": {
                        "plot_embedding_hf": vector,
                        "title": title,
                        "fullplot": fullplot,
                    }
                },
                upsert=True,
            )
            print("vector computed: " + str(x["_id"]))
        else:
            print("vector already computed")
```


*Run the encoder.py file and you should see your movie documents embedded with new vectors*



**sample\_mflix.movies**

STORAGE SIZE: 20.15MB   LOGICAL DATA SIZE: 33.83MB   TOTAL DOCUMENTS: 21349   INDEXES TOTAL SIZE: 16.96MB

**Find**   Indexes   Schema Anti-Patterns 0   Aggregation   Search Indexes

**Filter**    Type a query: { field: 'value' }

**QUERY RESULTS: 1-20 OF MANY**

```
_id: ObjectId('573a1390f29313caabcd42e8')
plot: "A group of bandits stage a brazen train hold-up, only to find a determ..."
▶ genres: Array (2)
runtime: 11
▶ cast: Array (4)
poster: "https://m.media-amazon.com/images/M/MV5BMTU3NjE5NzYtYTYyNS00MDVmLWIwYj..."
title: "The Great Train Robbery"
fullplot: "Among the earliest existing films in American cinema - notable as the ..."
▶ languages: Array (1)
released: 1903-12-01T00:00:00.000+00:00
▶ directors: Array (1)
rated: "TV-G"
▶ awards: Object
lastupdated: "2015-08-13 00:27:59.177000000"
year: 1903
▶ imdb: Object
▶ countries: Array (1)
type: "movie"
▶ tomatoes: Object
num_mflix_comments: 0
▼ plot_embedding_hf: Array (384)
  0: -0.07133565098047256
  1: -0.03232232853770256
  2: 0.01139499619603157
  3: 0.05617179721593857
  4: 0.04561607912182808
  5: 0.011870251037180424
  6: 0.04580720514059067
  7: 0.008101689629256725
  8: -0.0945034995675087
```

## Step 5: Building the Streamlit App

Leverage Streamlit to create a user-friendly interface for your application.

Implement input fields for user queries and buttons to trigger the retrieval and generation process. Display the results dynamically to provide users with immediate feedback.

*Create a main.py file and add the below code*

```
import streamlit as st
from langchain.callbacks.manager import CallbackManager
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
from langchain.chains import RetrievalQA
from langchain.llms import LlamaCpp
from langchain.vectorstores import MongoDBAtlasVectorSearch
from langchain.embeddings import HuggingFaceEmbeddings
from pymongo import MongoClient
import config
from langchain_community.llms import Ollama

# Initialize MongoDB client
uri = config.mongo_uri
client = MongoClient(uri)
db_name = config.db_name
coll_name = config.coll_name
collection = client[db_name][coll_name]

# Initialize text embedding model (encoder)
embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM
index_name = "vector_index"
vector_field_name = "plot_embedding_hf"
text_field_name = "title"

# Specify the MongoDB Atlas database and collection for vector search
vectorStore = MongoDBAtlasVectorSearch(
    collection=collection,
    embedding=embeddings,
    index_name=index_name,
    embedding_key=vector_field_name,
    text_key=text_field_name,
)
```

```
# Callbacks support token-wise streaming
callback_manager = CallbackManager([StreamingStdOutCallbackHandler()])

# Run the LLM from Ollama
llm = Ollama(model="llama2", callback_manager=callback_manager)

# Streamlit App
def main():
    st.title("Movies Retrieval GPT App")

    # User input
    query = st.text_input("Enter your query:")

    # Retrieve context data from MongoDB Atlas Vector Search
    retriever = vectorStore.as_retriever()

    # Query LLM with user input and context data
    if st.button("Query LLM"):
        with st.spinner("Querying LLM..."):
            qa = RetrievalQA.from_chain_type(
                llm, chain_type="stuff", retriever=retriever
            )

            response = qa({"query": query})

            st.text("Llama2 Response:")
            st.text(response["result"])

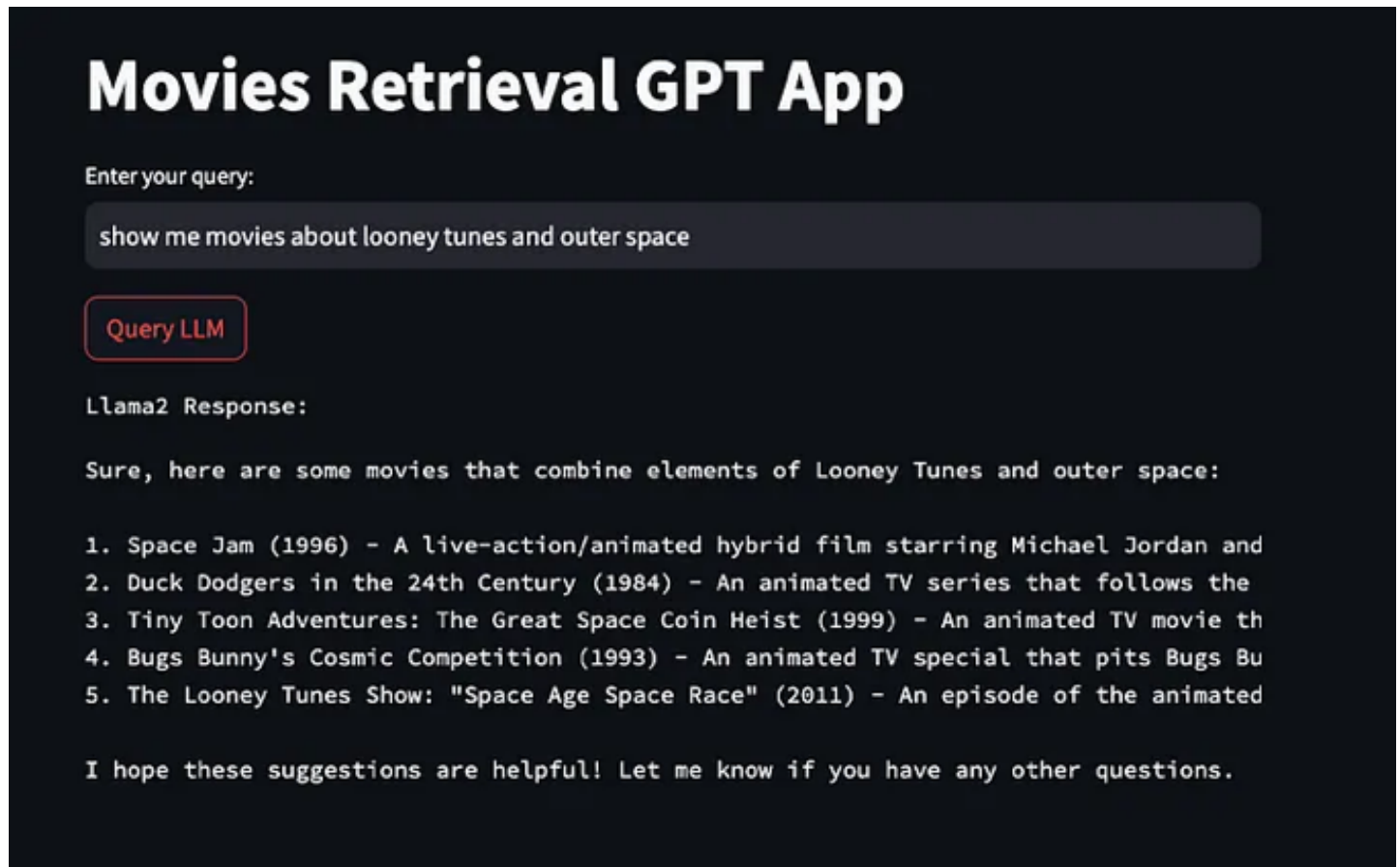
if __name__ == "__main__":
    main()
```

## Step 6: Retrieval and Response Generation

Run the main.py code above,

```
streamlit run main.py
```

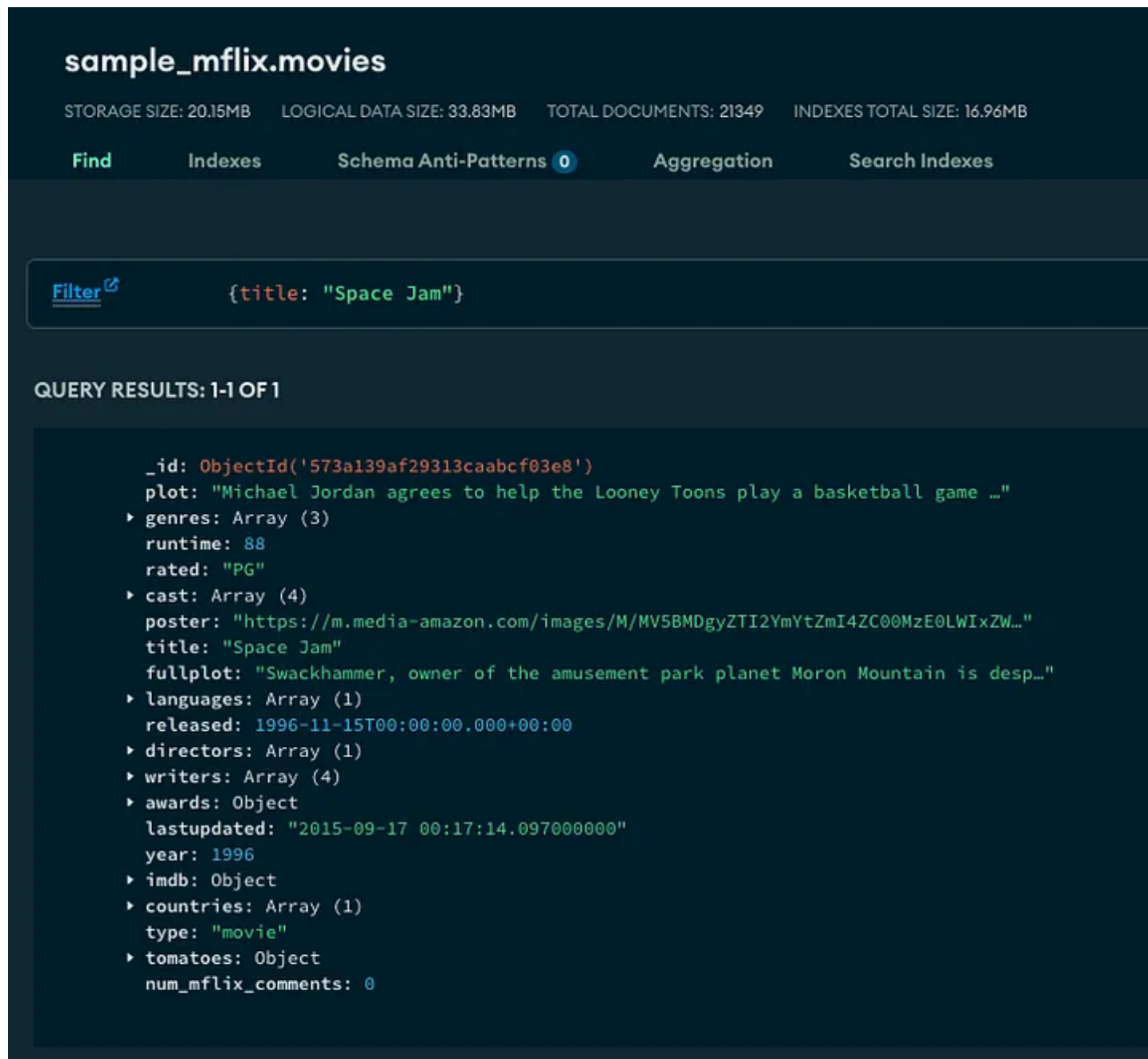
Go to localhost:8501 and ask a question about any movie.



Upon receiving a user query, Langchain will use the configured vector search to retrieve the most relevant movie data from MongoDB Atlas. Then, it will pass this context along with the query to Ollama to generate a tailored response, showcasing the power of retrieval-augmented generation.

You can also check your MongoDB Database to verify the existence of a

movie like “Space Jam” in the above example.



The screenshot displays the MongoDB Atlas web interface for a database named `sample_mflix.movies`. At the top, it shows database statistics: STORAGE SIZE: 20.15MB, LOGICAL DATA SIZE: 33.83MB, TOTAL DOCUMENTS: 21349, and INDEXES TOTAL SIZE: 16.96MB. Below these are navigation tabs: Find, Indexes, Schema Anti-Patterns (0), Aggregation, and Search Indexes. A filter bar contains the query `{title: "Space Jam"}`. The query results section shows 1 result of 1 total. The result is a document for the movie "Space Jam" with the following fields:

```
_id: ObjectId('573a139af29313caabcf03e8')
plot: "Michael Jordan agrees to help the Looney Toons play a basketball game ..."
genres: Array (3)
runtime: 88
rated: "PG"
cast: Array (4)
poster: "https://m.media-amazon.com/images/M/MV5BMDgyZTI2YmYtZmI4ZC00MzE0LWIxZW..."
title: "Space Jam"
fullplot: "Swackhammer, owner of the amusement park planet Moron Mountain is desp..."
languages: Array (1)
released: 1996-11-15T00:00:00.000+00:00
directors: Array (1)
writers: Array (4)
awards: Object
lastupdated: "2015-09-17 00:17:14.097000000"
year: 1996
imdb: Object
countries: Array (1)
type: "movie"
tomatoes: Object
num_mflix_comments: 0
```

## Conclusion:

This guide offers a glimpse into how easily it is to get started creating a local quantized LLM and building a RAG application together with Ollama’s ease

of use and MongoDB Atlas Vector Search's flexibility as a vector store. With these tools in hand, let's get stuck in and time to build our next AI powered app that will change the world!

If you want to dig into the weeds a little more, check out the Github [here](#)! Happy building folks!

Genai

Vector Database

Ollama

Langchain

Mongodb Atlas

**Written by Eugene Tan**

7 Followers

Follow

**Recommended from Medium**



## RECOMMENDED FROM MEDIUM



 Mark OBrien

## Mastering RAG: Local Intelligent Apps with Langchain & Ollama

★ · 10 min read · May 26, 2024

 59  1



 (λx.x)eranga in Effectz.AI

## Build RAG Application Using a LLM Running on Local Computer...

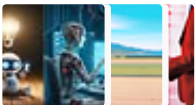
Privacy-preserving LLM without GPU

16 min read · Mar 17, 2024

 665  5



### Lists



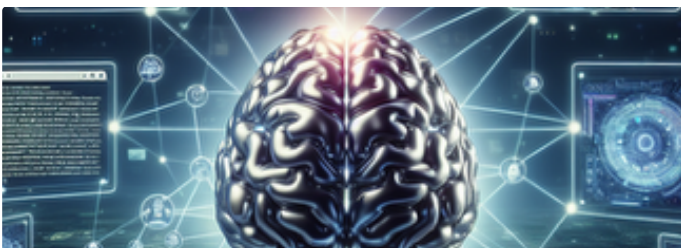
#### Natural Language Processing

1499 stories · 1026 saves



#### Staff Picks

660 stories · 1039 saves





 Kelvin Campelo


## How I've Optimized Document Interactions with Open WebUI an...

In this article, I'll share how I've enhanced my experience using my own private version of...

8 min read · May 5, 2024

 26  1



 Rohan Sharma

## LangChain vs LlamaIndex vs LiteLLM vs Ollama vs No...

LangChain vs LlamaIndex vs LiteLLM vs Ollama vs No Frameworks: A 3-Minute...

2 min read · May 12, 2024

 5  1



 Matthew Brazel

## Open Source & Self-hosted RAG LLM Server with ChromaDB,...

Subject-specific AI enriched on the knowledge base of your choice.

9 min read · Apr 18, 2024

 58 



# LangChain

 Meir Michanie

## Using Ollama Embedding services

Going local while doing deepLearning.ai "Build LLM Apps with LangChain.js" course.

2 min read · Mar 19, 2024

 13 





See more recommendations