

Boosting AI: Build Your Chatbot Over Your Data With MongoDB Atlas Vector Search and LangChain Templates Using the RAG Pattern

**Arek Borucki**

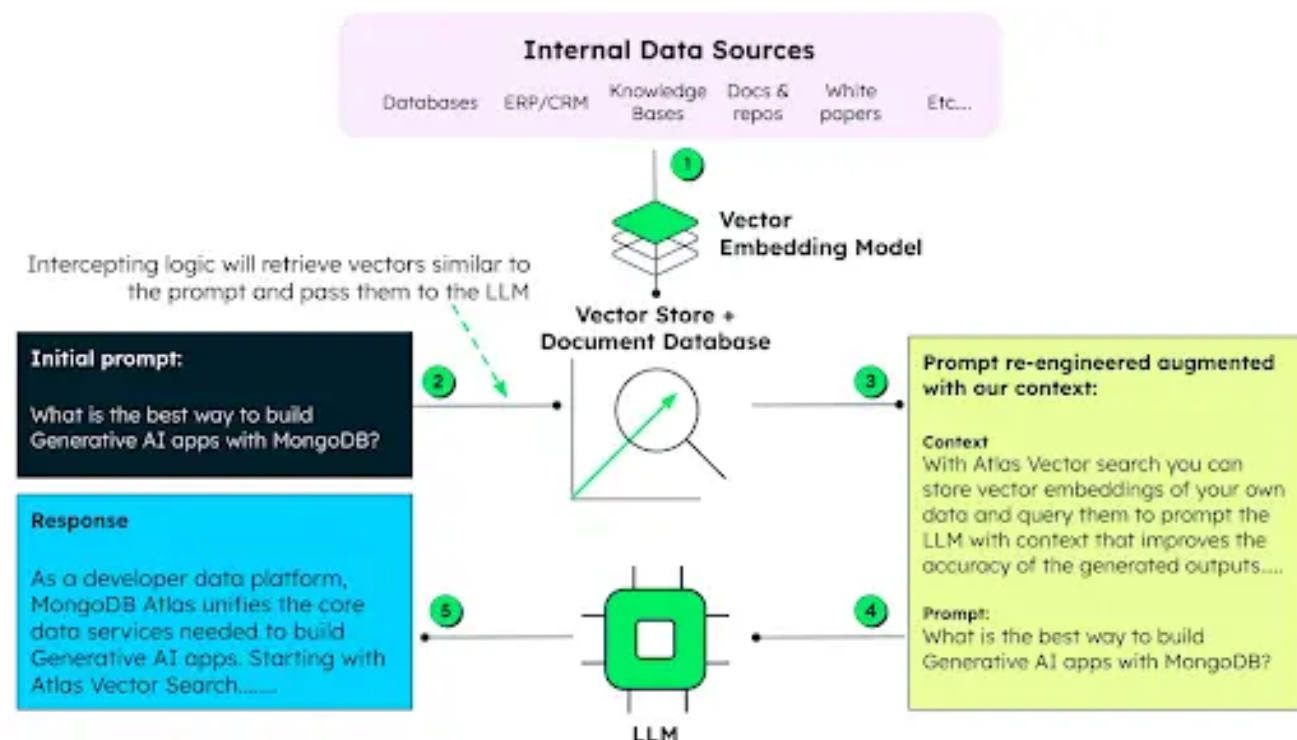
6 min read • Published Dec 12, 2023 • Updated Jan 16, 2024

AI

Atlas

Search

Python

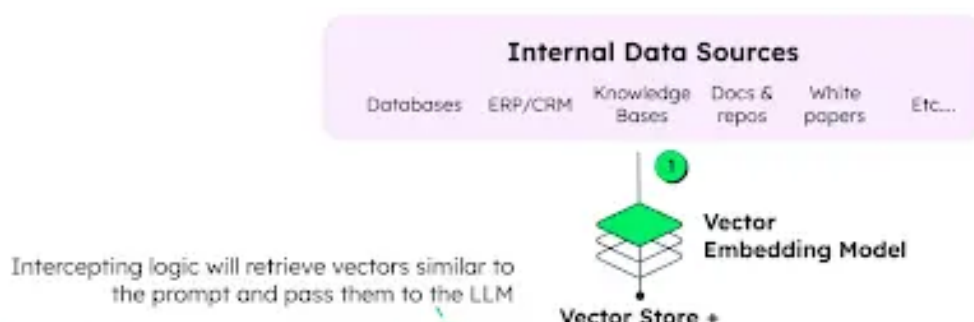


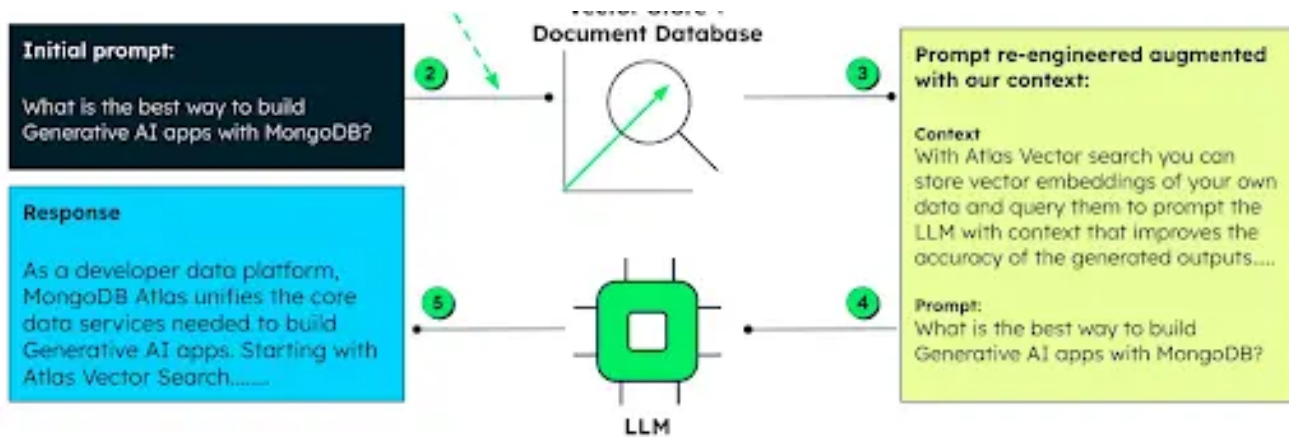
Rate this tutorial ☆ ☆ ☆ ☆ ☆

In this tutorial, I will show you the simplest way to implement an AI chatbot-style application using MongoDB Atlas Vector Search with LangChain Templates and the retrieval-augmented generation (RAG) pattern for more precise chat responses.

Retrieval-augmented generation (RAG) pattern

The retrieval-augmented generation (RAG) model enhances LLMs by supplementing them with additional, relevant data, ensuring grounded and precise responses for business purposes. Through vector search, RAG identifies and retrieves pertinent documents from databases, which it uses as context sent to the LLM along with the query, thereby improving the LLM's response quality. This approach decreases inaccuracies by anchoring responses in factual content and ensures responses remain relevant with the most current data. RAG optimizes token use without expanding an LLM's token limit, focusing on the most relevant documents to inform the response process.





To elaborate on the implementation: Consider a question-answering system that uses an LLM like OpenAI, operating with the RAG model. It starts with Atlas Vector Search to pinpoint relevant documents or text snippets within a database, providing the necessary context for the question. This context, along with the question, is then processed through OpenAI's API, enabling a more informed and accurate response.

Atlas Vector Search plays a vital role for developers within the retrieval-augmented generation framework. A key technology in funnelling external data into LLMs is LangChain. This framework facilitates the development of applications that integrate LLMs, covering a range of uses that align with the capabilities of language models themselves. These uses encompass tasks like document analysis and summarization, the operation of chatbots, and the scrutiny of code.

MongoDB has streamlined the process for developers to integrate AI into their applications by teaming up with LangChain for the introduction of [LangChain Templates](#). This collaboration has produced a retrieval-augmented generation template that capitalizes on the strengths of MongoDB Atlas Vector Search along with OpenAI's technologies. The template offers a developer-friendly approach to crafting and deploying chatbot applications tailored to specific data sets. The LangChain templates serve as a deployable reference framework, accessible as a REST API via [LanaServe](#).

The alliance has also been instrumental in showcasing the latest Atlas Vector Search advancements, notably the `$vectorSearch` aggregation stage, now embedded within LangChain's Python and JavaScript offerings. The joint venture is committed to ongoing development, with plans to unveil more templates. These future additions are intended to further accelerate developers' abilities to realise and launch their creative projects.

LangChain Templates

LangChain Templates present a selection of reference architectures that are designed for quick deployment, available to any user. These templates introduce an innovative system for the crafting, exchanging, refreshing, acquiring, and tailoring of diverse chains and agents. They are crafted in a uniform format for smooth integration with LangServe, enabling the swift deployment of production-ready APIs. Additionally, these templates provide a free sandbox for experimental and developmental purposes.

The `rag-mongo` template is specifically designed to perform retrieval-augmented generation utilizing MongoDB and OpenAI technologies. We will take a closer look at the `rag-mongo` template in the following section of this tutorial.

Using LangChain RAG templates

To get started, you only need to install the `langchain-cli`.

```
1 pip3 install -U "langchain-cli[serve]"
```



Use the LangChain CLI to bootstrap a LangServe project quickly. The application will be named `my-blog-article`, and the name of the template

must also be specified. I'll name it `rag-mongo`.

```
1 langchain app new my-blog-article --package rag-mongo
```

This will create a new directory called my-app with two folders:

- `app`: This is where LangServe code will live.
- `packages`: This is where your chains or agents will live.

Now, it is necessary to modify the `my-blog-article/app/server.py` file by adding the following code:

```
1 from rag_mongo import chain as rag_mongo_chain
2 add_routes(app, rag_mongo_chain, path="/rag-mongo")
```

We will need to insert data to MongoDB Atlas. In our exercise, we utilize a publicly accessible PDF document titled "[MongoDB Atlas Best Practices](#)" as a data source for constructing a text-searchable vector space. The data will be ingested into the MongoDB `langchain.vectorSearch` namespace.

In order to do it, navigate to the directory `my-blog-article/packages/rag-mongo` and in the file `ingest.py`, change the default names of the MongoDB database and collection. Additionally, modify the URL of the document you wish to use for generating embeddings.

```
1 cd my-blog-article/packages/rag-mongo
```

My `ingest.py` is located on [GitHub](#). Note that if you change the database

and collection name in `ingest.py`, you also need to change it in `rag_mongo` / `chain.py`. My `chain.py` is also located on [GitHub](#). Next, export your OpenAI API Key and MongoDB Atlas URI.

```
1 export OPENAI_API_KEY="xxxxxxxxxxxxx"
2 export MONGO_URI
3 ="mongodb+srv://user:passwd@vectorsearch.abc.mongodb.net/"
```

Creating and inserting embeddings into MongoDB Atlas using LangChain templates is very easy. You just need to run the `ingest.py` script. It will first load a document from a specified URL using the PyPDFLoader. Then, it splits the text into manageable chunks using the `RecursiveCharacterTextSplitter`. Finally, the script uses the OpenAI Embeddings API to generate embeddings for each chunk and inserts them into the MongoDB Atlas `langchain.vectorSearch` namespace.

```
1 python3 ingest.py
```

Now, it's time to initialize Atlas Vector Search. We will do this through the Atlas UI. In the Atlas UI, choose `Search` and then `Create Search`. Afterwards, choose the JSON Editor to declare the index parameters as well as the database and collection where the Atlas Vector Search will be established (`langchain.vectorSearch`). Set index name as `default`. The definition of my index is presented below.

```
1 {
2   "type": "vectorSearch",
3   "fields": [
```

```
4      {
5          "path": "embedding",
6          "dimensions": 1536,
7          "similarity": "cosine",
8          "type": "vector"
9      }
10 ]
11 }
```



A detailed procedure is [available on GitHub](#).

Let's now take a closer look at the central component of the LangChain `rag-mongo` template: the `chain.py` script. This script utilizes the `MongoDBAtlasVectorSearch`

class and is used to create an object – `vectorstore` – that interfaces with MongoDB Atlas's vector search capabilities for semantic similarity searches. The `retriever` is then configured from `vectorstore` to perform these searches, specifying the search type as "similarity."

```
1 vectorstore = MongoDBAtlasVectorSearch.from_connection_string(
2     MONGO_URI,
3     DB_NAME + "." + COLLECTION_NAME,
4     OpenAIEmbeddings(disallowed_special=()),
5     index_name=ATLAS_VECTOR_SEARCH_INDEX_NAME,
6 )
7 retriever = vectorstore.as_retriever()
```



This configuration ensures the most contextually relevant document is

retrieved from the database. Upon retrieval, the script merges this document with a user's query and leverages the `ChatOpenAI` class to process the input through OpenAI's GPT models, crafting a coherent answer. To further enhance this process, the `ChatOpenAI` class is initialized with the `gpt-3.5-turbo-16k-0613` model, chosen for its optimal performance. The temperature is set to 0, promoting consistent, deterministic outputs for a streamlined and precise user experience.

```
1 model = ChatOpenAI(model_name="gpt-3.5-turbo-16k-0613", temperature=0)
```

This class permits tailoring API requests, offering control over retry attempts, token limits, and response temperature. It adeptly manages multiple response generations, response caching, and callback operations. Additionally, it facilitates asynchronous tasks to streamline response generation and incorporates metadata and tagging for comprehensive API run tracking.

LangServe Playground

After successfully creating and storing embeddings in MongoDB Atlas, you can start utilizing the LangServe Playground by executing the `langchain serve` command, which grants you access to your chatbot.


```
1 langchain serve
2
3 INFO:      Will watch for changes in these directories:
4 INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
5 INFO:      Started reloader process [50552] using StatReload
6 INFO:      Started server process [50557]
7 INFO:      Waiting for application startup.
8
9 LANGSERVE: Playground for chain "/rag-mongo" is live at:
10 LANGSERVE: |
11 LANGSERVE: └─> /rag-mongo/playground
12 LANGSERVE:
13 LANGSERVE: See all available routes at /docs
```



This will start the FastAPI application, with a server running locally at

<http://127.0.0.1:8000>. All templates can be viewed at

<http://127.0.0.1:8000/docs>, and the playground can be accessed at

<http://127.0.0.1:8000/rag-mongo/playground/>.

The chatbot will answer questions about best practices for using MongoDB Atlas with the help of context provided through vector search. Questions on other topics will not be considered by the chatbot.


Go to the following URL:

...to the following URL:

1 <http://127.0.0.1:8000/rag-mongo/playground/>



And start using your template! You can ask questions related to MongoDB Atlas in the chat.

 **LangServe Playground**

Try it

Inputs

Reset

QUESTION

Explain how MongoDB Atlas uses auditing

Output

MongoDB Atlas uses auditing to track and record activities and changes made within its dedicated clusters. This auditing configuration applies to all dedicated clusters within an Atlas project. Audit logs can be downloaded through the UI or retrieved using the MongoDB Atlas API. This auditing feature allows for increased security and compliance, as it provides visibility into who accessed the database, what actions were taken, and when they occurred.

Intermediate steps 6

Share

Start

By expanding the **Intermediate steps** menu, you can trace the entire process of formulating a response to your question. This process encompasses searching for the most pertinent documents related to your query, and

forwarding them to the Open AI API to serve as the context for the query. This methodology aligns with the RAG pattern, wherein relevant documents are retrieved to furnish context for generating a well-informed response to a specific inquiry.

We can also use `curl` to interact with `LangServe` REST API and contact endpoints, such as `/rag-mongo/invoke`:

```
1 curl -X POST "https://127.0.0.1:8000/rag-mongo/invoke" \  
2   -H "Content-Type: application/json" \  
3   -d '{"input": "Does MongoDB support transactions?"}'
```

```
1 {"output": "Yes, MongoDB supports transactions.", "callback": "content"}
```

We can also send batch requests to the API using the `/rag-mongo/batch` endpoint, for example:

```
1 curl -X POST "https://127.0.0.1:8000/rag-mongo/batch" \  
2   -H "Content-Type: application/json" \  
3   -d '[{"input": "Does MongoDB support transactions?"}]'
```

```
2 -H "Content-Type: application/json" \  
3 -d '{  
4     "inputs": [  
5         "What options do MongoDB Atlas Indexes include?",  
6         "Explain Atlas Global Cluster",  
7         "Does MongoDB Atlas provide backups?"  
8     ],  
9     "config": {},  
10    "kwargs": {}  
11    }'
```



```
1 spatial indexes\n- Text search indexes\n- Unique indexes\n- Ar
```



For comprehensive documentation and further details, please visit

<http://127.0.0.1:8000/docs>.

Summary

In this article, we've explored the synergy of MongoDB Atlas Vector Search with LangChain Templates and the RAG pattern to significantly improve chatbot response quality. By implementing these tools, developers can ensure their AI chatbots deliver highly accurate and contextually relevant answers. Step into the future of chatbot technology by applying the insights and instructions

provided here. Elevate your AI and engage users like never before. Don't just build chatbots – craft intelligent conversational experiences.

Start now with MongoDB Atlas and LangChain!

Rate this tutorial ☆ ☆ ☆ ☆ ☆

Related

CODE EXAMPLE

Build Your Own Wordle in Bash with the Data API

Aug 26, 2022 | 6 min read

ARTICLE

Using Atlas Data Federation to Control Access to Your Analytics Node

Jun 28, 2023 | 9 min read

ARTICLE

Querying the MongoDB Atlas Price Book with Atlas Data Federation

Jun 15, 2023 | 4 min read

TUTORIAL

Get Hyped: Synonyms in Atlas Search

Feb 27, 2023 | 9 min read