Revolutionizing AI Interaction: Integrating Mistral AI and MongoDB for a Custom LLM GenAI Application

Han Heloir

11 min read • Published Feb 08, 2024 • Updated Feb 13, 2024

MongoDB

Rate this tutorial

Large language models (LLMs) are known for their ability to converse with us in an almost human-like manner. Yet, the complexity of their inner workings often remains covered in mystery, sparking intrigue. This intrigue intensifies when we factor in the privacy challenges associated with AI technologies.

In addition to privacy concerns, cost is another significant challenge. Deploying a large language model is crucial for AI applications, and there are two primary options: self-hosted or API-based models. With API-based LLMs, the model is hosted by a service provider, and costs accrue with each API request. In contrast, a self-hosted LLM runs on your own infrastructure, giving you complete control over costs. The bulk of expenses for a self-hosted LLM pertains to the necessary hardware.

Another aspect to consider is the availability of LLM models. With API-based models, during times of high demand, model availability can be compromised. In contrast, managing your own LLM ensures control over availability. You will be able to make sure all your queries to your self-managed LLM can be handled properly and under your control.

Mistral AI, a French startup, has introduced innovative solutions with the Mistral 7B model, Mistral Mixture of Experts, and Mistral Platform, all standing for a spirit of openness. This article explores how Mistral AI, in collaboration with MongoDB, a developer data platform that unifies operational, analytical, and vector search data services, is revolutionizing our interaction with AI. We will delve into the integration of Mistral AI with MongoDB Atlas and discuss its impact on privacy, cost efficiency, and AI accessibility.

Mistral AI: a game-changer

Mistral AI has emerged as a pivotal player in the open-source AI community, setting new standards in AI innovation. Let's break down what makes Mistral AI so transformative.

A beacon of openness: Mistral AI's philosophy

Mistral AI's commitment to openness is at the core of its philosophy. This commitment extends beyond just providing open-source code; it's about advocating for transparent and adaptable AI models. By prioritizing transparency, Mistral AI empowers users to truly own and shape the future of AI. This approach is fundamental to ensuring AI remains a positive, accessible force for everyone.

Unprecedented performance with Mistral 8x7B

Mistral AI has taken a monumental leap forward with the release of Mixtral 8x7B, an innovative sparse mixture of experts model (SMoE) with open weights. An SMoE is a neural network architecture that boosts traditional model efficiency and scalability. It utilizes specialized “expert” sub-networks to handle different input segments. Mixtral incorporates eight of these expert sub-networks.

Licensed under Apache 2.0, Mixtral sets a new benchmark in the AI landscape. Here's a closer look at what makes Mixtral 8x7B a groundbreaking advancement.

High-performance with sparse architectures

Mixtral 8x7B stands out for its efficient utilization of parameters and high-quality performance. Despite its total parameter count of 46.7 billion, it operates using only 12.9 billion parameters per token. This unique architecture allows Mixtral to maintain the speed and cost efficiency of a 12.9 billion parameter model while offering the capabilities of a much larger model.

Superior performance, versatility, and cost-performance optimization

Mixtral rivals leading models like Llama 2 70B and GPT-3.5, excelling in handling large contexts, multilingual processing, code generation, and instruction-following. The Mixtral 8x7B model combines cost efficiency with high performance, using a sparse mixture of experts network for optimized resource usage, offering premium outputs at lower costs compared to similar models.

Mistral “La plateforme”

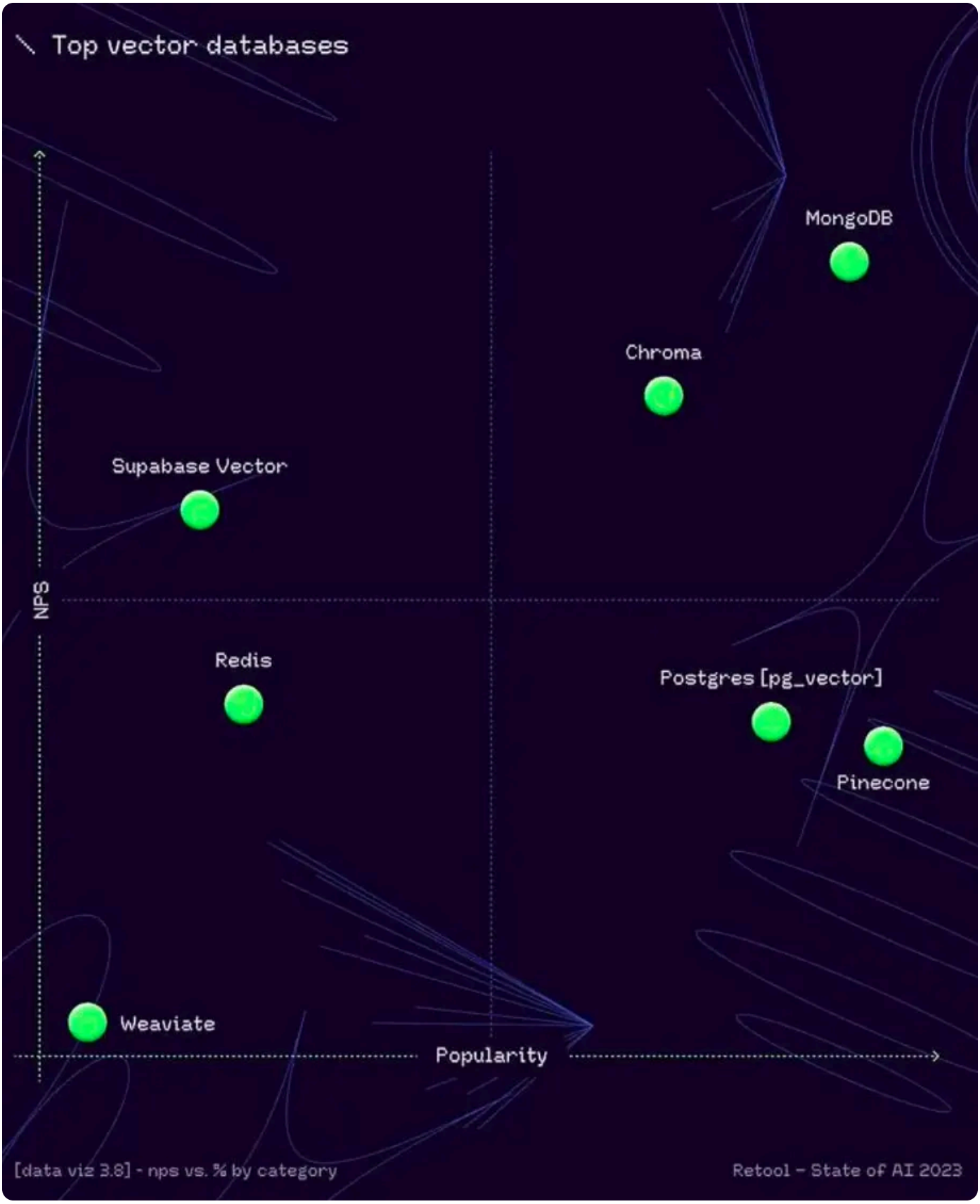
Mistral AI's beta platform offers developers generative models focusing on simplicity: Mistral-tiny for cost-effective, English-only text generation (7.6 MT-Bench score), Mistral-small for multilingual support including coding (8.3 score), and Mistral-medium for high-quality, multilingual output (8.6 score). These user-friendly, accurately fine-tuned models facilitate efficient AI deployment, as demonstrated in our article using the Mistral-tiny and the platform's embedding model.

Why MongoDB Atlas as a vector store?

MongoDB Atlas is a unique, fully-managed platform integrating enterprise data, vector search, and analytics, allowing the creation of tailored AI applications. It goes beyond standard vector search with a comprehensive ecosystem, including models like Mistral, setting it apart in terms of unification, scalability, and security.

MongoDB Atlas unifies operational, analytical, and vector search data services to streamline the building of generative AI-enriched apps. From proof-of-concept to production, MongoDB Atlas empowers developers with scalability, security, and performance for their mission-critical production applications.

According to the Retool AI report, MongoDB takes the lead, earning its place as the top-ranked vector database.



- Vector store easily works together with current MongoDB databases, making it a simple addition for groups already using MongoDB for managing their

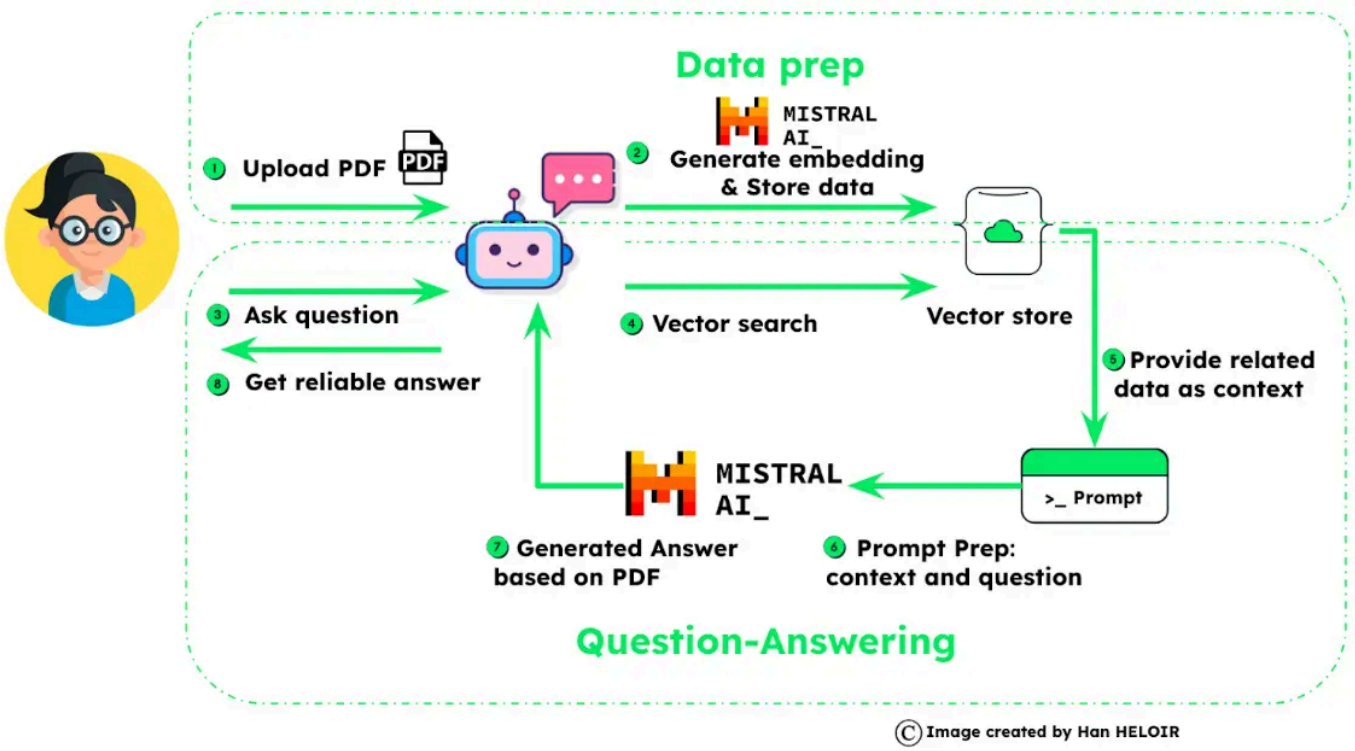
data. This means they can start using vector storage without needing to make big changes to their systems.

- MongoDB Atlas is purpose-built to handle large-scale, operation-critical applications, showcasing its robustness and reliability. This is especially important in applications where it's critical to have accurate and accessible data.
- Data in MongoDB Atlas is stored in JSON format, making it an ideal choice for managing a variety of data types and structures. This is particularly useful for AI applications, where the data type can range from embeddings and text to integers, floating-point values, GeoJSON, and more.
- MongoDB Atlas is designed for enterprise use, featuring top-tier security, the ability to operate across multiple cloud services, and is fully managed. This ensures organizations can trust it for secure, reliable, and efficient operations.

With MongoDB Atlas, organizations can confidently store and retrieve embeddings alongside your existing data, unlocking the full potential of AI for their applications.

Overview and implementation of your custom LLM GenAI app

Creating a self-hosted LLM GenAI application integrates the power of open-source AI with the robustness of an enterprise-grade vector store like MongoDB. Below is a detailed step-by-step guide to implementing this innovative system:



1. Data acquisition and chunk

The first step is gathering data relevant to your application's domain, including text documents, web pages, and importantly, operational data already stored in MongoDB Atlas. Leveraging Atlas's operational data adds a layer of depth, ensuring your AI application is powered by comprehensive, real-time data, which is crucial for contextually enriched AI responses.

Then, we divide the data into smaller, more manageable chunks. This division is crucial for efficient data processing, guaranteeing the AI model interacts with data that is both precise and reflective of your business's operational context.

2.1 Generating embeddings

Utilize **Mistral AI embedding endpoint** to transform your segmented text data into embeddings. These embeddings are numerical representations that capture the essence of your text, making it understandable and usable by AI models.

2.2 Storing embeddings in MongoDB vector store

Once you have your embeddings, store them in MongoDB's vector store. MongoDB Atlas, with its advanced search capabilities, allows for the efficient storing and managing of these embeddings, ensuring that they are easily accessible when needed.

2.3 Querying your data

Use **MongoDB's vector search** capability to query your stored data. You only need to create a vector search index on the embedding field in your document. This powerful feature enables you to perform complex searches and retrieve the most relevant pieces of information based on your query parameters.

3. & 4. Embedding questions and retrieving similar chunks

When a user poses a question, generate an embedding for this query. Then, using MongoDB's search functionality, retrieve data chunks that are most similar to this query embedding. This step is crucial for finding the most relevant information to answer the user's question.

5. Contextualized prompt creation

Combine the retrieved segments and the original user query to create a comprehensive prompt. This prompt will provide a context to the AI model, ensuring that the responses generated are relevant and accurate.

6. & 7. Customized answer generation from Mistral AI

Feed the contextualized prompt into the Mistral AI 7B LLM. The model will then generate a customized answer based on the provided context. This step leverages the advanced capabilities of Mistral AI to provide specific, accurate, and relevant answers to user queries.

Implementing a custom LLM GenAI app with Mistral AI and MongoDB Atlas

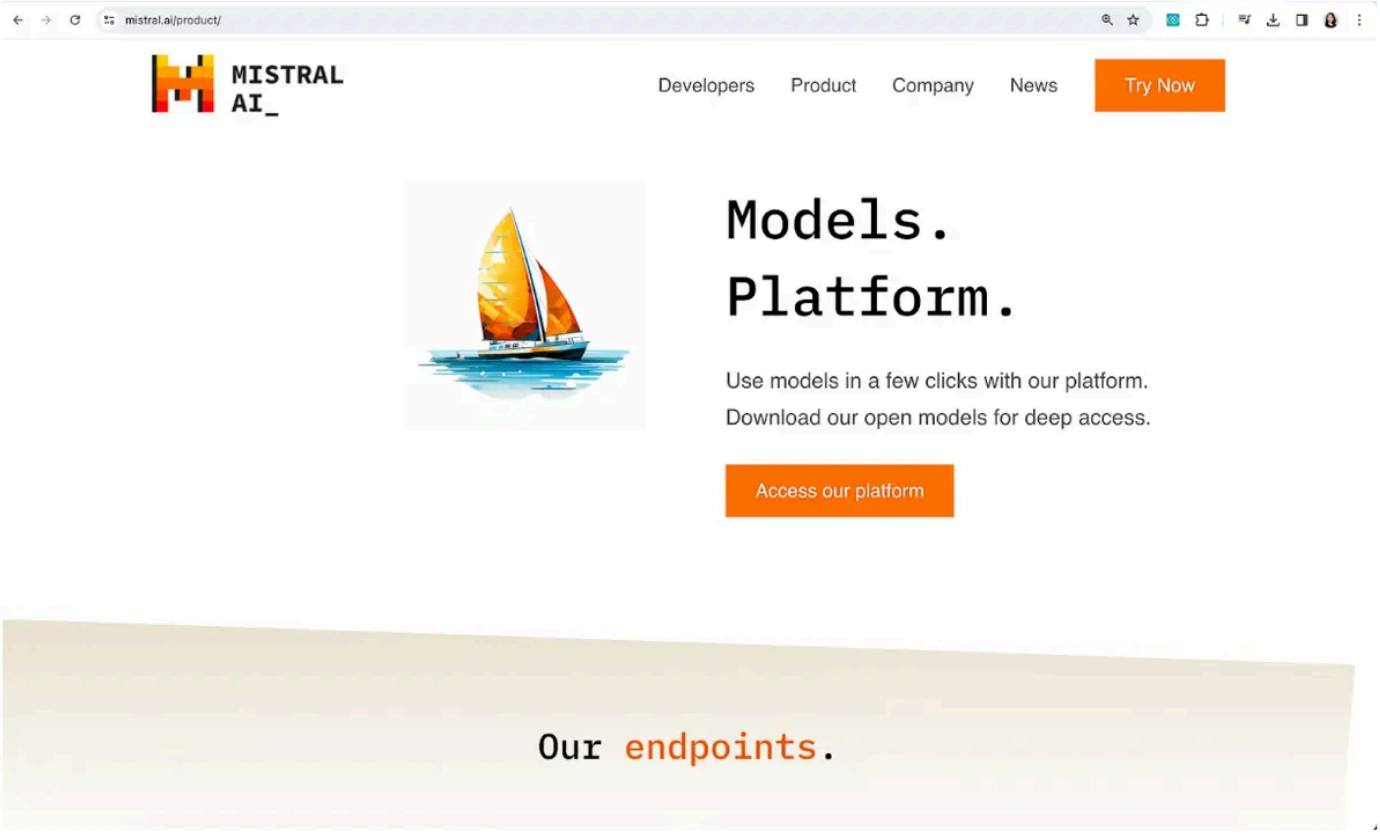
Now that we have a comprehensive understanding of Mistral AI and MongoDB Atlas and the overview of your next custom GenAI app, let's dive into implementing a custom large language model GenAI app. This app will allow you to have your own personalized AI assistant, powered by the Mistral AI and supported by the efficient data management of MongoDB Atlas.

In this section, we'll explain the prerequisites and four parts of the code:

- Needed libraries
- Data preparation process
- Question and answer process
- User interface through Gradio

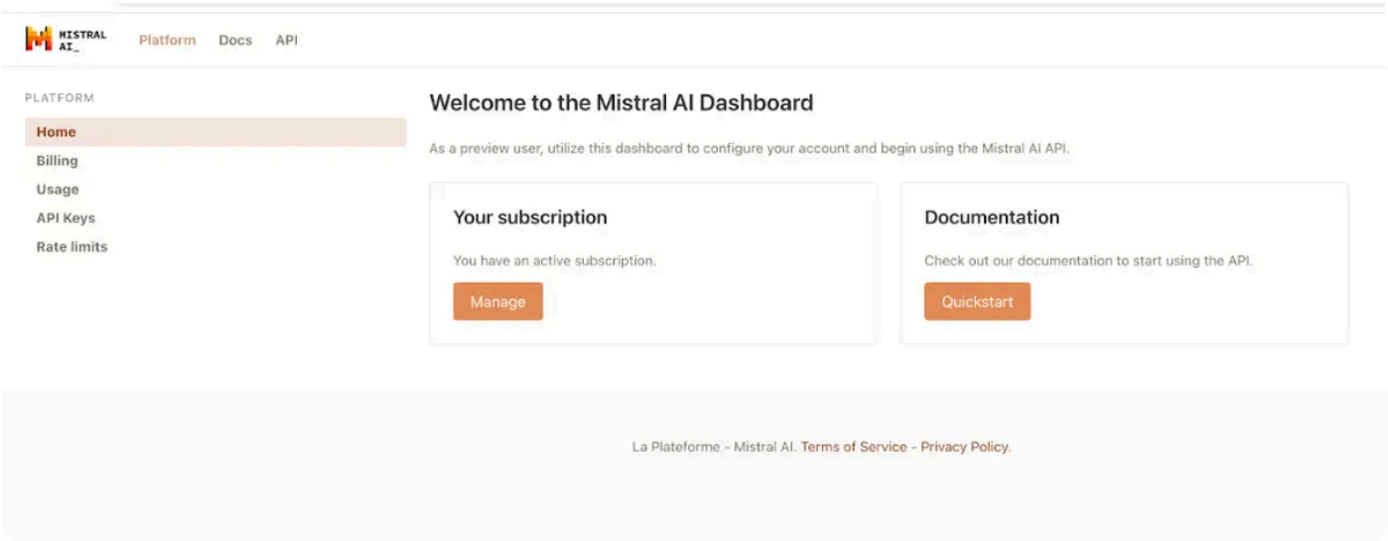
0. Prerequisites

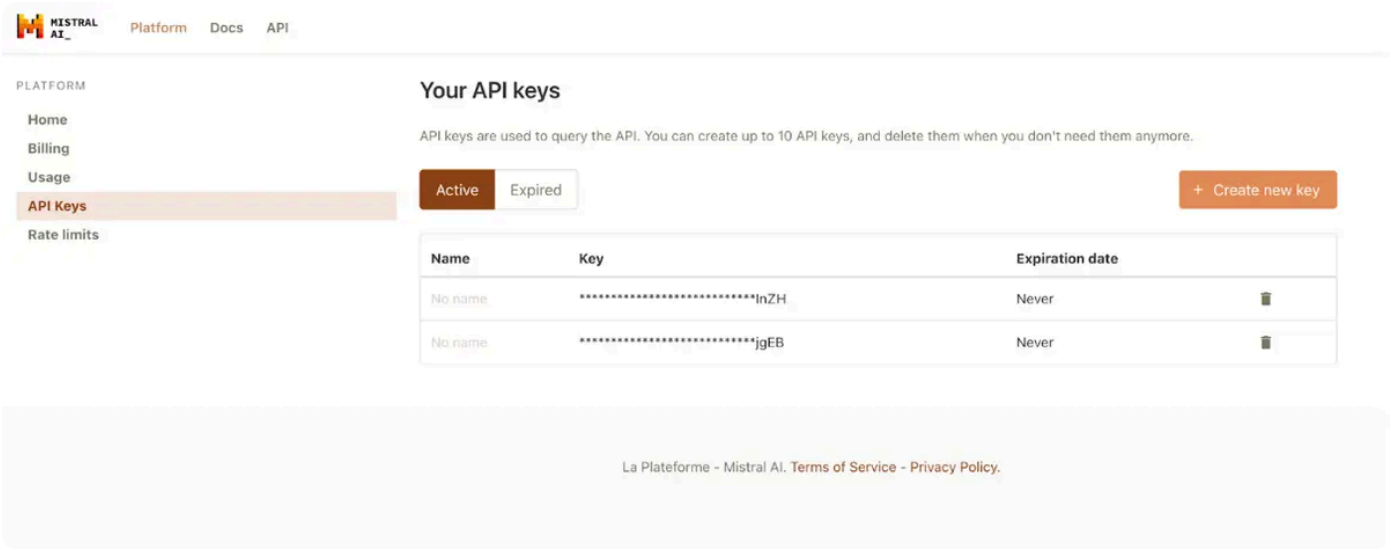
As explained above, in this article, we are going to leverage the Mistral AI model through Mistral "La plateforme." To get access, you should first create an account on Mistral AI. You may need to wait a few hours (or one day) before your account is activated.



Once your account is activated, you can add your subscription. Follow the instructions step by step on the Mistral AI platform.

Once you have set up your subscription, you can then generate your API key for future usage.





Besides using the Mistral AI “La plateforme,” you have another option to implement the Mistral AI model on a machine featuring Nvidia V100, V100S, or A100 GPUs (not an exhaustive list). If you want to deploy a self-hosted large language model on a public or private cloud, you can refer to my previous article on how to deploy Mistral AI within 10 minutes.

1. Import needed libraries

This section shows the versions of the required libraries. Personally, I run my code in VScode. So you need to install the following libraries beforehand. Here is the version at the moment I’m running the following code.

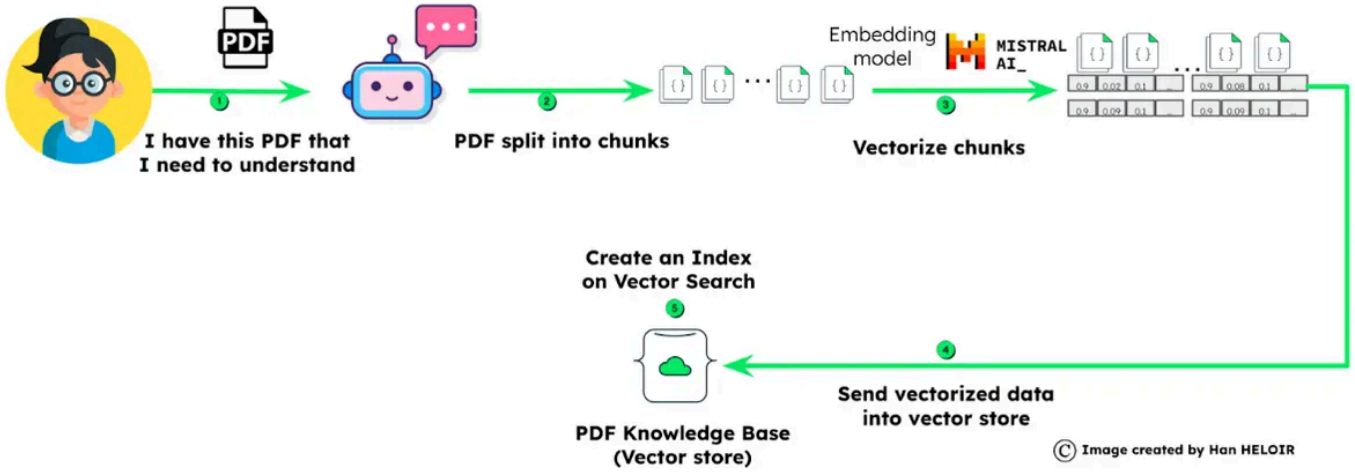
```
mistralai
pymongo                    4.3.3
gradio                     4.10.0
gradio_client              0.7.3
langchain                  0.0.348
langchain-core             0.0.12
pandas                     2.0.3
```

These include libraries for data processing, web scraping, AI models, and database interactions.

```
import gradio as gr
import os
import pymongo
import pandas as pd
from mistralai.client import MistralClient
from mistralai.models.chat_completion import ChatMessage
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitte
```

2. Data preparation

The data_prep() function loads data from a PDF, a document, or a specified URL. It extracts text content from a webpage/documentation, removes unwanted elements, and then splits the data into manageable chunks.



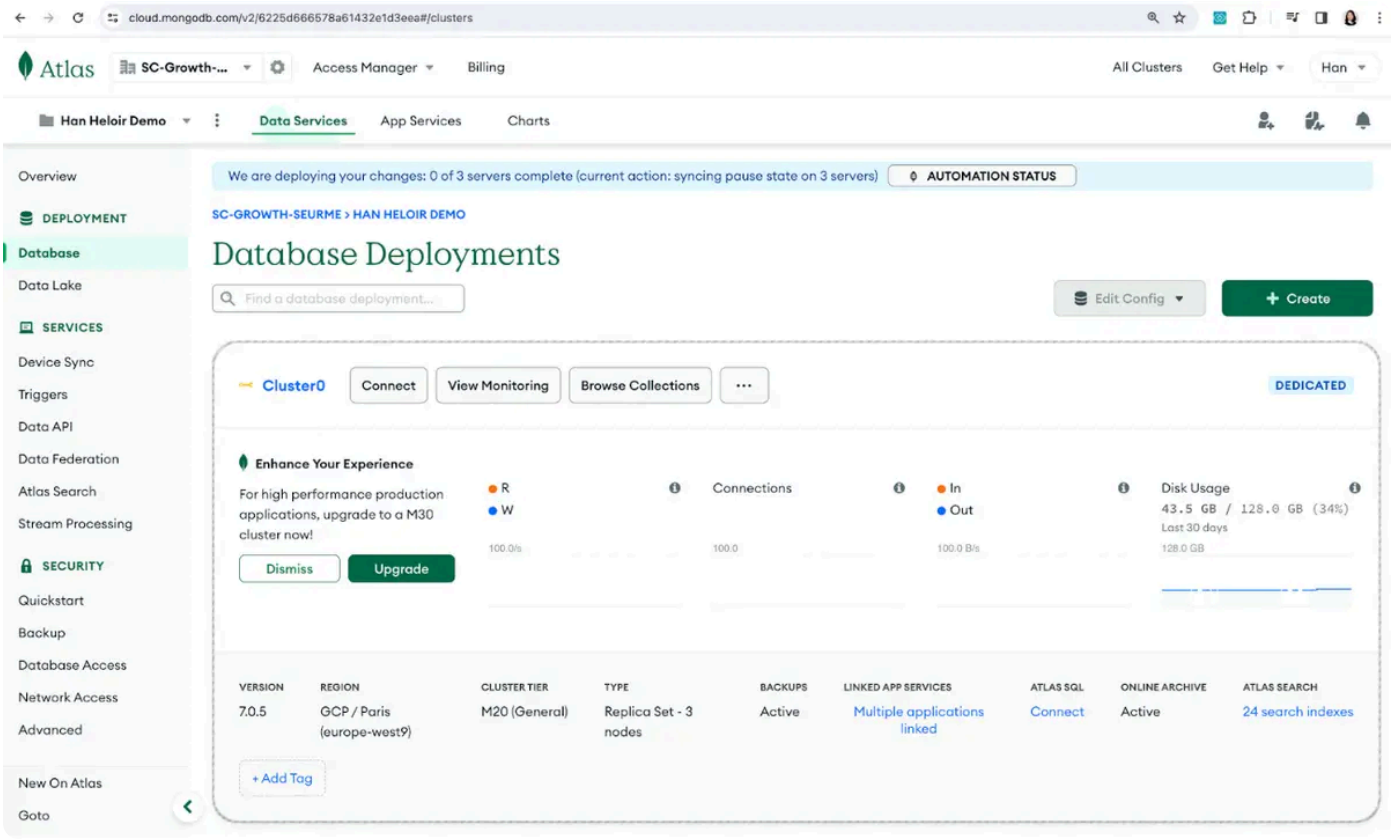
Once the data is chunked, we use the Mistral AI embedding endpoint to compute embeddings for every chunk and save them in the document. Afterward, each document is added to a MongoDB collection.

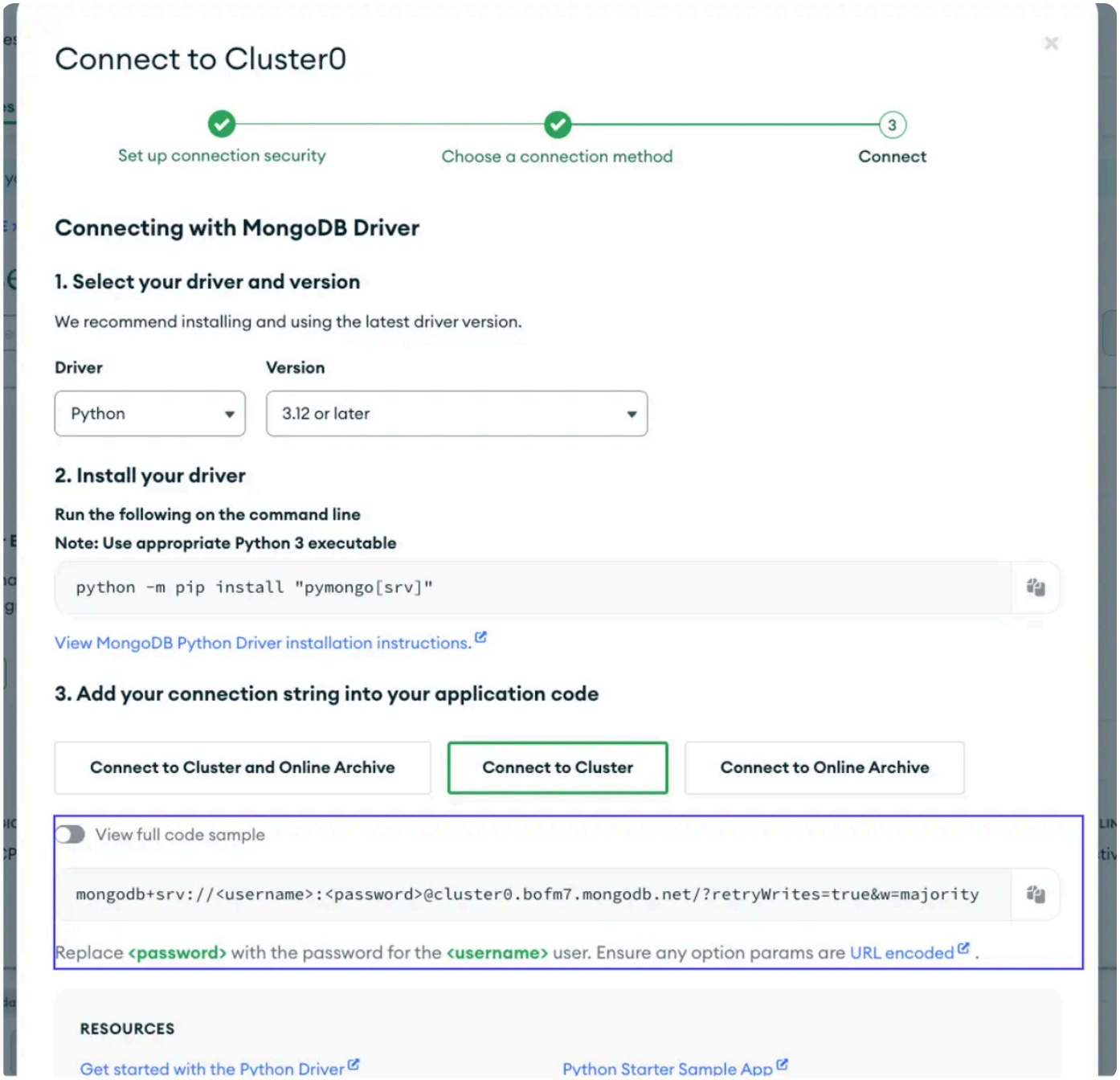
```
def data_prep(file):
    # Set up Mistral client
    api_key = os.environ["MISTRAL_API_KEY"]
    client = MistralClient(api_key=api_key)

    # Process the uploaded file
    loader = PyPDFLoader(file.name)
    pages = loader.load_and_split()

    # Split data
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=100,
        chunk_overlap=20,
        separators=["\n\n", "\n", "(?<=\. )", " ", ""],
        length_function=len,
    )
    docs = text_splitter.split_documents(pages)

    # Calculate embeddings and store into MongoDB
    text_chunks = [text.page_content for text in docs]
    df = pd.DataFrame({'text_chunks': text_chunks})
    df['embedding'] = df.text_chunks.apply(lambda x: get_embeddin

    collection = connect_mongodb()
    df_dict = df.to_dict(orient='records')
    collection.insert_many(df_dict)

    return "PDF processed and data stored in MongoDB."
```

Connecting to MongoDB server

The `connect_mongodb()` function establishes a connection to a MongoDB server. It returns a collection object that can be used to interact with the database. This function will be called in the `data_prep()` function.

In order to get your MongoDB connection string, you can go to your MongoDB Atlas console, click the “Connect” button on your cluster, and choose the Python driver.






```python
def connect_mongodb():
    # Your MongoDB connection string
    mongo_url = os.environ["MONGO_URI"]
    client = pymongo.MongoClient(mongo_url)
    db = client["mistralpdf"]
    collection = db["pdfRAG"]
    return collection
```

You can import your mongo_url by doing the following command in shell.

```
export MONGO_URI="Your_cluster_connection_string"
```

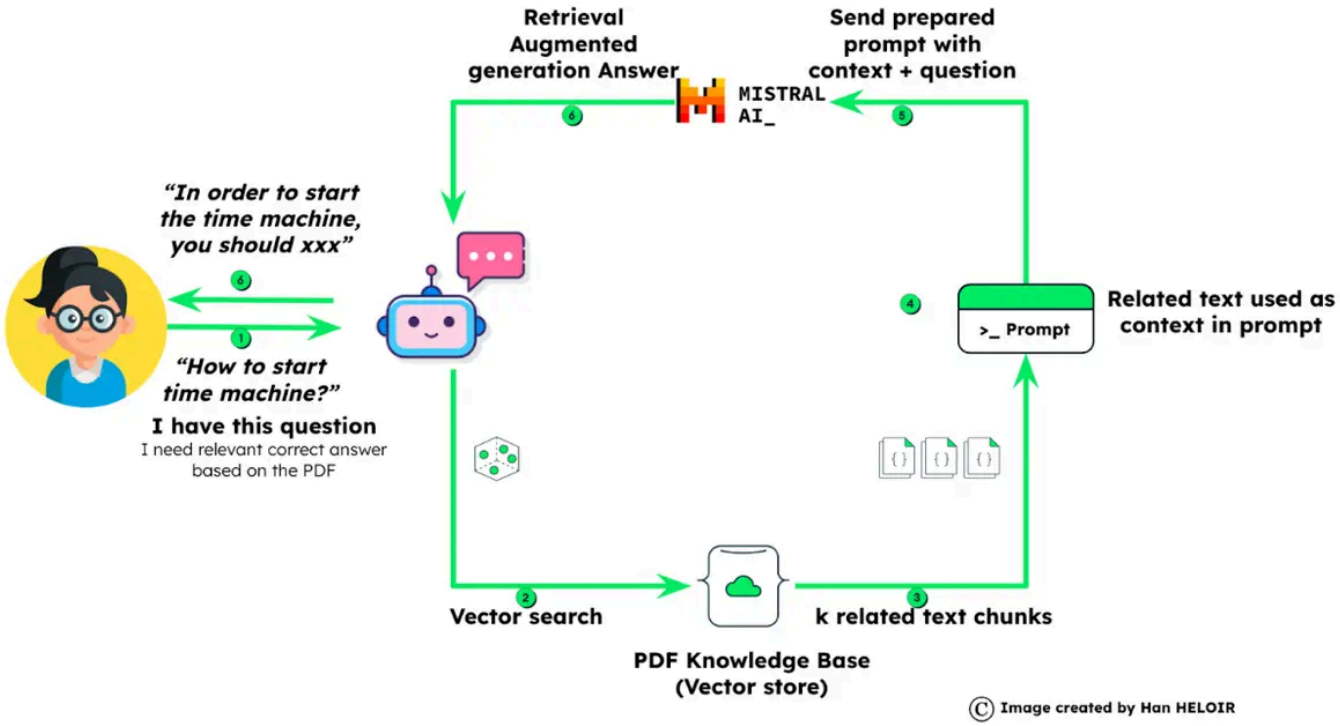
Getting the embedding

The get_embedding(text) function generates an embedding for a given text. It replaces newline characters and then uses Mistral AI “La plateforme” embedding endpoints to get the embedding. This function will be called in both data preparation and question and answering processes.

```
def get_embedding(text, client):
    text = text.replace("\n", " ")
    embeddings_batch_response = client.embeddings(
        model="mistral-embed",
        input=text,
    )
    return embeddings_batch_response.data[0].embedding
```

3. Question and answer function

This function is the core of the program. It processes a user's question and creates a response using the context supplied by Mistral AI.



This process involves several key steps. Here's how it works:

- Firstly, we generate a numerical representation, called an embedding, through a Mistral AI embedding endpoint, for the user's question.
- Next, we run a vector search in the MongoDB collection to identify the documents similar to the user's question.
- It then constructs a contextual background by combining chunks of text from these similar documents. We prepare an assistant instruction by combining all this information.
- The user's question and the assistant's instruction are prepared into a prompt for the Mistral AI model.
- Finally, Mistral AI will generate responses to the user thanks to the retrieval-augmented generation process.

```
1    def qna(users_question):
2        # Set up Mistral client
```

```
    api_key = os.environ["MISTRAL_API_KEY"]
    client = MistralClient(api_key=api_key)

    question_embedding = get_embedding(users_question, client)
    print("-----Here is user question------")
    print(users_question)
    documents = find_similar_documents(question_embedding)

    print("-----Retrieved documents------")
    print(documents)
    for doc in documents:
        doc['text_chunks'] = doc['text_chunks'].replace('\n', ' '

    for document in documents:
        print(str(document) + "\n")

    context = " ".join([doc["text_chunks"] for doc in documents])
    template = f"""
    You are an expert who loves to help people! Given the followi
    question using only the given context. If you are unsure and
    explicitly written in the documentation, say "Sorry, I don't

    Context sections:
    {context}

    Question:
    {users_question}

    Answer:
    """
```

```
    messages = [ChatMessage(role="user", content=template)]
    chat_response = client.chat(
        model="mistral-tiny",
        messages=messages,
    )
    formatted_documents = '\n'.join([doc['text_chunks'] for doc i

    return chat_response.choices[0].message, formatted_documents
```

The last configuration on the MongoDB vector search index

In order to run a vector search query, you only need to create a vector search index in MongoDB Atlas as follows. (You can also learn more about how to create a vector search index.)

```
{
 "type": "vectorSearch",
 "fields": [
   {
     "numDimensions": 1536,
     "path": "'embedding'",
     "similarity": "cosine",
     "type": "vector"
   }
 ]
}
```

Finding similar documents

The find_similar_documents(embedding) function runs the vector search query in a MongoDB collection. This function will be called when the user asks a question. We will use this function to find similar documents to the questions in the question and answering process.

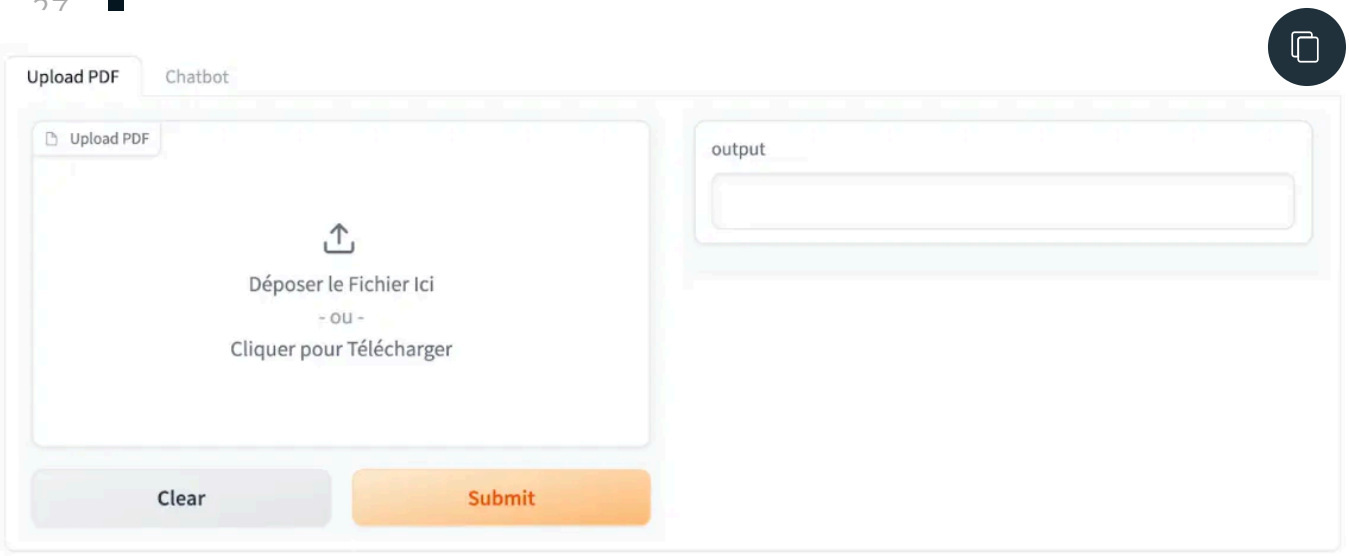
```
def find_similar_documents(embedding):
    collection = connect_mongodb()
    documents = list(
        collection.aggregate([
            {
                "$vectorSearch": {
                    "index": "vector_index",
                    "path": "embedding",
                    "queryVector": embedding,
                    "numCandidates": 20,
                    "limit": 10
                }
            },
            {"$project": {"_id": 0, "text_chunks": 1}}
        ]))
    return documents
```

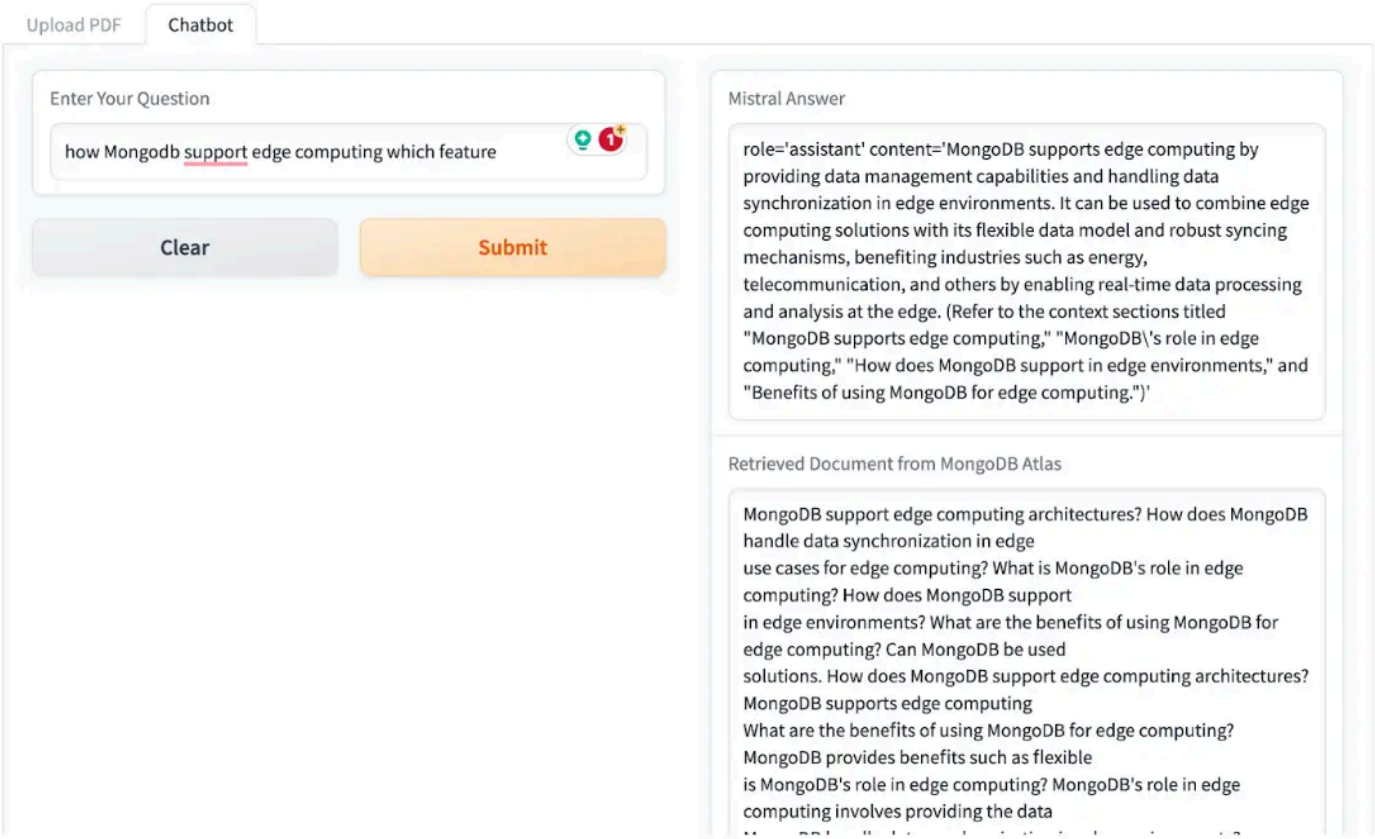
4. Gradio user interface

In order to have a better user experience, we wrap the PDF upload and chatbot into two tabs using Gradio. Gradio is a Python library that enables the fast creation of customizable web applications for machine learning models and data processing workflows. You can put this code at the end of your Python file. Inside of this function, depending on which tab you are using, either data preparation or

question and answering, we will call the explained dataprep() function or qna() function.

```
# Gradio Interface for PDF Upload
pdf_upload_interface = gr.Interface(
    fn=data_prep,
    inputs=gr.File(label="Upload PDF"),
    outputs="text",
    allow_flagging="never"
)

# Gradio Interface for Chatbot
chatbot_interface = gr.Interface(
    fn=qna,
    inputs=gr.Textbox(label="Enter Your Question"),
    outputs=[
        gr.Textbox(label="Mistral Answer"),
        gr.Textbox(label="Retrieved Documents from MongoDB Atlas"
    ],
    allow_flagging="never"
)

# Combine interfaces into tabs
iface = gr.TabbedInterface(
    [pdf_upload_interface, chatbot_interface],
    ["Upload PDF", "Chatbot"]
)

# Launch the Gradio app
```





Conclusion

This detailed guide has delved into the dynamic combination of Mistral AI and MongoDB, showcasing how to develop a bespoke large language model GenAI application. Integrating the advanced capabilities of Mistral AI with MongoDB's

robust data management features enables the creation of a custom AI assistant that caters to unique requirements.

We have provided a straightforward, step-by-step methodology, covering everything from initial data gathering and segmentation to the generation of embeddings and efficient data querying. This guide serves as a comprehensive blueprint for implementing the system, complemented by practical code examples and instructions for setting up Mistral AI on a GPU-powered machine and linking it with MongoDB.

Leveraging Mistral AI and MongoDB Atlas, users gain access to the expansive possibilities of AI applications, transforming our interaction with technology and unlocking new, secure ways to harness data insights while maintaining privacy.

Learn more

To learn more about how Atlas helps organizations integrate and operationalize GenAI and LLM data, take a look at our Embedding Generative AI whitepaper to explore RAG in more detail.

If you want to know more about how to deploy a self-hosted Mistral AI with MongoDB, you can refer to my previous articles:
Unleashing AI Sovereignty: Getting Mistral.ai 7B Model Up and Running in Less Than 10 Minutes
and
Starting Today with Mistral AI & MongoDB: A Beginner's Guide to a Self-Hosted LLM Generative AI Application
. Mixture of Experts Explained

Rate this tutorial

Related

CODE EXAMPLE

A Spotify Song and Playlist Recommendation Engine

Nov 13, 2023 | 6 min read

QUICKSTART

Getting Started with MongoDB and Starlette

Sep 23, 2022 | 5 min read

TUTORIAL

Symfony and MongoDB Workshop: Building a Rental Listing Application

Apr 08, 2024 | 3 min read

ARTICLE

Real-Time Card Fraud Solution Accelerator with MongoDB and Databricks

Jul 11, 2023 | 7 min read

Request a Tutorial

Facebook