


★ Member-only story

Talk to your files in a local RAG application using Mistral 7B, LangChain and Chroma DB (No internet needed)



Rubentak · [Follow](#)

9 min read · Oct 25

 Listen

 Share

... More

In this article I will show how you can use the Mistral 7B model on your local machine to talk to your personal files in a Chroma vector database.



Mistral 7B is a 7 billion parameter language model introduced by Mistral AI, a new company in the field of artificial intelligence. This model has garnered attention as one of the **most powerful** 7 billion parameter models available.

Mistral 7B is presented as a foundational model, signifying its significance as a core building block in natural language processing. It's part of the evolving landscape of large language models.

Mistral is a powerful language model that can be used for various applications such as:

- text classification
- sentiment analysis
- question-answering

- code completion

Although it is a strong and impressive model, the full potential of Mistral can only be realised when it is integrated with a vector database like Chroma Vector Database to create a RAG (Recallable AI Guide) application.

If you want to know more about Mistral 7B, read my former article below:

Mistral 7B : The best 7 billion parameter LLM yet.

Mistral 7B is the best open-source 7B parameter LLM to date. It can be run locally and online using Ollama.

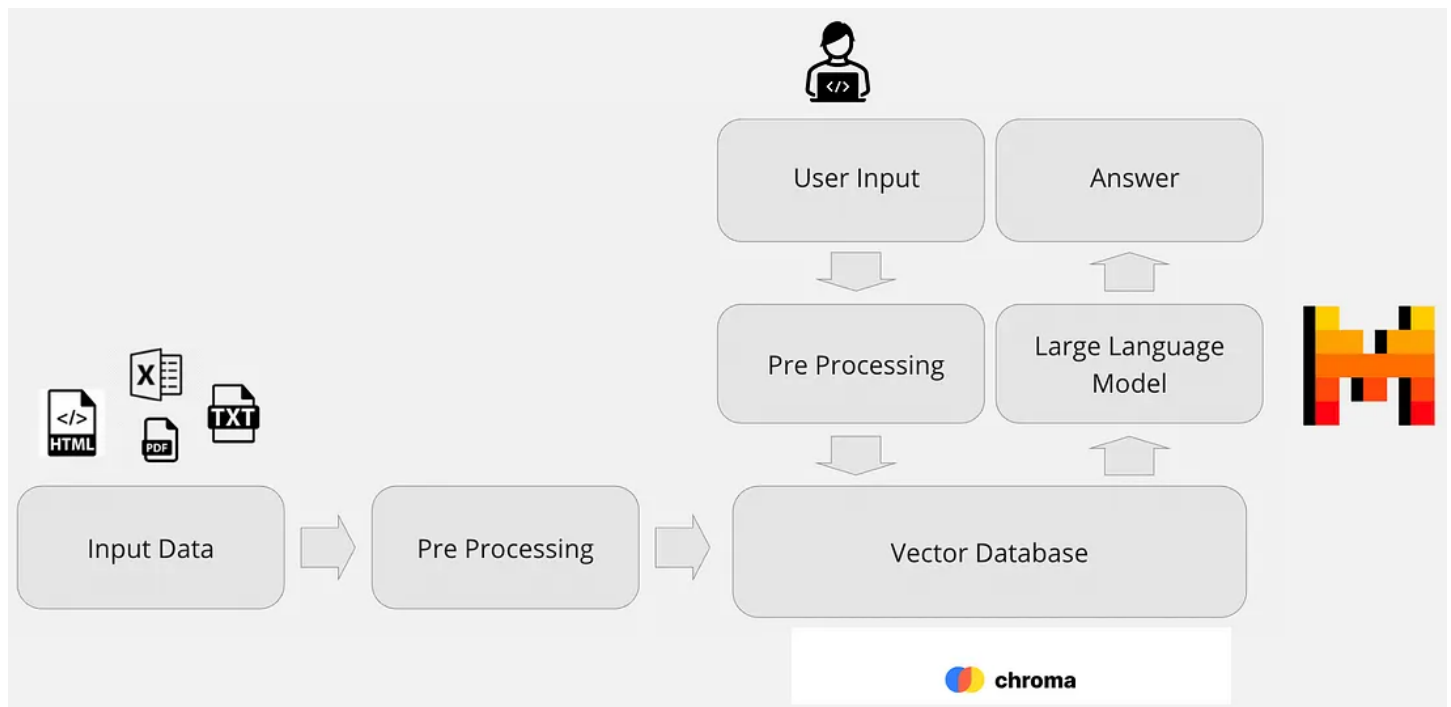
medium.com

Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) applications are a type of large language model (LLM) application that augment their generation capabilities by retrieving relevant information from an external knowledge base. This allows RAG applications to produce more informative and comprehensive responses to a wider range of prompts and questions. Below is an example of the structure of an RAG application.

To enable the local model to also have knowledge of data outside of its training data, e.g. company or research data, you can embed this data into a vector database and let an LLM retrieve the relevant documents and data. The LLM will then construct a coherent answer with the retrieved data. It enables you to connect pre-trained models to external, up-to-date information sources that can generate more accurate and more useful outputs.

In this article I will show how to build a RAG application with the Mistral 7B model and a Chroma Vector database. The architecture will be the following:



ChromaDB



Chroma

Majachkova

The specific vector database that I will use is the ChromaDB vector database.

Chroma website:

Chroma is a database for building AI applications with embeddings. It comes with everything you need to get started built in, and runs on your machine. ChromaDB

Chroma is an open-source project on [GitHub](#). This open-source database is specifically designed to power artificial intelligence applications. The repository has 'just' 6.4K stars currently, which does not seem like a lot. However, in early April 2023, Chroma Inc. announced that it had raised **\$18 million** in seed funding, showing that investors see great potential in the service. As it is free, local, very easy to use, and also has an in-memory mode to speed things up even more, I recommend using it. Other embedding databases are [Pinecone](#), [Milvus](#) or [Weaviate](#).

Ollama

If you want to want to use the Mistral 7B locally on your own machine, you can use Ollama. Ollama is a lightweight, extensible framework for building and running language models on the local machine. It provides a simple API for creating, running, and managing models, as well as a library of pre-built models that can be easily used in a variety of applications. You can download Ollama via the following link:

Download Ollama on macOS

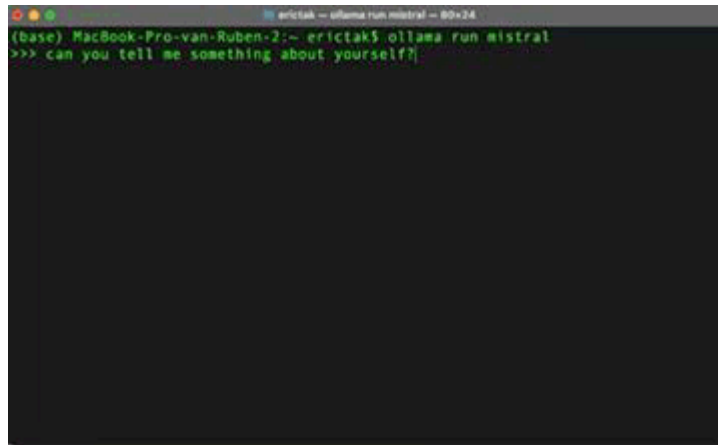
Get up and running with large language models, locally.

ollama.ai

After downloading Ollama, you can run the following command in your terminal:

```
ollama run mistral
```

This will start a chat and you can ask Mistral 7B questions without any data leaving your computer to see if it works.



The code for the RAG application using Mistral 7B and Chroma can be found in my GitHub repository [here](#).

Quickstart

Let us start by importing the necessary libraries:

```
# Import libraries
import os
from langchain.vectorstores import Chroma
from langchain.embeddings import OpenAIEmbeddings
from langchain.llms import Ollama
from langchain.callbacks.manager import CallbackManager
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
from langchain.embeddings import OllamaEmbeddings
from langchain.prompts import PromptTemplate
from langchain.document_loaders import UnstructuredPDFLoader, OnlinePDFLoader,
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.document_loaders import DirectoryLoader
from langchain.chains import RetrievalQA
```

Set your OpenAI API key if you want to use the OpenAI embeddings model.

```
os.environ["OPENAI_API_KEY"] = "sk-..."
```

Set your LLM and embedding model. For this notebook I will use the Mistral 7B model and the Ollama embeddings. You can also use the OpenAI embeddings.

```
# Ollama embeddings
embeddings_open = OllamaEmbeddings(model="mistral")
# OpenAI embeddings
#embedding = OpenAIEmbeddings()

llm_open = Ollama( model="mistral",
                  #model='Llama2',
                  callback_manager = CallbackManager([StreamingStdOutCallback
```

Load the data from the directory. There are multiple ways to load a file or multiple files from your directory. I loaded the LangChain documentation.

[Open in app](#)



Search



```
# Print number of txt files in directory
loader = DirectoryLoader('.../data/langchain_doc_small', glob="/*.txt")

# load pdfs from directory and print number of pdfs
loader = PyPDFLoader('.../data/PDFs/How_to_build_your_career_in_AI.pdf')

# load another file directly
loader = DirectoryLoader('/your/path/to/file.txt')

doc = loader.load ( )
len(doc)
```

After running this cell, you get returned the number of documents in that directory.

Print the first document to see if it is loaded correctly:

```
print(doc[0])
```

Split the documents into chunks before embeddign them in your vector database. I choose for chunks of 500 tokens. You can change this to fit your specific usecase.

```
text_splitter = RecursiveCharacterTextSplitter (chunk_size=500, chunk_overlap=50)
texts = text_splitter.split_documents(doc)
```

When counting the number of chunks after splitting them, you will see that the number increased:

```
len(texts)
```

I had two different directories that i wanted to embed in separate databases: some PDFs and a part of the LangChain documentation. In this article I loaded the text files and created a database from these files named 'vdb_langchain_doc_small'. You can choose a name and your own database. You can also choose your own embeddings again.

```
# PDFs from directory
#persist_directory = 'PDFs_How_to_build_your_career_in_AI'

# Langchain documentation
persist_directory = 'vdb_langchain_doc_small'

# Your documents
#persist_directory = 'your_new_database'

vectoradb = Chroma.from_documents(documents=texts,

                                   # Chose the embedding you want to use
                                   # embedding=embeddings_open,
                                   embedding=embeddings_open,

                                   persist_directory=persist_directory)
```

Persist database to your disc to save it.

```
vectoradb.persist()
```



```
vectordb = None
```

After saving, you can choose a persistence directory and load from disk

```
# LangChain documentation from directory
persist_directory = 'vdb_langchain_doc_small'
```

Now we can load the persisted database from disk, and use it as normal. emember to choose the same embeddign model as before.

```
vectordb = Chroma(persist_directory=persist_directory,
                  embedding_function=embeddings_open
                  #embedding_function=embeddings_open
                  )
```

Now it is time to create a retriever to see the relevent documents that get returned from a your vector database based on a question:

```
retriever = vectordb.as_retriever()
###
docs = retriever.get_relevant_documents("What is this document about?")
```

After running:

```
docs
len(docs)
```

4 text chunks from the original document are returned based on the question that was asked.

If you want your RAG to also state the sources of the chunks that were returned, you can use the following function:

```
def process_llm_response(llm_response):  
    print(llm_response['result'])  
    print('\n\nSources:')  
    for source in llm_response["source_documents"]:  
        print(source.metadata['source'])
```

Create a Q&A chain:

```
qa_chain = RetrievalQA.from_chain_type(llm=llm_open,  
                                       chain_type="stuff",  
                                       retriever=retriever,  
                                       return_source_documents=True,  
                                       verbose=True)
```

Now we can ask questions, and the retrieved chunks will be used by the Mistral 7B model to make a coherent answer:

```
# Question  
query = "What is this document about?"  
llm_response = qa_chain(query)  
process_llm_response(llm_response)
```

Answer: This document appears to be a Quickstart Guide for getting started with a software product or service. It contains sections on Getting Started, Modules, Use Cases, Reference Docs, and Ecosystem, as well as Additional Resources.

This was indeed a quickstart guide for LangChain.

You can improve the responses from your model by using a prompt template:

```
def build_prompt(template_num="template_1"):
    template = """ You are a helpful chatbot, named RSLT. You answer the question.
    Do not say anything that is not in the website.
    You are to act as though you're having a conversation with a human.
    You are only able to answer questions, guide and assist, and provide recommendations.
    Your tone should be professional and friendly.
    Your purpose is to answer questions people might have, however if the question is unclear, ask for clarification.
    Your responses should always be one paragraph long or less.
    Context: {context}
    Question: {question}
    Helpful Answer: """

    template2 = """You are a helpful chatbot, named RSLT. You answer the question.
    Your responses should always be one paragraph long or less.
    Question: {question}
    Helpful Answer: """

    if template_num == "template_1":
        prompt = PromptTemplate(input_variables=["context", "question"], template=template)
        return prompt

    elif template_num == "template_2":
        prompt = PromptTemplate(input_variables=["question"], template=template2)
        return prompt

    else:
        print("Please choose a valid template")
```

Enhance the Q&A chain:

```
qa_chain = RetrievalQA.from_chain_type(llm=llm_open,
                                       chain_type="stuff",
                                       retriever=retriever,
                                       return_source_documents=True,
                                       verbose=True,
                                       chain_type_kwargs={"prompt": build_prompt("template_1")})
```

When asking the same question:

This document appears to be a guide for users who want to quickly get started with a specific software or system. The guide is divided into several sections, including Getting

Started, Modules, Use Cases, Reference Docs, Ecosystem, and Additional Resources. It seems that the guide provides an overview of the features and functionalities of the software, as well as some practical examples of how to use it in different contexts. The author of the guide is Harrison Chase, and it was last updated on June 14, 2023.

Sources:

`../data/langchain_doc_small/2_Quickstart_Guide_Contents.txt`

`../data/langchain_doc_small/8_Getting_Started_Getting.txt`

`../data/langchain_doc_small/7_LLMs_LLMs_Note.txt`

`../data/langchain_doc_small/0_Welcome_to_LangChain.txt`

We can see that the answer improved by using a prompt template.

You can run this whole notebook again for other files and create another database. After doing this, you can always reconnect to a previously saved database to talk with your files.

You can switch between databases whenever you want.

End note

This article on Mistral 7B provides a basic understanding of how to make a RAG application works and how you can use it. Mistral 7B represents a powerful and efficient 7 billion-parameter language model with unique architecture and impressive performance across a wide range of language tasks. It offers versatility and potential for fine-tuning, making it a valuable tool for developers, researchers, and organisations seeking advanced language generation and understanding capabilities. However, its potential uncensored nature requires responsible use and monitoring when integrating it into applications. This serves as the basis for more advanced capabilities that we will discuss in the future.


Thank you for reading, if you liked the article please leave a clap  and leave a comment if you have any feedback or thoughts that you would like to share!

Illustration :

The wonderful illustration is made by UX designer and Researcher [MalachkovaK](#). Be sure to check out her Medium and Instagram and contact her for more beautiful designs!

- <https://medium.com/@malachkovak>

- https://instagram.com/ph_malachkova?igshid=ZjE2NGZiNDQ=
- <https://katmalachkova.framer.website/>

RSLT agency

I recently founded a Data and AI consultancy RSLT agency. If you have any ideas for your business on how you would like to make use of this advancing technology and outcompete your competitors, contact me. We are here to empower you with the possibilities that AI creates.

- <https://rslt.agency/>
- <https://www.linkedin.com/company/96695171/admin/feed/posts/>

<div><div>RSLT Agency — Let us turn complexity into opportunity!</div><div>rslt.agency</div></div>	
---	--

Also, be sure to follow me on GitHub and LinkedIn. If you like the work that I’m doing you could buy me a coffee:

- <https://github.com/rubentak>
- <https://www.linkedin.com/in/ruben-tak-665b66194/>
- <https://www.buymeacoffee.com/rubentak>

References and Additional Sources 🗉:

<div><div>Mistral7B/mistral_langchain_RAG.ipynb at main · rubentak/Mistral7B</div><div>This is a repository where I show how to use Mistral 7B - Mistral7B/mistral_langchain_RAG.ipynb at main · ...</div><div>github.com</div></div>	
--	--

<div><div>Mistral 7B</div></div>	
---	--