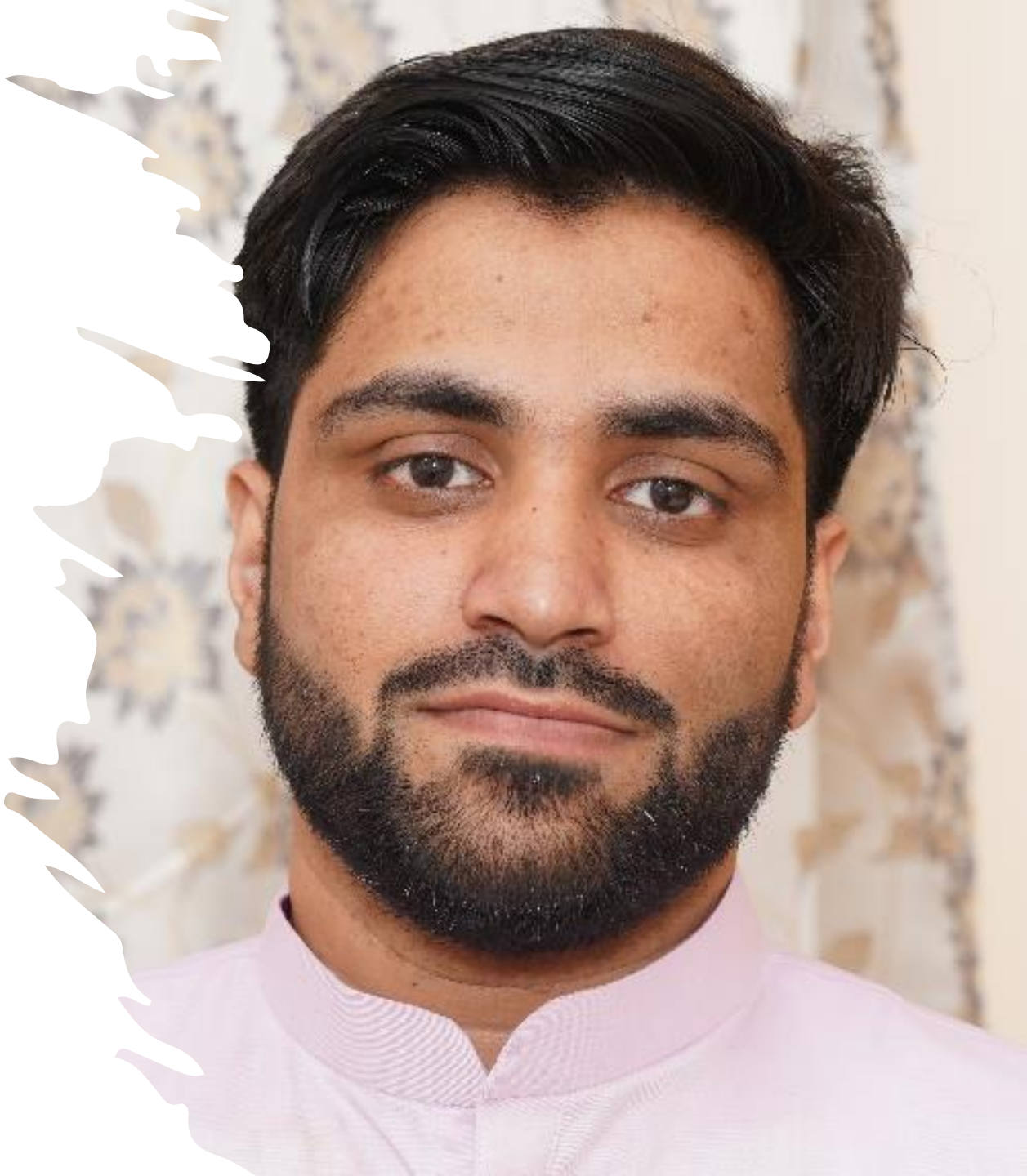
Retreival Augmented Generation from web data

Md Khalid Siddiqui

Introduction



Md Khalid Siddiqui

- Masters Student
- Technology and Innovation Management
- Hochschule Harz
- Faculty of Automation and Computer Science

Interest domain:

Large Language Model Applications, Prompt Engineering, Deep Learning particularly NLP

Completed Projects:

- RAG using web data (Current Presentation)
- Deep Learning Model Deployment on Hugging face using Gradio(model generation and deployment)
- Prompt Engineering chat bot project on Huggingface

This notebook makes a question answering chain with a specified website as a context data.

Setting up

Install dependencies

[1]
```
!pip install langchain
!pip install pinecone-client
!pip install openai
!pip install tiktoken
!pip install nest_asyncio
```

Set up OpenAI API key

```
[2]
import os
os.environ["OPENAI_API_KEY"] = "sk-0eXpSaxE[hidden]3u"
```

Set up Pinecone API keys

```
import pinecone

# initialize pinecone
pinecone.init(
    api_key="78f1[hidden]find at app.pinecone.io
    environment="asia-southeast1-gcp-free"  # next to api key in console
)
```

Index

Load data from Web

Extends from the WebBaseLoader, this will load a sitemap from a given URL, and then scrape and load all the pages in the sitemap, returning each page as a document.

The scraping is done concurrently, using WebBaseLoader. There are reasonable limits to concurrent requests, defaulting to 2 per second.

Link to the documentation

```
!pip install unstructured
```

+ Code + Markdown

```
from langchain.document_loaders import UnstructuredURLLoader
```

```
urls = [
    "https://www.naeco.blue/"
]
```

```
loader = UnstructuredURLLoader(urls=urls)
```

```
data = loader.load()
```

documentation | 🦜🔗 Langchain

Split the text from docs into smaller chunks

There are many ways to split the text. We are using the text splitter that is recommended for generic texts. For more ways to slit the text check the documentation

[32]
```
len(data)
```

```
1
```

+ Code + Markdown

[33]
```
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 1000,
    chunk_overlap  = 100,
    length_function = len,
)

docs_chunks = text_splitter.split_documents(data)
```

[34]
```
len(docs_chunks)
```

```
2
```

Document transformers | 🦜🔗 Langchain

Create embeddings

```
[36]
from langchain.embeddings.openai import OpenAIEmbeddings

embeddings = OpenAIEmbeddings()
```

Creating a vectorstore

A vectorstore stores Documents and associated embeddings, and provides fast ways to look up relevant Documents by embeddings.

There are many ways to create a vectorstore. We are going to use Pinecone. For other types of vectorstores visit the documentation

First you need to go to Pinecone and create an index there. Then type the index name in "index_name"

```
[37]
from langchain.vectorstores import Pinecone

index_name = "hsharzirina"

# #create a new index
docsearch = Pinecone.from_documents(docs_chunks, embeddings, index_name=index_name)

# if you already have an index, you can load it like this
#docsearch = Pinecone.from_existing_index(index_name, embeddings)
```

Vector stores | 🦜🔗 Langchain

Vector Database for Vector Search | Pinecone

Vectorstore is ready. Let's try to query our docsearch with similarity search

```
[39]
    query = " Who is Felix Mertens?"
    data = docsearch.similarity_search(query)
    print(data)
```

```
...  [Document(page_content='Business Case\n\nProblem\n\nVolatilität der erneuerbaren Ene
```

Making a question answering chain

The question answering chain will enable us to generate the answer based on the relevant context chunks. See the documentation for more explanation.

Additionally, we can return the source documents used to answer the question by specifying an optional parameter when constructing the chain. For more information visit the documentation

[40]
```
from langchain.chains import RetrievalQA
from langchain.llms import OpenAI
llm=OpenAI()

qa_with_sources = RetrievalQA.from_chain_type(llm=llm, chain_type="stuff", retriever=docsearch.as_retriever(), return_source_documents=True)

query = "Who is Felix Mertens?"
result = qa_with_sources({"query": query})
result["result"]
```

```
' Felix Mertens is the founder and CEO of NAECO Blue Management.'
```

Stuff | 🦜🔗 Langchain

QA using a Retriever | 🦜🔗 Langchain

Output source documents that were found for the query

```
[41]
result["source_documents"]
```

```
[Document(page_content='Business Case\n\nProblem\n\nVolatilität der erneu
 Document(page_content='English\n\nDeutsch\n\nNEWSNEWSCareer\n\nLegalLega
```

```
[42]
query = "What does Naeco Blue do?"
result = qa_with_sources({"query": query})
result["result"]
```

```
... ' Naeco Blue develops digital solutions that aim to minimi
```

Thankyou

Md Khalid Siddiqui
Questions : academia.mdkhalid@gmail.com
Contact: +491635346918