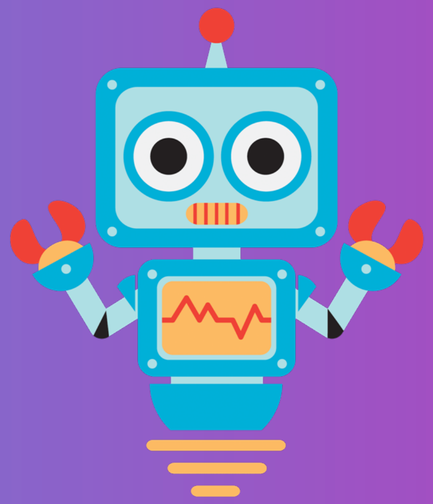
Dr. Abulkarim Albanna

Chat with your documents App

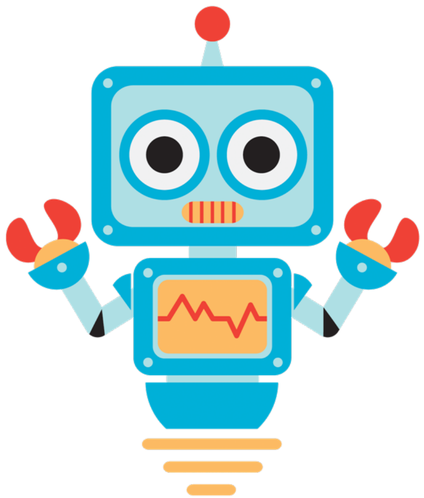


Day 6

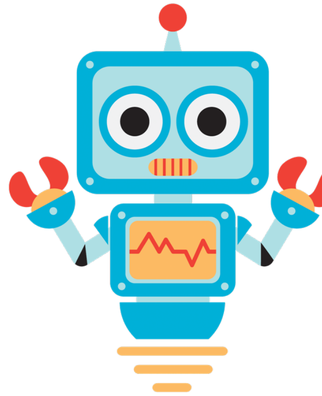
January 28th 2024

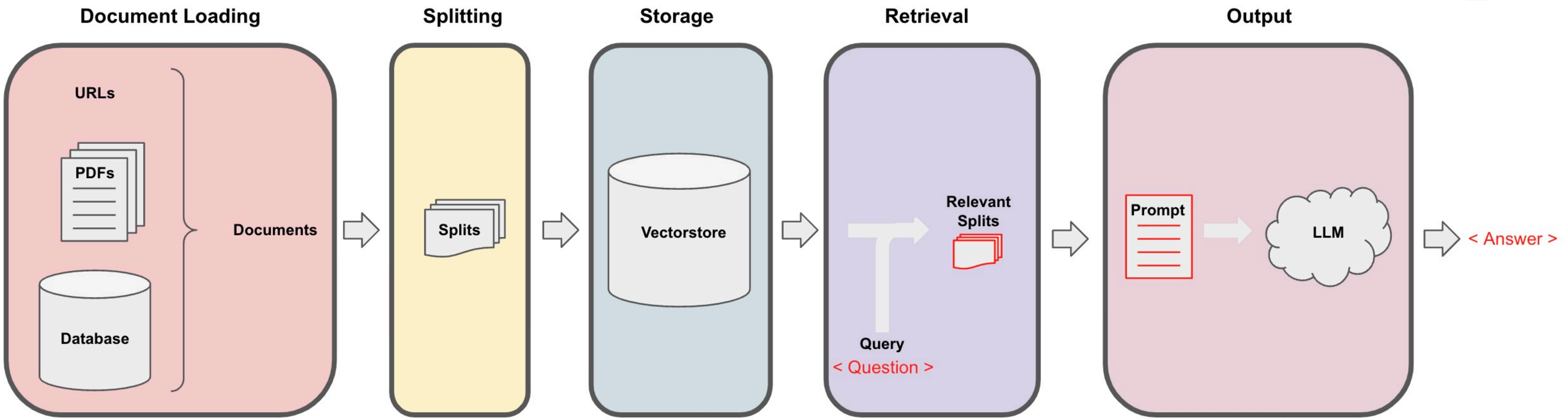
Content

- Storage
- Similarity Search
- Retrieval
- Maximum Marginal Relevance(MMR)
- Contextual Compression
- RetrievalQA Chain



Chat with your documents

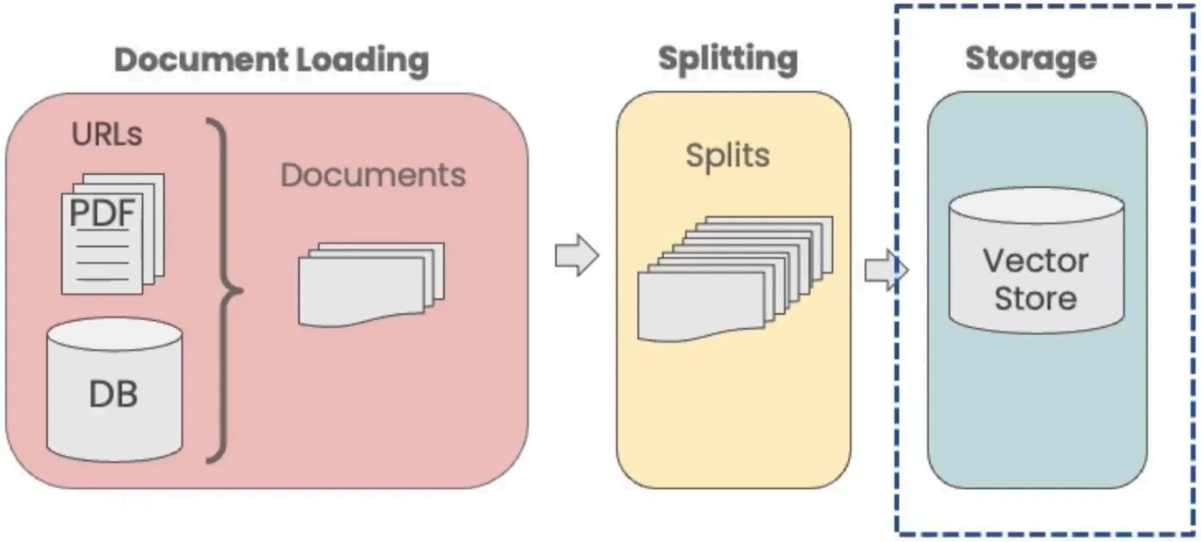




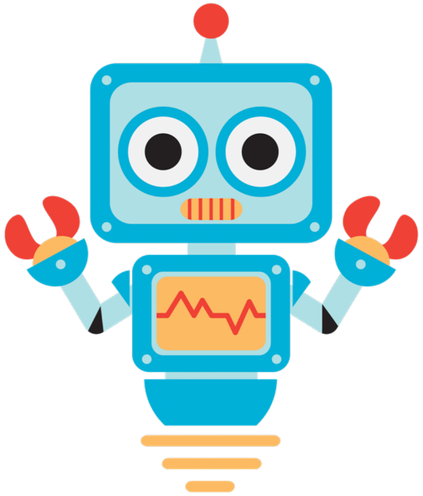
In retrieval augmented generation (RAG) framework, an LLM retrieves contextual documents from an external dataset as part of its execution.

Storage

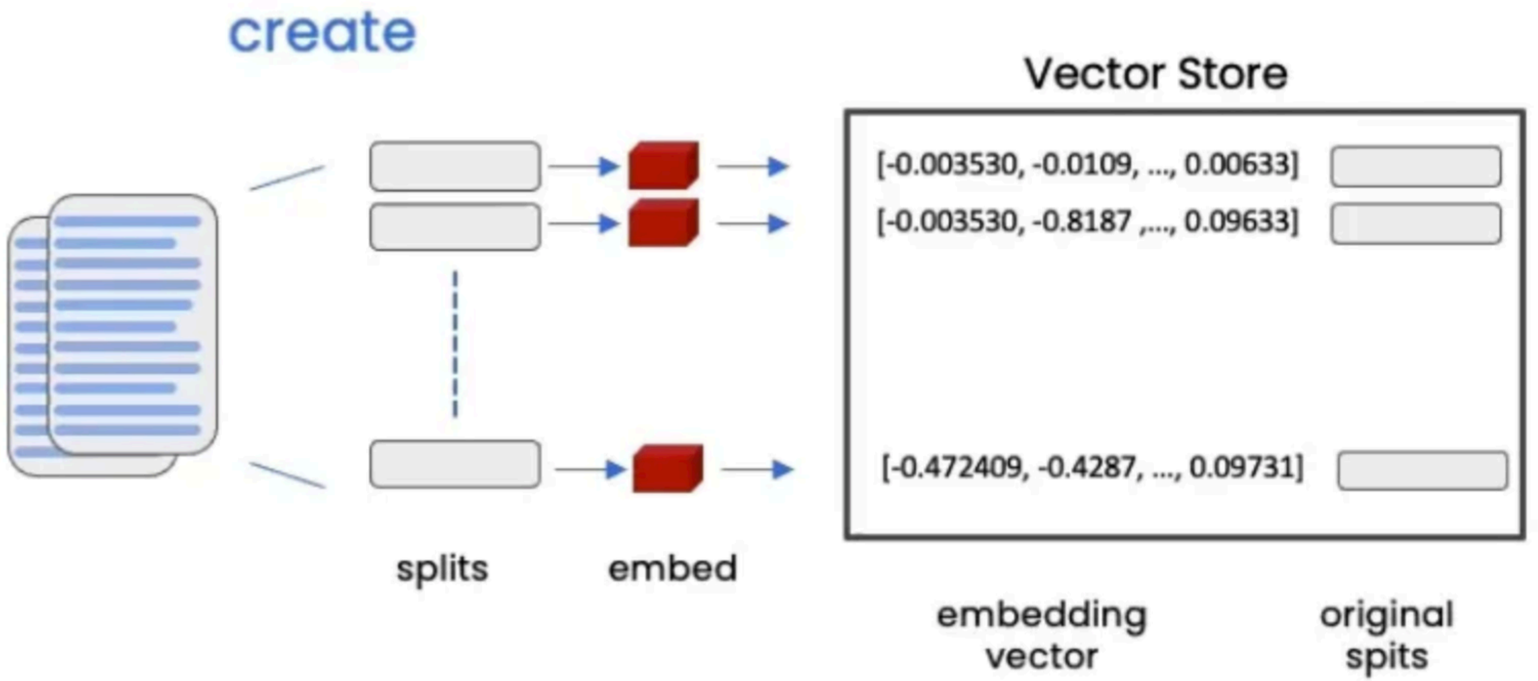
We split up our document into small chunks and now we need to put these chunks into an index so that we are able to retrieve them easily when we want to answer questions on this document. We use **embeddings and vector stores** for this purpose.

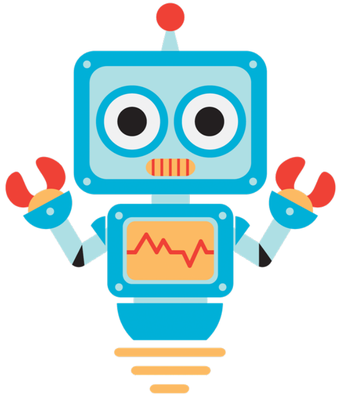


we store all these embeddings in a vector store.



Storage


create
Vector Store
[-0.003530, -0.0109, ..., 0.00633]
[-0.003530, -0.8187 ,..., 0.09633]
[-0.472409, -0.4287, ..., 0.09731]
splits
embed
embedding vector
original spits



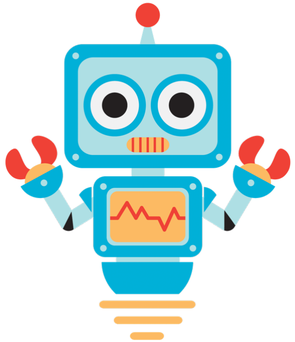
Storage

```
loaders=[
    PyPDFLoader('docs/security.pdf'),
    PyPDFLoader('docs/spec.pdf'),
]

docs=[]
for loader in loaders:
    docs.extend(loader.load())

r_splitter = RecursiveCharacterTextSplitter(
    chunk_size=200,
    chunk_overlap=50,
    separators=["\n\n", "\n", " ", ""]
)

#
# #Create a split of the document using the text splitter
splits = r_splitter.split_documents(docs)
#
```

```
import pinecone
from langchain.vectorstores import Pinecone

pinecone.init(
    api_key="api_key",
    environment="gcp-starter"

)
os.environ["OPENAI_API_KEY"] = 'api_key'

embedding = OpenAIEmbeddings(model="text-embedding-ada-002")
vectorstore = Pinecone.from_existing_index("test", embedding)
```

```
from langchain.vectorstores import Chroma
from langchain.embeddings import OpenAIEmbeddings
embedding = OpenAIEmbeddings(model="text-embedding-ada-002")

persist_directory = 'docs/chroma/'
#
# Create the vector store
vectordb = Chroma.from_documents(
    documents=splits,
    embedding=embedding,
    persist_directory=persist_directory
)

print(vectordb._collection.count())
```

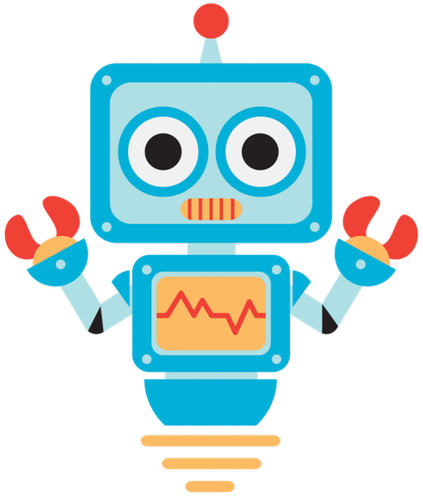
```
search=Pinecone.from_documents(splits, embedding, index_name="test")
```



Similarity Search

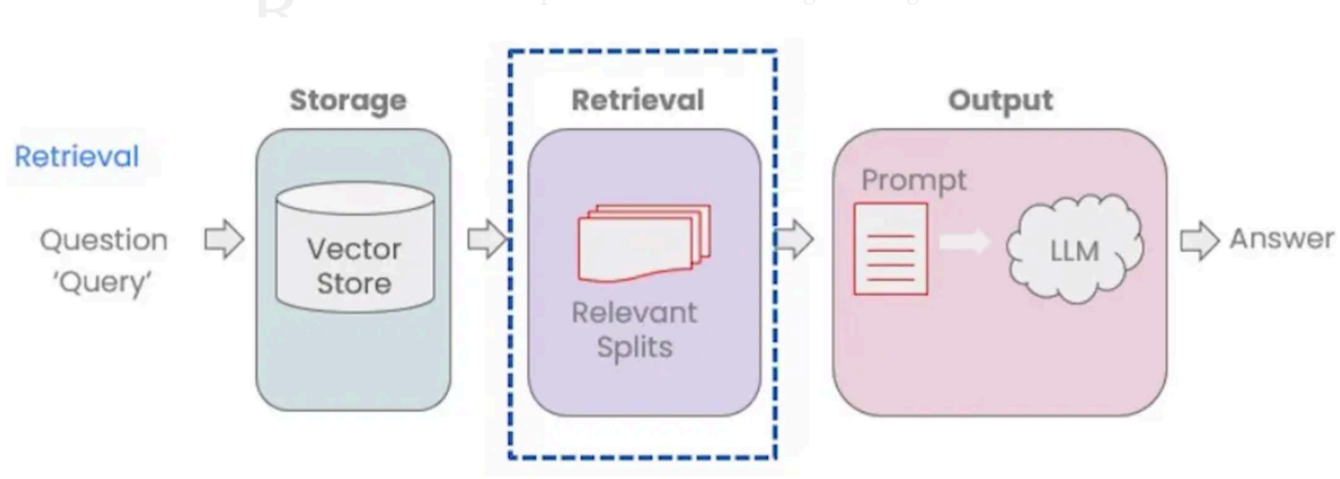
We will now ask questions using the similarity search method and pass **k**, which specifies the number of documents that we want to return.

```
84  question = "Describe the dataset proposed in this paper"
85
86  # Similarity search with k = 5
87  docs = vectordb.similarity_search(question, k=5)
88
89  # Check for first two results
90  print(docs[0])
91  print(docs[1])
```

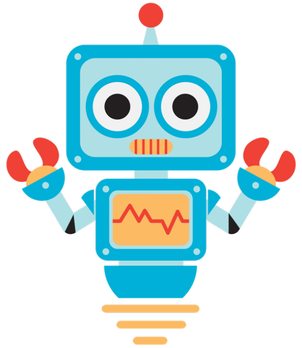


Retrieval

Retrieval is important at query time when a query comes in and we want to retrieve the most **relevant splits.**



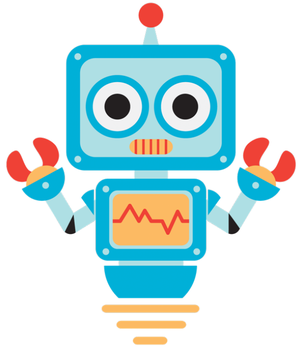
1. Accessing/indexing data in the vector store
    - Basic semantic similarity
    - Maximum Marginal Relevance
    - Including Metadata
2. LLM-aided retrieval
3. Contextual Compression



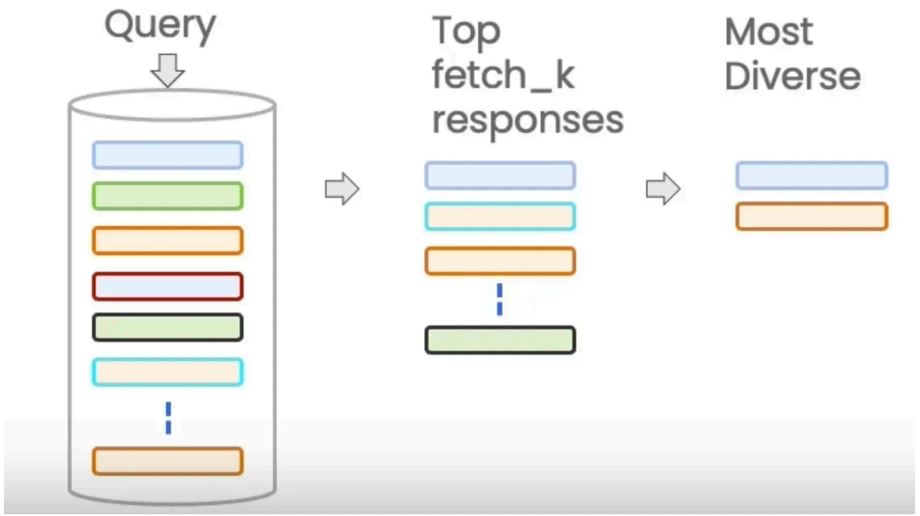
Maximum Marginal Relevance(MMR)

MMR is an important method to **enforce diversity in the search results**.
In the case of semantic search, we get documents that are most similar to the query in the embedding space and we may miss out on diverse information.

For example, if the query is “Tell me about all-white mushrooms with large fruiting bodies”, we get the first two most similar results in the first two documents with information similar to the query about a fruiting body and being all-white. But we miss out on information that is important but not similar to the first two documents. Here, MMR helps to solve this problem as it helps to select a diverse set of documents.



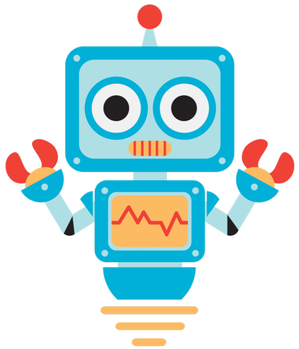
Maximum Marginal Relevance(MMR)




```
texts = [
    """The Amanita phalloides has a large and imposing epigeous (aboveground) fr
    """A mushroom with a large fruiting body is the Amanita phalloides. Some var
    """A. phalloides, a.k.a Death Cap, is one of the most poisonous of all known
]

smalldb = Chroma.from_texts(texts, embedding=embedding)
question = "Tell me about all-white mushrooms with large fruiting bodies"
smalldb.max_marginal_relevance_search(question,k=2, fetch_k=3)
```

The **“fetch_k”** most similar responses. Now, we work on this smaller set of **“fetch_k”** documents and optimize to achieve both relevance to the query and diversity among the results. Finally, we choose the **“k”** most diverse response within these “fetch_k”.

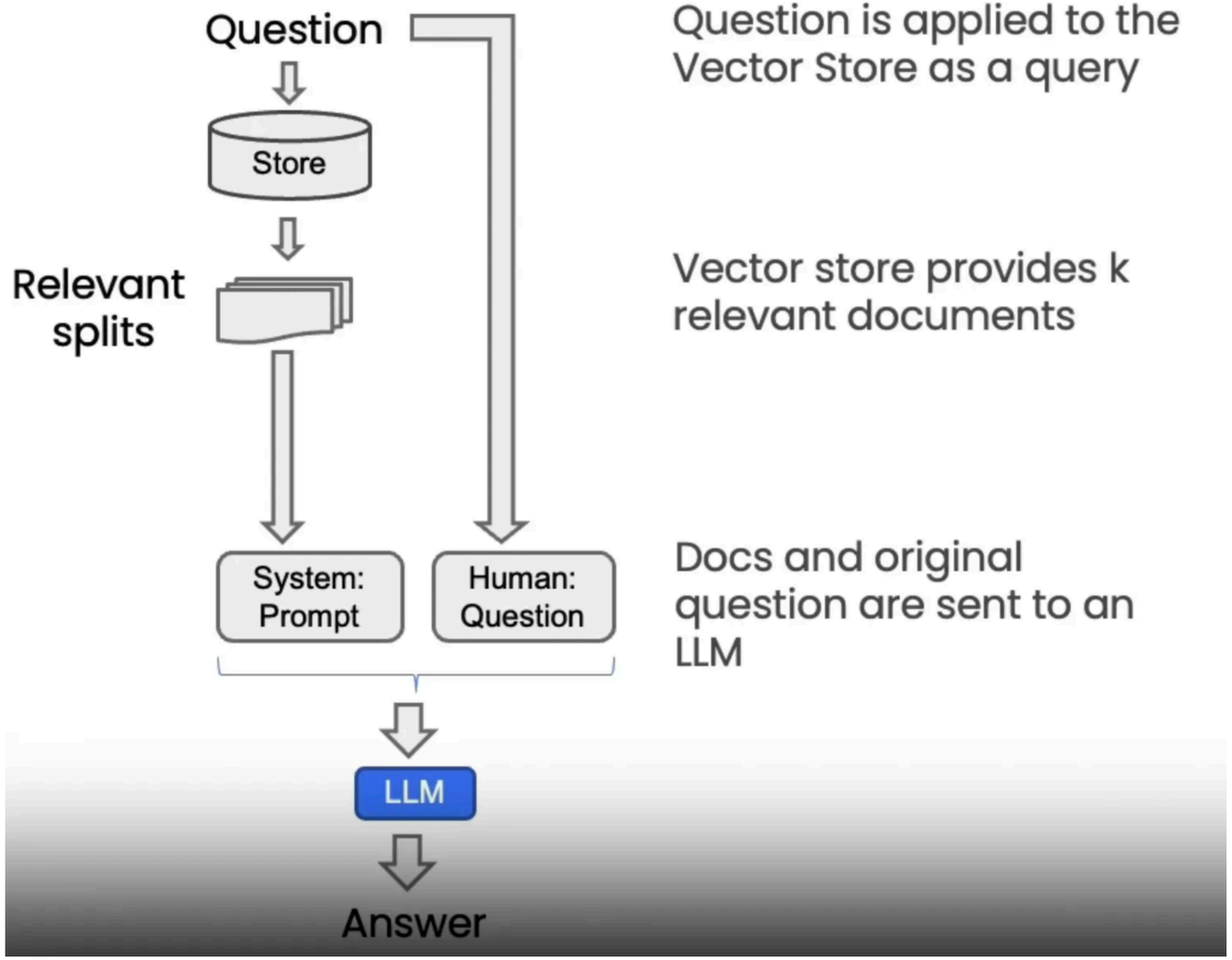
Contextual Compression



Compression is another approach to improve the quality of retrieved docs. Since passing the full document through the application can lead to more expensive LLM calls and poorer response, it is useful to pull out only the most relevant bits of the retrieved passages.

```
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import LLMChainExtractor

# Wrap our vectorstore
llm = OpenAI(temperature=0)
compressor = LLMChainExtractor.from_llm(llm)

compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=vectordb.as_retriever()
)

question = "what did they say about matlab?"
compressed_docs = compression_retriever.get_relevant_documents(question)
pretty_print_docs(compressed_docs)
```

RetrievalQA Chain

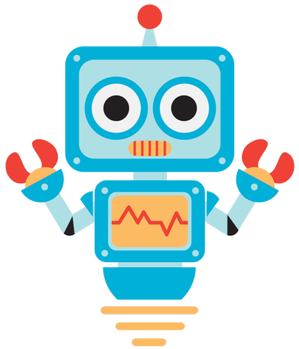


Question is applied to the Vector Store as a query

Vector store provides k relevant documents

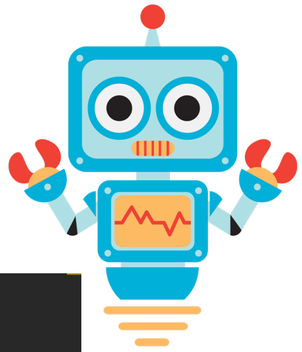
Docs and original question are sent to an LLM

RetrievalQA Chain



```
32  llm = ChatOpenAI(temperature=0.9, model_name="gpt-4")
33  qa_chain = RetrievalQA.from_chain_type(
34      llm,
35      retriever=retriever,
36      return_source_documents=True
37  )
38
39  result = qa_chain({"query": question})
40  print(result["result"])
41
```

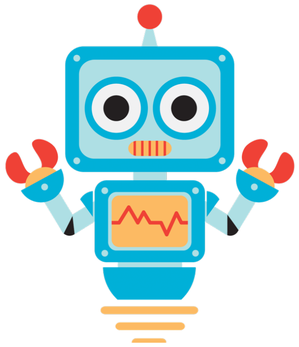
QA chain with Prompt



```
1 usage
def retrieval(question):
    template = """/
    your role is bid manager,you will extract response  from given context given in docs/spec.pdf source document, you are creative persona and expert in suadi laws for bids
     answer in arabic language the given question in professional bussiness
    language  start your asnwer with شكرا على السؤال  وفقا لنموذج كراسة الشروط و المواصفات  if you  don't know answer " عذرا لا يوجد نظام المنافسة في نموذج كراسة الشروط و المواصفات

    Question: {question}
    context:{context}
    Answer

    """

    PROMPT = PromptTemplate(template=template, input_variables=['question', 'context'])

    llm = ChatOpenAI(temperature=0, model_name="gpt-4")
```

```
47      qa_chain = RetrievalQA.from_chain_type(
48          llm,
49          chain_type="stuff",
50          retriever=load_vector_store(),
51          # return_source_documents=True
52          chain_type_kwargs={"prompt": PROMPT}
53      )
54      ask = qa_chain({"query": question})
55      return ask
56
```

QA chain with Prompt



Chat with your document

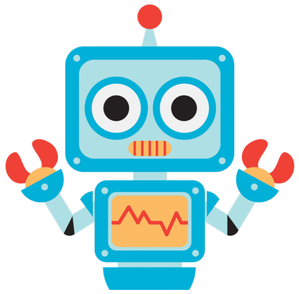
Question

علي ماذا تحتوي الكراسة

Response

شكرا على السؤال، وفقا لنموذج كراسة الشروط و المواصفات، الكراسة تحتوي على معلومات حول السلامة ووسائل الصحة والرعاية، والنقل ووسائل الصحة السكنية. كما تتضمن مرحلة التأهيل المسبق في استخدامها في المتنافس، وذلك بتأهيل القيام عدم لها فيجوز للمتنافس السابق بتأهيل الحكومية الجهة قامت حال في. بالإضافة إلى ذلك، تتضمن الكراسة معلومات حول التمييز بين المتنافسين والتزام الجهة بتقديم العروض في موعد محدد قبل تنفيذها. وأخيرا، تتضمن الكراسة معلومات إضافية حول الجهة الحكومية مع مفاوضات بأي المرتبطة التكاليف إلى بالإضافة للجهة، والمعلومات المتعلقة بتقديم العمل.

QA chain with Prompt



Chat with your document

Question

ما هو الاستبعاد من المنافسة

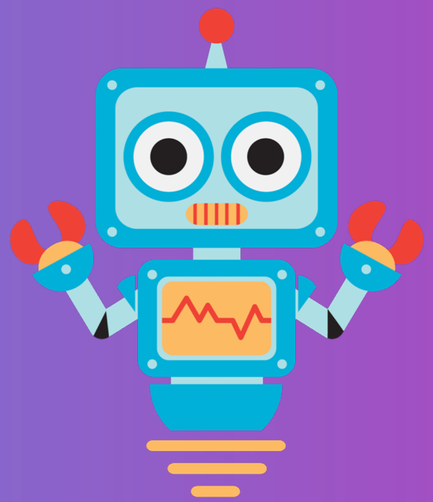
Response

شكرا على السؤال، وفقا لنموذج كراسة الشروط و المواصفات، الاستبعاد من المنافسة يعني أن الجهة الحكومية قد تقرر استبعاد متقدم معين من المنافسة بناءً على معايير محددة. قد تكون هذه المعايير تتعلق بالجودة، المواصفات، الشروط، أو أي معايير أخرى محددة في كراسة الشروط والمواصفات. إذا لم يتمكن المتقدم من تلبية هذه المتطلبات، فقد يتم استبعاده من المنافسة.

Thank you for your attention

Day 6

January 23th 2024



GAME Khoot