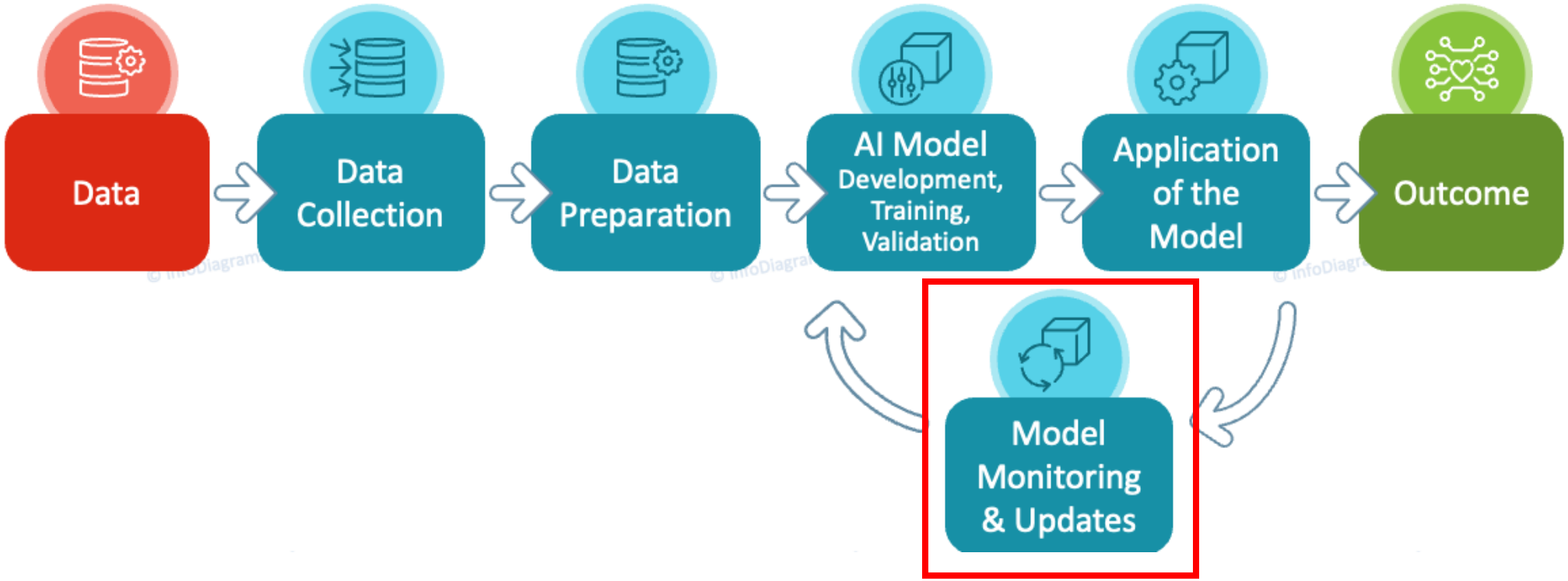
IMEN891K/IDSC501 Data Engineering

18. **LLM Systems II**: RAG

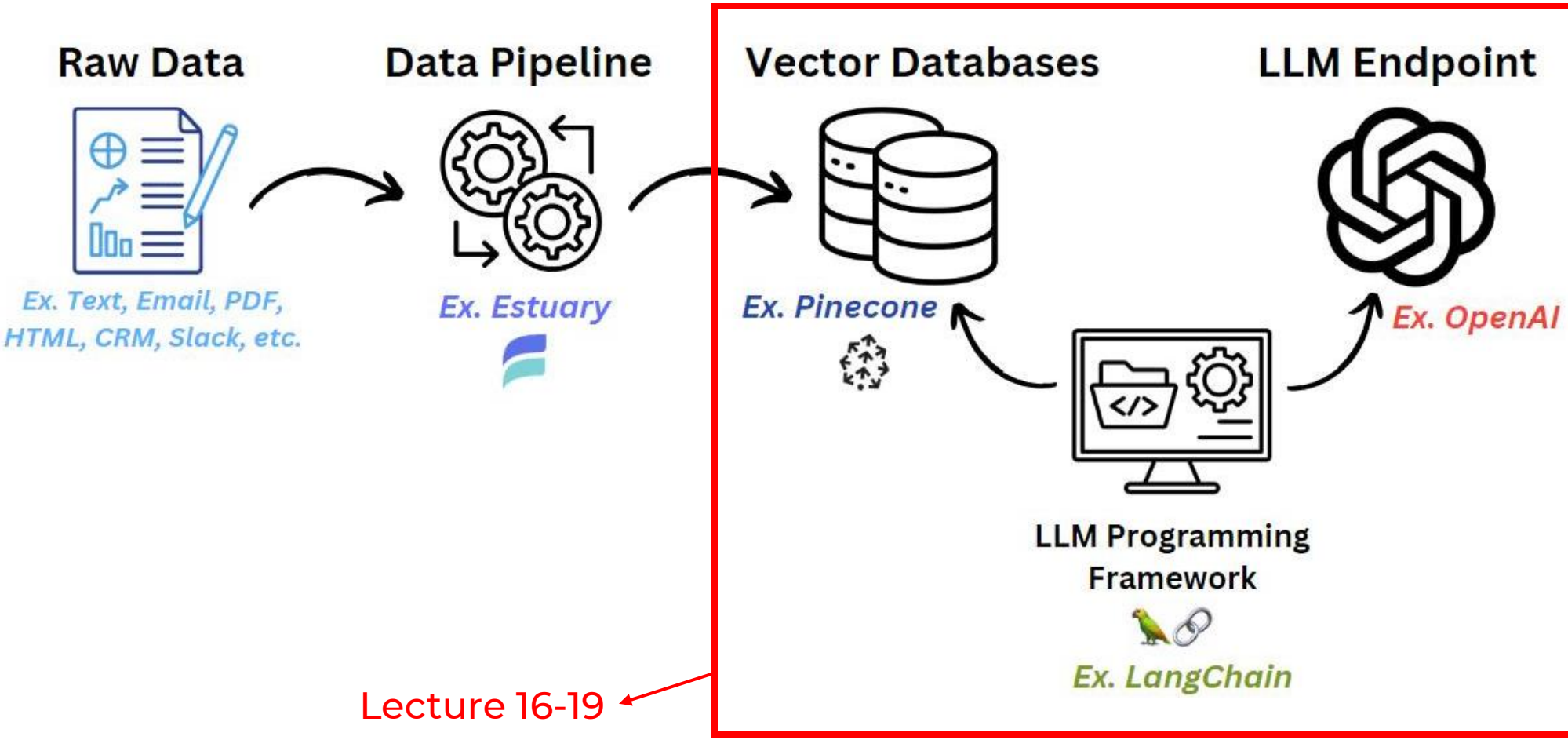
Fall 2025

Instructor: **Sangwoo Mo**

Recap: Course Overview

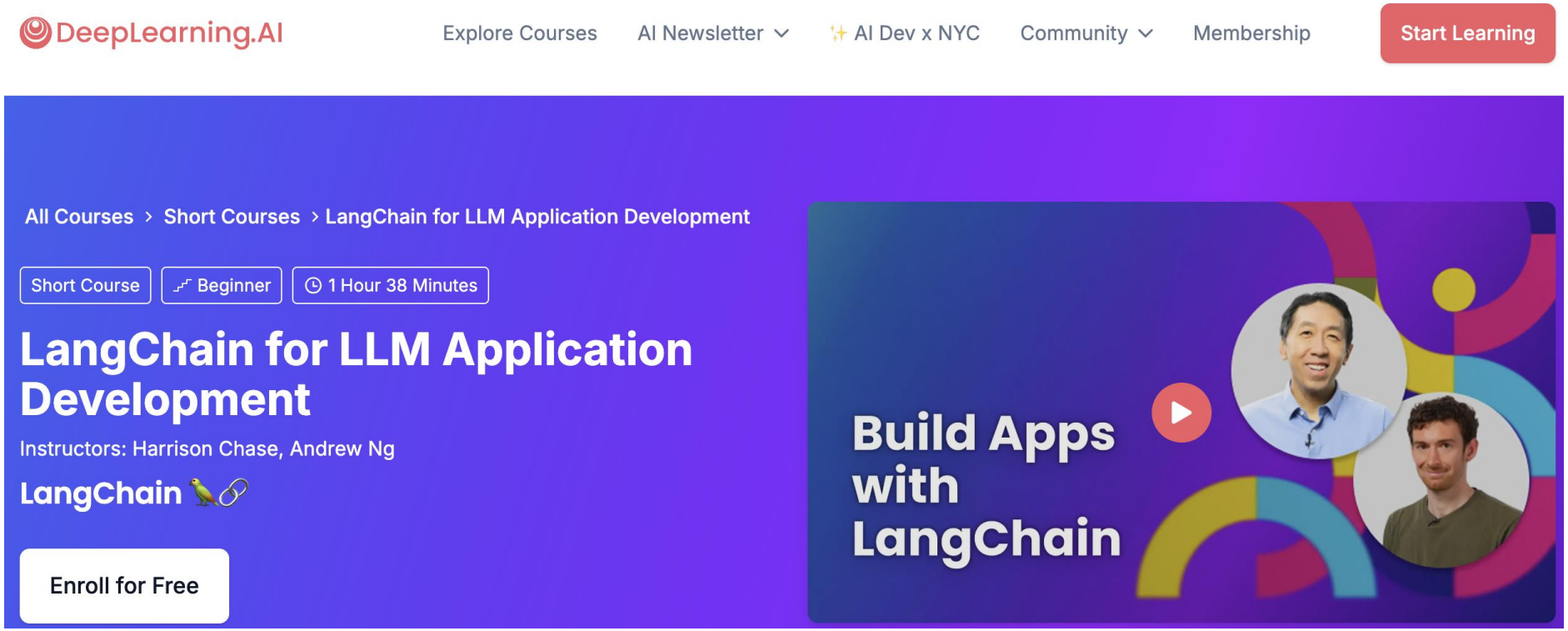


Recap: Course Overview

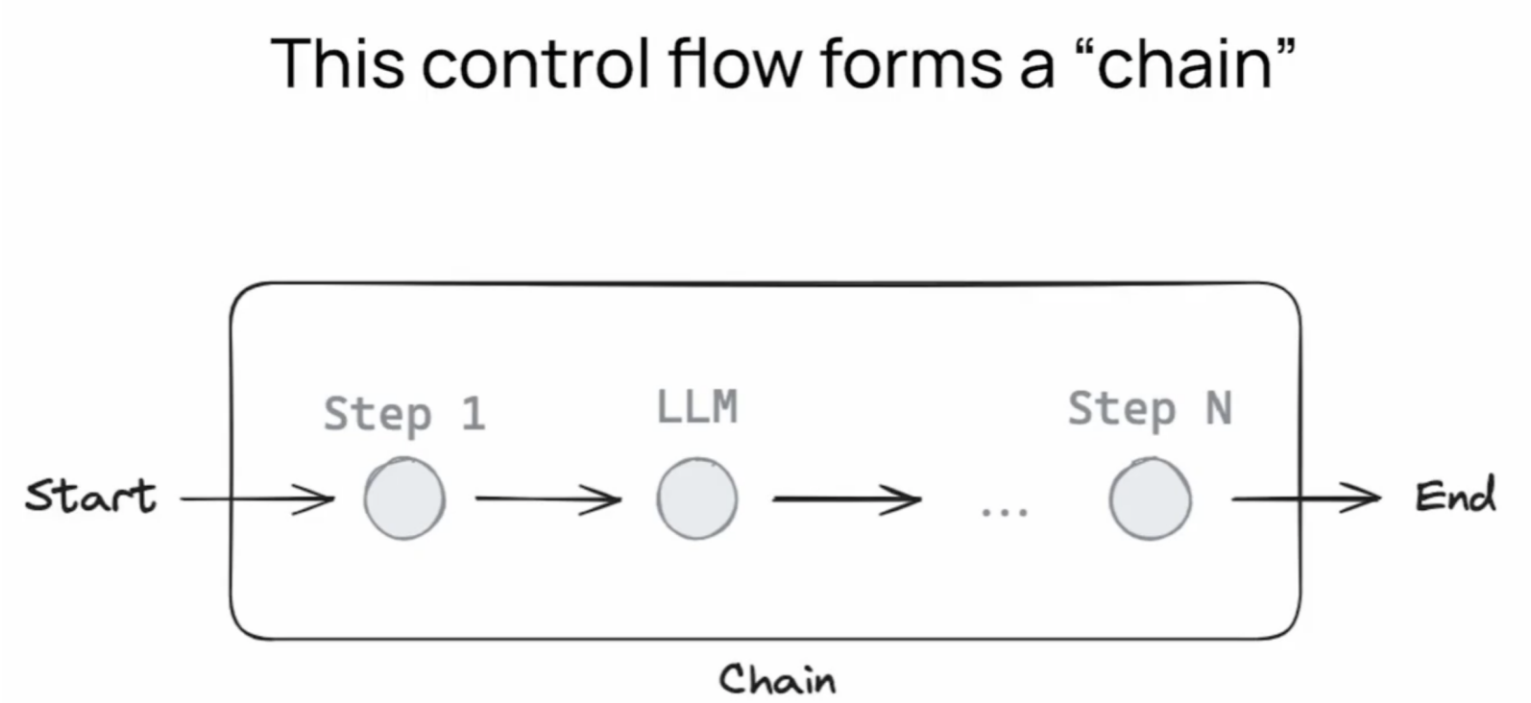


Disclaimer

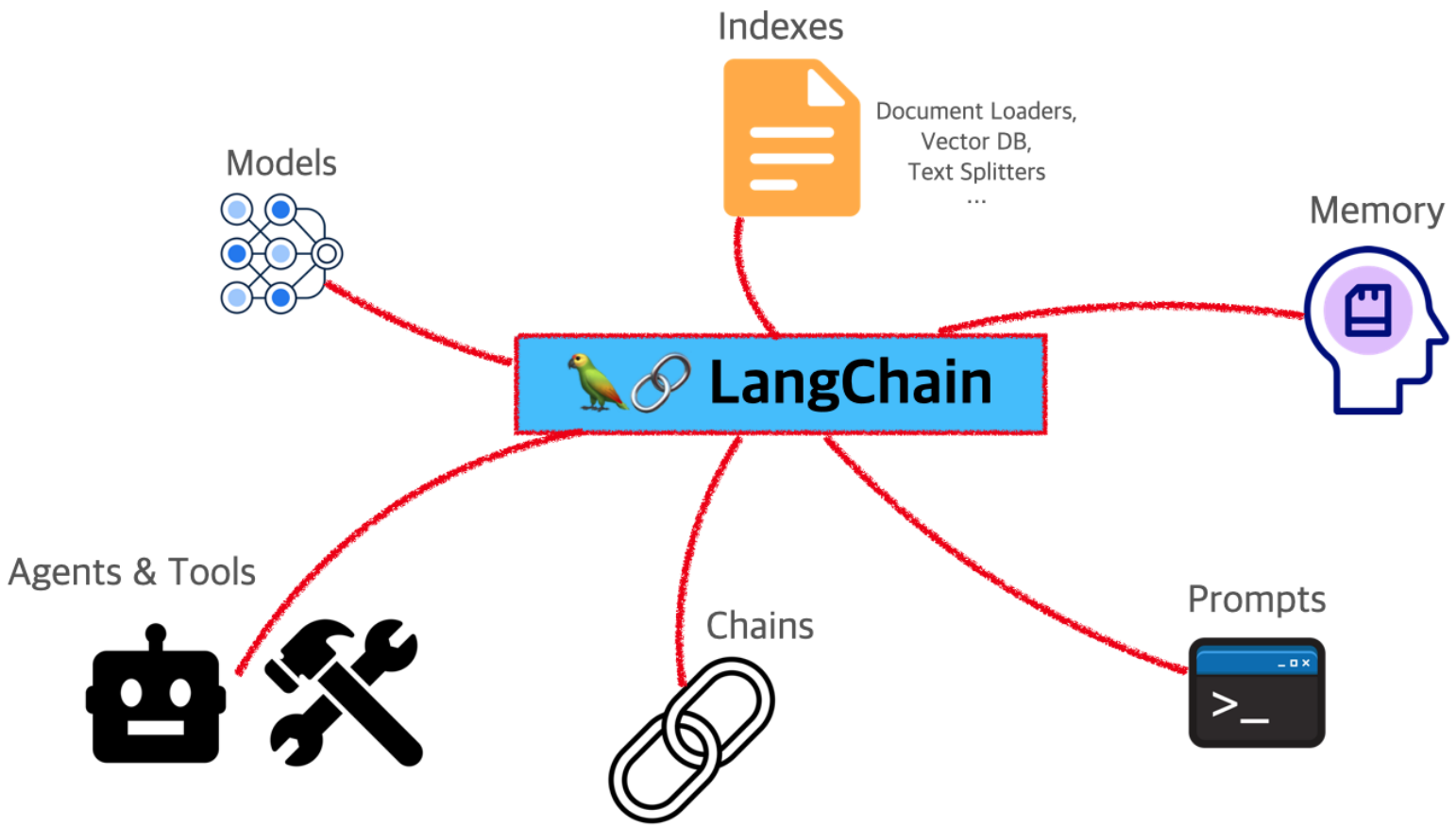
The LangChain part was heavily copy-pasted from DeepLearning.AI



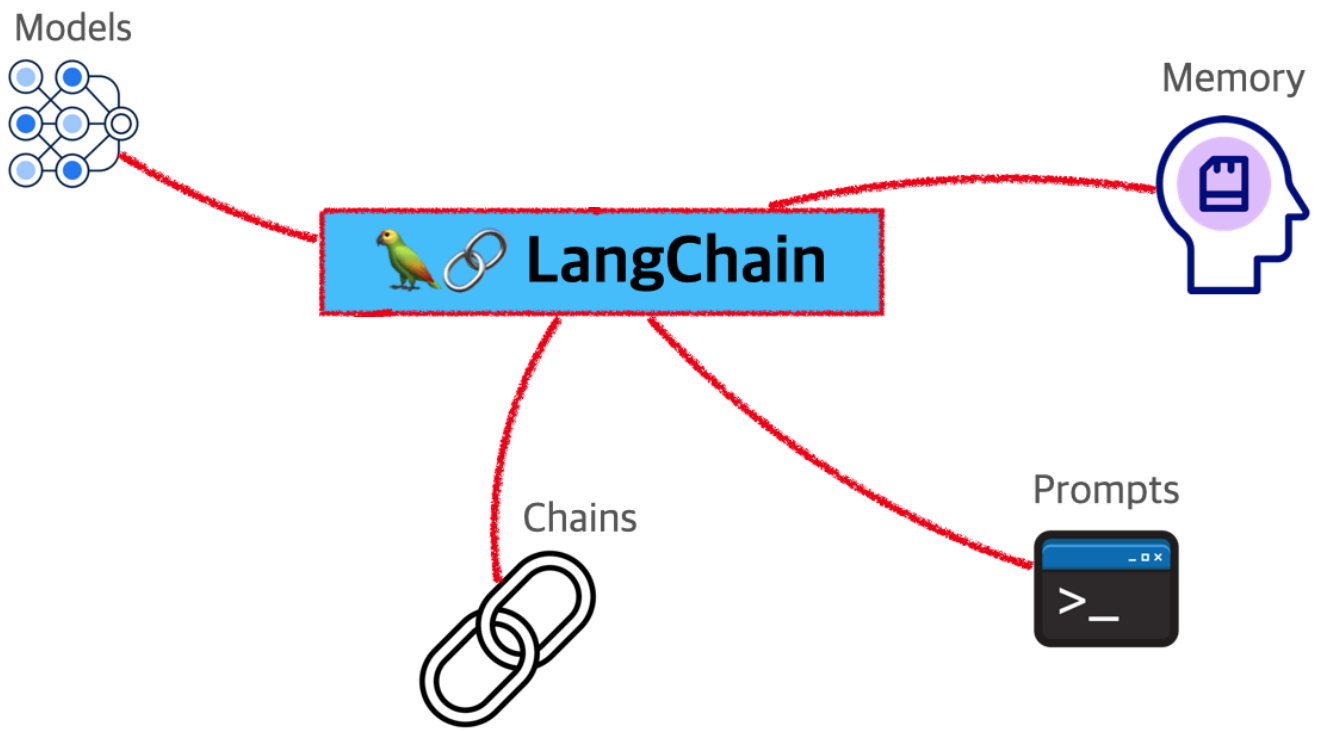
Recap: LangChain = LLM + Chain



Recap: LangChain Overview



Recap: LangChain – What We've Learned



Recap: LangChain – Models & Prompts

Chat API : LangChain

Let's try how we can do the same using LangChain.

In [ ]:
```
#!pip install --upgrade langchain
```

Model

In [ ]:
```
from langchain.chat_models import ChatOpenAI
```

In [ ]:
```
# To control the randomness and creativity of the generated
# text by an LLM, use temperature = 0.0
chat = ChatOpenAI(temperature=0.0, model=llm_model)
chat
```

Recap: LangChain – Models & Prompts

```
customer_style = """American English \
in a calm and respectful tone
"""
```

```
customer_email = """
Arrr, I be fuming that me blender lid \
flew off and splattered me kitchen walls \
with smoothie! And to make matters worse, \
the warranty don't cover the cost of \
cleaning up me kitchen. I need yer help \
right now, matey!
"""
```

```
customer_messages = prompt_template.format_messages(
                    style=customer_style,
                    text=customer_email)
```

```
print(type(customer_messages))
print(type(customer_messages[0]))
```

```
<class 'list'>
<class 'langchain.schema.HumanMessage'>
```

Recap: LangChain – Models & Prompts

```
customer_response = chat(customer_messages)
```

```
print(customer_response.content)
```

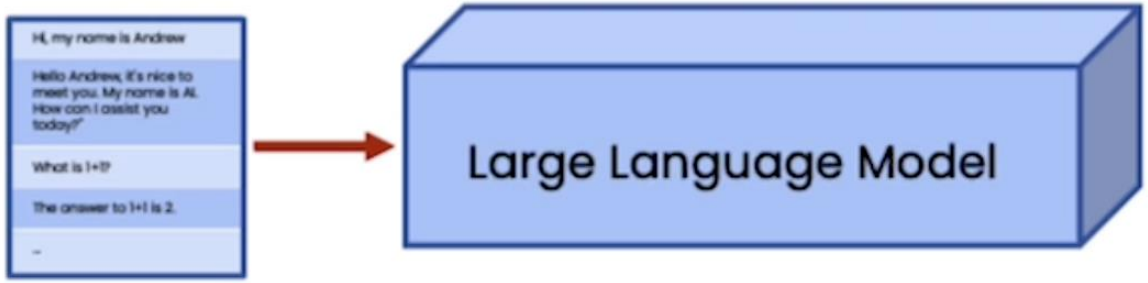
```
I'm really frustrated that my blender lid flew off and mad
e a mess of my kitchen walls with smoothie. To add to my f
rustration, the warranty doesn't cover the cost of cleanin
g up my kitchen. Can you please help me out, friend?
```

Recap: LangChain – Memory

Large Language Models are ‘stateless’

- Each transaction is independent

Chatbots appear to have memory by providing the full conversation as ‘context’



LangChain provides several kinds of ‘memory’ to store and accumulate the conversation

Recap: LangChain – Memory

ConversationBufferMemory

In [ ]:
```
from langchain.chat_models import ChatOpenAI
from langchain.chains import ConversationChain
from langchain.memory import ConversationBufferMemory
```

In [ ]:
```
llm = ChatOpenAI(temperature=0.0, model=llm_model)
memory = ConversationBufferMemory()
conversation = ConversationChain(
    llm=llm,
    memory = memory,
    verbose=True
)
```

In [ ]:
```
conversation.predict(input="Hi, my name is Andrew")
```

In [ ]:
```
conversation.predict(input="What is 1+1?")
```

In [ ]:
```
conversation.predict(input="What is my name?")
```

Recap: LangChain – Chains

LLMChain

In [ ]:
```
from langchain.chat_models import ChatOpenAI
from langchain.prompts import ChatPromptTemplate
from langchain.chains import LLMChain
```

In [ ]:
```
llm = ChatOpenAI(temperature=0.9, model=llm_model)
```

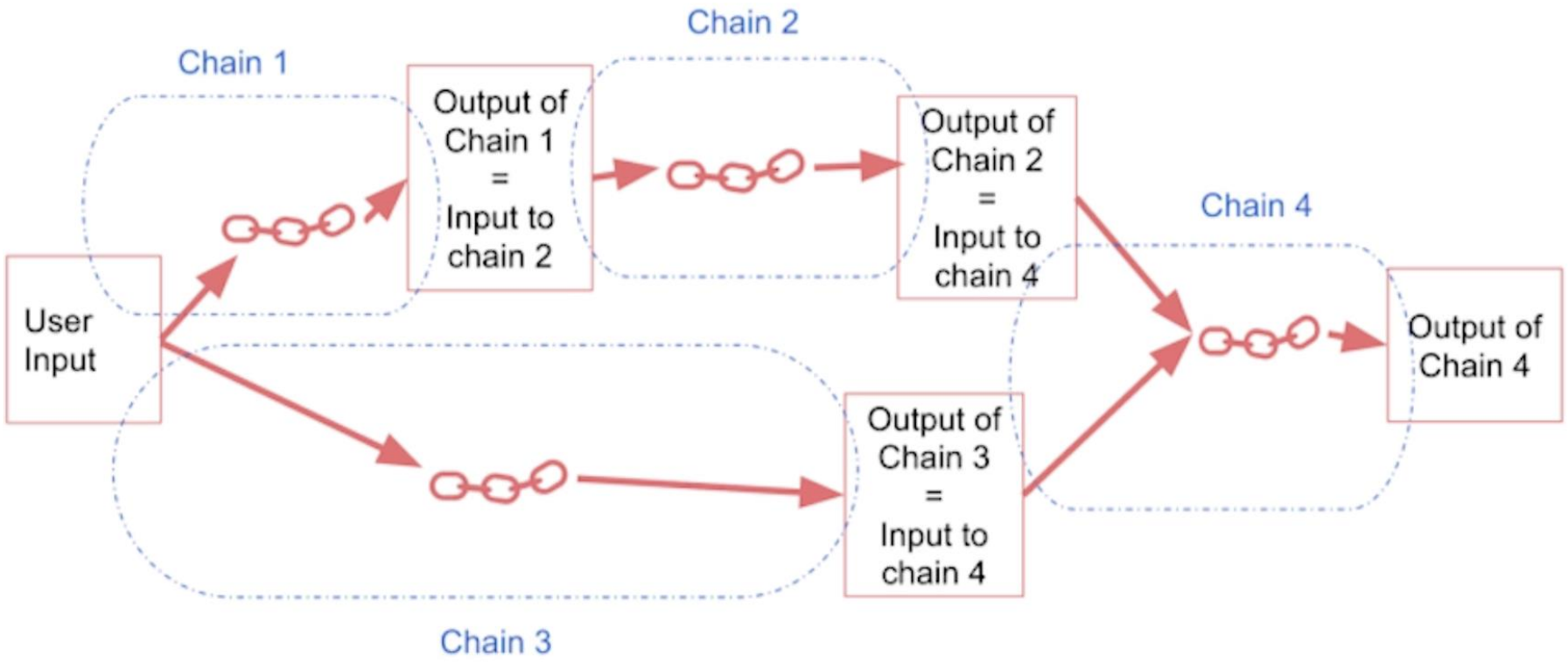
In [ ]:
```
prompt = ChatPromptTemplate.from_template(
    "What is the best name to describe \
    a company that makes {product}?"
)
```

In [ ]:
```
chain = LLMChain(llm=llm, prompt=prompt)
```

```
product = "Queen Size Sheet Set"
chain.run(product)
```

```
'Royal Beddings.'
```

Recap: LangChain – Chains



Recap: LangChain – Chains

SequentialChain

In [ ]:
```
from langchain.chains import SequentialChain
```

In [ ]:
```
llm = ChatOpenAI(temperature=0.9, model=llm_model)

# prompt template 1: translate to english
first_prompt = ChatPromptTemplate.from_template(
    "Translate the following review to english:"
    "\n\n{Review}"
)
# chain 1: input= Review and output= English_Review
chain_one = LLMChain(llm=llm, prompt=first_prompt,
                     output_key="English_Review"
                    )
```

In [ ]:
```
second_prompt = ChatPromptTemplate.from_template(
    "Can you summarize the following review in 1 sentence:"
    "\n\n{English_Review}"
)
# chain 2: input= English_Review and output= summary
chain_two = LLMChain(llm=llm, prompt=second_prompt,
                     output_key="summary"
                    )
```

```
# prompt template 3: translate to english
third_prompt = ChatPromptTemplate.from_template(
    "What language is the following review:\n\n{Review}"
)
# chain 3: input= Review and output= language
chain_three = LLMChain(llm=llm, prompt=third_prompt,
                       output_key="language"
                      )
```

```
# prompt template 4: follow up message
fourth_prompt = ChatPromptTemplate.from_template(
    "Write a follow up response to the following "
    "summary in the specified language:"
    "\n\nSummary: {summary}\n\nLanguage: {language}"
)
# chain 4: input= summary, language and output= followup_message
chain_four = LLMChain(llm=llm, prompt=fourth_prompt,
                      output_key="followup_message"
                     )
```

Recap: LangChain – Chains

```
# overall_chain: input= Review
# and output= English_Review,summary, followup_message
overall_chain = SequentialChain(
    chains=[chain_one, chain_two, chain_three, chain_four],
    input_variables=["Review"],
    output_variables=["English_Review", "summary","followup_messag
    verbose=True
)
```

Recap: LangChain – Chains

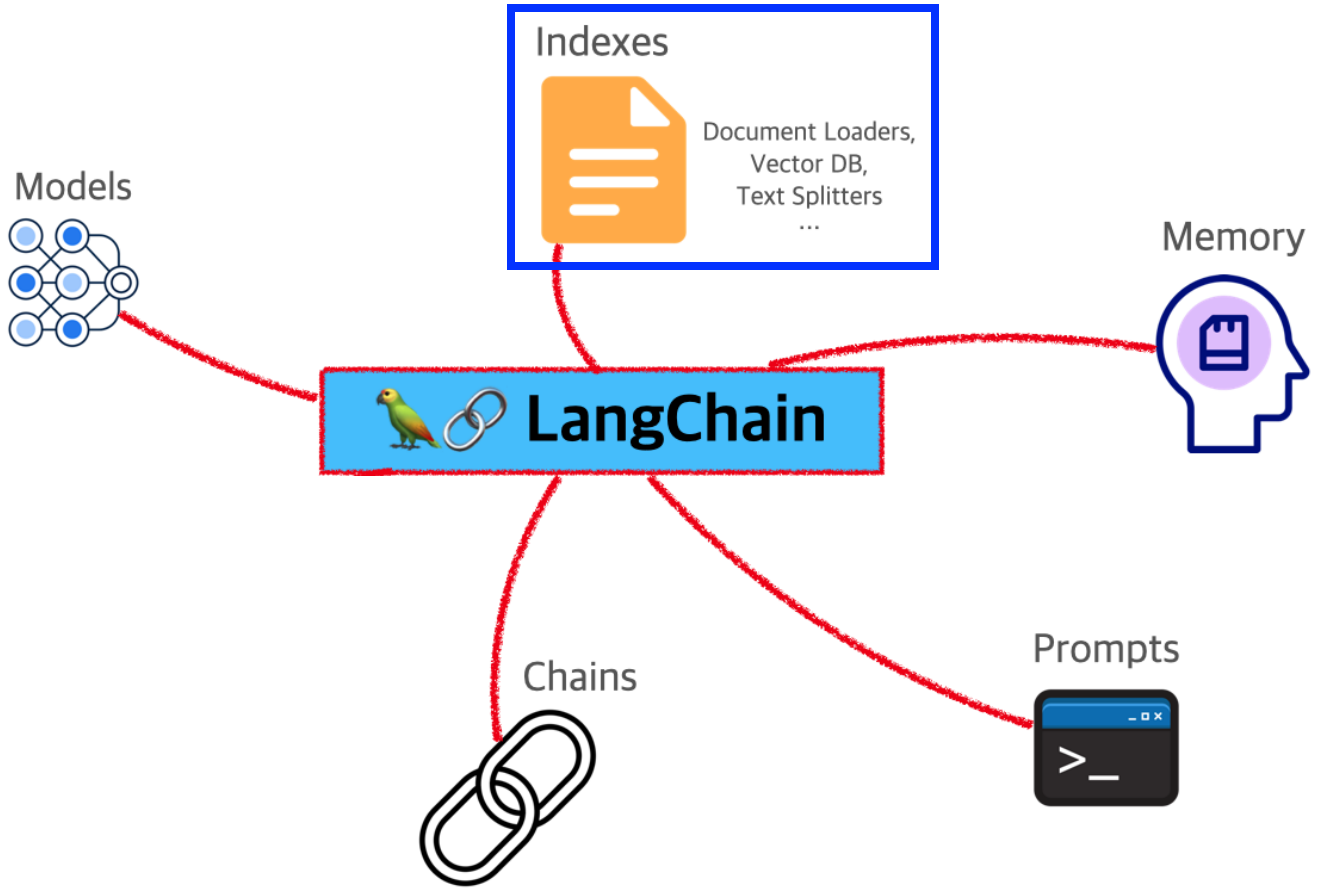
```
review = df.Review[5]
overall_chain(review)
```

```
> Entering new SequentialChain chain...

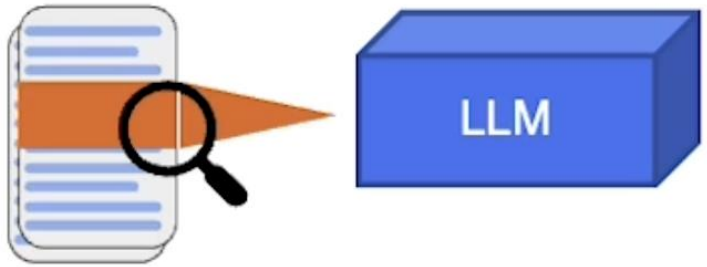
> Finished chain.

{'Review': "Je trouve le goût médiocre. La mousse ne tien
t pas, c'est bizarre. J'achète les mêmes dans le commerce
et le goût est bien meilleur...\nVieux lot ou contrefaçon
!?",
 'English_Review': "I find the taste mediocre. The foam d
oesn't hold, it's weird. I buy the same ones in stores an
d the taste is much better... Old batch or counterfei
t?!",
 'summary': "The reviewer thinks the taste is mediocre, t
he foam doesn't hold, and suspects it could be an old bat
ch or counterfeit product.",
 'followup_message': "Réponse : Le critique pense que le
goût est médiocre, que la mousse ne tient pas et soupçonn
e qu'il pourrait s'agit d'un ancien lot ou d'un produit c
ontrefait. Nous vous remercions pour votre retour sur ce
produit. Nous travaillons en étroite collaboration avec n
os fournisseurs pour nous assurer que nos clients reçoive
nt toujours des produits frais et authentiques. Nous allo
ns enquêter sur votre problème de mousse et faire tout no
tre possible pour améliorer la qualité de notre produit.
Nous espérons que vous nous donnerez une autre chance de
gagner votre confiance et votre satisfaction."}
```

Retrieval Augmented Generation (RAG)

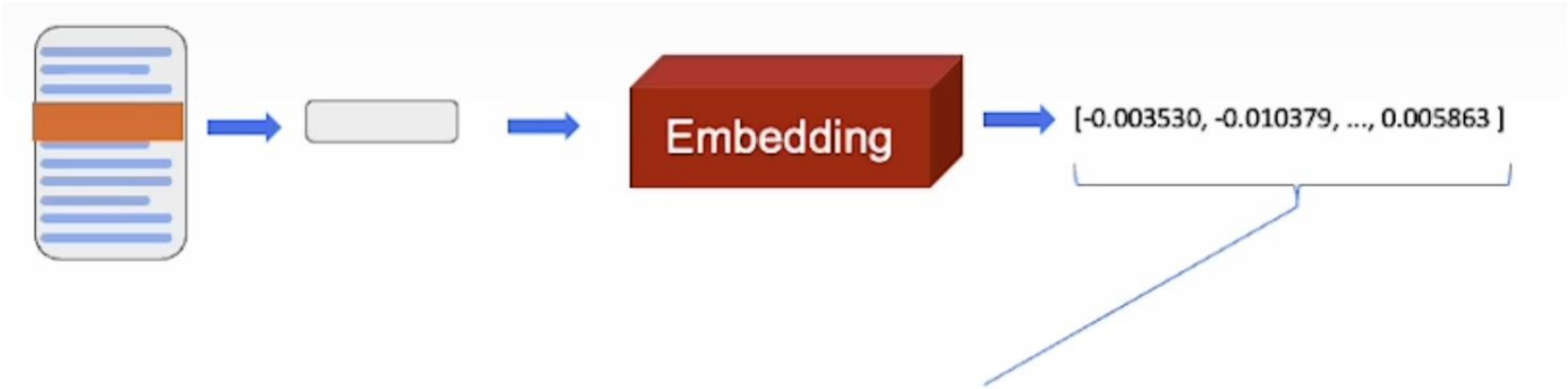


RAG – Loader



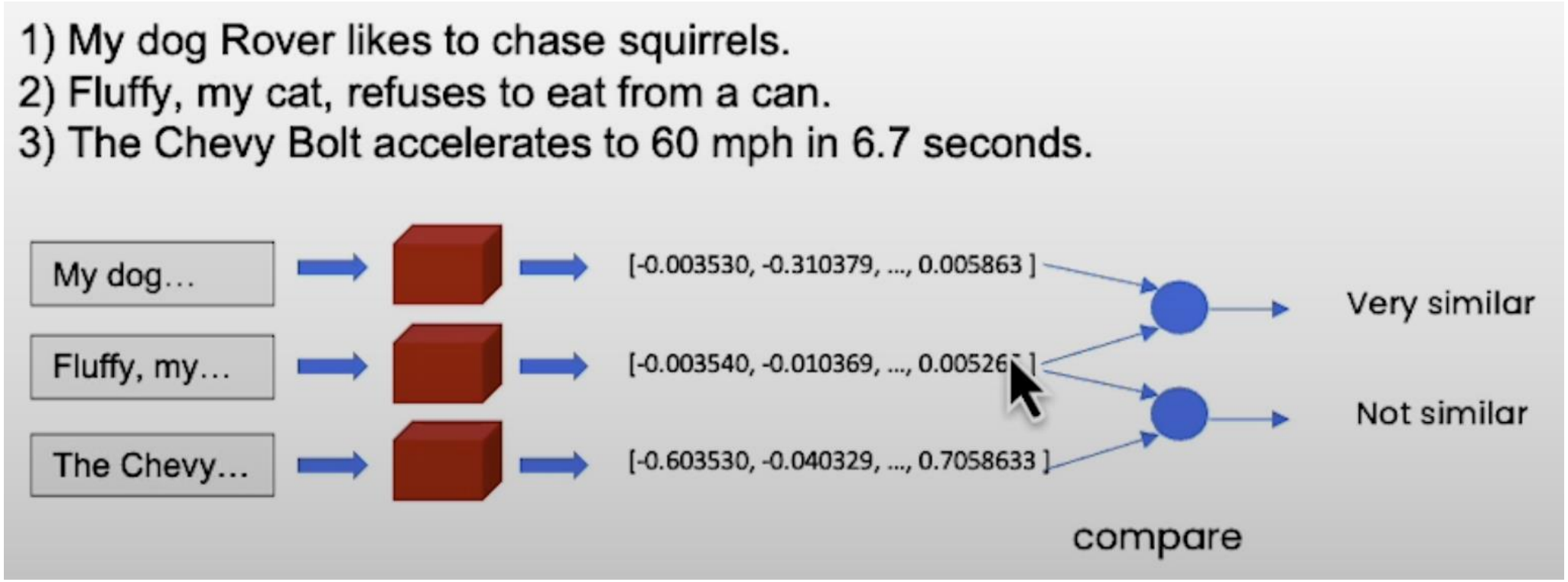
LLM's can only inspect a few thousand words at a time.

RAG – Embeddings

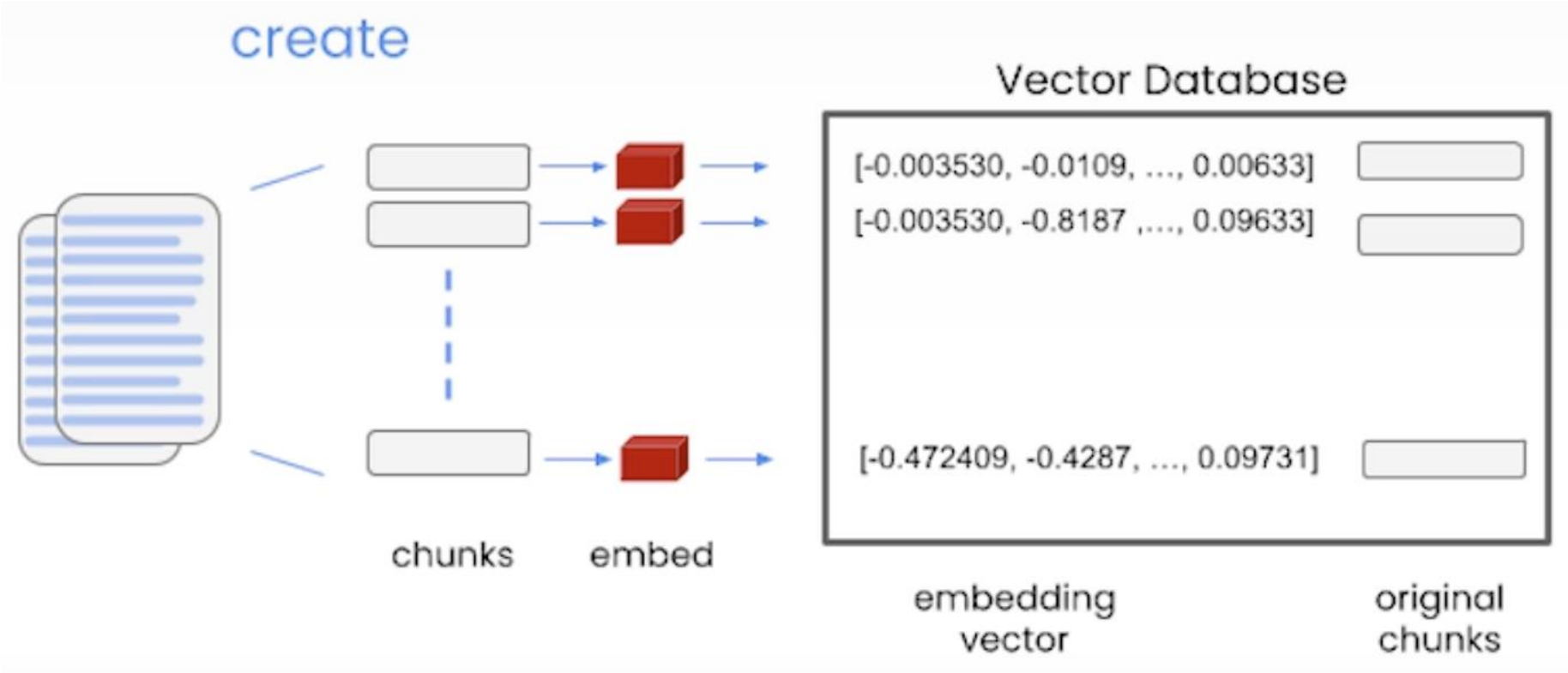


- Embedding vector captures content/meaning
- Text with similar content will have similar vectors

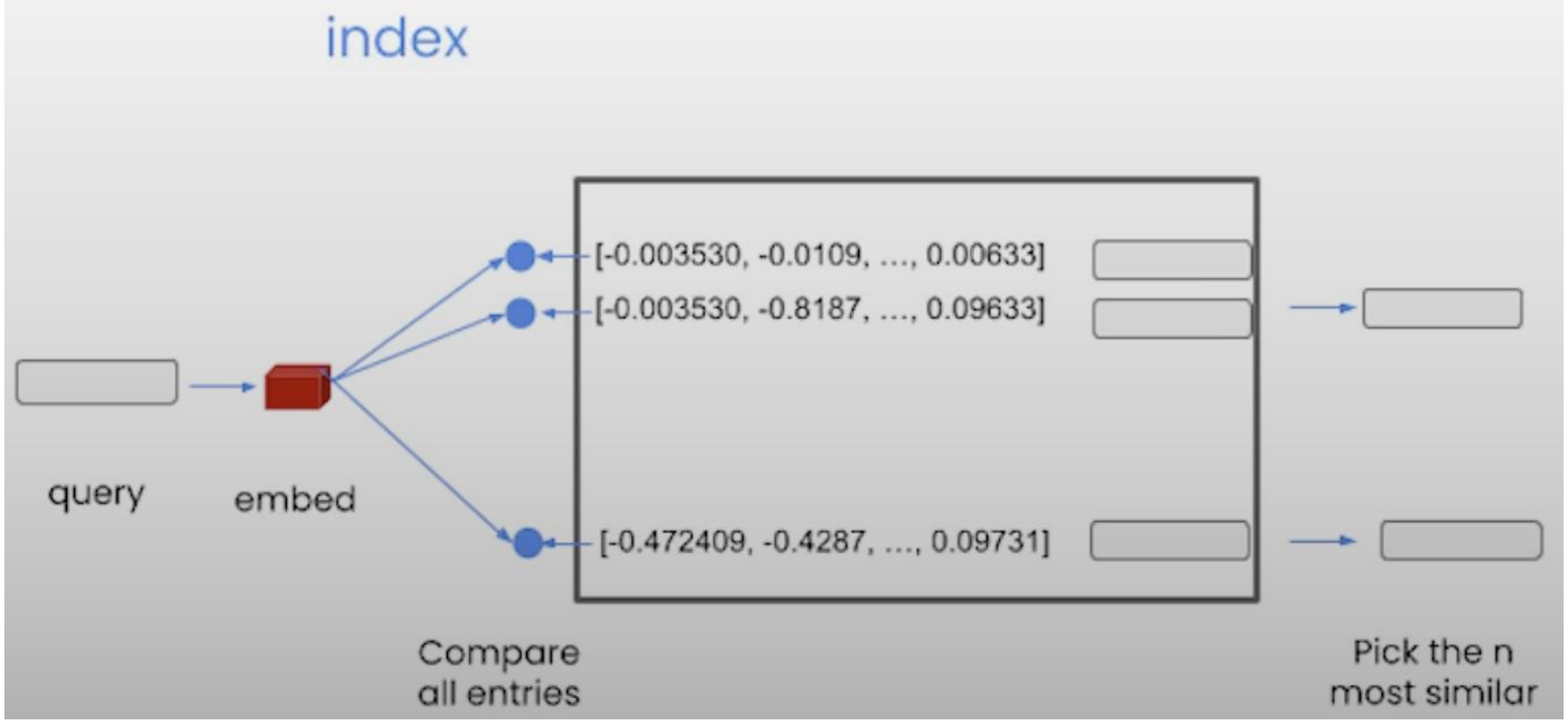
RAG – Embeddings



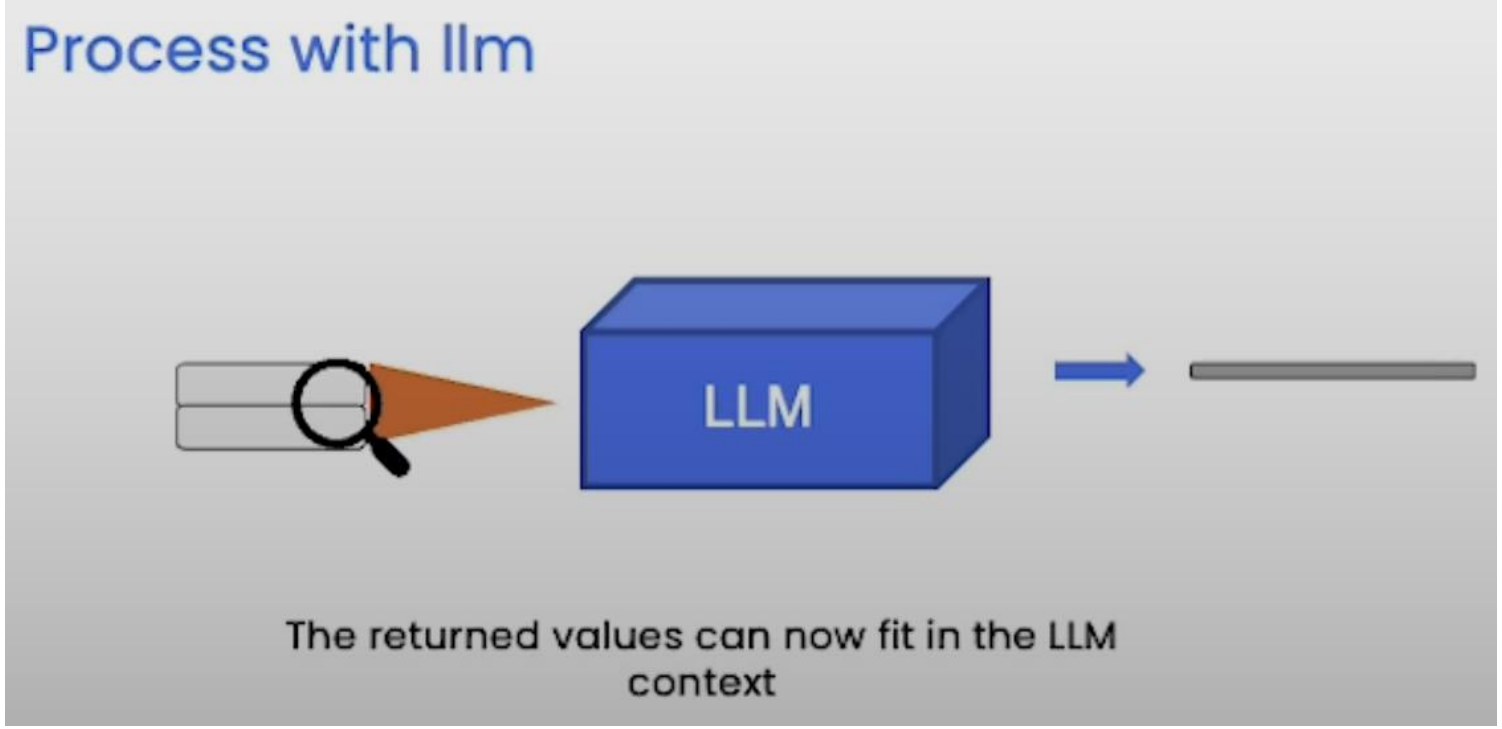
RAG – Vector DB



RAG – Vector DB



RAG – Process w/ LLM



RAG – Example

```
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain.document_loaders import CSVLoader
from langchain.vectorstores import DocArrayInMemorySearch
from IPython.display import display, Markdown
from langchain.llms import OpenAI
```

```
file = 'OutdoorClothingCatalog_1000.csv'
loader = CSVLoader(file_path=file)
```

```
from langchain.indexes import VectorstoreIndexCreator
```

```
#pip install docarray
```

```
index = VectorstoreIndexCreator(
    vectorstore_cls=DocArrayInMemorySearch
).from_loaders([loader])
```

RAG – Example

```
query ="Please list all your shirts with sun protection \
in a table in markdown and summarize each one."
```

```
response = index.query(query)
```

```
display(Markdown(response))
```

Name	Description
Men's Tropical Plaid Short-Sleeve Shirt	UPF 50+ rated, 100% polyester, wrinkle-resistant, front and back cape venting, two front bellows pockets
Men's Plaid Tropic Shirt, Short-Sleeve	UPF 50+ rated, 52% polyester and 48% nylon, machine washable and dryable, front and back cape venting, two front bellows pockets
Men's TropicVibe Shirt, Short-Sleeve	UPF 50+ rated, 71% Nylon, 29% Polyester, 100% Polyester knit mesh, machine wash and dry, front and back cape venting, two front bellows pockets
Sun Shield Shirt by	UPF 50+ rated, 78% nylon, 22% Lycra Xtra Life fiber, handwash, line dry, wicks moisture, fits comfortably over swimsuit, abrasion resistant

All four shirts provide UPF 50+ sun protection, blocking 98% of the sun's harmful rays. The Men's Tropical Plaid Short-Sleeve Shirt is made of 100% polyester and is wrinkle-resistant

RAG – Step By Step (Loader)

```
from langchain.document_loaders import CSVLoader
loader = CSVLoader(file_path=file)
```

```
docs = loader.load()
```

```
docs[0]
```

```
Document(page_content=": 0\nname: Women's Campside Oxford
s\ndescription: This ultracomfortable lace-to-toe Oxford
boasts a super-soft canvas, thick cushioning, and quality
construction for a broken-in feel from the first time you
put them on. \n\nSize & Fit: Order regular shoe size. For
half sizes not offered, order up to next whole size. \n\n
Specs: Approx. weight: 1 lb.1 oz. per pair. \n\nConstruct
ion: Soft canvas material for a broken-in feel and look.
Comfortable EVA innersole with Cleansport NXT® antimicrob
ial odor control. Vintage hunt, fish and camping motif on
innersole. Moderate arch contour of innersole. EVA foam m
idsole for cushioning and support. Chain-tread-inspired m
olded rubber outsole with modified chain-tread pattern. I
mported. \n\nQuestions? Please contact us for any inquiri
es.", metadata={'source': 'OutdoorClothingCatalog_1000.cs
v', 'row': 0})
```

RAG – Step By Step (Embeddings)

```
from langchain.embeddings import OpenAIEmbeddings
embeddings = OpenAIEmbeddings()
```

```
embed = embeddings.embed_query("Hi my name is Harrison")
```

```
print(len(embed))
```

```
1536
```

```
print(embed[:5])
```

```
[-0.021933607757091522, 0.00669704517349600 8, -0.01819835
603237152, -0.03911325708031 6544, -0.01406065095216 0358]
```

RAG – Step By Step (Vector DB)

```
db = DocArrayInMemorySearch.from_documents(
    docs,
    embeddings
)
```

```
query = "Please suggest a shirt with sunblocking"
```

```
docs = db.similarity_search(query)
```

```
len(docs)
```

```
4
```

RAG – Step By Step (Vector DB)

```
docs[0]
```

```
Document(page_content=': 255\nname: Sun Shield Shirt by\n
description: "Block the sun, not the fun – our high-perfo
rmance sun shirt is guaranteed to protect from harmful UV
rays. \n\nSize & Fit: Slightly Fitted: Softly shapes the
body. Falls at hip.\n\nFabric & Care: 78% nylon, 22% Lycr
a Xtra Life fiber. UPF 50+ rated – the highest rated sun
protection possible. Handwash, line dry.\n\nAdditional Fe
atures: Wicks moisture for quick-drying comfort. Fits com
fortably over your favorite swimsuit. Abrasion resistant
for season after season of wear. Imported.\n\nSun Protect
ion That Won\'t Wear Off\nOur high-performance fabric pro
vides SPF 50+ sun protection, blocking 98% of the sun\'s
harmful rays. This fabric is recommended by The Skin Canc
er Foundation as an effective UV protectant.', metadata=
{'source': 'OutdoorClothingCatalog_1000.csv', 'row': 25
5})
```

RAG – Step By Step (Process w/ LLM)

```
retriever = db.as_retriever()
```

```
llm = ChatOpenAI(temperature = 0.0)
```

```
qdocs = "".join([docs[i].page_content for i in range(len(dc
```

```
response = llm.call_as_llm(f"{qdocs} Question: Please list
shirts with sun protection in a table in markdown and summa
```

RAG – Step By Step (Process w/ LLM + Chains)

```
qa_stuff = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever,
    verbose=True
)
```

```
query =  "Please list all your shirts with sun protection i
in markdown and summarize each one."
```

```
response = qa_stuff.run(query)
```

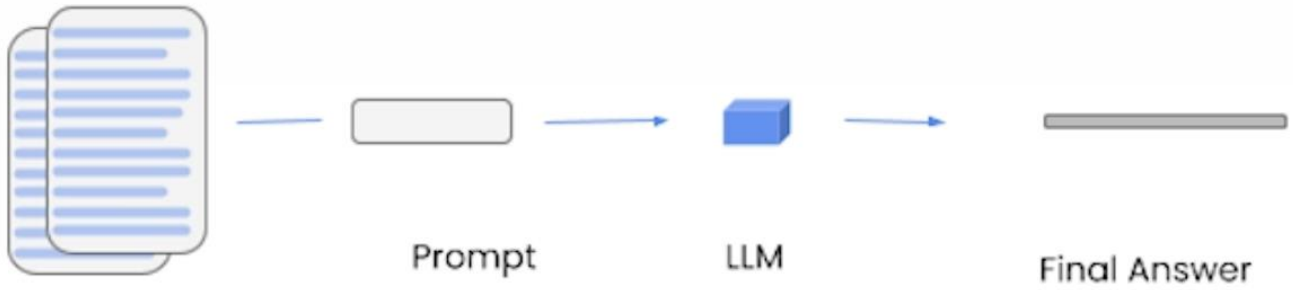
RAG – Chain Types

```
qa_stuff = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever,
    verbose=True
)
```

```
query =  "Please list all your shirts with sun protection i
in markdown and summarize each one."
```

```
response = qa_stuff.run(query)
```

RAG – Chain Types

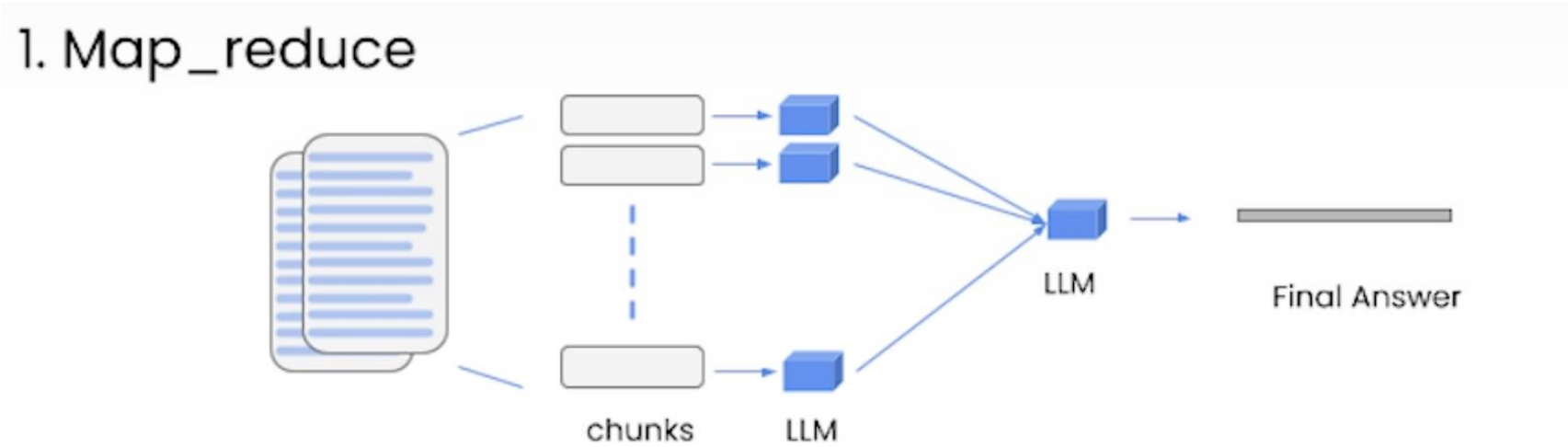


Stuffing is the simplest method. You simply stuff all data into the prompt as context to pass to the language model.

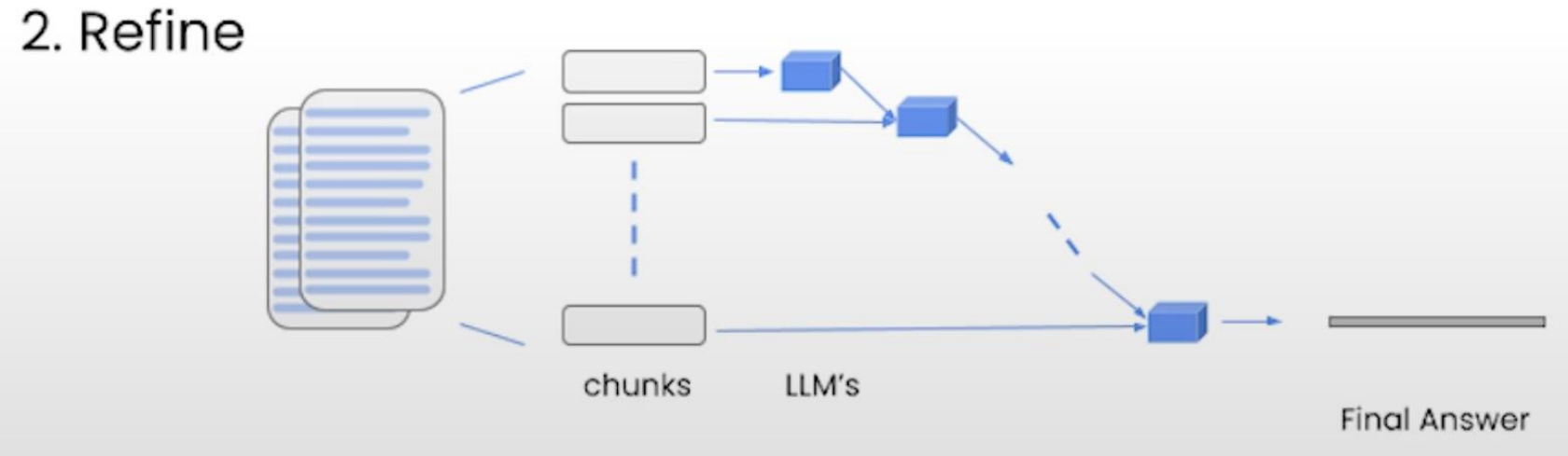
Pros: It makes a single call to the LLM. The LLM has access to all the data at once.

Cons: LLMs have a context length, and for large documents or many documents this will not work as it will result in a prompt larger than the context length.

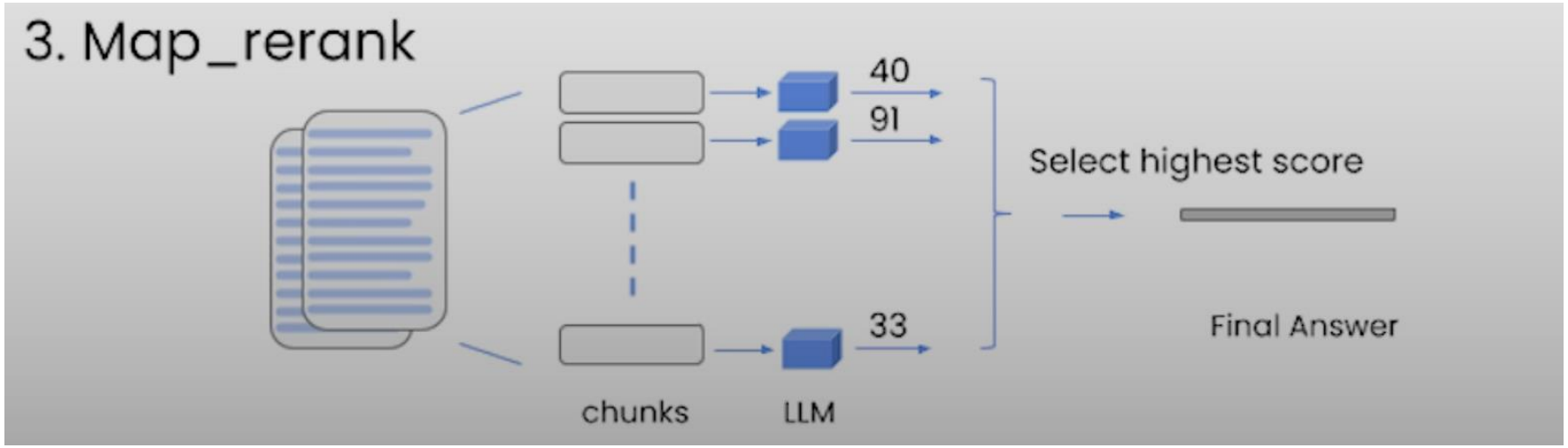
RAG – Chain Types



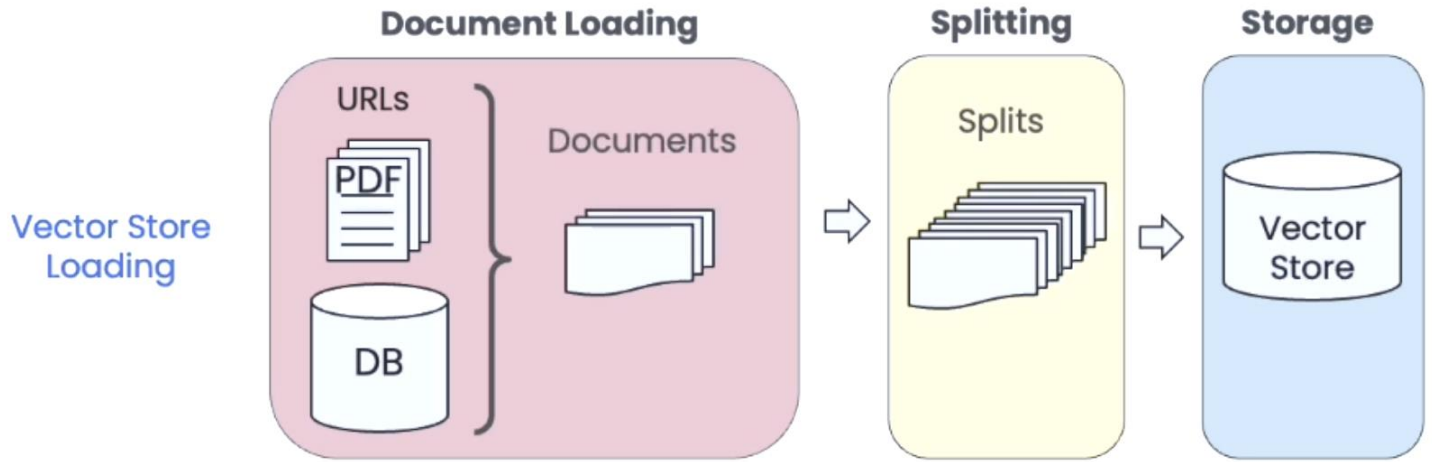
RAG – Chain Types

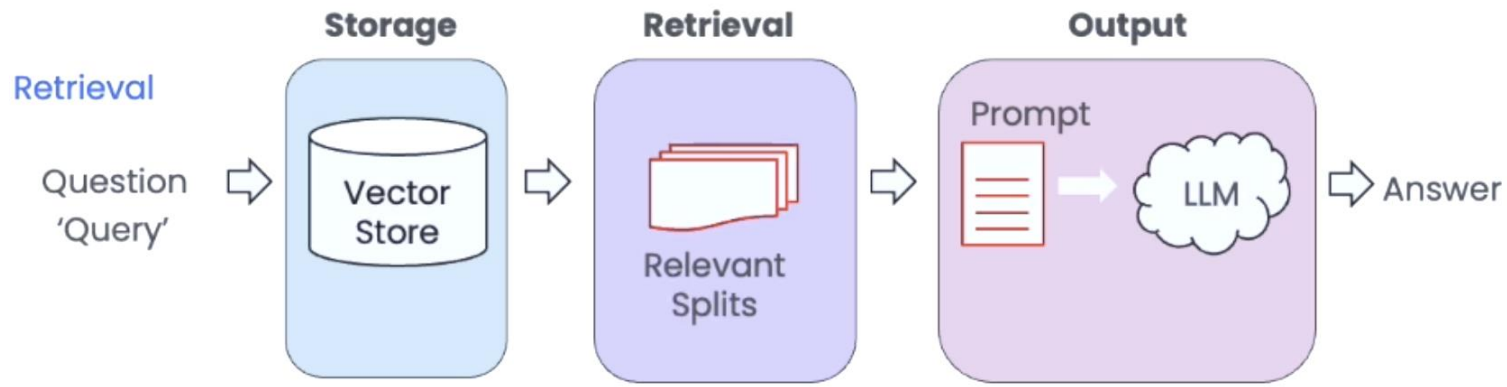


RAG – Chain Types

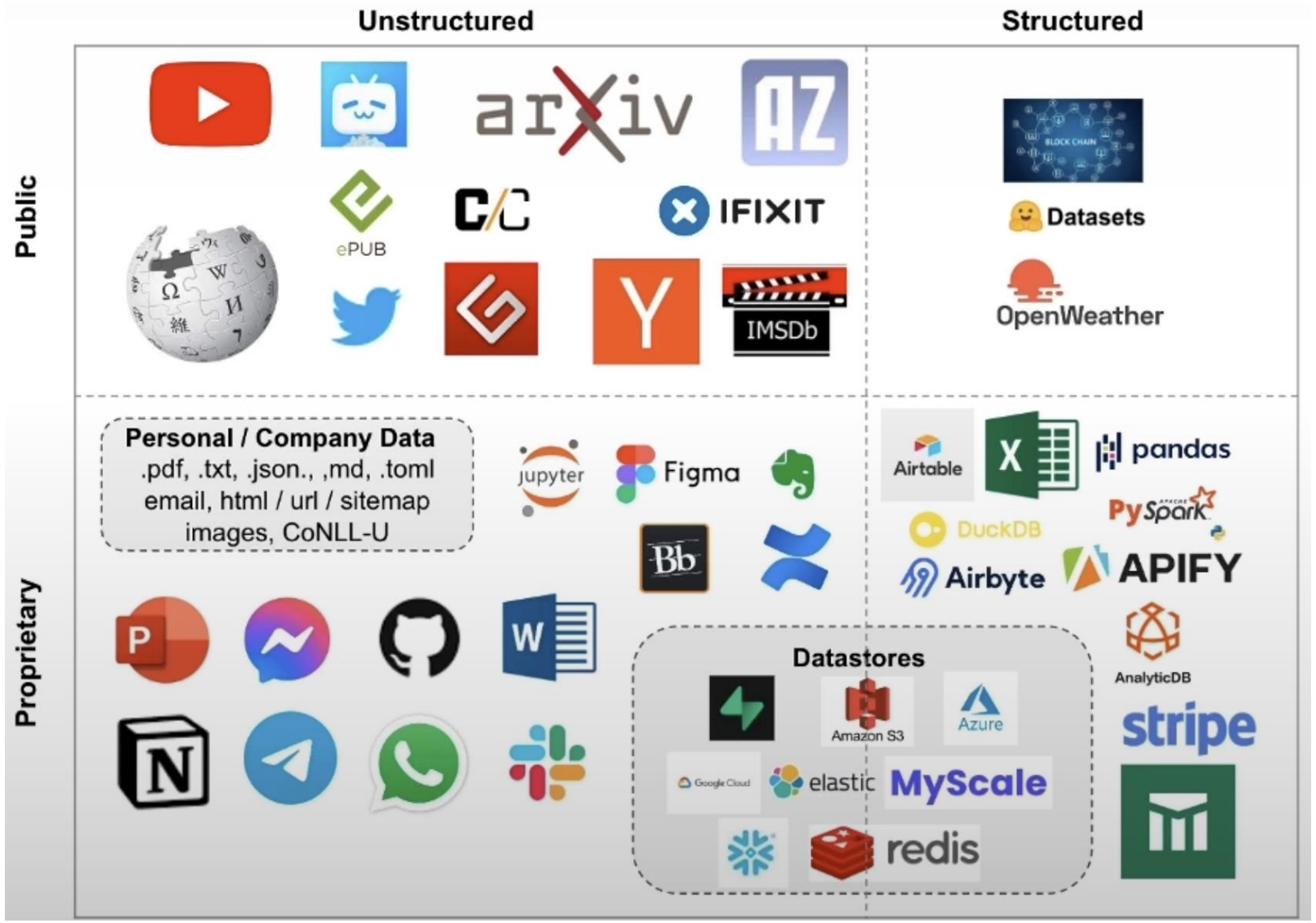


RAG Overview





More RAG – Document Loading



More RAG – Document Loading

```python
from langchain.document_loaders import PyPDFLoader
```

```python
loader = PyPDFLoader("docs/cs229_lectures/MachineLearning-Lecture01.pdf")
```

```python
pages = loader.load()
```

```python
len(pages)
```

```
22
```

```python
page = pages[0]
```

```python
print(page.page_content[:500])
```

```
MachineLearning-Lecture01
Instructor (Andrew Ng):  Okay. Good morning. Welcome to CS229, the machine
learning class. So what I wanna do today is ju st spend a little time going ov
er the logistics
of the class, and then we'll start to  talk a bit about machine learning.
By way of introduction, my name's  Andrew Ng and I'll be instru ctor for this
class. And so
I personally work in machine learning, and I' ve worked on it for about 15 yea
rs now, and
I actually think that machine learning i
```

More RAG – Document Loading

```
from langchain.document_loaders.generic import GenericLoader
from langchain.document_loaders.parsers import OpenAIWhisperParser
from langchain.document_loaders.blob_loaders.youtube_audio import YoutubeAudio
```

```
url="https://www.youtube.com/watch?v=jGwO_UgTS7I"
save_dir="docs/youtube/"
loader = GenericLoader(
    YoutubeAudioLoader([url],save_dir),
    OpenAIWhisperParser()
)
docs = loader.load()
```

```
[youtube] Extracting URL: https://www.youtube.com/watch?v=jGwO_UgTS7I
[youtube] jGwO_UgTS7I: Downloading webpage
[youtube] jGwO_UgTS7I: Downloading android player API JSON
[info] jGwO_UgTS7I: Downloading 1 format(s): 140
[download] docs/youtube//Stanford CS229: Machine Learning Course, Lecture 1 -
Andrew Ng (Autumn 2018).m4a has already been downloaded
[download] 100% of   69.71MiB
[ExtractAudio] Not converting audio docs/youtube//Stanford CS229: Machine Lea
rning Course, Lecture 1 - Andrew Ng (Autumn 2018).m4a; file is already in targ
et format m4a
Transcribing part 1!
```

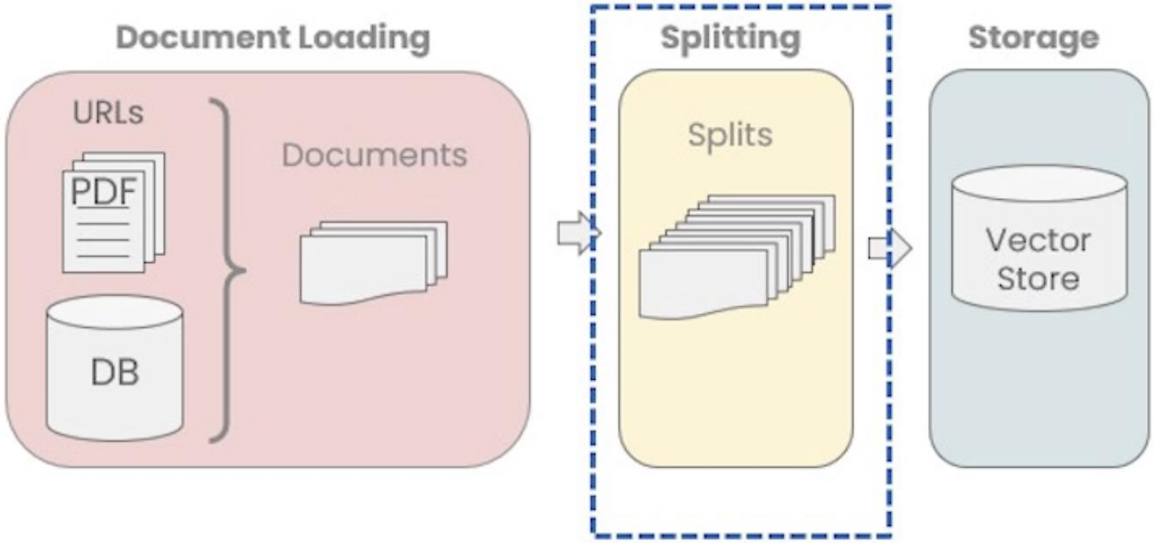
More RAG – Document Loading

```
docs[0].page_content[0:500]
```

```
"Welcome to CS229 Machine Learning. Some of you know that this is a class tha
t's taught at Stanford for a long time. And this is often the class that, um,
I most look forward to teaching each year because this is where we've helped,
I think several generations of Stanford students become experts in machine lea
rning, got- built many of their products and services and startups that I'm su
re, many of you or probably all of you are using, uh, uh, today. Um, so what I
want to do today was spend some t"
```

More RAG – Document Splitting

- Splitting Documents into smaller chunks
  - Retaining meaningful relationships!



...
on this model. The Toyota Camry has a head-snapping
80 HP and an eight-speed automatic transmission that will
...

Chunk 1: on this model. The Toyota Camry has a head-snapping

Chunk 2: 80 HP and an eight-speed automatic transmission that will

Question: What are the specifications on the Camry?

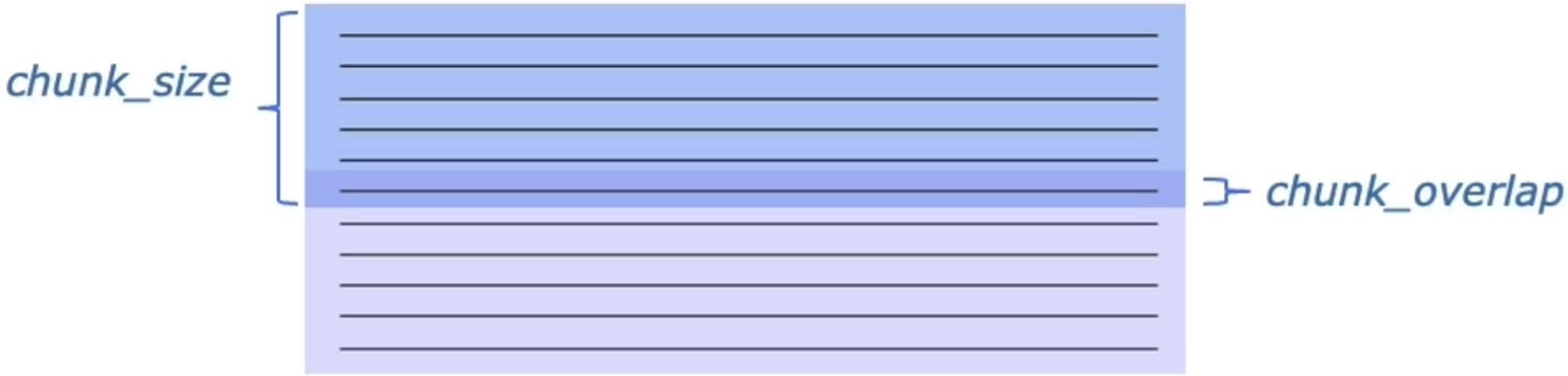
More RAG – Document Splitting

```
langchain.text_splitter.CharacterTextSplitter(
    separator: str = "\n\n"
    chunk_size=4000,
    chunk_overlap=200,
    length_function=<builtin function len>,
)
```

Methods:

create_documents() - Create documents from a list of texts.

split_documents() - Split documents.



More RAG – Document Splitting

`RecursiveCharacterTextSplitter` is recommended for generic text.

In [ ]:
```
some_text = """When writing documents, writers will use document s
This can convey to the reader, which idea's are related. For examp
are in sentances. Similar ideas are in paragraphs. Paragraphs form
Paragraphs are often delimited with a carriage return or two carri
Carriage returns are the "backslash n" you see embedded in this st
Sentences have a period at the end, but also, have a space.\
and words are separated by space."""
```

In [ ]:
```
c_splitter = CharacterTextSplitter(
    chunk_size=450,
    chunk_overlap=0,
    separator = ' '
)
r_splitter = RecursiveCharacterTextSplitter(
    chunk_size=450,
    chunk_overlap=0,
    separators=["\n\n", "\n", " ", ""]
)
```

More RAG – Document Splitting

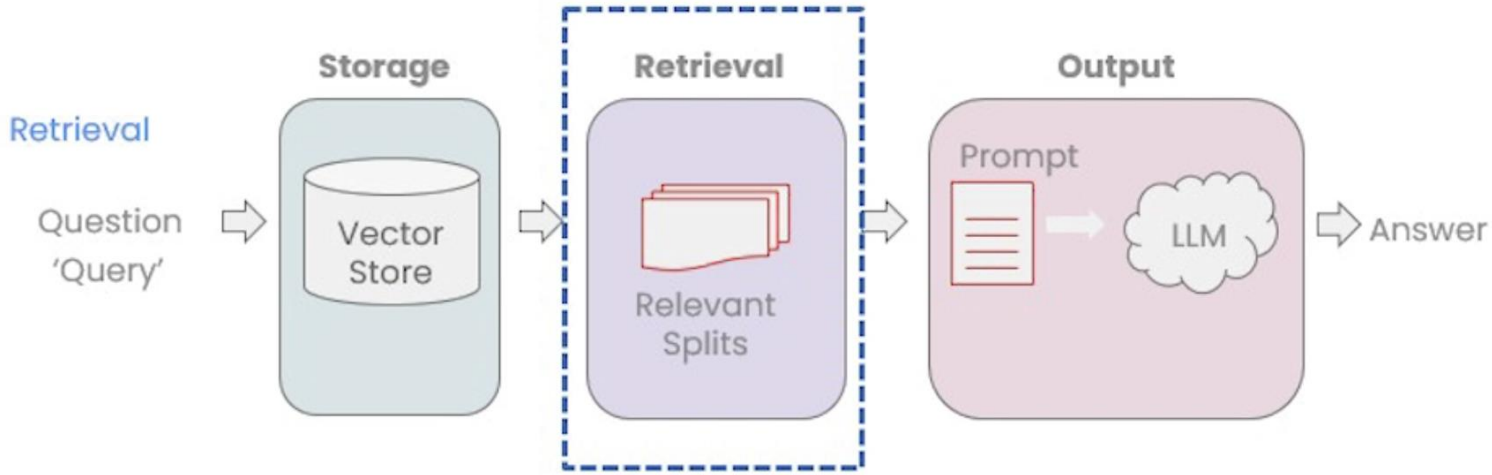
```
c_splitter.split_text(some_text)
```

```
['When writing documents, writers will use document structure to group conten
t. This can convey to the reader, which idea\'s are related. For example, clos
ely related ideas are in sentances. Similar ideas are in paragraphs. Paragraph
s form a document. \n\n Paragraphs are often delimited with a carriage return
or two carriage returns. Carriage returns are the "backslash n" you see embedd
ed in this string. Sentences have a period at the end, but also,',
 'have a space.and words are separated by space.']
```

```
r_splitter.split_text(some_text)
```

```
["When writing documents, writers will use document structure to group conten
t. This can convey to the reader, which idea's are related. For example, close
ly related ideas are in sentances. Similar ideas are in paragraphs. Paragraphs
form a document.",
 'Paragraphs are often delimited with a carriage return or two carriage return
s. Carriage returns are the "backslash n" you see embedded in this string. Sen
tences hav a period at the end, but also, have a space.and words are separate
d by space.']
```

More RAG – Retrieval



- Accessing/indexing the data in the vector store
  - Basic semantic similarity
  - Maximum marginal relevance
  - Including Metadata
- LLM Aided Retrieval

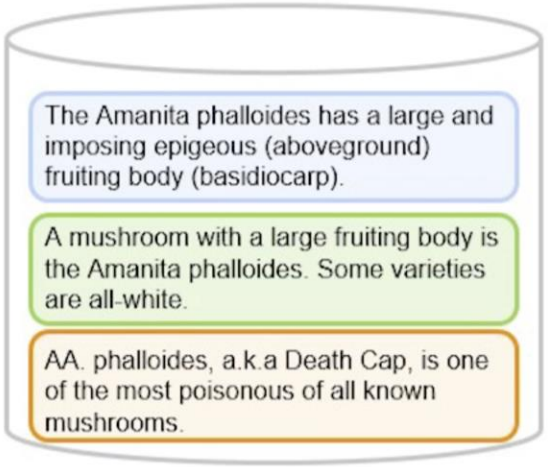
More RAG – Retrieval

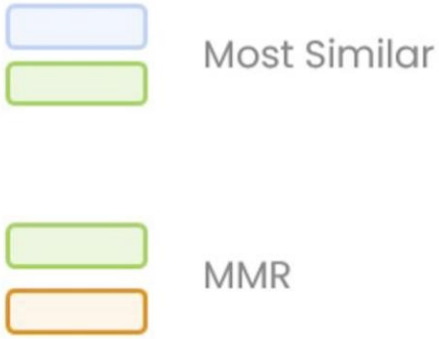
Maximum marginal relevance(MMR)

- You may not always want to choose the most similar responses



Tell me about all-white mushrooms with large fruiting bodies

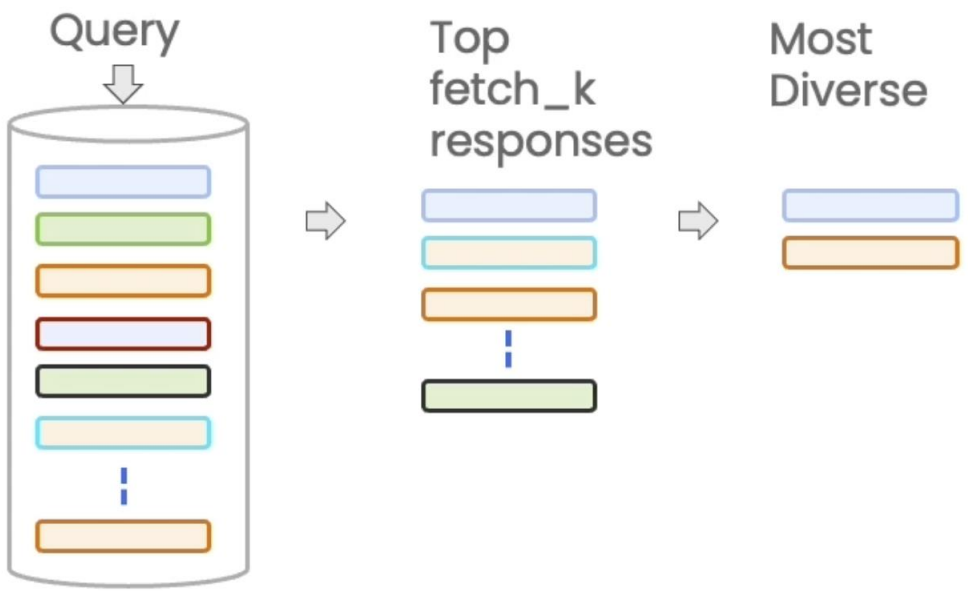




More RAG – Retrieval

MMR algorithm

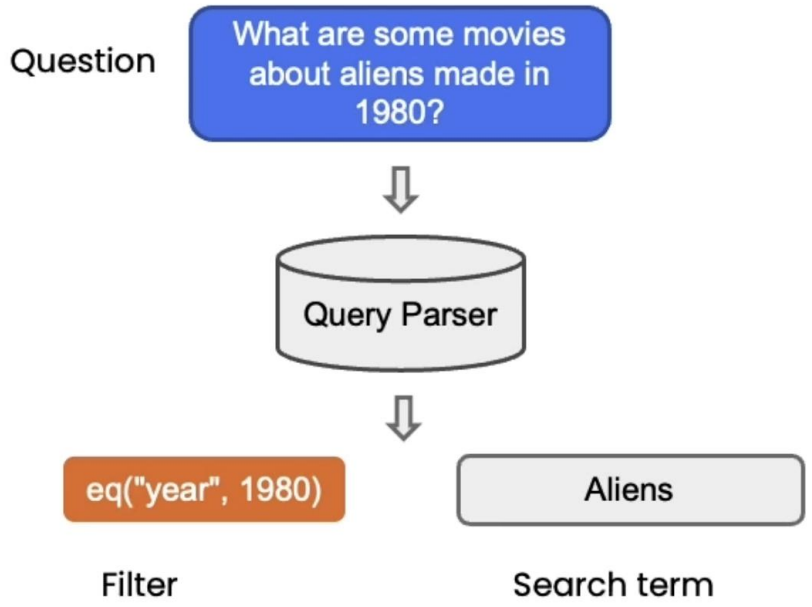
- Query the Vector Store
- Choose the `fetch_k` most similar responses
- Within those responses choose the `k` most diverse



More RAG – Retrieval

LLM Aided Retrieval

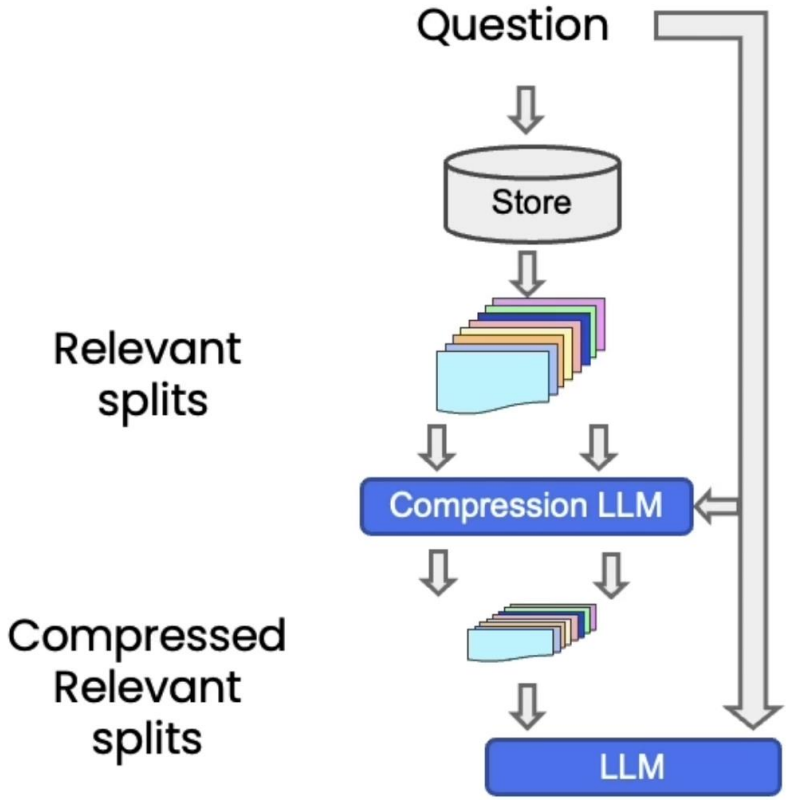
- There are several situations where the Query applied to the DB is more than just the Question asked.
- One is SelfQuery, where we use an LLM to convert the user question into a query



More RAG – Retrieval

Compression

- Increase the number of results you can put in the context by shrinking the responses to only the relevant information.



More RAG – Retrieval

```
question = "Tell me about all-white mushrooms with large fruiting bodies"
```

```
smalldb.similarity_search(question, k=2)
```

```
[Document(page_content='A mushroom with a large fruiting body is the Amanita p
halloides. Some varieties are all-white.', metadata={}),
 Document(page_content='The Amanita phalloides has a large and imposing epigeo
us (aboveground) fruiting body (basidiocarp).', metadata={})]
```

```
smalldb.max_marginal_relevance_search(question,k=2, fetch_k=3)
```

```
[Document(page_content='A mushroom with a large fruiting body is the Amanita p
halloides. Some varieties are all-white.', metadata={}),
 Document(page_content='A. phalloides, a.k.a Death Cap, is one of the most poi
sonous of all known mushrooms.', metadata={})]
```

More RAG – Retrieval

Addressing Specificity: working with metadata

In last lecture, we showed that a question about the third lecture can include results from other lectures as well.

To address this, many vectorstores support operations on `metadata`.

`metadata` provides context for each embedded chunk.

```
question = "what did they say about regression in the third lecture?"
```

```
docs = vectordb.similarity_search(
    question,
    k=3,
    filter={"source":"docs/cs229_lectures/MachineLearning-Lecture03.pdf"})
```

```
for d in docs:
    print(d.metadata)
```

```
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 0}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 14}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 4}
```

More RAG – Retrieval

Addressing Specificity: working with metadata using self-query retriever

But we have an interesting challenge: we often want to infer the metadata from the query itself.

To address this, we can use `SelfQueryRetriever`, which uses an LLM to extract:

1. The `query` string to use for vector search
2. A metadata filter to pass in as well

Most vector databases support metadata filters, so this doesn't require any new databases or indexes.

In [ ]:
```
from langchain.llms import OpenAI
from langchain.retrievers.self_query.base import SelfQueryRetrieve
from langchain.chains.query_constructor.base import AttributeInfo
```

In [ ]:
```
metadata_field_info = [
    AttributeInfo(
        name="source",
        description="The lecture the chunk is from,
        type="string",
    ),
    AttributeInfo(
        name="page",
        description="The page from the lecture",
        type="integer",
    ),
]
```

```
document_content_description = "Lecture notes"
llm = OpenAI(model='gpt-3.5-turbo-instruct', temperature=0)
retriever = SelfQueryRetriever.from_llm(
    llm,
    vectordb,
    document_content_description,
    metadata_field_info,
    verbose=True
)
```

More RAG – Retrieval

Additional tricks: compression

Another approach for improving the quality of retrieved docs is compression.

Information most relevant to a query may be buried in a document with a lot of irrelevant text.

Passing that full document through your application can lead to more expensive LLM calls and poorer responses.

Contextual compression is meant to fix this.

In [ ]:
```
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import LLMChainExtr
```

In [ ]:
```
# Wrap our vectorstore
llm = OpenAI(temperature=0, model="gpt-3.5-turbo-instruct")
compressor = LLMChainExtractor.from_llm(llm)
```

In [ ]:
```
compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=vectordb.as_retriever()
)
```

More RAG – Retrieval

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

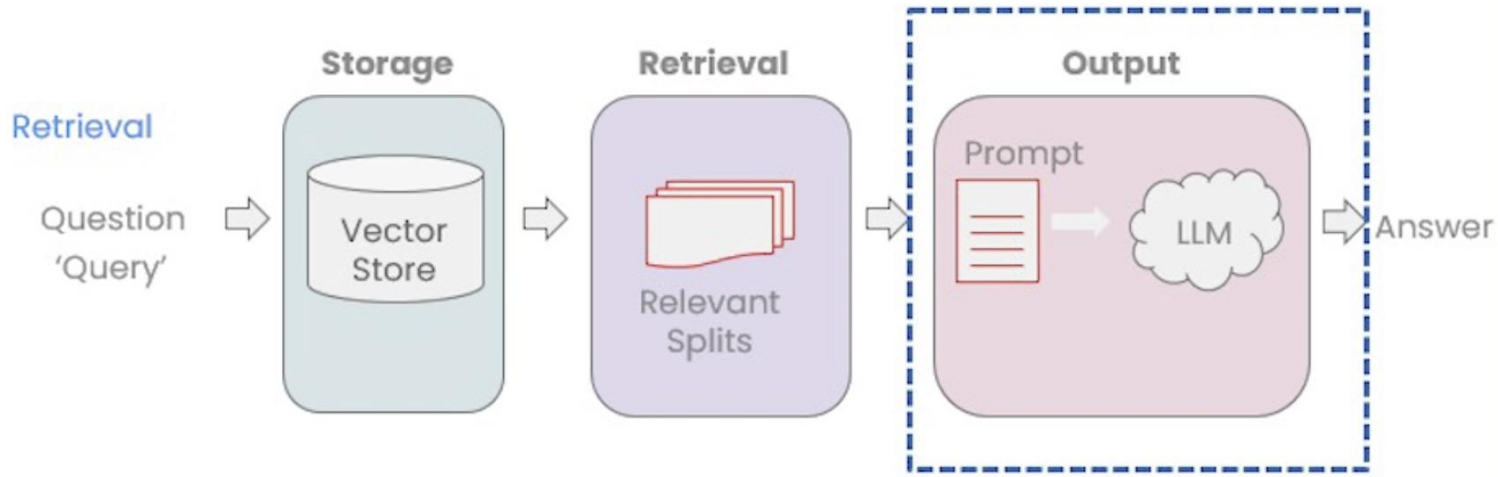
```
Document 1:

"MATLAB is I guess part of the programming language that makes it very easy to
write codes using matrices, to write code for numerical routines, to move data
around, to plot data. And it's sort of an extremely easy to learn tool to use
for implementing a lot of learning algorithms."
----------------------------------------------------------------------------
----------------------
Document 2:

"MATLAB is I guess part of the programming language that makes it very easy to
write codes using matrices, to write code for numerical routines, to move data
around, to plot data. And it's sort of an extremely easy to learn tool to use
for implementing a lot of learning algorithms."
----------------------------------------------------------------------------
----------------------
Document 3:

"And the student said, "Oh, it was the MATLAB." So for those of you that don't
know MATLAB yet, I hope you do learn it. It's not hard, and we'll actually hav
e a short MATLAB tutorial in one of the discussion sections for those of you t
hat don't know it."
```

More RAG – Question Answering (QA)

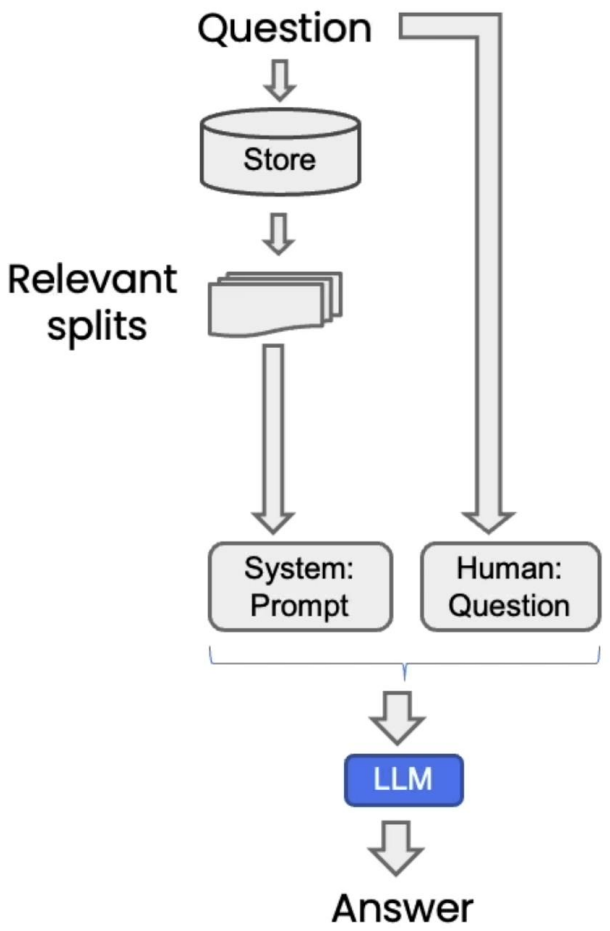


- Multiple relevant documents have been retrieved from the vector store
- Potentially compress the relevant splits to fit into the LLM context
- Send the information along with our question to an LLM to select and format an answer

More RAG – Question Answering (QA)

RetrievalQA chain

```
RetrievalQA.from_chain_type(, chain_type="stuff",...)
```



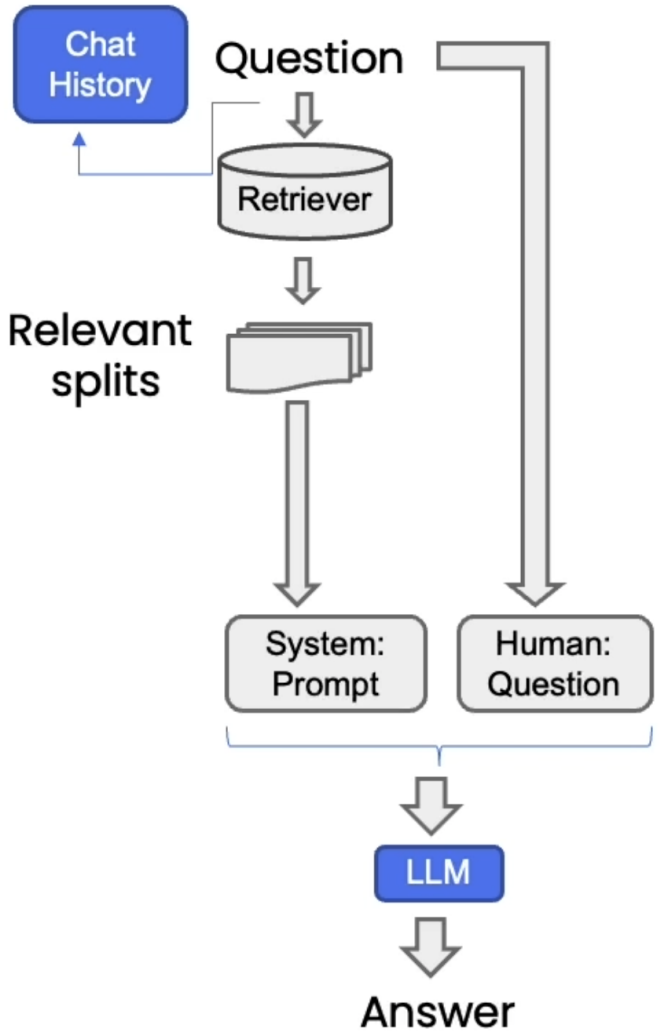
Question is applied to the Vector Store as a query

Vector store provides k relevant documents

Docs and original question are sent to an LLM

More RAG – QA + Chat

```
qa = ConversationalRetrievalChain.from_llm(ChatOpenAI(temperature=0),
vectorstore.as_retriever(), memory=memory)
```



More RAG – QA + Chat

```
# Build prompt
from langchain.prompts import PromptTemplate
template = """Use the following pieces of context to answer the question at th
{context}
Question: {question}
Helpful Answer:"""
QA_CHAIN_PROMPT = PromptTemplate(input_variables=["context", "question"],templ

# Run chain
from langchain.chains import RetrievalQA
question = "Is probability a class topic?"
qa_chain = RetrievalQA.from_chain_type(llm,
                                       retriever=vectordb.as_retriever(),
                                       return_source_documents=True,
                                       chain_type_kwargs={"prompt": QA_CHAIN_P


result = qa_chain({"query": question})
result["result"]
```

```
'Yes, probability is assumed to be a prerequisite for this class. The instruct
or assumes familiarity with basic probability and statistics, and will go over
some of the prerequisites in the discussion sections as a refresher course. Th
anks for asking!'
```

More RAG – QA + Chat

```
from langchain.memory import ConversationBufferMemory
memory = ConversationBufferMemory(
    memory_key="chat_history",
    return_messages=True
)
```

```
from langchain.chains import ConversationalRetrievalChain
retriever=vectordb.as_retriever()
qa = ConversationalRetrievalChain.from_llm(
    llm,
    retriever=retriever,
    memory=memory
)
```

```
question = "Is probability a class topic?"
result = qa({"question": question})
```

```
result["answer"]
```

```
'Yes, probability is a topic that will be assumed to be familiar to students i
n this class. The instructor mentions that basic probability and statistics ar
e prerequisites for the class and assumes that most undergraduate statistics c
lasses, like Stat 116 taught at Stanford, will be more than enough.'
```

```
question = "why are those prerequesites needed?"
result = qa({"question": question})
```

```
result["answer"]
```

```
'The reason for requiring basic probability and statistics as prerequisites fo
r the class is that the class assumes familiarity with these topics. The instr
uctor assumes that most undergraduate statistics classes, like Stat 116 taught
at Stanford, will be more than enough. The class will cover machine learning,
which is a huge field, and assumes that students already know what random vari
ables are, what expectation is, what a variance or a random variable is.'
```

More RAG – QA + Chat

