LangChain
Chat with Your Data

Harrison Chase

LangChain

Overview

Open-source developer framework for building LLM applications

Python and TypeScript packages

Focused on composition and modularity

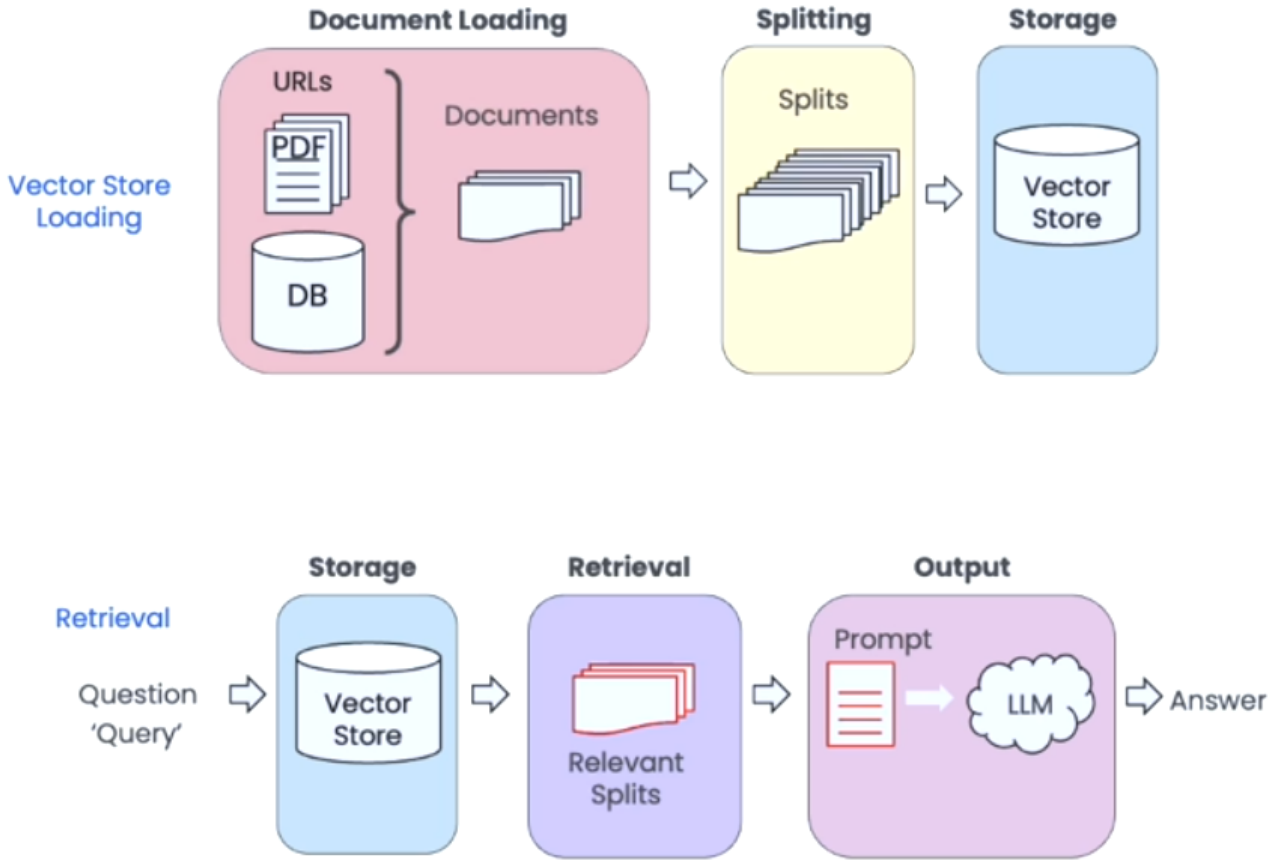
Key value adds:

1. Modular components (and implementations of those components)
2. Use cases - common ways to combine those components together

Components

- Prompts
  - Prompt Templates
  - Output Parsers: 5+ implementations
    - Retry/fixing logic
  - Example Selectors: 5+ implementations
- Models
  - LLM's: 20+ integrations
  - Chat Models
  - Text Embedding Models: 10+ integrations
- Indexes
  - Document Loaders: 50+ implementations
  - Text Splitters: 10+ implementations
  - Vector stores: 10+ integrations
  - Retrievers: 5+ integrations/implementations
- Chains
  - Can be used as building blocks for other chains
  - More application specific chains: 20+ different types
- Agents
  - Agent Types: 5+ types
    - Algorithms for getting LLMs to use tools
  - Agent Toolkits: 10+ implementations
    - Agents armed with specific tools for a specific application

Retrieval Augmented Generation



LangChain

DeepLearning.AI

Acknowledgments

Ankush Gola
LangChain

Lance Martin
LangChain

Geoff Ladwig
DeepLearning.AI

Diala Ezzeddine
DeepLearning.AI

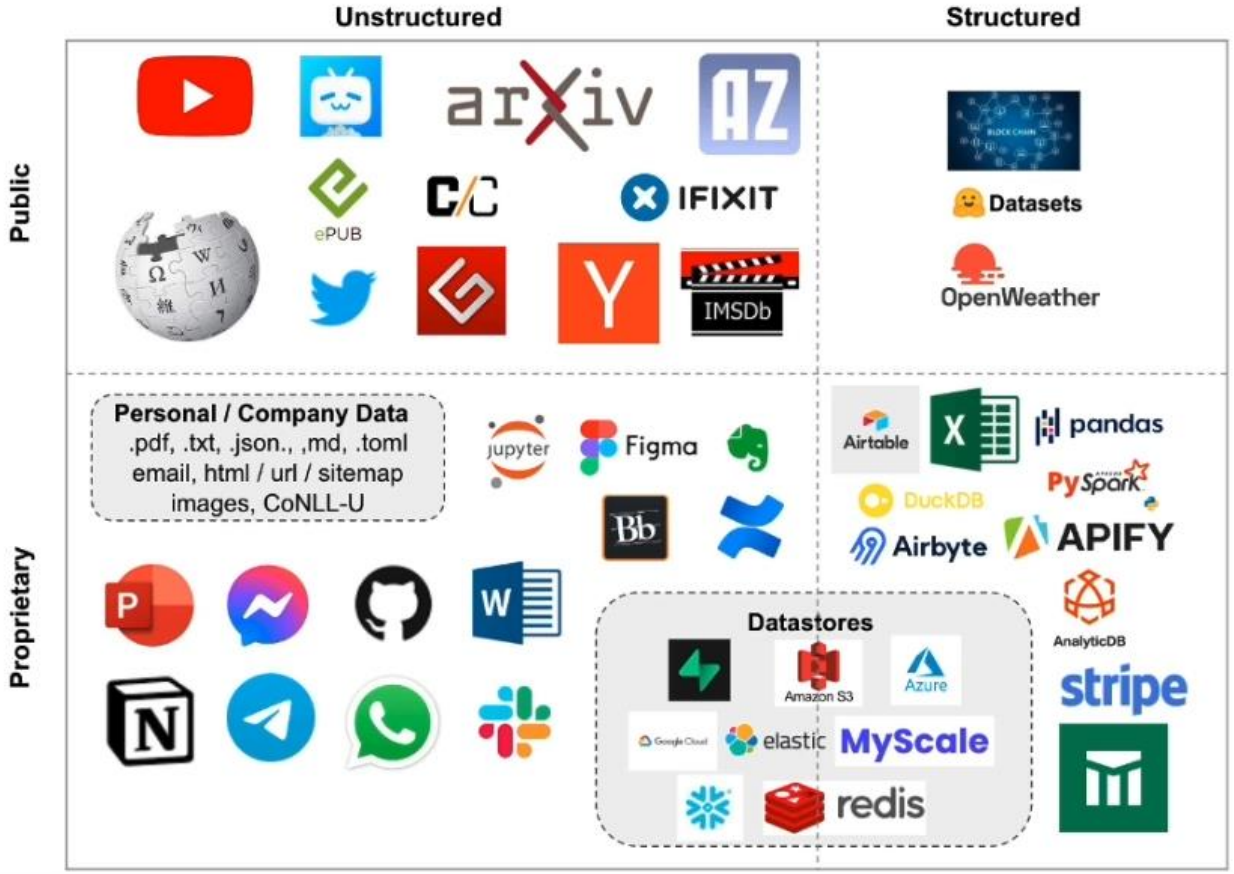
Loaders

- Loaders deal with the specifics of accessing and converting data
  - Accessing
    - Web Sites
    - Data Bases
    - YouTube
    - arXiv
    - ...
  - Data Types
    - PDF
    - HTML
    - JSON
    - Word, PowerPoint...
- Returns a list of `Document` objects:

```
[
Document(page_content='MachineLearning-Lecture01 \nInstructor (Andrew Ng): Okay. Good morning. Welcome to CS229....',
metadata={'source': 'docs/cs229_lectures/MachineLearning-Lecture01.pdf', 'page': 0})
...
 Document(page_content='[End of Audio]  \nDuration: 69 minutes  ',
 metadata={'source': 'docs/cs229_lectures/MachineLearning-Lecture01.pdf', 'page': 21})
]
```

LangChain DeepLearning.AI

Document Loaders



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

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

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

```
len(pages)
```

```
22
```

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

```
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
```

```
page.metadata
```

```
{'source': 'docs/cs229_lectures/MachineLearning-Lecture01.pdf', 'page': 0}
```

```
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
```

```
page.metadata
```

```
{'source': 'docs/cs229_lectures/MachineLearning-Lecture01.pdf', 'page': 0}
```

```
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!
```

```
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"
```

Transcript is done via OpenAI Whisper
https://openai.com/research/whisper

Whisper is an automatic speech recognition (ASR) system trained on 680,000 hours of multilingual and multitask supervised data collected from the web. We show that the use of such a large and diverse dataset leads to improved robustness to accents, background noise and technical language. Moreover, it enables transcription in multiple languages, as well as translation from those languages into English. We are open-sourcing models and inference code to serve as a foundation for building useful applications and for further research on robust speech processing.

Run it localy:
https://www.youtube.com/watch?v=U2TqRKjw1-k

Multitask training data (680k hours)

English transcription

🗣️ "Ask not what your country can do for ⋯"

📝 Ask not what your country can do for ⋯

Any-to-English speech translation

🗣️ "El rápido zorro marrón salta sobre ⋯"

📝 The quick brown fox jumps over ⋯

Non-English transcription

🗣️ "언덕 위에 올라 내려다보면 너무나 넓고 넓은 ⋯"

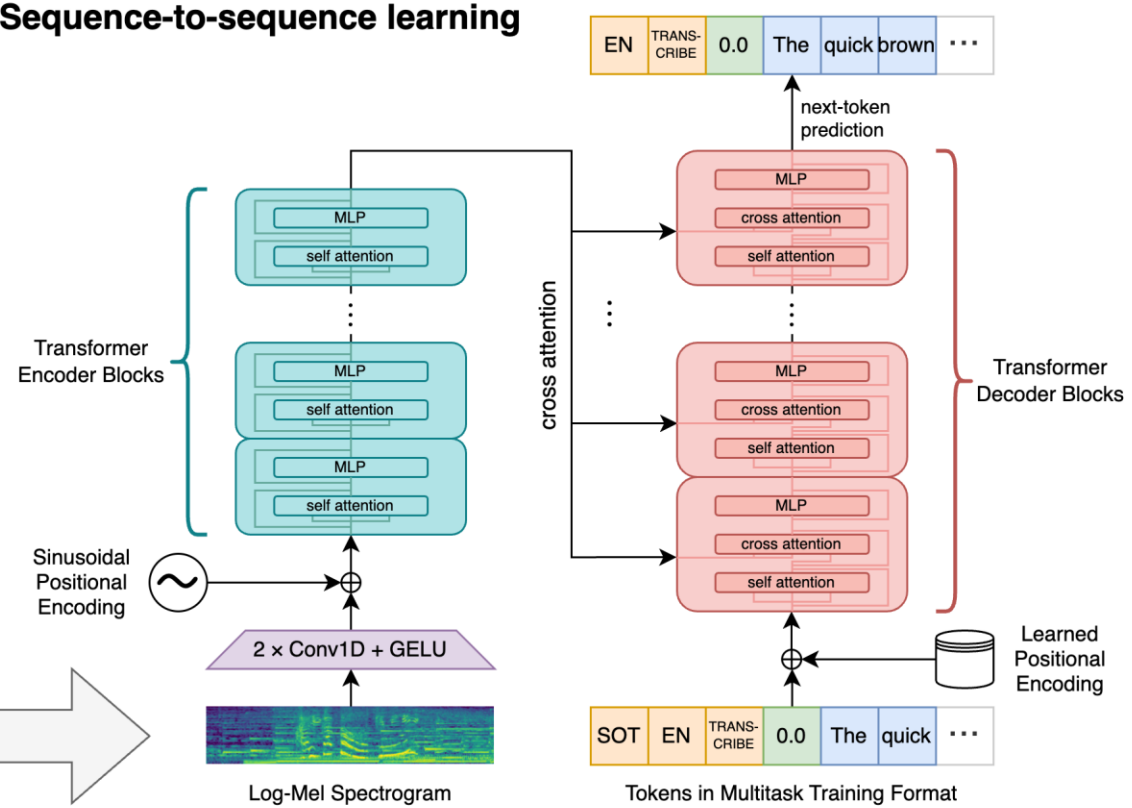
📝 언덕 위에 올라 내려다보면 너무나 넓고 넓은 ⋯

No speech

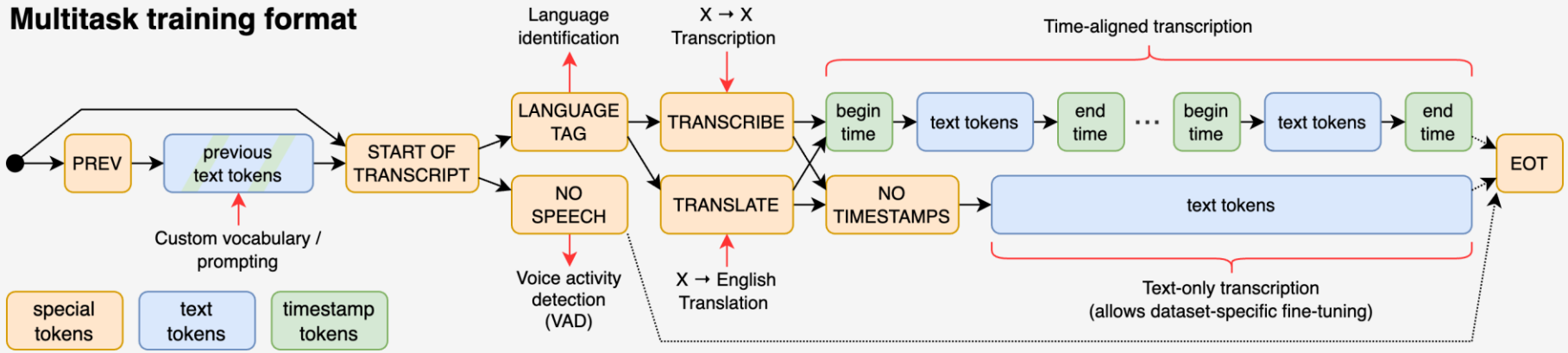
🔊 (background music playing)

📝 ∅

Sequence-to-sequence learning



Multitask training format



```python
from langchain.document_loaders import NotionDirectoryLoader
loader = NotionDirectoryLoader("docs/Notion_DB")
docs = loader.load()
```

```python
print(docs[0].page_content[0:200])
```

```
# Blendle's Employee Handbook

This is a living document with everything we've learned working with people wh
ile running a startup. And, of course, we continue to learn. Therefore it's a
document that
```

LangChain
Chat with Your Data

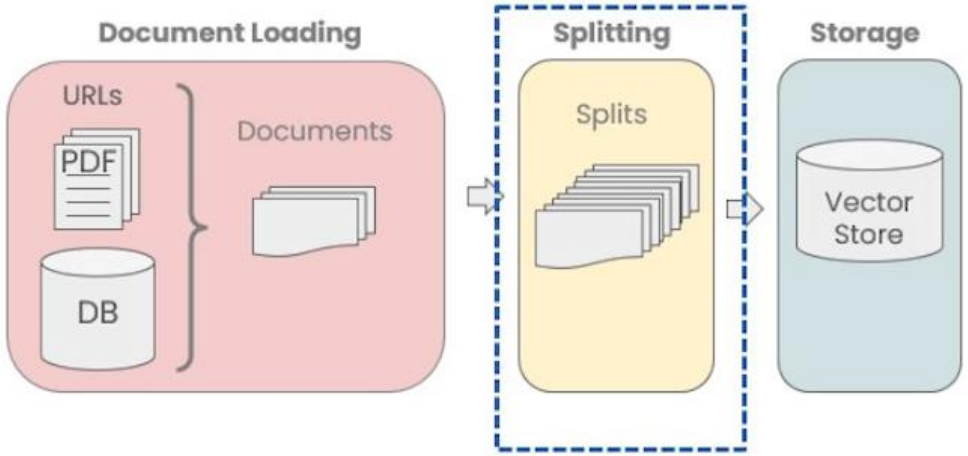
Document Splitting

LangChain

DeepLearning.AI

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?

It’s tricky to split and not to end with sentance spliting to get full information.

Sometime its splitting with chung size and overlapping

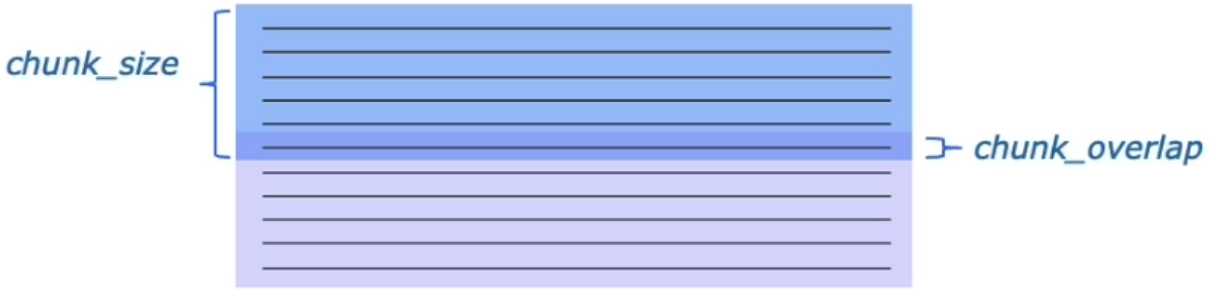
Example Splitter

```
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.



Types of splitters

langchain.text_splitter.

- **CharacterTextSplitter()**- Implementation of splitting text that looks at characters.
- **MarkdownHeaderTextSplitter**() - Implementation of splitting markdown files based on specified headers.
- **TokenTextSplitter**() - Implementation of splitting text that looks at tokens.
- **SentenceTransformersTokenTextSplitter**() - Implementation of splitting text that looks at tokens.
- ***RecursiveCharacterTextSplitter()*** **-** Implementation of splitting text that looks at characters. Recursively tries to split by different characters to find one that works.
- **Language()** – for CPP, Python, Ruby, Markdown etc
- **NLTKTextSplitter**() - Implementation of splitting text that looks at sentences using NLTK (Natural Language Tool Kit)
- **SpacyTextSplitter() -** Implementation of splitting text that looks at sentences using Spacy

```
_ = load_dotenv(find_dotenv()) # read local .env file

openai.api_key  = os.environ['OPENAI_API_KEY']
```

```
from langchain.text_splitter import RecursiveCharacterTextSplitter, CharacterT
```

```
chunk_size = 26
chunk_overlap = 4
```

```
r_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size,chunk_overla
c_splitter = CharacterTextSplitter(chunk_size=chunk_size,chunk_overlap=chunk_o
```

```
text1 = 'abcdefghijklmnopqrstuvwxyz'
```

```
r_splitter.split_text(text1)
```

```
['abcdefghijklmnopqrstuvwxyz']
```

Splitting characters by position, can be empty - space

```
text2 = 'abcdefghijklmnopqrstuvwxyzabcdefg'
```

```
r_splitter.split_text(text2)
```

```
['abcdefghijklmnopqrstuvwxyz', 'wxyzabcdefg']
```

```
text3 = "a b c d e f g h i j k l m n o p q r s t u v w x y z"
```

```
r_splitter.split_text(text3)
```

```
['a b c d e f g h i j k l m', 'l m n o p q r s t u v w x', 'w x y z']
```

There are 3 chunks with overlaping characters like „l m” , „ w x”

```
['abcdefghijklmnopqrstuvwxyz', 'wxyzabcdefg']
```

```
text3 = "a b c d e f g h i j k l m n o p q r s t u v w x y z"
```

```
r_splitter.split_text(text3)
```

```
['a b c d e f g h i j k l m', 'l m n o p q r s t u v w x', 'w x y z']
```

```
c_splitter.split_text(text3)
```

Splitting characters, can be empty - space

```
['a b c d e f g h i j k l m n o p q r s t u v w x y z']
```

```
c_splitter = CharacterTextSplitter(
    chunk_size=chunk_size,
    chunk_overlap=chunk_overlap,
    separator = ' '
)
```

```
c_splitter.split_text(text3)
```

```
['a b c d e f g h i j k l m', 'l m n o p q r s t u v w x', 'w x y z']
```

Splitting characters, separator can be empty space, so there is no empty space in from or end of a chunk

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

```
len(some_text)
```

```
496
```

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

Recursive splitter first splits by “\n\n” and if it needs further to split then in does it with “\n” and further with “ “..

```
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.']
```

It splits with the “ “ and creates

```
 '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 have a period at the end, but also, have a space.and words are separate
d by space.']
```

Recursive splitter first splits by “\n\n” and if it needs further to split then in does it with “\n” and further with “ “..
It’s properly splitted in two chunks just before 450 characters.

```
    chunk_overlap=0,
    separators=["\n\n", "\n", "\. ", " ", ""]
)
```

```
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, closely 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',
 '. Sentences have a period at the end, but also, have a space.and words are s
eparated by space.']
```

```
r_splitter = RecursiveCharacterTextSplitter(
    chunk_size=150,
    chunk_overlap=0,
    separators=["\n\n", "\n", "(?<=\. )", " ", ""]
)
```

```
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, closely related ideas are in sentances. Similar ideas are in pa
ragraphs. 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.',
 'Sentences have a period at the end, but also, have a space.and words are sep
arated by space.']
```

Not properly splited with the “. “

With regex separator sentences are properly splitted.

```python
from langchain.document_loaders import PyPDFLoader
loader = PyPDFLoader("docs/cs229_lectures/MachineLearning-Lecture01.pdf")
pages = loader.load()
```

```python
from langchain.text_splitter import CharacterTextSplitter
text_splitter = CharacterTextSplitter(
    separator="\n",
    chunk_size=1000,
    chunk_overlap=150,
    length_function=len
)
```

```python
docs = text_splitter.split_documents(pages)
```

```python
len(docs)
```

```
77
```

```python
len(pages)
```

```
22
```

Splitting PDF documents

Using method split_documents for PDFs

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

```
from langchain.text_splitter import CharacterTextSplitter
text_splitter = CharacterTextSplitter(
    separator="\n",
    chunk_size=1000,
    chunk_overlap=150,
    length_function=len
)
```

```
docs = text_splitter.split_documents(pages)
```

```
len(docs)
```

```
77
```

```
len(pages)
```

```
22
```

```
from langchain.document_loaders import NotionDirectoryLoader
loader = NotionDirectoryLoader("docs/Notion_DB")
notion_db = loader.load()
```

```
docs = text_splitter.split_documents(notion_db)
```

```
len(notion_db)
```

```
52
```

```
len(docs)
```

```
353
```

Using method for database loading and splitting

```
from langchain.text_splitter import TokenTextSplitter
```

```
text_splitter = TokenTextSplitter(chunk_size=1, chunk_overlap=0)
```

```
text1 = "foo bar bazzyfoo"
```

```
text_splitter.split_text(text1)
```

```
['foo', ' bar', ' b', 'az', 'zy', 'foo']
```

```
text_splitter = TokenTextSplitter(chunk_size=10, chunk_overlap=0)
```

```
docs = text_splitter.split_documents(pages)
```

```
docs[0]
```

```
Document(page_content='MachineLearning-Lecture01  \n', metadata={'source': 'do
cs/cs229_lectures/MachineLearning-Lecture01.pdf', 'page': 0})
```

```
pages[0].metadata
```

```
{'source': 'docs/cs229_lectures/MachineLearning-Lecture01.pdf', 'page': 0}
```

Token splitting

We can also split on token count explicity, if we want.

This can be useful because LLMs often have context windows designated in tokens.

Tokens are often ~4 characters.

Metadata of the source is preserved, so for QA we can do a reference to a document from a chunk.

```python
from langchain.text_splitter import MarkdownHeaderTextSplitter
```

```python
markdown_document = """# Title\n\n \
## Chapter 1\n\n \
Hi this is Jim\n\n Hi this is Joe\n\n \
### Section \n\n \
Hi this is Lance \n\n
## Chapter 2\n\n \
Hi this is Molly"""
```

```python
headers_to_split_on = [
    ("#", "Header 1"),
    ("##", "Header 2"),
    ("###", "Header 3"),
]
```

```python
markdown_splitter = MarkdownHeaderTextSplitter(
    headers_to_split_on=headers_to_split_on
)
md_header_splits = markdown_splitter.split_text(markdown_document)
```

```python
md_header_splits[0]
```

```
Document(page_content='Hi this is Jim  \nHi this is Joe', metadata={'Header
1': 'Title', 'Header 2': 'Chapter 1'})
```

```python
md_header_splits[1]
```

```
Document(page_content='Hi this is Lance', metadata={'Header 1': 'Title', 'Head
er 2': 'Chapter 1', 'Header 3': 'Section'})
```

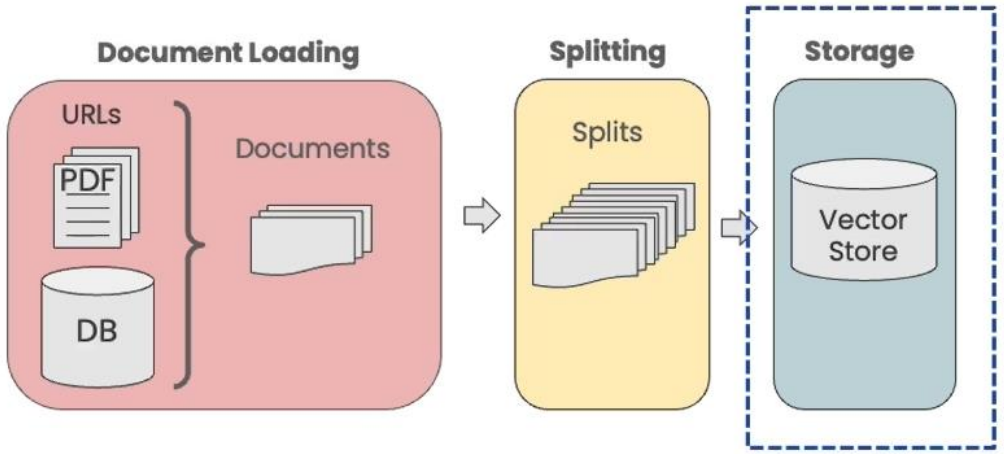
LangChain
Chat with Your Data

Vector Stores and Embeddings

LangChain

DeepLearning.AI

Vector Stores

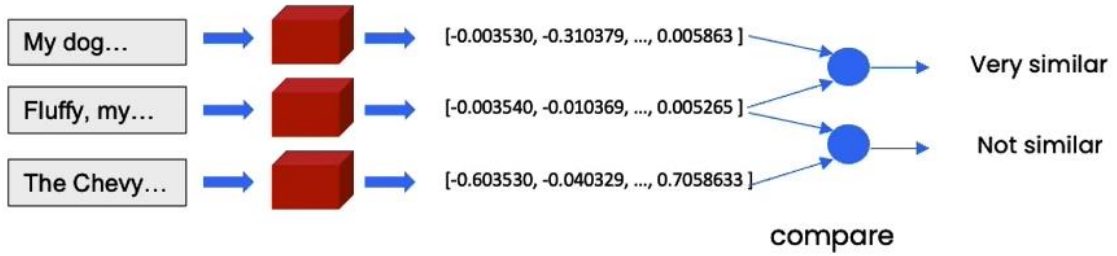


LangChain DeepLearning.AI

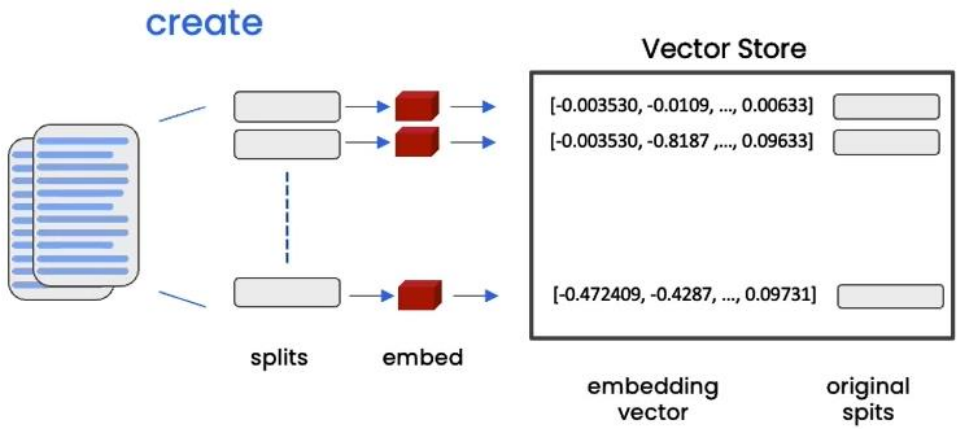
Embeddings

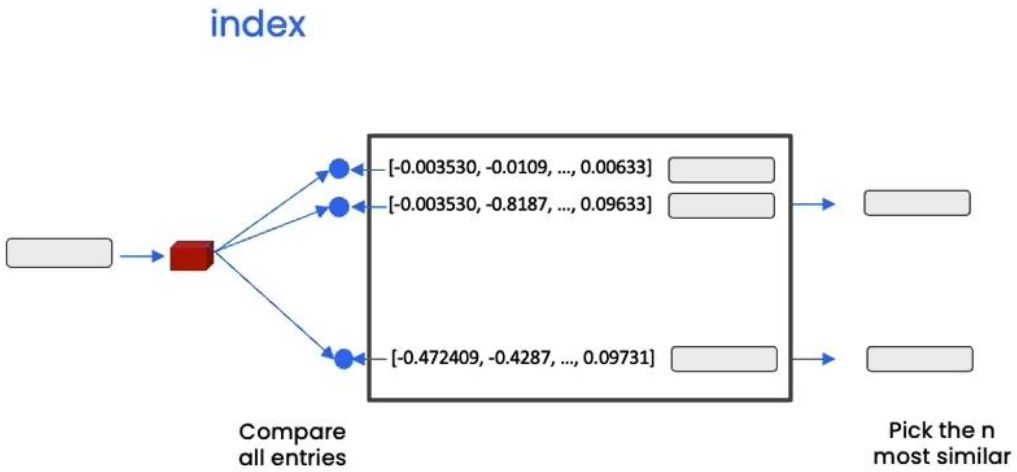
- Embedding vector captures content/meaning
- Text with similar content will have similar vectors

1) My dog Rover likes to chase squirrels.
2) Fluffy, my cat, refuses to eat from a can.
3) The Chevy Bolt accelerates to 60 mph in 6.7 seconds.



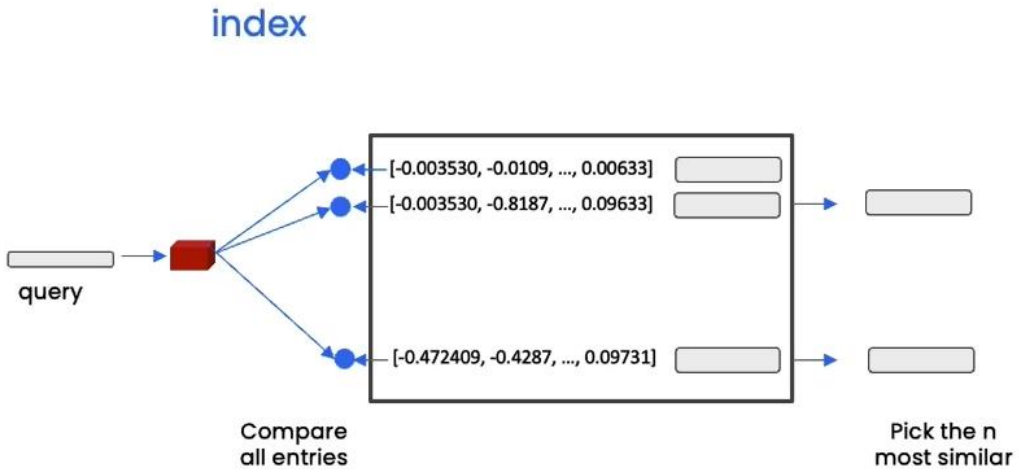
Vector Store

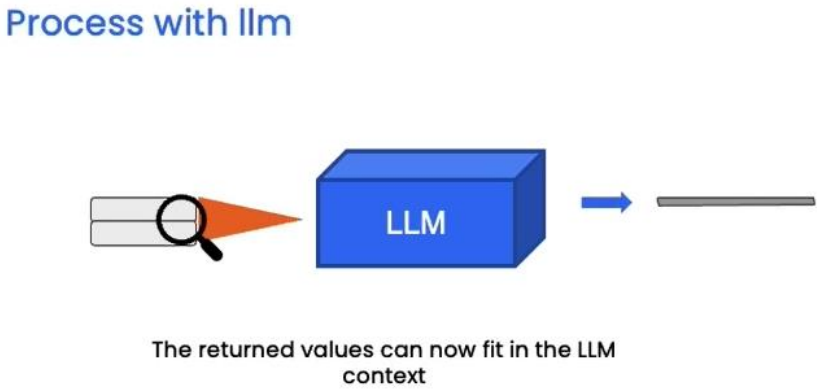




LangChain DeepLearning.AI

Vector Store/Database





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

```
sentence1 = "i like dogs"
sentence2 = "i like canines"
sentence3 = "the weather is ugly outside"
```

```
embedding1 = embedding.embed_query(sentence1)
embedding2 = embedding.embed_query(sentence2)
embedding3 = embedding.embed_query(sentence3)
```

```
import numpy as np
```

```
np.dot(embedding1, embedding2)
```

```
0.9631853877103519
```

```
np.dot(embedding1, embedding3)
```

```
0.770999765129468
```

```
np.dot(embedding2, embedding3)
```

```
0.7596334120325541
```

```
from langchain.vectorstores import Chroma
```

```
persist_directory = 'docs/chroma/'
```

```
!rm -rf ./docs/chroma
```

```
vectordb = Chroma.from_documents(
    documents=splits,
    embedding=embedding,
    persist_directory=persist_directory
)
```

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

```
209
```

```
question = "is there an email i can ask for help"
```

```
docs = vectordb.similarity_search(question, k=3)
```

```
len(docs)
```

```
3
```

```
print(docs[0].page_content)
```

```
cs229-qa@cs.stanford.edu. This goes to an acc ount that's read by all the TAs
and me. So
rather than sending us email individually, if you send email to this account,
it will
actually let us get back to you maximally quickly with answers to your questio
ns.
If you're asking questions abt homework probl ems, please say in the subject
line which
assignment and which question the email refers to, since that will also help u
s to route
your question to the appropriate TA or to me  appropriately and get the respon
se back to
you quickly.
Let's see. Skipping ahead — let's see — for homework, one midterm, one open an
d term
project. Notice on the honor code. So one thi ng that I think will help you to
succeed and
do well in this class and even help you to enjoy this cla ss more is if you fo
rm a study
group.
So start looking around where you' re sitting now or at the end of class toda
y, mingle a
little bit and get to know your classmates. I strongly encourage you to form s
tudy groups
and sort of have a group of people to study with and have a group of your fell
ow students
to talk over these concepts with. You can also  post on the class news group i
f you want to
use that to try to form a study group.
But some of the problems sets in this cla ss are reasonably difficult.  People
```

```
vectordb.persist()
```

```
question = "what did they say about matlab?"
```

```
docs = vectordb.similarity_search(question,k=5)
```

```
docs[0]
```

```
Document(page_content='those homeworks will be done in either MATLA B or in Oc
tave, which is sort of — I \nknow some people call it a free ve rsion of MATLA
B, which it sort  of is, sort of isn\'t.  \nSo I guess for those of you that h
aven\'t s een MATLAB before, and I know most of you \nhave, 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 \nplot
data. And it\'s sort of an extremely easy to  learn tool to use for implementi
ng a lot of \nlearning algorithms.  \nAnd in case some of you want to work on
your  own home computer or something if you \ndon\'t have a MATLAB license, fo
r the purposes of  this class, there\'s also — [inaudible] \nwrite that down
[inaudible] MATLAB — there\' s also a software package called Octave \nthat yo
u can download for free off the Internet. And it has somewhat fewer features t
han MATLAB, but it\'s free, and for the purposes of  this class, it will work
for just about \neverything.  \nSo actually I, well, so yeah, just a side comm
ent for those of you that haven\'t seen \nMATLAB before I guess, once a collea
gue of mine at a different univers!ty, not at \nStanford, actually teaches ano
ther machine l earning course. He\'s taught it for many years. \nSo one day, h
e was in his office, and an old student of his from, lik e, ten years ago came
\ninto his office and he said, "Oh, professo r, professor, thank you so much f
or your', metadata={'source': 'docs/cs229_lectures/MachineLearning-Lecture01.p
df', 'page': 8})
```

```
docs[1]
```

```
Document(page_content='those homeworks will be done in either MATLA B or in Oc
tave, which is sort of — I \nknow some people call it a free ve rsion of MATLA
B, which it sort  of is, sort of isn\'t.  \nSo I guess for those of you that h
aven\'t s een MATLAB before, and I know most of you \nhave, 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 \nplot
data. And it\'s sort of an extremely easy to  learn tool to use for implementi
ng a lot of \nlearning algorithms.  \nAnd in case some of you want to work on
your  own home computer or something if you \ndon\'t have a MATLAB license, fo
r the purposes of  this class, there\'s also — [inaudible] \nwrite that down
[inaudible] MATLAB — there\' s also a software package called Octave \nthat yo
```

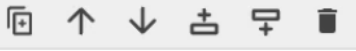
Two same answers because 2 same documents were loaded, and 2 chunks were returned.

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

```
docs = vectordb.similarity_search(question,k=5)
```

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

```
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 0}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 14}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture02.pdf', 'page': 0}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 6}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture01.pdf', 'page': 8}
```



```
docs = vectordb.similarity_search(question,k=5)
```

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

```
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 0}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 14}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture02.pdf', 'page': 0}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 6}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture01.pdf', 'page': 8}
```

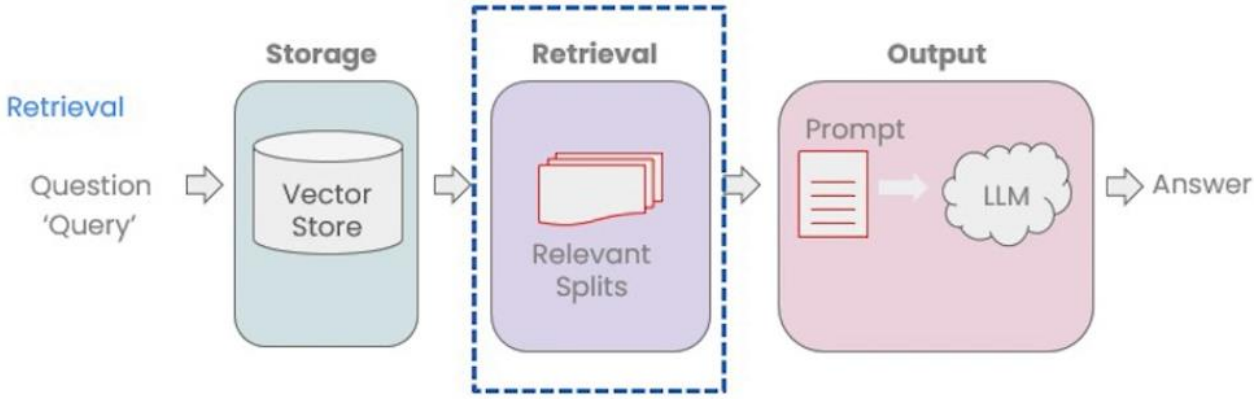
```
print(docs[4].page_content)
```

```
into his office and he said, "Oh, professo r, professor, thank you so much for
your
machine learning class. I learned so much from it. There's this stuff that I l
earned in your
class, and I now use every day. And it's help ed me make lots of money, and he
re's a
picture of my big house."
So my friend was very excited. He said, "W ow. That's great. I'm glad to hear
this
machine learning stuff was actually useful. So what was it that you learned? W
as it
logistic regression? Was it the PCA? Was it the data ne tworks? What was it th
at you
learned that was so helpful?" 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 n
ot hard,
and we'll actually have a short MATLAB tutori al in one of the discussion sect
ions for
those of you that don't know it.
Okay. The very last piece of logistical th ing is the discussion s ections. So
discussion
sections will be taught by the TAs, and atte ndance at discussion sections is
optional,
although they'll also be recorded and televi sed. And we'll use the discussion
sections
mainly for two things. For the next two or th ree weeks, we'll use the discuss
ion sections
to go over the prerequisites to this class or if some of you haven't seen prob
ability or
```

LangChain
Chat with your data

Retrieval

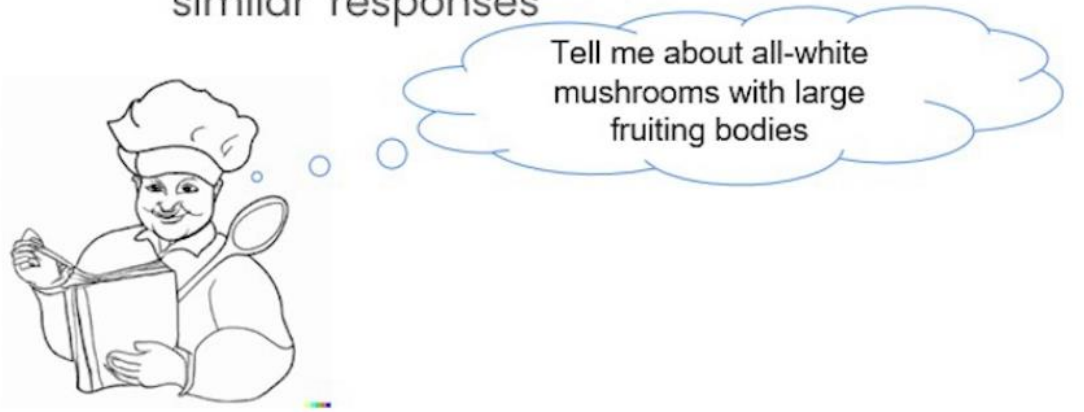
Retrieval

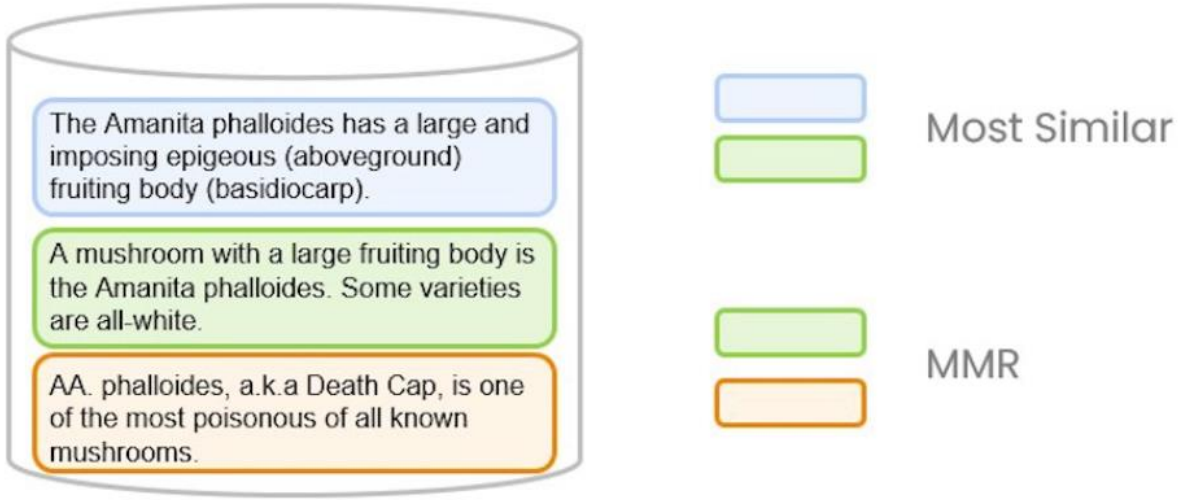


LangChain DeepLearning.AI

Maximum marginal relevance(MMR)

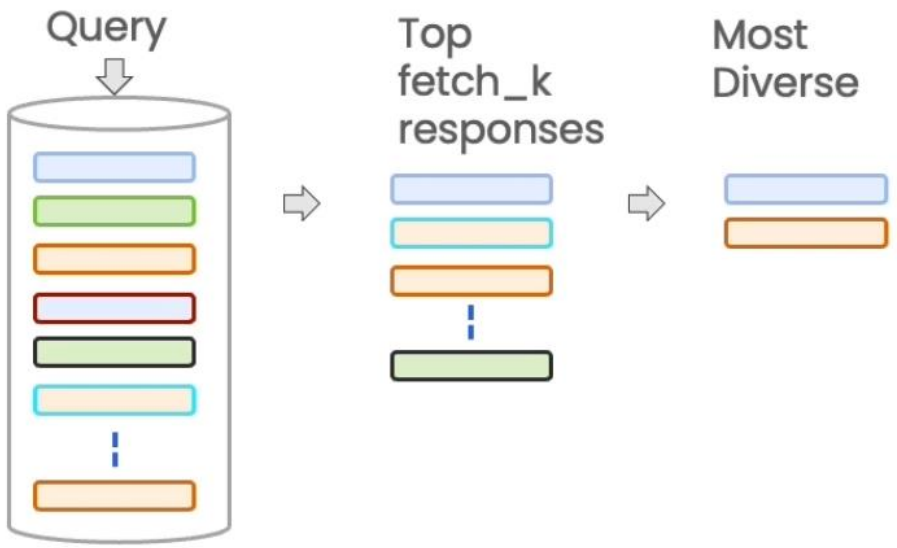
- You may not always want to choose the most similar responses





MMR algorithm

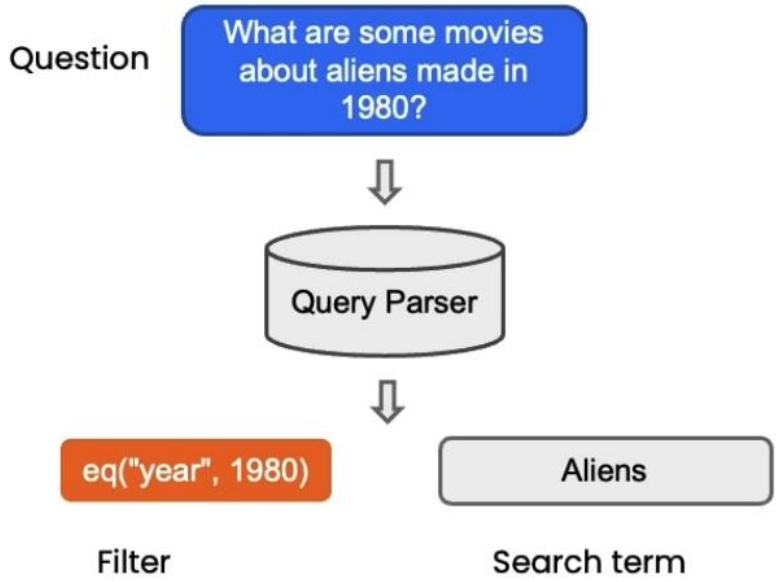
- Query the Vector Store
- Choose the `fetch_k` most similar responses
- Within those responses choose the `k` most diverse



LangChain DeepLearning.AI

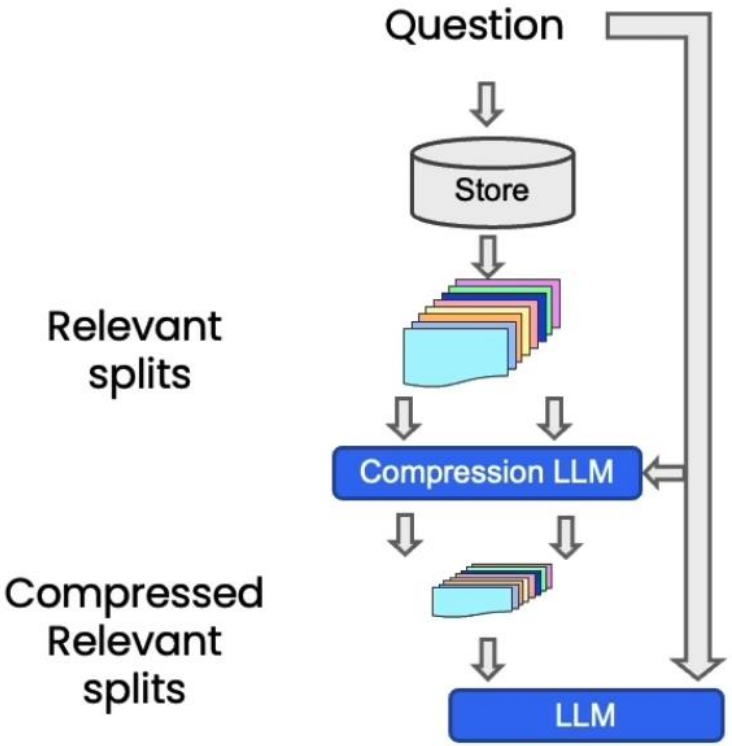
LLM Aided Retrieval

- 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



Compression

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



```
import os
import openai
import sys
sys.path.append('../..')

from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file

openai.api_key  = os.environ['OPENAI_API_KEY']
```

```
from langchain.vectorstores import Chroma
from langchain.embeddings.openai import OpenAIEmbeddings
persist_directory = 'docs/chroma/'
```

```
embedding = OpenAIEmbeddings()
vectordb = Chroma(
    persist_directory=persist_directory,
    embedding_function=embedding
)
```

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

```
209
```

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

```
smalldb = Chroma.from_texts(texts, embedding=embedding)
```

```
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={})]
```

```
question = "what did they say about matlab?"
docs_ss = vectordb.similarity_search(question,k=3)
```

```
sonous of all known mushrooms.', metadata={})]
```

```
question = "what did they say about matlab?"
docs_ss = vectordb.similarity_search(question,k=3)
```

```
docs_ss[0].page_content[:100]
```

```
'those homeworks will be done in either MATLA B or in Octave, which is sort of
— I \nknow some people '
```

```
docs_ss[1].page_content[:100]
```

```
'those homeworks will be done in either MATLA B or in Octave, which is sort of
— I \nknow some people '
```

```
docs_mmr = vectordb.max_marginal_relevance_search(question,k=3)
```

```
docs_mmr[0].page_content[:100]
```

```
'those homeworks will be done in either MATLA B or in Octave, which is sort of
— I \nknow some people '
```

```
docs_mmr[1].page_content[:100]
```

```
'algorithm then? So what's different? How come  I was making all that noise ea
rlier about \nleast squa'
```

```
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"})
```

Manually defined source

```
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}
```

```
from langchain.llms import OpenAI
from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain.chains.query_constructor.base import AttributeInfo
```

LLM auto defined source

```
metadata_field_info = [
    AttributeInfo(
        name="source",
        description="The lecture the chunk is from, should be one of `docs/cs2
        type="string",
    ),
    AttributeInfo(
        name="page",
        description="The page from the lecture",
        type="integer",
    ),
]
```

Metadata info for source and page is defined

```
metadata_field_info = [
    AttributeInfo(
        name="source",
        description="The lecture the chunk is from, should be one of `docs/cs2
        type="string",
    ),
    AttributeInfo(
        name="page",
        description="The page from the lecture",
        type="integer",
    ),
]
```

```
document_content_description = "Lecture notes"
llm = OpenAI(temperature=0)
retriever = SelfQueryRetriever.from_llm(
    llm,
    vectordb,
    document_content_description,
    metadata_field_info,
    verbose=True
)
```

Self query retrieval defines the source and page that we want to search and return, it will be done by LLM

```
docs = retriever.get_relevant_documents(question)
```

```
query='regression' filter=Comparison(comparator=<Comparator.EQ: 'eq'>, attribu
te='source', value='docs/cs229_lectures/MachineLearning-Lecture03.pdf') limit=
None
```

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

```
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 14}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 0}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 10}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 10}
```

Self query retrieval shows from which documents the answer is produced

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

```python
def pretty_print_docs(docs):
    print(f"\n{'-' * 100}\n".join([f"Document {i+1}:\n\n" + d.page_content for
```

```python
llm = OpenAI(temperature=0)
compressor = LLMChainExtractor.from_llm(llm)
```

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

```python
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
```

ContextualCompressionRetrival compress answers from all documents. LLMChainExtractor will return only relevant parts of document

We got shorter documents, but it can be repeated

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

```python
def pretty_print_docs(docs):
    print(f"\n{'-' * 100}\n".join([f"Document {i+1}:\n\n" + d.page_content for
```

```python
llm = OpenAI(temperature=0)
compressor = LLMChainExtractor.from_llm(llm)
```

```python
compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=vectordb.as_retriever(search_type="mmr")
)
```

With MMR we will not get duplicate documents

```python
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:

"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."
```

Other types of retrieval

Not using a vector database, such as:

- SVM
- TF-IDF
- ...

Support Vector Machine retrieval, term frequency–inverse document frequency, is a measure of importance of a word to a document in a collection or corpus, adjusted for the fact that some words appear more frequently in general.

```
from langchain.retrievers import SVMRetriever
from langchain.retrievers import TFIDFRetriever
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

```
# Load PDF
loader = PyPDFLoader("docs/cs229_lectures/MachineLearning-Lecture01.pdf")
pages = loader.load()
all_page_text=[p.page_content for p in pages]
joined_page_text=" ".join(all_page_text)

# Split
text_splitter = RecursiveCharacterTextSplitter(chunk_size = 1500,chunk_overlap
splits = text_splitter.split_text(joined_page_text)
```

```
svm_retriever = SVMRetriever.from_texts(splits,embedding)
tfidf_retriever = TFIDFRetriever.from_texts(splits)
```

```
question = "what did they say about matlab?"
docs_svm=svm_retriever.get_relevant_documents(question)
docs_svm[0]
```

```
Document(page_content='don\'t have a MATLAB license, for the purposes of  this class, there\'s also — [inaudible] \nwrite that down [inaudible] MATLAB — ther e\' s also a software package called Octave \nthat you can download for free o ff the Internet. And it has somewhat fewer features than MATLAB, but it\'s fre e, and for the purposes of  this class, it will work for just about \neverythi ng.  \nSo actually I, well, so yeah, just a side comment for those of you that haven\'t seen \nMATLAB before I guess, once a colleague of mine at a different university, not at \nStanford, actually teaches another machine l earning cours se. He\'s taught it for many years. \nSo one day, he was in his office, and an old student of his from, lik e, ten years ago came \ninto his office and he sa id, "Oh, professo r, professor, thank you so much for your \nmachine learning class. I learned so much from it. There\'s this stuff that I learned in your \nclass, and I now use every day. And it\'s help ed me make lots of money, and here\'s a \npicture of my big house."  \nSo my friend was very excited. He sai d, "W ow. That\'s great. I\'m glad to hear this \nmachine learning stuff was a ctually useful. So what was it that you learned? Was it \nlogistic regression? Was it the PCA? Was it the data ne tworks? What was it that you \nlearned that was so helpful?" And the student said, "Oh, it was the MATLAB."  \nSo for thos e of you that don\'t know MATLAB yet, I hope you do learn it. It\'s not har d,', metadata={})
```

```
question = "what did they say about matlab?"
docs_tfidf=tfidf_retriever.get_relevant_documents(question)
docs_tfidf[0]
```

```
Document(page_content="Saxena and Min Sun here did, wh ich is given an image l ike this, right? This is actually a \npicture taken of the Stanford campus. Yo u can apply that sort of cl ustering algorithm and \ngroup the picture into re gions. Let me actually blow that up so that you can see it more \nclearly. Oka y. So in the middle, you see the lines sort of groupi ng the image together, \ngrouping the image into [inaudible] regions.  \nAnd what Ashutosh and Min di d was they then  applied the learning algorithm to say can \nwe take this clus tering and us e it to build a 3D model of the world? And so using the \ncluste ring, they then had a lear ning algorithm try to learn what the 3D structure o f the \nworld looks like so that they could come up with a 3D model that you c an sort of fly \nthrough, okay? Although many people used to th ink it's not p ossible to take a single \nimage and build a 3D model, but using a lear ning a lgorithm and that sort of clustering \nalgorithm is the first step. They were able to.  \nI'll just show you one more example. I like this  because it's a p icture of Stanford with our \nbeautiful Stanford campus. So again, taking th e same sort of clustering algorithms, taking \nthe same sort of unsupervised lea rning algor ithm, you can group the pixels into different \nregions. And using that as a pre-processing step, they eventually built this sort of 3D model of Stanford campus in a single picture.  You can sort of walk  into the ceiling, look", metadata={})
```

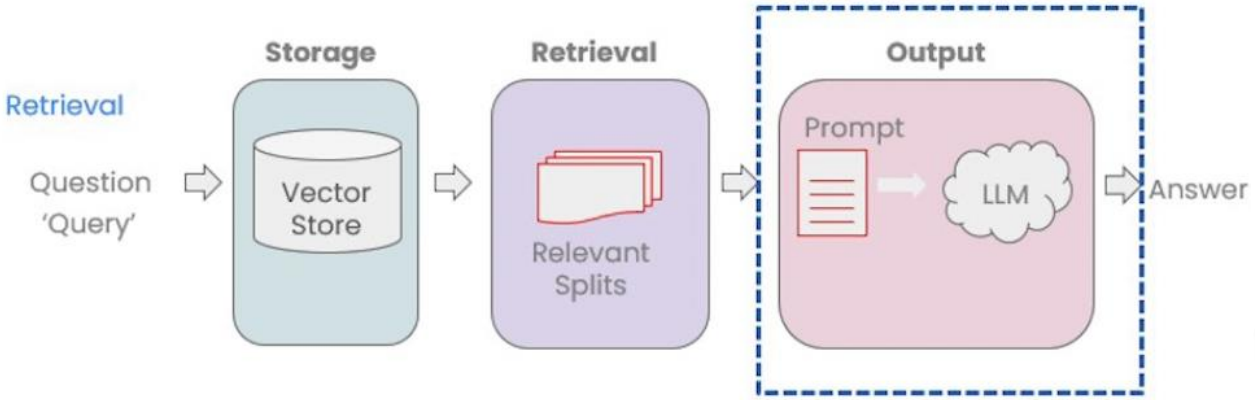
LangChain
Chat with Your Data

Question Answering ChatBot

LangChain

DeepLearning.AI

Question Answering

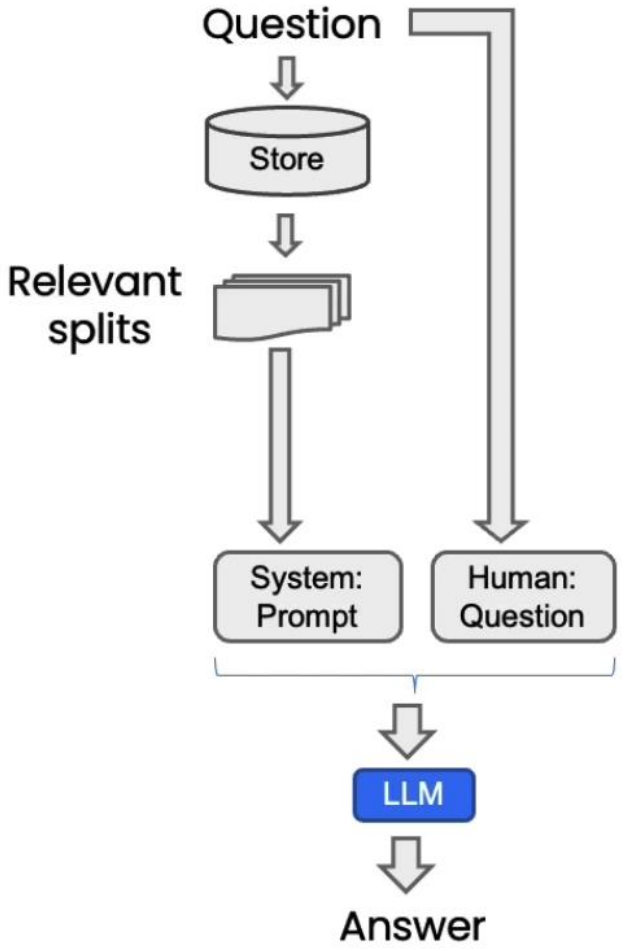


- Multiple relevant documents have been retrieved from the vector store
- Potentially compress the relevant splits to fit into the LLM context
- Send the information along with our question to an LLM to select and format an answer

LangChain DeepLearning.AI

RetrievalQA chain

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



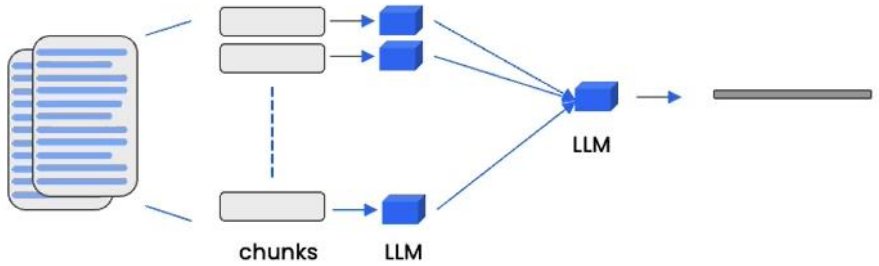
Question is applied to the Vector Store as a query

Vector store provides k relevant documents

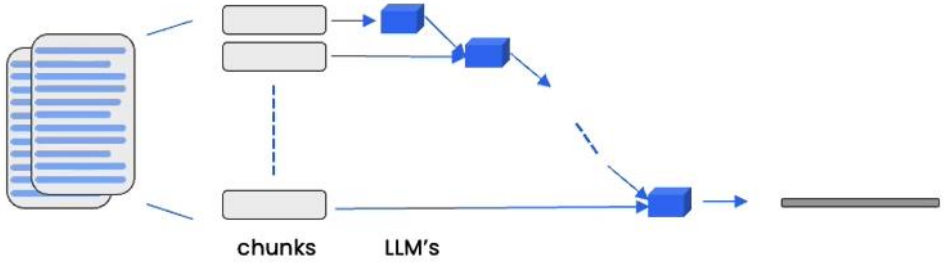
Docs and original question are sent to an LLM

3 additional methods

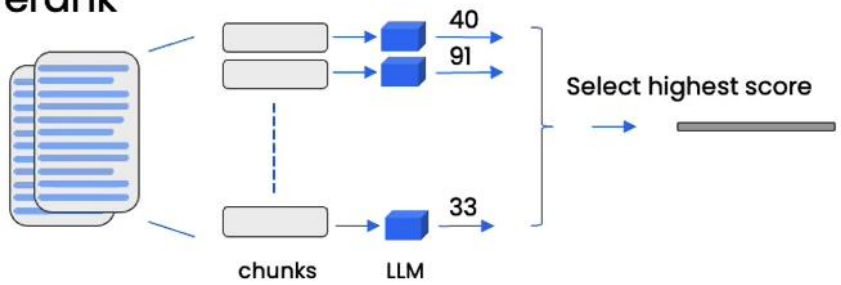
1. Map_reduce



2. Refine



3. Map_rerank



Multiple methods to use to tackle issue of short context window

```
import openai
import sys
sys.path.append('../..')

from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file

openai.api_key  = os.environ['OPENAI_API_KEY']
```

```
from langchain.vectorstores import Chroma
from langchain.embeddings.openai import OpenAIEmbeddings
persist_directory = 'docs/chroma/'
embedding = OpenAIEmbeddings()
vectordb = Chroma(persist_directory=persist_directory, embedding_function=embe
```

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

```
209
```

```
question = "What are major topics for this class?"
docs = vectordb.similarity_search(question,k=3)
len(docs)
```

```
3
```

```
from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)
```

```
from langchain.chains import RetrievalQA
```

```
qa_chain = RetrievalQA.from_chain_type(
    llm,
    retriever=vectordb.as_retriever()
)
```

```
qa_chain = RetrievalQA.from_chain_type(
    llm,
    retriever=vectordb.as_retriever()
)
```

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

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

```
'The major topic for this class is machine learning. Additionally, the class m
ay cover statistics and algebra as refreshers in the discussion sections. Late
r in the quarter, the discussion sections will also cover extensions for the m
aterial taught in the main lectures.'
```

Temperature 0 for lowest variability.

Import the RetriavalQA chain to send result chunks to LLM for answer

```
aterial taught in the main lectures.
```

```
from langchain.prompts import PromptTemplate

# Build prompt
template = """Use the following pieces of context to answer the question at th
{context}
Question: {question}
Helpful Answer:"""
QA_CHAIN_PROMPT = PromptTemplate.from_template(template)
```

```
qa_chain = RetrievalQA.from_chain_type(
    llm,
    retriever=vectordb.as_retriever(),
    return_source_documents=True,
    chain_type_kwargs={"prompt": QA_CHAIN_PROMPT}
)
```

```
question = "Is probability a class topic?"
```

```
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!'
```

```
result["source_documents"][0]
```

```
    chain_type_kwargs={"prompt": QA_CHAIN_PROMPT}
)
```

```
question = "Is probability a class topic?"
```

```
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!'
```

```
result["source_documents"][0]
```

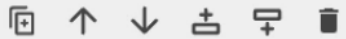
```
Document(page_content="of this class will not be very program ming intensive,
although we will do some \nprogramming, mostly in either MATLAB or Octa ve.
I'll say a bit more about that later.  \nI also assume familiarity with basic
proba bility and statistics. So most undergraduate \nstatistics class, like St
at 116 taught here at Stanford, will be more than enough. I'm gonna \nassume a
ll of you know what ra ndom variables are, that all of you know what expectati
on \nis, what a variance or a random variable is. And in case of some of you,
it's been a while \nsince you've seen some of this material. At some of the di
scussion sections, we'll actually \ngo over some of the prerequisites, sort of
as  a refresher course under prerequisite class. \nI'll say a bit more about t
hat later as well.  \nLastly, I also assume familiarity with basi c linear alg
ebra. And again, most undergraduate \nlinear algebra courses are more than eno
ugh. So if you've taken courses like Math 51, \n103, Math 113 or CS205 at Stan
ford, that would be more than enough. Basically, I'm \ngonna assume that all o
f you know what matrix es and vectors are, that you know how to \nmultiply mat
rices and vectors and multiply matrix and matrices, that you know what a matri
x inverse is. If you know what an eigenvect or of a matrix is, that'd be even
better. \nBut if you don't quite know or if you're not qu ite sure, that's fin
e, too. We'll go over it in \nthe review sections.", metadata={'source': 'doc
s/cs229_lectures/MachineLearning-Lecture01.pdf', 'page': 4})
```

```
qa_chain_mr = RetrievalQA.from_chain_type(
    llm,
    retriever=vectordb.as_retriever(),
    chain_type="map_reduce"
)
```

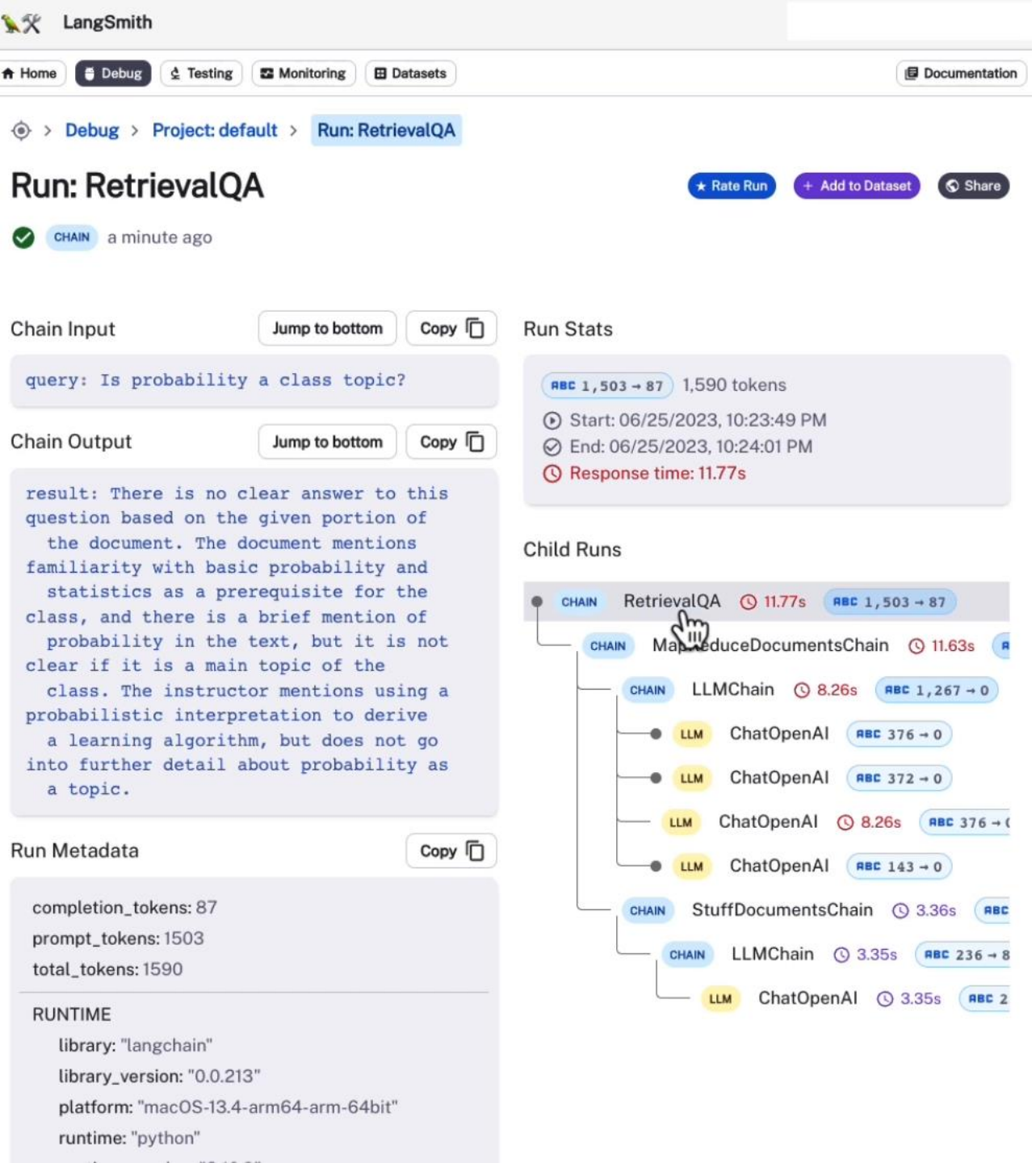
```
result = qa_chain_mr({"query": question})
```

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

```
'There is no clear answer to this question based on the given portion of the d
ocument. The document mentions familiarity with basic probability and statisti
cs as a prerequisite for the class, and there is a brief mention of probabilit
y in the text, but it is not clear if it is a main topic of the class. The ins
tructor mentions using a probabilistic interpretation to derive a learning alg
orithm, but does not go into further detail about probability as a topic.'
```



- It can take multiple documents
- It takes a lot slower
- Response is even worse, it’s responding based on different documents, context is distributed over multiple documents



- Check with LangSmith what is happening with map reduce

```python
import os
os.environ["LANGCHAIN_TRACING_V2"] = "true"
os.environ["LANGCHAIN_ENDPOINT"] = "https://api.langchain.plus"
os.environ["LANGCHAIN_API_KEY"] = LCP_API_KEY
```

```python
qa_chain_mr = RetrievalQA.from_chain_type(
    llm,
    retriever=vectordb.as_retriever(),
    chain_type="map_reduce"
)
result = qa_chain_mr({"query": question})
result["result"]
```

```
'There is no clear answer to this question based on the given portion of the d
ocument. The document mentions familiarity with basic probability and statisti
cs as a prerequisite for the class, and there is a brief mention of probabilit
y in the text, but it is not clear if it is a main topic of the class. The ins
tructor mentions using a probabilistic interpretation to derive a learning alg
orithm, but does not go into further detail about probability as a topic.'
```

```python
qa_chain_mr = RetrievalQA.from_chain_type(
    llm,
    retriever=vectordb.as_retriever(),
    chain_type="refine"
)
result = qa_chain_mr({"query": question})
result["result"]
```

```
"The class assumes familiarity with basic probability and statistics, but will
have review sections to refresh the prerequisites. The instructor also mention
s that they will use the discussion sections to go over extensions for the mat
erial that they are teaching in the main lectures. These extensions will cover
topics that they didn't have time to cover in the main lectures, as machine le
arning is a vast field."
```

Refine retrieval method gives better results than map reduce because it combines information sequentially and takeovers more information than map reduce

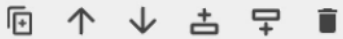
```
a_chain = RetrievalQA.from_chain_type(
    llm,
    retriever=vectordb.as_retriever()
```

```
uestion = "Is probability a class topic?"
esult = qa_chain({"query": question})
esult["result"]
```

```
Yes, probability is a topic that will be assumed to be familiar to students i
 this class. The instructor assumes that students have a basic understanding
f probability and statistics, and will go over some of the prerequisites as a
efresher course in the discussion sections.'
```

```
uestion = "why are those prerequesites needed?"
esult = qa_chain({"query": question})
esult["result"]
```

```
The prerequisites for the class are assumed to be basic knowledge of computer
cience and basic computer skills and principles. The instructor assumes that
ll students know about big-O notation and have a basic understanding of compu
er science. These prerequisites are needed to ensure that all students can fo
low the technical content of the class.'
```



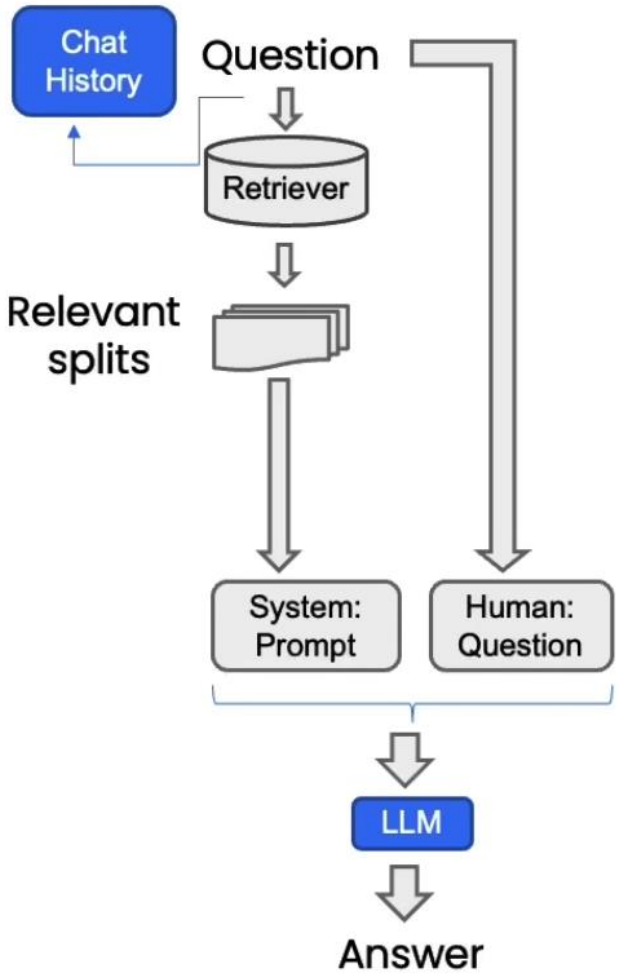
Not a good answer as the CHAIN does not have a memory what was a previous question or answer

LangChain
Chat with Your Data

Question Answering ChatBot

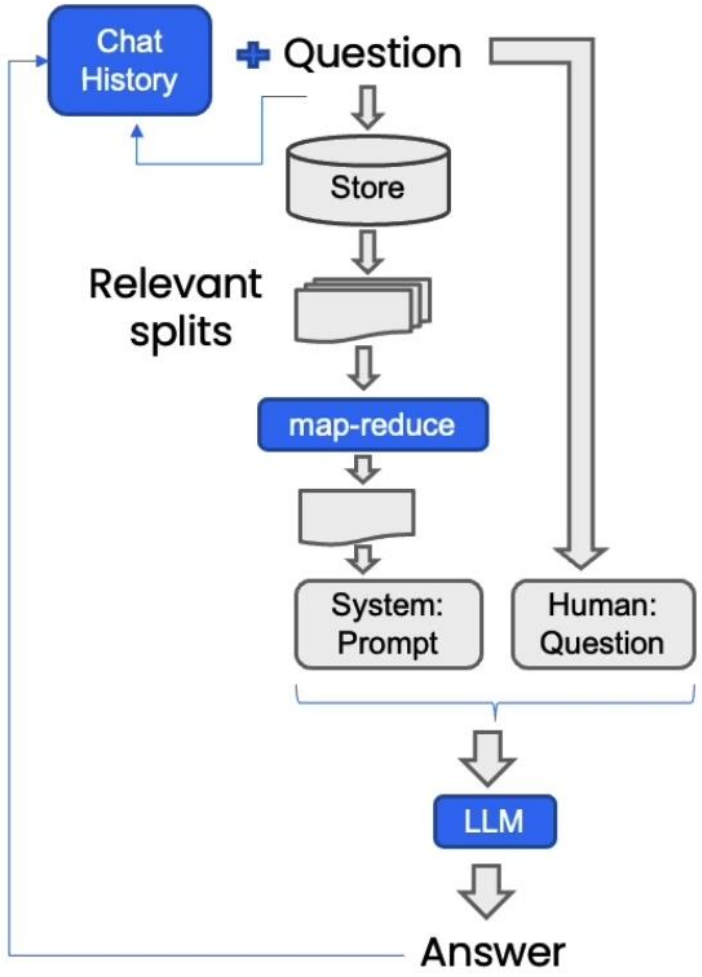
ConversationalRetrievalChain

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



Chat history will give context from previous answers/question and continue the conversation with followup.

ConversationalRetrievalChain



You can add additional Retriever and compression features as needed

All types of retrival can be used.

```
from langchain.vectorstores import Chroma
from langchain.embeddings.openai import OpenAIEmbeddings
persist_directory = 'docs/chroma/'
embedding = OpenAIEmbeddings()
vectordb = Chroma(persist_directory=persist_directory, embedding_function=embe
```

```
question = "What are major topics for this class?"
docs = vectordb.similarity_search(question,k=3)
len(docs)
```

```
3
```

```
from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)
llm.predict("Hello world!")
```

```
'Hello there! How can I assist you today?'
```

```
# 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!'
```

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

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

New chain type -> conversation retrival chain, adds memory, takes history and the new question and condenses into stand-alone question to pass to the vector store to look up for relevant documents.

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

```python
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.'
```

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

```python
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.'
```

Follow up question on previous answer.

LangSmith

Home | Debug | Testing | Monitoring | Datasets

Debug > Project: default

Project: default

Runs | Setup

P50 Latency: 3.91s | P99 Latency: 37.00s | Total Tokens: 35,269

eg. eq(run_type, "chain") | Quick Searches

Columns | Root Runs | All Runs

Run Type	Name	Input	Output	Start Time	Tags
CHAIN	ConversationalRe...	{"question":"...	The reason f...	6/25/2023, 3:55:41 ...	
CHAIN	ConversationalRe...	{"question":"I...	Yes, probabili...	6/25/2023, 3:55:39 ...	
CHAIN	ConversationalRe...	{"question":"...	The reason f...	6/25/2023, 3:55:00 ...	
CHAIN	ConversationalRe...	{"question":"I...	Yes, probabili...	6/25/2023, 3:54:41 ...	
CHAIN	RetrievalQA	Is probability ...	{"result":"Yes...	6/25/2023, 3:52:03 ...	
LLM	ChatOpenAI	human: Hello...	{"run":null,"ll...	6/25/2023, 3:51:26 ...	
CHAIN	RetrievalQA	why are thos...	The prerequi...	6/25/2023, 3:31:27 ...	
CHAIN	RetrievalQA	Is probability ...	Yes, probabili...	6/25/2023, 3:31:09 ...	
CHAIN	RetrievalQA	Is probability ...	The class as...	6/25/2023, 3:25:56 ...	
CHAIN	RetrievalQA	Is probability ...	There is no c...	6/25/2023, 3:23:49 ...	

Rows per page: 10 1–10 of 43

LangSmith

Home | Debug | Testing | Monitoring | Datasets | Documentation

CHAIN a few seconds ago

Chain Input

Jump to bottom | Copy

```
question: why are those prerequesites
needed?
chat_history:
  - content: Is probability a class topic?
    example: false
    additional_kwargs: {}
  - content: Yes, probability is a topic
that will be assumed to be familiar to
      students in this class. The
instructor mentions that basic probability
and
      statistics are prerequisites for the
class and assumes that most
      undergraduate statistics classes,
like Stat 116 taught at Stanford, will
      be more than enough.
    example: false
    additional_kwargs: {}
```

Chain Output

Jump to bottom | Copy

```
answer: The reason for requiring basic
probability and statistics as
  prerequisites for the class is that the
class assumes familiarity with these
  topics. The instructor 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 variables are,
what expectation is, what a variance
  or a random variable is.
```

Run Metadata

Copy

Run Stats

ABC 1,038 → 99 1,137 tokens

Start: 06/25/2023, 10:55:41 PM

End: 06/25/2023, 10:55:45 PM

Response time: 4.27s

Child Runs

- CHAIN ConversationalRetrievalChain 4.27s ABC 1,
  - CHAIN LLMChain 0.69s ABC 112 → 16
    - LLM ChatOpenAI 0.69s ABC 112 → 16
  - CHAIN StuffDocumentsChain 3.48s ABC 926
    - CHAIN LLMChain 3.48s ABC 926 → 83
      - LLM ChatOpenAI 3.48s ABC 926 →

1. Call to llm

Run: ChatOpenAI

★ Rate Run | + Add to Dataset | Share

LLM a minute ago

Chat Model Inputs

HUMAN Copy

```
Given the following conversation and a
follow up question, rephrase the follow up
question to be a standalone question, in
its original language.

Chat History:

Human: Is probability a class topic?
Assistant: Yes, probability is a topic
that will be assumed to be familiar to
students in this class. The instructor
mentions that basic probability and
statistics are prerequisites for the class
and assumes that most undergraduate
statistics classes, like Stat 116 taught
at Stanford, will be more than enough.
Follow Up Input: why are those
prerequesites needed?
Standalone question:
```

Chat Model Outputs

ASSISTANT Copy

```
What is the reason for requiring basic
probability and statistics as
prerequisites for the class?
```

Run Stats

ABC 112 → 16 128 tokens

Start: 06/25/2023, 10:55:41 PM
End: 06/25/2023, 10:55:42 PM
Response time: 0.69s

Run Metadata

completion_tokens: 16
prompt_tokens: 112
total_tokens: 128

INVOCATION_PARAMS
- _type: "openai-chat"
- max_tokens: null
- model: "gpt-3.5-turbo"
- model_name: "gpt-3.5-turbo"
- n: 1
- request_timeout: null
- stop: null
- stream: false
- temperature: 0

OPTIONS
- stop: null

RUNTIME
- library: "langchain"
- library_version: "0.0.213"
- platform: "macOS-13.4-arm64-arm-64bit"

LangSmith

Documentation | Home | Debug | Testing | Monitoring | Datasets | Documentation

Debug > Project: default > Run: ConversationalRet...

Run: ConversationalRetrievalChain

★ Rate Run | + Add to Dataset | Share

CHAIN 2 minutes ago

Chain Input

Jump to bottom | Copy

```
question: why are those prerequesites
needed?
chat_history:
  - content: Is probability a class topic?
    example: false
    additional_kwargs: {}
  - content: Yes, probability is a topic
that will be assumed to be familiar to
      students in this class. The
instructor mentions that basic probability
and
      statistics are prerequisites for the
class and assumes that most
      undergraduate statistics classes,
like Stat 116 taught at Stanford, will
      be more than enough.
    example: false
    additional_kwargs: {}
```

Chain Output

Jump to bottom | Copy

```
answer: The reason for requiring basic
probability and statistics as
  prerequisites for the class is that the
class assumes familiarity with these
  topics. The instructor 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
```

Run Stats

ABC 1,038 → 99 1,137 tokens

Start: 06/25/2023, 10:55:41 PM
End: 06/25/2023, 10:55:45 PM
Response time: 4.27s

Child Runs

- CHAIN ConversationalRetrievalChain 4.27s ABC 1,
  - CHAIN LLMChain 0.69s ABC 112 → 16
    - LLM ChatOpenAI 0.69s ABC 112 → 16
  - StuffDocumentsChain 3.48s ABC 926
    - CHAIN LLMChain 3.48s ABC 926 → 83
      - LLM ChatOpenAI 3.48s ABC 926 →

Debug > Project: default > Run: ConversationalRet...

Run: ConversationalRetrievalChain

★ Rate Run | + Add to Dataset | Share

CHAIN 2 minutes ago

Chain Input

Jump to bottom | Copy

```
question: why are those prerequesites
needed?
chat_history:
  - content: Is probability a class topic?
    example: false
    additional_kwargs: {}
  - content: Yes, probability is a topic
that will be assumed to be familiar to
      students in this class. The
instructor mentions that basic probability
and
      statistics are prerequisites for the
class and assumes that most
      undergraduate statistics classes,
like Stat 116 taught at Stanford, will
      be more than enough.
    example: false
    additional_kwargs: {}
```

Chain Output

Jump to bottom | Copy

```
answer: The reason for requiring basic
probability and statistics as
  prerequisites for the class is that the
class assumes familiarity with these
  topics. The instructor 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 variables are,
```

Run Stats

ABC 1,038 → 99 1,137 tokens

Start: 06/25/2023, 10:55:41 PM

End: 06/25/2023, 10:55:45 PM

Response time: 4.27s

Child Runs

- CHAIN ConversationalRetrievalChain 4.27s ABC 1,
  - CHAIN LLMChain 0.69s ABC 112 → 16
    - LLM ChatOpenAI 0.69s ABC 112 → 16
  - CHAIN StuffDocumentsChain 3.48s ABC 926
    - CHAIN LLMChain 3.48s ABC 926 → 83
      - LLM ChatOpenAI 3.48s ABC 926 →

LangSmith

Home | Debug | Testing | Monitoring | Datasets | Documentation

```
that you don't know, don't try to make up
an answer.
----------
of this class will not be very program
ming intensive, although we will do some
programming, mostly in either MATLAB or
Octa ve. I'll say a bit more about that
later.
I also assume familiarity with basic proba
bility and statistics. So most
undergraduate
statistics class, like Stat 116 taught
here at Stanford, will be more than
enough. I'm gonna
assume all of you know what ra ndom
variables are, that all of you know what
expectation
is, what a variance or a random variable
is. And in case of some of you, it's been
a while
since you've seen some of this material.
At some of the discussion sections, we'll
actually
go over some of the prerequisites, sort of
as  a refresher course under prerequisite
class.
I'll say a bit more about that later as
well.
Lastly, I also assume familiarity with
basi c linear algebra. And again, most
undergraduate
linear algebra courses are more than
enough. So if you've taken courses like
Math 51,
103, Math 113 or CS205 at Stanford, that
would be more than enough. Basically, I'm
gonna assume that all of you know what
matrix es and vectors are, that you know
how to
multiply matrices and vectors and multiply
matrix and matrices, that you know what a
matrix inverse is. If you know what an
eigenvect or of a matrix is, that'd be
```

Run Metadata

completion_tokens: 83

prompt_tokens: 926

total_tokens: 1009

INVOCATION_PARAMS

- _type: "openai-chat"
- max_tokens: null
- model: "gpt-3.5-turbo"
- model_name: "gpt-3.5-turbo"
- n: 1
- request_timeout: null
- stop: null
- stream: false
- temperature: 0

OPTIONS

- stop: null

RUNTIME

- library: "langchain"
- library_version: "0.0.213"
- platform: "macOS-13.4-arm64-arm-64bit"
- runtime: "python"
- runtime_version: "3.10.9"
- sdk_version: "0.0.17"

LLM ChatOpenAI

Copy

```
id:
  - langchain
  - chat_models
  - openai
  - ChatOpenAI
lc: 1
```

Create a chatbot that works on your documents

```python
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import DocArrayInMemorySearch
from langchain.document_loaders import TextLoader
from langchain.chains import RetrievalQA,  ConversationalRetrievalChain
from langchain.memory import ConversationBufferMemory
from langchain.chat_models import ChatOpenAI
from langchain.document_loaders import TextLoader
from langchain.document_loaders import PyPDFLoader
```

```python
# This will initialize your database and reciever chain
def load_db(file, chain_type, k):
    # load documents
    loader = PyPDFLoader(file)
    documents = loader.load()
    # split documents
    text_splitter = CharacterTextSplitter(chunk_size=650, chunk_overlap=0, sep
    docs = text_splitter.split_documents(documents)
    # define embedding
    embeddings = OpenAIEmbeddings()
    # create vector database from data
    db = DocArrayInMemorySearch.from_documents(docs, embeddings)
    # define retriever
    retriever = db.as_retriever(search_type="similarity", search_kwargs={"k":
    # create a chatbot chain. Memory is managed separately
    qa = ConversationalRetrievalChain.from_llm(
        llm=ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0),
        chain_type=chain_type,
        retriever=retriever,
        return_source_documents=True,
        return_generated_question=True,
    )
    return qa
```

```python
def call_load_db(count):
    global qa
    if count == 0 or file_input.value is None:  # init or no file
        file = "docs/cs229_lectures/MachineLearning-Lecture01.pdf"
        button_load.button_style="outline"
        qa = load_db(file, "stuff", 4)
        button_load.button_style="solid"
    else:
        file_input.save("temp.pdf")
        file = file_input.filename
        button_load.button_style="outline"
        qa = load_db("temp.pdf", "stuff", 4)
        button_load.button_style="solid"
    clr_history()
    return pn.pane.Markdown(f"Loaded File: {file}")

def convchain(count):
    global qa, chat_history, last_question, last_source
    if count == 0:
        return
    query = inp.value_input
    inp.value = ''
    button_send.button_style="outline"
    result = qa({"question": query, "chat_history": chat_history})
    button_send.button_style="solid"
    chat_history.extend([(query, result["answer"])])
    last_question[0] = result["generated_question"]
    last_source[0] = result["source_documents"]
    print(result['answer'])
    panels.extend([
        pn.Row('User:', pn.pane.Markdown(query, width=600)),
        pn.Row('ChatBot:', pn.pane.Markdown(result['answer'], width=600, style
    ])

    return pn.WidgetBox(*panels,scroll=True)

def get_lquest(count):
    global last_question
    if count == 0:
```

```
)
tab3= pn.Column(
    pn.panel(lhist),
    pn.layout.Divider(),
)
tab4=pn.Column(
    pn.Row( file_input, button_load, bound_button_load),
    pn.Row( button_clearhistory, pn.pane.Markdown("Clears chat history. Can us
    pn.layout.Divider(),
)
dashboard = pn.Column(
    pn.Row(pn.pane.Markdown('# QnA_Bot')),
    pn.Tabs(('Conversation', tab1), ('Database', tab2), ('Chat History', tab3)
)
dashboard
```