LangChain
Chat with your data

Overview

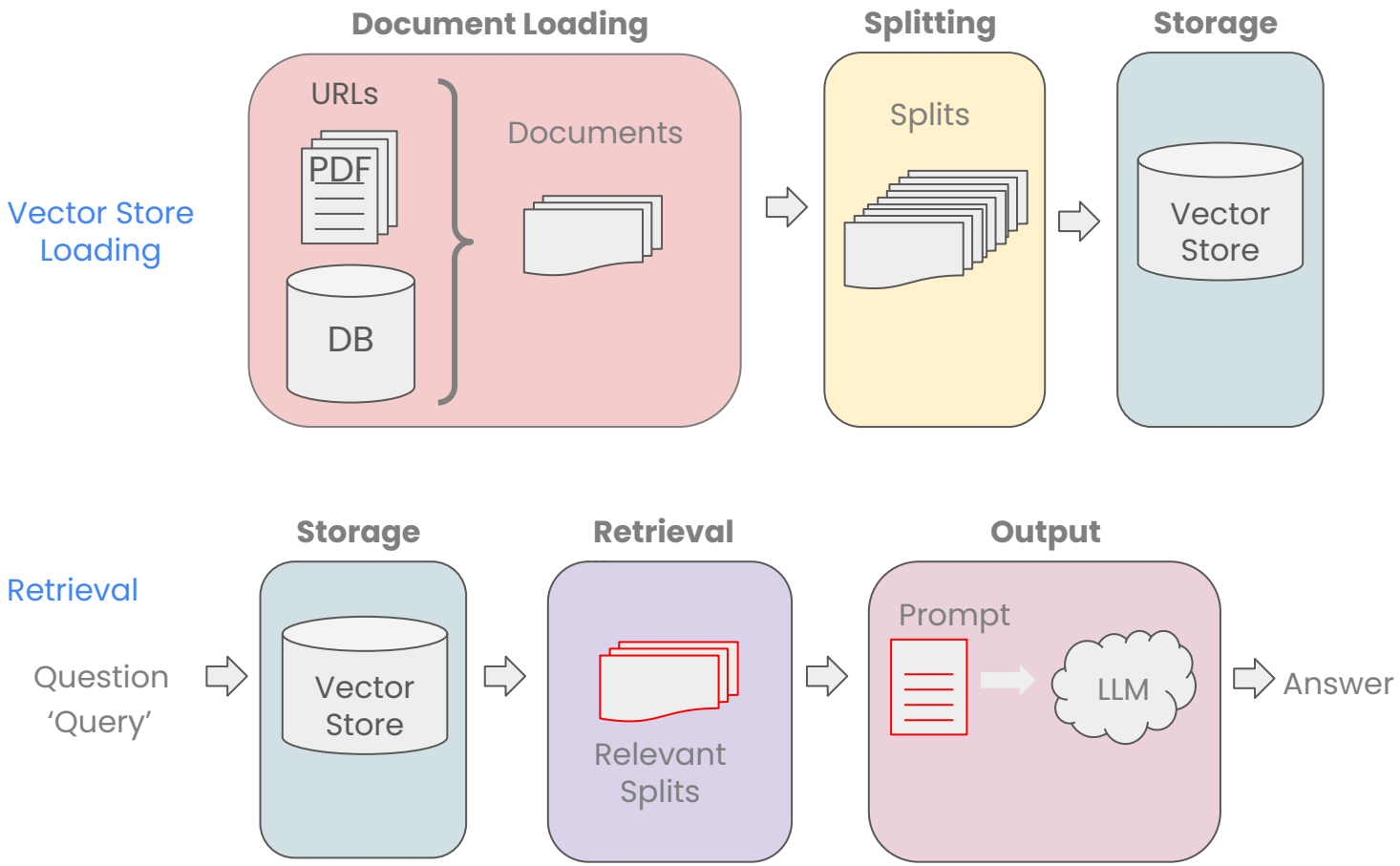
LangChain

DeepLearning.AI

Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) is a very popular paradigm.

- Retrieve relevant documents and load into “working memory” / context window.



LangChain
Chat with your data

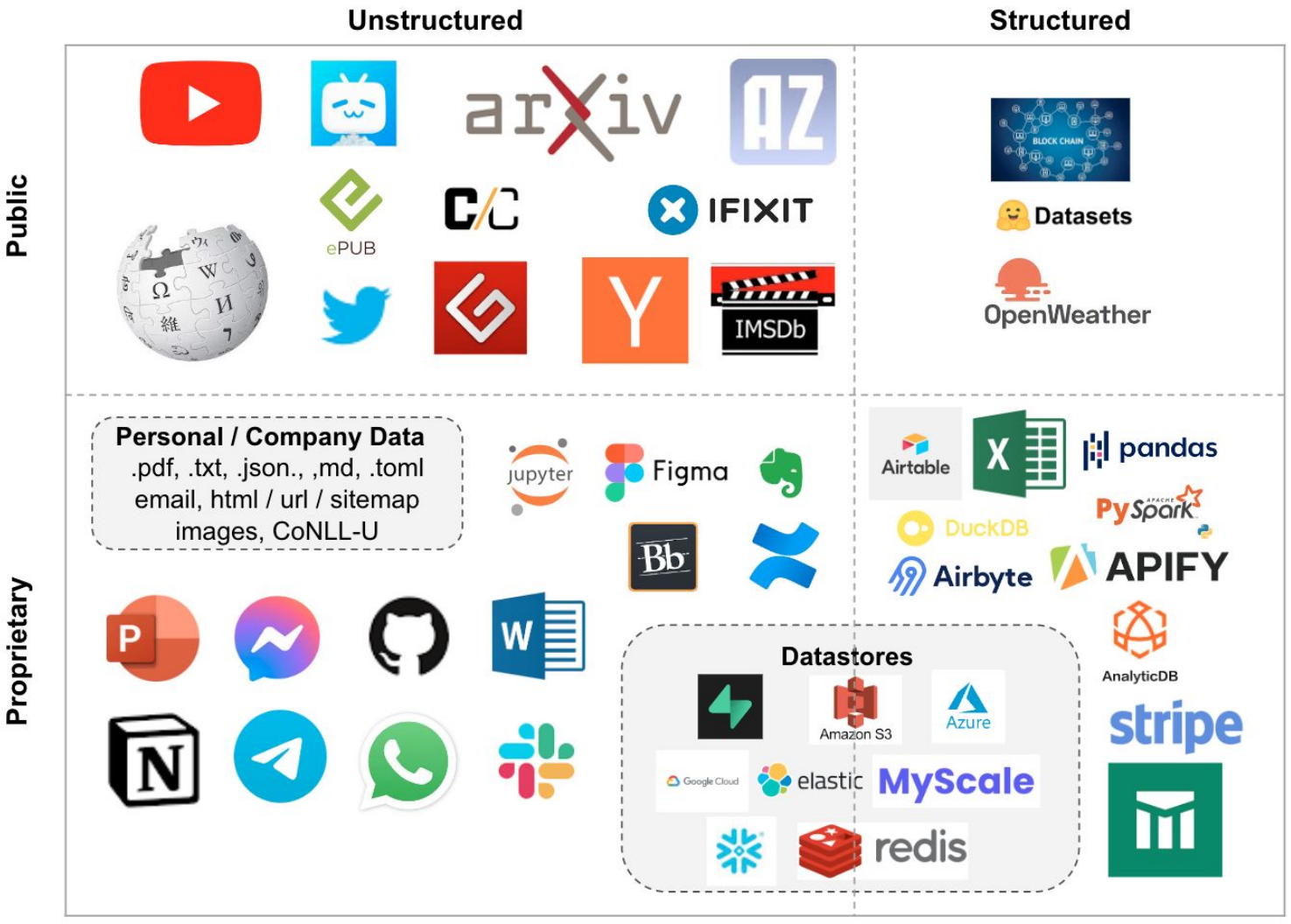
Document Loading

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})
]
```

Document Loaders



LangChain
Chat with your data

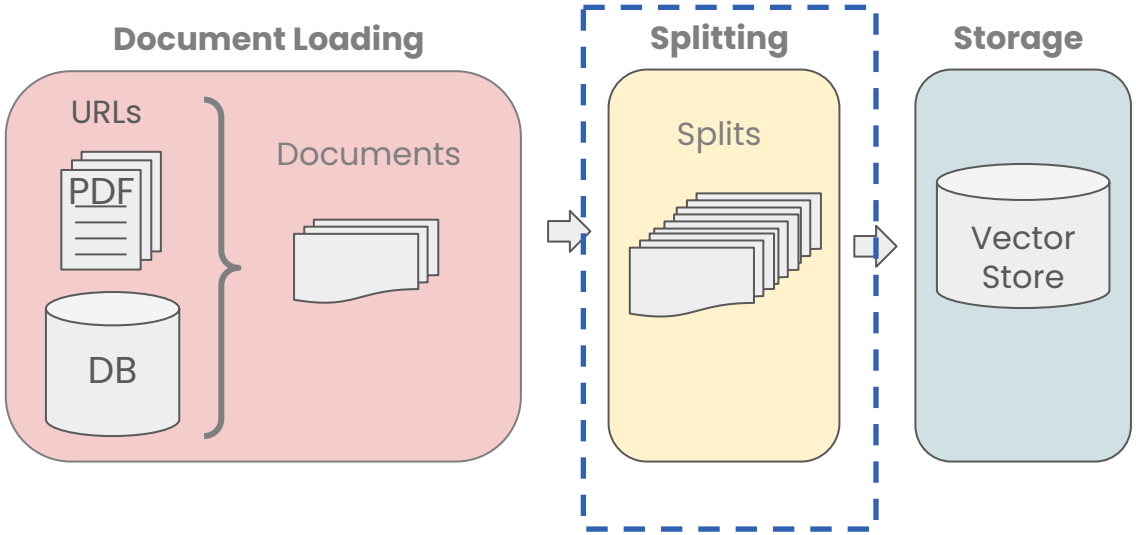
Document Splitting

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?

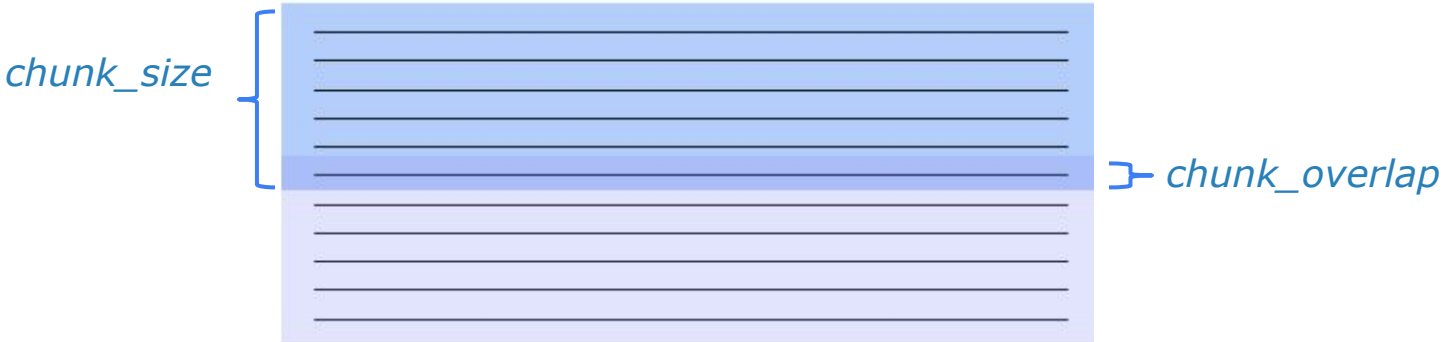
Example Splitter

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

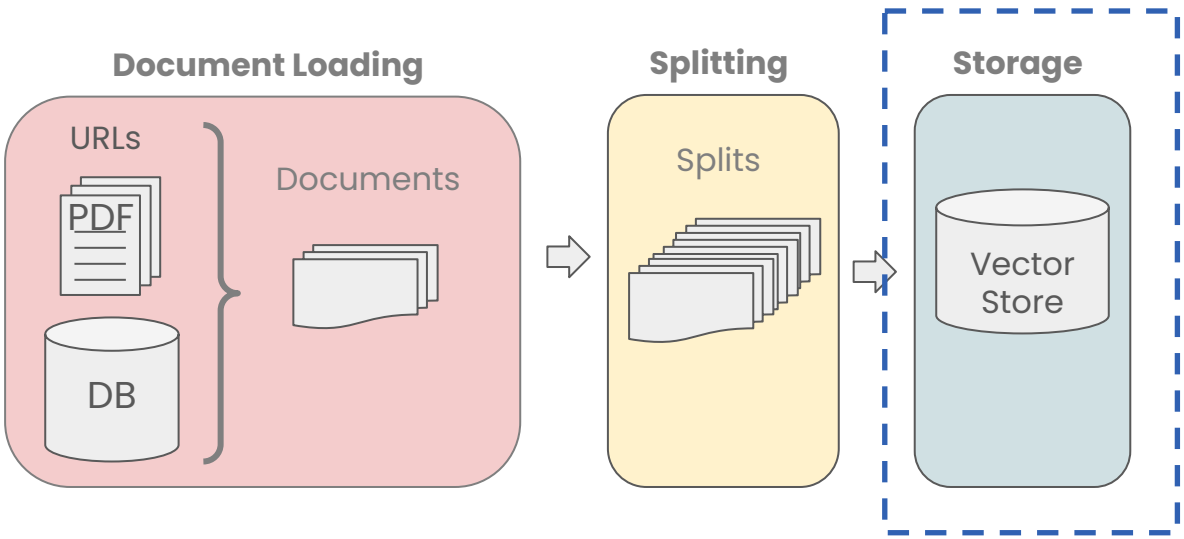
LangChain
Chat with your data

Vector Stores and Embeddings

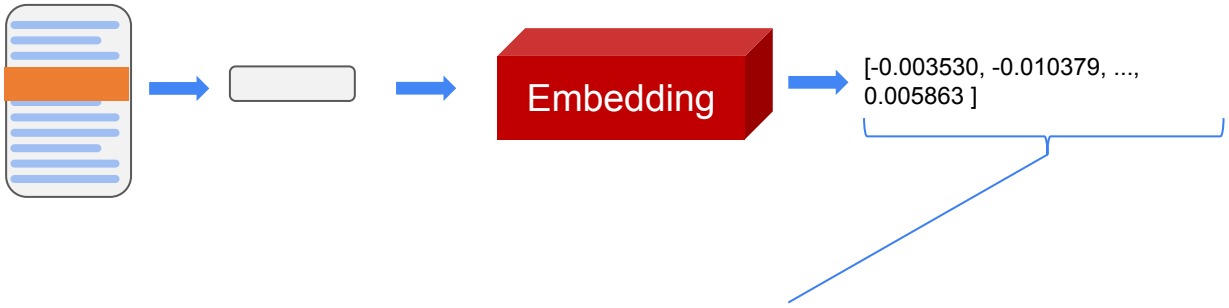
LangChain

DeepLearning.AI

Vector Stores

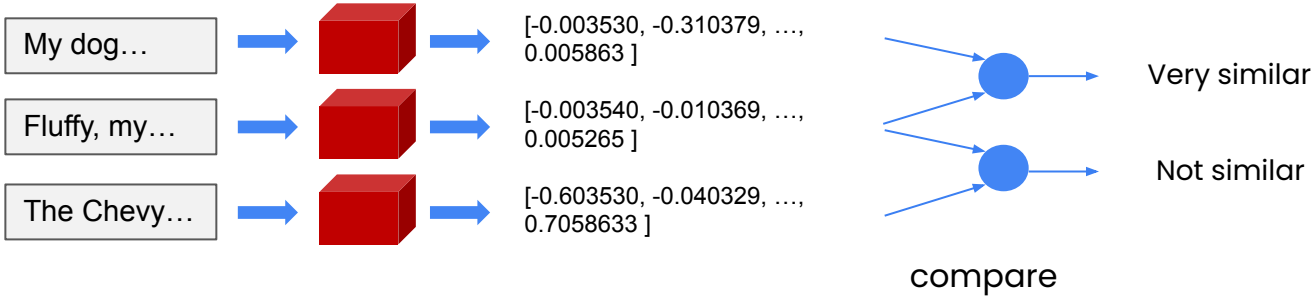


Embeddings

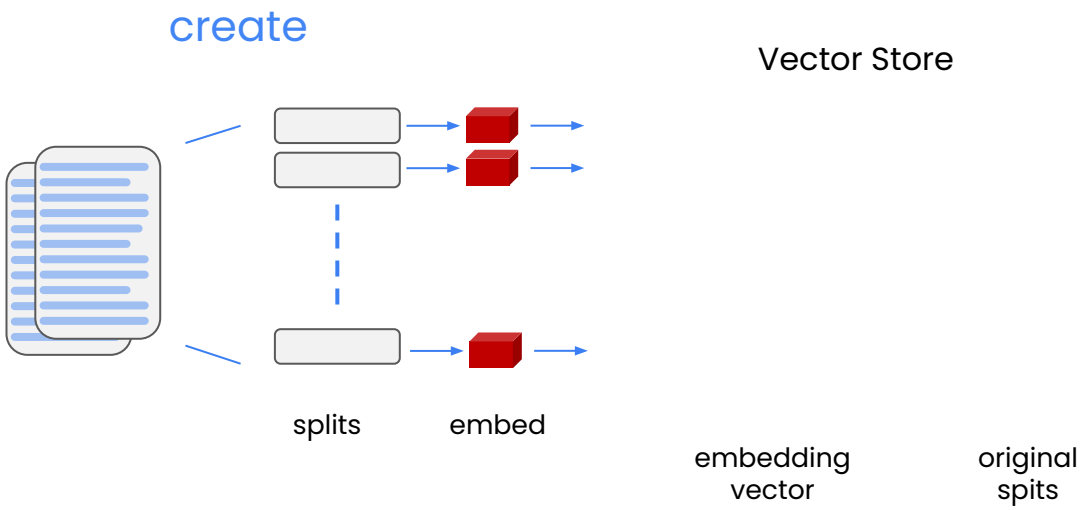


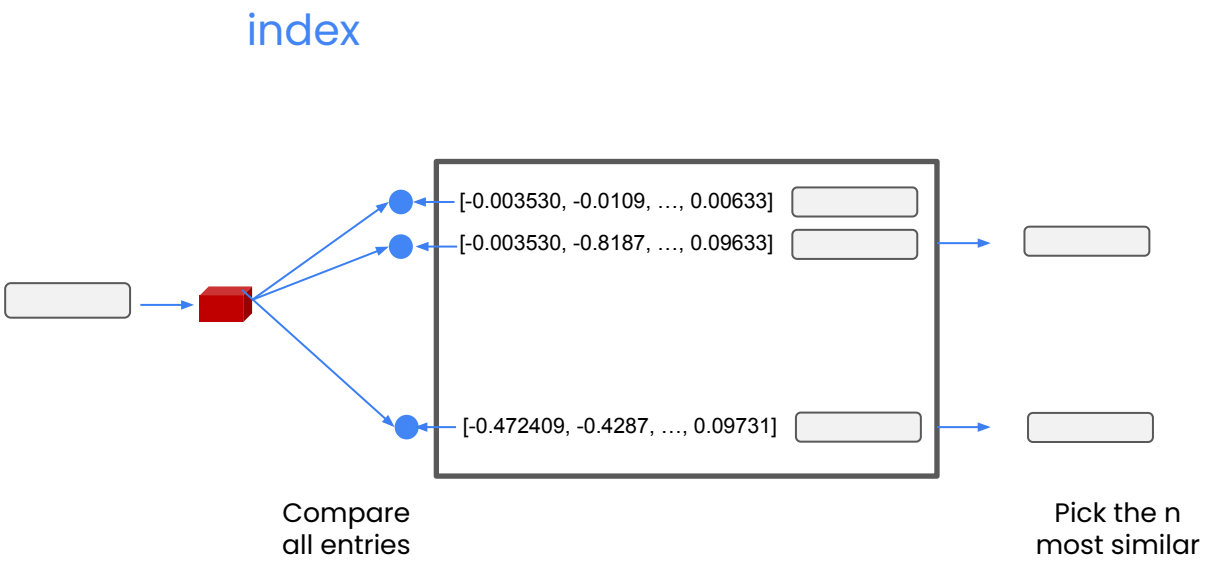
- Embedding vector captures content/meaning
- Text with similar content will have similar vectors

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.



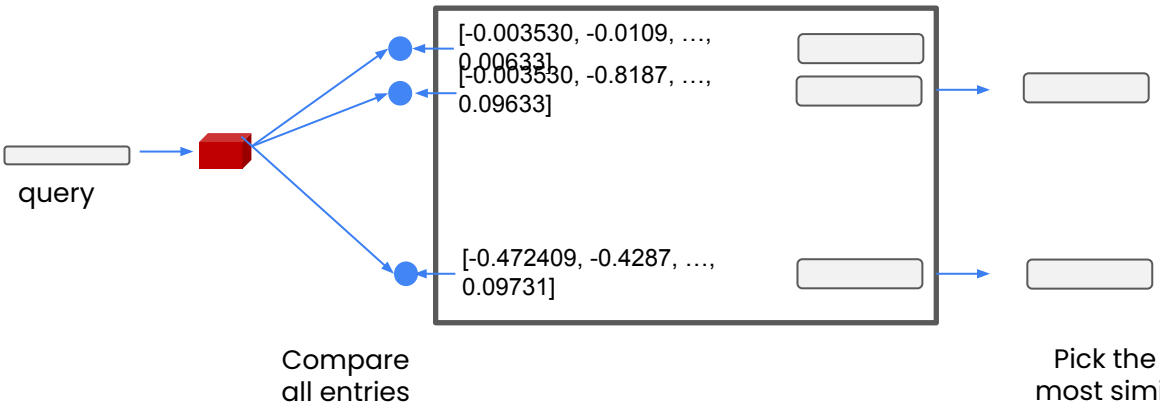
Vector Store





Vector Store/Database

index



Process with llm

The returned values can now fit in the LLM context

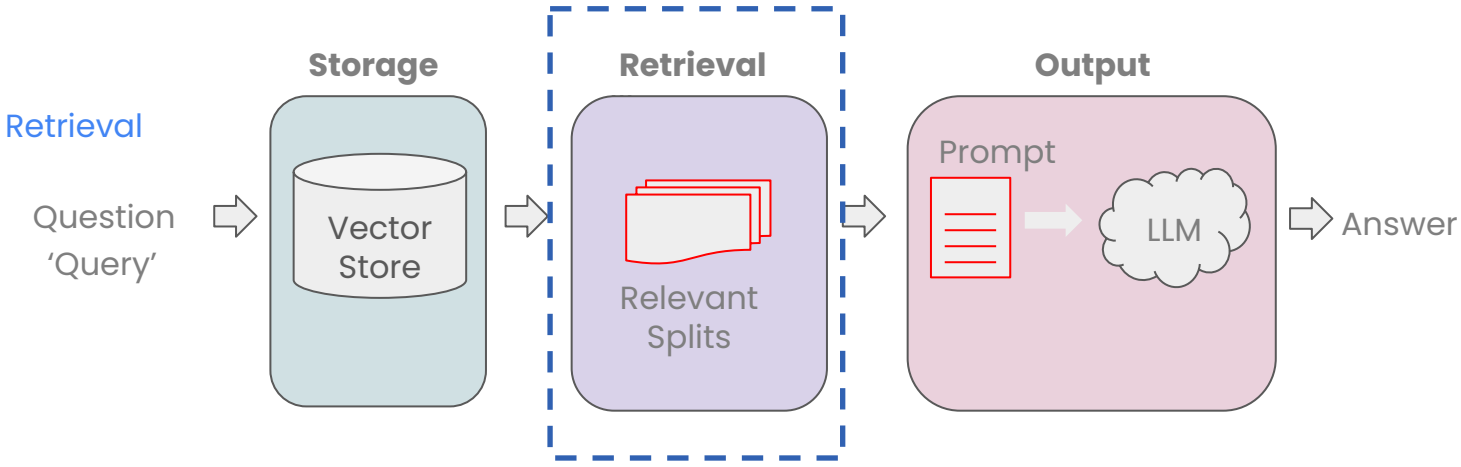
LangChain
Chat with your data

Retrieval

LangChain

DeepLearning.AI

Retrieval



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

The Amanita phalloides has a large and imposing epigeous (aboveground) fruiting body (basidiocarp).

A mushroom with a large fruiting body is the Amanita phalloides. Some varieties are all-white.

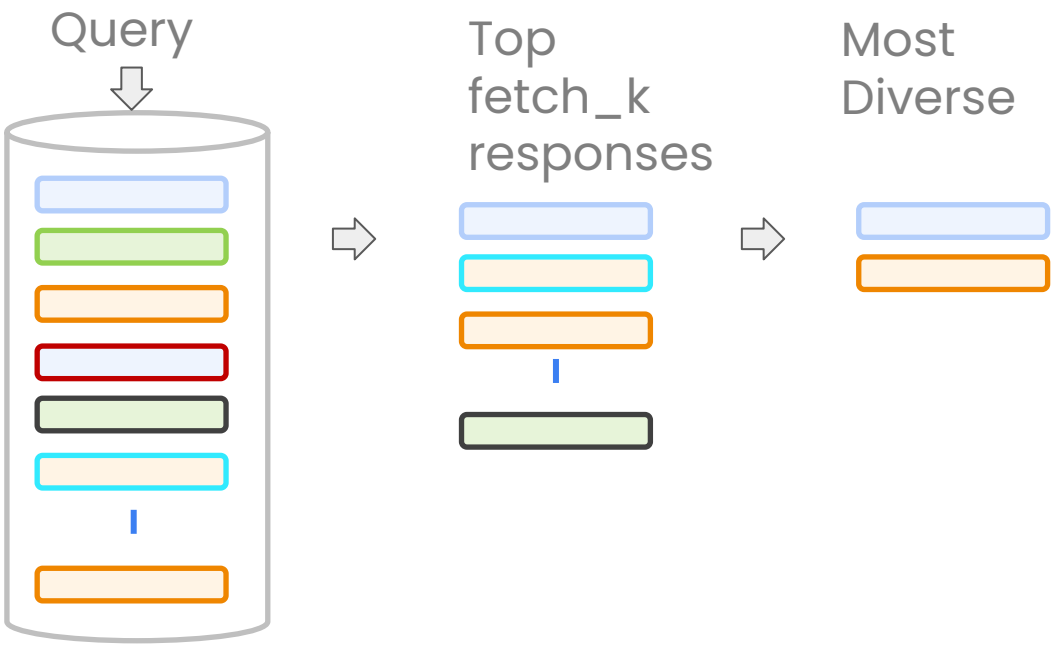
AA. phalloides, a.k.a Death Cap, is one of the most poisonous of all known mushrooms.

Most Similar

MMR

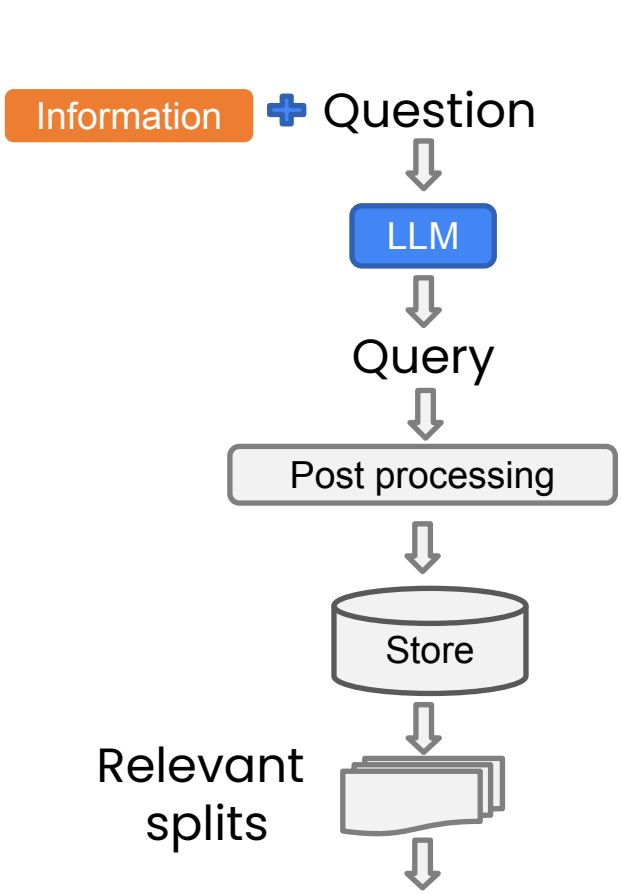
MMR algorithm

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



LLM Aided Retrieval

- There are several situations where the Query applied to the DB is more than just the Question asked.
- One is SelfQuery



Self-query

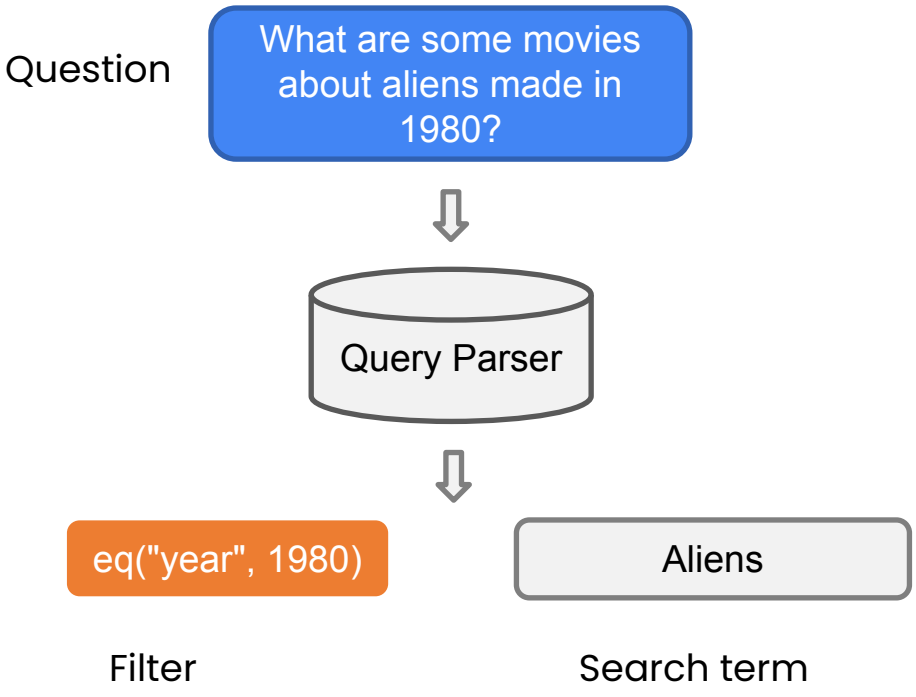
Information: Query format

Query: Question
Filter: eq["section", "testing"]

Query parser

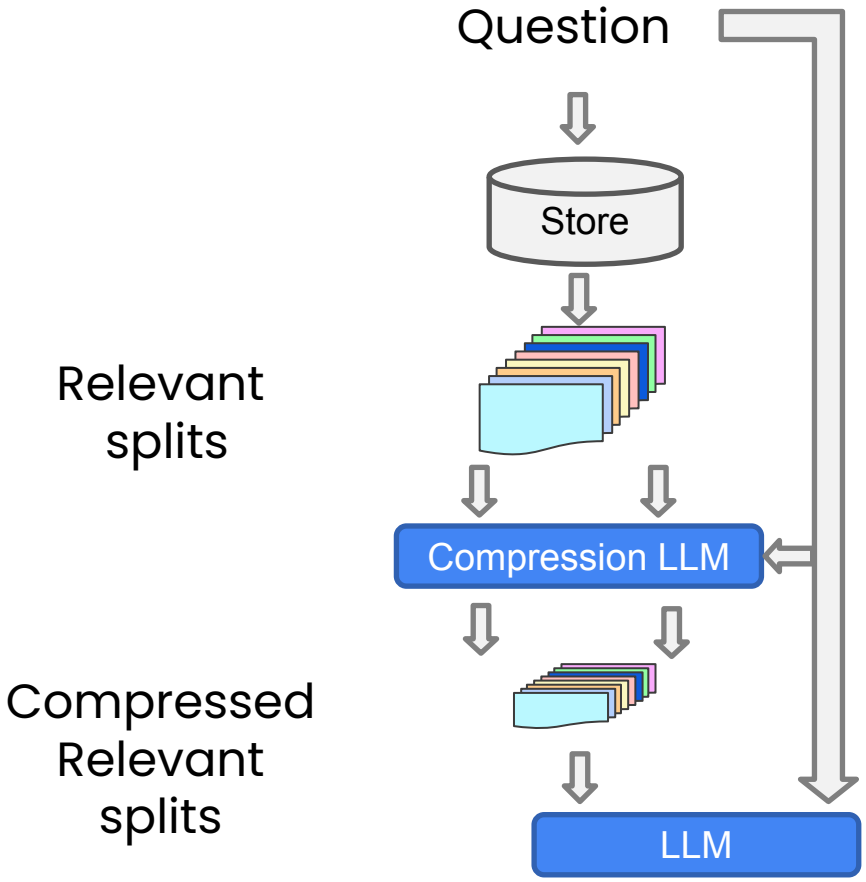
LLM Aided Retrieval

- 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.



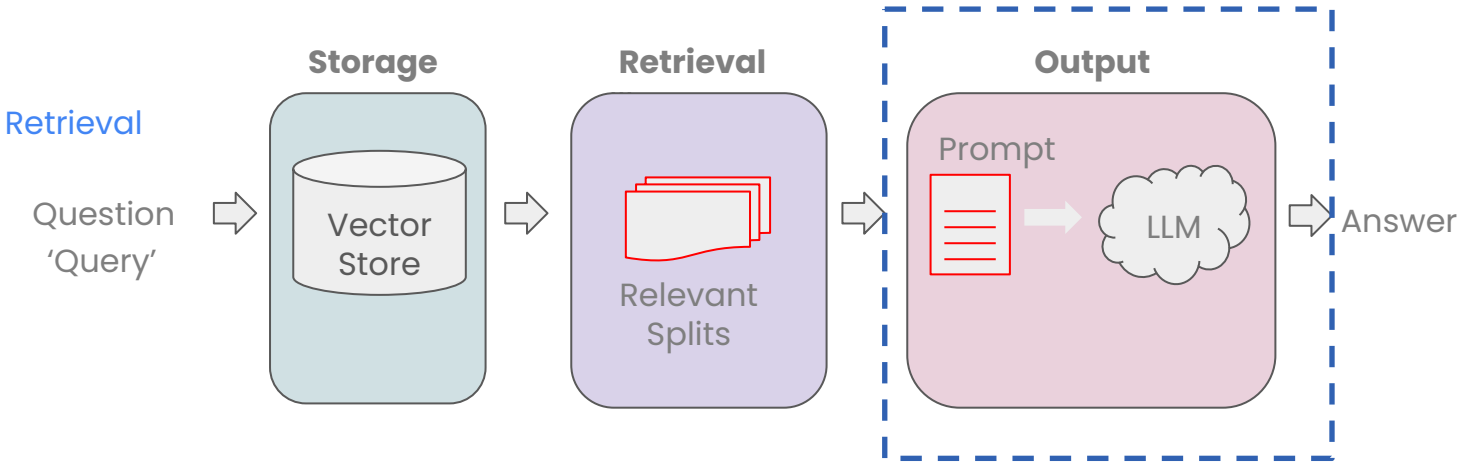
LangChain
Chat with your data

Question Answering

LangChain

DeepLearning.AI

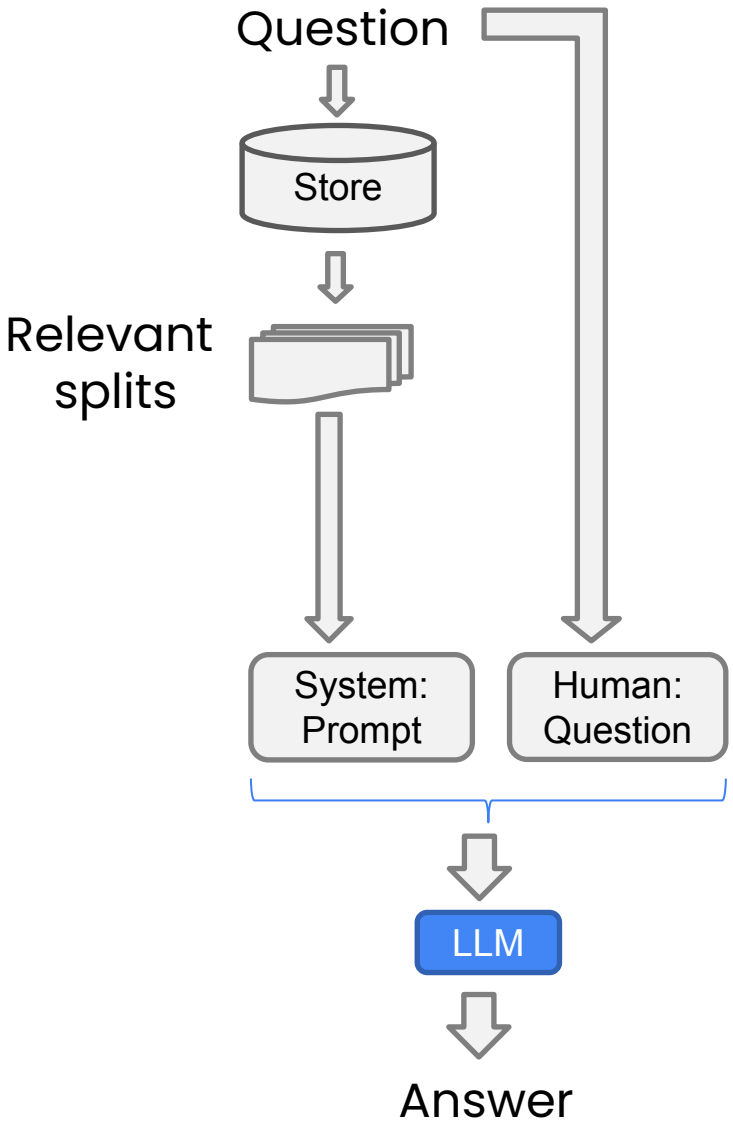
Question Answering



- 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

RetrievalQA chain

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



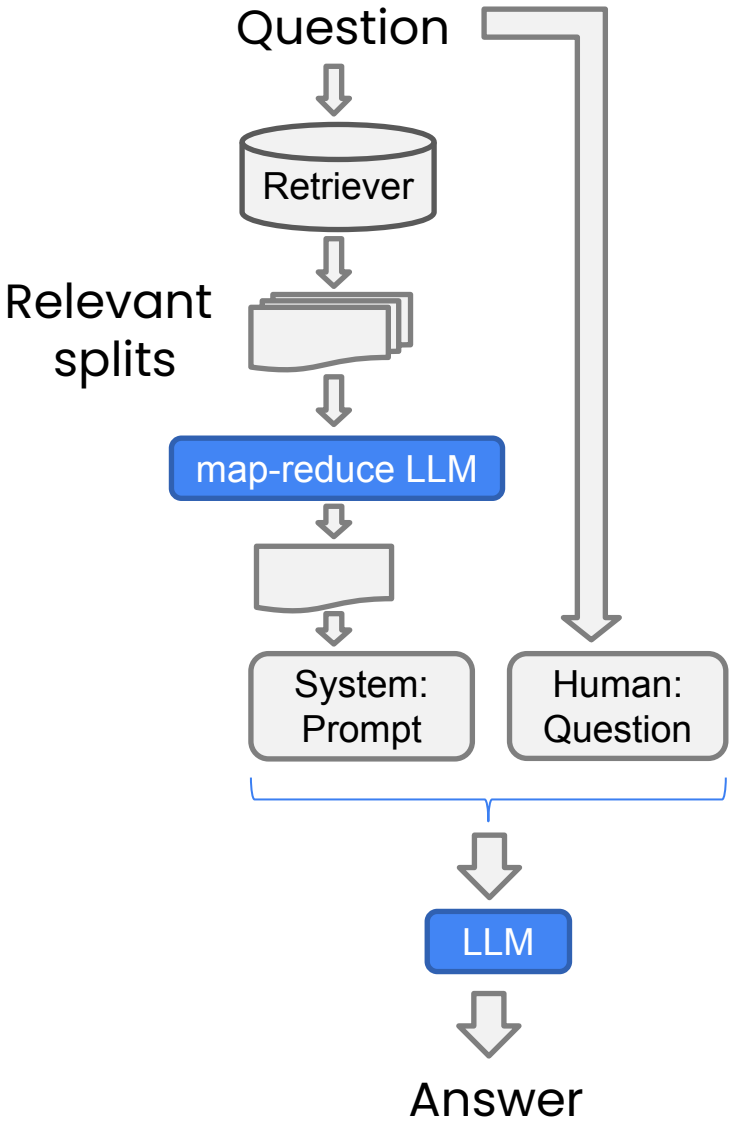
Question is applied to the Vector Store as a query

Vector store provides k relevant documents

Docs and original question are sent to an LLM

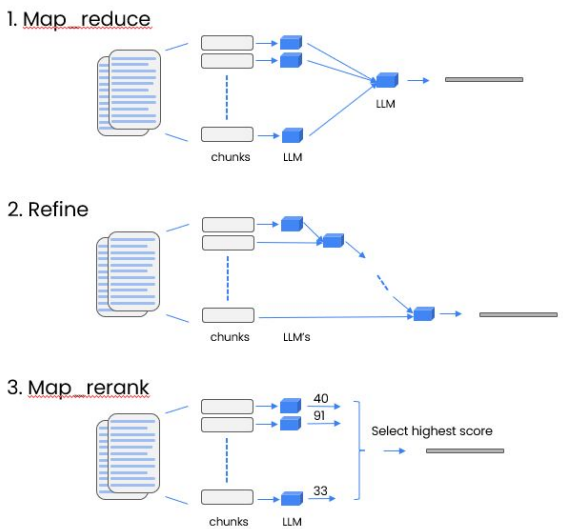
Retrieval Chain with LLM selection

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



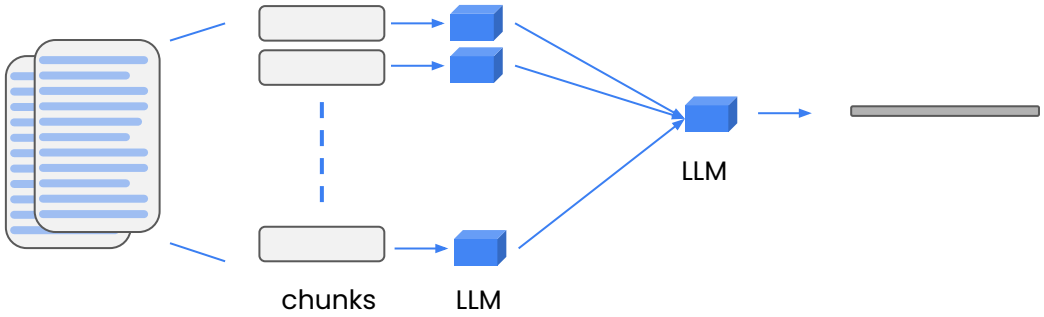
You many have too many docs to fit into an LLM context. The solution is to use an LLM to select the 'most relevant' information

3 additional methods

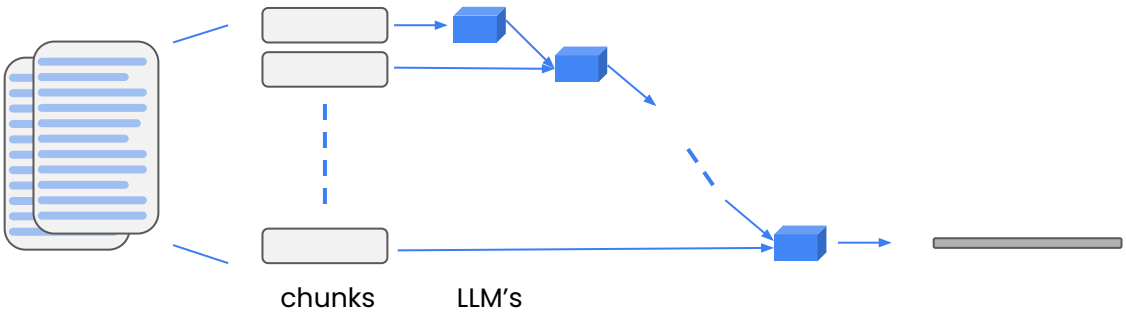


3 additional methods

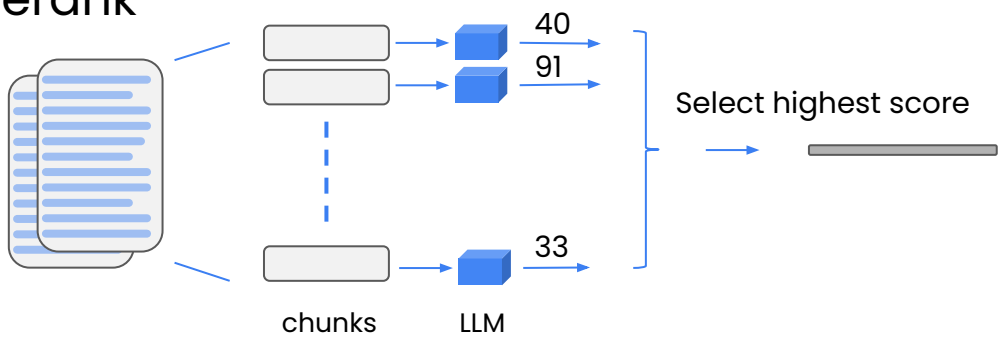
1. Map_reduce



2. Refine



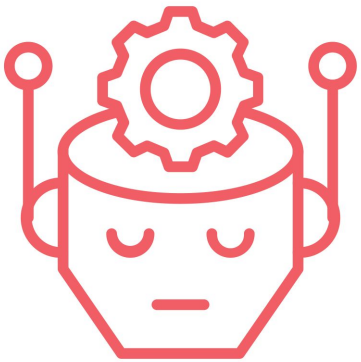
3. Map_rerank



LangChain for LLM Application Development

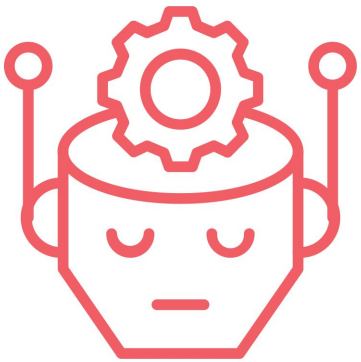
Agents

Agents



Agent refers to the idea of using large language models as reasoning engine to determine which actions to take and in what order.

Agents



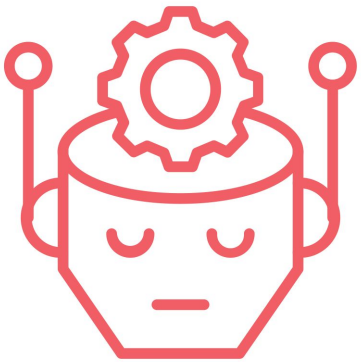
Agents use an LLM to determine which actions to take and in what order.

An action can be using a tool and observing its output and deciding what to return to the user.

```
from langchain.agents import
initialize_agent, AgentType

agent = initialize_agent(tools, llm,
        agent=AgentType.ZERO_SHOT_REACT_DESCRI
PTION,        verbose=True)
```

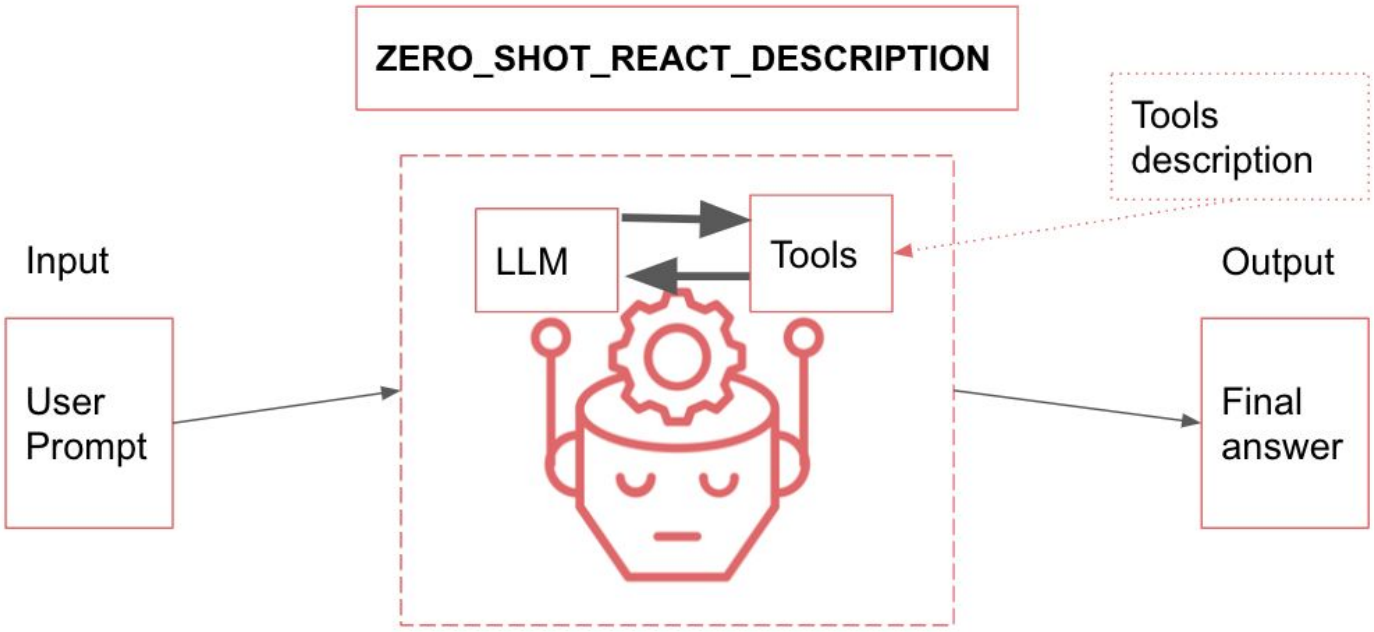
Agents



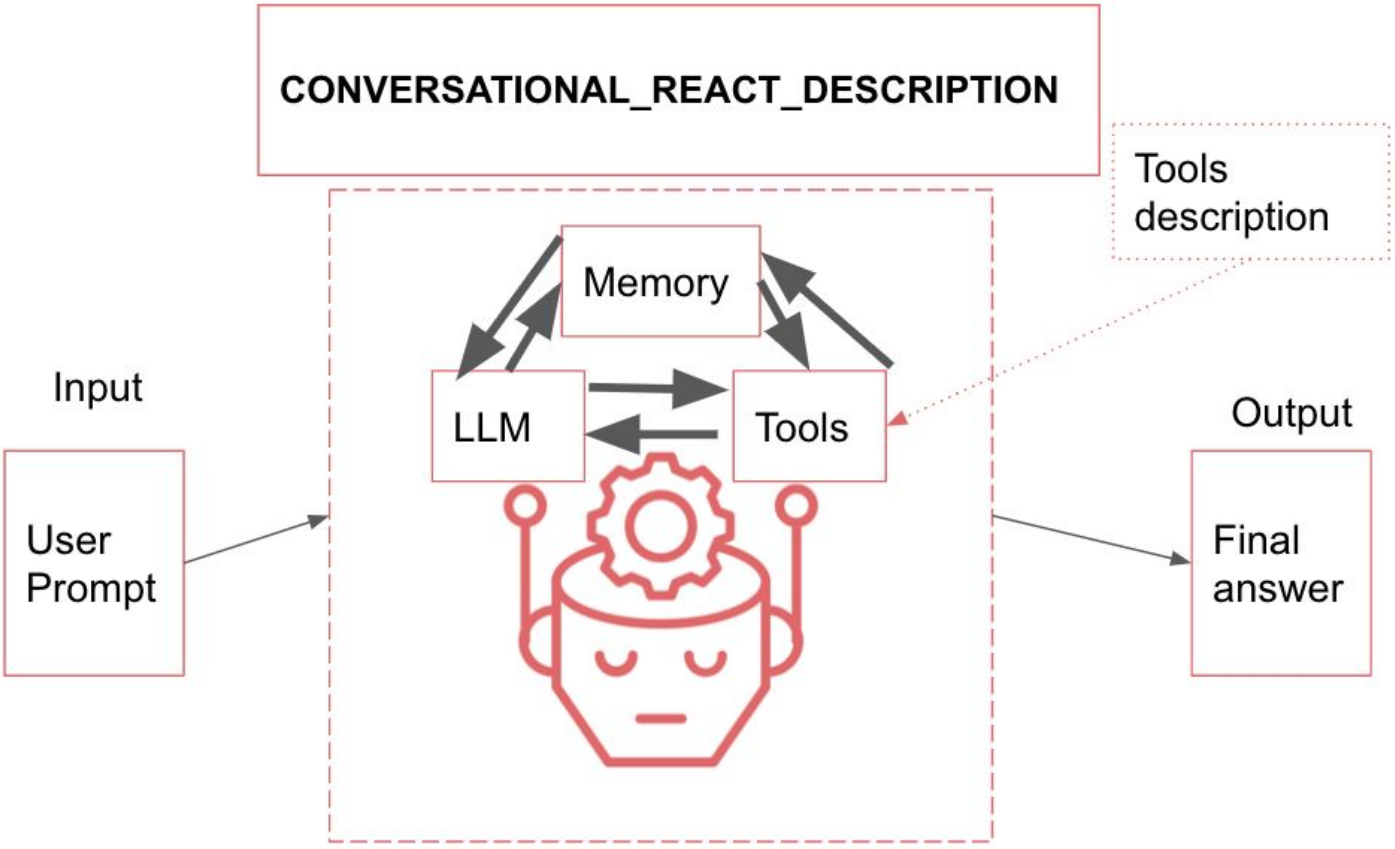
To construct an agent, you need:

- PromptTemplate: this is responsible for taking the user input and previous steps and constructing a prompt to send to the language model
- Language Model: this takes the prompt constructed by the PromptTemplate and returns some output
- Output Parser: this takes the output of the Language Model and parses it into an AgentAction or AgentFinish object.

Agents


ZERO_SHOT_REACT_DESCRIPTION
Tools description
Input
LLM
Tools
Output
User Prompt
Final answer

Agents


CONVERSATIONAL_REACT_DESCRIPTION
Tools description
Memory
Input
LLM
Tools
Output
User Prompt
Final answer