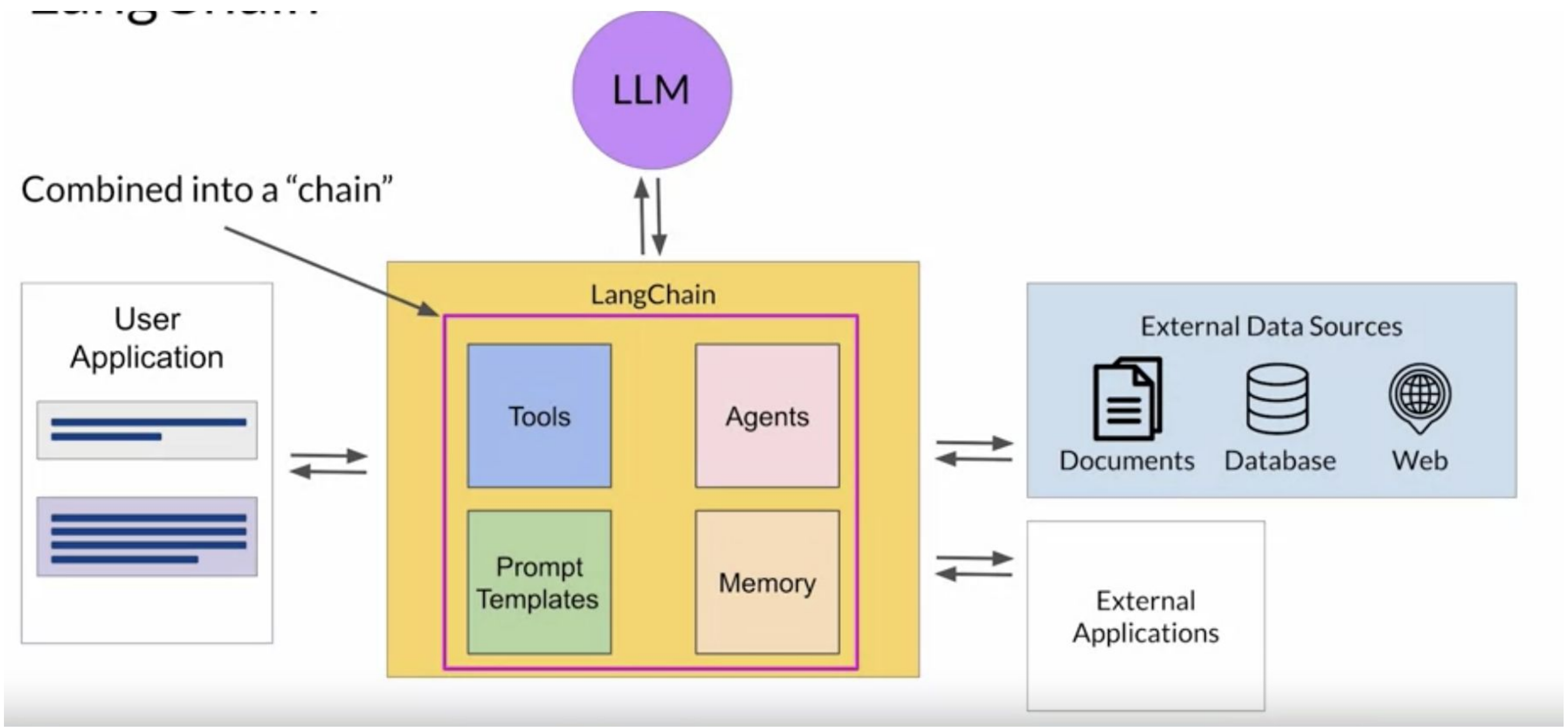
LLM Application Development

Arin Ghazarian
Chapman University

LangChain

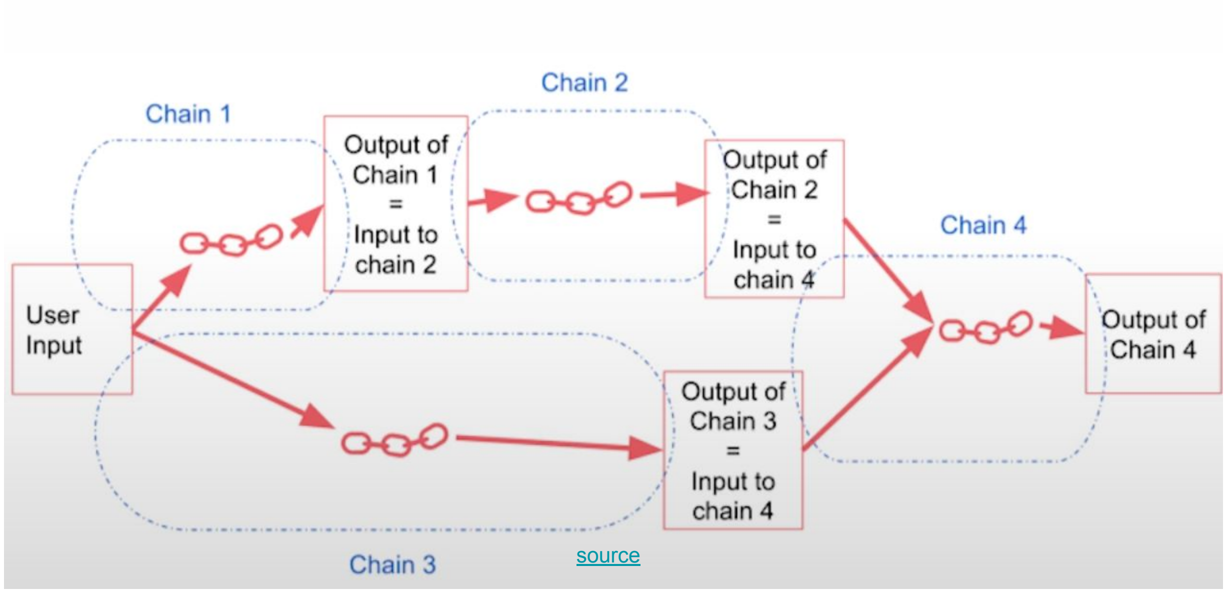
LangChain


LLM
Combined into a "chain"
User Application
LangChain
Tools
Agents
Prompt Templates
Memory
External Data Sources
Documents
Database
Web
External Applications

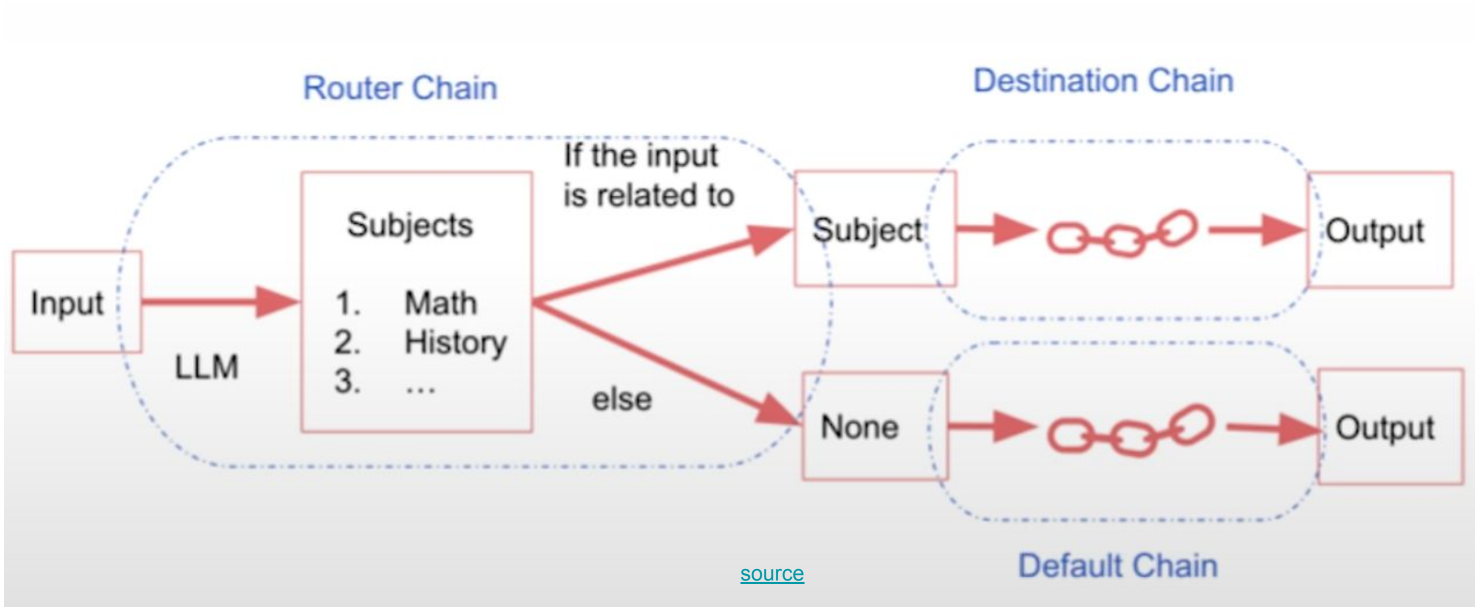
LangChain

- Chatbots, retrieval-augmented generation, ReAct, document summarization, and synthetic data generation.
- Document loaders
  - SQL and NoSQL databases including JSON support
  - Integrations with cloud storage systems including Amazon, Google, and Microsoft Azure API
  - Can read from more than 50 document types and data sources.
- OpenAI, Anthropic, and Hugging Face language models
- Chains
- Agents
- Few-shot learning prompt generation support
- Online data access systems
  - Wrappers for news, movie information, and weather
  - multiple web scraping subsystems and templates
  - Google Drive documents, spreadsheets, and presentations summarization, extraction, and creation
  - Google Search and Microsoft Bing web search
  - iFixit repair guides
  - wikis search and summarization;
- Vector databases:
  - Milvus vector database to store and retrieve vector embeddings
  - Weaviate vector database to cache embedding and data objects
  - Redis cache database storage;
- Code generation and analysis
  - Python and JavaScript code generation, analysis, and debugging
  - Python RequestsWrapper and other methods for API requests
  - finding and summarizing "todo" tasks in code;

Sequential Chain


Chain 1
Chain 2
Chain 3
Chain 4
User Input
Output of Chain 1 = Input to chain 2
Output of Chain 2 = Input to chain 4
Output of Chain 3 = Input to chain 4
Output of Chain 4
source

Router Chain



source

LangChain Expression Language (LCEL)

- LCEL is a declarative way to easily compose chains together:

```
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate

model = ChatOpenAI(model="gpt-4")
prompt = ChatPromptTemplate.from_template("tell me a short joke about {topic}")
output_parser = StrOutputParser()

chain = prompt | model | output_parser

chain.invoke({"topic": "ice cream"})
```

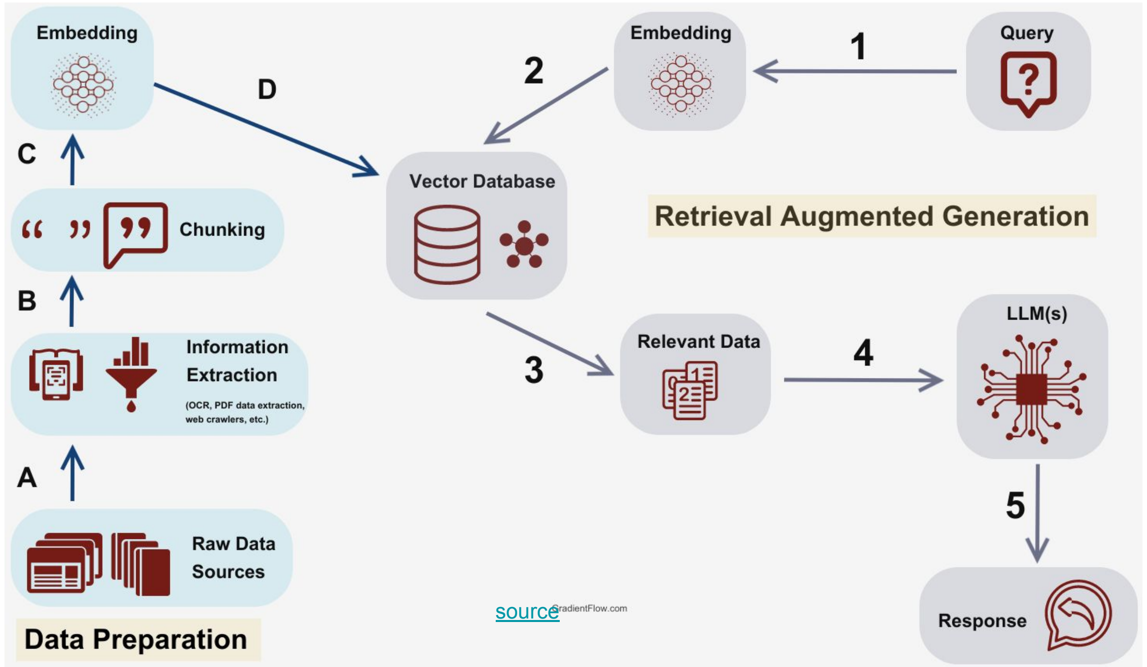
Demo: LangChain Intro

- https://github.com/SamurAIGPT/langchain-course/blob/main/getting-started/Introduction.ipynb
- LangChain Handbook: Intro to LangChain

Retrieval-Augmented Generation (RAG)

RAG

- Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks"
- Retrieve a relevant document and load it into the prompt context window

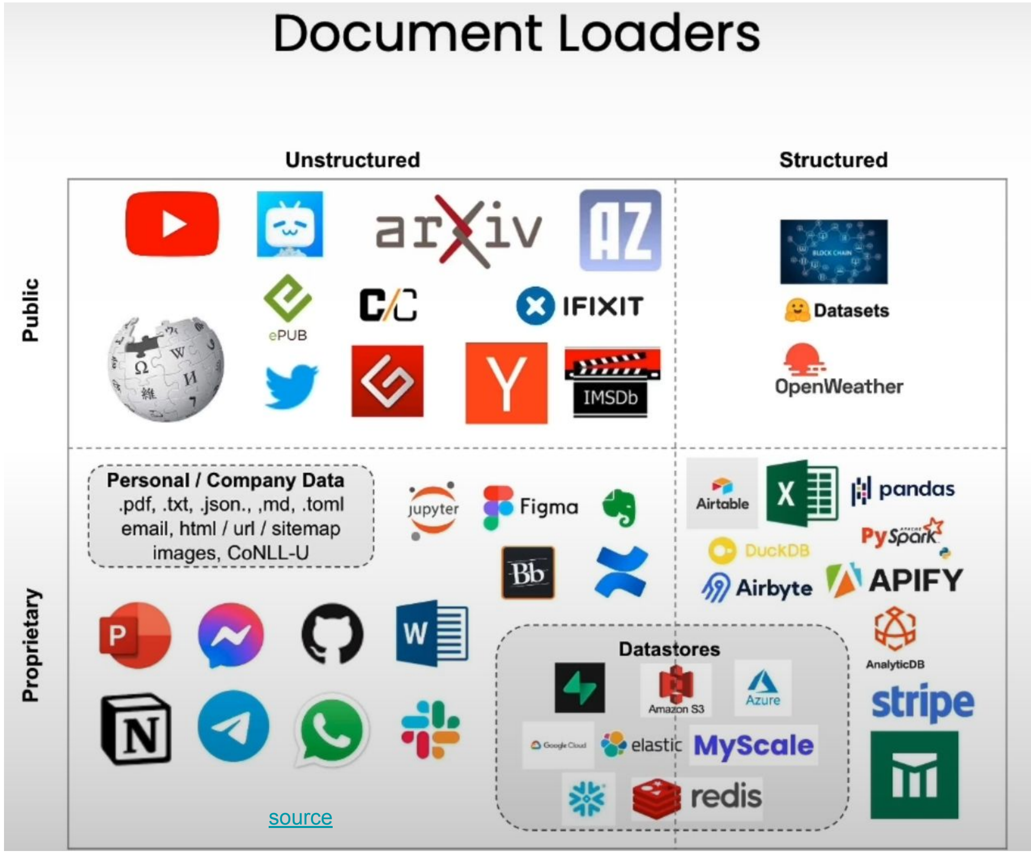


Document Loaders

- Load data from different sources
  - PDFs
  - Websites
  - Databases
  - JSON
  - HTML
  - Youtube
  - etc.

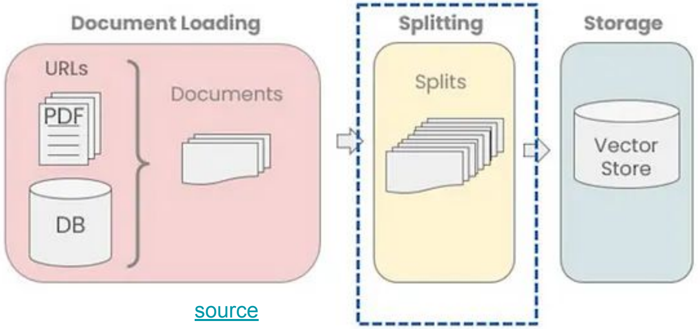
from langchain.document_loaders import PyPDFLoader

loader = PyPDFLoader("docs/my.pdf")

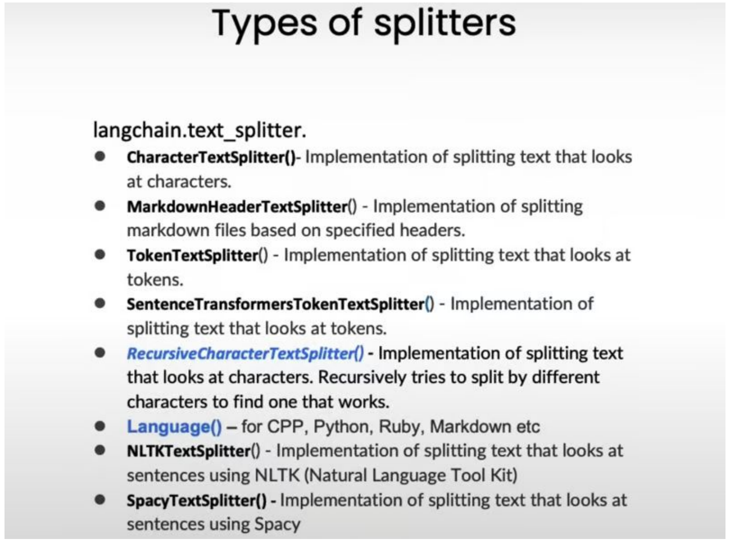


Document Splitting

- Splitting documents into smaller chunks to be processed



source



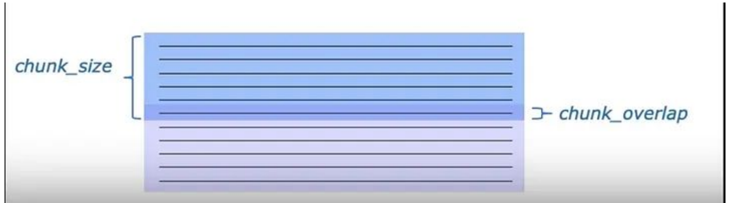
Types of splitters

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

```
from langchain.text_splitter import RecursiveCharacterTextSplitter


splitter = RecursiveCharacterTextSplitter(
  chunk_size=25,
  chunk_overlap=7
)
```

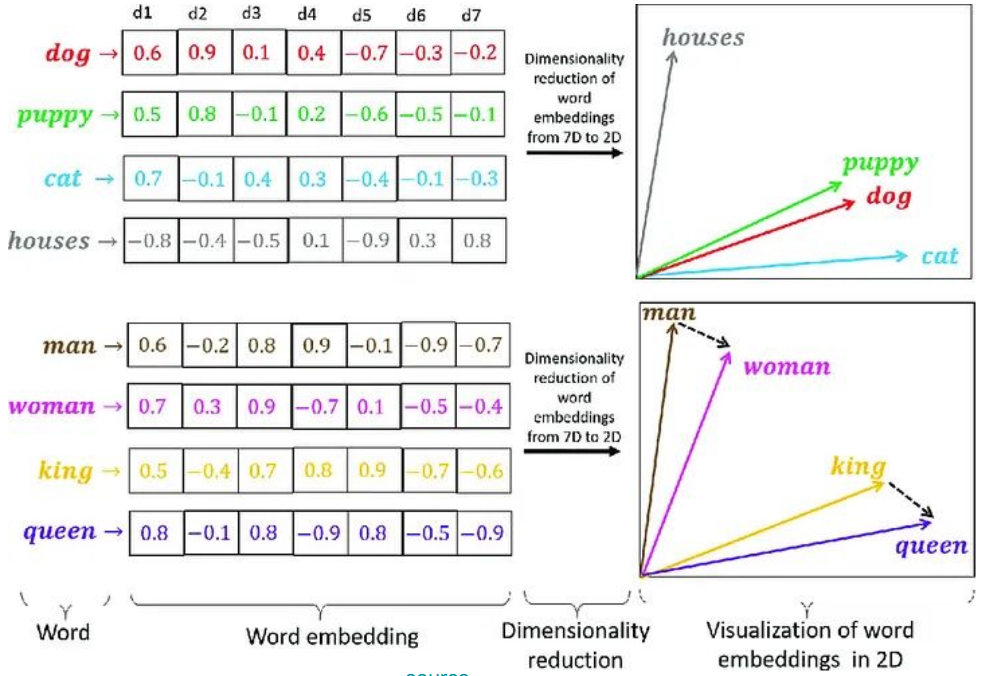


RecursiveCharacterTextSplitter

- Understanding LangChain's RecursiveCharacterTextSplitter

Embedding Vectors

- Text with similar content will be close in the embedding space



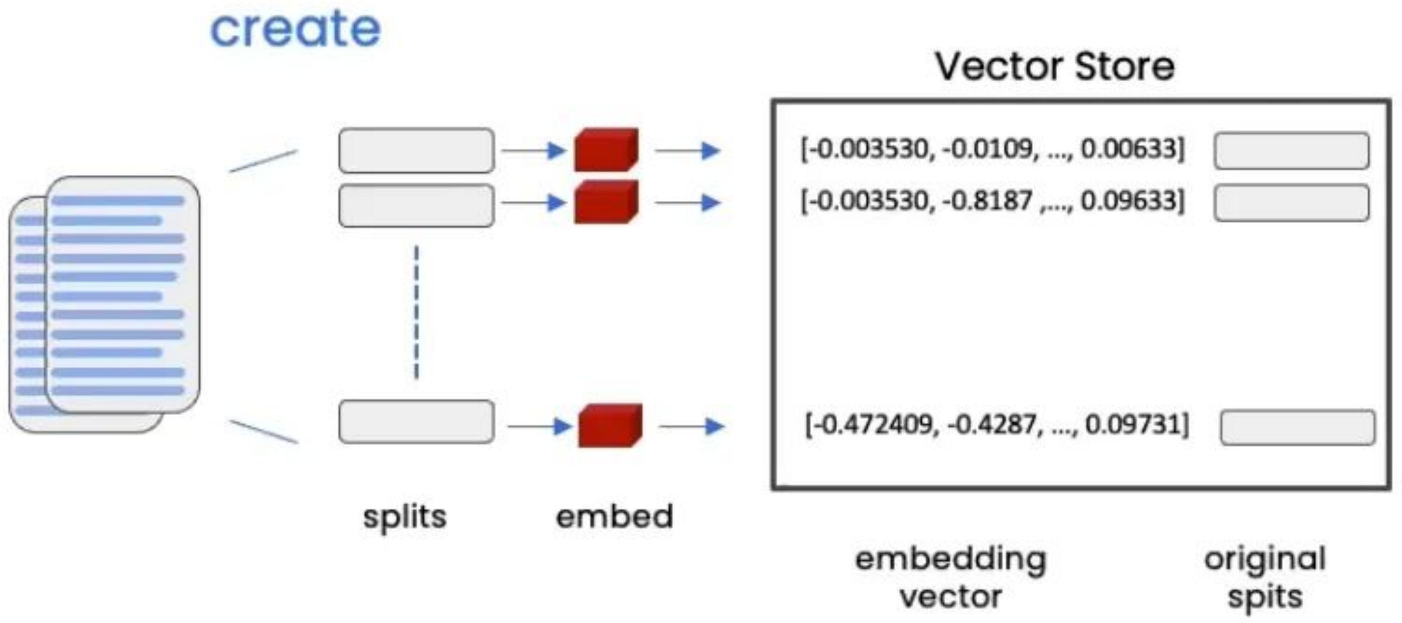
	d1	d2	d3	d4	d5	d6	d7
dog →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
puppy →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
cat →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Dimensionality reduction of word embeddings from 7D to 2D
houses
puppy
dog
cat
Dimensionality reduction of word embeddings from 7D to 2D
man
woman
king
queen
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

source

Vector Databases



Vector Databases: Example

- OpenAI embeddings with Chroma vector database

```
from langchain.vectorstores import Chroma

from langchain.embeddings.openai import OpenAIEmbeddings


embedding = OpenAIEmbeddings()

vectordb = Chroma.from_documents(

  documents=splits,

  embedding=embedding,

  persist_directory='/my_dir'

)

vectordb.persist()
```

```
question = "are there horses that can fly?"

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

docs[0].page_content
```

Maximum Marginal Relevance(MMR)

- First queries the vector store and choose the “fetch_k” most similar responses.
- Then, we select of “fetch_k” documents and optimize to achieve both relevance to the query and diversity among the results.

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

SelfQuery

- Used to to infer metadata from the query itself
- It is usually used to apply a metadata filter

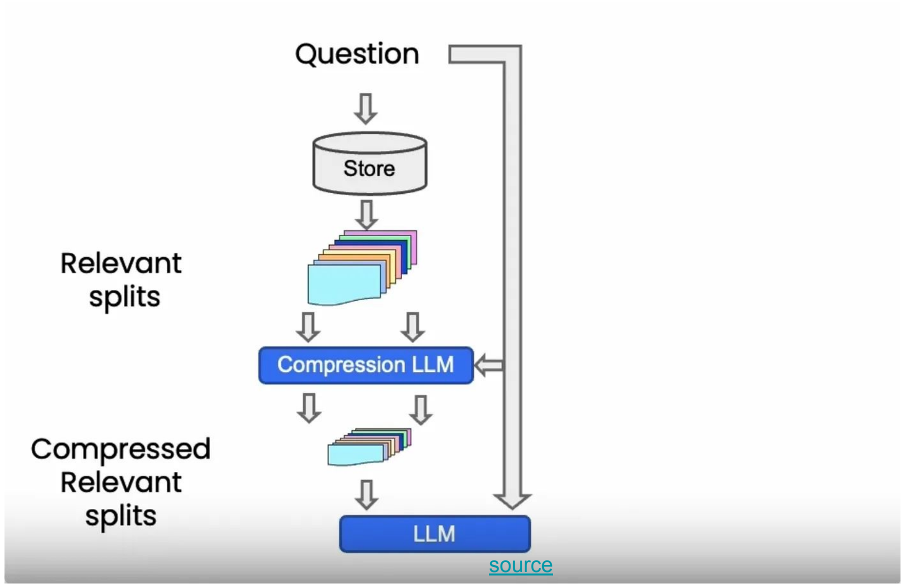
```
from langchain.llms import OpenAI
from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain.chains.query_constructor.base import AttributeInfo
metadata_field_info = [AttributeInfo( name="source",
      description="The lecture the chunk is from, should be one of
`docs/cs229_lectures/MachineLearning-Lecture01.pdf`,
`docs/cs229_lectures/MachineLearning-Lecture02.pdf`, or
`docs/cs229_lectures/MachineLearning-Lecture03.pdf`",
      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
)

question = "what did they say about regression
in the third lecture?"
docs =
retriever.get_relevant_documents(question)
```

Contextual Compression

- Improves the quality of retrieved docs
- Passing the full document might cause expensive LLM calls and poorer response
- Run all the documents through an LLM to extract the most relevant segments



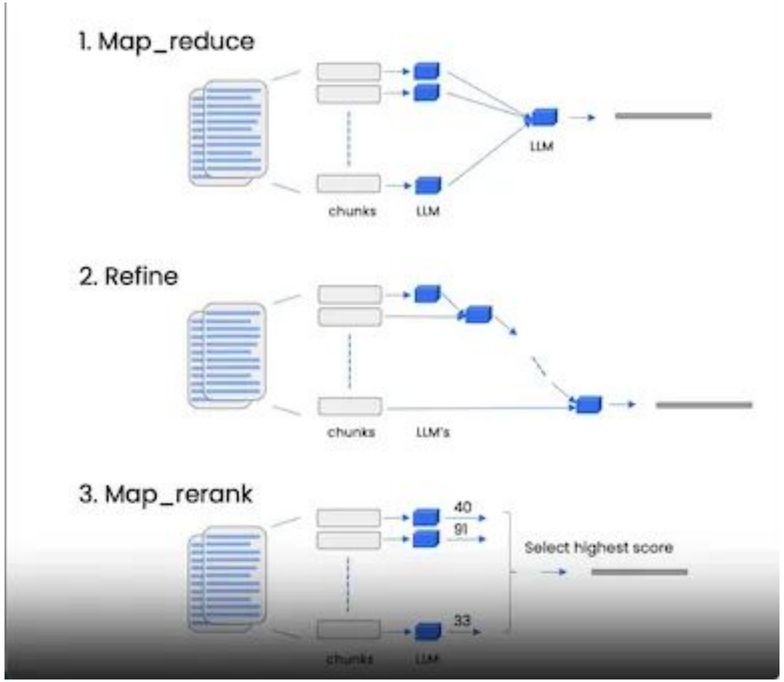
RAG using LangChain

```
from langchain.chains import RetrievalQA


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

Chain Type

- **Stuff:** By default, the RetrievalQA "stuff" method (chain_type="stuff"), which stuffs all the documents into the final prompt. requires one call to the language model. But, if there are too many documents retrieved, they may not fit inside the context window. In such cases, we may use different techniques namely map-reduce, refine and map_rerank.
- **Refine:** The final prompt that we send to the next language model is a sequence that combines the previous response with new data and asks for an improved/refined response with the added context.
- **Map_reduce:** Individual documents are passed through the LMM to get an answer and then these answers are combined into a final answer with a final call to the LLM. Requires more calls to the language model, but can operate over arbitrarily many documents
- **Map_rerank:** It separates texts into batches, feeds each batch to the LLM, scores how fully each one answers the question, and generates the final answer based on the highest-scored responses from each batch.



RetrievalQA Chain with MapReduce

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

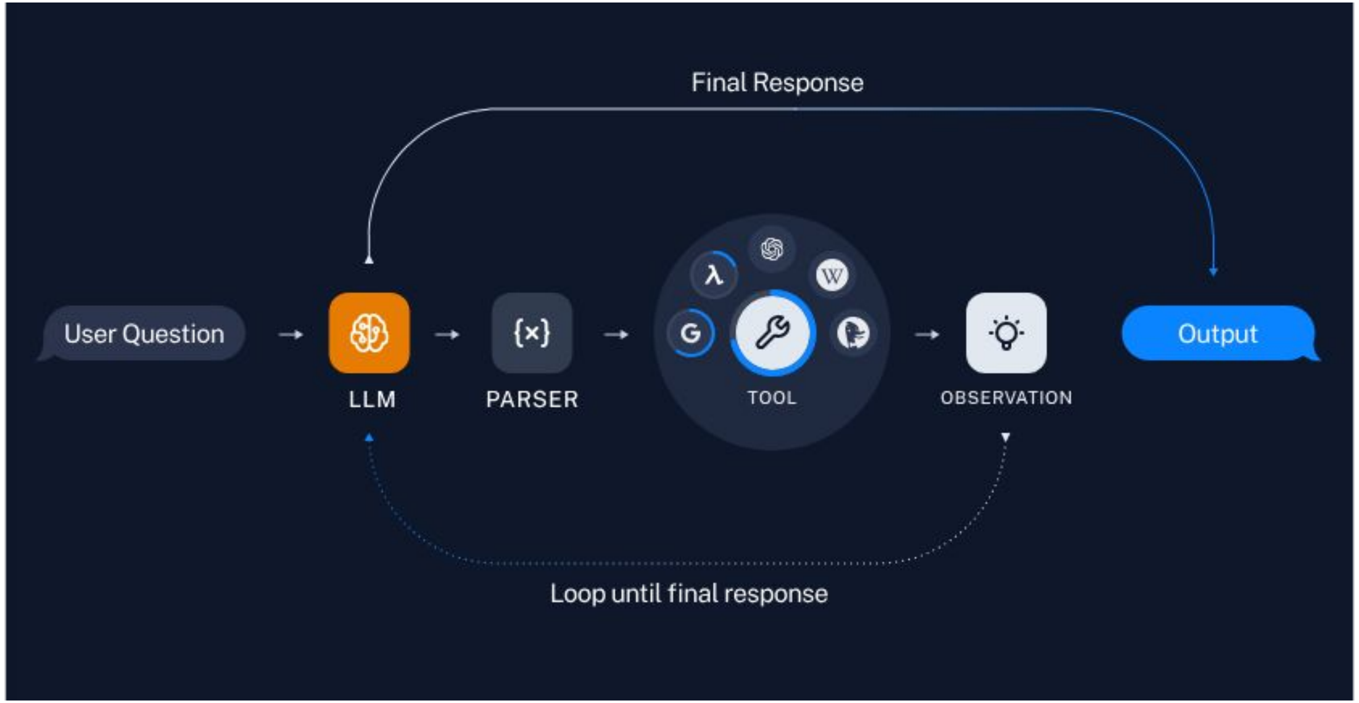
What is
RAG?
planet
moons
LLM
LLM
moons?
Jupiter
Saturn
user

Demo: RAG Using LangChain

- LangChain Handbook: Retrieval Augmentation (LangChain and Pinecone)

Agents

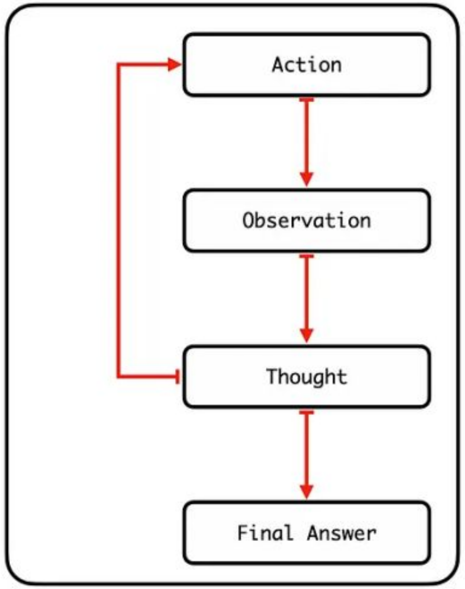
- Agents: leverage an LLM to decide what actions to take and in what order

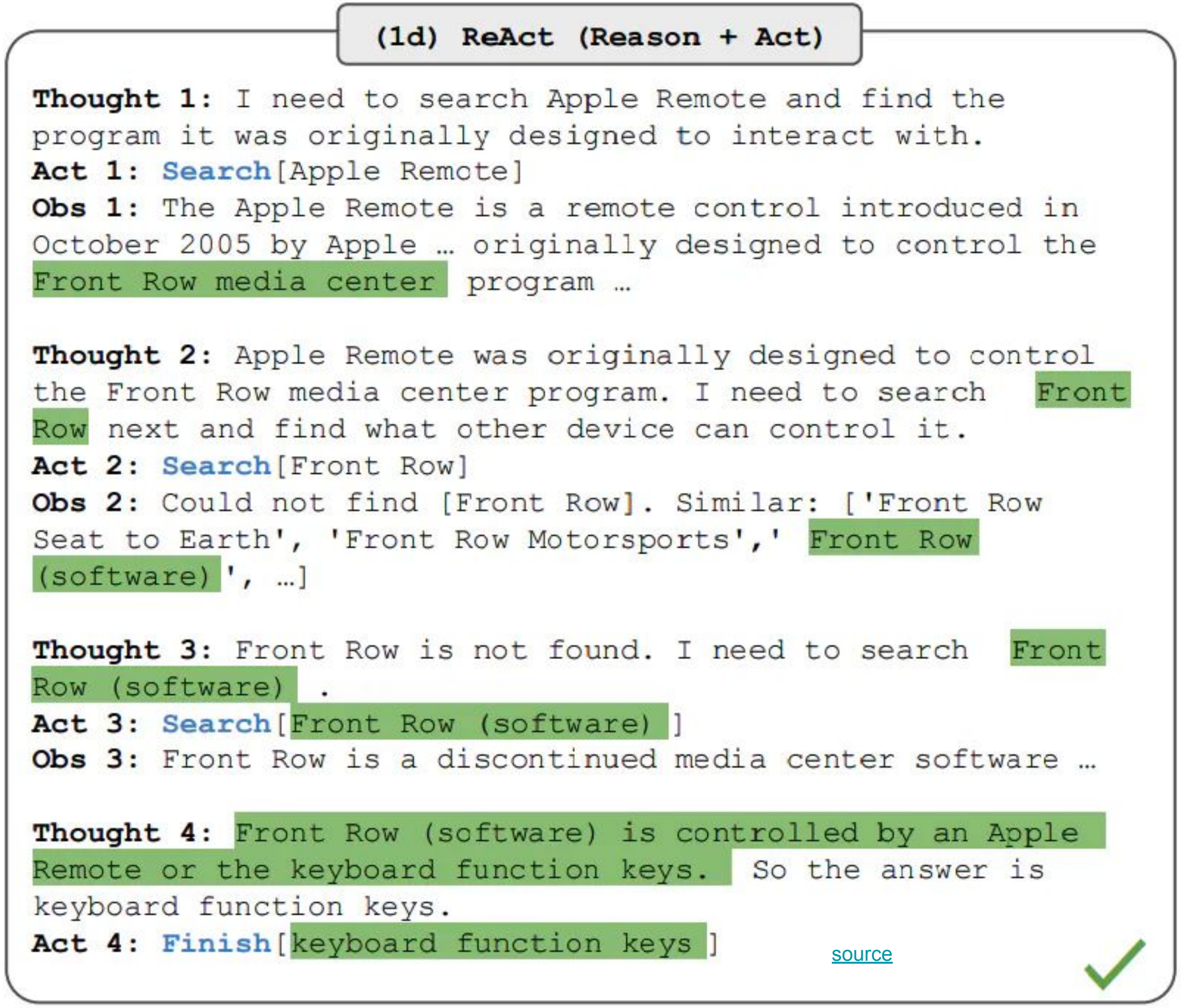


source

ReAct

- Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models"
- For tasks which require reading, multiple steps, and tools





(1d) ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.
Act 1: Search[Apple Remote]
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple … originally designed to control the Front Row media center program …

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports',' Front Row (software) ', …]

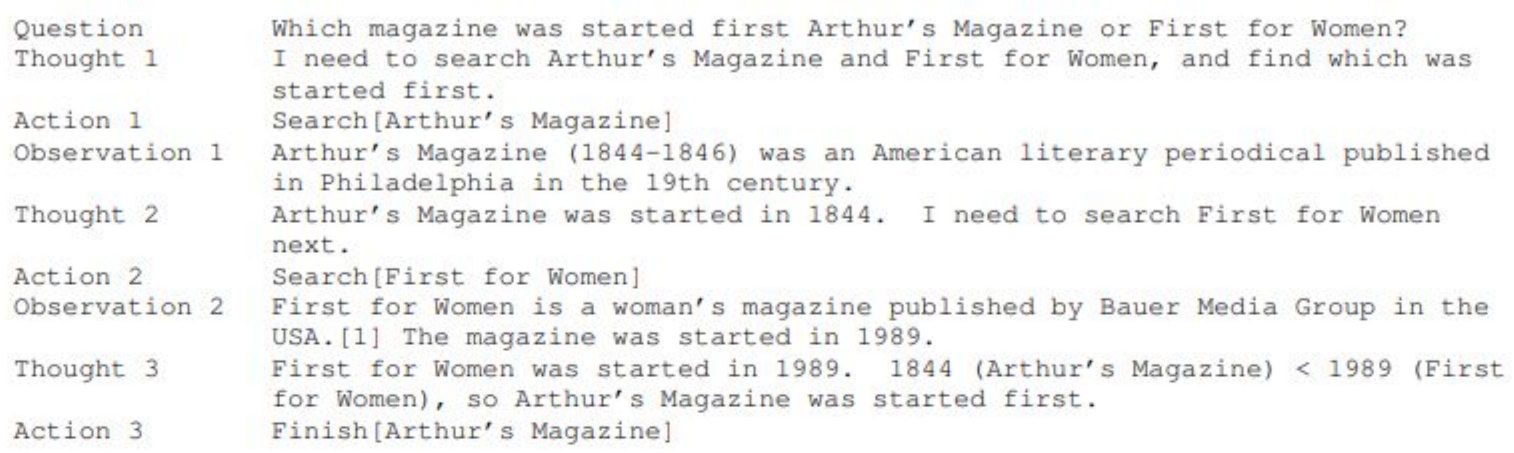
Thought 3: Front Row is not found. I need to search Front Row (software) .
Act 3: Search[Front Row (software)]
Obs 3: Front Row is a discontinued media center software …

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
Act 4: Finish[keyboard function keys]

✓

source

ReAct



Question	Which magazine was started first Arthur’s Magazine or First for Women?
Thought 1	I need to search Arthur’s Magazine and First for Women, and find which was started first.
Action 1	Search[Arthur’s Magazine]
Observation 1	Arthur’s Magazine (1844-1846) was an American literary periodical published in Philadelphia in the 19th century.
Thought 2	Arthur’s Magazine was started in 1844. I need to search First for Women next.
Action 2	Search[First for Women]
Observation 2	First for Women is a woman’s magazine published by Bauer Media Group in the USA.[1] The magazine was started in 1989.
Thought 3	First for Women was started in 1989. 1844 (Arthur’s Magazine) < 1989 (First for Women), so Arthur’s Magazine was started first.
Action 3	Finish[Arthur’s Magazine]

source

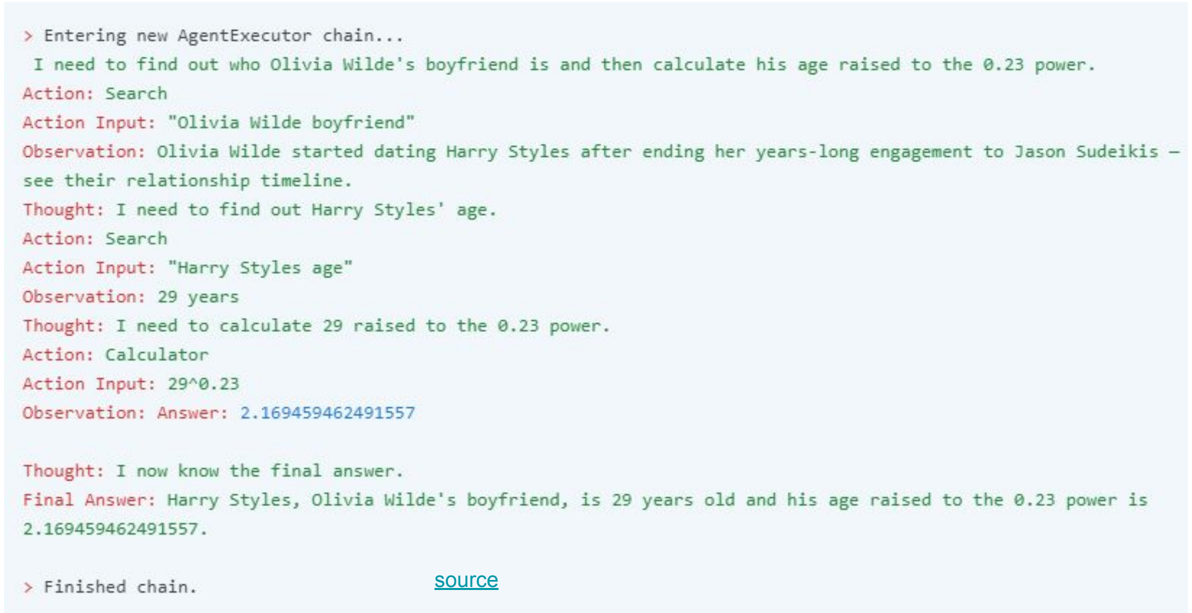
LangChain ReAct: Example

llm = OpenAI(model_name="text-davinci-003" ,temperature=0)

tools = load_tools(["google-serper", "llm-math"], llm=llm)

agent = initialize_agent(tools, llm, agent="zero-shot-react-description", verbose=True)

agent.run("Who is Olivia Wilde's boyfriend? What is his current age raised to the 0.23 power?")



```
> Entering new AgentExecutor chain...
 I need to find out who Olivia Wilde's boyfriend is and then calculate his age raised to the 0.23 power.
Action: Search
Action Input: "Olivia Wilde boyfriend"
Observation: Olivia Wilde started dating Harry Styles after ending her years-long engagement to Jason Sudeikis —
see their relationship timeline.
Thought: I need to find out Harry Styles' age.
Action: Search
Action Input: "Harry Styles age"
Observation: 29 years
Thought: I need to calculate 29 raised to the 0.23 power.
Action: Calculator
Action Input: 29^0.23
Observation: Answer: 2.169459462491557

Thought: I now know the final answer.
Final Answer: Harry Styles, Olivia Wilde's boyfriend, is 29 years old and his age raised to the 0.23 power is
2.169459462491557.

> Finished chain.
```

source

Zero-shot ReAct Agent (No Memory)

```
from langchain.agents import initialize_agent
agent = initialize_agent(
    agent="zero-shot-react-description",
    tools=tools,
    llm=llm,
    verbose=True,
    max_iterations=5
)
```

Conversational ReAct Agent (With Memory)

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

conversational_agent = initialize_agent(
   agent="conversational-react-description",
   tools=tools,
   llm=llm,
   verbose=True,
   max_iterations=5,
   memory=memory
)
```

Demo

- https://colab.research.google.com/github/run-llama/llama_index/blob/main/docs/docs/examples/agent/react_agent.ipynb
- https://github.com/dair-ai/Prompt-Engineering-Guide/blob/main/notebooks/react.ipynb

Online Resources

- https://github.com/pinecone-io/examples/tree/master/learn/generation/langchain/handbook
- https://www.trychroma.com/
- https://github.com/ksm26/vector-databases-embeddings-applications
- https://python.langchain.com/v0.1/docs/cookbook/