

# **Intro to LangChain**

**Framework to build applications using LLMs**

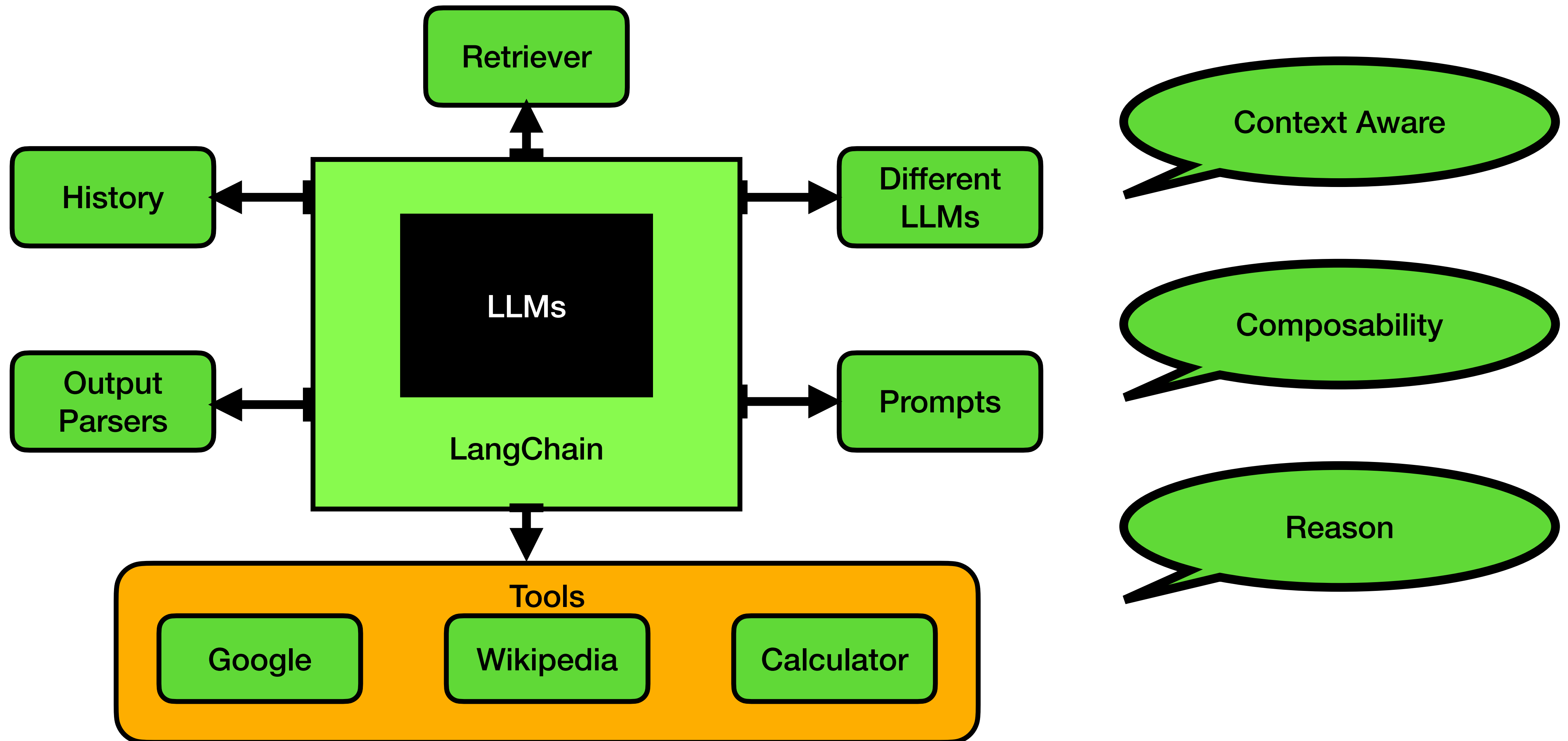
**By Sarath Chandra on 16th Jan 2024**

# Table of Contents

- What is LangChain?
- Build a ChatBot to understand its components
  - Simple LLM Chain
  - Retrieval Chain
  - Conversational Retrieval Chain
  - Agents
- LangChain Framework overview

# What is LangChain?

Framework to build end-to-end applications using LLMs



# Build a ChatBot to understand its components

Task is to answer: What is LangSmith?

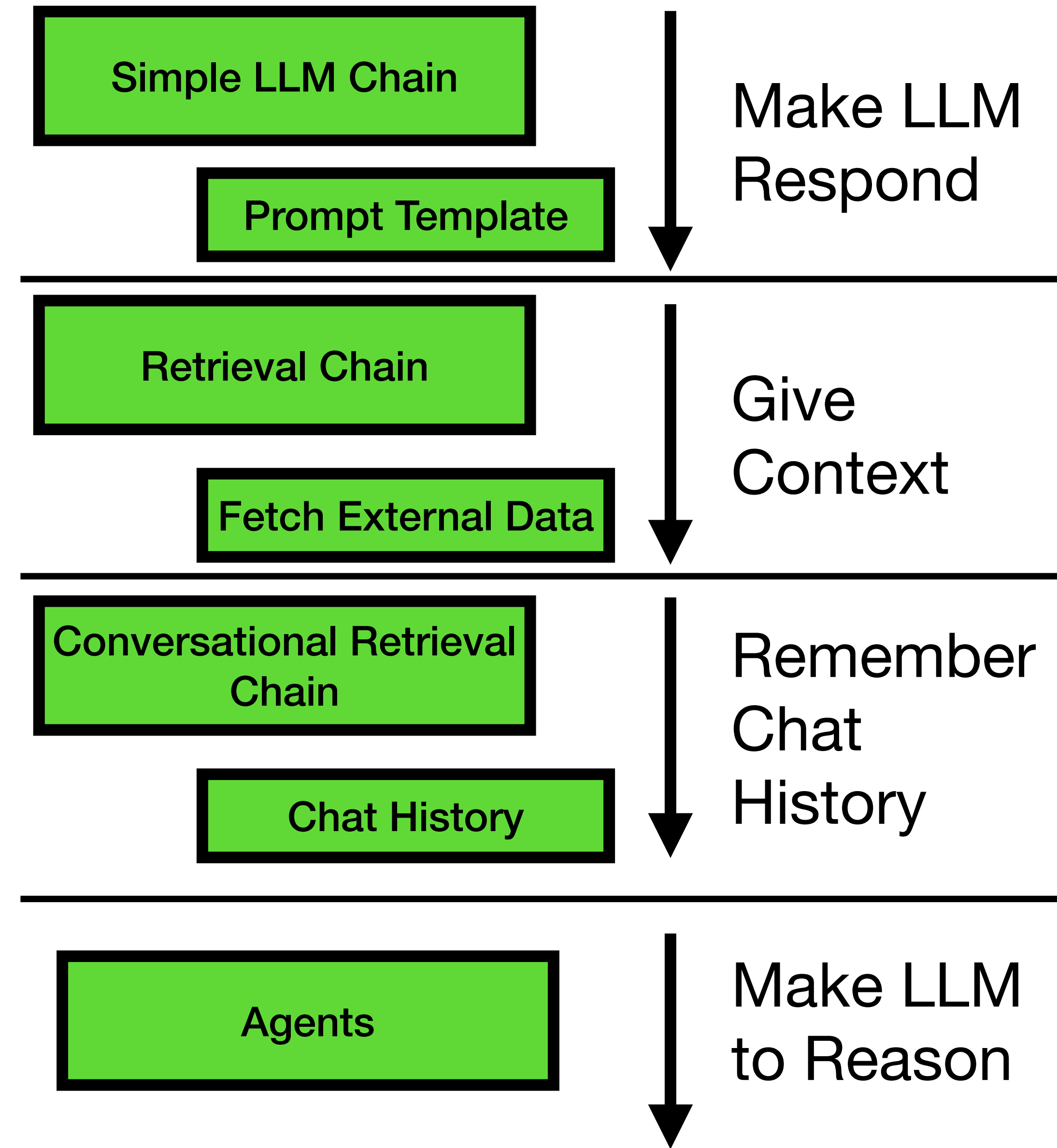
SA You  
What is LangSmith?

ChatGPT  
As of my last knowledge update in January 2022, I don't have specific information about something called "LangSmith." It's possible that there have been developments or new information since then. If "LangSmith" refers to a technology, product, person, or concept that emerged after that date, I recommend checking the latest sources such as news articles, company websites, or other reliable resources for the most up-to-date information. If "LangSmith" is a common term or a less-known entity, providing additional context might help me better understand and provide relevant information.

LangSmith is a platform for building production-grade LLM applications.

It lets you **debug, test, evaluate, and monitor chains and intelligent agents** built on any LLM framework and seamlessly integrates with LangChain

<https://docs.smith.langchain.com/>





# Simple LLM Chain

```
# using OpenAI (a popular model available via API)
from langchain openai import ChatOpenAI
llm = ChatOpenAI()
```

ChatModels

```
llm.invoke("What is langsmith?")
```

```
AIMessage(content='Langsmith is a surname of English origin. It is derived from the Old English words "lang," meaning long, and "smith," meaning a worker in metal. Therefore, Langsmith likely originally referred to someone who worked as a blacksmith or metalworker.')
```

```
# We can guide it's response with a prompt template.
# Prompt templates are used to convert raw user input to a better input to the LLM
from langchain core.prompts import ChatPromptTemplate
prompt = ChatPromptTemplate.from_messages([
    ("system", "You are world class technical documentation writer. Be as true as possible"),
    ("user", "{input}")
])
```

PromptTemplates  
SystemMessage  
HumanMessage  
AIMessage

```
# combine these into a simple LLM chain
chain = prompt | llm
```

Chains  
(part of LCEL)  
Intuitive way to  
combine  
components

```
# invoke it and ask the same question.
# It still won't know the answer, but it should respond in a more proper tone for a technical writer!
chain.invoke({"input": "What is LangSmith?"})
```

```
AIMessage(content='LangSmith is a fictional company that provides language translation and localization services for businesses and organizations around the world. With a team of expert linguists and advanced technology, LangSmith offers a wide range of language solutions to help clients effectively communicate and connect with their global audience. The company is known for its high-quality translations, cultural adaptation, and efficient project management, making it a trusted partner for multilingual communication needs.')
```

```
# simple output parser to convert the chat message to a string
from langchain_core.output_parsers import StrOutputParser
output_parser = StrOutputParser()
```

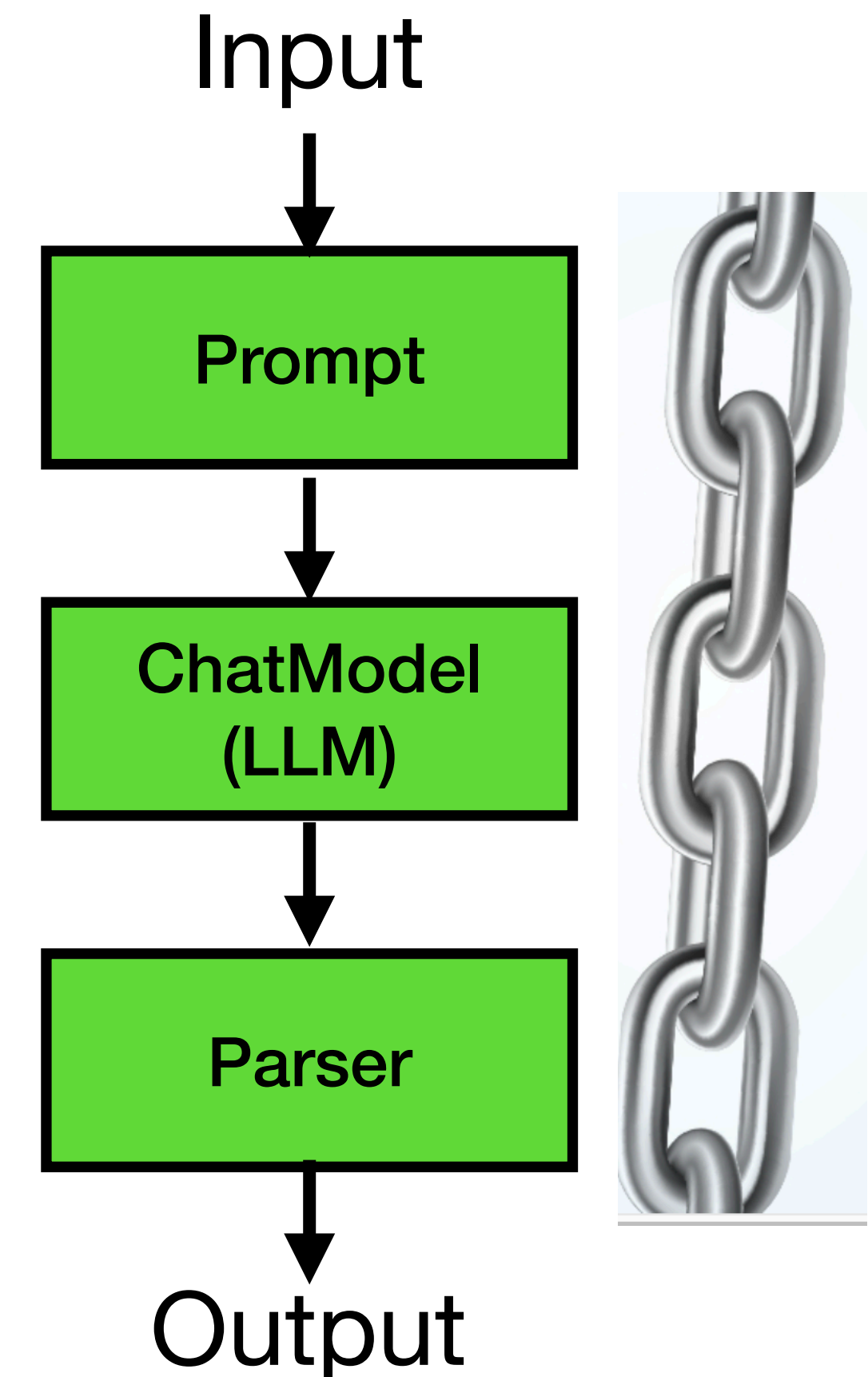
Output Parser

```
chain = prompt | llm | output_parser
```

```
chain.invoke({"input": "What is LangSmith?"})
```

```
'LangSmith is a fictional company that provides language translation and localization services for businesses and organizations around the world. With a team of expert linguists and advanced technology, LangSmith offers a wide range of language solutions to help clients effectively communicate and connect with their global audience. From document translation to website localization, LangSmith is committed to delivering high-quality and accurate language services to meet the diverse needs of its clients.'
```

- ChatModels
- PromptTemplates
- Chains
- Output Parsers





# Retrieval Chain

```
from langchain_community.document_loaders import WebBaseLoader
loader = WebBaseLoader("https://docs.smith.langchain.com/overview")

docs = loader.load()
```

Document Loaders

```
from langchain_openai import OpenAIEmbeddings
embeddings = OpenAIEmbeddings()
```

Embeddings

```
from langchain_community.vectorstores import FAISS
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter()
documents = text_splitter.split_documents(docs)
vector = FAISS.from_documents(documents, embeddings)
```

Vector Stores

```
from langchain.chains import create_retrieval_chain
retriever = vector.as_retriever()
```

Retriever

```
from langchain.chains.combine_documents import create_stuff_documents_chain

prompt = ChatPromptTemplate.from_template("""Answer the following question based only on the provided context:
<context>
{context}
</context>
Question: {input}""")

document_chain = create_stuff_documents_chain(llm, prompt)
```

Takes a list of documents and formats them all into a prompt, then passes that prompt to an LLM

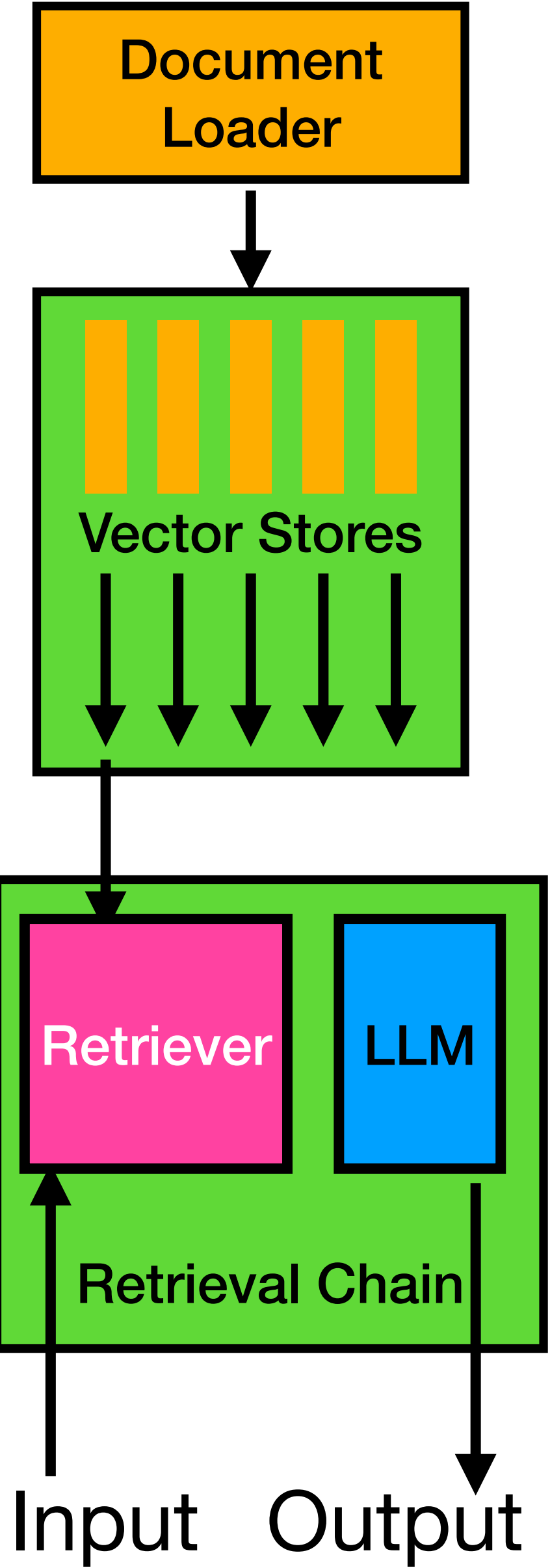
```
retrieval_chain = create_retrieval_chain(retriever, document_chain)
```

Takes in a user inquiry, passes to the retriever to fetch relevant documents.

```
response = retrieval_chain.invoke({"input": "What is LangSmith?"})
print(response["answer"])
```

Those documents (and original inputs) are then passed to an LLM to generate a response

LangSmith is a tool developed by LangChain to help with debugging, monitoring, collaborative debugging, collecting examples, testing and evaluation of LLM (Large Language Model) applications. It provides features such as tracing, monitoring latency and token usage statistics, visualizing the sequence of events in complicated chains and agents, and exporting datasets for use in other contexts.



# Conversational Retrieval Chain

```
from langchain_core.messages import HumanMessage, AIMessage
chat_history = [HumanMessage(content="Can LangSmith help test my LLM applications?"), AIMessage(content="Yes!")]

from langchain.chains import create_history_aware_retriever
from langchain_core.prompts import MessagesPlaceholder

# First we need a prompt that we can pass into an LLM to generate this search query
prompt = ChatPromptTemplate.from_messages([
    MessagesPlaceholder(variable_name="chat_history"),
    ("user", "{input}"),
    ("user", "Given the above conversation, generate a search query to look up in order to get information relevant to the conversation")
])
retriever_chain = create_history_aware_retriever(llm, retriever, prompt)

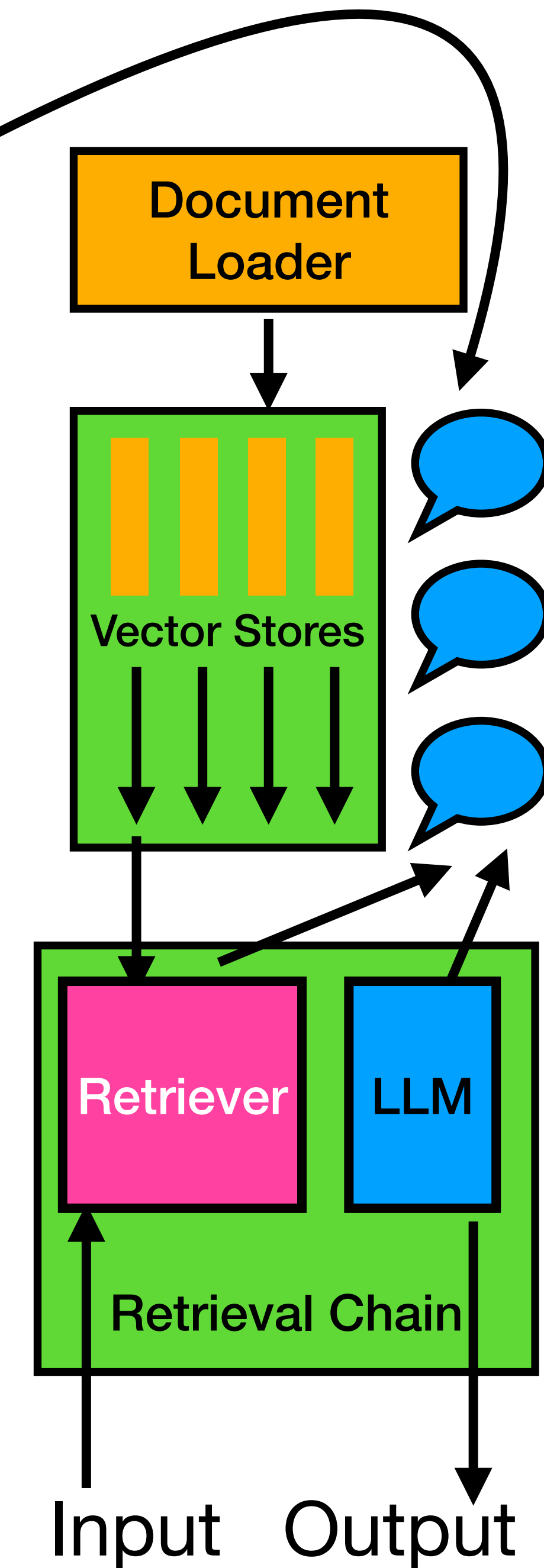
retriever_chain.invoke({
    "chat_history": chat_history,
    "input": "Tell me how"
})

prompt = ChatPromptTemplate.from_messages([
    ("system", "Answer the user's questions based on the below context:\n\n{context}"),
    MessagesPlaceholder(variable_name="chat_history"),
    ("user", "{input}"),
])
document_chain = create_stuff_documents_chain(llm, prompt)
retrieval_chain = create_retrieval_chain(retriever_chain, document_chain)

chat_history = [HumanMessage(content="Can LangSmith help test my LLM applications?"), AIMessage(content="Yes!")]
retrieval_chain.invoke({
    "chat_history": chat_history,
    "input": "Tell me how"
})
```

Chat History  
history.messages

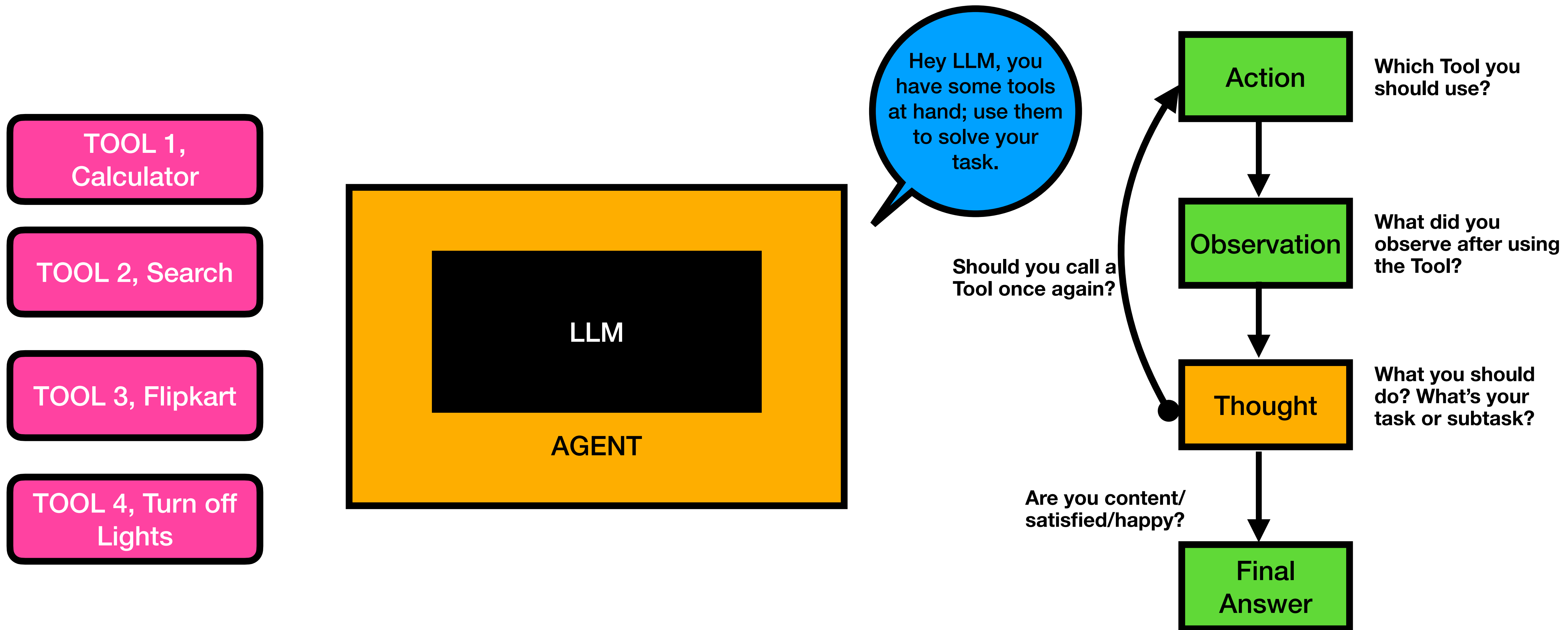
Takes in conversation history and then uses that to generate a search query which is passed to the underlying retriever.



- Memory

# Agents

**Agents are systems that use a language model to interact with other tools**





# Agents

**Agents are systems that use a language model to interact with other tools**

```
# Chain that interprets a prompt and executes python code to do math.
llm_math = LLMMathChain.from_llm(OpenAI())
```

```
@tool
def calculator(query: str) -> str:
    """A calculator for performing math."""
    return llm_math.invoke(query)
```

Way to Define Custom Tools

```
# Tavily's Search API is a search engine built specifically for AI agents (LLMs)
search = TavilySearchResults(max_results=1)
```

Search Tool for Performing Web Search

```
tools = [search, calculator]
```

ReAct Prompt where LLMs are used to generate both reasoning traces and task-specific actions in an interleaved manner

```
# Get the prompt to use - you can modify this!
prompt = hub.pull("hwchase17/react")
prompt
```

ReAct: Synergizing Reasoning and Acting in Language Models

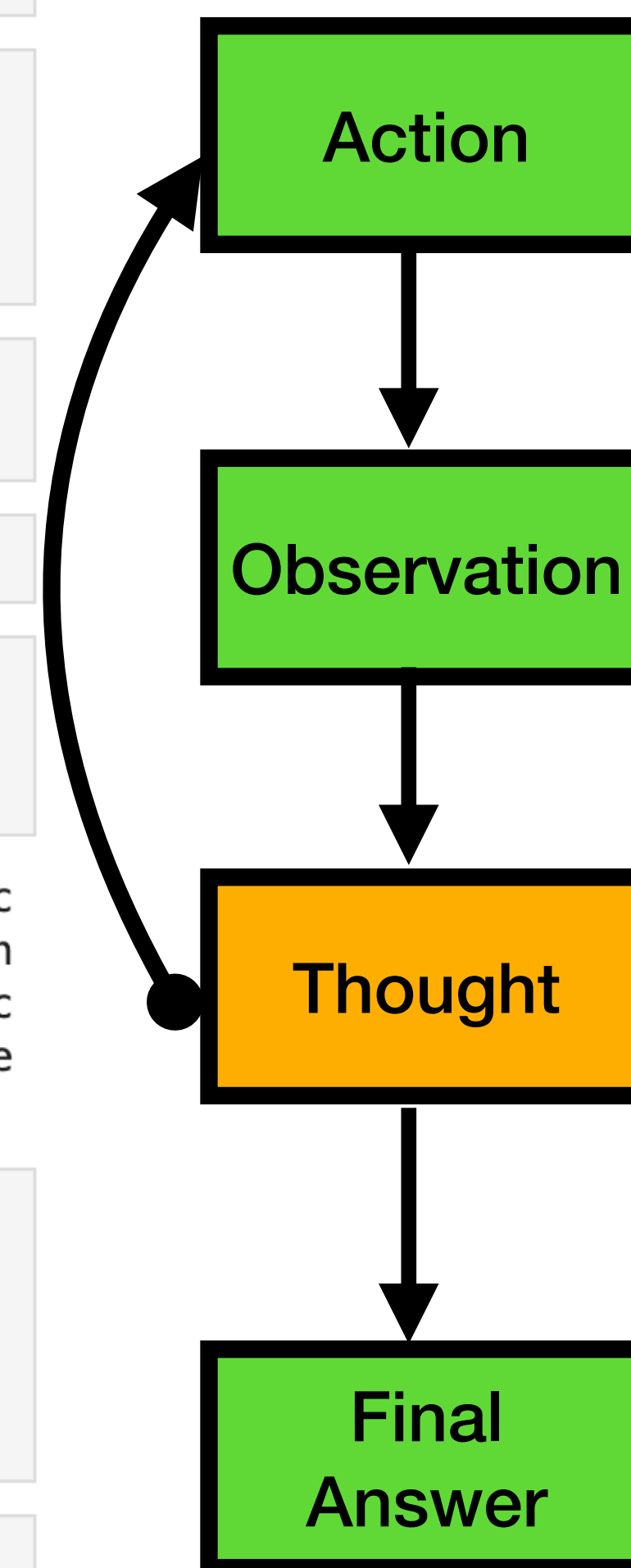
```
PromptTemplate(input_variables=['agent_scratchpad', 'input', 'tool_names', 'tools'], template='Answer the following questions as best you can. You have access to the following tools:\n\n{tools}\n\nUse the following format:\n\nQuestion: the input question you must answer\nThought: you should always think about what to do\nAction: the action to take, should be one of [{tool_names}]\nAction Input: the input to the action\nObservation: the result of the action\n... (this Thought/Action/Action Input/Observation can repeat N times)\nThought: I now know the final answer\nFinal Answer: the final answer to the original input question\n\nBegin!\n\nQuestion: {input}\nThought:{agent_scratchpad}')
```

```
# Choose the LLM to use
llm = OpenAI()
```

```
# Construct the ReAct agent
agent = create_react_agent(llm, tools, prompt)
```

Agents are systems that use a language model to interact with other tools

```
# Create an agent executor by passing in the agent and tools
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
```





```
agent_executor.invoke({"input": "What is the weather in Bangalore? Convert it to Celsius."})
```

> Entering new AgentExecutor chain...

We should use a search engine optimized for current events to find the latest weather information.

Action: tavily\_search\_results\_json

Action Input: "weather in Bangalore" [{"url": "https://weatherspark.com/h/m/108998/2024/1/Historical-Weather-in-January-2024-in-Bangaluru-India", "content": "January 2024 Weather History in Bengaluru India Bengaluru Temperature History January 2024 Hourly Temperature in January 2024 in Bengaluru Observed Weather in January 2024 in Bengaluru Hours of Daylight and Twilight in January 2024 in Bengaluru Compare Bengaluru to another city: Cloud Cover in January 2024 in BengaluruPartly Cloudy 1,500 ft. This report shows the past weather for Bengaluru, providing a weather history for January 2024. It features all historical weather data series we have available, including the Bengaluru temperature history for January 2024. You can drill down from year to month and even day level reports by clicking on the graphs."}]

We should use a calculator to convert the temperature from Fahrenheit to Celsius.

Action: calculator

Action Input: (77-32) \* 5/9

> Entering new LLMMathChain chain...

(77-32) \* 5/9

...numexpr.evaluate("(77-32) \* 5/9")...  
Answer: 25.0

> Finished chain.

{'question': '(77-32) \* 5/9', 'answer': 'Answer: 25.0'} 25.0 degrees Celsius is the current temperature in Bangalore.

Final Answer: The current weather in Bangalore is 25.0 degrees Celsius.

> Finished chain.

{'input': 'What is the weather in Bangalore? Convert it to Celsius.',  
'output': 'The current weather in Bangalore is 25.0 degrees Celsius.'}

Thought

Action

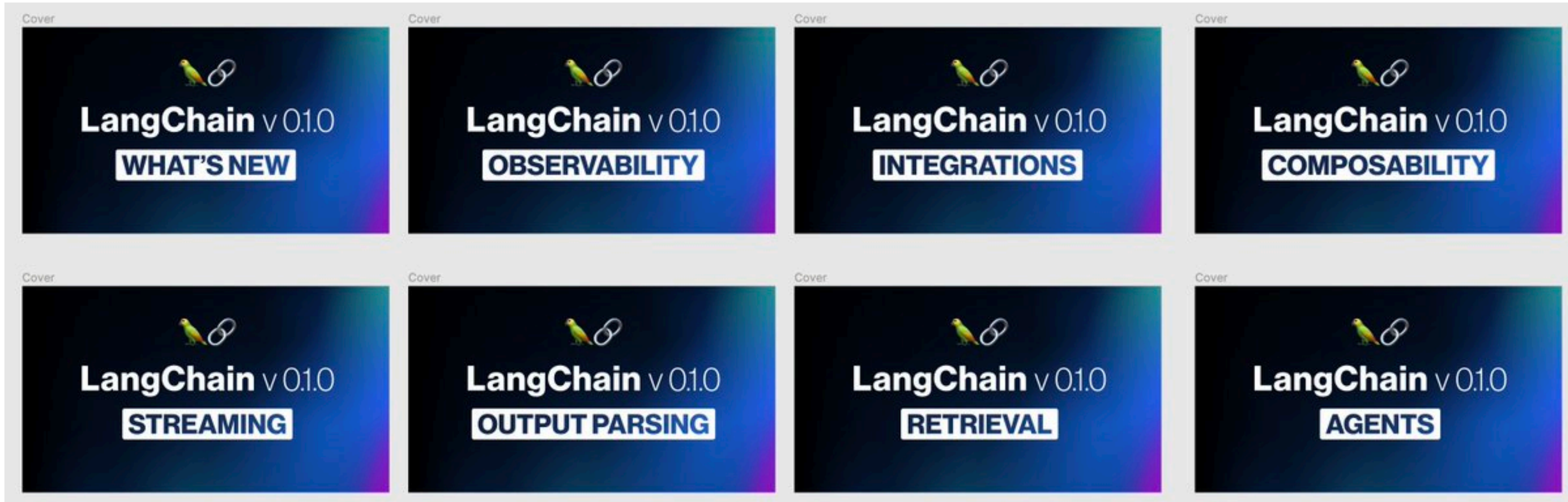
Observation

Final Answer



# LangChain Framework overview

LangChain Releases their v0.1.0, first stable version on Jan 8 2024.



- [LangChain's Official Page](#), [LangChain v0.1.0 Launch YouTube Playlist](#)