



# LangChain

The Chain of Thought for LLM Application Development


Nithiroj Tripatarasit



## Agenda

1. Introduction
2. Quickstart
3. Components
4. QA Application – RAG
5. Use cases
6. Q & A

# Introduction



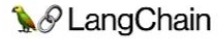
**Large Language Model (LLM)**  
enable us to build context-aware  
applications easier, better and faster.

**LangChain**  

is a framework for developing  
applications powered by language  
models.

# Quickstart

# Language Expression Language (LCEL)



## Interface

- Components implement **"Runnable"** protocol
- Common methods include:
  - `invoke` [`ainvoke`]
  - `stream` [`astream`]
  - `batch` [`abatch`]
- Common properties
  - `input_schema`, `output_schema`
- Common I/O

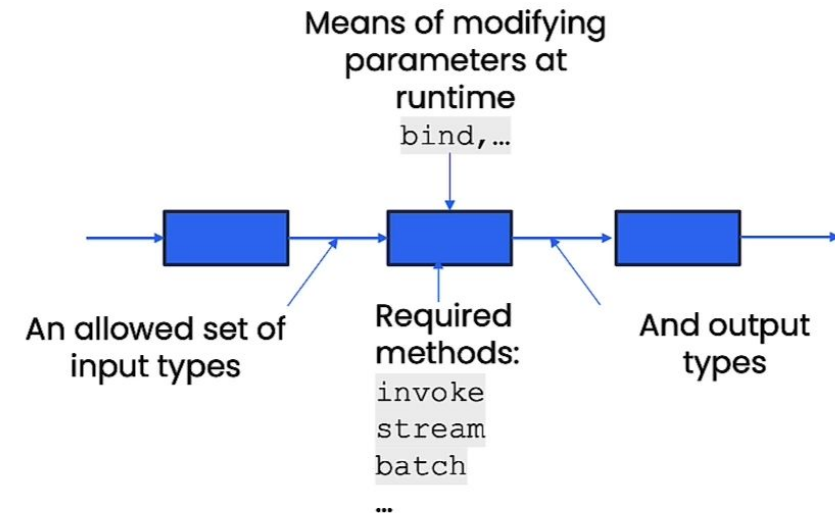
Component	Input Type	Output Type
Prompt	Dictionary	Prompt Value
Retriever	Single String	List of Documents
LLM	String, list of messages or Prompt Value	String
ChatModel	String, list of messages or Prompt Value	ChatMessage
Tool	String/Dictionary	Tool dependent
Output Parser	Output of LLM or ChatModel	Parser dependent



## LangChain Expression Language (LCEL)

LangChain composes chains of **components**

LCEL and the runnable protocol define:



Composition can now use the Linux pipe syntax: |

<https://learn.deeplearning.ai/courses/functions-tools-agents-langchain>

# Building with LangChain

## LLM / Chat Model

- LLM: String
- ChatModel: Message

## Message

- HumanMessage: human/user
- AIMessage; AI/assistant
- SystemMessage: System
- FunctionMessage / ToolMessage: function or tool



LLM Chain = Prompt Template + LLM

## LLM Chain

```
from langchain.prompts import PromptTemplate
from langchain.llms import OpenAI

template = "What is a good name for a company that makes {product}?"
prompt = PromptTemplate.from_template(template)
llm = OpenAI()

chain = prompt | llm

chain.invoke(product="colorful socks")

Bright Socks Co.
```

Chat Chain = Chat Prompt Template + Chat Model

## Chat Prompt Template

```
from langchain.prompts.chat import ChatPromptTemplate

system_template = "You are a helpful assistant that translates  
                    {input_language} to {output_language}."

human_template = "{text}"

chat_prompt = ChatPromptTemplate.from_messages([
    ("system", system_template),
    ("human", human_template)])
```

Chat Chain = Chat Prompt Template + Chat Model

## Chat Chain

```
from langchain.chat_models import ChatOpenAI

chat_model = ChatOpenAI()

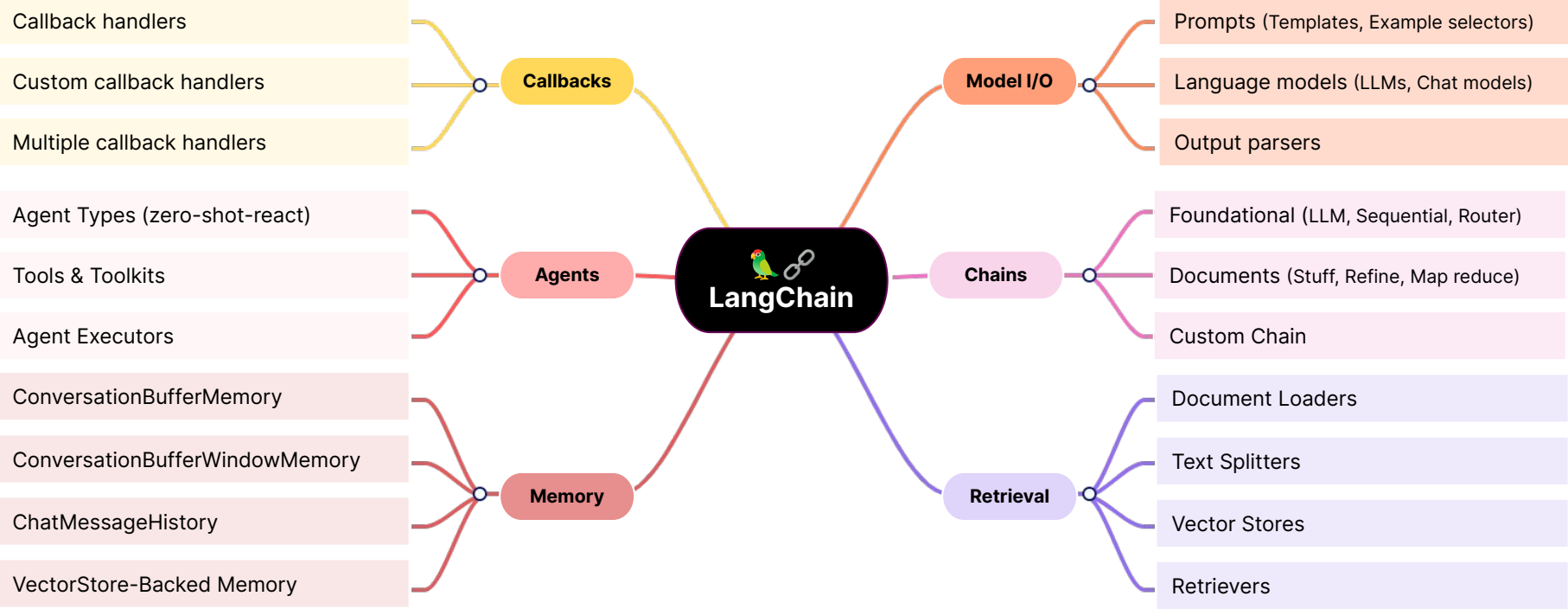
chain = chat_prompt | chat_model

chain.invoke({"input_language": "English",
             "output_language": "French",
             "text": "I love programming."})


AIMessage(content="J'adore programmer.")
```

# Components

# Components





https://integrations.langchain.com/

 **LangChain**

539 integrations [+ Request an integration](#)


[Document Loaders \(160\)](#) [Vector Stores \(63\)](#) [Embedding Models \(47\)](#) [Chat Models \(26\)](#) [LLMs \(78\)](#) [Callbacks \(27\)](#) [Tools \(104\)](#) [Toolkits \(18\)](#) [Message Histories \(16\)](#)

Popularity 

**AirbyteJSONLoader** 


Load local `Airbyte` json files.

[Docs](#) [Github](#) [❤️ 46](#)

**ApifyDatasetLoader** 


Load datasets from `Apify` web scraping,...

[Docs](#) [Github](#) [❤️ 38](#)

**UnstructuredHTMLLoader** 


Load `HTML` files using `Unstructured`. Yo...

[Docs](#) [Github](#) [❤️ 30](#)

**UnstructuredPDFLoader** 


Load `PDF` files using `Unstructured`. You...

[Docs](#) [Github](#) [❤️ 30](#)

**UnstructuredCSVLoader** 

Load `CSV` files using `Unstructured`. Like...

[Docs](#) [Github](#) [❤️ 28](#)

**UnstructuredURLLoader** 


Load files from remote URLs using...

[Docs](#) [Github](#) [❤️ 26](#)

**OnlinePDFLoader**


Load online `PDF`.

[Docs](#) [Github](#) [❤️ 25](#)

**UnstructuredMarkdownLoader** 

Load `Markdown` files using `Unstructured`...

[Docs](#) [Github](#) [❤️ 24](#)

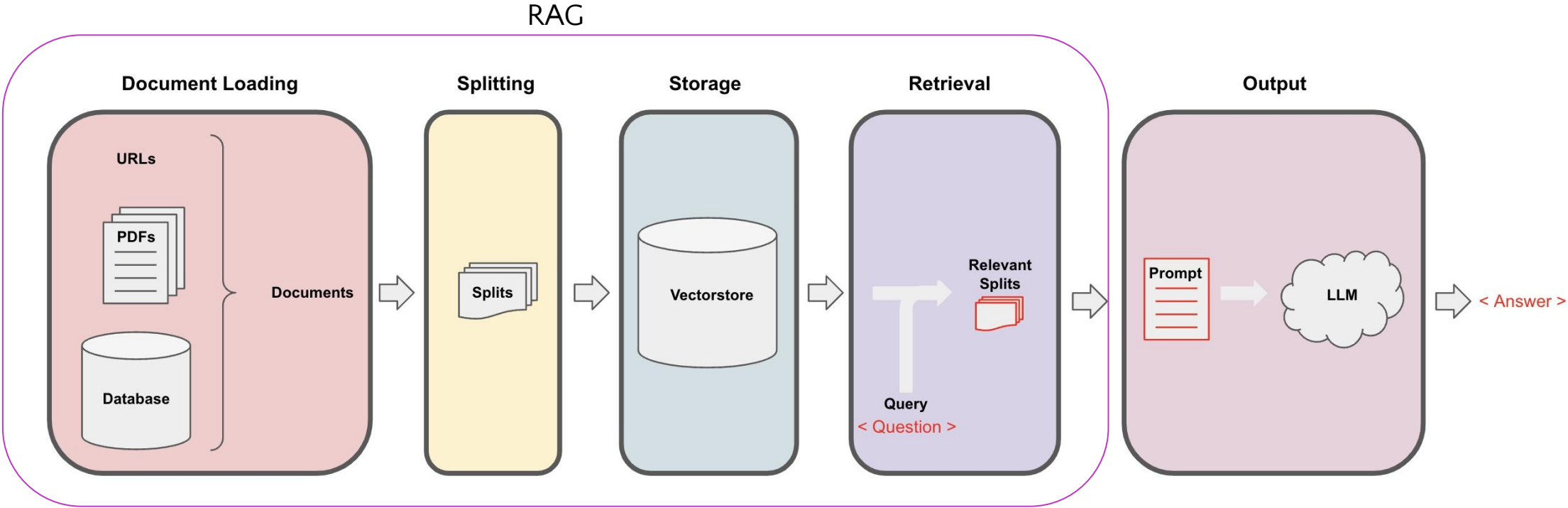
**UnstructuredFileLoader** 

Load files using `Unstructured`. The file...

[Docs](#) [Github](#) [❤️ 23](#)

# QA Application – RAG

# Retrieval-Augmented Generation (RAG)



[https://python.langchain.com/docs/use\\_cases/question\\_answering/code\\_understanding](https://python.langchain.com/docs/use_cases/question_answering/code_understanding)



# Scenario

Lil'Log

[Posts](#) [Archive](#) [Search](#) [Tags](#) [FAQ](#) [emojisearch.app](#)

## LLM Powered Autonomous Agents

Date: June 23, 2023 | Estimated Reading Time: 31 min | Author: Lilian Weng

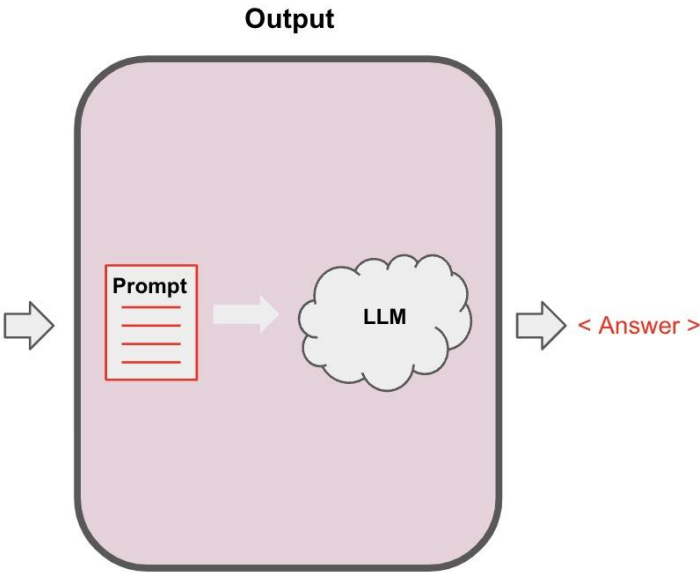
▶ Table of Contents

Building agents with LLM (large language model) as its core controller is a cool concept. Several proof-of-concepts demos, such as [AutoGPT](#), [GPT-Engineer](#) and [BabyAGI](#), serve as inspiring examples. The potentiality of LLM extends beyond generating well-written copies, stories, essays and programs; it can be framed as a powerful general problem solver.

### Agent System Overview

In a LLM-powered autonomous agent system, LLM functions as the agent's brain, complemented by several key components:

- **Planning**
  - Subgoal and decomposition: The agent breaks down large tasks into smaller, manageable subgoals, enabling efficient handling of complex tasks.
  - Reflection and refinement: The agent can do self-criticism and self-reflection over past actions, learn from mistakes and refine them for future steps, thereby improving the quality of final results.
- **Memory**



<https://lilianweng.github.io/posts/2023-06-23-agent/>

## RAG for QA on Documents

### Step 1. Document Loading

```
from langchain.document_loaders import WebBaseLoader

loader = WebBaseLoader(
    "https://lilianweng.github.io/posts/2023-06-23-agent/"
)

data = loader.load()
```

## RAG for QA on Documents

### Step 2. Splitting

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
                                              chunk_overlap=0)

all_splits = text_splitter.split_documents(data)
```

## RAG for QA on Documents

### Step 3. Storage

```
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import Chroma

vectorstore = Chroma.from_documents(documents=all_splits,
                                     embedding=OpenAIEmbeddings())
```

## RAG for QA on Documents

### Step 4. Retrieval

```
retriever = vectorstore.as_retriever()
```

## RAG for QA on Documents

### Step 5. Generate Output

```
# Prompt, https://smith.langchain.com/hub/r1m/rag-prompt  
# pip install langchainhub
```

```
from langchain import hub  
rag_prompt = hub.pull("r1m/rag-prompt")  
rag_prompt
```

```
ChatPromptTemplate(input_variables=['context', 'question'],  
messages=[HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['context',  
'question'], template="You are an assistant for question-answering tasks. Use the  
following pieces of retrieved context to answer the question. If you don't know the  
answer, just say that you don't know. Use three sentences maximum and keep the answer  
concise.\nQuestion: {question} \nContext: {context} \nAnswer:"))])
```

### Step 5. Generate Output

```
from langchain.chat_models import ChatOpenAI
from langchain.schema.runnable import RunnablePassthrough

# Chat model
model = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)

# RAG chain
rag_chain = {"context": retriever,
            "question": RunnablePassthrough()}
            | rag_prompt
            | model
```

## RAG for QA on Documents

### Step 5. Generate Output

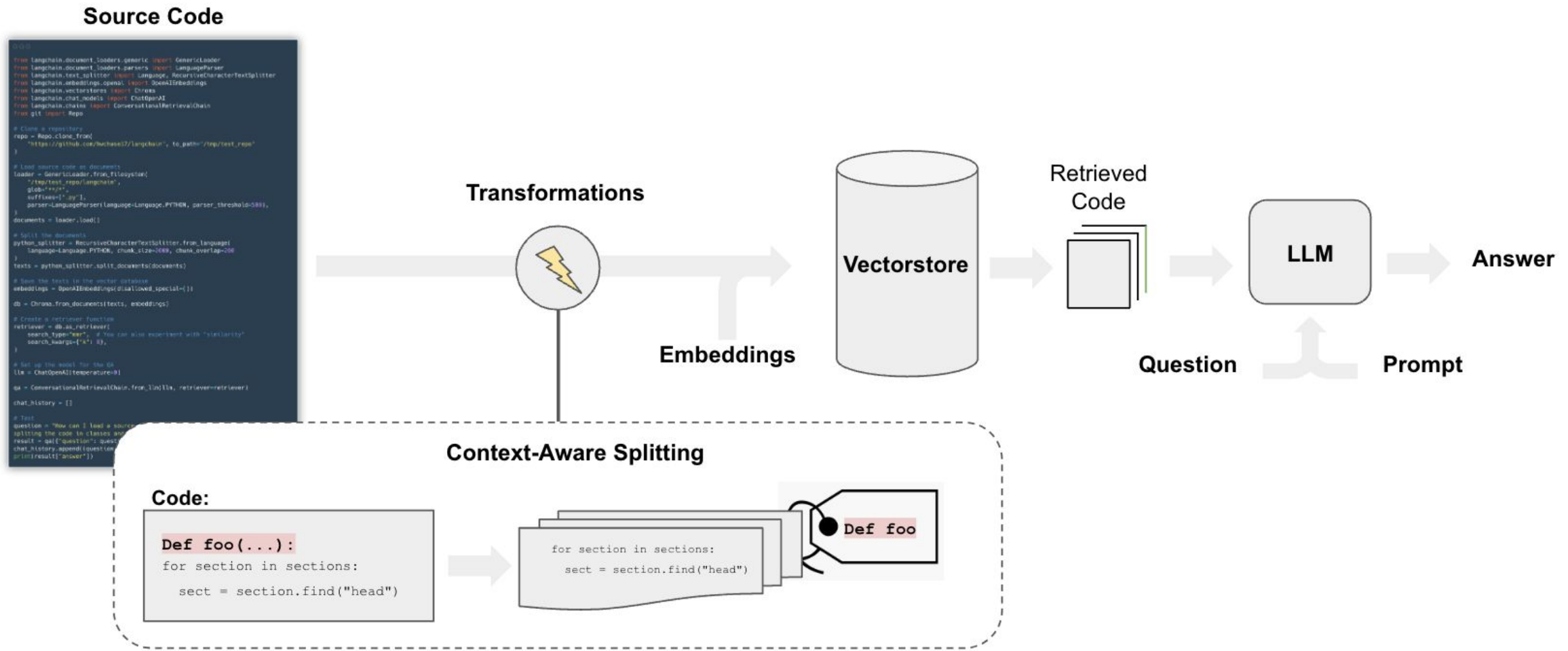
```
rag_chain.invoke("What is Task Decomposition?")
```

```
AIMessage(content='Task decomposition is the process of breaking down a large task into smaller, manageable subgoals. It enables efficient handling of complex tasks and improves the quality of final results. However, long-term planning and effective exploration of the solution space remain challenging for language models like LLM.')
```



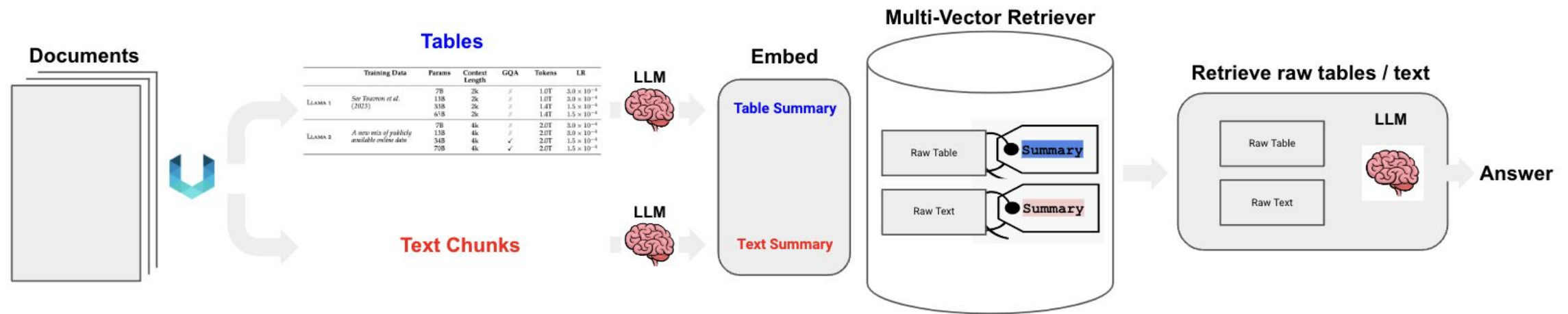
# Use cases

# RAG over code



[https://python.langchain.com/docs/use\\_cases/question\\_answering/code\\_understanding](https://python.langchain.com/docs/use_cases/question_answering/code_understanding)

# Semi-structured RAG



[https://github.com/langchain-ai/langchain/blob/master/cookbook/Semi\\_Structured\\_RAG.ipynb](https://github.com/langchain-ai/langchain/blob/master/cookbook/Semi_Structured_RAG.ipynb)

# Resources

## LangChain

- <https://python.langchain.com>
- <https://blog.langchain.dev/>
- <https://smith.langchain.com/>
- <https://learn.deeplearning.ai/>

## Discord

<https://discord.gg/eeePdCdvGH>

## Youtube

<https://youtube.com/@AlekLearn>

## Jupyter Notebooks

- [https://github.com/aleklearn/repo/blob/main/generative-ai/lc\\_quickstart.ipynb](https://github.com/aleklearn/repo/blob/main/generative-ai/lc_quickstart.ipynb)
- [https://github.com/aleklearn/repo/blob/main/generative-ai/lc\\_rag\\_qa\\_document.ipynb](https://github.com/aleklearn/repo/blob/main/generative-ai/lc_rag_qa_document.ipynb)

## Slide

[https://github.com/aleklearn/repo/blob/main/documents/PyCon\\_Thailand\\_2023.pdf](https://github.com/aleklearn/repo/blob/main/documents/PyCon_Thailand_2023.pdf)

## The Impact of LLM



*"Multilingual  
(English + Python)  
can accomplish  
much more"*



*"The hottest new  
programming  
language is  
English."*



# Q & A