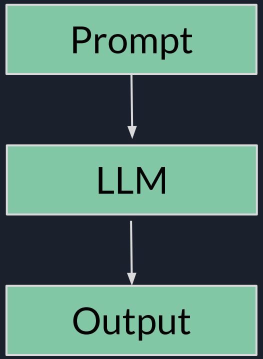
LangChain + RAG

Aman Gupta
amaan.gupta19@gmail.com

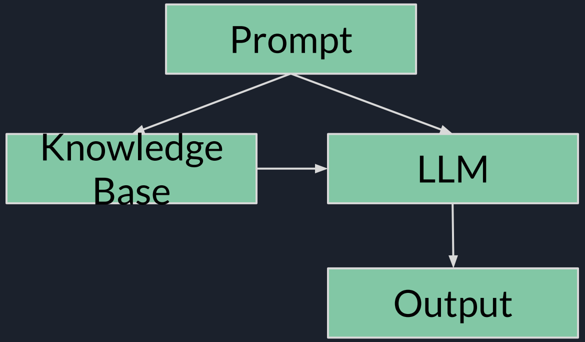
What is RAG

RAG: Retrieval Augmented Generation

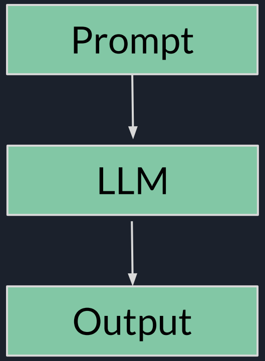
Normal Generation



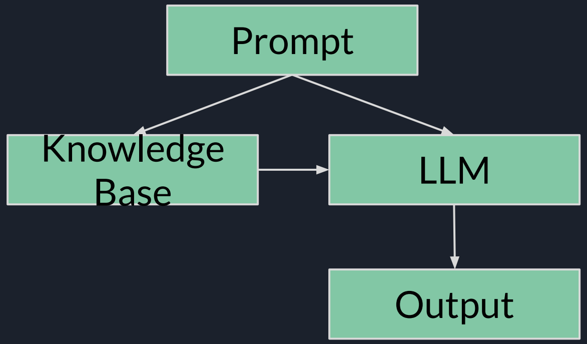
Retrieval Augmented Generation (RAG)



Normal Generation



Retrieval Augmented Generation (RAG)



RAG is using **retrieval** from a knowledge base to **augment generation** of text from a LLM.

Example Without RAG

Prompt

{

prompt: “Who are our largest shareholders and what are their motivations”

}

Response

No Conclusion:

I have no idea who our largest shareholder are

Delusion:

Our largest shareholder is the sovereign wealth fund of Zambia. The sovereign wealth fund of Zambia is funded by Zambia’s large copper mining sector. As such, they are interested in promoting the interest of copper processing companies such as ourselves.

Example with RAG

Augmented Prompt

```
{

context: "Top Institutional Holders

Holder                    % Out
Vanguard Group Inc                          8.97%
Blackrock Inc.          7.77%
State Street Corporation                    5.01%
Wellington Management Group, LLP    4.04%
Capital World Investors                     3.23%
Geode Capital Management, LLC          1.90%
Charles Schwab Investment Management, Inc.                    1.85%
Morgan Stanley       1.50%
Northern Trust Corporation                 1.12%
Norges Bank Investment Management  1.09%
Top Mutual Fund Holders
Holder                    % Out
Vanguard Total Stock Market Index Fund                        3.11%
Vanguard 500 Index Fund                     2.40%
Fidelity 500 Index Fund                     1.18%
SPDR S&P 500 ETF Trust                      1.13%
Vanguard Specialized-Health Care Fund                         1.04%
Schwab Strategic Tr-Schwab U.S. Dividend Equity ETF        1.02%
iShares Core S&P 500 ETF                    0.99%
Washington Mutual Investors Fund        0.93%
Vanguard Index-Value Index Fund         0.86%
Select Sector SPDR Fund-Health Care  0.77%
",

prompt: "Who are our largest shareholders and what are their motivations"

}
```

Response

Our largest shareholders are all **mutual fund investors** who are putting our shares into broad market funds. Their motivation is to represent the US economy for their investors by ho**lding shares in proportion to those of the indexes they are mirroring**.

Why RAG

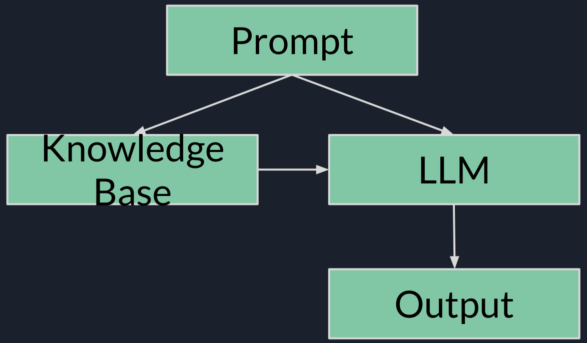
Without RAG, LLMs are **people with amnesia**. Have no specific knowledge.

With RAG, LLMs are people with amnesia with a **newspaper**.

Building a RAG System

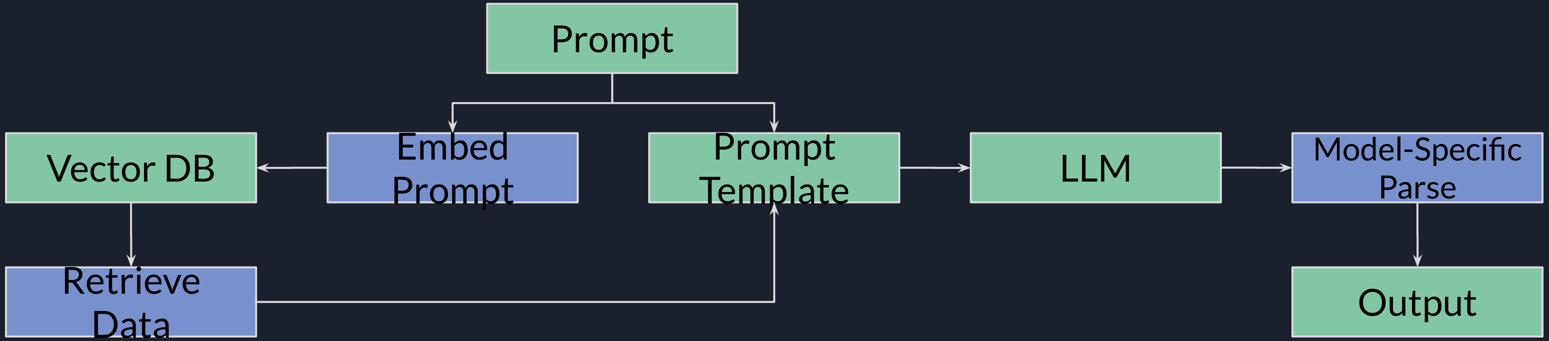
RAG: Retrieval Augmented Generation

Retrieval Augmented Generation (RAG)

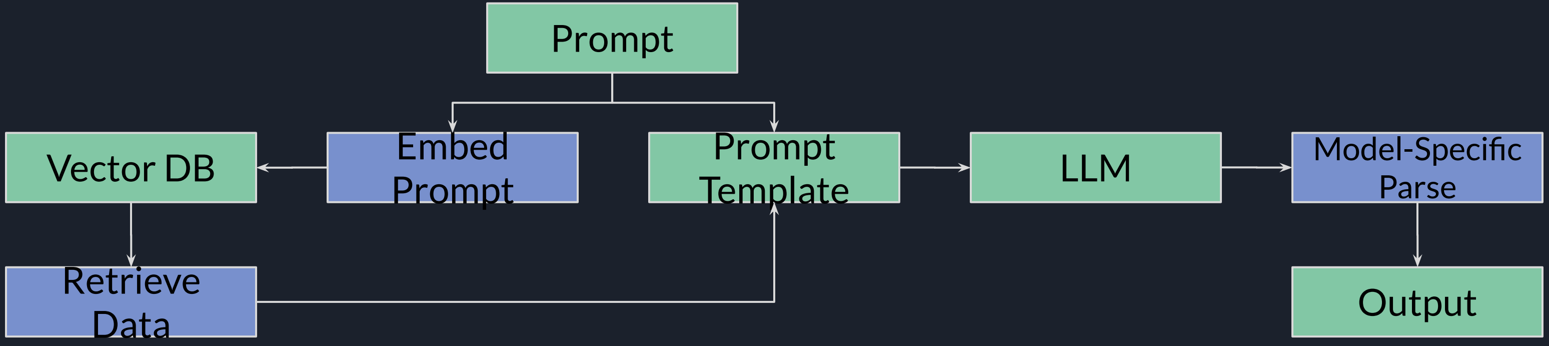


RAG In Practice

Retrieval Augmented Generation (RAG)



Retrieval Augmented Generation (RAG)



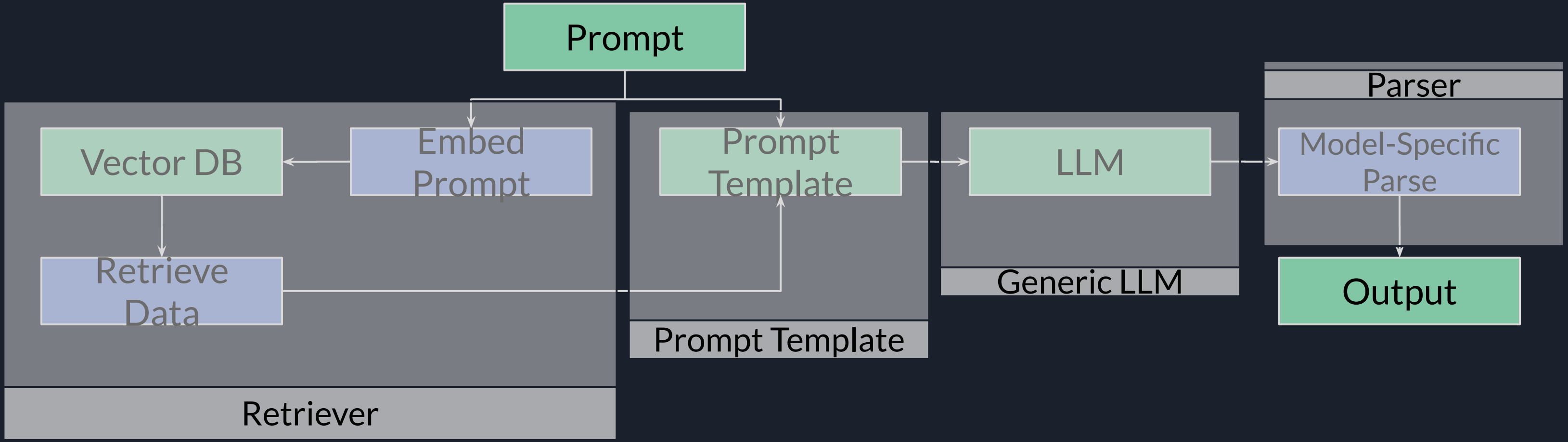
Lots of details to implement such a simple idea, our pipeline view is gone. We now have all this plumbing and work to be done

Langchain

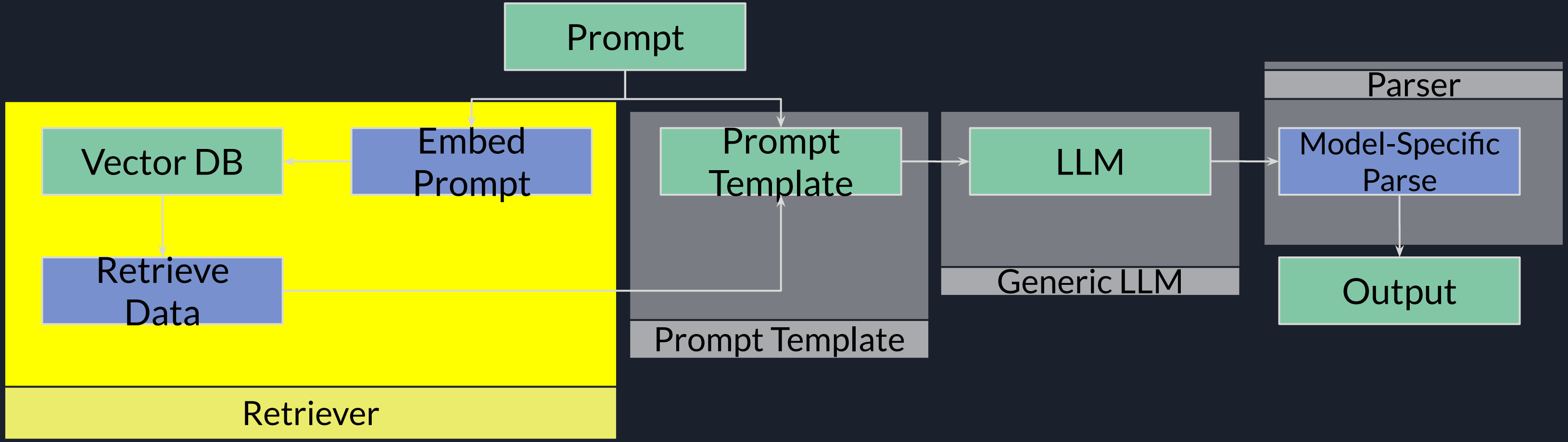
Focus on the LLM App, not API plumbing

RAG - Langchain View

Retrieval Augmented Generation (RAG)

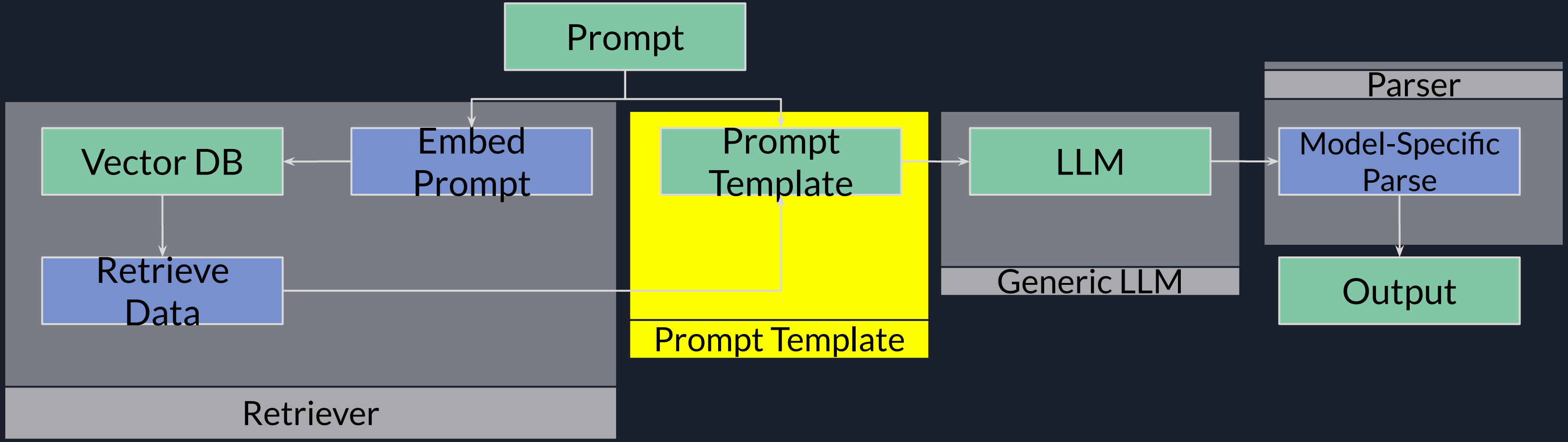


Retrieval Augmented Generation (RAG)



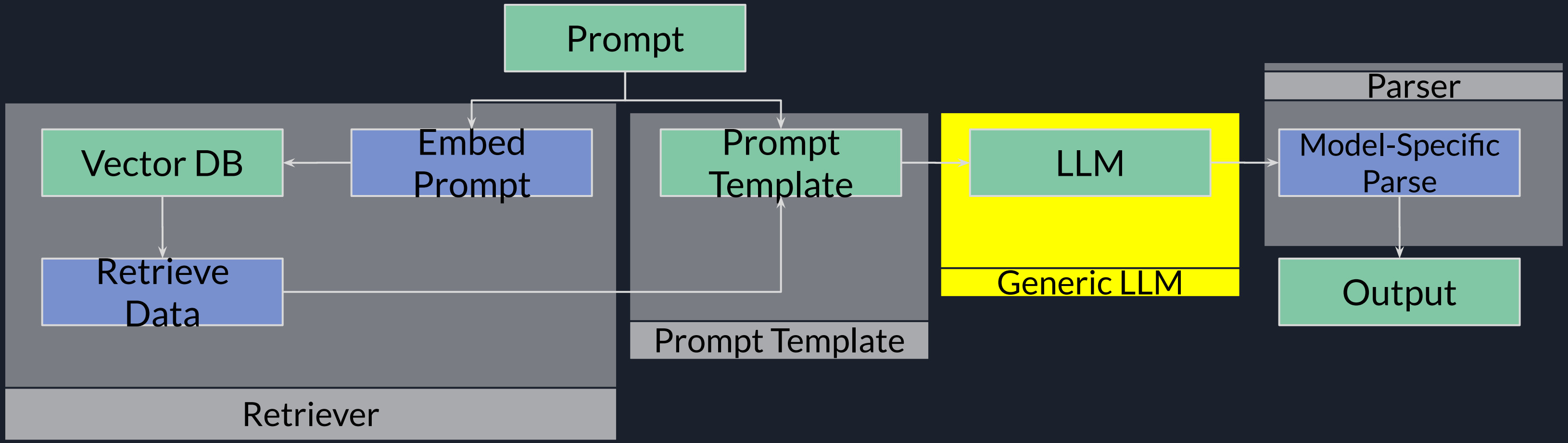
Retriever - Langchain has modules to handle the entire embedd/fetch process. Means the knowledge base is easily changeable as project requirements change.

Retrieval Augmented Generation (RAG)



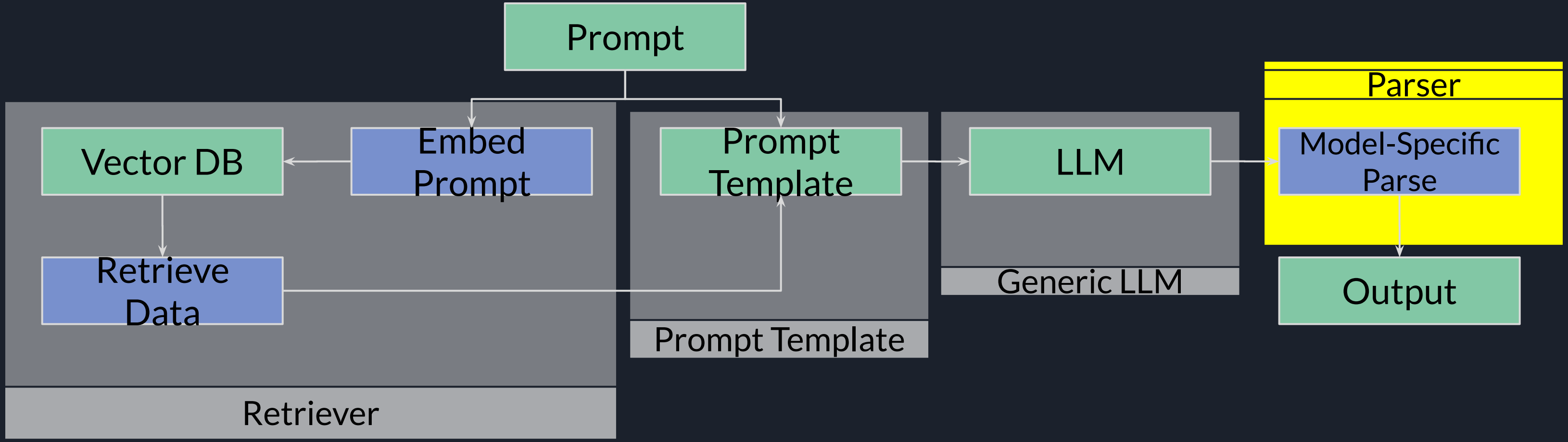
Prompt Template - Prompt normally has instructions for LLM and then has to organize the prompt and data for generation. Prompt template is a way to organize this information similar to how HTML templates are used for backend generation.

Retrieval Augmented Generation (RAG)



Generic LLM - Instead of having to deal with all the API details for a change in LLM. Langchain encapsulates these concerns away and usually there are pre-implemented modules for LLMs so they become plug and playable.

Retrieval Augmented Generation (RAG)



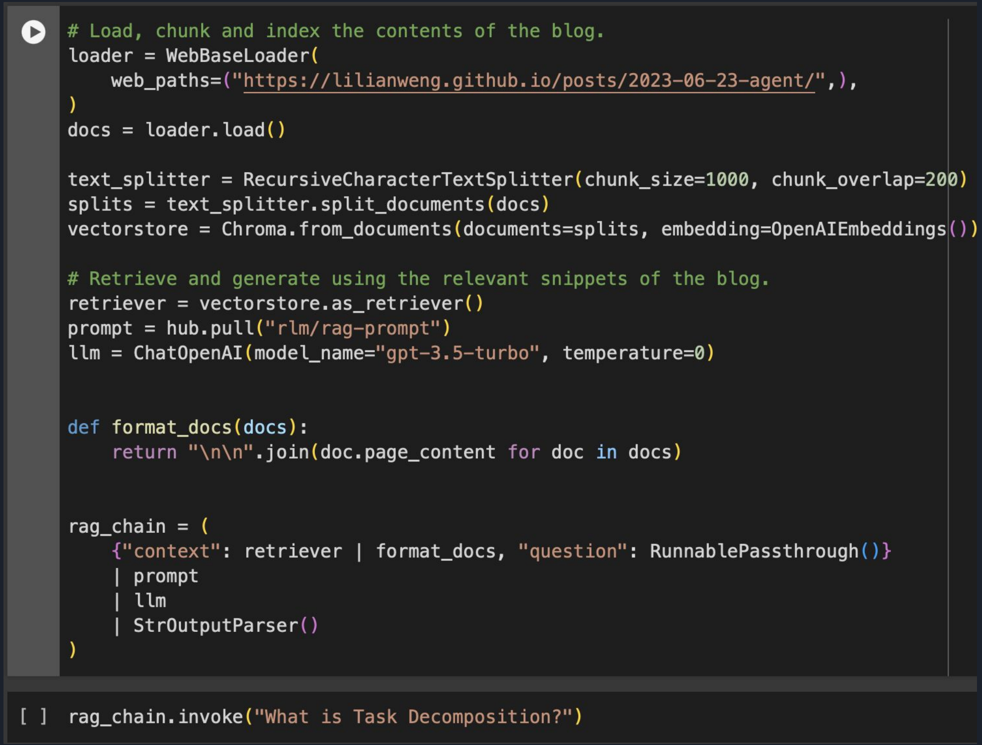
Parser - To handle output from a model normally there has to be a parsing strategy for the model. Langchain LLMs standardized so they can allow for the output to be parsed in a standardized way!

Why LangChain

- Pipeline View
    - Focus on implementation details and data flows
    - Standardized modules allows for easy swapping of models, data stores, prompt templates
        - Models from Internet or Machine Learning Engineers
        - Data Stores to change databases
        - Prompt Templates from ML/Gen-AI Engineers using Prompt Engineering
- Maintainability
    - Focused on business logic, code more understandable and easier to maintain/upgrade
    - Less likely to mess something up in connecting APIs together

Langchain RAG Demo Walkthrough

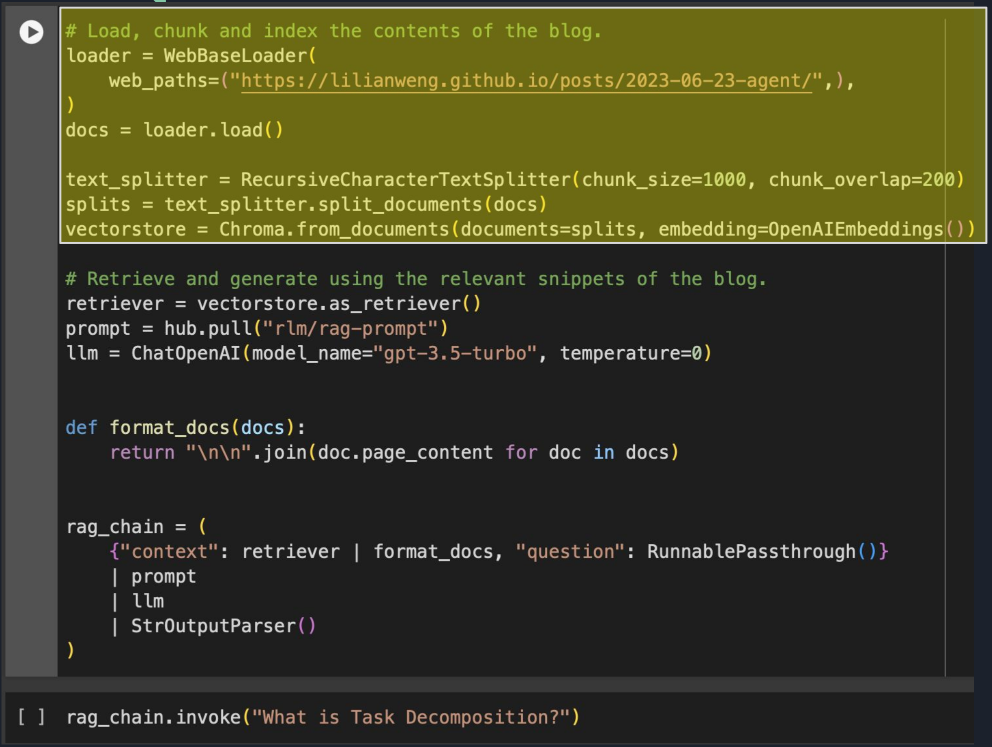
Demo Code



```python
# Load, chunk and index the contents of the blog.
loader = WebBaseLoader(
    web_paths=("https://lilianweng.github.io/posts/2023-06-23-agent/",),
)
docs = loader.load()

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(docs)
vectorstore = Chroma.from_documents(documents=splits, embedding=OpenAIEmbeddings())

# Retrieve and generate using the relevant snippets of the blog.
retriever = vectorstore.as_retriever()
prompt = hub.pull("rlm/rag-prompt")
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)


def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)


rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

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

Vector DB Population

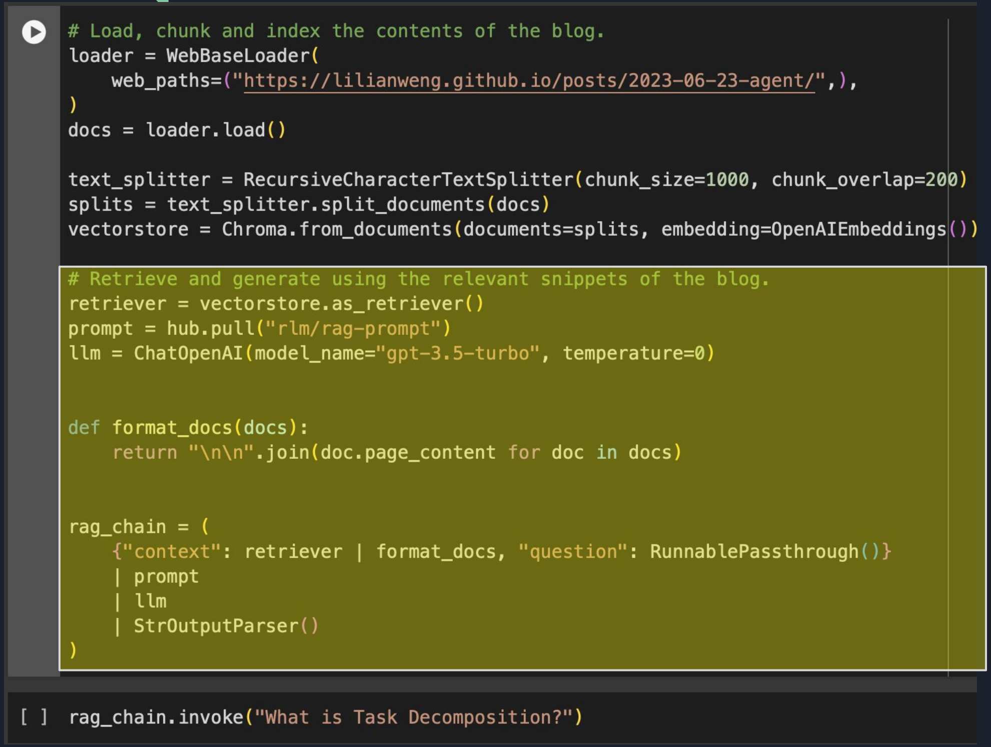


```
# Load, chunk and index the contents of the blog.
loader = WebBaseLoader(
    web_paths=("https://lilianweng.github.io/posts/2023-06-23-agent/",),
)
docs = loader.load()

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(docs)
vectorstore = Chroma.from_documents(documents=splits, embedding=OpenAIEmbeddings())

# Retrieve and generate using the relevant snippets of the blog.
retriever = vectorstore.as_retriever()
prompt = hub.pull("rlm/rag-prompt")
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)


def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)


rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

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

- Populate Knowledge Base (Vector DB)
- Embed chunks of text and store
    - Embedding?
    - ML Method to capture meaning of text as a vector.
    - Usually a good approximation of the meaning of the text.
- Vector DB Limitations?
    - Vector DB stores vectors and metadata
    - Vector is whatever can be embedded, OpenAI embedding vector space supports text, and image allowing for cross comparison
    - DB storage is very large can store from as many images/documents as desired

Pipeline

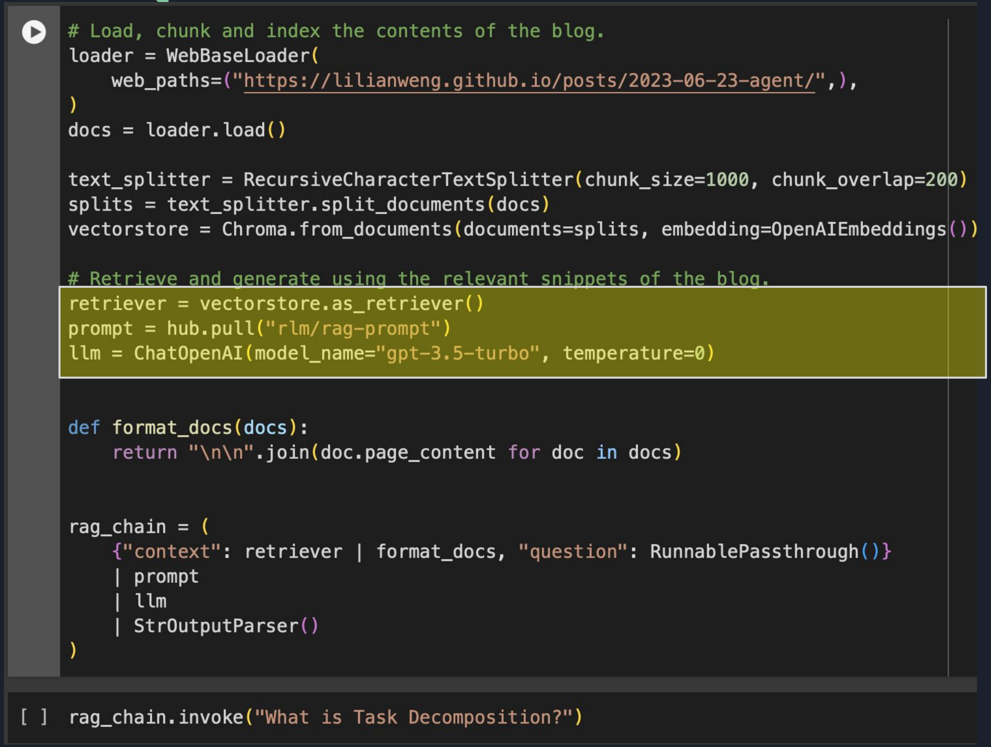


```
# Load, chunk and index the contents of the blog.
loader = WebBaseLoader(
    web_paths=("https://lilianweng.github.io/posts/2023-06-23-agent/",),
)
docs = loader.load()

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(docs)
vectorstore = Chroma.from_documents(documents=splits, embedding=OpenAIEmbeddings())

# Retrieve and generate using the relevant snippets of the blog.
retriever = vectorstore.as_retriever()
prompt = hub.pull("rlm/rag-prompt")
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)


def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)


rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

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

- Sets up Pipeline that will be run for every invocation

Pipeline Components

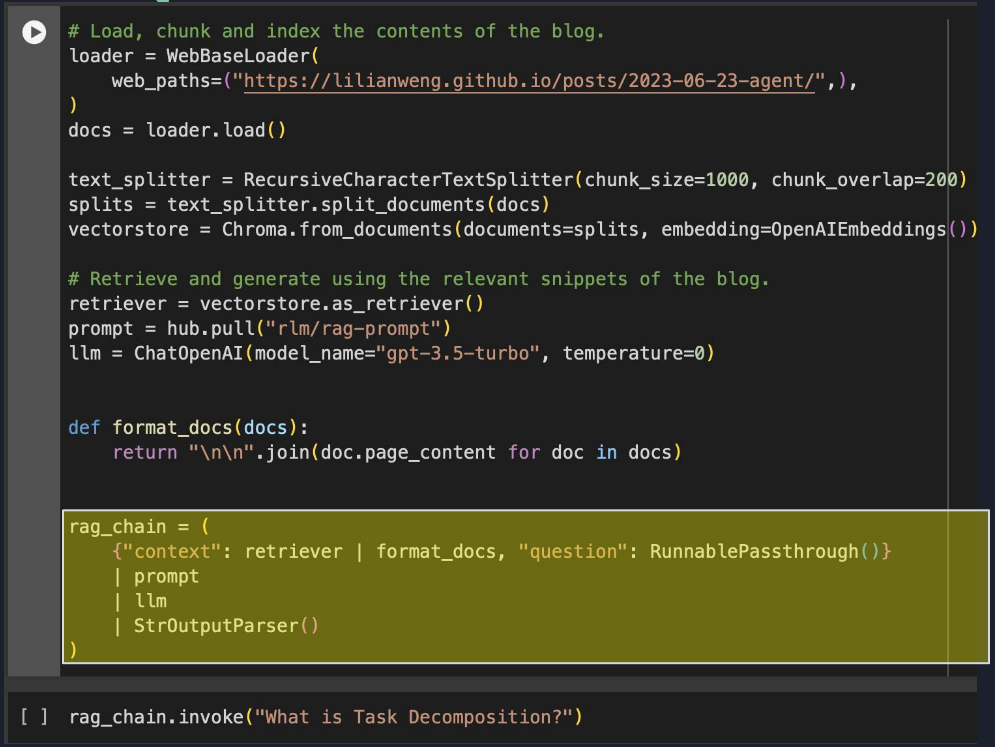


```python
# Load, chunk and index the contents of the blog.
loader = WebBaseLoader(
    web_paths=("https://lilianweng.github.io/posts/2023-06-23-agent/",),
)
docs = loader.load()

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(docs)
vectorstore = Chroma.from_documents(documents=splits, embedding=OpenAIEmbeddings())

# Retrieve and generate using the relevant snippets of the blog.
retriever = vectorstore.as_retriever()
prompt = hub.pull("rlm/rag-prompt")
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)


def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)


rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

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

- Retriever
  - Vector DB is a DB. Its role in this pipeline is as a retrieval source. So the object in the pipeline is a retriever.
- Prompt
  - Template being pulled from langchain hub. Prompt engineering already done by someone else!
- LLM
  - Using a Langchain wrapper for OpenAI Chat model. Changing LLMs is as easy as changing this line.

Pipeline

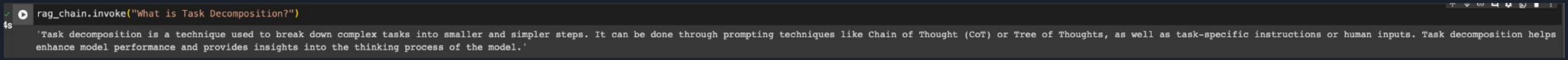


```
# Load, chunk and index the contents of the blog.
loader = WebBaseLoader(
    web_paths=("https://lilianweng.github.io/posts/2023-06-23-agent/",),
)
docs = loader.load()

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(docs)
vectorstore = Chroma.from_documents(documents=splits, embedding=OpenAIEmbeddings())

# Retrieve and generate using the relevant snippets of the blog.
retriever = vectorstore.as_retriever()
prompt = hub.pull("rlm/rag-prompt")
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)


def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)


rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

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

Chain

- Create a JSON object with context and question
    - Context is the result from the retriever passes through format_docs which uses double new lines to indicate new context snippet
    - Question is the prompt (hence passthrough)
- Prompt takes in previous JSON and puts into the imported template, has additional prompt engineering stuff configured like model instructions
- LLM takes prompt and generated output
- StrOutputParser generic parser to extract output from LLM ignoring other call meta-data



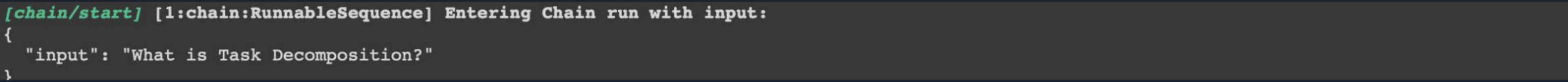
```
rag_chain.invoke("What is Task Decomposition?")
```

```
'Task decomposition is a technique used to break down complex tasks into smaller and simpler steps. It can be done through prompting techniques like Chain of Thought (CoT) or Tree of Thoughts, as well as task-specific instructions or human inputs. Task decomposition helps enhance model performance and provides insights into the thinking process of the model.'
```

End Result

Pipeline - Start

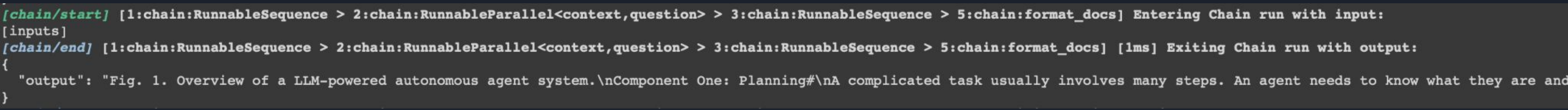
Chain receives prompt



```
[chain/start] [1:chain:RunnableSequence] Entering Chain run with input:
{
  "input": "What is Task Decomposition?"
}
```

Pipeline - Retrieval

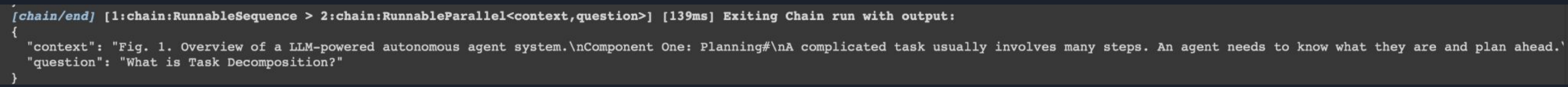
Chain gets context from vector database



```
[chain/start] [1:chain:RunnableSequence > 2:chain:RunnableParallel<context,question> > 3:chain:RunnableSequence > 5:chain:format_docs] Entering Chain run with input:
[inputs]
[chain/end] [1:chain:RunnableSequence > 2:chain:RunnableParallel<context,question> > 3:chain:RunnableSequence > 5:chain:format_docs] [1ms] Exiting Chain run with output:
{
  "output": "Fig. 1. Overview of a LLM-powered autonomous agent system.\nComponent One: Planning#\nA complicated task usually involves many steps. An agent needs to know what they are and
}
```

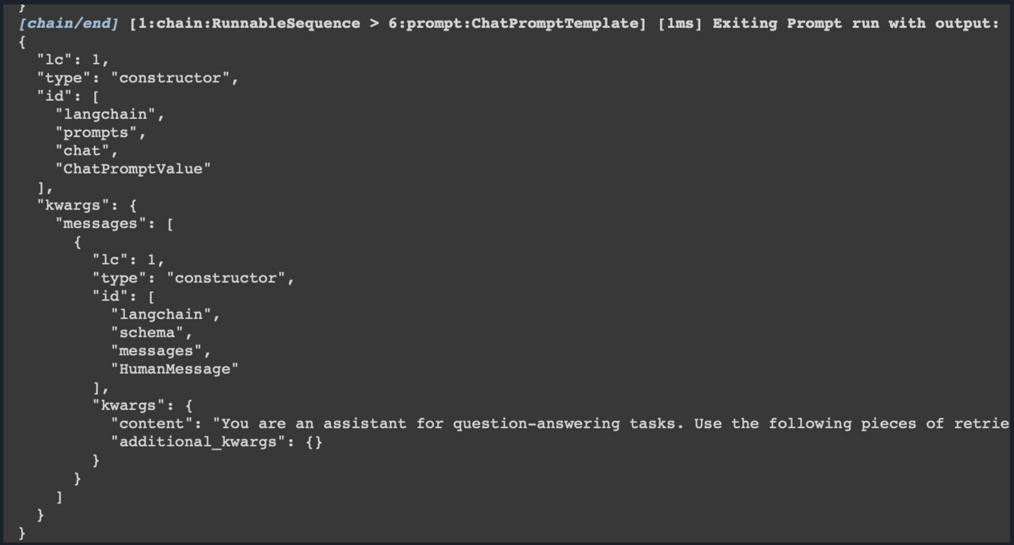
Pipeline - Augmentation (Prompt Template Preparation)

Chain adds context into prompt



```
[chain/end] [1:chain:RunnableSequence > 2:chain:RunnableParallel<context,question>] [139ms] Exiting Chain run with output:
{
  "context": "Fig. 1. Overview of a LLM-powered autonomous agent system.\nComponent One: Planning#\nA complicated task usually involves many steps. An agent needs to know what they are and plan ahead.'
  "question": "What is Task Decomposition?"
}
```

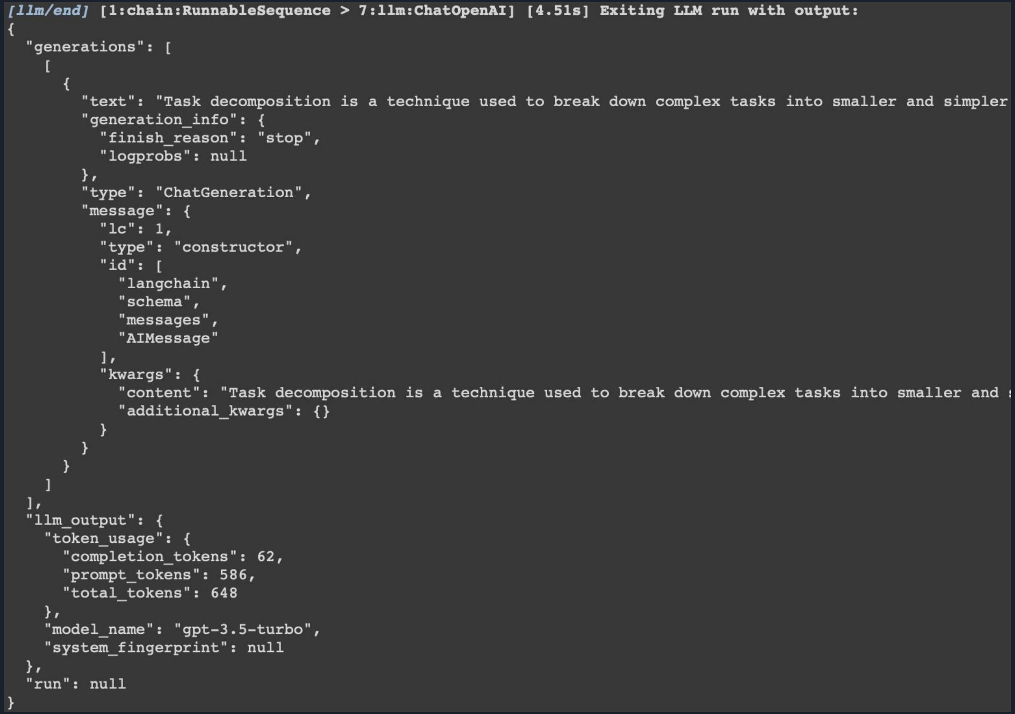
Pipeline - Augmentation (Prompt Generation)



```
}
[chain/end] [1:chain:RunnableSequence > 6:prompt:ChatPromptTemplate] [1ms] Exiting Prompt run with output:
{
  "lc": 1,
  "type": "constructor",
  "id": [
    "langchain",
    "prompts",
    "chat",
    "ChatPromptValue"
  ],
  "kwargs": {
    "messages": [
      {
        "lc": 1,
        "type": "constructor",
        "id": [
          "langchain",
          "schema",
          "messages",
          "HumanMessage"
        ],
        "kwargs": {
          "content": "You are an assistant for question-answering tasks. Use the following pieces of retrie
          "additional_kwargs": {}
        }
      }
    ]
  }
}
```

Chain generated prompt for LLM with user prompt and context

Pipeline - Generation

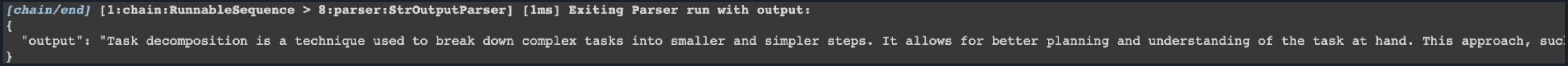


```
[llm/end] [1:chain:RunnableSequence > 7:llm:ChatOpenAI] [4.51s] Exiting LLM run with output:
{
  "generations": [
    [
      {
        "text": "Task decomposition is a technique used to break down complex tasks into smaller and simpler
        "generation_info": {
          "finish_reason": "stop",
          "logprobs": null
        },
        "type": "ChatGeneration",
        "message": {
          "lc": 1,
          "type": "constructor",
          "id": [
            "langchain",
            "schema",
            "messages",
            "AIMessage"
          ],
          "kwargs": {
            "content": "Task decomposition is a technique used to break down complex tasks into smaller and
            "additional_kwargs": {}
          }
        }
      }
    ]
  ],
  "llm_output": {
    "token_usage": {
      "completion_tokens": 62,
      "prompt_tokens": 586,
      "total_tokens": 648
    },
    "model_name": "gpt-3.5-turbo",
    "system_fingerprint": null
  },
  "run": null
}
```

Chain prompts LLM with generated prompt and receives

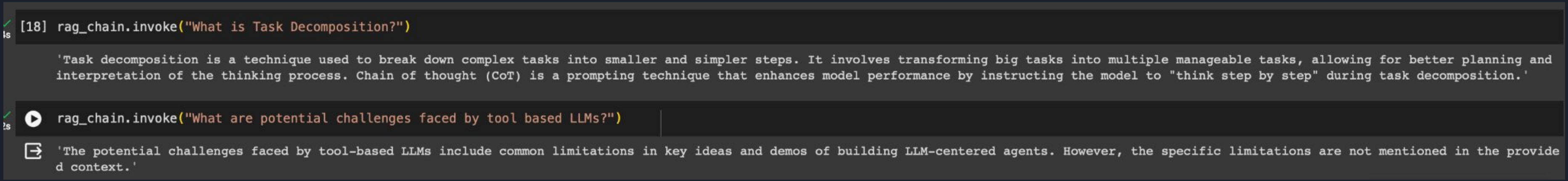
Pipeline - Parsing

Chain parses the output. As can be seen before the prompt template is designed to make the output generically parsable.



```
[chain/end] [1:chain:RunnableSequence > 8:parser:StrOutputParser] [1ms] Exiting Parser run with output:
{
  "output": "Task decomposition is a technique used to break down complex tasks into smaller and simpler steps. It allows for better planning and understanding of the task at hand. This approach, suc
}
```

I/O Without Debugging



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

'Task decomposition is a technique used to break down complex tasks into smaller and simpler steps. It involves transforming big tasks into multiple manageable tasks, allowing for better planning and
interpretation of the thinking process. Chain of thought (CoT) is a prompting technique that enhances model performance by instructing the model to "think step by step" during task decomposition.'

rag_chain.invoke("What are potential challenges faced by tool based LLMs?")

'The potential challenges faced by tool-based LLMs include common limitations in key ideas and demos of building LLM-centered agents. However, the specific limitations are not mentioned in the provide
d context.'
```