SEMANTIC SPOTTER - BUILD RAG SYSTEM

Submitted by: Shashi Khurana

Table of Contents

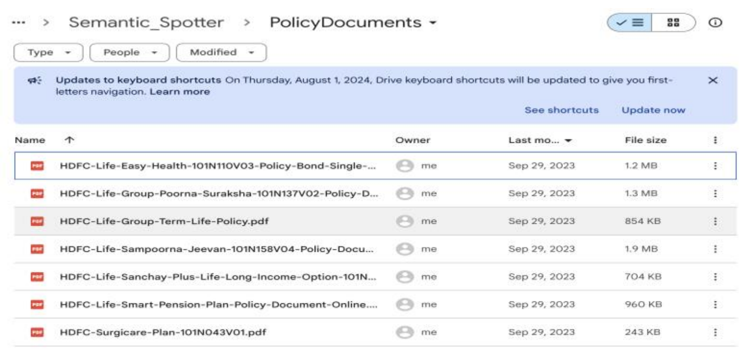
Building RAG System – Semantic Spotter

Objective

The goal of the project will be to build a robust generative search system capable of effectively and accurately answering questions from various policy documents. We shall be using LLamaindex to build the generative search application. We will discuss the overall system design and architecture in following section(s).

Problem statement

Here we are provided with private data that belongs to insurance domain, we are using documents from insurance domain(HDFC Policy screenshot below) , LLamaindex, (previously known as GPT Index), is a data framework specifically designed for LLM apps. Its primary focus is on ingesting, structuring, and accessing private or domain-specific data. It offers a set of tools that facilitate the integration of custom data into LLMs. Using LLamaindex we will be ingesting this domain specific data which is not common to LLM’s.



Based on my experience with LlamaIndex, it is an ideal solution if you're looking to work with vector embeddings. Using its many available plugins you could load (or ingest) data from many sources easily, and generate vector embeddings using an embedding model.
One key feature of LlamaIndex is that it is optimized for index querying. After the data is ingested, an index is created. This index represents your vectorized data and can be easily queried like so

query_engine = index.as_query_engine()
response = query_engine.query("GenAI is Awesome.")

LlamaIndex abstracts this but it is essentially taking your query "GenAI is Awesome." and comparing it with the most relevant information from your vectorized data (or index) which is then provided as context to the LLM.

What is LlamaIndex

LLMs offer a natural language interface between humans and data. LLMs come pre-trained on huge amounts of publicly available data, but they are not trained on your data. Our data may be private or specific to the problem we are trying to solve. It's behind APIs, in SQL databases, or trapped in PDFs and slide decks.

Context augmentation makes your data available to the LLM to solve the problem at hand. LlamaIndex provides the tools to build any of context-augmentation use case, from prototype to production. Our tools allow you to ingest, parse, index and process your data and quickly implement complex query workflows combining data access with LLM prompting.

The most popular example of context-augmentation is Retrieval-Augmented Generation or RAG, which combines context with LLMs at inference time.

LlamaIndex is the Data Framework for Context-Augmented LLM Apps

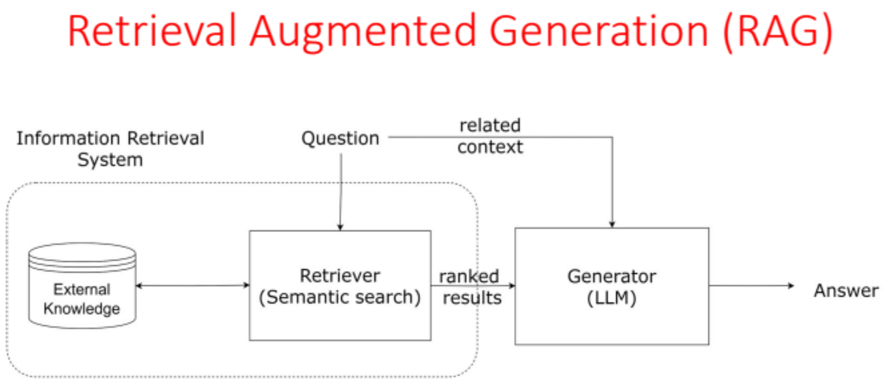
LlamaIndex imposes no restriction on how you use LLMs. You can use LLMs as auto-complete, chatbots, semi-autonomous agents, and more. It just makes using them easier.

Overall System Design

LlamaIndex framework consists of the following:

- **Data connectors** ingest your existing data from their native source and format. These could be APIs, PDFs, SQL, and (much) more.
- **Data indexes** structure your data in intermediate representations that are easy and performant for LLMs to consume.
- **Engines** provide natural language access to your data. For example:
  - Query engines are powerful interfaces for question-answering (e.g. a RAG pipeline).
  - Chat engines are conversational interfaces for multi-message, "back and forth" interactions with your data.
- **Agents** are LLM-powered knowledge workers augmented by tools, from simple helper functions to API integrations and more.
- **Observability/Evaluation integrations** that enable you to rigorously experiment, evaluate, and monitor your app in a virtuous cycle.

Architecture



Implementation Steps

Step 1 - Product Specifications / Solution Strategy

Solution Strategy - Build a solution which should solve the following requirements:

- Users would responses from set of Insurance documents
- If they want to refer to the original document from which the bot is responding, the bot should provide a citation as well.

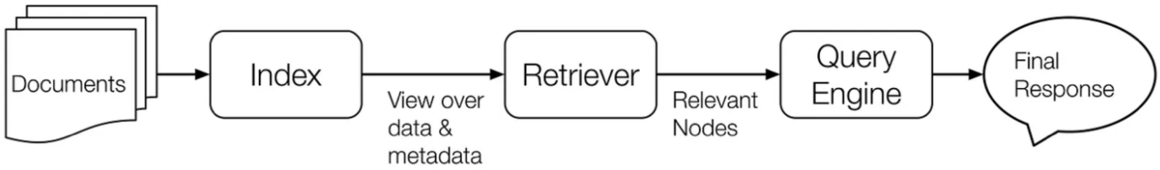
Goal - Solving the above two requirements well in the solution would ensure that the accuracy of the overall model is good and therefore further improvisations and customizations make sense.

Data Used - 5 Policy documents stored in a single folder

Tools used - LlamaIndex (only for now) has been used due to its powerful query engine, fast data processing using data loaders and directory readers as well as easier and faster implementation using fewer lines of code.

Step 2 - Solution approach

In this section, we go ahead and actually build solution that we proposed in the previous step.



Step 3 - Data Loading

2 ways to do this

- If you have a single file containing all the required data, use a data loader from LLamahub
- If you have multiple files, use Simple Directory Reader Just ensure that for reading each file type the necessary dependency libraries are already installed.
- In this step we have loaded 7 documents that belong to specific domain.

```
[14] os.chdir("/content/drive/My Drive/Colab Notebooks/Semantic_Spotter/PolicyDocuments")
     !ls

 HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
 HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf
 HDFC-Life-Group-Term-Life-Policy.pdf
'HDFC-Life-Sampoorna-Jeevan-101N158V04-Policy-Document (1).pdf'
 HDFC-Life-Sanchay-Plus-Life-Long-Income-Option-101N134V19-Policy-Document.pdf
 HDFC-Life-Smart-Pension-Plan-Policy-Document-Online.pdf
 HDFC-Surgicare-Plan-101N043V01.pdf
```

```
[15] from llama_index.core import SimpleDirectoryReader
     reader = SimpleDirectoryReader(input_dir="/content/drive/My Drive/Colab Notebooks/Semantic_Spotter/PolicyDocuments/")
```

Step 4 - Building the query engine

- In this step we have parsed the loaded 7 documents in previous step, Build the index and exposed that index for query engine

Step 4 - Building the query engine

```
[20] from llama_index.core.node_parser import SimpleNodeParser
     from llama_index.core import VectorStoreIndex
     from IPython.display import display, HTML

     # create parser and parse document into nodes
     parser = SimpleNodeParser.from_defaults()
     nodes = parser.get_nodes_from_documents(documents)
     # documents --> nodes

     # # build index
     index = VectorStoreIndex(documents)
     #nodes --> index

     # Construct Query Engine
     query_engine = index.as_query_engine()

[21] parser

SentenceSplitter(include_metadata=True, include_prev_next_rel=True, callback_manager=<llama_index.core.callbacks.base.CallbackManager object at 0x7b453338fee0>, id_func=<function default_id_func at 0x7b453aeb9090>, chunk_size=1024, chunk_overlap=200, separator=' ',
paragraph_separator='\n\n\n', secondary_chunking_regex='[^,.;。？！]+[,.;。？！]?')
```

Step 5 - Checking response and response parameters

- In this step we are passing the query to our query engine and checking what response we get, it appears it is giving accurate response as listed below.

Step 5 - Checking response and response parameters

```
[22] response = query_engine.query("What is this document talking about?")

[23] #Checking the response
     response.response

'This document is discussing the HDFC Life Sampoorna Jeevan policy.'
```

Step 6 - Creating a response Pipeline

User receives the response and the document that they can refer to

Step 6 - Creating a response Pipeline

User receives the response and the document that they can refer to

```python
## Query response function
def query_response(user_input):
  response = query_engine.query(user_input)
  file_name = response.source_nodes[0].node.metadata['file_name']
  final_response = response.response + '\n Check further at ' + file_name + ' document'
  return final_response
```

```python
[28] def initialize_conv():
  print('Feel free to ask Questions regarding Insurance Policy documents. Press exit once you are done')
  while True:
    user_input = input()
    if user_input.lower() == 'exit':
      print('Exiting the program... bye')
      break
    else:
      response = query_response(user_input)
      display(HTML(f'<p style="font-size:20px">{response}</p>'))
```

Below is the conversation, asking multiple queries and getting appropriate responses.

what is the eligibility criteria for Accidental claims

The eligibility criteria for Accidental claims include death by or due to a bodily injury caused by an Accident, independent of all other causes of death, and must occur within 180 days of any bodily injury. Check further at HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf document

what will be the premium for term plan policy

The premium for a term plan policy will be determined based on factors such as the age of the life assured, the chosen sum assured, the policy term, payment method, and payment frequency as specified in the policy schedule. Check further at HDFC-Surgicare-Plan-101N043V01.pdf document

what is the max age limit to avail the policy

The maximum age limit to avail the policy is determined by the Maximum Maturity Age specified in the document. Check further at HDFC-Life-Sampoorna-Jeevan-101N158V04-Policy-Document (1).pdf document

```
testing_pipeline(questions)
```

```
What is this document all about?
This document serves as the Master Policy for a Non-Linked Non-Participating Group Term Insurance Policy issued by HDFC Life Insurance Company Limited. It outlines the contractual agreement between the in
 Check further at HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf document

 Please provide your feedback on the response provided by the bot
Good
What are the various plans offered?
The various plans offered include Lump Sum Option, Income Option, Lump Sum with Income Option, and Income with Lump Sum Option. Each plan has different components such as Basic Sum Assured, Applicable Bo
 Check further at HDFC-Life-Smart-Pension-Plan-Policy-Document-Online.pdf document

 Please provide your feedback on the response provided by the bot
Good
What is the premium and eligibility criteria?
The premium must be paid promptly and regularly during the Premium Payment Term as mentioned in the Policy Schedule. In case of monthly premium payment mode, three months premiums are collected in advance
 Check further at HDFC-Life-Sampoorna-Jeevan-101N158V04-Policy-Document (1).pdf document

 Please provide your feedback on the response provided by the bot
Good
What documents need to produce for insurance policy?
Completed claim form, Member Information/Enrollment Form (MIF), Separate Member Authorization (if applicable), all medical reports for diagnosis and treatment of critical illness, all current and past me
 Check further at HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf document

 Please provide your feedback on the response provided by the bot
Good
```

1 to 4 of 4 entries Filter

index	Question	Response	Good or Bad
0	What is this document all about?	This document is the Master Policy Document for a Non-Linked Non-Participating Group Term Insurance Policy issued by HDFC Life Insurance Company Limited. It outlines the contractual agreement between the insurer and the Master Policyholder for the benefit of Scheme Members or their nominees, detailing the terms and conditions of the policy and the benefits payable under it. Check further at HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf document	Good
1	What are the various plans offered?	The various plans offered include Lump Sum Option, Income Option, Lump Sum with Income Option, and Income with Lump Sum Option. Each plan has different components such as Basic Sum Assured, Applicable Bonus, Applicable Terminal Bonus, Guaranteed Maturity Benefit, Guaranteed Income Benefit, and specific payout structures based on the chosen options. Check further at HDFC-Life-Smart-Pension-Plan-Policy-Document-Online.pdf document	Good
2	What is the premium and eligibility criteria?	The premium must be paid promptly and regularly during the Premium Payment Term as mentioned in the Policy Schedule. In case of monthly premium payment mode, three months premiums are collected in advance. Eligibility criteria include being 'at Active Service' or 'an Active Member' as specified in the Policy, and providing the necessary information and documentation as required by the Company. Check further at HDFC-Life-Sampoorna-Jeevan-101N158V04-Policy-Document (1).pdf document	Good
3	What documents need to produce for insurance policy?	Completed claim form, Member Information/Enrollment Form (MIF), Separate Member Authorization (if applicable), all medical reports for diagnosis and treatment of critical illness, all current and past medical records of Scheme Member, NEFT details along with supporting documents of insured member, translation of all vernacular documents if required. Check further at HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf document	Good

Show 25 per page

Step 7 - Build a Testing Pipeline

Here we feed a series of questions to the Q/A bot and store the responses along with the feedback on whether it's accurate or not from the user

```
[30] questions = ['What is this document all about?', "What are the various plans offered?",
                'What is the premium and eligibility criteria?',  'What documents need to produce for insurance policy?']
```

```
[31] def testing_pipeline(questions):
       test_feedback  = []
       for i in questions:
         print(i)
         print(query_response(i))
         print('\n Please provide your feedback on the response provided by the bot')
         user_input = input()
         test_feedback.append((i,query_response(i),user_input))
       feedback_df = pd.DataFrame(test_feedback, columns =['Question', 'Response', 'Good or Bad'])
       return feedback_df
```

```
[32] import pandas as pd
```

```
[33] testing_pipeline(questions)
```

```
What is this document all about?
This document serves as the Master Policy for a Non-Linked Non-Participating Group Term Insurance Policy issued by HDFC Life Insurance Company Limited. It outlines the contractual agreement between the insurer and the Master Policyholder for the benefit of Scheme Memb
 Check further at HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf document

 Please provide your feedback on the response provided by the bot
Good
What are the various plans offered?
The various plans offered include Lump Sum Option, Income Option, Lump Sum with Income Option, and Income with Lump Sum Option. Each plan has different components such as Basic Sum Assured, Applicable Bonus, Applicable Terminal Bonus, Guaranteed Maturity Benefit, Guar
 Check further at HDFC-Life-Smart-Pension-Plan-Policy-Document-Online.pdf document

 Please provide your feedback on the response provided by the bot
Good
What is the premium and eligibility criteria?
The premium must be paid promptly and regularly during the Premium Payment Term as mentioned in the Policy Schedule. In case of monthly premium payment mode, three months premiums are collected in advance. Eligibility criteria include being 'at Active Service' or 'an
 Check further at HDFC-Life-Sampoorna-Jeevan-101N158V04-Policy-Document (1).pdf document

 Please provide your feedback on the response provided by the bot
Good
What documents need to produce for insurance policy?
Completed claim form, Member Information/Enrollment Form (MIF), Separate Member Authorization (if applicable), all medical reports for diagnosis and treatment of critical illness, all current and past medical records of Scheme Member, NEFT details along with supportir
 Check further at HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf document

 Please provide your feedback on the response provided by the bot
Good
```

1 to 4 of 4 entries Filter

index	Question	Response	Good or B
0	What is this document all about?	This document is the Master Policy Document for a Non-Linked Non-Participating Group Term Insurance Policy issued by HDFC Life Insurance Company Limited. It outlines the contractual agreement between the insurer and the Master Policyholder for the benefit of Scheme Members or their nominees, detailing the terms and conditions of the policy and the benefits payable under it. Check further at HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf document	Good
1	What are the various plans offered?	The various plans offered include Lump Sum Option, Income Option, Lump Sum with Income Option, and Income with Lump Sum Option. Each plan has different components such as Basic Sum Assured, Applicable Bonus, Applicable Terminal Bonus, Guaranteed Maturity Benefit, Guaranteed Income Benefit, and specific payout structures based on the chosen options. Check further at HDFC-Life-Smart-Pension-Plan-Policy-Document-Online.pdf document	Good
2	What is the premium and eligibility criteria?	The premium must be paid promptly and regularly during the Premium Payment Term as mentioned in the Policy Schedule. In case of monthly premium payment mode, three months premiums are collected in advance. Eligibility criteria include being 'at Active Service' or 'an Active Member' as specified in the Policy, and providing the necessary information and documentation as required by the Company. Check further at HDFC-Life-Sampoorna-Jeevan-101N158V04-Policy-Document (1).pdf document	Good
3	What documents need to produce for insurance policy?	Completed claim form, Member Information/Enrollment Form (MIF), Separate Member Authorization (if applicable), all medical reports for diagnosis and treatment of critical illness, all current and past medical records of Scheme Member, NEFT details along with supporting documents of insured member, translation of all vernacular documents if required. Check further at HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf document	Good

Step 8 - Prompt Setup

Below, we take the default prompts and customize them to always answer, even if the context is not helpful.

In this step, we have added the prompt template as wrapper and passing the context and query string in form of question answer template to LLM.

```
[34] from llama_index.core import PromptTemplate

     text_qa_template_str = (
         "Context information is"
         " below.\n---------------------\n{context_str}\n---------------------\nUsing"
         " both the context information and also using your own knowledge, answer"
         " the question: {query_str}\nIf the context isn't helpful, you can also"
         " answer the question on your own.\n"
     )
     text_qa_template = PromptTemplate(text_qa_template_str)

     refine_template_str = (
         "The original question is as follows: {query_str}\nWe have provided an"
         " existing answer: {existing_answer}\nWe have the opportunity to refine"
         " the existing answer (only if needed) with some more context"
         " below.\n------------\n{context_msg}\n------------\nUsing both the new"
         " context and your own knowledge, update or repeat the existing answer.\n"
     )
     refine_template = PromptTemplate(refine_template_str)
```

```
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader
from llama_index.llms.openai import OpenAI

llm = OpenAI(model="gpt-3.5-turbo")

documents = SimpleDirectoryReader("/content/drive/My Drive/Colab Notebooks/Semantic_Spotter/PolicyDocuments/").load_data()

index = VectorStoreIndex.from_documents(documents)
```

```
print(
    index.as_query_engine(
        text_qa_template=text_qa_template,
        refine_template=refine_template,
        llm=llm,
    ).query("how is the premium for term plan evaluated?")
)
```

```
The premium for a term plan is evaluated based on factors such as the sum assured, the age and gender of the insured, the term of the policy, the payment method, and the payment frequency. Additionally, i
```

Appendix A

Alternative way to implement RAG based solution is through LangChain

LangChain

LangChain is a framework that simplifies the development of LLM applications LangChain offers a suite of tools, components, and interfaces that simplify the construction of LLM-centric applications. LangChain enables developers to build applications that can generate creative and contextually relevant content LangChain provides an LLM class designed for interfacing with various language model providers, such as OpenAI, Cohere, and Hugging Face.

LangChain's versatility and flexibility enable seamless integration with various data sources, making it a comprehensive solution for creating advanced language model-powered applications.

LangChain's open-source framework is available to build applications in Python or JavaScript/TypeScript. Its core design principle is composition and modularity. By combining modules and components, one can quickly build complex LLM-based applications. LangChain is an open-source framework that makes it easier to build powerful and personalize able applications with LLMs relevant to user's interests and needs. It connects to external systems to access information required to solve complex problems. It provides abstractions for most of the functionalities needed for building an LLM application and has integrations that can readily read and write data, reducing the development speed of the application.

LangChain's framework allows for building applications that are agnostic to the underlying language model. With its ever-expanding support for various LLMs, LangChain offers a unique value proposition to build applications and iterate continuously.

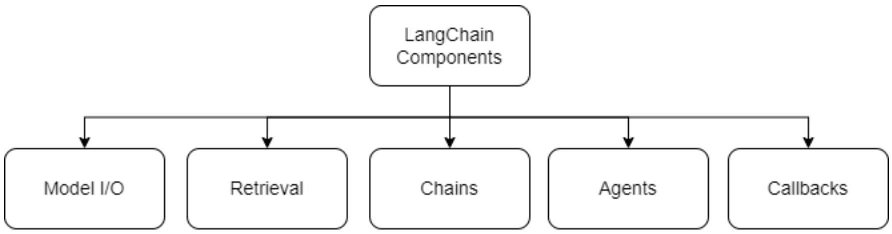
LangChain framework consists of the following:

Components: LangChain provides modular abstractions for the components necessary to work with language models. LangChain also has collections of implementations for all these abstractions. The components are designed to be easy to use, regardless of whether you are using the rest of the LangChain framework or not.

Use-Case Specific Chains: Chains can be thought of as assembling these components in particular ways to best accomplish a particular use case. These are intended to be a higher level interface through which people can easily get started with a specific use case. These chains are also designed to be customizable.

The LangChain framework revolves around the following building blocks:

- Model I/O: Interface with language models (LLMs & Chat Models, Prompts, Output Parsers)
- Retrieval: Interface with application-specific data (Document loaders, Document transformers, Text embedding models, Vector stores, Retrievers)
- Chains: Construct sequences/chains of LLM calls
- Memory: Persist application state between runs of a chain
- Agents: Let chains choose which tools to use given high-level directives
- Callbacks: Log and stream intermediate steps of any chain

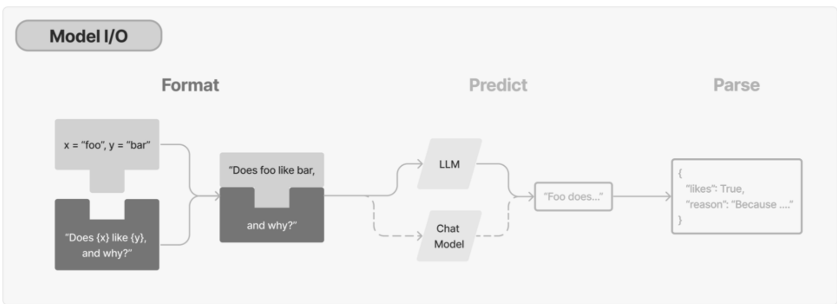


Model I/O

LangChain's Model I/O component provides support to interface with the LLM and generate responses. The Model I/O consists of

- Prompts: Templatize, dynamically select, and manage model inputs
- Language Models: Make calls to language models through common interfaces
- Output Parsers: Extract information from model outputs

  The general flow of Model I/O in LangChain is illustrated in the image below



Model

LangChain provides an easy out-of-the box support to work with LLMs. LangChain provides interfaces and integrations for two classes of LLM models

- LLMs: Models that take a text string as input and return a text string
- Chat models: Models that are backed by a language model but take a list of Chat Messages as input and return a Chat Message.

LLMs and chat models are subtly but importantly different. LLMs in LangChain refer to pure text completion models - where a string prompt is taken as the input and the LLM outputs a string.

Chat Models are LLMs that have been tuned specifically for having turn-based conversations such as ChatGPT. Instead of a single string, they take a list of chat messages as input. Usually these models have labelled messages such as "System", "Human" and provides a AI chat message ("AI"/ "Output Response") as the output.

LLM

The LLM class of LangChain is designed to provide a standard interface for all the major LLM provides such as OpenAI, Cohere, Hugging Face, etc. LangChain provided a standard interface for interacting with many different LLMs to perform standard text completion tasks.

This, however, has been deprecated and no longer is supported by LangChain. The text completion model is now categorised as legacy by OpenAI.

Chat Model

Chat models are a variation on language models. While chat models use language models under the hood, the interface they use is a bit different. Rather than using a "text in, text out" API, they use an interface where "chat messages" are the inputs and outputs.

Retrievers

Retrievers provide Easy way to combine documents with language models.

A retriever is an interface that returns documents given an unstructured query. It is more general than a vector store. A retriever does not need to be able to store documents, only to return (or retrieve) them. Retriever stores data for it to be queried by a language model. It provides an interface that will return documents based on an unstructured query. Vector stores can be used as the backbone of a retriever, but there are other types of retrievers as well. There are many different types of retrievers, the most widely supported is the VectoreStoreRetriever.

Chains

Using an LLM in isolation is fine for simple applications, but more complex applications require chaining LLMs - either with each other or with other components.

LangChain provides Chains that can be used to combine multiple components together to create a single, coherent application.

For example, we can create a chain that takes user input, formats it with a PromptTemplate, and then passes the formatted response to an LLM. We can build more complex chains by combining multiple chains together, or by combining chains with other components.

The fundamental unit of Chains is a LLMChain object which takes an input and provides an output.

Agents

The core idea of agents is to use a language model to choose a sequence of actions to take. In chains, a sequence of actions is hardcoded (in code). In agents, a language model is used as a reasoning engine to determine which actions to take and in which order.