Setup

```
# install required packages
!pip install -q openai langchain chromadb faiss-cpu pypdf tiktoken docarray
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.3/67.3 kB 3.4 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 19.0/19.0 MB 87.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 94.9/94.9 kB 6.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 31.3/31.3 MB 62.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 303.4/303.4 kB 22.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.8/302.8 kB 19.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 284.2/284.2 kB 19.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 55.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.6/101.6 kB 9.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 83.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.8/65.8 kB 4.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.9/55.9 kB 4.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.9/194.9 kB 14.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 119.0/119.0 kB 8.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 96.7/96.7 kB 8.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 4.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 459.8/459.8 kB 33.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.5/71.5 kB 4.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.0/4.0 MB 101.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 454.8/454.8 kB 29.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.0/46.0 kB 3.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 86.8/86.8 kB 5.5 MB/s eta 0:00:00
  Building wheel for pypika (pyproject.toml) ... done
```

```
# install the necessary library for working with pdf
!pip install PyPDF
```

```
Collecting PyPDF
  Downloading pypdf-5.5.0-py3-none-any.whl.metadata (7.2 kB)
Downloading pypdf-5.5.0-py3-none-any.whl (303 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 303.4/303.4 kB 4.6 MB/s eta 0:00:00
Installing collected packages: PyPDF
Successfully installed PyPDF-5.5.0
```

```
# Installing tiktoken library for working with OpenAI's embedding model
!pip install tiktoken
```

```
Requirement already satisfied: tiktoken in /usr/local/lib/python3.11/dist-packages (0.9.0)
Requirement already satisfied: regex>=2022.1.18 in /usr/local/lib/python3.11/dist-packages (from tiktoken) (2024.
Requirement already satisfied: requests>=2.26.0 in /usr/local/lib/python3.11/dist-packages (from tiktoken) (2.32.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26.0->ti
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26
```

```
# install the ChatOpenAI model
!pip install -qU langchain-openai
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.4/63.4 kB 1.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 438.4/438.4 kB 8.1 MB/s eta 0:00:00
```

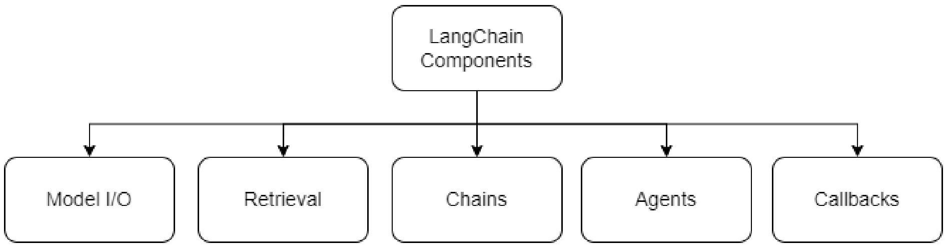
Set the OpenAI Key

```
# import the necessary libraies
import os
import openai

folder_path = './'
os.chdir(folder_path)

# Read the text file containing the API key
with open(folder_path + "OpenAI_API_Key.txt", "r") as f:
  openai.api_key = ' '.join(f.readlines())

# Update the OpenAI API key by updating the environment variable
os.environ["OPENAI_API_KEY"] = openai.api_key
```

LangChain

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

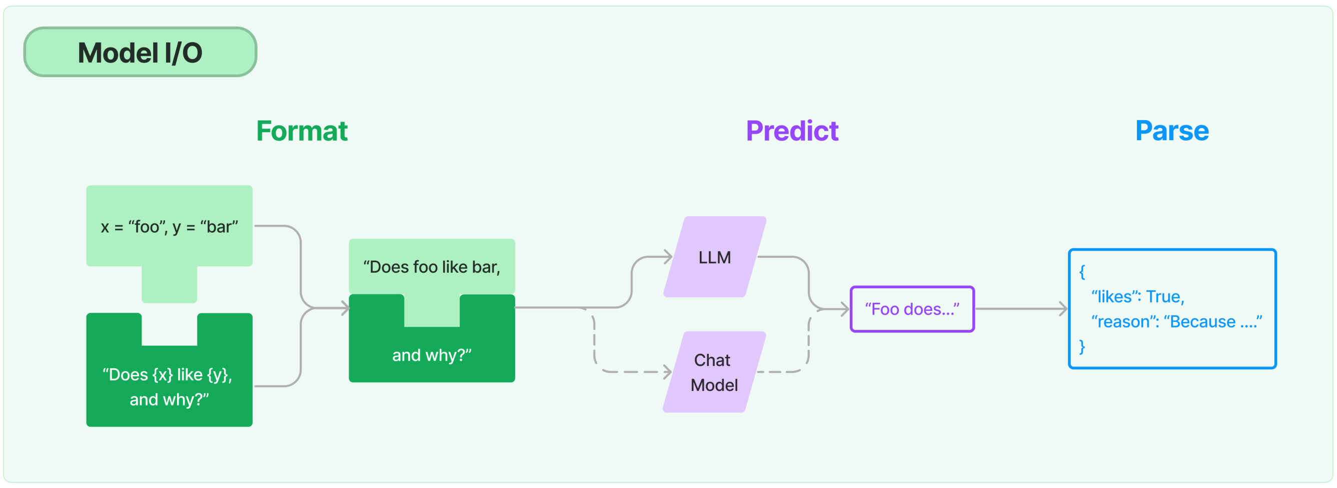


1. Model I/O

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

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

The general flow of Model I/O in LangChain is illustrated in the image below (source).



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.

LLMs

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 hence for the remainder of the course, we will work with OpenAI's chat model.

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.

The OpenAI chat model can be imported using the code below:

```
pip install -qU langchain-openai
from langchain_openai import ChatOpenAI
```

There are two ways to initiate the OpenAI LLM class once the necessary libraries have been imported. If you'd prefer not to set an environment variable you can pass the key in directly via the openai_api_key named parameter when initiating the OpenAI LLM class:

```
chat = ChatOpenAI(openai_api_key="...")
```

Otherwise you can initialize without any params:

```
from langchain_openai import ChatOpenAI
chat = ChatOpenAI()
```

```
# import required libraries
from langchain_openai import ChatOpenAI, OpenAI

# instantiate OpenAI's Chat Model
llm_chat = ChatOpenAI()
```

The ChatOpenAI() can take multiple arguments. The API reference contains the complete list of arguments that can be passed to the chat model. A few important ones include:

- max_tokens
- model_name: If not defined, the default model is `gpt-3.5-turbo`
- max_retries: If not defined, the default value is 6
- temperature: If not defined, the default value `temperature = 0.7`

2. Data Connections and Retrieval

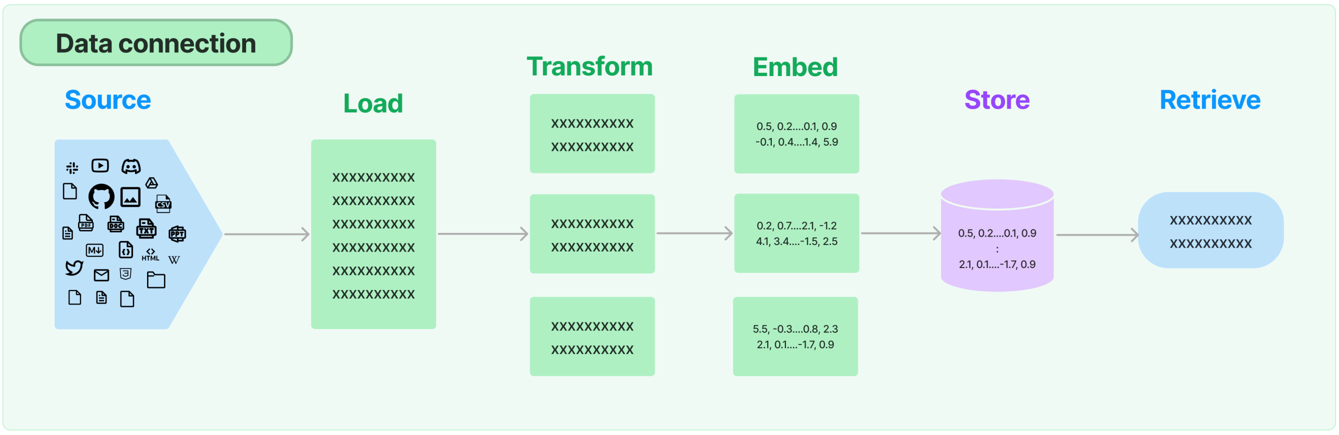
In addition to making API calls easier, LangChain also provides various methods to work with external documents efficiently.

Many LLM applications require user-specific data that is not part of the model's training set. The primary way of accomplishing this is through Retrieval Augmented Generation (RAG). In this process, external data is retrieved and then passed to the LLM when doing the generation step.

LangChain provides all the building blocks for RAG applications - from simple to complex. This section of the documentation covers everything related to the retrieval step - e.g. the fetching of the data. Although this sounds simple, it can be subtly complex. This encompasses several key modules.

The following methods provided by LangChain help process documents efficiently:

- Document Loaders
- Text Splitters
- Vector Stores
- Retrievers



Document Loaders

Document loaders provide an easy method to import data from different sources or formats as a DOcument, which contains the text content and the associated metadata.

Document Loaders load documents from different sources like HTML, PDF, text, etc., from various locations like cloud storage buckets and public websites. LangChain provides over 100 different document loaders as well as integrations with other major providers in the space, like AirByte and Unstructured. Refer to the official documentation for the complete list of supported document loaders in the API reference and the official documentation.

It should be noted that some document loaders require the associated libraries to be installed.

PDF Documents

Langchain can load and parse PDF documents using various pdf connectors. We will see the PyPDFLoader to load documents into an array of documents. The document loader requires the python package of `pypdf` to be installed.

Each document contains the page content and metadata with the associated page number.

An added advantage of using PyPDFLoader is that the documents can be retrieved with page numbers.

```
pip install -U langchain-community
```

```
Collecting dataclasses-json<0.7,>=0.5.7 (from langchain-community)
  Downloading dataclasses_json-0.6.7-py3-none-any.whl.metadata (25 kB)
Collecting pydantic-settings<3.0.0,>=2.4.0 (from langchain-community)
  Downloading pydantic_settings-2.9.1-py3-none-any.whl.metadata (3.8 kB)
Requirement already satisfied: langsmith<0.4,>=0.1.125 in /usr/local/lib/python3.11/dist-packages (from langchain
Collecting httpx-sse<1.0.0,>=0.4.0 (from langchain-community)
  Downloading httpx_sse-0.4.0-py3-none-any.whl.metadata (9.0 kB)





  Downloading marshmallow-3.26.1-py3-none-any.whl.metadata (7.3 kB)
  Downloading marshmallow-3.26.1-py3-none-any.whl.metadata (7.3 kB)
Collecting typing-inspect<1,>=0.4.0 (from dataclasses-json<0.7,>=0.5.7->langchain-community)
  Downloading typing_inspect-0.9.0-py3-none-any.whl.metadata (1.5 kB)
Requirement already satisfied: langchain-text-splitters<1.0.0,>=0.3.8 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: pydantic<3.0.0,>=2.7.4 in /usr/local/lib/python3.11/dist-packages (from langchain<
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.11/dist-packages (from langchain-co
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.11/dist-packages (from langchain-cor
Requirement already satisfied: typing-extensions>=4.7 in /usr/local/lib/python3.11/dist-packages (from langchain-
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=
Requirement already satisfied: orjson<4.0.0,>=3.9.14 in /usr/local/lib/python3.11/dist-packages (from langsmith<0
Requirement already satisfied: requests-toolbelt<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from l
Requirement already satisfied: zstandard<0.24.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from langsmi
Collecting python-dotenv>=0.21.0 (from pydantic-settings<3.0.0,>=2.4.0->langchain-community)
  Downloading python_dotenv-1.1.0-py3-none-any.whl.metadata (24 kB)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2->lang
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.11/dist-packages (from SQLAlchemy<3,>=1.4->l
Requirement already satisfied: anyio in /usr/local/lib/python3.11/dist-packages (from httpx<1,>=0.23.0->langsmith
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx<1,>=0.23.0->l
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx<1,
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.11/dist-packages (from jsonpatch<2.0,>=
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<3
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<3.
Collecting mypy-extensions>=0.3.0 (from typing-inspect<1,>=0.4.0->dataclasses-json<0.7,>=0.5.7->langchain-communi
  Downloading mypy_extensions-1.1.0-py3-none-any.whl.metadata (1.1 kB)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio->httpx<1,>=0.2
Downloading langchain_community-0.3.24-py3-none-any.whl (2.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 23.6 MB/s eta 0:00:00
Downloading dataclasses_json-0.6.7-py3-none-any.whl (28 kB)
Downloading httpx_sse-0.4.0-py3-none-any.whl (7.8 kB)
Downloading pydantic_settings-2.9.1-py3-none-any.whl (44 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.4/44.4 kB 3.7 MB/s eta 0:00:00
Downloading marshmallow-3.26.1-py3-none-any.whl (50 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.9/50.9 kB 4.1 MB/s eta 0:00:00
Downloading python_dotenv-1.1.0-py3-none-any.whl (20 kB)
Downloading typing_inspect-0.9.0-py3-none-any.whl (8.8 kB)
Downloading mypy_extensions-1.1.0-py3-none-any.whl (5.0 kB)
Installing collected packages: python-dotenv, mypy-extensions, marshmallow, httpx-sse, typing-inspect, pydantic-s
Successfully installed dataclasses-json-0.6.7 httpx-sse-0.4.0 langchain-community-0.3.24 marshmallow-3.26.1 mypy-
```

```python
import sys
import subprocess

subprocess.check_call([sys.executable, "-m", "pip", "install", "pypdf"])
```

```
0
```

```python
from langchain_community.document_loaders import PyPDFDirectoryLoader

# Read the insurance documents from directory
pdf_directory_loader = PyPDFDirectoryLoader('/content/Policy Documents')
documents = pdf_directory_loader.load()
print(documents)
```

```
[Document(metadata={'producer': 'Microsoft® Office Word 2007', 'creator': 'Microsoft® Office Word 2007', 'creatic
```

```python
from google.colab import drive
drive.mount('/content/drive')
```

```python
# print details and first 100 lines from each docucment
for doc in documents:
    print(f"Source: {doc.metadata['source']}")
    print(f"Page Number: {doc.metadata['page']}")
    print(f"Content: {doc.page_content[:100]}...")  # Displaying the first 100 characters
```

```
Raebareli, Sra...
Source: /content/Policy Documents/HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
Page Number: 23
Content: Email: bimalokpal.pune@ecoi.co.in

B. Power of Ombudsman-
1) The Ombudsman shall receive and con...
Source: /content/Policy Documents/HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
Page Number: 24
Content: iii. after expiry of a period of one month from the date of sending the written representation to th...
Source: /content/Policy Documents/HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
Page Number: 25
Content: Annexure I
LIST OF 138 SURGERIES
 The Surgeries are divided into 4 Categories depending upon the ...
Source: /content/Policy Documents/HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
Page Number: 26
Content: maxillary lesions
33 Open excision of benign mediastinal
lesions   54 Hysterectomy for malignant c...
Source: /content/Policy Documents/HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
Page Number: 27
Content: Cement)
80 Total hip replacement- Others   110 Kidney injury repair
81 Total Knee replacement(With...
Source: /content/Policy Documents/HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
Page Number: 28
Content: Annexure II
Section 39 - Nomination by Policyholder Nomination of a life insurance Policy is as bel...
Source: /content/Policy Documents/HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
Page Number: 29
Content: Disclaimer: This is not a comprehensive list of amendments of Insurance Laws (Amendment) Act, 2015
...
Source: /content/Policy Documents/HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
Page Number: 30
Content: Annexure III
Provisions regarding Policy not being called into question in terms of Section 45 of t...
Source: /content/Policy Documents/HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
Page Number: 31
Content: Annexure IV

Section 38 - Assignment or Transfer of Insurance Policies
Assignment or transfer of ...
Source: /content/Policy Documents/HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf
Page Number: 32
Content: Disclaimer: This is not a comprehensive list of amendments of Insurance Laws (Amendment) Act, 2015
...
```

Document Transformers / Text Splitters

Often times your document is too long (like a book) for your LLM. You need to split it up into chunks. Text splitters help with this.

There are many ways you could split your text into chunks, experiment with different ones to see which is best for you.

LangChain offers different text splitters for splitting the data such as:

- Split by Character
- Recursive Splitter
- Token Splitter

Split by Character - This is the simplest method. This splits based on characters (by default "\n\n") and measure chunk length by number of characters.

- How the text is split: by single character.
- How the chunk size is measured: by number of characters.

Recursive Text Splitter - This text splitter is the recommended one for generic text. It is parameterized by a list of characters. It tries to split on them in order until the chunks are small enough. The default list is ["\n\n", "\n", " ", ""]. This has the effect of trying to keep all paragraphs (and then sentences, and then words) together as long as possible, as those would generically seem to be the strongest semantically related pieces of text.

- How the text is split: by list of characters.
- How the chunk size is measured: by number of characters.

Split by tokens - Language models have a token limit. You should not exceed the token limit. When you split your text into chunks it is therefore a good idea to count the number of tokens. There are many tokenizers. When you count tokens in your text you should use the same tokenizer as used in the language model.

```python
from langchain_text_splitters import RecursiveCharacterTextSplitter

# Initialize the RecursiveCharacterTextSplitter (customize chunk size and overlap as needed)
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)

splits = text_splitter.split_documents(documents)
```

```python
# print a sample chunk
print(splits)
```

```
[Document(metadata={'producer': 'Microsoft® Office Word 2007', 'creator': 'Microsoft® Office Word 2007', 'creatio
```

```python
print ("Text Preview:") # Preview the split texts and the character count
print (splits[0].page_content,"-", len(splits[0].page_content), "\n")
print (splits[1].page_content,"-", len(splits[1].page_content), "\n")
print (splits[2].page_content,"-", len(splits[2].page_content), "\n")
print (splits[3].page_content,"-", len(splits[3].page_content), "\n")
print (splits[4].page_content,"-", len(splits[4].page_content), "\n")
print (splits[5].page_content,"-", len(splits[5].page_content), "\n")
```

```
Text Preview:
F&U dated 15th October 2022                  UIN-101N169V02  P a g e  | 0





   HDFC Life Group Term Life

OF


«OWNERNAME»








Based on the Proposal and the declarations and
any
statement made or referred to therein,
We will pay the Benefits mentioned in this Policy
subject to the terms and conditions contained
herein








<< Designation of the Authorised Signatory >> - 430

F&U dated 15th October 2022                  UIN-101N169V02  P a g e  | 1


PART A: Covering Letter with Policy Schedule

__________________
__________________
__________________
__________________
__________________


Your HDFC Life <Policy Name> with Policy No. <Policy no.>

Dear Mr./Ms.___________________________,

We thank you for choosing HDFC Life Insurance as your preferred life insurance solution provider..

We are pleased to enclose your Policy Bond, which carries the following details of your recently
purchased HDFC Life Insurance Policy:

 Policy Schedule   :  Summary of key features of your HDFC Life Insurance Policy
 Premium Receipt   :  Acknowledgement of the first Premium paid by you - 911

purchased HDFC Life Insurance Policy:
```

Text Embedding Models

The Embeddings class is a class designed for interfacing with text embedding models. LangChain provides support for most of the embedding model providers (OpenAI, Cohere) including sentence transformers library from Hugging Face.

Embeddings create a vector representation of a piece of text and supports all the operations such as similarity search, text comparison, sentiment analysis etc.

The base Embeddings class in LangChain provides two methods: one for embedding documents and one for embedding a query.

The first method takes as input multiple texts, while the second method returns the embedding representation for a single text.

```
# Import the OpenAI Embeddings class from LangChain
from langchain.embeddings import OpenAIEmbeddings
embeddings_model = OpenAIEmbeddings()
```

```
<ipython-input-57-7d47d4db7bf6>:3: LangChainDeprecationWarning: The class `OpenAIEmbeddings` was deprecated in La
  embeddings_model = OpenAIEmbeddings()
```

```
import sys
import subprocess

subprocess.check_call([sys.executable, "-m", "pip", "install", "sentence-transformers"])
```

```
0
```

```
from sentence_transformers import SentenceTransformer

# Load a local embedding model
model = SentenceTransformer('all-MiniLM-L6-v2')

# Embed your text
embeddings = model.encode([splits[0].page_content])
len(embeddings), len(embeddings[0])
```

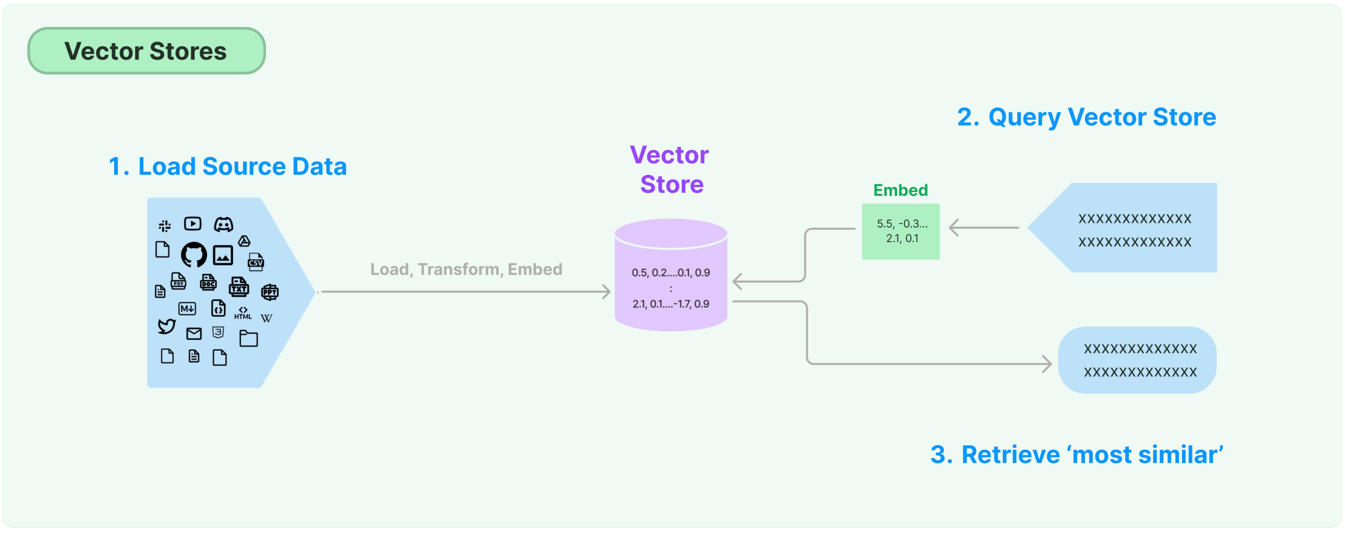
```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/t
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100%                          349/349 [00:00<00:00, 20.2kB/s]
config_sentence_transformers.json: 100%     116/116 [00:00<00:00, 10.6kB/s]
README.md: 100%                             10.5k/10.5k [00:00<00:00, 887kB/s]
sentence_bert_config.json: 100%             53.0/53.0 [00:00<00:00, 4.85kB/s]
config.json: 100%                           612/612 [00:00<00:00, 44.7kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP dow
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not insta
model.safetensors: 100%                     90.9M/90.9M [00:00<00:00, 211MB/s]
tokenizer_config.json: 100%                 350/350 [00:00<00:00, 32.8kB/s]
vocab.txt: 100%                             232k/232k [00:00<00:00, 643kB/s]
tokenizer.json: 100%                        466k/466k [00:00<00:00, 2.70MB/s]
special_tokens_map.json: 100%               112/112 [00:00<00:00, 8.70kB/s]
config.json: 100%                           190/190 [00:00<00:00, 20.0kB/s]
(1, 384)
```

```
type(embeddings)
```

```
numpy.ndarray
```

Vector Stores

One of the most common ways to store and search over unstructured data is to embed it and store the resulting embedding vectors, and then at query time to embed the unstructured query and retrieve the embedding vectors that are 'most similar' to the embedded query. A vector store takes care of storing embedded data and performing vector search for you.



```
from langchain.vectorstores import Chroma
# Initialize OpenAIEmbeddings
openai_embeddings = OpenAIEmbeddings()
```

```
# creating a cache backed embeddings
from langchain.storage import InMemoryStore
from langchain.embeddings import CacheBackedEmbeddings

cache_store = InMemoryStore()
cached_embeddings = CacheBackedEmbeddings.from_bytes_store(
    openai_embeddings,
    cache_store,
    namespace="embeddings_namespace"
)
```

```
import subprocess
import sys

def install(package):
    subprocess.check_call([sys.executable, "-m", "pip", "install", package])

install('chromadb')
```

```
from langchain.embeddings import HuggingFaceEmbeddings

local_embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")

db = Chroma.from_documents(
    documents=splits,
    embedding=local_embeddings,
    persist_directory="./chroma_persistence"
)
```

```
<ipython-input-80-f3a65a001ab1>:3: LangChainDeprecationWarning: The class `HuggingFaceEmbeddings` was deprecated
  local_embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
```

Perform Similarity Search

```
def similarity_search(query):
    return db.similarity_search(query)

docs =  similarity_search("what is the life insurance coverage for disability?")
print(docs[0])
```

```
page_content='□ Terms & Conditions  : Detailed terms of your Policy contract with HDFC Life Insurance
□ Service Options  : Wide range of Policy servicing options that you can Benefit from

We request you to carefully go through the information given in this document. You are also advised to keep the F
```

```
    with utmost care and safety.

    You shall have a period of ___ days from the date of receipt of the Policy Document to review the terms and condi
    Policy and in case of disagreements with any of the terms and conditions, an option to return the Policy to the C
    cancellation can be exercised , stating the reasons for objections. Upon such Free -Look cancellation, the Compan
    the Premium paid subject to deduction of a  proportionate risk Premium for the period of insurance cover in addit
    expenses incurred on medical examination (if any) and the stamp duty charges. All Benefits and rights under this
```

LangChain also support all major vector stores and databases such as FAISS, ElasticSearch, LanceDB, Milvus, Pinecone etc. Refer to the API documentation for the complete list.

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.

The official documentation and API reference contains a list of retriever integrations supported by LangChan.

```
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import CrossEncoderReranker
from langchain_community.cross_encoders import HuggingFaceCrossEncoder

# Initialize a document retriever using the existing vector storage (db).
# The retriever is configured to retrieve a top 20 documents with  mmr score more than 0.8 with cross encoding enable

def get_retriever(topk):
    search_kwargs={"k": topk, "score_threshold": 0.8}
    retriever = db.as_retriever(search_type="mmr", search_kwargs=search_kwargs)

    # Initialize cross-encoder model
    cross_encoder = HuggingFaceCrossEncoder(model_name="BAAI/bge-reranker-base")

    # Set up reranker
    reranker = CrossEncoderReranker(model=cross_encoder, top_n=20)
    return ContextualCompressionRetriever(base_compressor=reranker, base_retriever=retriever)



# Combine retriever and reranker
def get_topk_relevant_documents(query, topk):
    retriever = get_retriever(topk)
    relevant_docs = retriever.invoke(query)
    return relevant_docs
```

```
retriever_docs = get_topk_relevant_documents("what is the life insurance coverage for disability?", 50)
```

```
config.json: 100%                        799/799 [00:00<00:00, 59.3kB/s]
model.safetensors: 100%                  1.11G/1.11G [00:12<00:00, 151MB/s]
tokenizer_config.json: 100%              443/443 [00:00<00:00, 41.4kB/s]
sentencepiece.bpe.model: 100%            5.07M/5.07M [00:00<00:00, 56.0MB/s]
tokenizer.json: 100%                     17.1M/17.1M [00:00<00:00, 81.7MB/s]
special_tokens_map.json: 100%            279/279 [00:00<00:00, 32.7kB/s]
README.md: 100%                          34.1k/34.1k [00:00<00:00, 3.60MB/s]
```

```
import pkg_resources

installed_packages = {pkg.key: pkg.version for pkg in pkg_resources.working_set}
print(installed_packages.get("langchain", "langchain not installed"))
```

```
0.3.25
```

```
# print one page content
retriever_docs[0]
```

```
Document(metadata={'title': 'HDFC Life Easy Health - 101N110V03 - Policy Bond (Single Pay)', 'page': 2,
'creationdate': '2021-11-29T10:03:02+00:00', 'author': 'ANINDYAA', 'creator': 'PyPDF', 'page_label': '3',
'total_pages': 33, 'producer': 'Microsoft: Print To PDF', 'source': '/content/Policy Documents/HDFC-Life-Easy-
Health-101N110V03-Policy-Bond-Single-Pay.pdf', 'moddate': '2021-12-09T06:23:28+00:00'}, page_content='POLICY
DOCUMENT- HDFC LIFE EASY HEALTH \n \nUnique Identification Number: <<101N110V03>> \n \n Your Policy is a Single
Premium paying non participating non linked fixed benefit health plan. This document is \nthe evidence of a
contract between HDFC Life Insurance Company Limited and the Policyholder as described in \nthe Policy Schedule
given below. This Policy is based on the Proposal made by the within named Policyholder \nand submitted to the
Company along with the required documents, declarations, statements,  any response given \nto the Short Medical
Questionnaire (SMQ) by the Life Assured, and other information received by the Company \nfrom the Policyholder,
Life Assured or on behalf of the Policyholder. This Policy is effective upon receipt and \nrealisation, by the
Company, of the consideration payable as Premium under the Policy. This Policy is written \nunder and will be
governed by the applicable laws in force in India and all Premiums and Benefits are expressed')
```

```
# method for combining all relevant page content
def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)
```

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

```
/usr/local/lib/python3.11/dist-packages/langsmith/client.py:272: LangSmithMissingAPIKeyWarning: API key must be p
  warnings.warn(
```

4. 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.

```
# In LangChain, the rag-prompt is a prompt template designed for Retrieval-Augmented Generation (RAG) tasks,
# such as chat and question-answering applications. It facilitates the integration of external context into
# the language model's responses, enhancing the relevance and accuracy of the generated content.

# pulling rag prompt from LangChain hub
from langchain import hub
prompt = hub.pull("rlm/rag-prompt")
```

```
/usr/local/lib/python3.11/dist-packages/langsmith/client.py:272: LangSmithMissingAPIKeyWarning: API key must be p
  warnings.warn(
```

Let's now create a simple LLMChain that takes an input, formats it, and passes it to an LLM for processing. The basic components are PromptTemplate, input queries, an LLM, and optional output parsers.

```
from langchain_openai import ChatOpenAI
llm = ChatOpenAI()
```

```
from langchain_core.runnables import RunnablePassthrough
from langchain_core.output_parsers import StrOutputParser

retriever = get_retriever(50)
rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

```
# test another query
query = "what is condition of deatht while not wearing Seat Belt?"
rag_chain.invoke(query)
```

```
---------------------------------------------------------------------------
RateLimitError                            Traceback (most recent call last)
<ipython-input-97-1a7edd154e7f> in <cell line: 0>()
      1 # test another query
      2 query = "what is condition of deatht while not wearing Seat Belt?"
----> 3 rag_chain.invoke(query)

9 frames
/usr/local/lib/python3.11/dist-packages/openai/_base_client.py in request(self, cast_to, options, stream,
stream_cls)
   1032
   1033                 log.debug("Re-raising status error")
-> 1034                 raise self._make_status_error_from_response(err.response) from None
   1035
   1036             break

RateLimitError: Error code: 429 - {'error': {'message': 'You exceeded your current quota, please check your plan
and billing details. For more information on this error, read the docs:
https://platform.openai.com/docs/guides/error-codes/api-errors.', 'type': 'insufficient_quota', 'param': None,
'code': 'insufficient_quota'}}
```

Next steps: Explain error

```
# test another query
query = "what is criteria for HDFC group insurance?"
rag_chain.invoke(query)
```

```
'The criteria for HDFC group insurance include coverage for eligible members as per the policy terms and
conditions, with the policy prevailing in case of any inconsistencies. HDFC Life Group Term Life is a non-
linked, non-participating policy, and exclusions such as intentional self-inflicted injury, substance abuse,
war, and pre-existing conditions apply. The policy also outlines specific exclusions related to treatments and
their complications.'
```

```
# test another query
query = "what are the benifits of HDFC Sampoorna-Jeevan insurance?"
rag_chain.invoke(query)
```

```
"The benefits of HDFC Sampoorna-Jeevan insurance include non-linked participating individual life insurance
savings, entitlement to participate in the company's profits, and options for lump sum or income benefits on
maturity. The policy also offers guaranteed income benefits and bonus options based on the chosen plan. The
benefits are subject to the terms and conditions outlined in the policy document and schedule."
```

```
# test another query
query = "what are HDFC Life Sanchay Plus Life Long Income Option ?"
rag_chain.invoke(query)
```

```
'HDFC Life Sanchay Plus Life Long Income Option is a part of the HDFC Life Sanchay Plus plan, which is a non-
participating, non-linked savings insurance plan. It provides a life-long income option to policyholders.
Additional servicing charges are not applicable in this policy.'
```

Start coding or generate with AI.