**Semantic Spotter - RAG Project**

Prepared by – Manish Thapa Date: 24-Nov-2024

**Goal:**

Build a robust generative search system capable of effectively and accurately answering questions from various policy documents. You may use LangChain or LlamaIndex to build the generative search application.

**Data Source:** Policy documents present in PDF form as below:

https://github.com/ManishThapa27/semantic-spotter-project/tree/main/insurance-documents

**Collab Notebook PDF: https://github.com/ManishThapa27/semantic-spotter-project/tree/main**[/Semantic-spotter-project.ipynb%20-%20Colab.pdf](https://github.com/arunksinghbuee/semantic-spotter-project/blob/main/Semantic-spotter-project.ipynb%20-%20Colab.pdf)

**System Design:**

A diagram of a search engine

Description automatically generated

RAG (Retrieval-Augmented Generation) is a framework in GenAI that combines the strengths of both retrieval-based and generative models to create more efficient AI assistants. It uses a retrieval mechanism to find relevant information from a large dataset and then generates a response based on that information. This approach improves the accuracy and relevance of responses compared to purely generative models.

LangChain and Llama index can be instrumental in building an efficient AI assistant using RAG. LangChain is a blockchain-based language platform that provides secure and scalable language services. It can be integrated with RAG to enhance its language processing capabilities, allowing the AI assistant to understand and generate more complex and nuanced responses.

On the other hand, Llama index is a large-scale index for efficient language model retrieval. It can be used to quickly retrieve relevant information from a large dataset, which is crucial for the retrieval component of RAG. By leveraging the speed and efficiency of Llama index, the AI assistant can provide faster and more accurate responses to user queries. **We would be using Llama Index for implementing RAG, for efficient searching in insurance policy documents.**

Integrating LangChain and Llama index with RAG can significantly improve the performance and efficiency of AI assistants, making them more capable of understanding and responding to user queries in a more natural and human-like manner.

**Implementation:**

# Importing necessary libraries

!pip install llama-index

# Document loaders for SimpleDirectoryReader

!pip install docx2txt

!pip install pypdf

# Install OpenAI

!pip install openai

#import openAI

from llama\_index.llms.openai import OpenAI

#import ChatMessage

from llama\_index.core.llms import ChatMessage

#import os

import os

import openai

from google.colab import drive

drive.mount('/content/drive', force\_remount=True)

#Set API key

filepath="/content/drive/MyDrive/GenAI/RAG-project/"

with open(filepath + "openai\_api\_key.txt","r") as f:

  openai.api\_key=''.join(f.readlines())

#import SimpleDirectoryReader

from llama\_index.core import SimpleDirectoryReader

# Create object of SimpleDirectoryReader

reader=SimpleDirectoryReader(input\_dir="/content/drive/MyDrive/GenAI/RAG-DEMO/Policy-Documents/")

documents=reader.load\_data()

#lenth of documents

print(f"Loaded {len(documents)} documents/pages successfully.")

Building Query Engine:

# import SimpleNodeParser

from llama\_index.core.node\_parser import SimpleNodeParser

# import VectorStoreIndex

from llama\_index.core import VectorStoreIndex

# import display, HTML

from IPython.display import display, HTML

# Create parser and parse docuemnts into nodes

parser=SimpleNodeParser.from\_defaults()

nodes=parser.get\_nodes\_from\_documents(documents)

# build index

index=VectorStoreIndex(nodes)

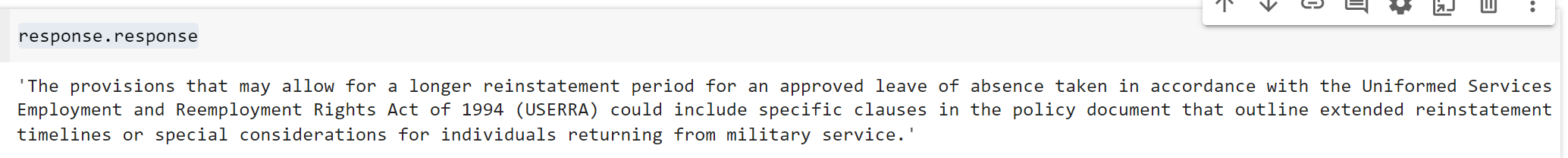
#construct query engine

query\_engine=index.as\_query\_engine()

#query

response=query\_engine.query("What provisions may allow for a longer reinstatement period for an approved leave of absence taken in accordance with the Uniformed Services Employment and Reemployment Rights Act of 1994 (USERRA)?")

response.response

****

**A screenshot of a computer

Description automatically generated**

**A text on a screen

Description automatically generated**

# Extract the score

print(response.source\_nodes[0].score)

# Response Node Text

response.source\_nodes[0].node.text

Creating Response Pipeline:

# Query response function

def query\_response(user\_input):

  response=query\_engine.query(user\_input)

  file\_name=response.source\_nodes[0].node.metadata['file\_name'] + " Page No " + response.source\_nodes[0].node.metadata['page\_label']

  final\_response=response.response + "\nCheck further at " + file\_name

  return final\_response

def initialize\_conv():

  print("Feel free to ask questions related to insurance policies. Enter 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>'))

initialize\_conv()

A close-up of a text

Description automatically generated

**Building a Test Pipeline:**

questions=["What provisions may allow for a longer reinstatement period for an approved leave of absence taken in accordance with the Uniformed Services Employment and Reemployment Rights Act of 1994 (USERRA)?",

           "How is the peroid of time during which a reinstated Member's insurance was not in force treated for the purpose of determining the length of continuous coverage under the Group Policy?",

           "What are the requirements for placing in force any Scheduled benefit that would have been subject to Proof of Good Health has the member remained continuously insured?"]

import pandas as pd

def testing\_pipeline(questions):

  test\_feedback=[]

  for i in questions:

    print(i)

    print(query\_response(i))

    print("\nPlease provide your feedback on the response provided by bot")

    user\_input=input()

    page=query\_response(i).split()[-1]

    test\_feedback.append((i,query\_response(i),page,user\_input))

  feedback\_df=pd.DataFrame(test\_feedback,columns=["Question","Response","Page","Good/Bad"])

  return feedback\_df

testing\_pipeline(questions)

A screenshot of a computer

Description automatically generated

**Building a Custom Prompt Template:**

response=query\_engine.query("What provisions may allow for a longer reinstatement period for an approved leave of absence taken in accordance with the Uniformed Services Employment and Reemployment Rights Act of 1994 (USERRA)?")

retrieved = response.source\_nodes[0].node.text + response.source\_nodes[1].node.text

messages=[

          {

              "role":"system",

              "content":"You are AI assistent to user."

          },

          {

              "role":"user",

              "content": f"""What provisions may allow for a longer reinstatement period for an approved leave of absence taken

              in accordance with the Uniformed Services Employment and Reemployment Rights Act of 1994 (USERRA)? Check in '{retrieved}'

              """

          }

        ]

response2=openai.chat.completions.create(

    model="gpt-3.5-turbo",

    messages=messages

)

response2.choices[0].message.content

**A close-up of a text

Description automatically generated**

Further Improvement:

This model can be further improved by using better/cleaner dataset or utilizing better data pre-processing techniques.

We can also use custom node and LLMs for better results, shown as below:



#Query the engine with specific question

response=query\_engine.query("""What provisions may allow for a longer reinstatement period for an approved leave of absence taken

              in accordance with the Uniformed Services Employment and Reemployment Rights Act of 1994 (USERRA)?""")

A close up of text

Description automatically generated