Mr.HelpMate AI Project

1. Project Goals

The primary objectives of this RAG (Retrieval-Augmented Generation) solution are as follows:

1. **Enable Querying and Information Retrieval:** The main goal is to build a system capable of answering user queries by retrieving relevant information directly from a corpus of insurance policy PDF documents.
2. **Provide Accurate and Contextualized Answers:** The solution aims to deliver precise and contextually appropriate answers to specific questions posed by users, leveraging the content within the insurance policies.
3. **Enhance Information Access:** A key objective is to significantly improve the ease of access and retrieval of critical information from complex and often lengthy insurance policy documents, making it more efficient for users to find what they need.
4. **Ensure Transparency and Trustworthiness through Citation:** The solution intends to provide citations, including the exact source (policy name and page number) of the retrieved information, thereby ensuring transparency, verifiability, and trustworthiness of the generated answers.

2. Data Source

The primary data source for this project is the `Principal-Sample-Life-Insurance-Policy.pdf` file. This PDF document contains the content of a sample life insurance policy, which is crucial for building a RAG system to answer insurance-related queries.

To process this PDF, the `pdfplumber` library was utilized. Its role was fundamental in opening the PDF document, iterating through each page, and meticulously extracting both regular text and tabular data. The library allows for detailed access to the layout of each page, including word coordinates and table bounding boxes.

A custom function, `check_bboxes`, was implemented to assist in the table extraction process. This function's purpose was to distinguish between words that are part of a table and those that are regular text. By comparing the bounding box of each word with the bounding boxes of identified tables, it ensured accurate segregation of content types.

After identifying both non-table words and extracted tables, `pdfplumber.utils.cluster_objects` was employed. This utility is critical for maintaining the chronological order of text and tables as they appear in the original PDF, ensuring that the extracted content preserves its natural reading flow. It groups related elements together based on their vertical position.

Finally, the extracted and ordered data was structured into a Pandas DataFrame named `insurance_pdfs_data`. This DataFrame includes columns for 'Page No.', 'Page_Text' (containing the aggregated text and table content for each page), and 'Document Name', providing a clean and organized dataset for further processing.

```
import chromadb
import json
import pdfplumber
import pandas as pd
from chromadb.utils.embedding_functions import OpenAIEmbeddingFunction
from openai import OpenAI
from operator import itemgetter
from sentence_transformers import CrossEncoder, util
```

```
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1772042928.027230  159584 instrument.cc:563] Metric with
name 'grpc.resource_quota.calls_dropped' registered more than once.
Ignoring later registration.
E0000 00:00:1772042928.027258  159584 instrument.cc:563] Metric with
name 'grpc.resource_quota.calls_rejected' registered more than once.
Ignoring later registration.
E0000 00:00:1772042928.027260  159584 instrument.cc:563] Metric with
name 'grpc.resource_quota.connections_dropped' registered more than
once. Ignoring later registration.
E0000 00:00:1772042928.027262  159584 instrument.cc:563] Metric with
name 'grpc.resource_quota.instantaneous_memory_pressure' registered
more than once. Ignoring later registration.
E0000 00:00:1772042928.027264  159584 instrument.cc:563] Metric with
name 'grpc.resource_quota.memory_pressure_control_value' registered
more than once. Ignoring later registration.
/opt/anaconda3/envs/openaienv/lib/python3.13/site-packages/tqdm/auto.p
y:21: TqdmWarning: IProgress not found. Please update jupyter and
ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
# Function to check whether a word is present in a table or not for
segregation of regular text and tables

def check_bboxes(word, table_bbox):
    # Check whether word is inside a table bbox.
    l = word['x0'], word['top'], word['x1'], word['bottom']
    r = table_bbox
    return l[0] > r[0] and l[1] > r[1] and l[2] < r[2] and l[3] < r[3]

# Function to extract text from a PDF file.
# 1. Declare a variable p to store the iteration of the loop that will
help us store page numbers alongside the text
# 2. Declare an empty list 'full_text' to store all the text files
# 3. Use pdfplumber to open the pdf pages one by one
# 4. Find the tables and their locations in the page
# 5. Extract the text from the tables in the variable 'tables'
# 6. Extract the regular words by calling the function check_bboxes()
and checking whether words are present in the table or not
```

```
# 7. Use the cluster_objects utility to cluster non-table and table
words together so that they retain the same chronology as in the
original PDF
# 8. Declare an empty list 'lines' to store the page text
# 9. If a text element in present in the cluster, append it to
'lines', else if a table element is present, append the table
# 10. Append the page number and all lines to full_text, and increment
'p'
# 11. When the function has iterated over all pages, return the
'full_text' list

def extract_text_from_pdf(pdf_path):
    p = 0
    full_text = []


    with pdfplumber.open(pdf_path) as pdf:
        for page in pdf.pages:
            page_no = f"Page {p+1}"
            text = page.extract_text()

            tables = page.find_tables()
            table_bboxes = [i.bbox for i in tables]
            tables = [{'table': i.extract(), 'top': i.bbox[1]} for i
in tables]
            non_table_words = [word for word in page.extract_words()
if not any(
                [check_bboxes(word, table_bbox) for table_bbox in
table_bboxes])]
            lines = []

            for cluster in
pdfplumber.utils.cluster_objects(non_table_words + tables,
itemgetter('top'), tolerance=5):

                if 'text' in cluster[0]:
                    try:
                        lines.append(' '.join([i['text'] for i in
cluster]))
                    except KeyError:
                        pass

                elif 'table' in cluster[0]:
                    lines.append(json.dumps(cluster[0]['table']))


            full_text.append([page_no, " ".join(lines)])
            p +=1

    return full_text
```

```
pdf_path = 'Principal-Sample-Life-Insurance-Policy.pdf'

# Process the PDF file
print(f"...Processing {pdf_path}")

# Call the function to extract the text from the PDF
extracted_text = extract_text_from_pdf(pdf_path)

# Convert the extracted list to a PDF, and add a column to store
document names
insurance_pdfs_data = pd.DataFrame(extracted_text, columns=['Page
No.', 'Page_Text'])
insurance_pdfs_data['Document Name'] = pdf_path

# Print a message to indicate progress
print(f"Finished processing {pdf_path}")
```

```
...Processing Principal-Sample-Life-Insurance-Policy.pdf
Finished processing Principal-Sample-Life-Insurance-Policy.pdf
```

```
insurance_pdfs_data.head()
```

```
  Page No.                                          Page_Text  \
0   Page 1  DOROTHEA GLAUSE S655 RHODE ISLAND JOHN DOE 01/...
1   Page 2                 This page left blank intentionally
2   Page 3  POLICY RIDER GROUP INSURANCE POLICY NO: S655 C...
3   Page 4                 This page left blank intentionally
4   Page 5  PRINCIPAL LIFE INSURANCE COMPANY (called The P...

                               Document Name
0  Principal-Sample-Life-Insurance-Policy.pdf
1  Principal-Sample-Life-Insurance-Policy.pdf
2  Principal-Sample-Life-Insurance-Policy.pdf
3  Principal-Sample-Life-Insurance-Policy.pdf
4  Principal-Sample-Life-Insurance-Policy.pdf
```

3. Design Choices

3.1 Data Preprocessing

After extracting text from the PDF, the `insurance_pdfs_data` DataFrame was created to store the page content along with its origin. To ensure that only meaningful pages were retained, a `Text_Length` column was generated by calculating the number of words on each page (`insurance_pdfs_data['Page_Text'].apply(lambda x: len(x.split(' '))`). Pages with fewer than 40 words were filtered out (`insurance_pdfs_data = insurance_pdfs_data.loc[insurance_pdfs_data['Text_Length'] >= 40]`). This helps in removing blank pages or pages with minimal content that would not be useful for retrieval.

Subsequently, a `Metadata` column was created for each page. This column stores a dictionary containing `Policy_Name` (derived from the document name by removing the '.pdf' extension) and the original `Page_No.`. This metadata is crucial for later stages, especially for citations in the RAG response.

```
# Let's also check the length of all the texts as there might be some
empty pages or pages with very few words that we can drop
insurance_pdfs_data['Text_Length'] =
insurance_pdfs_data['Page_Text'].apply(lambda x: len(x.split(' ')))

insurance_pdfs_data['Text_Length'].head()
```

```
0     30
1      5
2    230
3      5
4    110
Name: Text_Length, dtype: int64
```

```
# Retain only the rows with a text length of at least 10
insurance_pdfs_data =
insurance_pdfs_data.loc[insurance_pdfs_data['Text_Length'] >= 40]
insurance_pdfs_data.head()
```

```
  Page No.                                          Page_Text  \
2   Page 3  POLICY RIDER GROUP INSURANCE POLICY NO: S655 C...
4   Page 5  PRINCIPAL LIFE INSURANCE COMPANY (called The P...
5   Page 6  TABLE OF CONTENTS PART I - DEFINITIONS PART II...
6   Page 7  Section A – Eligibility Member Life Insurance ...
7   Page 8  Section A - Member Life Insurance Schedule of ...

                              Document Name  Text_Length
2  Principal-Sample-Life-Insurance-Policy.pdf          230
4  Principal-Sample-Life-Insurance-Policy.pdf          110
5  Principal-Sample-Life-Insurance-Policy.pdf          153
6  Principal-Sample-Life-Insurance-Policy.pdf          176
7  Principal-Sample-Life-Insurance-Policy.pdf          171
```

```
# Store the metadata for each page in a separate column
insurance_pdfs_data['Metadata'] = insurance_pdfs_data.apply(lambda x:
{'Policy_Name': x['Document Name'][:-4], 'Page_No.': x['Page No.']},
axis=1)

insurance_pdfs_data.head()
```

```
  Page No.                                          Page_Text  \
2   Page 3  POLICY RIDER GROUP INSURANCE POLICY NO: S655 C...
4   Page 5  PRINCIPAL LIFE INSURANCE COMPANY (called The P...
5   Page 6  TABLE OF CONTENTS PART I - DEFINITIONS PART II...
6   Page 7  Section A – Eligibility Member Life Insurance ...
7   Page 8  Section A - Member Life Insurance Schedule of ...
```

```
                                  Document Name  Text_Length  \
2  Principal-Sample-Life-Insurance-Policy.pdf          230
4  Principal-Sample-Life-Insurance-Policy.pdf          110
5  Principal-Sample-Life-Insurance-Policy.pdf          153
6  Principal-Sample-Life-Insurance-Policy.pdf          176
7  Principal-Sample-Life-Insurance-Policy.pdf          171

                                             Metadata
2  {'Policy_Name': 'Principal-Sample-Life-Insuran...
4  {'Policy_Name': 'Principal-Sample-Life-Insuran...
5  {'Policy_Name': 'Principal-Sample-Life-Insuran...
6  {'Policy_Name': 'Principal-Sample-Life-Insuran...
7  {'Policy_Name': 'Principal-Sample-Life-Insuran...
```

```
insurance_pdfs_data.iloc[2, 4]
```

```
{'Policy_Name': 'Principal-Sample-Life-Insurance-Policy', 'Page_No.':
'Page 6'}
```

3.2 Vector Store Setup

ChromaDB was chosen as the vector database for this RAG implementation due to its simplicity, ease of use, and local persistent storage capabilities. The client was initialized using `client = chromadb.PersistentClient()`, which allows the database to persist data across sessions at the specified `chroma_data_path = './ChromaDB_Data'`.

The main collection for storing insurance policy documents, named `RAG_on_Insurance`, was created or retrieved using `insurance_collection = client.get_or_create_collection(name='RAG_on_Insurance', embedding_function=embedding_function)`. This ensures that if the collection already exists, it is loaded, otherwise a new one is created. The `embedding_function` parameter is crucial here, as it dictates how documents added to this collection will be converted into embeddings.

```
key = ''
try:
    with open('openai_api_key.json', 'r') as file:
        data = json.load(file)
    key = data['api_key']
except FileNotFoundError:
    print("Error: The file 'data.json' was not found. Please check the
file path.")
except json.JSONDecodeError as e:
    print(f"Error: Failed to decode JSON from the file. Details: {e}")

# Set the API key
openai_client = OpenAI(api_key=key)
```

```
# Define the path where chroma collections will be stored
chroma_data_path = './ChromaDB_Data'

# Call PersistentClient()
client = chromadb.PersistentClient()
```

3.3 Embedding Function

For converting the text into vector embeddings, **OpenAI's `text-embedding-ada-002` model** was utilized. This model is known for its strong performance in semantic similarity tasks. The embedding function was set up using `OpenAIEmbeddingFunction(api_key=key, model_name=model)`, where `key` is the OpenAI API key and `model` refers to `"text-embedding-ada-002"`. This `embedding_function` is then passed to the ChromaDB collections (`insurance_collection` and `cache_collection`) to ensure that all documents added to these collections are embedded using this specific OpenAI model.

```
# Set up the embedding function using the OpenAI embedding model
model = "text-embedding-ada-002"
embedding_function = OpenAIEmbeddingFunction(api_key=key,
model_name=model)

# Initialise a collection in chroma and pass the embedding_function to
it so that it used OpenAI embeddings to embed the documents
insurance_collection =
client.get_or_create_collection(name='RAG_on_Insurance',
embedding_function=embedding_function)

# Convert the page text and metadata from your dataframe to lists to
be able to pass it to chroma
documents_list = insurance_pdfs_data["Page_Text"].tolist()
metadata_list = insurance_pdfs_data['Metadata'].tolist()

# Add the documents and metadata to the collection alongwith generic
integer IDs. You can also feed the metadata information as IDs by
combining the policy name and page no.
insurance_collection.add(
    documents= documents_list,
    ids = [str(i) for i in range(0, len(documents_list))],
    metadatas = metadata_list
)

# Let's take a look at the first few entries in the collection
insurance_collection.get(
    ids = ['0','1','2'],
    include = ['embeddings', 'documents', 'metadatas']
)
```

```
{'ids': ['0', '1', '2'],
 'embeddings': array([[-1.31132873e-02,  8.84512160e-03, -4.58381791e-
03, ...,
```

```
        -1.56453662e-02, -7.67150486e-05,  7.29430886e-03],
       [-1.21258916e-02,  1.41026508e-02, -3.38967773e-03, ...,
        -2.84971725e-02, -9.47252382e-03,  1.02420002e-02],
       [ 2.90325540e-03, -3.95935756e-04,  7.50349555e-03, ...,
        -3.55480565e-03,  2.13879161e-02, -1.91559065e-02]],
      shape=(3, 1536)),
 'documents': ['POLICY RIDER GROUP INSURANCE POLICY NO: S655 COVERAGE: Life EMPLOYER: RHODE ISLAND JOHN DOE Effective on the later of the Date of Issue of this Group Policy or March 1, 2005, the following will apply to your Policy: From time to time The Principal may offer or provide certain employer groups who apply for coverage with The Principal a Financial Services Hotline and Grief Support Services or any other value added service for the employees of that employer group. In addition, The Principal may arrange for third party service providers (i.e., optometrists, health clubs), to provide discounted goods and services to those employer groups who apply for coverage with The Principal or who become insureds/enrollees of The Principal. While The Principal has arranged these goods, services and/or third party provider discounts, the third party service providers are liable to the applicants/insureds/enrollees for the provision of such goods and/or services. The Principal is not responsible for the provision of such goods and/or services nor is it liable for the failure of the provision of the same. Further, The Principal is not liable to the applicants/insureds/enrollees for the negligent provision of such goods and/or services by the third party service providers. EXCEPT AS SPECIFICALLY DESCRIBED IN THIS RIDER, ALL OTHER BENEFITS AND PROVISIONS WILL BE AS DESCRIBED IN THE GROUP POLICY. PRINCIPAL LIFE INSURANCE COMPANY DES MOINES, IOWA 50392-0001 GC 806 VAL',
  "PRINCIPAL LIFE INSURANCE COMPANY (called The Principal in this Group Policy) Des Moines, Iowa 50392-0002 This group insurance policy is issued to: RHODE ISLAND JOHN DOE (called the Policyholder in this Group Policy) The Date of Issue is November 1, 2007. In return for the Policyholder's application and payment of all premiums when due, The Principal agrees to provide: MEMBER LIFE INSURANCE MEMBER ACCIDENTAL DEATH AND DISMEMBERMENT INSURANCE DEPENDENT LIFE INSURANCE subject to the terms and conditions described in this Group Policy. GROUP POLICY NO. GL S655 RENEWABLE TERM - NON-PARTICIPATING CONTRACT STATE OF ISSUE: RHODE ISLAND This policy has been updated effective January 1, 2014 GC 6000 TITLE PAGE",
  'TABLE OF CONTENTS PART I - DEFINITIONS PART II - POLICY ADMINISTRATION Section A – Contract Entire Contract Article 1 Policy Changes Article 2 Policyholder Eligibility Requirements Article 3 Policy Incontestability Article 4 Individual Incontestability Article 5 Information to be Furnished Article 6 Certificates Article 7 Assignments Article 8 Dependent Rights Article 9 Policy Interpretation Article 10 Electronic Transactions Article 11 Section B – Premium Payment Responsibility; Due Dates; Grace Period Article 1 Premium Rates Article 2 Premium Rate Changes Article 3 Premium Amount Article 4 Contributions from Members Article 5 Section C - Policy Termination
```

```
Failure to Pay Premium Article 1 Termination Rights of the
Policyholder Article 2 Termination Rights of The Principal Article 3
Policyholder Responsibility to Members Article 4 Section D - Policy
Renewal Renewal Article 1 PART III - INDIVIDUAL REQUIREMENTS AND
RIGHTS This policy has been updated effective January 1, 2014 GC 6001
TABLE OF CONTENTS, PAGE 1'],
 'uris': None,
 'included': ['embeddings', 'documents', 'metadatas'],
 'data': None,
 'metadatas': [{'Page_No.': 'Page 3',
   'Policy_Name': 'Principal-Sample-Life-Insurance-Policy'},
  {'Policy_Name': 'Principal-Sample-Life-Insurance-Policy',
   'Page_No.': 'Page 5'},
  {'Policy_Name': 'Principal-Sample-Life-Insurance-Policy',
   'Page_No.': 'Page 6'}]}
```

```
cache_collection =
client.get_or_create_collection(name='Insurance_Cache',
embedding_function=embedding_function)

cache_collection.peek()
```

```
{'ids': [],
 'embeddings': array([], dtype=float64),
 'documents': [],
 'uris': None,
 'included': ['metadatas', 'documents', 'embeddings'],
 'data': None,
 'metadatas': []}
```

3.4 Caching Mechanism

To optimize performance and reduce API calls, a caching mechanism was implemented using a separate ChromaDB collection named `Insurance_Cache`. This cache stores previous queries and their corresponding retrieval results. Before performing a full semantic search on the `insurance_collection`, the system first queries the `cache_collection` with the user's input (`cache_results = cache_collection.query(query_texts=query, n_results=1)`).

A `threshold` (set to `0.2`) determines whether a cached result is sufficiently similar to the current query. If `cache_results['distances'][0]` is empty (no cache hit) or the `distance` of the closest cached entry is *greater* than the `threshold`, it indicates that the current query is not closely matched by any cached query. In this scenario, a new semantic search is performed on the `insurance_collection`.

Crucially, if a new search is performed, the query and its top 10 retrieval results (including IDs, documents, distances, and metadatas) are then added to the `cache_collection`. The query itself is stored as the document, and the retrieved information is stored as metadata for that cache entry. This pre-populates the cache for future, similar queries. If, however, the `distance` of a cached result is *less than or equal to* the `threshold`, it signifies a good match, and the

cached results (documents, metadatas, etc.) are directly retrieved and used, avoiding a redundant search on the main collection.

```
# Implementing Cache in Semantic Search
def cache_layer(query):
    threshold = 0.2

    ids = []
    documents = []
    distances = []
    metadatas = []
    results_df = pd.DataFrame()

    # Search the Cache collection first
    # Query the collection against the user query and return the top
20 results

    cache_results = cache_collection.query(query_texts=query,
n_results=1)

    # If the distance is greater than the threshold, then return the
results from the main collection.

    if cache_results["distances"][0] == [] or
cache_results["distances"][0][0] > threshold:
        # Query the collection against the user query and return the
top 10 results
        results = insurance_collection.query(query_texts=query,
n_results=10)

        # Store the query in cache_collection as document w.r.t to
ChromaDB so that it can be embedded and searched against later
        # Store retrieved text, ids, distances and metadatas in
cache_collection as metadatas, so that they can be fetched easily if a
query indeed matches to a query in cache
        Keys = []
        Values = []

        for key, val in results.items():
            if val is None or key in ["embeddings", "uris",
"included", "data"]:
                continue
            for i in range(10):
                Keys.append(str(key) + str(i))
                Values.append(str(val[0][i]))

        cache_collection.add(
            documents=[query],
            ids=[
                query
```

```
            ],  # Or if you want to assign integers as IDs 0,1,2,.., then you can use "len(cache_results['documents'])" as will return the no. of queries currently in the cache and assign the next digit to the new query."
            metadatas=dict(zip(Keys, Values)),
        )

        print("Not found in cache. Found in main collection.")

        result_dict = {
            "Metadatas": results["metadatas"][0],
            "Documents": results["documents"][0],
            "Distances": results["distances"][0],
            "IDs": results["ids"][0],
        }
        results_df = pd.DataFrame.from_dict(result_dict)

    # If the distance is, however, less than the threshold, you can return the results from cache

    elif cache_results["distances"][0][0] <= threshold:
        cache_result_dict = cache_results["metadatas"][0][0]

        # Loop through each inner list and then through the dictionary
        for key, value in cache_result_dict.items():
            if "ids" in key:
                ids.append(value)
            elif "documents" in key:
                documents.append(value)
            elif "distances" in key:
                distances.append(value)
            elif "metadatas" in key:
                metadatas.append(value)

        print("Found in cache!")

        # Create a DataFrame
        results_df = pd.DataFrame(
            {
                "IDs": ids,
                "Documents": documents,
                "Distances": distances,
                "Metadatas": metadatas,
            }
        )

    return results_df
```

3.5 Reranking Model

After the initial semantic search retrieves a set of potentially relevant documents from the `insurance_collection`, a reranking step is applied to further refine the results and improve their relevance to the user's query. This is achieved using a **CrossEncoder model**, specifically `cross-encoder/ms-marco-MiniLM-L-6-v2`.

The `CrossEncoder` takes a pair of inputs (the user query and each retrieved document) and outputs a single score indicating the semantic similarity or relevance between them. Unlike semantic search, which embeds query and document independently, a cross-encoder jointly embeds them, often leading to more precise relevance scores. The `cross_encoder.predict()` method is used to generate these `cross_rerank_scores` for the top `n_results` (e.g., 10) documents obtained from the semantic search. These scores are then used to sort the documents, presenting the most relevant ones to the LLM for RAG.

```
# Initialise the cross encoder model
cross_encoder = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2')

def rerank_layer(results_df, query):
    # Input (query, response) pairs for each of the top 10 responses
received from the semantic search to the cross encoder
    # Generate the cross_encoder scores for these pairs
    cross_inputs = [[query, response] for response in
results_df['Documents']]
    cross_rerank_scores = cross_encoder.predict(cross_inputs)

    # Store the rerank_scores in results_df
    results_df['Reranked_scores'] = cross_rerank_scores

    # Return the top 3 results after reranking
    top_3_rerank = results_df.sort_values(by='Reranked_scores',
ascending=False)
    return top_3_rerank[:3]
```

3.6 Test Queries - Search Layer

3.6.1 Query 1

```
# Read the user query
query_1 = input()
```

```
 How can I renew my policy?
```

```
results_df_1 = cache_layer(query_1)
```

```
Not found in cache. Found in main collection.
```

```
results_df_1
```

```
                                                Metadatas  \
0  {'Page_No.': 'Page 25', 'Policy_Name': 'Princi...
```

```
1  {'Policy_Name': 'Principal-Sample-Life-Insuran...
2  {'Page_No.': 'Page 21', 'Policy_Name': 'Princi...
3  {'Policy_Name': 'Principal-Sample-Life-Insuran...
4  {'Page_No.': 'Page 16', 'Policy_Name': 'Princi...
5  {'Page_No.': 'Page 40', 'Policy_Name': 'Princi...
6  {'Policy_Name': 'Principal-Sample-Life-Insuran...
7  {'Policy_Name': 'Principal-Sample-Life-Insuran...
8  {'Page_No.': 'Page 23', 'Policy_Name': 'Princi...
9  {'Page_No.': 'Page 42', 'Policy_Name': 'Princi...
```

```
                                           Documents  Distances IDs
0  Section D - Policy Renewal Article 1 - Renewal...   0.321832  20
1  T he Principal may terminate the Policyholder'...   0.394551  19
2  b . on any date the definition of Member or De...   0.409399  16
3  a. be actively engaged in business for profit ...   0.409487  12
4  PART II - POLICY ADMINISTRATION Section A - Co...   0.415039  11
5  Section E - Reinstatement Article 1 - Reinstat...   0.415444  35
6  Section B - Premiums Article 1 - Payment Respo...   0.415885  15
7  c . a copy of the form which contains the stat...   0.416359  13
8  Section C - Policy Termination Article 1 - Fai...   0.418073  18
9  Section F - Individual Purchase Rights Article...   0.418707  37
```

```
top_3_RAG_1 = rerank_layer(results_df_1, query_1)
```

```
top_3_RAG_1
```

```
                                           Metadatas  \
0  {'Page_No.': 'Page 25', 'Policy_Name': 'Princi...
1  {'Policy_Name': 'Principal-Sample-Life-Insuran...
9  {'Page_No.': 'Page 42', 'Policy_Name': 'Princi...

                                           Documents  Distances IDs  \
0  Section D - Policy Renewal Article 1 - Renewal...   0.321832  20
1  T he Principal may terminate the Policyholder'...   0.394551  19
9  Section F - Individual Purchase Rights Article...   0.418707  37

   Reranked_scores
0         2.522425
1        -3.997698
9        -4.721786
```

3.6.2 Query 2

```
# Read the user query
query_2 = input()
```

```
 Do I have to submit any medical report to apply for insurance?
```

```
results_df_2 = cache_layer(query_2)
```

```
Not found in cache. Found in main collection.
```

```
results_df_2
```

```
                                                Metadatas  \
0  {'Page_No.': 'Page 29', 'Policy_Name': 'Princi...
1  {'Page_No.': 'Page 13', 'Policy_Name': 'Princi...
2  {'Page_No.': 'Page 22', 'Policy_Name': 'Princi...
3  {'Page_No.': 'Page 17', 'Policy_Name': 'Princi...
4  {'Page_No.': 'Page 50', 'Policy_Name': 'Princi...
5  {'Policy_Name': 'Principal-Sample-Life-Insuran...
6  {'Policy_Name': 'Principal-Sample-Life-Insuran...
7  {'Page_No.': 'Page 51', 'Policy_Name': 'Princi...
8  {'Page_No.': 'Page 28', 'Policy_Name': 'Princi...
9  {'Page_No.': 'Page 40', 'Policy_Name': 'Princi...

                                           Documents  Distances IDs
0  Insurance for which Proof of Good Health is re...   0.400756  24
1  a . A licensed Doctor of Medicine (M.D.) or Os...   0.407622   9
2  The number of Members insured for Dependent Li...   0.414135  17
3  a. be actively engaged in business for profit ...   0.418969  12
4  The Principal may require that a ADL Disabled ...   0.428522  45
5  c . a copy of the form which contains the stat...   0.431274  13
6  A claimant may request an appeal of a claim de...   0.435729  57
7  Coverage During Disability will cease on the e...   0.437953  46
8  Section B - Effective Dates Article 1 - Member...   0.438268  23
9  Section E - Reinstatement Article 1 - Reinstat...   0.439320  35
```

```
top_3_RAG_2 = rerank_layer(results_df_2, query_2)
```

```
top_3_RAG_2
```

```
                                                Metadatas  \
0  {'Page_No.': 'Page 29', 'Policy_Name': 'Princi...
7  {'Page_No.': 'Page 51', 'Policy_Name': 'Princi...
4  {'Page_No.': 'Page 50', 'Policy_Name': 'Princi...

                                           Documents  Distances IDs  \
0  Insurance for which Proof of Good Health is re...   0.400756  24
7  Coverage During Disability will cease on the e...   0.437953  46
4  The Principal may require that a ADL Disabled ...   0.428522  45

   Reranked_scores
0        -4.240024
7        -5.841960
4        -6.769630
```

3.6.3 Query 3

```
# Read the user query
query_3 = input()
```

```
 Will I have insurance cover if I travel outside the country?
```

```
results_df_3 = cache_layer(query_3)
```

```
Not found in cache. Found in main collection.
```

```
results_df_3
```

```
                                           Metadatas  \
0  {'Policy_Name': 'Principal-Sample-Life-Insuran...
1  {'Page_No.': 'Page 37', 'Policy_Name': 'Princi...
2  {'Policy_Name': 'Principal-Sample-Life-Insuran...
3  {'Page_No.': 'Page 36', 'Policy_Name': 'Princi...
4  {'Policy_Name': 'Principal-Sample-Life-Insuran...
5  {'Policy_Name': 'Principal-Sample-Life-Insuran...
6  {'Policy_Name': 'Principal-Sample-Life-Insuran...
7  {'Policy_Name': 'Principal-Sample-Life-Insuran...
8  {'Page_No.': 'Page 17', 'Policy_Name': 'Princi...
9  {'Page_No.': 'Page 18', 'Policy_Name': 'Princi...

                                           Documents  Distances IDs
0  I f coverage for a Member or Dependent termina...   0.387098  36
1  b. a business assignment; or c. full-time stud...   0.387501  32
2  Exposure Exposure to the elements will be pres...   0.420368  50
3  A Member's insurance under this Group Policy f...   0.429939  31
4  Section E - Reinstatement Article 1 - Reinstat...   0.433905  35
5  Section A – Eligibility Member Life Insurance ...   0.434418   3
6  Section F - Individual Purchase Rights Article...   0.447884  37
7  a . A licensed Doctor of Medicine (M.D.) or Os...   0.449487   9
8  a. be actively engaged in business for profit ...   0.450953  12
9  c . a copy of the form which contains the stat...   0.453485  13
```

```
top_3_RAG_3 = rerank_layer(results_df_3, query_3)
```

```
top_3_RAG_3
```

```
                                           Metadatas  \
0  {'Policy_Name': 'Principal-Sample-Life-Insuran...
1  {'Page_No.': 'Page 37', 'Policy_Name': 'Princi...
3  {'Page_No.': 'Page 36', 'Policy_Name': 'Princi...

                                           Documents  Distances IDs  \
0  I f coverage for a Member or Dependent termina...   0.387098  36
1  b. a business assignment; or c. full-time stud...   0.387501  32
3  A Member's insurance under this Group Policy f...   0.429939  31

   Reranked_scores
0        -2.569919
1        -3.941121
3        -5.215133
```

3.7 RAG Prompt Structure

The RAG (Retrieval-Augmented Generation) process culminates in the `generate_response` function, which orchestrates the interaction with the Large Language Model (LLM). This function constructs a comprehensive prompt using both a system message and a user message to guide the LLM's response generation.

LLM Model: The **`gpt-5-nano`** model is specified for generating the final answer.

System Message: A system message sets the persona of the LLM: `"You are a helpful assistant in the insurance domain who can effectively answer user queries about insurance policies and documents."` This ensures the LLM adopts an appropriate tone and focuses on the relevant subject matter.

User Message: The user message is the core of the RAG prompt, providing the LLM with the user's `query` and the top 3 `results_df` (which contains `Documents` and `Metadatas`) obtained from the reranking step. Key instructions within this message include:

1. **Utilize Search Results:** The LLM is explicitly instructed to use the provided search results (`top_3_RAG` DataFrame) to answer the `query`.
2. **Informative Answer:** The goal is to frame an informative answer.
3. **Citations:** The LLM must return relevant policy names and page numbers as citations, using the `Metadata` column from the search results.
4. **Relevance Filter:** It's advised not to use all information from the DataFrame, but only what is relevant.
5. **Table Handling:** If the document text contains relevant tables (represented as a list of lists), the LLM should reformat and present this information in a tabular format within its answer.
6. **Partial Answers:** If a complete answer cannot be provided, the LLM should offer guidance on how the user can find more information in the cited documents.
7. **Customer-Facing:** The response should be direct, customer-facing, and avoid internal workings details.
8. **Irrelevant Queries:** If the query is deemed irrelevant to the provided documents, the LLM should state so.
9. **Formatting:** The final response should be well-formatted, easily readable, and present the complete answer before the citations.

```
# Define the function to generate the response. Provide a
comprehensive prompt that passes the user query and the top 3 results
to the model
def generate_response(query, top_3_RAG):
    """
    Generate a response using GPT-3.5's ChatCompletion based on the
user query and retrieved information.
    """
    messages = [
                {"role": "system", "content":  "You are a helpful
assistant in the insurance domain who can effectively answer user
queries about insurance policies and documents."},
```

```
                {"role": "user", "content": f"""You have a question
asked by the user in '{query}' and you have some search results from a
corpus of insurance documents in the dataframe '{top_3_RAG}'. These
search results are essentially one page of an insurance document that
may be relevant to the user query.

                                                The column 'documents'
inside this dataframe contains the actual text from the policy
document and the column 'metadata' contains the policy name and source
page. The text inside the document may also contain tables in the
format of a list of lists where each of the nested lists indicates a
row.

                                                Use the documents in
'{top_3_RAG}' to answer the query '{query}'. Frame an informative
answer and also, use the dataframe to return the relevant policy names
and page numbers as citations.

                                                Follow the guidelines
below when performing the task.
                                                1. Try to provide
relevant/accurate numbers if available.
                                                2. You don't have to
necessarily use all the information in the dataframe. Only choose
information that is relevant.
                                                3. If the document
text has tables with relevant information, please reformat the table
and return the final information in a tabular in format.
                                                3. Use the Metadatas
columns in the dataframe to retrieve and cite the policy name(s) and
page numbers(s) as citation.
                                                4. If you can't
provide the complete answer, please also provide any information that
will help the user to search specific sections in the relevant cited
documents.
                                                5. You are a customer
facing assistant, so do not provide any information on internal
workings, just answer the query directly.

                                                The generated response
should answer the query directly addressing the user and avoiding
additional information. If you think that the query is not relevant to
the document, reply that the query is irrelevant. Provide the final
response as a well-formatted and easily readable text along with the
citation. Provide your complete response first with all information,
and then provide the citations.
                                                """},
            ]

    response = openai_client.chat.completions.create(
```

```
        model="gpt-5-nano",
        messages=messages
    )

    return response.choices[0].message.content.split('\n')
```

3.8 Test Queries - RAG Layer

3.8.1 Query 1

```
# Generate the response
response = generate_response(query_1, top_3_RAG_1)

# Print the response
print("\n".join(response))
```

```
Here’s how you can renew your policy:

- Check the renewal or expiry date on your current policy and any
renewal notice you received.
- Choose your renewal option: automatic renewal (if offered) or manual
renewal. You can usually renew online via the insurer’s portal, by
contacting your agent, or by calling the customer service center.
- Update any details if needed (address, contact information,
beneficiaries, or any riders you may want to keep or adjust).
- Pay the renewal premium by the due date using your preferred method
(online payment, bank transfer, phone payment, or in person).
- Confirm renewal receipt and keep the renewed policy document or
endorsement for your records.
- If you want changes to coverage or to add/remove riders, request the
renewal endorsement or a policy change during the renewal process.
- If there are health or risk changes, the insurer may require
underwriting as part of the renewal.

For this policy, renewal guidance is described in Section D - Policy
Renewal Article 1 - Renewal of the Principal-Sample-Life-Insurance
policy on Page 25 of the document.

Citations:
- Principal-Sample-Life-Insurance, Page 25 (Section D - Policy Renewal
Article 1 - Renewal)
```

3.8.2 Query 2

```
# Generate the response
response = generate_response(query_2, top_3_RAG_2)

# Print the response
print("\n".join(response))
```

```
Yes. Some insurance products require medical evidence to apply. The
documents reference a “Proof of Good Health” requirement, which
implies you may need to submit medical information or a medical report
depending on the policy. There are also notes indicating that the
insurer may request medical details in disability-related scenarios.

Cited policies and pages:
- Policy: Princi... — Page 29 (document text: "Insurance for which
Proof of Good Health is re...")
- Policy: Princi... — Page 50 (document text: "The Principal may
require that a ADL Disabled ...")
- Policy: Princi... — Page 51 (document text: "Coverage During
Disability will cease on the e...")

If you want to know the exact requirements for a specific product,
view the underwriting/medical evidence sections on those pages.
```

3.8.3 Query 3

```
# Generate the response
response = generate_response(query_3, top_3_RAG_3)

# Print the response
print("\n".join(response))
```

```
Thanks for checking. Based on the excerpts you provided, there isn’t
an explicit statement about coverage while traveling outside the
country.

What the available text does show (not specifically about
international travel):
- A note about termination of coverage for a member or dependent.
- Mentions of coverage implications during situations like business
assignments or full-time study.
- A reference to “A Member’s insurance under this Group Policy…”
without any clear travel-related detail.

Because there is no explicit travel-outside-the-country coverage
information in these snippets, you’ll want to verify directly in the
policy sections that cover international travel, out-of-country
benefits, or travel emergencies. If those sections aren’t in your
current documents, consider contacting your insurer or requesting a
rider/endorsement for travel coverage.

Citations (policy name and page numbers from the provided data):
- Principal-Sample-Life-Insuran... — Page 36
- Principal-Sample-Life-Insuran... — Page 37

If you’d like, I can help you search for specific terms like “travel,”
“out-of-country,” “international,” or “emergency medical evacuation”
within those pages to try to pinpoint any relevant provisions.
```

4. Challenges Faced

Developing this RAG solution involved several key challenges and considerations across different stages:

1. **Accurate PDF Content Extraction:**
   - **Complex Layouts:** PDFs often have varied and complex layouts, including multi-column text, images, and non-standard tables. `pdfplumber`, while powerful, can struggle with maintaining the correct reading order or accurately extracting text from highly stylized documents, leading to fragmented or jumbled content.
   - **Non-Standard Fonts and Encoding:** Documents using custom fonts or unusual text encodings can result in character misinterpretations or missing text during extraction.
   - **Embedded Objects:** Information embedded within images or non-textual objects (like scanned documents) is difficult to extract programmatically, potentially leading to loss of crucial data.
2. **Determining Effective Caching Strategies:**
   - **Similarity Threshold (`threshold`):** Choosing an optimal similarity threshold for the cache is critical. A high threshold might lead to too many cache misses, while a low threshold could result in irrelevant cached responses being returned. This requires careful empirical tuning to balance relevance and cache hit rate.
   - **Information to Store:** Deciding what information to store in the `cache_collection` (e.g., just the query and response, or also the retrieved documents, metadatas, and distances) impacts storage requirements and retrieval complexity. Storing more metadata enhances debugging and re-ranking capabilities but increases storage and potential cache invalidation overhead.
   - **Performance vs. Storage/Invalidation:** A robust caching strategy must balance the performance gains from faster responses against the storage costs and the complexity of cache invalidation (e.g., if underlying documents change).
3. **Fine-tuning the RAG Prompt for Optimal Results:**
   - **Prompt Engineering:** Crafting an effective prompt for the Large Language Model (LLM) is an iterative process. It involves clearly instructing the LLM on its role, the context provided (query and retrieved documents), expected output format, and constraints (e.g., citing sources). Over-constraining or under-constraining the prompt can lead to poor quality or irrelevant answers.
   - **Accurate Citation Generation:** Ensuring the LLM accurately cites the provided source documents (policy name and page number) can be challenging. The LLM might hallucinate citations or miss relevant ones if the prompt isn't precise enough or if the retrieved context is dense.
   - **Handling Insufficient/Ambiguous Information:** When retrieved documents don't contain a direct answer or provide ambiguous information, the LLM might struggle to provide a concise and accurate response. Fine-tuning the prompt to gracefully handle these scenarios (e.g., by stating it cannot find the information or suggesting further areas to search) is crucial for user experience.

5. Conclusion

This project successfully developed and demonstrated a Retrieval-Augmented Generation (RAG) solution designed to enable effective querying and information retrieval from complex insurance policy PDF documents. The primary achievement lies in transforming unstructured PDF content into a searchable knowledge base, allowing users to ask natural language questions and receive precise, relevant answers.

The overall effectiveness of the RAG solution is evident in its ability to provide highly contextualized answers. By combining semantic search with a reranking mechanism and a large language model, the system can understand the intent behind a user's query and retrieve the most pertinent sections from the insurance policies. A key strength of this implementation is the inclusion of verifiable citations, automatically referencing the policy name and specific page number from which the information was extracted. This feature significantly enhances the trustworthiness and utility of the generated responses, allowing users to cross-reference information directly within the source documents.

Such a system holds substantial value and potential impact, especially in domains like insurance where documents are often lengthy, complex, and critical for decision-making. It drastically reduces the time and effort required to locate specific information, improves accuracy in understanding policy details, and empowers both customers and internal stakeholders with quick, reliable access to policy knowledge. This RAG solution serves as a robust tool for navigating and extracting insights from intricate documentation.