Project Report: Generative Search System for Life Insurance Policy Document

1. Introduction

In this project, we were tasked with creating a generative search system to answer questions about a life insurance policy document. The system was designed using a layered architecture consisting of three key components: the **Embedding Layer**, the **Search Layer**, and the **Generation Layer**.

- **Embedding Layer**: This layer is responsible for processing the policy document, chunking it into manageable sections, and converting the text into numerical representations known as embeddings.
- **Search Layer**: In this layer, the document embeddings are indexed, and relevant chunks are retrieved based on user queries. The retrieved chunks are then reranked to ensure the best results are selected.
- **Generation Layer**: This final layer uses a language model to generate concise and informative answers based on the retrieved chunks.

2. Design

The system is built to answer user queries related to the content of an insurance policy document by leveraging the following architecture:

- **Embedding Layer**:
    - **Document Processing**: The life insurance policy document is first converted from its PDF format to plain text using libraries such as PyPDF2. This text is then cleaned to remove unnecessary formatting, punctuation, and stop words, ensuring that only meaningful content is processed.
    - **Chunking**: The cleaned document is divided into smaller sections or chunks, which are used for embedding. Various chunking strategies are explored:
        - **Fixed-size chunks**: The text is divided into chunks of a fixed number of tokens.
        - **Sentence-based chunking**: The document is split into individual sentences.
        - **Semantic chunking**: More advanced NLP techniques are applied to identify semantically coherent chunks based on the meaning of the content.
    - **Embedding**: A pre-trained language model like **BERT** or **RoBERTa** is used to generate embeddings for each chunk. For efficient sentence-level embeddings, **Sentence Transformers** are considered.
- **Search Layer**:

    - **Vector Database**: A vector database, such as **Faiss** or **ChromaDB**, is used to store the document embeddings. These databases facilitate fast similarity searches for retrieving relevant chunks based on the user's query.
    - **Query Embedding**: The user's query is also embedded using the same model employed for the document chunks, ensuring compatibility in the search process.
    - **Similarity Search**: A similarity search is conducted on the vector database to find the most relevant chunks of the document based on the user's query.
    - **Re-ranking**: To ensure the relevance of the retrieved chunks, a cross-encoder model like **Sentence-BERT** or **BART** is used to re-rank the results based on the query-document relevance.
- **Generation Layer**:
    - **Prompt Engineering**: A clear and concise prompt is constructed for the language model. This prompt includes:
        - The user's query.
        - The most relevant chunks retrieved from the search layer.
        - Specific instructions to guide the model on how to format the answer (e.g., concise summary, detailed explanation).
    - **Language Model**: A powerful language model, such as **GPT-3** or **Jurassic-1 Jumbo**, is used to generate the final answer. Various prompt engineering techniques are explored to improve answer quality, such as providing few-shot examples or using system messages to guide the model.

3. Implementation

- **Embedding Layer**:
    - **Document Processing**: The first step involves converting the PDF policy document into text, followed by cleaning and pre-processing to remove noise.
    - **Chunking Strategies**: The document is chunked using the aforementioned strategies, and embeddings are generated using **BERT** or **RoBERTa**.
- **Search Layer**:
    - **Vector Database Setup**: **Faiss** or **ChromaDB** is used to index the embeddings.
    - **Similarity Search and Re-ranking**: After embedding the query, the system performs similarity search and uses a cross-encoder model to re-rank the results.
- **Generation Layer**:
    - **Prompt Design**: The retrieved chunks and user query are used to create a prompt for the language model.
    - **Final Answer Generation**: The language model generates the final answer based on the constructed prompt.

4. Challenges

- **Data Preprocessing**: Extracting clean, structured text from a PDF document proved to be a significant challenge. The text often required substantial cleanup to remove irrelevant formatting, which could impact the quality of the embeddings.

- **Chunking Decisions**: Determining the most effective chunking strategy was complex. Each approach had its advantages, and finding the right balance between size and coherence for the chunks was crucial to improve both retrieval and generation performance.
- **Retriever Accuracy**: Ensuring the search layer retrieved the most relevant chunks was challenging. Fine-tuning the similarity search and reranking mechanism required multiple iterations to achieve optimal results.
- **Generation Quality**: The generated answers were sometimes verbose or lacked specificity, requiring careful tuning of the prompts and language model parameters.

5. Results and Evaluation

To evaluate the performance of the system, a series of test queries were designed based on typical questions users might ask about an insurance policy. These included questions like:

- "What is the eligibility for member life insurance?"
- "What does the policy say about accidental death and dismemberment?"
- "What is a 'Qualifying Event' under the policy?"

For each query, the following was evaluated:

- **Relevance of Retrieved Chunks**: The ability of the search layer to retrieve relevant document chunks was assessed.
- **Coherence and Accuracy of Generated Answers**: The generated answers were evaluated for their clarity, relevance, and accuracy in relation to the content of the policy document.

6. Lessons Learned

- **Importance of Clean Data**: The quality of the input data, particularly the document processing and chunking, significantly impacts the effectiveness of the entire system. Ensuring clean, structured text leads to better embeddings and more relevant search results.
- **Effective Chunking**: Fine-tuning the chunking strategy is critical for improving retrieval accuracy. Combining semantic chunking with sentence-based strategies yielded the best results.
- **Hybrid Model Strength**: The combination of retrieval and generation (RAG) proved to be highly effective in answering complex queries by grounding the answers in specific document content.
- **Scalability Considerations**: As the corpus grows, ensuring efficient storage and retrieval of embeddings becomes essential. Optimizing the vector database and retrieval algorithms is crucial for maintaining speed and accuracy.

7. Conclusion

This project demonstrated the value of a generative search system for answering queries about life insurance policy documents. By combining retrieval with generation, we were able to significantly improve the relevance and accuracy of answers compared to purely generative or purely retrieval-based approaches. Despite the challenges faced during implementation, the RAG architecture showed great promise in answering knowledge-intensive questions effectively.