Project Report: Generative Search System for Life Insurance Policy Document

1. Introduction

This project implements a retrieval-augmented generative (RAG) search system designed to answer user queries about a life insurance policy document. The system is structured into three key layers:

- **Embedding Layer**: Processes the document, chunks it into smaller units, and converts the chunks into numerical embeddings.
- **Search Layer**: Retrieves the most relevant chunks based on the user query using a vector database and reranks them for relevance.
- **Generation Layer**: Generates accurate, concise answers using a language model based on the retrieved chunks.

2. System Design

The architecture of the generative search system incorporates the following components:

- **Embedding Layer**:
  - **Document Processing**: The life insurance policy document is converted from PDF to plain text using `PyPDF2`. The text is cleaned to normalize whitespace and remove unnecessary formatting.
  - **Chunking**: The cleaned text is divided into smaller sections for embedding, with the following strategies:
    - **Fixed-size chunks**: Splits text into chunks of a predefined number of tokens.
    - **Sentence-based chunking**: Breaks the document into individual sentences.
    - **Semantic chunking**: Placeholder logic to split text based on semantic coherence.
  - **Embedding**: A pre-trained SentenceTransformer model (`all-MiniLM-L6-v2`) is used to generate embeddings for each chunk.
- **Search Layer**:
  - **Vector Database**: Uses ChromaDB or FAISS to index the embeddings and facilitate similarity searches.
  - **Query Embedding**: The user query is converted into an embedding using the same model for compatibility.
  - **Similarity Search**: Searches the vector database for chunks most relevant to the user query.
  - **Re-ranking**: Applies a cross-encoder model (e.g., Sentence-BERT) to refine the relevance of retrieved chunks.
- **Generation Layer**:
  - **Prompt Engineering**: Constructs a prompt for the language model, including:
    - User query.
    - Relevant chunks retrieved from the search layer.

        - Instructions to guide the language model (e.g., concise summaries).
    - **Language Model**: Uses advanced models (e.g., OpenAI GPT or similar) to generate final answers.

3. Implementation Details

- **Embedding Layer**:
    - Document processing uses `PdfReader` to extract text, which is cleaned and normalized.
    - Text chunking is implemented with fixed-size and sentence-based strategies, leaving room for semantic improvements.
    - Embeddings are generated using a pre-trained SentenceTransformer.
- **Search Layer**:
    - A vector database is initialized using ChromaDB or FAISS.
    - Similarity searches are conducted to retrieve document chunks most relevant to user queries.
    - Retrieved chunks are reranked for improved relevance using a cross-encoder.
- **Generation Layer**:
    - Prompts are crafted by combining user queries and retrieved chunks with detailed instructions.
    - The language model generates the final answer.

4. Challenges

- **Data Preprocessing**: Extracting structured text from the PDF required significant cleaning to ensure quality embeddings.
- **Chunking Strategies**: Balancing the size and coherence of text chunks was critical for effective retrieval and generation.
- **Search Accuracy**: Achieving high precision in retrieving the most relevant chunks required iterative fine-tuning of the similarity search and reranking processes.
- **Generation Quality**: The output quality depended heavily on prompt design and model parameters.

5. Results and Evaluation

Test queries were created to evaluate the system, such as:

- "What are the eligibility criteria for member life insurance?"
- "What benefits are provided under accidental death and dismemberment coverage?"
- "Define a 'Qualifying Event' as per the policy."

Performance was assessed based on:

- **Relevance of Retrieved Chunks**: Evaluated the search layer's ability to fetch relevant document sections.

- **Coherence and Accuracy of Generated Answers**: Assessed the clarity and correctness of responses.

6. Lessons Learned

- **Clean Data Improves Results**: Preprocessing and structuring the text significantly impacted the effectiveness of retrieval and generation.
- **Effective Chunking is Key**: Combining sentence-based and semantic chunking approaches yielded optimal results.
- **RAG Architecture is Powerful**: The combination of retrieval and generation proved highly effective for answering complex questions.
- **Scalability Considerations**: Efficient storage and retrieval mechanisms are essential as the document corpus scales.

7. Conclusion

This project demonstrates the potential of a generative search system for querying policy documents. The layered architecture integrates retrieval and generation to provide relevant, accurate, and concise answers. Despite challenges in preprocessing and chunking, the RAG approach successfully addresses knowledge-intensive queries, showcasing its practical value in the insurance domain.