## **Introduction**

This project focuses on developing a generative search system capable of answering questions from a life insurance policy document.

Traditional keyword-based search often fails to capture the semantic meaning of queries, leading to incomplete or irrelevant results. To overcome this, the system leverages embeddings, vector search, re-ranking, and large language models (LLMs) to provide more accurate and context-aware answers.

By combining these components, the system ensures that information from lengthy and complex policy documents can be retrieved and presented in a user-friendly, natural language format.

This not only improves accessibility but also enables quick decision-making for users seeking clarity from dense legal or insurance texts.

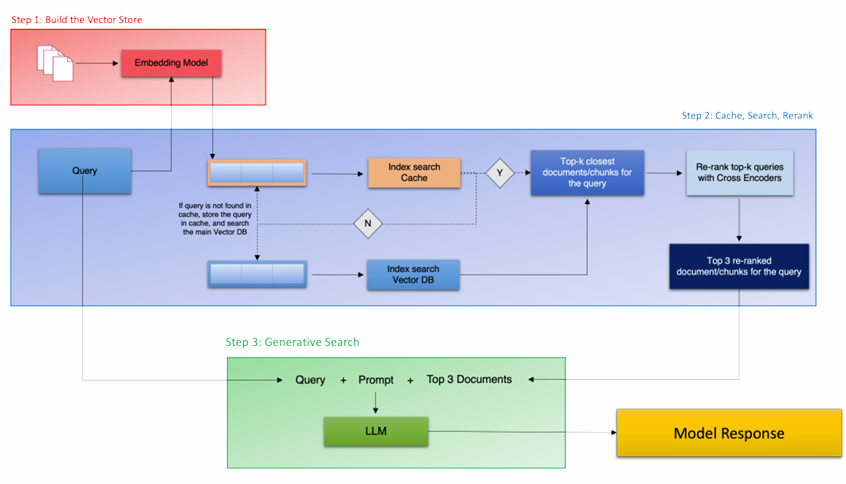
## **Approach / Workflow**

The workflow of the project is structured into three main layers:

1. **Embedding Layer**
   * Process, clean, and chunk the policy document.
   * Experiment with fixed-size chunking strategies.
   * Generate embeddings using models such as OpenAI embeddings or SentenceTransformers.
2. **Search Layer**
   * Formulate at least 3 queries based on the policy document.
   * Embed the queries and perform semantic search using ChromaDB.
   * Implement a cache mechanism for efficiency.
   * Apply a re-ranking step using cross-encoder models to improve retrieval accuracy.
3. **Generation Layer**
   * Design a comprehensive prompt to guide the LLM.
   * Pass the retrieved results to the LLM for final answer generation.
   * Optionally, include few-shot examples to enhance response quality.

## **Detailed Methodology**

1. **Reading & Processing PDF File**
   * The policy document is read using **pdfplumber**, which provides robust parsing capabilities.
   * Unlike basic text extractors, pdfplumber can handle text, tables, and images, offering more accurate preprocessing.
   * It also supports visual debugging, making it easier to validate the extracted content.
2. **Document Chunking**
   * Since the policy document is lengthy, the text is split into smaller chunks before embedding.
   * We begin with a **fixed-size chunking strategy**, but further experimentation with overlap or semantic chunking can improve retrieval quality.
3. **Generating Embeddings**
   * Each chunk is converted into vector representations using the **SentenceTransformer “all-MiniLM-L6-v2” model**.
   * These embeddings capture the semantic meaning of the text, enabling more relevant retrieval.
4. **Storing Embeddings in ChromaDB**
   * The generated embeddings are stored in **ChromaDB**, a vector database optimized for similarity search.
   * This enables efficient retrieval of relevant chunks based on user queries.
5. **Semantic Search with Cache**
   * User queries are embedded and searched against the database.
   * A **cache layer** is added to speed up repeated queries and reduce computational cost.
6. **Re-Ranking with a Cross Encoder**
   * Initial retrieval results are refined using a **cross-encoder model**, which evaluates query–response pairs.
   * This improves the relevance and ranking of the final results.
7. **Retrieval-Augmented Generation (RAG)**
   * The top-ranked chunks are combined with the user query and passed into GPT-3.5 (or any chosen LLM).
   * A carefully designed prompt ensures the model generates accurate, well-structured, and context-aware answers.
   * Optionally, few-shot examples can be added to further guide the generation process.



## **Technical Setup**

**Language Model**: Perplexity AI (*sonar-pro*) accessed via OpenAI client wrapper.

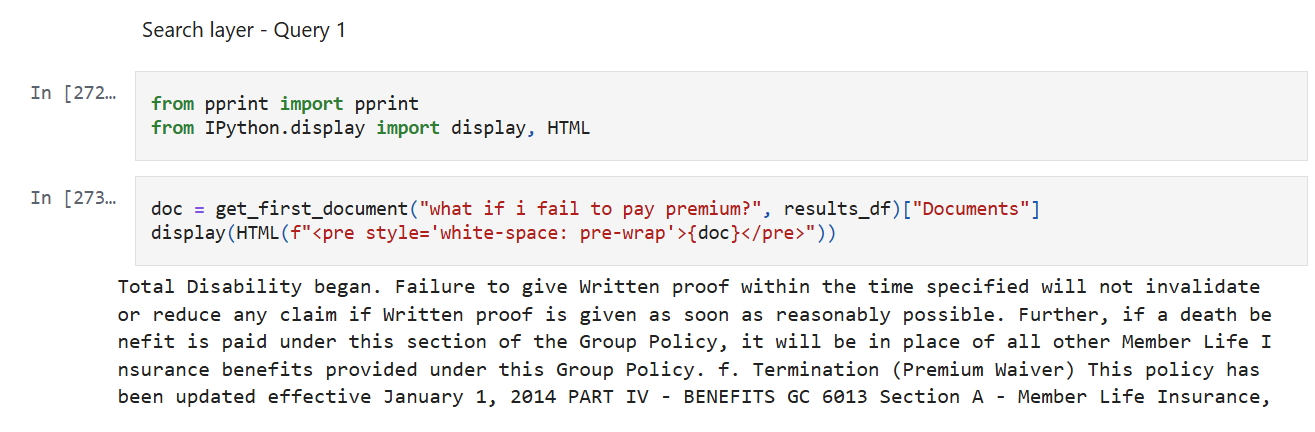
**Programming Language**: Python.

**Interface**: Jupyter Notebook (real-time user input handled via input()).

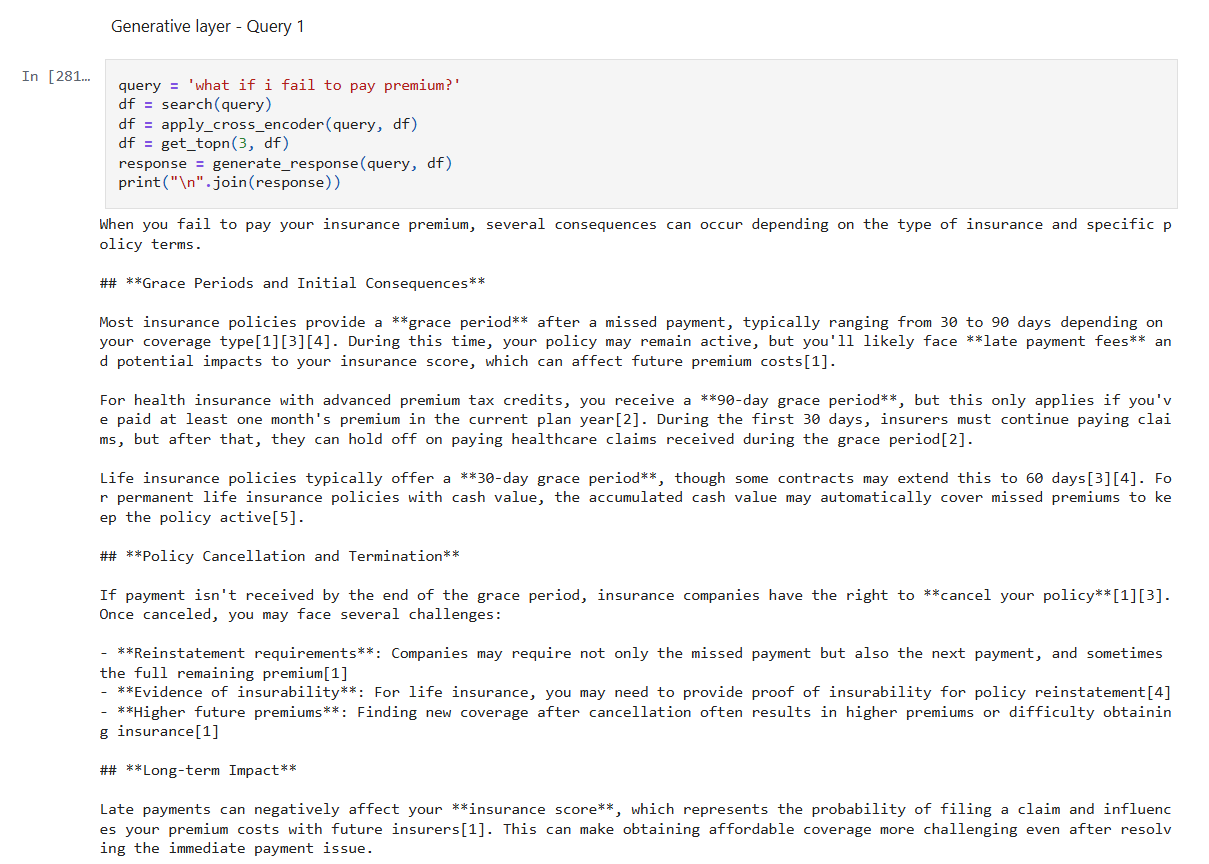
**Results**

**Query 1 : what if i fail to pay premium?**

**Search Layer O/P**

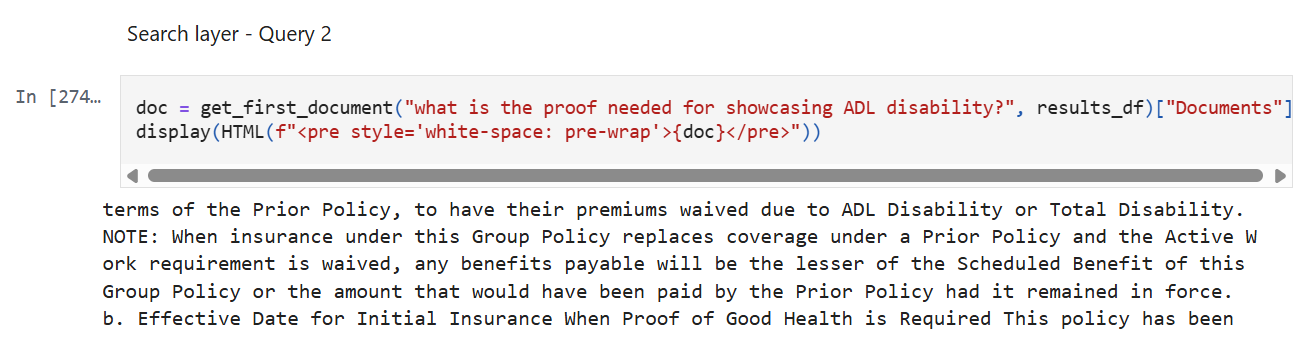
****

**Generation Layer O/P**

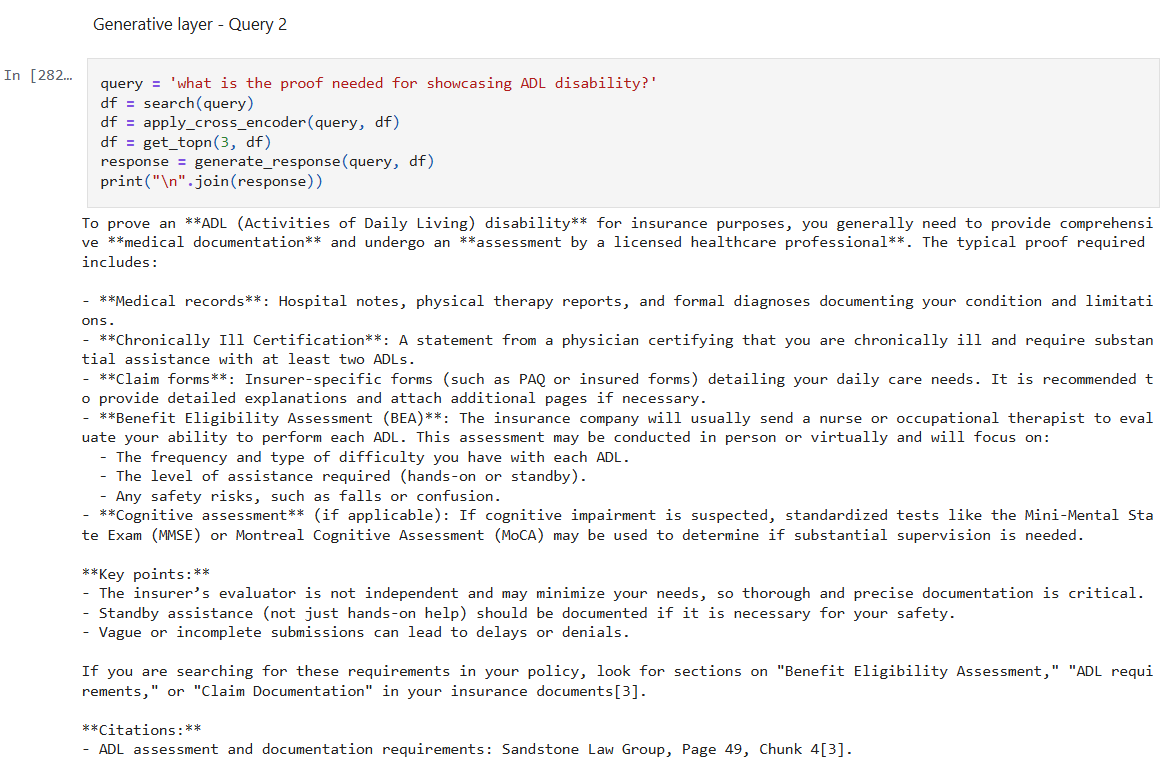
****

**Query 2 : what is the proof needed for showcasing ADL disability?**

**Search Layer O/P**

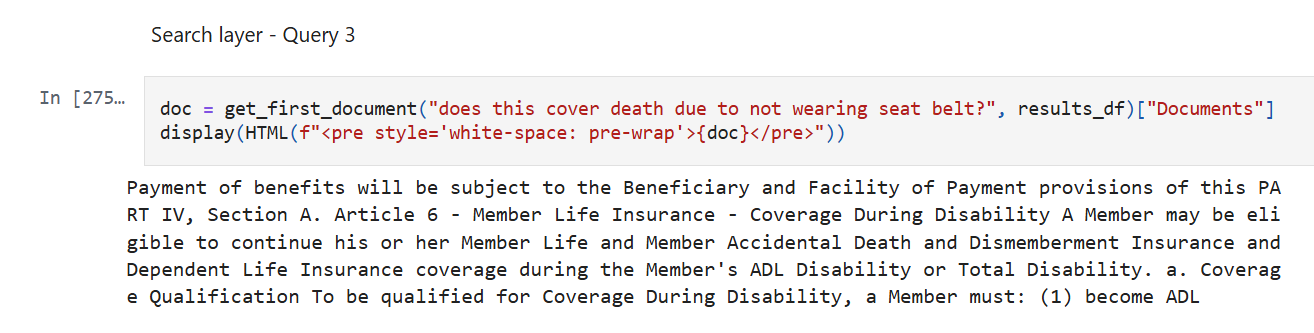
****

**Generation Layer O/P**

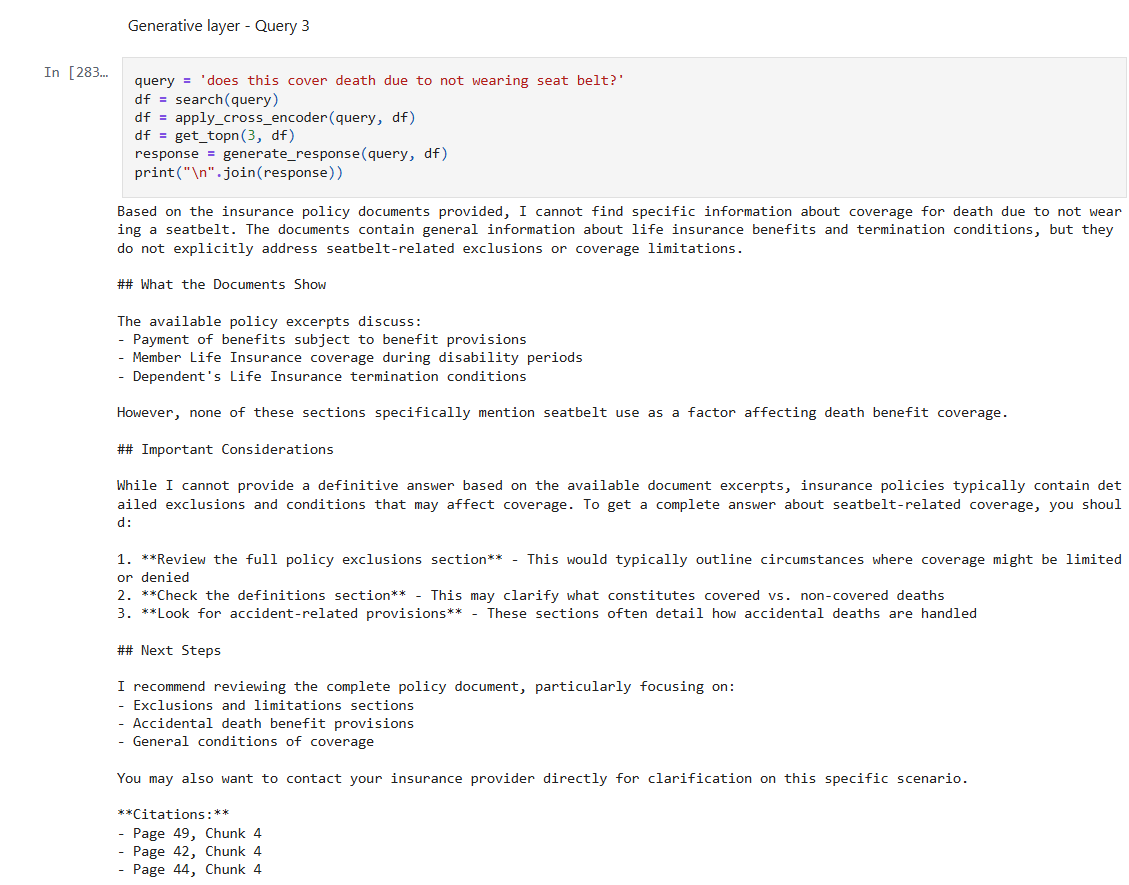
****

**Query 3 : does this cover death due to not wearing seat belt?**

**Search Layer O/P**

****

**Generation Layer O/P**

****

## **Challenges Faced and Possible Improvements**

**Challenges Faced**

1. Selecting the right chunking strategy was critical, as poor chunking reduced retrieval accuracy.
2. Ensuring relevant results during semantic search sometimes required tuning and experimentation with different models.
3. Re-ranking introduced additional computation, slightly impacting system latency.

**Possible Improvements**

1. Experiment with advanced chunking techniques (semantic or overlap-based) for better context preservation.
2. Use larger or domain-specific embedding models to improve semantic representation.
3. Optimize the caching mechanism to handle more queries efficiently.
4. Fine-tune prompts or use few-shot / chain-of-thought prompting for better generation quality.
5. Integrate a user-friendly interface (e.g., web app) for broader accessibility beyond Jupyter Notebook.