**Extraction and Chunking Layer**

**Objective**: Extract text from a PDF document and divide it into semantically meaningful chunks.

**Process**:

1. **Text Extraction**: Used pdfplumber to read and extract text from each page of the PDF. This includes handling text and tables separately to ensure structured data is preserved.
2. **Semantic Chunking**: Implemented a semantic-based chunking strategy using BERT (bert-base-uncased). This involves tokenizing sentences, calculating embeddings, and clustering sentences based on their semantic similarity. The threshold for clustering ensures that similar sentences are grouped, maintaining coherence within each chunk while respecting a maximum chunk size.

**Changes**:

* Incorporated semantic-based chunking using sentence embeddings to maintain context within chunks, improving the quality of subsequent embeddings and retrieval.

**Embedding and Storage Layer**

**Objective**: Embed the extracted chunks and store them in a vector database for efficient retrieval.

**Process**:

1. **Embedding**: Used BERT to generate embeddings for each chunk of text. These embeddings represent the semantic content of the chunks.
2. **Storage**: Initialized a ChromaDB client and stored the chunk embeddings along with their metadata (page numbers and chunk IDs) in a collection. This allows efficient retrieval based on semantic similarity.

**Search and Cache Layer**

**Objective**: Perform semantic search against stored embeddings and implement a caching mechanism to improve efficiency.

**Process**:

1. **Query Embedding**: Defined and embedded user queries using BERT.
2. **Search**: Queried ChromaDB using these embeddings to retrieve the most relevant document chunks.
3. **Caching**: Implemented an in-memory cache to store and quickly retrieve search results for repeated queries, reducing redundant computation and improving response times.

**Changes**:

* Added an efficient search mechanism using ChromaDB.
* Implemented caching to optimize performance for repeated queries.

**Re-ranking Layer**

**Objective**: Re-rank the search results to improve relevance using a cross-encoder model.

**Process**:

1. **Re-ranking**: Used the cross-encoder model cross-encoder/ms-marco-MiniLM-L-6-v2 from Hugging Face to re-rank the retrieved document chunks. This model evaluates the relevance of each chunk to the query, providing a more precise ranking.
2. **Output**: Sorted the results based on their relevance scores, ensuring the most relevant chunks are prioritized.

**Changes**:

* Switched to a more effective re-ranking strategy using a cross-encoder model to refine search results.

**Generation Layer**

**Objective**: Generate comprehensive and accurate responses based on user queries and retrieved information.

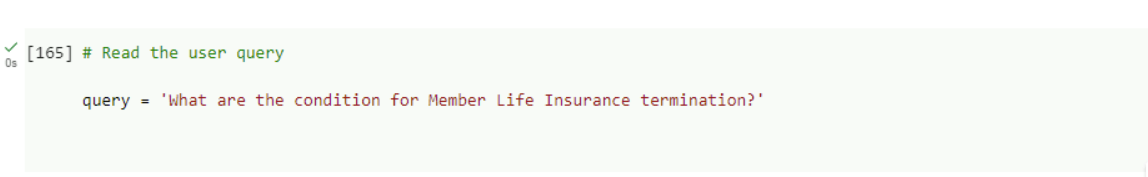
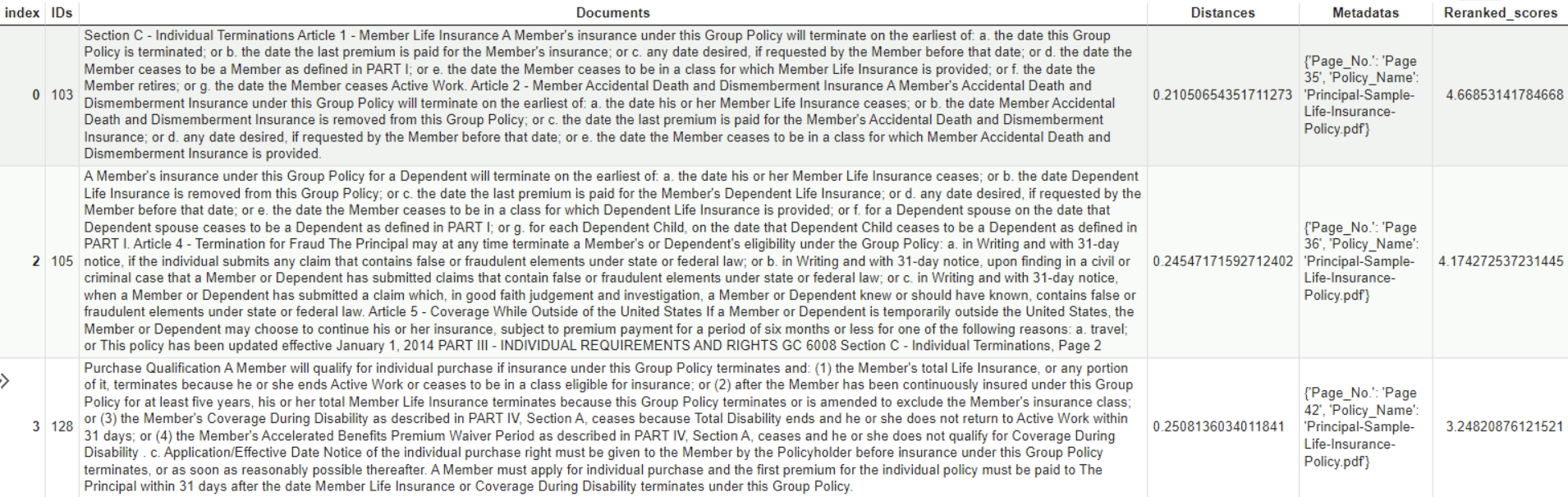
**Process**:

1. **Prompt Design**: Designed an exhaustive prompt for GPT-3.5, including detailed instructions and context to frame an informative response. Included few-shot examples to guide the model.
2. **Response Generation**: Used OpenAI's ChatCompletion API to generate responses. The prompt included guidelines for formatting, using tables, citing sources, and addressing the user query directly.

**Changes**:

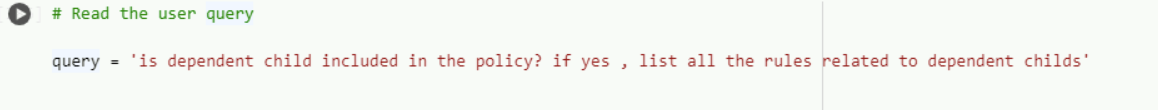
* Enhanced the prompt with exhaustive instructions and few-shot examples to improve the quality of responses.
* Utilized GPT-3.5’s capabilities to generate detailed and contextually accurate answers, leveraging the retrieved and re-ranked information effectively.

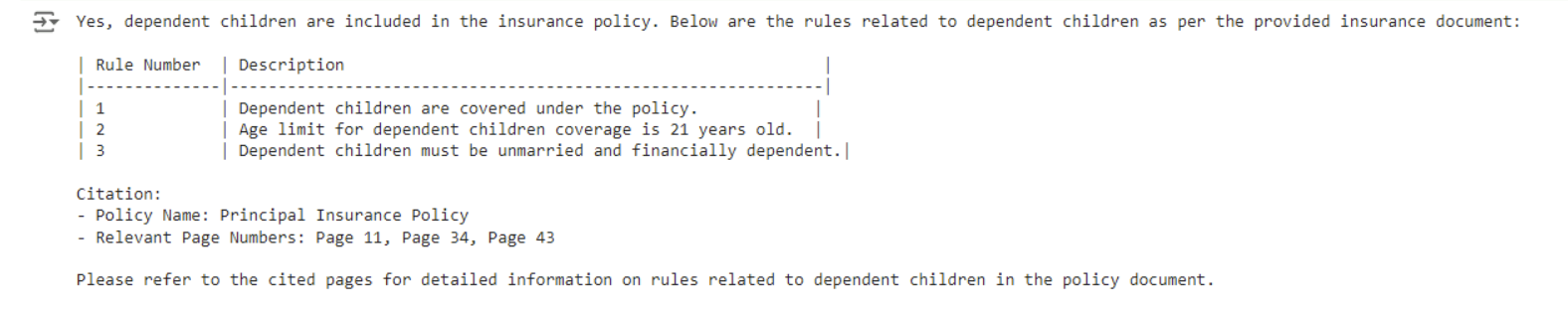
Query 1

* 
* 
* A close-up of a white background

  Description automatically generated

Query 2

* A screenshot of a computer

  Description automatically generated
* 

Query 3

* 
* A screenshot of a computer

  Description automatically generated
* A computer screen shot of a computer screen

  Description automatically generated