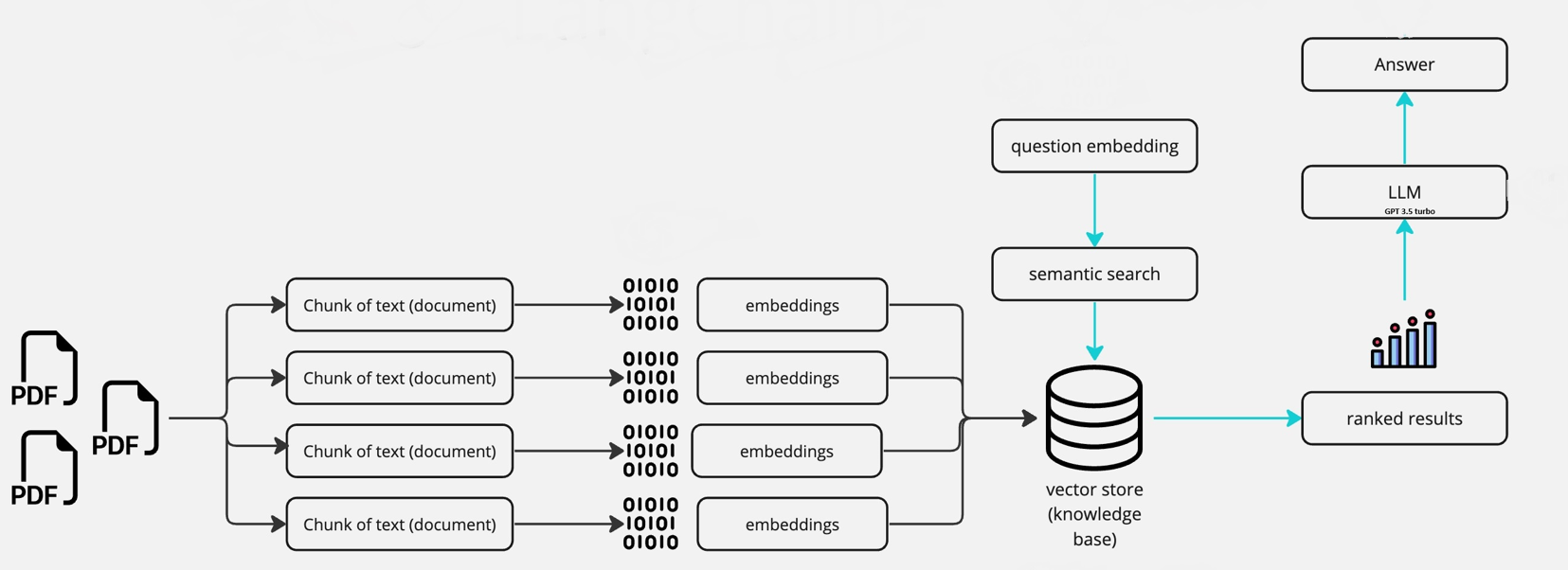
# ESG Survey Response Automation



## **Design Approach Diagram for PDF Processing and Answer Generation**

Key Points before moving to design approach explanation: -

* **Vector Store**: The Vector Store is a crucial component that enables efficient retrieval of information from the PDF documents. It serves as a repository for storing vector representations of the text chunks extracted from the PDFs. These vector representations are generated using language models and embeddings.
* **Embeddings**: Embeddings play a significant role in the chatbot’s architecture. They capture the semantic and contextual information of the text chunks and enable meaningful comparison and retrieval. Embeddings are generated by feeding the text chunks into pre-trained language models or embeddings models, such as OpenAI models or Hugging Face models. These models encode the textual information into dense vector representations.
* **Large Language Models (LLMs):** LLMs are a crucial component in the chatbot architecture. They provide the chatbot with advanced language generation and understanding capabilities. LLMs are pre-trained models that have learned to understand and generate human-like text. By fine-tuning these models on specific tasks or domains, such as conversational retrieval, the chatbot can engage in more interactive and context-aware conversations.

Steps for Pdf Processing and Embeddings: -

1. **PDF Processing**: -

* Upon uploading PDF files, the application reads the PDF content and splits it into chunks of size 2000 characters with an overlap of 200 characters. The PyPdf2 library is utilized for PDF processing. After extracting each chunk, an object is created with the following attributes: [chunk, name, page\_number+1, token\_length, year\_of\_report]. The page number and name are crucial for citations.
* **Design Objective**: The aim is to break down large textual data into smaller, manageable chunks to facilitate efficient processing and analysis.
* **Approach:** The function adopts a modular approach by encapsulating the logic for text segmentation. It utilizes a configurable text splitter utility, RecursiveCharacterTextSplitter, to perform the segmentation. The splitTextIntoChunks function defines parameters such as chunk size and overlap to control the segmentation process, enhancing flexibility and adaptability. By segmenting the text into chunks of 1000 characters each, the function ensures that the processing load is distributed evenly and that memory usage remains within manageable limits.

**Benefits:**

* Improved Processing Efficiency: Breaking down the text into smaller units enables parallel processing and reduces the computational burden.
* Memory Optimization: Limiting the size of each chunk helps prevent memory overflow and enhances system stability during processing.

1. **Create Embeddings of Chunks and storing in Vector Store: -**

Next, the application establishes a collection dedicated to storing PDF-related data, defining specific field schemas, and creating an index on the embedding field to facilitate efficient similarity searches. This structured approach streamlines data storage and retrieval processes, ensuring quick access to text chunks, metadata, and embeddings. By leveraging Milvus as the vector database, the application benefits from optimized storage and retrieval of high-dimensional embeddings, enhancing search capabilities and overall performance.

The process involves storing text chunks, metadata, and embeddings in the designated collection. Each text chunk undergoes embedding generation using OpenAI's Embedding API, employing the "text-embedding-ada-002" model tailored for text embedding tasks. By ensuring that the token limit is not exceeded through careful calculation of tokens per chunk, the application mitigates rate limit errors and reduces costs associated with OpenAI service usage. This proactive approach maintains efficient operation while minimizing expenses, contributing to cost-effectiveness and resource optimization.

Benefits of utilizing Milvus for vector storage and retrieval include:

* **Efficient High-Dimensional Vector Storage**: Milvus excels in storing and indexing high-dimensional vectors efficiently, enabling quick retrieval and similarity searches.
* **Scalability**: Milvus is designed to handle large-scale datasets and high-dimensional embeddings, making it suitable for applications with growing data volumes.
* **Real-Time Search**: The indexing capabilities of Milvus allow for real-time similarity search operations, enhancing responsiveness and user experience.
* **Flexibility**: Milvus supports various indexing methods and configurations, providing flexibility to adapt to different use cases and requirements.
* **Cost-Effective**: By optimizing storage and retrieval processes and minimizing resource usage, Milvus helps reduce overall operational costs, contributing to cost-effectiveness and budget management.

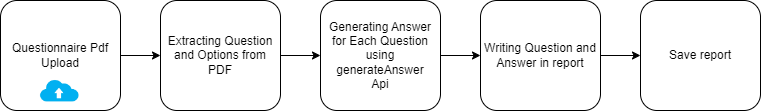
In summary, integrating Milvus as the vector database enhances the application's performance, scalability, and cost-effectiveness, ensuring efficient storage, retrieval, and analysis of PDF-related data and embeddings.

**Steps for Generating Answer for a Question:-**

* **User Question Reception:** Upon receiving a user question, the system initiates the process of generating an answer.
* **Embedding Creation:** The user question is converted into embeddings using a predefined model. These embeddings capture the semantic representation of the question.
* **Similarity Search:** Utilizing the Milvus vector database, the system performs a similarity search on a collection of PDF documents.
* The search retrieves the top 5 most relevant documents based on their similarity to the user question.
* **Document Retrieval:** Each retrieved document contains crucial information such as the content of the text chunk, the file name of the PDF document, and the page number where the text chunk is located.
* **Prompt Construction:** Prompts are constructed for the ChatGPT model, incorporating the content, file name, and page number of the top 5 documents. These prompts provide contextual information to the model for generating the answer.
* **Answer Generation:** The ChatGPT model, specifically the GPT-3.5-turbo variant, is employed to generate the answer. The prompts, along with the user question, are provided to the model to facilitate answer generation.
* **Efficiency and Relevance:** Limiting the results to the top 5 most similar documents ensures efficiency and focuses on the most pertinent information for generating the answer. By combining similarity search with advanced natural language processing capabilities, the system delivers informative and relevant responses to user queries regarding the PDF documents.

## 

## Architecture Diagram



## Steps for Report Generation

**Here are the high-level user stories for each of the 6 APIs:**

**SurveyQuestionnaireUpload API:**

* As a user, I want to upload a survey questionnaire document.
* As a user, I want to specify the type and name of the document being uploaded.
* As a user, I want to provide metadata such as the year for which the report needs to be generated and my user ID.
* After uploading the document, I expect to receive a unique task ID and confirmation of successful upload.

**ReportStatus API:**

* As a user, I want to check the status of a generated report.
* I should be able to specify the year of the report and the task ID to retrieve the status.
* The API should indicate whether the report is in progress or completed.
* It should provide the creation timestamp of the report.

**ReportPDF API:**

* As a user, I want to download a generated PDF report.
* I should be able to specify the year of the report.
* The API should retrieve the PDF file associated with the specified year and provide it as a downloadable response.

**ESGUploadView API:**

* As a user, I want to upload ESG (Environmental, Social, Governance) reports in PDF format.
* I should be able to specify the year of the report and provide URLs to the document.
* Upon successful upload, I expect to receive a confirmation message along with a unique tracker ID.

**AnswerAPIView API:**

* As a user, I want to get answers to questions related to uploaded PDF reports.
* I should be able to provide the question and specify the report year.
* The API should return the answer along with any citations if available.

**ListReportAPIView API:**

* As a user, I want to list uploaded reports for a specific year.
* I should be able to specify the year of the reports I want to list.
* The API should return a list of documents uploaded for the specified year, including document names, types, reference links, and other metadata.