**Project Documentation**

**PDF Query System using LangChain, OpenAI Embeddings, and Cassandra AstraDB**

**1. Introduction**

The integration of artificial intelligence with natural language processing has evolved rapidly, allowing machines to read, interpret, and reason over human-written documents. Among the most significant advancements is the ability to perform **semantic search** and **retrieval-augmented question answering (RAG)**, where systems can understand context rather than rely purely on keyword matches. This project — *PDF Query with LangChain and Cassandra AstraDB* demonstrates how such a system can be implemented end-to-end.

The goal of the project is simple yet powerful: **to read any PDF document and answer natural-language questions about its content automatically**. By combining **LangChain**, **OpenAI embeddings**, and **Cassandra AstraDB**, the system creates a scalable, efficient, and intelligent pipeline capable of interpreting unstructured text. It transforms the static data inside a PDF into a searchable, query-ready knowledge base.

At its core, this project illustrates how **large language models (LLMs)** can be augmented with **external knowledge sources** for precise, context-driven responses. Unlike traditional chatbots that rely solely on pre-trained data, this solution reads from the user’s own documents, converting them into **vector embeddings** — numerical representations of meaning. These embeddings are stored in a **vector database**, enabling fast similarity search when a user poses a query.

This documentation explains every step of the process: how the text is extracted from PDFs, chunked into sections, embedded via OpenAI’s models, stored in Cassandra AstraDB, and finally retrieved and answered using LangChain’s question-answering framework.

**2. Objectives**

The main objectives of the PDF Query with LangChain and Cassandra AstraDB project are as follows:

1. **Text Extraction:** Read and clean raw textual data from PDF files using PyPDF2.
2. **Text Chunking:** Divide large text blocks into manageable chunks suitable for token-based processing in language models.
3. **Embedding Generation:** Convert these chunks into dense vector embeddings using OpenAI’s embedding model.
4. **Vector Storage:** Store these embeddings inside a Cassandra AstraDB vector table using CassIO integration.
5. **Semantic Search:** Retrieve the most relevant document chunks through vector similarity comparison.
6. **Question Answering:** Use a Large Language Model (LLM) to formulate natural-language answers based on retrieved text.
7. **Interactive Interface:** Provide a command-line interface that allows users to ask follow-up questions in real time.

Each objective plays a crucial role in forming a complete pipeline from document ingestion to dynamic information retrieval and contextual response generation.

**3. Tools and Technologies**

This project integrates several modern tools and libraries, each playing a distinct role in the system’s overall architecture.

| **Technology** | **Purpose / Description** |
| --- | --- |
| **PyPDF2** | Extracts textual data from PDF documents page by page. |
| **LangChain** | Orchestrates the interaction between the LLM, embeddings, and vector database. |
| **OpenAI API** | Generates embeddings and provides the LLM used for answering questions. |
| **CassIO** | Python integration layer that connects LangChain to Cassandra AstraDB. |
| **AstraDB (Cassandra)** | Cloud-native NoSQL database used as a persistent vector store. |
| **CharacterTextSplitter** | Utility from LangChain to divide long text into smaller overlapping chunks. |
| **Python** | The main programming language for building and running the pipeline. |

**4. Conceptual Overview**

Traditional keyword search systems match words literally, often missing semantically similar ideas expressed differently. This project overcomes that limitation by using **embeddings**, where text is represented as high-dimensional vectors based on meaning rather than syntax.

When a user uploads a PDF, the system:

1. Extracts and cleans its text content.
2. Splits the text into logical segments or “chunks.”
3. Embeds those chunks using OpenAI’s vector model.
4. Stores them in Cassandra AstraDB for efficient retrieval.
5. At query time, the user’s question is also embedded.
6. A similarity search identifies the top-matching chunks.
7. Those chunks are passed to the LLM, which formulates a coherent, context-aware answer.

This structure embodies the **Retrieval-Augmented Generation (RAG)** paradigm — enhancing the reasoning power of LLMs with specific, factual information retrieved from external data sources.

**5. Workflow and Architecture**

The project follows a modular architecture consisting of five primary stages:

1. **PDF Text Extraction**
2. **Text Splitting**
3. **Embedding Creation**
4. **Vector Storage and Retrieval**
5. **Interactive Question-Answering Loop**

Let’s examine each stage in detail.