# PDF Question Answering System - Technical Documentation

## 1. System Overview

The PDF Question Answering System is designed to extract text from PDF documents, create searchable vector stores, and answer questions based on the content of the PDFs. It uses a hybrid search approach combining vector search and BM25, and leverages language models for generating answers.

## 2. Key Components

### 2.1 PDF Processing

* Location: `src/document\_processing/pdf\_extractor.py`
* Main Function: `extract\_text\_from\_pdf()`
* Description: Uses PyMuPDFLoader to extract text from PDF files. The text is split into chunks with a specified size and overlap.

### 2.2 Vector Store Creation

* Location: `src/document\_processing/vector\_store.py`
* Main Function: `create\_or\_load\_vector\_store()`
* Description: Creates or loads Chroma vector stores and BM25 retrievers for PDF documents. Uses HuggingFace embeddings for vector representation.

### 2.3 Embedding Model

* Location: `src/embeddings/embedding\_model.py`
* Main Function: `get\_embeddings()`
* Description: Utilizes the "sentence-transformers/paraphrase-mpnet-base-v2" model to generate embeddings for text.

### 2.4 Hybrid Search

* Location: `src/search/hybrid\_search.py`
* Main Functions: `hybrid\_search()`, `process\_questions()`
* Description: Implements a hybrid search combining vector search (Chroma) and BM25. Processes questions by retrieving relevant contexts and generating answers using a language model.

### 2.5 API

* Location: `main\_api.py`
* Description: Implements a FastAPI application providing endpoints for PDF upload and querying.

## 3. Workflow

1. PDF Upload:

- User uploads a PDF through the API.

- System extracts text and creates vector stores (Chroma and BM25).

- Stores are persisted for future use.

2. Querying:

- User sends a query or multiple queries through the API.

- System performs hybrid search to retrieve relevant contexts.

- Retrieved contexts are combined and sent to the language model.

- Language model generates an answer.

- System determines the most relevant page for the answer.

## 4. Key Algorithms and Methods

### 4.1 Text Chunking

* Splits PDF text into overlapping chunks for better context preservation.
* Chunk size and overlap are configurable.

### 4.2 Hybrid Search

* Combines results from vector search (Chroma) and BM25.
* Scores from both methods are normalized and combined.
* Top K results are returned based on combined scores.

### 4.3 Answer Generation

* Uses Ollama language model to generate answers based on retrieved contexts.
* Implements error handling for LLM calls.

### 4.4 Page Determination

* Uses Jaccard similarity to match the generated answer with the most relevant context.
* Returns the page number of the highest similarity context.

## 5. Data Structures

### 5.1 PDF Store

* In-memory dictionary storing information about uploaded PDFs.
* Key: PDF ID
* Value: Dictionary containing file path, Chroma DB, BM25 retriever, and BM25 texts.

### 5.2 Vector Stores

* Chroma: Persistent vector store for efficient similarity search.
* BM25: In-memory index for keyword-based retrieval.

## 6. API Endpoints

### 6.1 PDF Upload

* Endpoint: `/upload\_pdf`
* Method: POST
* Input: PDF file (multipart/form-data)
* Output: PDF ID and status message

### 6.2 Single Query

* Endpoint: `/query`
* Method: POST
* Input: JSON with query and PDF ID
* Output: Answer, relevant page number

### 6.3 Multiple Queries

* Endpoint: `/multiple\_queries`
* Method: POST
* Input: JSON with list of queries and PDF ID
* Output: List of answers with relevant page numbers

## 7. Error Handling

* Implements try-except blocks for major operations.
* Returns meaningful error messages for API calls.
* Handles LLM errors gracefully.

## 8. Performance Considerations

* Uses efficient vector search techniques for fast retrieval.
* Implements persistence for vector stores to avoid recomputation.
* Utilizes chunking to balance between context preservation and search efficiency.