Architecture Document: Enterprise Document Query System

Date: March 26, 2025

1. Overview

The Enterprise Document Query System is a web-based application designed to allow users to upload PDF documents, index their contents, and query them using natural language. It leverages vector embeddings for semantic search and an LLM (Large Language Model) for interpreting results, providing concise and relevant answers. The system is built using Python, Streamlit for the frontend, and integrates several third-party libraries for document processing, embedding generation, vector search, and LLM inference.

Purpose

- Enable users to upload enterprise PDF documents.
- Index document contents for efficient retrieval.
- Allow natural language queries to extract and interpret relevant information.

Key Features

- PDF upload and text extraction.
- Semantic search using vector embeddings.
- LLM-powered interpretation of search results.
- Interactive web interface.

2. System Architecture

2.1 High-Level Architecture

The system follows a modular architecture with the following layers:

1. **Frontend Layer**: Streamlit-based UI for user interaction.
2. **Application Logic Layer**: Python code handling document processing, indexing, search, and LLM integration.
3. **Data Processing Layer**: Libraries for PDF parsing, embedding generation, and vector storage.
4. **External Services Layer**: OpenAI API for LLM inference.

text
CollapseWrapCopy

```
[Uploaded PDFs]     [FAISS Index + Embeddings]
```

2.2 Components

2.2.1 Frontend (Streamlit UI)

- **Purpose**: Provides a web interface for uploading PDFs and querying the system.
- **Technologies**: Streamlit.
- **Features**:
  - File uploader for multiple PDFs.
  - Text input for queries.
  - Display of search results and status messages.

2.2.2 Application Logic

- **Purpose**: Orchestrates the workflow from PDF upload to query response.
- **Key Functions**:
  - process_pdf: Extracts text from PDFs.
  - index_documents: Generates embeddings and builds the FAISS index.
  - search_documents: Performs vector search and invokes LLM interpretation.
  - interpret_with_llm: Queries the OpenAI API for result interpretation.
- **Technologies**: Python 3.13.

2.2.3 Data Processing

- **PDF Parsing**:
  - **Library**: PyPDF2 (with potential migration to pypdf).
  - **Role**: Extracts text from uploaded PDFs.
- **Embedding Generation**:
  - **Library**: Sentence Transformers (all-MiniLM-L6-v2 model).
  - **Role**: Converts document text and queries into 384-dimensional vector embeddings.
- **Vector Storage and Search**:
  - **Library**: FAISS (FlatL2 index).
  - **Role**: Stores embeddings and performs nearest-neighbor search.

2.2.4 External Services

- **OpenAI API**:

    - **Purpose**: Provides LLM capabilities (GPT-3.5-turbo) for interpreting search results.
    - **Interaction**: HTTP requests via the openai Python client.

2.2.5 File System

- **Purpose**: Stores uploaded PDFs temporarily in the pdfs/ directory.
- **Interaction**: Managed by the application logic for reading and writing.

2.3 Data Flow

1. **PDF Upload**:
    - User uploads PDFs via Streamlit UI.
    - Files are saved to the pdfs/ directory.
2. **Indexing**:
    - PDFs are processed to extract text (process_pdf).
    - Text is converted to embeddings (SentenceTransformer).
    - Embeddings are stored in a FAISS index (index_documents).
3. **Query Processing**:
    - User submits a query via the UI.
    - Query is converted to an embedding (model.encode).
    - FAISS searches for the nearest document embedding (search_documents).
    - Relevant document text is sent to OpenAI for interpretation (interpret_with_llm).
4. **Response**:
    - LLM response is displayed in the UI.

text
CollapseWrapCopy

```
[Query] --> [Embedding] --> [FAISS Search] --> [Document Text] --> [OpenAI API] --> [Response] --> User
```

3. Technologies

- **Programming Language**: Python 3.13.
- **Frontend**: Streamlit 1.x (specific version TBD based on compatibility).
- **Libraries**:
    - PyPDF2: PDF text extraction.
    - sentence-transformers: Embedding generation.
    - faiss-cpu: Vector search.
    - openai: LLM inference.
    - numpy: Array operations.
- **External API**: OpenAI (GPT-3.5-turbo model).

4. Deployment Considerations

- **Environment**: Virtual environment (e.g., .venv).
- **Execution**: streamlit run server.py --server.fileWatcherType none (workaround for PyTorch compatibility).
- **Dependencies**: Managed via pip (requirements.txt recommended).
- **API Key**: OpenAI key stored in environment variable (OPENAI_API_KEY).

5. Current Challenges and Workarounds

1. **Streamlit-PyTorch Compatibility**:
    - **Issue**: File watcher conflicts with torch (used by sentence-transformers).
    - **Workaround**: Run with --server.fileWatcherType none.
    - **Long-Term Fix**: Downgrade Python to 3.12 or pin Streamlit version.
2. **Tokenizers Parallelism Warning**:
    - **Issue**: huggingface/tokenizers warns about parallelism after forking.
    - **Workaround**: Set TOKENIZERS_PARALLELISM=false in environment or code.
3. **PDF Parsing Errors**:
    - **Issue**: Malformed PDFs cause incorrect startxref pointer errors.
    - **Workaround**: Use strict=False in PyPDF2.PdfReader or switch to pypdf.

6. Future Enhancements

- **Scalability**: Replace FAISS FlatL2 with a hierarchical index (e.g., HNSW) for larger datasets.
- **PDF Handling**: Support additional file formats (e.g., DOCX, TXT).
- **Performance**: Cache embeddings or pre-index documents offline.
- **Security**: Encrypt API keys and sanitize uploaded files.
- **UI**: Add advanced search options (e.g., filters, multi-result display).

7. Assumptions

- Users upload valid PDFs (though error handling covers malformed files).
- OpenAI API key is available and correctly configured.
- System runs on a single machine (no distributed architecture yet).

8. Diagram

Below is a simplified ASCII representation of the architecture:

text
CollapseWrapCopy

```
+------------------+
```

Notes for Implementation

- Save this document as architecture.md or architecture.docx in your project directory.
- Update the OpenAI API key placeholder in the code with your actual key or use an environment variable.
- If you need a more visual diagram (e.g., in PNG format), let me know, and I can guide you on generating one after confirming your intent (as per my guidelines, I'll ask for confirmation before creating images).