RAG Document Chat Application

A Retrieval-Augmented Generation (RAG) application built with LangChain, Streamlit, and ChromaDB that allows users to upload documents and interact with their content through an AI-powered conversational interface.

Features

- **Multi-Format Support**: Upload and process both `.txt` and `.pdf` files
- **Multiple Document Upload**: Add multiple documents and query across all of them
- **Intelligent Chunking**: Automatically splits large documents into optimally-sized chunks for better retrieval
- **Vector-Based Retrieval**: Uses ChromaDB for efficient similarity search
- **Conversational Interface**: Natural chat-style interaction with context-aware responses
- **Persistent Storage**: ChromaDB stores document embeddings for efficient querying

Requirements

All dependencies are listed in `requirements.txt`. Key packages include:

- `streamlit` - Web interface
- `langchain` & `langchain-openai` - RAG pipeline
- `chromadb` - Vector database
- `pypdf` - PDF processing
- `python-dotenv` - Environment variable management
- `openai` - LLM integration

Setup Instructions

1. Environment Setup

This application is designed to run in the provided GitHub Codespace environment.

1. Fork this repository to your GitHub account
2. Open the forked repository in GitHub Codespaces
3. Wait for the environment to finish setting up

2. Install Dependencies

```
pip install -r requirements.txt
```

3. Configure API Key

Create a `.env` file in the root directory and add your OpenAI API key:

```
```

```
OPENAI_API_KEY=your-api-key-here
```

⚠ **IMPORTANT**: Never commit your `.env` file to the repository. It's already included in `.gitignore`.

4. Run the Application

```
streamlit run app.py
```

A popup will appear in the bottom-right corner. Click "Open in Browser" to view the application.

If you miss the popup:

- Press `Ctrl + C` to stop the app
- Rerun the command
- The popup should appear again

Usage Guide

Uploading Documents

1. Click the **"Browse files"** button in the sidebar
2. Select one or more `.txt` or `.pdf` files
3. Click **"Process Documents"** to upload and index them
4. Wait for the success message

Chatting with Documents

1. After processing documents, type your question in the chat input at the bottom
2. Press Enter or click the send icon
3. The AI will retrieve relevant information and generate a response
4. Continue the conversation - the system maintains context

Managing Documents

- **View Processed Files**: Check the sidebar to see all uploaded documents
- **Add More Documents**: Upload additional files anytime - they'll be added to the existing knowledge base
- **Clear All**: Click "Clear All Documents" to reset the application

Implementation Details

Document Processing Pipeline

1. **Text Extraction**
   - `.txt` files: Direct UTF-8 decoding
   - `.pdf` files: PyPDF extraction with page-by-page processing

2. **Text Chunking**
   - Strategy: `RecursiveCharacterTextSplitter`
   - Chunk size: 1000 characters
   - Overlap: 200 characters
   - Ensures coherent chunks without losing context
3. **Vector Storage**
   - Embeddings: OpenAI's `text-embedding-ada-002`
   - Vector DB: ChromaDB with local persistence
   - Retrieval: Top-k similarity search (k=3)
4. **RAG Pipeline**
   - LLM: GPT-3.5-turbo
   - Chain: `ConversationalRetrievalChain`
   - Memory: `ConversationBufferMemory` for context retention
   - Temperature: 0.7 for balanced creativity

Design Choices

- **Chunking Strategy**: RecursiveCharacterTextSplitter was chosen for its ability to maintain semantic coherence by splitting on natural boundaries (paragraphs, sentences)
- **Chunk Size**: 1000 characters balances context preservation with retrieval precision
- **Overlap**: 200 character overlap prevents information loss at chunk boundaries
- **Vector DB**: ChromaDB chosen for its simplicity, local persistence, and excellent LangChain integration
- **Retrieval Count**: k=3 provides sufficient context without overwhelming the LLM

File Structure

```
.
├── app.py                  # Main Streamlit application
├── requirements.txt        # Python dependencies
├── .env                    # API keys (not committed)
├── .gitignore             # Git ignore rules
├── .devcontainer/         # Codespace configuration
├── data/                  # Sample data files
└── chroma_db/             # Vector database (auto-created)
```

Configuration Changes

No changes were made to the provided `.devcontainer` configuration. The following were added:

- `.env` for API key management
- `.gitignore` to prevent sensitive data commits
- Additional dependencies in `requirements.txt`: `chromadb`, `pypdf`, `python-dotenv`

License

Acknowledgments

- Built using the INFO 5940 Codespace template
- Powered by OpenAI's GPT-3.5 and embedding models
- LangChain framework for RAG implementation
- Streamlit for the web interface