We created a terminal-based chat assistant using Retrieval-Augmented Generation (RAG). The assistant answers user questions based solely on information extracted from a PDF knowledge base or YouTube video transcripts.

Technologies Used

1. **LangChain**: For vector search, knowledge base management, and prompt creation for the model.
2. **Chroma**: A local vector database for document storage and retrieval.
3. **OpenAI API (ChatGPT)**: Generates responses based on the retrieved context.
4. **YouTube API and Transcription Tools**: Extracts transcripts from YouTube videos for additional context.
5. **Colorama**: Adds terminal text styling (e.g., color-coded questions and answers).
6. **Python**: The primary programming language for the project.
7. **Docker**: For containerizing the application and simplifying deployment.

Considerations

1. **Russian Language Support**: All user interactions (questions and answers) are in Russian.
2. **Output Cleanup**: Suppressed unwanted warnings (e.g., DeprecationWarnings).
3. **Code Organization**:
    - Separate scripts for generating the knowledge base (PDFs and YouTube transcripts).
    - Separate scripts for handling user queries.
4. **Styled Output**: Questions, responses, and errors are color-coded for clarity.
5. **Flexibility**: Configurable context size (chunk size) and relevance score threshold.

File Descriptions

1. **01_Create_DB.py**
    - Responsible for creating the knowledge base from a PDF document.
    - **Key Steps:**
        - Loads the document.
        - Splits the text into chunks.
        - Saves the chunks into a local Chroma database.
2. **01_Youtube_videos_scripts_extracter.py**
    - Extracts transcripts from YouTube videos and processes them for inclusion in the knowledge base.
    - **Key Steps:**
        - Downloads transcripts from specified YouTube videos.
        - Cleans and formats the text.
        - Saves the processed transcripts as documents ready for vectorization.
3. **03_Query_responses.py**
    - The main file handling user queries and interacting with the OpenAI model.
    - **Key Steps:**
        - Reads user queries.
        - Searches for relevant data in the knowledge base.
        - Constructs a prompt and sends it to the OpenAI model.
        - Displays the response and its sources to the user.
4. **.env**
    - Stores API keys and other sensitive data.
5. **requirements.txt**
    - Contains all required libraries for the project.
6. **chroma/**
    - A local directory for storing the Chroma database.

Мы создали терминального чат-ассистента, использующего Retrieval-Augmented Generation (RAG). Ассистент отвечает на вопросы пользователей, основываясь исключительно на информации из базы знаний, созданной на основе PDF-документов или транскриптов YouTube-видео.

Использованные технологии

1. **LangChain**: Для векторного поиска, работы с базой знаний и генерации подсказок для модели.
2. **Chroma**: Локальная векторная база данных для хранения и поиска документов.
3. **OpenAI API (ChatGPT)**: Генерирует ответы на основе найденного контекста.
4. **YouTube API и инструменты транскрипции**: Для извлечения текстов из YouTube-видео.
5. **Colorama**: Добавляет стили текста в терминале (например, цвет для вопросов и ответов).
6. **Python**: Основной язык программирования проекта.
7. **Docker**: Для контейнеризации приложения и упрощения его развёртывания.

Что учли

1. **Поддержка русского языка**: Вся работа с пользователем (вопросы и ответы) ведется на русском.
2. **Очистка вывода**: Подавили ненужные предупреждения (например, DeprecationWarnings).
3. **Структура кода**:
    - Отдельные скрипты для создания базы знаний (PDF и YouTube транскрипты).
    - Отдельный скрипт для обработки запросов пользователя.
4. **Стилизация вывода**: Цветовая кодировка вопросов, ответов и ошибок для удобства.
5. **Гибкость**: Возможность настройки размера контекста (размера фрагментов) и порога релевантности.

Описание файлов

1. **01_Create_DB.py**
    - Отвечает за создание базы знаний из PDF-документа.
    - **Основные этапы:**
        - Загрузка документа.
        - Разделение текста на фрагменты.
        - Сохранение фрагментов в локальной базе Chroma.
2. **01_Youtube_videos_scripts_extracter.py**
    - Извлекает транскрипты из YouTube-видео и обрабатывает их для добавления в базу знаний.
    - **Основные этапы:**
        - Скачивание транскриптов указанных видео.
        - Очистка и форматирование текста.
        - Сохранение обработанных транскриптов в формате документов, готовых к векторизации.
3. **03_Query_responses.py**
    - Основной файл, обрабатывающий запросы пользователей и взаимодействующий с моделью OpenAI.
    - **Основные этапы:**
        - Приём запросов от пользователя.
        - Поиск релевантных данных в базе знаний.
        - Создание подсказки (prompt) и отправка её в OpenAI API.
        - Вывод ответа и источников пользователю.
4. **.env**
    - Хранит API-ключи и другие конфиденциальные данные.
5. **requirements.txt**
    - Содержит список всех необходимых библиотек для проекта.
6. **chroma/**
    - Локальная директория для хранения базы данных Chroma.