📚 Sistema de Generación Aumentada por Recuperación (RAG) con TTS (Text-to-Speech)

Este proyecto implementa un sistema RAG (Retrieval-Augmented Generation) avanzado utilizando **LangChain Expression Language (LCEL)**, bases de datos **Chroma**, modelos de *embeddings* de **Hugging Face**, y el modelo **Mistral** a través de **Ollama** para generación de respuestas. Además, incluye una función de voz (TTS) para leer las respuestas generadas.

🎯 Objetivo

Permitir a los usuarios consultar una base de datos de documentos cargados localmente, aplicar diferentes estrategias de *chunking* (segmentación de texto) y filtrar por tema, obteniendo respuestas precisas que pueden ser reproducidas en voz alta.

⚙️ Requisitos Previos

1. **Python 3.8+**
2. **Ollama**: Debe estar instalado y ejecutándose en tu sistema.
3. **Modelo Mistral**: Debes tener el modelo mistral descargado y corriendo en Ollama. Para descargarlo, usa el comando:
   ollama pull mistral

4. **Archivos de Datos**: Los documentos que deseas indexar deben estar en la carpeta ./libros/.

📦 Instalación de Dependencias

Asegúrate de tener un entorno virtual activo y luego instala todas las librerías de Python necesarias:

pip install streamlit langchain-chroma langchain-huggingface langchain-core langchain-ollama pypdf

🚀 Uso del Sistema

El flujo de trabajo consta de dos pasos principales: Indexación de datos y Ejecución de la aplicación.

Paso 1: Indexación de Datos (Creación de las Bases de Datos)

El script index_data.py es crucial. Se encarga de cargar todos los documentos de la carpeta ./libros/ y crear **tres bases de datos distintas de Chroma** (db_recursive, db_fixed, y db_structured) utilizando diferentes estrategias de *chunking* (segmentación) para la comparación.

Ejecución:

python index_data.py

> **Nota:** Este proceso solo necesita ejecutarse una vez, a menos que se añadan nuevos documentos a la carpeta ./libros/.

Paso 2: Ejecución de la Aplicación Streamlit

El script rag_streamlit_tts.py levanta la interfaz de usuario para interactuar con el sistema RAG y el TTS.

Ejecución:

streamlit run rag_streamlit_tts.py

La aplicación se abrirá en tu navegador web.

🧠 Arquitectura de la Cadena RAG (LCEL)

La cadena de RAG se construye utilizando **LCEL (LangChain Expression Language)**, priorizando la estabilidad y la compatibilidad con el modelo local OllamaLLM.

La cadena sigue estos pasos:

1. **Entrada ({context, question}):** Se define un diccionario de inputs.
   - context: Proviene del retriever (que filtra documentos según el tema y el tipo de *chunking* seleccionado). Los documentos recuperados se formatean a una cadena de texto simple (| format_docs).
   - question: Se pasa directamente del input del usuario (RunnablePassthrough()).
2. **Generación de Prompt:** El diccionario de inputs se pasa a PromptTemplate.
3. **Conversión de Tipo (CRÍTICO):** El output de la plantilla (PromptValue) se convierte explícitamente a una cadena de texto (str) usando | RunnableLambda(lambda prompt_value: prompt_value.text). Este paso es fundamental para evitar el error de tipado con OllamaLLM.
4. **Generación:** La cadena de texto simple se pasa al modelo OllamaLLM(model='mistral') para generar la respuesta.

5. **Output:** El texto generado se parsea a una cadena de texto simple (| StrOutputParser()).

🔊 Función de Voz (TTS)

La capacidad de Text-to-Speech se implementa usando el componente tts_component.html (el cual utiliza la API de Google Gemini para la síntesis de voz) renderizado por Streamlit.

- El texto de la respuesta del LLM se envía al componente HTML.
- El JavaScript embebido en el componente llama a la API de TTS, recibe los datos de audio en formato PCM.
- El mismo JavaScript convierte los datos PCM sin procesar a un archivo **WAV** reproducible en el navegador, solucionando el problema común de reproducción de audio.