Construcción de un Asistente Virtual RAG

Caso de Estudio: Trámites TEA en Andalucía

Ángel Manuel Pereira Rodríguez

13 de noviembre de 2025

Resumen

Este documento resume el proceso completo de desarrollo de un sistema de Asistencia de Generación Aumentada (RAG) para responder preguntas sobre trámites, ayudas y normativa del Trastorno del Espectro Autista (TEA) en Andalucía. El proyecto abarca desde la extracción de datos dispersos hasta la implementación de una interfaz de chat conversacional con memoria, utilizando para ello un pipeline de procesamiento de datos y modelos de lenguaje avanzados.

Índice

Índice de figuras

1 Fase 1: Recolección de Datos (Scraping)

El primer desafío fue unificar la información, que se encontraba dispersa en portales web, guías y boletines oficiales (BOJA, BOE) en formatos HTML y PDF.

1.1 Solución: 'scraper_tea.py'

Se implementó un script en Python (`scraper_tea.py`) utilizando las siguientes bibliotecas:

- `requests`: Para realizar las peticiones HTTP.
- `BeautifulSoup4`: Para analizar (parsear) el contenido HTML y extraer texto limpio y enlaces.
- `pypdf`: Para leer y extraer el texto de los documentos PDF.

1.2 Refinamiento Clave: Filtro de Relevancia

Para evitar la descarga de PDFs irrelevantes (ej. 'políticas de cookies'), se implementó un filtro de palabras clave (`RELEVANT_KEYWORDS`). Tras varias iteraciones, se priorizaron términos legales como **'boja'**, **'boe'**, **'ley'** y **'decreto'** para asegurar que ninguna normativa esencial fuera descartada, además de los términos obvios ('tea', 'autismo', 'dependencia').

2 Fase 2: Limpieza y Normalización

El texto extraído ('scrapeado') contenía una gran cantidad de 'ruido': cabeceras, pies de página, números de página, saltos de línea erráticos y artefactos de formato.

2.1 Solución: 'limpia_datos.py'

Se desarrolló un script (`limpia_datos.py`) para procesar todos los archivos `.txt` brutos. Sus tareas principales fueron:

- Eliminar líneas vacías o de ruido (ej. líneas solo numéricas).
- Detectar y eliminar cabeceras y pies de página repetitivos (ej. 'Junta de Andalucía').
- Normalizar los espacios en blanco y asegurar la codificación `UTF-8`.

El resultado de esta fase fue un directorio limpio (`documentos_tea_andalucia_limpio`), fundamental para la calidad del siguiente paso. Este proceso quedó documentado en el informe **'Informe de Limpieza y Recopilación de Textos TEA.pdf'**.

3 Fase 3: Fragmentación Semántica (Chunking)

Los modelos de lenguaje (LLMs) tienen una ventana de contexto limitada. Por tanto, los documentos limpios, que podían ser muy largos (ej. una ley completa), debían dividirse.

3.1 Solución: 'chunker_datos.py'

Se implementó una estrategia de 'chunking' semántico:

- **Tamaño y Solapamiento:** Se definió un tamaño objetivo (`CHUNK_SIZE` $\approx$ 1500 caracteres) y un solapamiento (`CHUNK_OVERLAP` $\approx$ 200 caracteres).

- **Lógica Mejorada:** En lugar de un corte brusco, el solapamiento se implementó de forma 'inteligente', buscando un espacio en blanco cercano para asegurar que el contexto (frases) no se partiera por la mitad entre chunks.
- **Salida:** El resultado fue un archivo `chunks_tea_andalucia.jsonl`, donde cada línea es un objeto JSON con el texto del chunk y sus metadatos (ID, fichero de origen, categoría, título y URL original).

4 Fase 4: Vectorización y Almacenamiento

Para que el sistema pudiera realizar búsquedas semánticas ('buscar por significado' y no solo por palabras clave), los chunks de texto debían convertirse en vectores numéricos (embeddings).

4.1 Solución: 'vectorizer_from_chunks.py'

Este script automatizó la creación de la base de datos vectorial:

- **Modelo de Embeddings:** Se eligió el modelo `models/text-embedding-004` de Google por su alto rendimiento en multilingües y, específicamente, en español.
- **Base de Datos Vectorial:** Se seleccionó **FAISS** (de Meta AI) por su alta velocidad y su capacidad de persistencia local (creando la carpeta `faiss_index_tea`), evitando la necesidad de un servidor de BBDD externo.
- **Proceso:** El script leyó el archivo `.jsonl`, procesó los chunks por lotes (batches) para optimizar las llamadas a la API de embeddings y guardó el índice final en disco.

La arquitectura de esta decisión quedó documentada en **'Documentación Técnica de Embeddings y Vector Store.pdf'**.

5 Fase 5: Motor RAG y Memoria Conversacional

Esta es la fase central, donde se construye el 'cerebro' que conecta la búsqueda con la generación de respuestas.

5.1 Solución: 'motor_rag.py'

Se creó la clase `TeaRagEngine` usando `LangChain` para orquestar el flujo:

- Carga del índice FAISS local y del modelo de embeddings.
- Carga del LLM generativo (se optó por `gemini-2.5-flash` por su rapidez y calidad).

5.2 Evolución Clave: La Memoria

El sistema RAG inicial (basado en `RetrievalQA`) funcionaba bien para preguntas directas, pero fallaba estrepitosamente en las preguntas de seguimiento.

El problema (Figura 1) era que al preguntar '¿Y dónde está el de Sevilla?', el sistema buscaba vectores para esa frase, sin saber que 'el' se refería al 'CVO'.

La Solución Definitiva fue refactorizar el motor para usar `create_history_aware_retriever`. Este nuevo flujo:

1. **Reformulación:** Primero, pasa la nueva pregunta y el historial de chat al LLM para crear una 'pregunta autónoma' (ej. '¿Dónde está el CVO de Sevilla?').

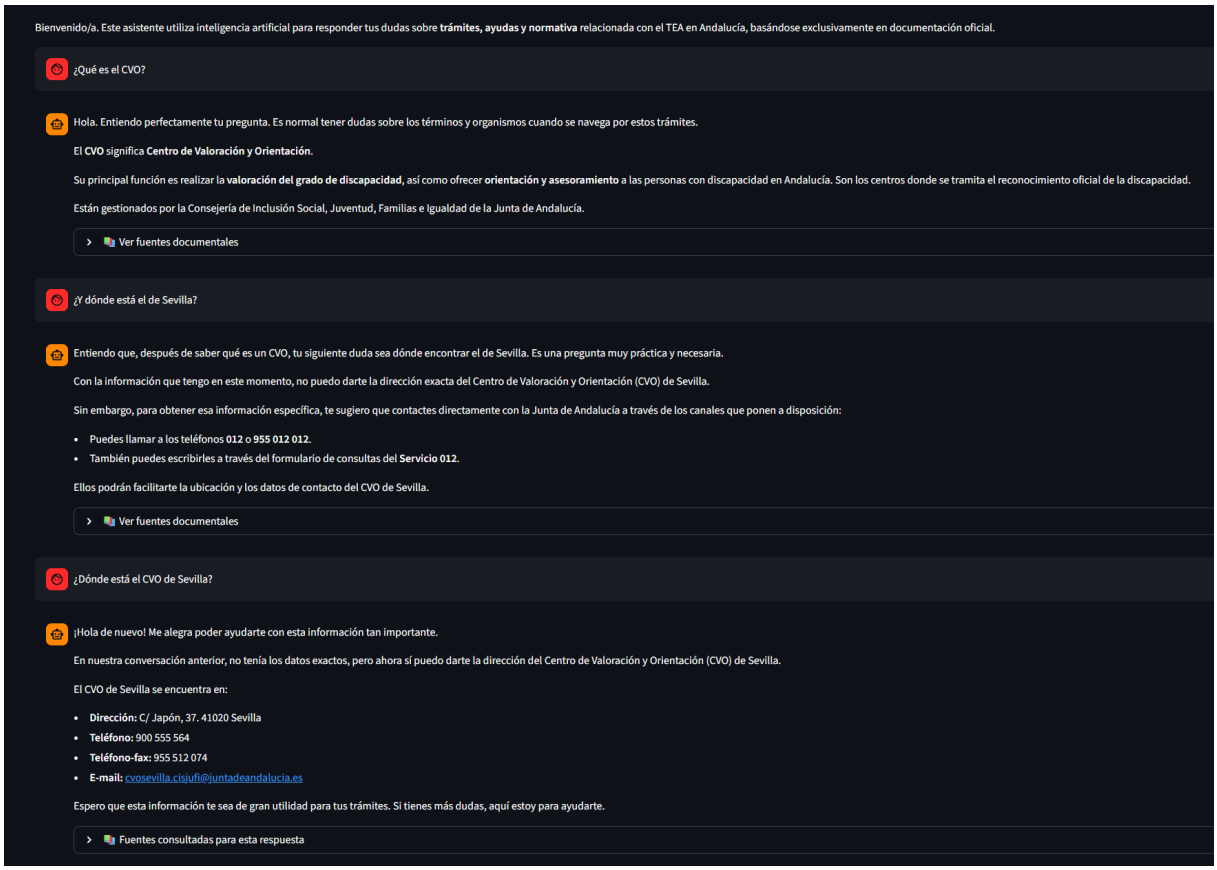

Figura 1: Caso de fallo que demostró la necesidad de reformulación de preguntas.

2. **Búsqueda:** Usa esa pregunta reformulada para buscar en FAISS.
3. **Respuesta:** Pasa los chunks recuperados Y el historial al LLM para generar la respuesta final.

```python
# Prompt para reformular la pregunta
contextualize_q_system_prompt = """Dado un historial de chat y la
      ltima  pregunta del usuario... formula una pregunta independiente..."""
contextualize_q_prompt = ChatPromptTemplate.from_messages([
    ("system", contextualize_q_system_prompt),
    MessagesPlaceholder("chat_history"),
    ("human", "{input}"),
])

# Cadena que reformula ANTES de buscar
history_aware_retriever = create_history_aware_retriever(
    llm, retriever, contextualize_q_prompt
)

# ... (se define el prompt de QA final) ...

# Cadena RAG completa
rag_chain = create_retrieval_chain(
    history_aware_retriever, question_answer_chain
)

# ... (se actualiza self.chat_history en get_answer) ...
```

Listing 1: Lógica clave del motor RAG con memoria

6 Fase 6: Interfaz Gráfica y Ética

Finalmente, se proporcionó una interfaz de usuario accesible para las familias.

6.1 Solución: 'interfaz_grafica.py' (Streamlit)

Se utilizó `Streamlit` para crear una aplicación web de chat interactiva (`app.py`, referenciada como `interfaz_grafica.py` en los archivos).

- Muestra el historial de chat (mensajes del usuario y del asistente).
- Incluye un expansor ('Ver fuentes documentales') debajo de cada respuesta, mostrando los fragmentos de texto exactos y los enlaces a la fuente original que el LLM utilizó, aportando total transparencia.

6.2 Consideraciones Éticas y Legales

Para asegurar un uso responsable, se añadió una barra lateral fija (`st.sidebar`) con un **'AVISO IMPORTANTE'** que especifica:

1. Que es una herramienta informativa y **no sustituye el asesoramiento profesional** o jurídico.
2. La **fecha de la última actualización** de la base documental, advirtiendo que la normativa puede cambiar.
3. El compromiso de utilizar un **lenguaje accesible** y respetuoso.

7 Fase 7: Evaluación y Pruebas del Sistema

Para validar la efectividad y robustez del motor RAG, se realizó una batería de 15 casos de prueba simulando consultas reales de usuarios (documentados en `resultados_prueba.json` y analizados en el informe **'Evaluación del Sistema RAG.pdf'**).

La evaluación se centró en tres criterios principales: la pertinencia de los *chunks* recuperados, la claridad y corrección de la respuesta generada, y el grado de apoyo práctico (accionabilidad) de la misma.

7.1 Resultados de la Evaluación

El análisis de las 15 pruebas arrojó un rendimiento general **sobresaliente**. El sistema demostró una alta capacidad en todos los criterios:

- **Recuperación Pertinente:** La relevancia de los *chunks* recuperados fue **'consistentemente alta'**. El sistema localizó con éxito detalles específicos como importes monetarios ('913,00 euros'), organismos ('Consejería de Inclusión Social, Juventud, Familias e Igualdad') y contactos clave (teléfonos y correos).
- **Respuestas Claras y Correctas:** La calidad de generación del LLM fue **'excelente'**. Las respuestas fueron claras, fáciles de entender y, crucialmente, **fielmente respaldadas** por la evidencia en los *chunks* recuperados, sin observarse contradicciones ni invención de datos.
- **Alto Apoyo Práctico:** El grado de accionabilidad fue **'muy alto'**. Las respuestas no solo 'informan', sino que 'guían' al usuario, detallando secuencias de pasos ('contactar con el departamento de orientación', 'acudir al pediatra') e identificando a los actores correctos (ej. 'Equipo de Orientación Educativa').

7.2 Hallazgo Clave: Gestión de Negativos

Un punto fuerte notable, detectado durante las pruebas, fue la capacidad del sistema para **evitar la alucinación**.

En la prueba sobre *'¿Qué beneficios fiscales...?'*, el sistema no encontró información específica sobre impuestos. En lugar de inventar una respuesta, admitió no tener esa información y pivotó correctamente hacia los datos que sí poseía (becas y ayudas económicas), recomendando además a un asesor fiscal. Este comportamiento es ideal y demuestra la robustez del *prompt* y del flujo RAG.

7.3 Limitaciones Identificadas en Pruebas

Las pruebas confirmaron que no se detectaron errores fácticos. Sin embargo, evidenciaron las limitaciones inherentes al enfoque RAG:

- **Dependencia Documental:** El sistema solo puede responder con la información contenida en su base de datos vectorial. La única 'imprecisión' detectada (no poder listar centros en Sevilla) fue una limitación de los datos fuente, no un error del RAG.
- **Normativa Cambiante:** La información estática (importes de becas, nombres de consejerías) quedará obsoleta, lo que subraya la necesidad de un *pipeline* de actualización periódica.
- **Ausencia de Asesoramiento Personalizado:** El sistema proporciona información general de alta calidad, pero se confirmó que no puede (ni debe) sustituir el asesoramiento profesional de un experto que evalúe un caso particular.

8 Repositorio del Proyecto

El código fuente completo y la documentación de este proyecto están disponibles en el siguiente repositorio de GitHub:

Ver el proyecto en GitHub