UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Informe Final WebIR

Chatbot OpenFing

Grupo 3

Josefina Guberna - 4.709.024-4,
Guido Dinello - 5.031.022-5,
Tomás Spoturno - 5.165.789-6,
Agustina Moraes - 5.000.418-1

Recuperación de Información y Recomendaciones en la Web
Docente Libertad Tansini

Montevideo, 1 de julio de 2024

Índice

1. Introducción

En el entorno educativo, los estudiantes requieren acceso a información confiable y específica para facilitar su aprendizaje. Sin embargo, la variabilidad en la calidad de las fuentes en línea puede representar un problema significativo. Además, encontrar recursos específicos puede ser un desafío debido a la gran cantidad de información disponible. Si bien los videos disponibles en OpenFing resuelven el problema de la confiabilidad, los estudiantes de la Facultad de Ingeniería de la Universidad de la República enfrentan el reto de gestionar el gran volumen de información al intentar extraer datos de los numerosos videos disponibles en esta plataforma.

Este proyecto presenta un sistema basado en un chatbot que facilita el acceso a la información contenida en los videos de OpenFing. El chatbot responde a consultas específicas, indicando los videos y segmentos exactos donde se discuten los temas consultados y proporcionando un resumen del contenido relevante.

Utilizando tecnologías como el modelo Whisper de OpenAI para la transcripción de videos, FAISS para la gestión de una base de datos vectorial y Langchain la construcción de un RAG(Lewis y cols., 2021), el sistema ofrece una solución eficiente para la recuperación de información educativa, mejorando la accesibilidad en la búsqueda de información académica que asegura la fiabilidad de los contenidos.

2. Problema/Necesidad

En el entorno educativo, los estudiantes a menudo enfrentan desafíos al buscar información específica para sus estudios. Uno de los principales problemas es la **fiabilidad de las fuentes**. Muchas veces, los estudiantes recurren a Internet, donde la calidad y precisión de los recursos pueden variar considerablemente. En este sentido, tener acceso a información directamente de profesores de la facultad para evacuar dudas u obtener información sobre cierto tema garantiza contenido confiable y preciso. En el caso de la Facultad de Ingeniería, hay una gran cantidad de clases filmadas disponibles en OpenFing, que ofrecen información directa de los profesores.

Sin embargo, utilizar las clases de OpenFing como fuente confiable de información implica manejar una gran cantidad de contenido y trae asociado el problema del **gran volumen de información**. Revisar videos completos para encontrar respuestas específicas es ineficiente y consume mucho tiempo, a menudo requiriendo buscar en varias clases para obtener la información necesaria. Incluso sabiendo que en una clase se habla de cierto tema, puede ser difícil y llevar mucho tiempo encontrar el minuto exacto en donde se discute. Esto frecuentemente lleva a que los estudiantes abandonen la idea de usar los videos de OpenFing como fuente de información y recurran a medios más rápidos, como Internet.

El sistema desarrollado para el proyecto de la asignatura WebIR aborda estos problemas proporcionando una forma eficiente de acceder a la información de los cursos disponibles en los videos de OpenFing. Facilita respuestas rápidas a las dudas de los estudiantes y ofrece información sobre diversos temas utilizando fuentes confiables, lo que a su vez mejora y facilita el proceso de aprendizaje.

3. Enfoque de la solución

El sistema desarrollado consiste en un chatbot que actúa como asistente para estudiantes interesados en obtener información sobre temas específicos abordados en los cursos de la Facultad de Ingeniería y disponibles en OpenFing. Permite a los estudiantes realizar preguntas sobre temas particulares, a las cuales el chatbot responde proporcionando el o los videos de OpenFing que abordan la consulta, indicando la clase y el curso correspondientes y el minuto específico en el cual se trata el tema en cuestión. Además ofrece un resumen del contenido de esa parte del video para que el estudiante pueda decidir si desea verlo.

En caso de no encontrar información relevante sobre el tema consultado o no poder recuperar una clase con información suficientemente pertinente, el chatbot se lo informará al usuario y responderá la consulta utilizando sus propios conocimientos, es decir, los del modelo de lenguaje utilizado como componente fundamental en el sistema. El chatbot indicará claramente que el contenido proporcionado no proviene de OpenFing, ya que es importante que el estudiante sepa que, en este caso, la información no tiene el mismo nivel de confiabilidad que la disponible en las clases de la facultad. Aunque el modelo de lenguaje utilizado es muy inteligente y es de los mejores disponibles hasta el momento, es sabido que cualquier modelo puede generar información errónea y sin sustento. Teniendo en cuenta esto, el estudiante podrá decidir cómo utilizar la información recibida.

Adicionalmente, el chatbot podrá seguir proporcionando información al usuario si este lo desea y continúa haciendo preguntas. Al igual que antes, en caso de tener algún video que resuelva la consulta especifica, el chatbot lo proporcionará, y en caso contrario responderá con sus conocimientos. Esto permitirá complementar el contenido de las clases o aclarar dudas adicionales que el estudiante pueda tener sobre las respuestas proporcionadas previamente.

La principal fortaleza de esta solución radica en su capacidad para ofrecer información confiable de los cursos de la facultad de manera rápida y eficiente. Permite proporcionar segmentos precisos de videos que abordan los temas consultados, facilitando a los usuarios el acceso directo a contenido académico verificado y permitiéndoles encontrar rápidamente la información que necesitan.

4. Diseño y Arquitectura

La arquitectura del sistema desarrollado se muestra en la Figura 1. En la sección de implementación se detalla cada uno de los módulos implementados:

- **Extracción de la información:** Scrapping de videos de OpenFing y su almacenamiento en MongoDB.
- **Transcripción y segmentación:** Transcripción de los videos.
- **Generación de embeddings:** La creación de embeddings de los segmentos transcritos, y su almacenamiento en una base de datos vectorial.
- **Recuperación y respuesta del chatbot (RAG):** Recupera segmentos relevantes de la base de datos vectorial y genera respuestas utilizando un modelo de lenguaje.
- **Interfaz gráfica:** Permite a los usuarios interactuar con el chatbot.

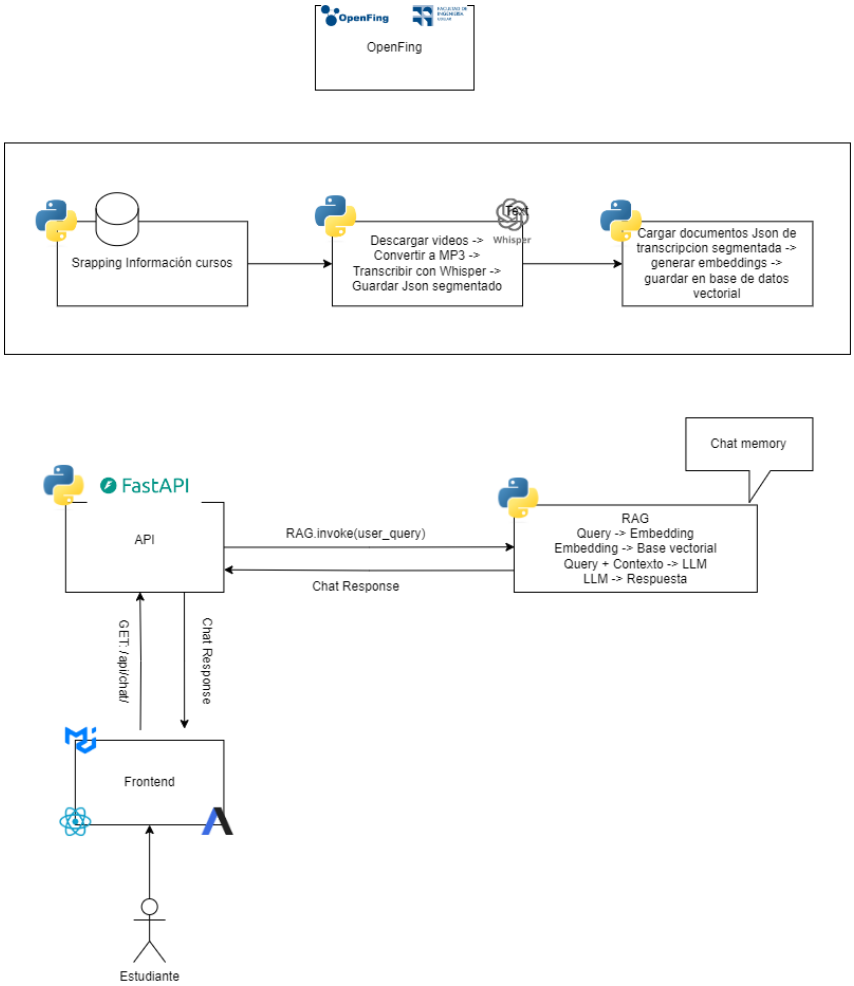

Figura 1: Arquitectura del sistema desarrollado.

5. Implementación

5.1. Extracción de la Información

El primer paso en la implementación de este proyecto fue extraer la información de los videos de todos los cursos de OpenFing[1] y almacenarla en MongoDB, una base de datos no relacional. Para ello, se realizó un scrapping de la página utilizando BeautifulSoup(Richardson, 2023), una librería de Python. La información se guardó en dos colecciones: una con documentos de cada curso y otra con documentos de los videos asociados a cada curso. En la figura 1 se muestra un ejemplo de la estructura de un documento de la colección de cursos, y en la figura 2 se muestra un ejemplo de la estructura de un documento de la colección de videos.

```
{
    "_id": "6656803d97cf1d12f025ffeb",
    "name": "Cálculo Diferencial e Integral en una Variable",
    "url": "https://open.fing.edu.uy/courses/civ/"
}
```

Listing 1: Ejemplo - documento de colección de cursos

```
{
    "_id": "6656804097cf1d12f025ffed",
    "name":"Curvas Paramétricas",
    "url":"https://open.fing.edu.uy/courses/c3/1/",
    "video":"https://open.fing.edu.uy/media/c3/c3_01.mp4",
    "transcribed":false,
    "subjectId": "6656803d97cf1d12f025ffeb"
}
```

Listing 2: Ejemplo - documento de colección de videos

5.2. Transcripción y Segmentación de Videos

Luego del proceso de extracción de información, se procedió a la transcripción de cada video. Debido a la gran cantidad de videos disponibles en OpenFing, se decidió limitar el problema a un subconjunto específico de videos. Para la transcripción, se utilizó Whisper(OpenAI, 2023), un modelo de OpenAI. Específicamente, se empleó el modelo base, el cual requiere 1 GB de memoria de video. Por defecto, Whisper procesa la entrada de audio y genera una salida de texto, dividiéndola en segmentos o fragmentos más pequeños. Cada segmento suele representar una oración o frase, lo que facilita el análisis y procesamiento de la transcripción.

5.3. Generación de Embeddings y Almacenamiento

A partir de los documentos de salida con las transcripciones segmentadas de cada clase se cargó una base de datos vectorial, con la cual se podrán realizar búsquedas por similitud sobre el corpus de las clases transcritas dada una pregunta de un estudiante.

[1] https://open.fing.edu.uy/

5.3.1. Base de datos vectorial

Se utilizó FAISS(Facebook, 2024) (Facebook AI Similarity Search) como base de datos vectorial para almacenar y gestionar los embeddings generados a partir de cada segmento de clase. Esta herramienta permite realizar las búsquedas por similitud de forma eficiente y recuperar los segmentos de información más relevantes para la pregunta planteada.

5.3.2. Modelo de Embeddings

Para generar los embeddings de las transcripciones, se utilizó el modelo BGE-M3 (Chen y cols., 2024) disponible en la plataforma Hugging Face. Este modelo open-source, desarrollado por la Academia de Inteligencia Artificial de Beijing (BAAI), fue seleccionado por ser multilingüe, lo cual es de particular interés dado que en este caso las transcripciones son en su gran mayoría en español. El modelo presenta un rendimiento destacable en español(Chen y cols., 2024) y sus requerimientos en cuanto a recursos son bajos, lo que lo convierte en una opción eficiente y práctica para el proyecto.

5.3.3. Agrupación de Segmentos

Una observación importante que se realizó durante el desarrollo fue la longitud de los segmentos devueltos por Whisper. A menudo, estos segmentos eran demasiado cortos (entre 3 y 5 palabras) para proporcionar un contexto valioso al modelo de lenguaje. Para abordar esto, agrupamos segmentos consecutivos hasta alcanzar una duración mínima de 2 minutos. Encontramos que este enfoque mejoró significativamente la calidad y utilidad de los segmentos recuperados desde la base y, por ende, la calidad de las respuestas generadas por el modelo de lenguaje al proveerle un contexto más valioso.

El proceso de agrupación de segmentos se implementó dentro de la lógica de procesamiento de datos antes de almacenar los embeddings en FAISS. Este enfoque garantiza que cada vector almacenado represente un fragmento coherente y significativo de información.

5.3.4. Metadata

Las bases de datos vectoriales suelen permitir almacenar metadata asociada a cada uno de los embeddings. Esto facilita la implementación de funcionalidades complejas, ya que posibilita, entre otras cosas, contextualizar los embeddings, es decir, entender su origen y significado.

En este caso, inicialmente se optó por guardar únicamente una referencia (ID) a la clase de la cual se extrajo el segmento, además de los timestamps de inicio y finalización del segmento. Con este ID de la clase, es posible acceder a toda la información correspondiente consultando la base de datos de MongoDB, evitando de esta manera la repetición innecesaria de información en cada segmento de la base de datos vectorial.

Sin embargo, esto presentó ciertos problemas. Una vez avanzado el desarrollo del RAG, se encontró que el enfoque seguido presentaba dificultades para acceder a la base de MongoDB antes de generar la respuesta con el LLM. Si bien era posible, implicaba reescribir gran parte del módulo mencionado. Por lo tanto, se decidió incluir dos campos adicionales en la metadata de los embeddings: el nombre de la clase y el nombre de la materia a la cual pertenecía.

De esta forma, al agregar estos dos datos a la metadata, el LLM obtenía toda la información que consideramos necesaria. Esto es debido a que lo instruimos para que mencione en su respuesta el nombre de la clase y la materia que utiliza como fuente. Además, evitamos almacenar datos innecesarios para el modelo, ya que, por ejemplo, los enlaces a los videos en el tiempo correspondiente los obtuvimos y presentamos directamente en el frontend.

5.4. Recuperación y Respuesta del Chatbot (RAG)

Una vez que los videos de OpenFing fueron transcritos, segmentados y almacenados en una base de datos vectorial junto con sus respectivos embeddings, se puede utilizar esta base para recuperar los segmentos relevantes dada una consulta. Este proceso sigue el enfoque de un modelo RAG (Retrieval Augmented Generation). La generación aumentada por recuperación (RAG) optimiza la salida de un modelo de lenguaje de gran tamaño consultando una base de conocimientos externa antes de generar una respuesta. Más concretamente un RAG funciona como se muestra en la figura 2, y sigue el siguiente flujo:

1. El usuario realiza una consulta
2. La consulta se convierte en un embedding y se compara con todos los embeddings almacenados en la base de datos vectorial. Se recuperan los k vectores más similares (en este caso k=2) utilizando métricas como similitud coseno, distancia euclidiana, distancia de Manhattan o producto punto. A los vectores recuperados se les denomina "contexto".
3. El contexto se proporciona al prompt del sistema. Dicho prompt sirve para guiar al modelo de lenguaje sobre cómo comportarse frente a una consulta y un contexto proporcionado. En este caso, se especifica al prompt que el sistema actuará como un asistente para ayudar a los estudiantes a encontrar información relevante de clases transcritas almacenadas en una base de datos. Se detalla cómo debe responder dependiendo de si se recupera contexto suficientemente relevante para la consulta o no. También se indica que se le proporcionará cierto contexto que se concatenará al final del prompt.
4. El prompt es entregado al modelo de lenguaje, que lo utiliza para determinar su comportamiento y generar una respuesta para el usuario.

Un aspecto fundamental en la interacción con un chatbot es que se pueda hacer referencia a elementos mencionados anteriormente y el chatbot pueda entender a qué se refiere el usuario, como sucede en una conversación humana. En otras palabras, es importante que tenga memoria, recuerde lo intercambiado previamente y pueda comprender las referencias que el usuario haga a ese contenido.

Para lograr esto, se agregaron algunos pasos adicionales al flujo descrito, que se muestran en la figura 3. En estos pasos, se utiliza un LLM para convertir la consulta del usuario en una pregunta ïndependientez así recuperar correctamente el contexto de la base de datos.

Por ejemplo, si ha habido una interacción entre el usuario y el chatbot sobre la inducción completa y el usuario pregunta: "¿Podrías darme un ejemplo que aplique esa técnica?", es deseable que el chatbot entienda que se refiere a un ejemplo o demostración que use inducción completa. Si la consulta original se convirtiera directamente en un embedding para recuperar el contexto no se obtendrían segmentos relevantes relacionados con la inducción completa.

Por esto, a veces es necesario procesar la pregunta del usuario para convertirla en una pregunta que pueda ser contestada sin el contexto de la conversación anterior. Más específicamente, se proporciona la pregunta original a un prompt junto con el historial de la conversación. Este prompt contiene indicaciones sobre cómo debe actuar: recibe las entradas y su tarea es reformular la pregunta, en caso de ser necesario, para que pueda ser entendida sin el historial del chat. Se aclara que no debe responder la pregunta, solo reformularla (la respuesta a la consulta será proporcionada por el LLM en un paso posterior, luego de recuperar el contexto relevante).

Para esta parte de recuperación y respuesta del chatbot se utilizó LangChain (LangChain, 2024a), un framework para el desarrollo de aplicaciones basadas en grandes modelos de lenguaje. LangChain simplifica el proceso de desarrollo de este tipo de aplicaciones al proporcionar bloques de construcción, componentes e integraciones de terceros. Provee una gran cantidad de funciones que facilitan la recuperación de la base de datos vectorial, la generación

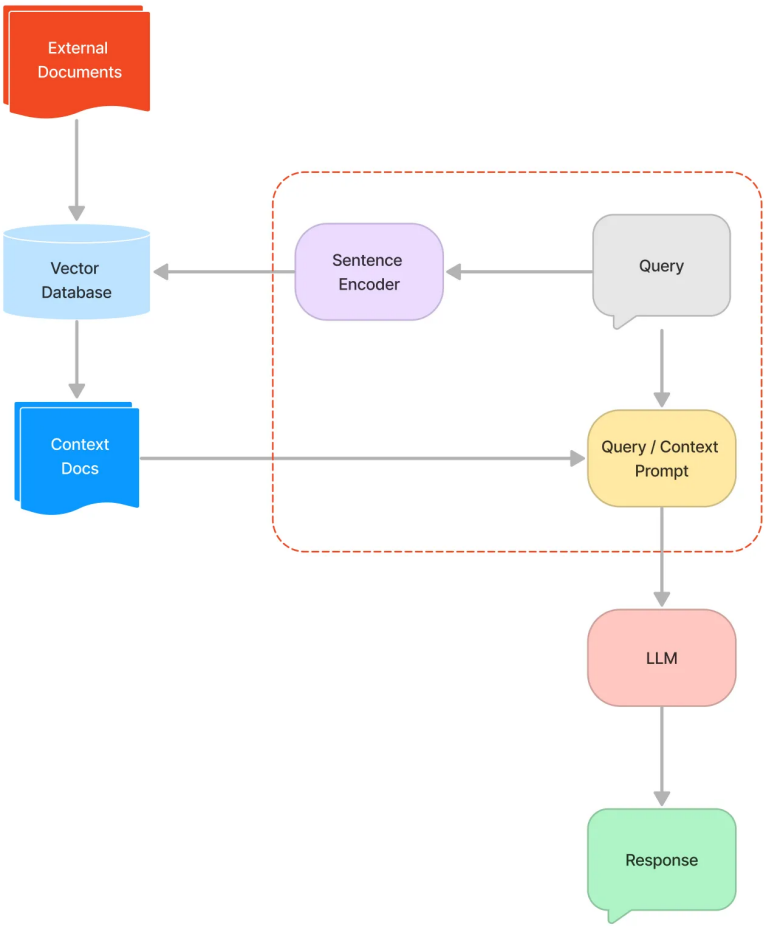

Figura 2: Arquitectura de un RAG (Thaker M., Medium, 2024)

de prompts, la integración de diversos modelos de lenguaje y modelos de embeddings, entre otras capacidades. Además, estas funciones son altamente personalizables, lo que permite adaptarlas a la aplicación específica que se esté desarrollando.

Incluso ofrece una herramienta de evaluación y debugging llamada LangSmith (LangChain, 2024b), diseñada para inspeccionar, monitorear y evaluar sistemas. Aunque en el proyecto se utilizaron solo algunas funcionalidades básicas, LangSmith tiene capacidades impresionantes, y fue fundamental durante el debugging para observar el comportamiento del sistema en tiempo real, así como para analizar las entradas y salidas de cada función específica. Esto facilitó la resolución de los problemas que fueron surgiendo durante el desarrollo. Además, tanto la comprensión de la herramienta como su integración en el sistema resultaron sencillas, considerando las grandes ventajas que proporcionó.

5.4.1. Decisiones de diseño y detalles de implementación

En el transcurso del proyecto fue necesario tomar algunas decisiones de diseño basadas en problemas que fueron surgiendo, así como decisiones inherentes a la aplicación, como la selección del modelo de lenguaje y la determinación de valores para ciertos parámetros.

En cuanto al modelo de lenguaje, se comenzó utilizando el modelo Mixtral 8x7B(Jiang y

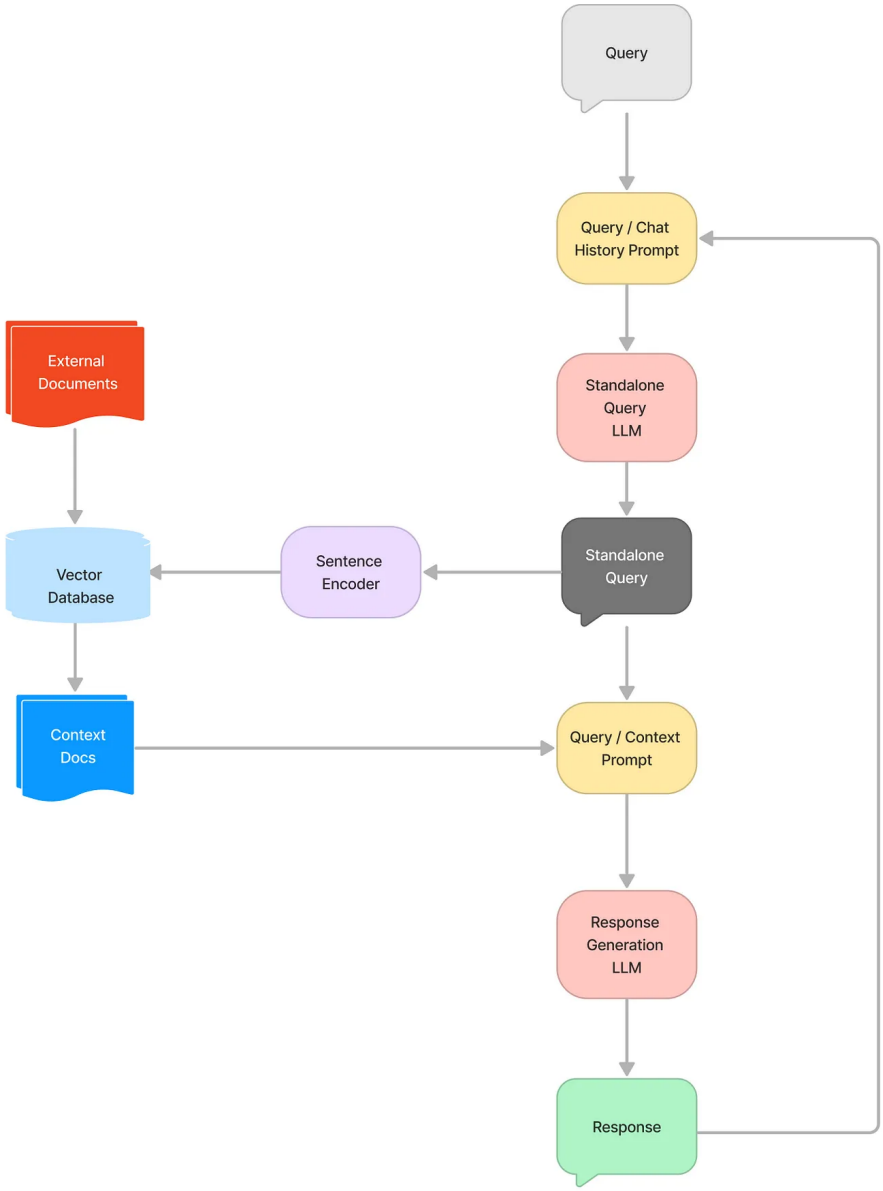

Figura 3: Arquitectura de un RAG con memoria (Thaker M., Medium, 2024)

cols., 2024) de Mistral AI, disponible en Hugging Face, el cual ha demostrado un rendimiento comparable a GPT-3.5 en términos de eficiencia y precisión. Este modelo de código abierto fue fundamental durante gran parte del desarrollo, permitiéndonos construir cada componente del RAG: la recuperación, la generación de un prompt que recibiera como entrada un contexto y una consulta, la integración de un LLM, la implementación de la memoria, entre otras partes del sistema. Nos permitió verificar que todo funcionara como queríamos a nivel funcional.

Sin embargo, cuando comenzamos a enfocarnos en el rendimiento y en la precisión de las respuestas del chatbot, observamos que, en muchas ocasiones, el modelo no seguía al pie de la

letra las instrucciones dadas en el prompt sobre cómo debía comportarse. Decidimos entonces cambiar al modelo GPT-4o de OpenAI, y notamos una gran mejora en las respuestas. Aunque fue necesario iterar varias veces sobre el prompt hasta lograr la versión final, el cambio a GPT-4o resultó en respuestas mucho más cercanas a las esperadas. Continuamos evaluando el modelo, probando con diferentes consultas, observando las respuestas y ajustando el prompt en consecuencia.

A pesar de que Mixtral 8x7B no tenía la precisión esperada, utilizar un modelo de código abierto fue muy útil durante la fase de desarrollo en la que el enfoque principal era la implementación y aún no se analizaba ni se priorizaba el rendimiento del sistema. Nos permitió probar muchas cosas sin incurrir en costos por cada consulta.

Por otro lado, otra de las decisiones que hubo que tomar surgió del problema de cómo determinar si el contexto recuperado en cada caso es suficientemente relevante para la consulta realizada. Una de las opciones, y probablemente la mejor, es proporcionar todos los segmentos recuperados al LLM para que éste, con su inteligencia, determine si cada uno es relevante o no dada la consulta del usuario. Basándose en esto, el LLM devolvería al usuario los links de los videos que haya considerado relevantes junto con un breve resumen, indicando la clase y asignatura correspondiente.

Sin embargo, se tiene el inconveniente de que cada segmento almacenado en la base de datos vectorial solo contiene la metadata del tiempo de inicio y finalización del segmento, el ID de la clase, el nombre de la clase y el nombre de la asignatura asignatura asociados. Los segmentos no incluyen la URL del video como metadata para evitar la repetición de esta información en cada segmento. Aunque es posible obtener otros datos con el ID de la clase, estos no se almacenan junto con cada segmento para optimizar el espacio de almacenamiento.

No obstante, con la implementación realizada y las funciones de LangChain utilizadas, es posible pasar al prompt la metadata asociada a cada segmento, pero no es posible realizar un procesamiento previo para obtener la URL y pasarla como entrada al prompt. Por lo tanto, el prompt, y por ende el LLM, no disponen del enlace (URL) que es fundamental para la respuesta del chatbot.

Por esta razón, se optó por determinar la relevancia de cada segmento según un umbral. Tras diversas evaluaciones, se decidió establecer un umbral de similitud de 0.2 en un rango de [0,2]. Si un segmento tiene una similitud igual o superior al umbral, se pasa al LLM como parte del contexto, junto con la instrucción de utilizar todos los segmentos en la respuesta. Se le indica al LLM que debe generar un párrafo independiente por cada segmento, incluyendo un breve resumen del mismo y mencionando el nombre de la clase y la asignatura correspondiente. Luego, en el frontend, se agregan los enlaces de cada segmento recuperado.

Se le da esta indicación al LLM de que siempre utilice todos los segmentos proporcionados porque de lo contrario podrían haber discrepancias entre los enlaces mostrados en el frontend y el texto generado por el modelo de lenguaje, resultando en enlaces sin un resumen asociado, lo cual no sería coherente.

Aunque la implementación final adoptó esta solución, algunas veces se enfrenta al problema de discrepancia entre los enlaces proporcionados y el texto generado por el modelo de lenguaje. Un ejemplo claro podría ser el caso de la Figura 4, donde el usuario realiza una pregunta que se espera que no recupere contexto ("Hola! Puedes ayudarme?"). A pesar de esto, el RAG recupera segmentos que superan el umbral de similitud y se devuelven los links asociados en el frontend. Sin embargo, el LLM ignora el contexto proporcionado y responde mostrando disposición para ayudar al usuario. En este caso, es correcto y esperable que el LLM no utilice el contexto para responder a esta pregunta, pero lo ideal sería continuar trabajando en la componente de recuperación para evitar recuperar segmentos en consultas como esta, que contienen palabras comunes que no se refieren a ningún tema o contenido específico que se desee recuperar de la base de datos vectorial.

Si bien el umbral establecido es muy bajo, permite la recuperación de contexto en casos como este, por lo que se debería seguir mejorando en este aspecto para alcanzar el comportamiento esperado.

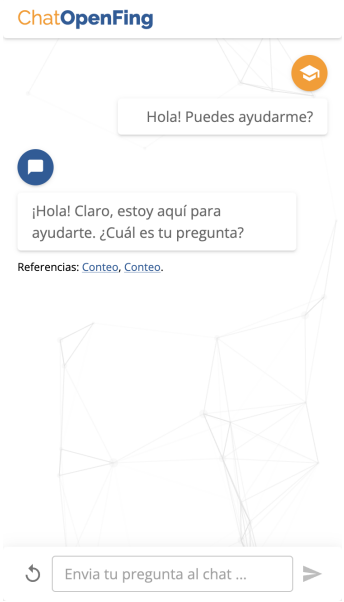

Figura 4: Ejemplo de discrepancia entre links y texto proporcionados.

Junto con los desafíos y decisiones mencionadas, también fue necesario abordar otros aspectos, como configurar parámetros como la temperatura del LLM y la cantidad de segmentos más similares a recuperar, entre otros ajustes clave para optimizar el rendimiento y la precisión del sistema. Estas configuraciones se determinaron después de varias iteraciones de evaluar y ajustar, y tuvieron un impacto directo en la experiencia del usuario final, garantizando respuestas más pertinentes y coherentes en cada interacción.

5.5. API del Chatbot

Hasta este punto, en el sistema se cuenta con una forma de recuperar y responder a las preguntas, pero no una forma de comunicación entre el cliente web (la interfaz gráfica) y el sistema de recuperación y respuesta del chatbot (RAG). El módulo de la API del chatbot permite esta comunicación y también maneja la preservación del historial de conversaciones con el usuario, mejorando la coherencia y el contexto de las interacciones.

Las conversaciones se almacenan en memoria en un diccionario, donde las claves son los identificadores de las conversaciones y el valor es la lista de mensajes intercambiados entre el usuario y el RAG.

La API fue escrita en Python usando FastAPI (Ramírez, 2023) y expone un único endpoint para la comunicación con el frontend: `/query`. Este endpoint acepta solicitudes HTTP POST y recibe como parámetros el mensaje del usuario y el identificador de la conversación (si ya fue generado anteriormente).

Para responder la solicitud, se recuperan los mensajes intercambiados con el identificador de la conversación que, junto con el mensaje del usuario se envían al RAG para proporcionar más contexto al modelo de lenguaje sobre las preguntas anteriores del usuario, facilitando así una comunicación más fluida y contextualizada.

5.6. Interfaz Gráfica del Chatbot

La interfaz gráfica que actua como el cliente web, es la forma de interacción que tiene el usuario para comunicarse con el sistema. Fue desarrollada usando React (Meta, 2023) y los componentes de UI de Materia UI (Team, 2023).

Figura 5: El Inicio

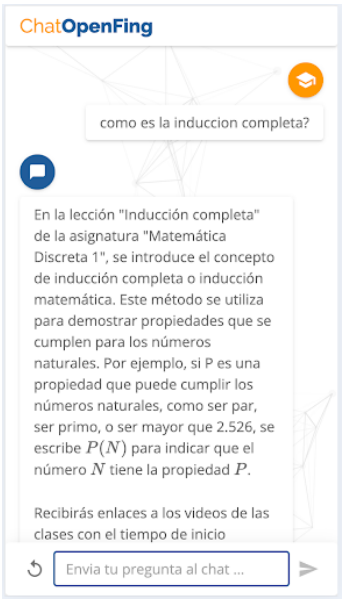

Figura 6: La Pregunta

Figura 7: La Carga

Como se muestra en las figuras anteriores, la interfaz gráfica consiste en una ventana de conversación donde el usuario puede ingresar preguntas que se envían al sistema. Una vez que usuario ingresa una pregunta, se le muestra un indicador de carga mientras se genera la respuesta del sistema, la cual se muestra en pantalla una vez resuelta la solicitud. Luego de cada interacción el usuario tiene la posibilidad de reiniciar la conversación, comenzando así con un contexto totalmente nuevo.

Es importante mencionar,que en cada respuesta del sistema se incluyen referencias al contenido correspondiente, vinculadas a la plataforma de OpenFing[2] con el minuto exacto en que el profesor aborda el tema relacionado con la respuesta del usuario, lo cual asegura fiabilidad en la información proporcionada.

6. Evaluación y resultados

A continuación se muestran ejemplos de preguntas realizadas al chatbot junto con las respuestas proporcionadas, con el objetivo de evaluar su desempeño y comportamiento ante diferentes consultas.

En la figura 8 se muestra una pregunta sobre un tema para el cual existe una clase disponible en OpenFing, y además la transcripción de esa clase se encuentra almacenada en la base de datos vectorial. Como se puede observar, el chatbot responde con un párrafo separado para cada segmento recuperado. En cada uno de estos párrafos realiza un breve resumen de lo que se menciona en el segmento, incluyendo la clase y asignatura correspondiente, lo cual es el comportamiento esperado. Accediendo a los enlaces proporcionados (https://open.fing.edu.uy/courses/md1/1/?t=0 y https://open.fing.edu.uy/courses/md1/1/?t=225.4), se puede verificar que cada video comienza en el segundo correspondiente al comienzo del segmento (t=0 y t=225.4), y también que el resumen generado por el chatbot es efectivamente un resumen del contenido de cada segmento del video.

Dado que los dos segmentos recuperados tienen menos de 4 minutos de diferencia, una posible mejora para esta respuesta sería devolver un único enlace que comience en el tiem-

[2] https://open.fing.edu.uy/

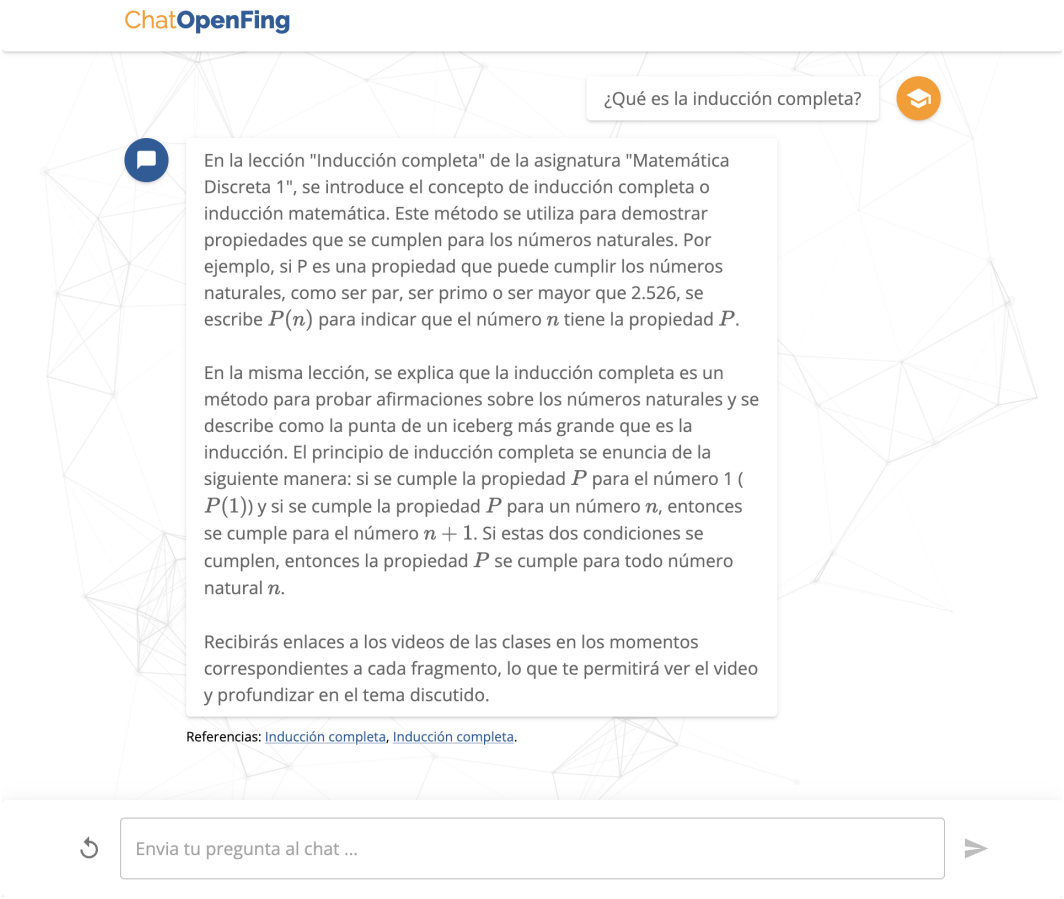

Figura 8: Ejemplo de conversación donde se consulta sobre un tema almacenado en la base de datos vectorial

po de inicio del primer segmento (t=0). Esto se debe a que al seguir el primer enlace, el estudiante avanzaría rápidamente hasta el comienzo del segundo segmento, lo cual hace innecesario proporcionarle un segundo enlace.

A continuación, en la misma conversación, se le hace una pregunta al chatbot relacionada con un tema discutido previamente. Se utiliza la expresión "este método" para hacer referencia a la inducción completa, tema del que se venía hablando. El chatbot comprende la referencia del usuario y responde proporcionando dos párrafos, uno para cada segmento recuperado. Aunque ambos videos devueltos están relacionados con demostraciones utilizando la inducción completa (https://open.fing.edu.uy/courses/md1/1/?t=2182.4 y https://open.fing.edu.uy/courses/md1/1/?t=111.8), el primero muestra una demostración específica, mientras que el segundo solo menciona las diferentes estructuras sobre las que se puede aplicar este método (números naturales, árboles, lenguajes lógicos). Entonces para este caso, también podría ser deseable que el chatbot devuelva únicamente el primer enlace. Este ejemplo se puede visualizar en la figura 9.

Otra conversación se muestra en la figura 10. En este caso, la respuesta a la pregunta no se encuentra en ningún curso de OpenFing, ya que trata sobre un tema de biología que no se enseña en la Facultad de Ingeniería. El chatbot responde como es esperado, indica claramente que no hay clases en la base de datos que cubran ese tema y proporciona una respuesta utilizando sus conocimientos. Luego, el usuario podría continuar formulando preguntas adicionales sobre el tema y cualquier consulta que surja a partir de la respuesta

Figura 9: Ejemplo de conversación haciendo referencia a un tema mencionado anteriormente

proporcionada, y el chatbot continuará respondiendo, como se puede observar en la figura 11.

En rasgos generales, consideramos un resultado satisfactorio. Las respuestas obtenidas cumplen en su mayoría con los estándares planteados al comienzo del proyecto, permitiendo a los estudiantes obtener respuestas a sus preguntas de forma rápida, sin tener que realizar búsquedas extensas por varios videos buscando la explicación más acertada a su inquietud. Las respuestas provienen de fuentes confiables, citando la fuente cuando corresponde o indicando que la respuesta es generada únicamente con el conocimiento del modelo. Además, se presentan de una forma clara y amigable, como si se estuviera charlando con otra persona o un instructor experimentado.

El feedback recibido fue sumamente positivo, incluso por parte de los estudiantes, quienes expresaron que la puesta en producción del proyecto por parte de la facultad sería bien recibida y de gran utilidad para ellos.

Figura 10: Ejemplo conversación donde se consulta sobre un tema que no está almacenado en la base de datos vectorial

7. Conclusiones y Trabajo Futuro

La primera implementación del sistema ha permitido realizar una prueba de concepto de los principales desafíos de un proyecto de esta naturaleza, demostrando su gran potencial y utilidad para los estudiantes. Sin embargo, es evidente que existe un amplio margen para continuar expandiendo y mejorando los diferentes componentes de esta arquitectura. Por ejemplo, en el modulo de extracción de información, se podría obtener la información de todas las clases, incluyendo los prácticos y la interacción con otras plataformas para estudiante, además de la carga de información desde otro tipo de documentos (notas, libros, etc).

En el transcriptor, se observo durante la implementación que algunas palabras no fueron escritas correctamente, por lo que consideramos implementar un post-procesamiento de las transcripciones utilizando un modelo de lenguaje.

Podríamos experimentar con una arquitectura de recuperación en dos etapas (Pinecone, 2023) (Two-Stage Retrieval), en la cual añadimos un paso extra utilizando un modelo Reranker. Este enfoque nos permite aumentar el recall en la recuperación de documentos desde la base, ya que podemos recuperar un mayor número de documentos inicialmente y luego ordenarlos. De esta manera, nos quedamos con los documentos más relevantes semánticamente, utilizando la puntuación de similitud entre la pregunta y el documento estimada por el modelo Reranker.

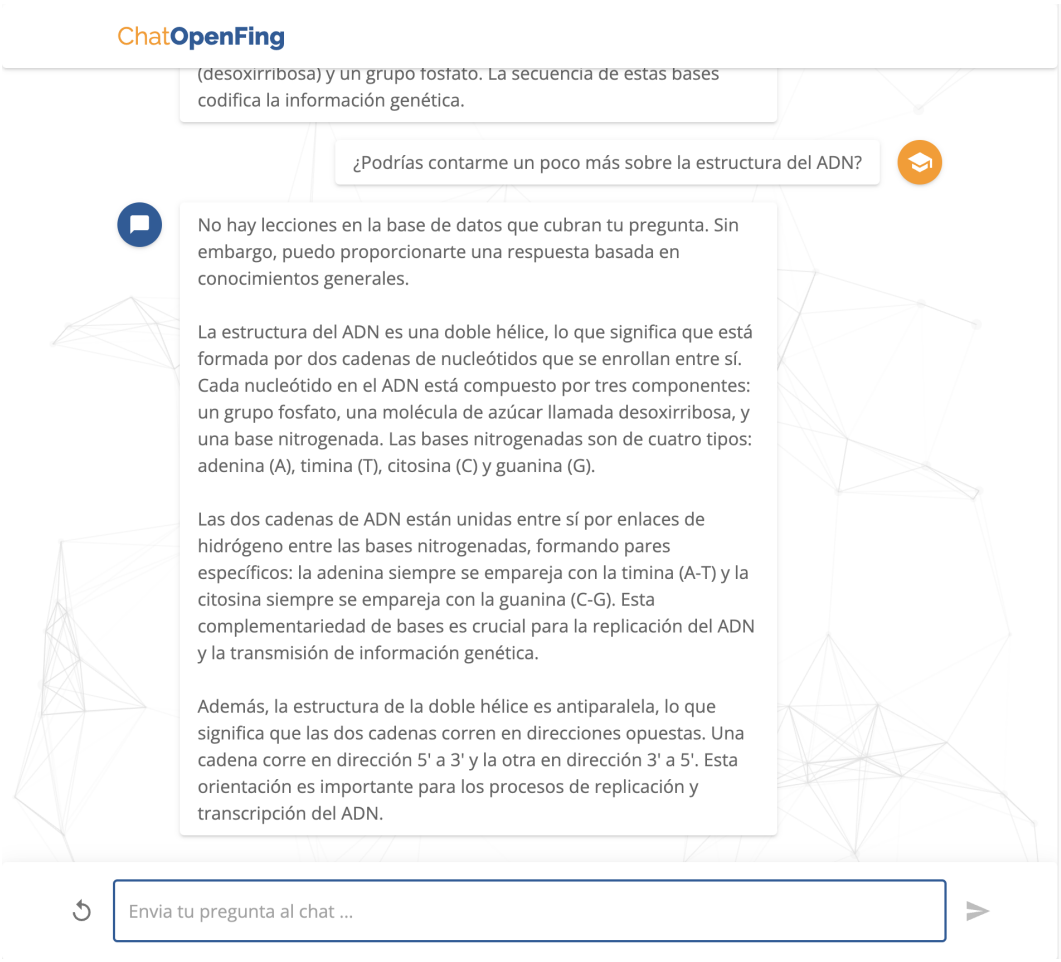

Figura 11: Ejemplo conversación donde el usuario continúa preguntando y conversando con el chatbot

Para mejorar el funcionamiento del RAG se puede ajustar el prompt del modelo de lenguaje y los parámetros del retriever como la temperatura y el umbral de relevancia. También sería beneficioso agregar ejemplos de preguntas con respuestas esperadas[3], mejorar la obtención de documentos ampliando la ventana de contexto[4].

En el API, se podría incluir funcionalidades para mantener conversaciones en base de datos, permitir la concurrencia y habilitar el streaming para mostrar información de manera progresiva. En la interfaz gráfica, considerar funciones como mantener un historial, filtrar por materia o mostrar una vista previa del vídeo.

Como se mencionó previamente, el potencial y utilidad de esta herramienta es inmenso para los estudiantes, tanto por su capacidad de hacer un seguimiento detallado del contenido como por todas mejoras que se pueden realizar en los componentes.

Además, cabe destacar, que una herramienta como esta, no solo facilita el acceso a recursos educativos, sino que también garantiza una experiencia de estudiante más eficiente y precisa a la hora de obtener la información que busca.

Por último, el uso de herramientas de inteligencia artificial tiene un enorme potencial en diversas aplicaciones. Sin embargo, en muchos casos, la fiabilidad puede verse compro-

[3]Técnica de few-shot examples.
[4]Técnica de Sentence Window Retrieval.

metida por la falta de transparencia sobre el origen de la información. En este sistema, se ha demostrado cómo es posible asegurar la fiabilidad proporcionando referencias claras al origen del conocimiento utilizado para responder a los mensajes del usuario. Esto no solo mejora la confianza en las respuestas, sino que también ofrece a los usuarios la posibilidad de verificar y profundizar en la información recibida.

Referencias

Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., y Liu, Z. (2024). *Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation.*

Facebook. (2024). *Faiss: A library for efficient similarity search and clustering of dense vectors.* Descargado de https://faiss.ai/

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., ... Sayed, W. E. (2024). *Mixtral of experts.* Descargado de https://arxiv.org/abs/2401.04088

LangChain. (2024a). *Applications that can reason. powered by langchain.* Descargado de https://www.langchain.com/

LangChain. (2024b). *Get started with langsmith.* Descargado de https://docs.smith.langchain.com/

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... Kiela, D. (2021). *Retrieval-augmented generation for knowledge-intensive nlp tasks.* Descargado de https://arxiv.org/abs/2005.11401

Meta. (2023). *React - a javascript library for building user interfaces.* Descargado de https://react.dev/

OpenAI. (2023). *Whisper by openai.* Descargado de https://openai.com/index/whisper/

Pinecone. (2023). *Pinecone rag rerankers.* Descargado de https://www.pinecone.io/learn/series/rag/rerankers/

Ramírez, S. (2023). *Fastapi.* Descargado de https://fastapi.tiangolo.com/

Richardson, L. (2023). *Beautiful soup 4.* Descargado de https://pypi.org/project/beautifulsoup4/

Team, T. M.-U. (2023). *Material-ui.* Descargado de https://mui.com/

Thaker M., Medium. (2024). *(part 2) build a conversational rag with mistral-7b and langchain.* https://medium.com/@thakermadhav/part-2-build-a-conversational-rag-with-langchain-and-mistral-7b-6a4ebe497185. (Accessed: 2024-06-10)