

SlackBot para Hyperloop UPV

Hugo Albert Bonet

Hyperloop UPV es un equipo de estudiantes que presenta una importante rotación anual (más del 70% de la plantilla suele renovarse de un año al siguiente), lo que requiere de grandes esfuerzos para mantener informadas a las nuevas generaciones sobre las tecnologías utilizadas en las ediciones anteriores. Además, para facilitar la separación del ámbito personal y profesional, la comunicación en el equipo se realiza a través de la aplicación de Slack.

En el presente trabajo se desarrolla un agente conversacional basado en GPT 3.5 Turbo con habilidad para distinguir entre distintas funciones, a parte de la propia actividad de un chatbot genérico, como enviar un correo electrónico o responder preguntas sobre trabajos previos realizados en el equipo de Hyperloop UPV. Todo ello integrado con la aplicación de Slack para mantenerlo dentro del entorno de trabajo del equipo.

Palabras clave: Retrieval Augmented Generation (RAG), Generative Pretrained Transformer (GPT), Chatbot, Slack

1 Introducción

1.1 Motivación

Hyperloop UPV es un equipo de estudiantes universitarios de la Universitat Politècnica de València encargados de desarrollar la tecnología del transporte del futuro, hyperloop, un vehículo capaz de levitar y desplazarse sin rozamiento a través de un tubo al vacío. Este funcionamiento le permite alcanzar grandes velocidades en paralelo con un importante ahorro energético.

Hyperloop UPV cumple este curso una década al frente del desarrollo de este medio de transporte. En estos diez años de trayectoria, el equipo ha fabricado más de 8 prototipos funcionales distintos, avanzando en sus implementaciones a gran velocidad, además de multitud de estudios ingenieriles y socioeconómicos. Demás, el equipo cuenta normalmente con 50 miembros, entre los que se encuentran, de media, alrededor de 13 miembros con más de un año de experiencia en el equipo y 37 completamente nuevos. Este aspecto hace evidente la necesidad de formar a las nuevas generaciones para conocer la historia del equipo y las tecnologías implementadas para que el aprendizaje de una generación persista en la siguiente. Para ello, se desarrollan numerosos documentos en los que se refleja el conocimiento adquirido hasta la fecha.

Sin embargo, el proceso de formación conlleva un esfuerzo superior, puesto que Hyperloop UPV es un equipo multidisciplinar. Esto implica la existencia de departamentos como el llamado *Partners*, formado por personas con grandes habilidades

comunicativas y negociadoras que deben conocer perfectamente las tecnologías del equipo y a su vez saber transmitir las a todos los públicos, sin tener muchas veces la capacidad de entender documentos tan técnicos.

Por estas razones, el presente trabajo consiste en desarrollar un agente conversacional capaz de facilitar este proceso interactivo de comprensión de las actividades relacionadas anteriormente en el equipo. Este chatbot permitirá una interacción común, como se realiza con herramientas como ChatGPT, además de permitir otro tipo de funciones: enviar un correo o realizar consultas sobre el equipo gracias a la tecnología de Retrieval Augmented Generation (RAG), que proporciona información sobre los documentos del equipo al agente conversacional.

Puesto que Hyperloop UPV valora en gran medida la separación del ambiente personal y profesional, todas las comunicaciones del grupo se realizan a través de la aplicación de Slack. Por ello, el agente estará integrado en esta misma aplicación, pudiendo interactuar con él como un miembro más del equipo.

1.2 Retrieval Augmented Generation (RAG)

La Retrieval Augmented Generation (RAG) o Generación Aumentada por Recuperación, es una técnica para aumentar la precisión de las respuestas de modelos de lenguaje. Para conseguirlo, se proporciona un contexto a cada *prompt* a partir de una consulta a una base de información concreta, comúnmente externa a los datos de entrenamiento. De esta manera, un modelo de lenguaje puede generar respuestas más precisas respecto a un ámbito específico sin necesidad de llevar a cabo un proceso de reentrenamiento y reduciendo el riesgo de olvidar información que había aprendido previamente. Dado que los Large Language Models (LLMs) presentan desafíos como las llamadas *hallucinations*, o alucinaciones, no podemos confiar en que la información proporcionada por ellos sea completamente veraz. RAG soluciona este problema al obligar al chatbot a consultar una fuente fiable antes de responder.

De este modo, la tecnología RAG proporciona numerosas ventajas como la restricción de las fuentes a las que puede acceder el modelo de lenguaje, una implementación más rentable, la capacidad de mantener el modelo actualizado de forma sencilla...

Su funcionamiento es el mismo que el de un *Query Engine* (Motor de Búsqueda), con la diferencia de que trabaja con una representación vectorial (*embeddings*) de la información. El usuario realiza una consulta (*query* o *prompt*) que es contrastada con una base de datos para encontrar la información más relevante al respecto. En este punto, en lugar de devolver los enlaces a los documentos más importantes relacionados con la consulta (como haría Google, por ejemplo), esta información se añade a la consulta inicial como contexto. De esta manera, la información enriquecida con el contexto sirve al LLM para generar una mejor respuesta.

A continuación, la figura 1 muestra un esquema de funcionamiento de un sistema RAG:

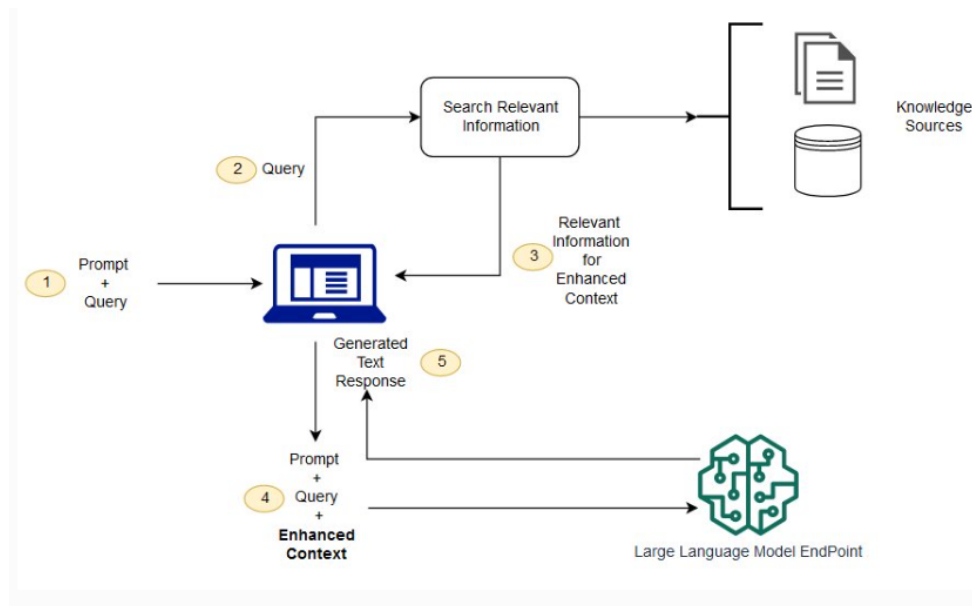


Figure 1: Esquema de un sistema RAG (1)

1.3 Limitaciones

Para el desarrollo de este trabajo, el tiempo y el presupuesto son las principales limitaciones. Por un lado, el requisito temporal establecido para su desarrollo limita la capacidad del sistema: cantidad de funciones posibles a realizar, optimización del funcionamiento, complejidad de las mismas... Por otro lado, la limitación en cuanto al acceso a recursos también es un aspecto a tener en cuenta, llevando al uso de GPT 3.5 Turbo en lugar de nuevos modelos como GPT 4o o GPT o1, o la necesidad de restringir la base de datos de consulta (puesto que su vectorización y consulta afectan al coste de este LLM).

En cuanto al código desarrollado para el proyecto, existe una limitación en el hecho de compartir ciertos archivos y carpetas, puesto que incluyen *api keys* e imágenes privadas. Todo ello ha sido eliminado de la carpeta que se comparte en la tarea.

2 Desarrollo e Implementación

Esta sección describe el desarrollo de la solución del SlackBot, así como las tecnologías empleadas para su implementación. La sección se encuentra desglosada en los distintos módulos que componen el sistema. En primer lugar se describe el módulo encargado de la combinación del resto, y posteriormente se detalla el funcionamiento de cada uno de ellos.

2.1 Welcome Bot

Welcome Bot es el módulo principal, encargado de unificar a todos los demás. Las tecnologías utilizadas en él incluyen *Flask* para funcionar como *backend* del bot, la API de Slack para interactuar con la plataforma, la API de OpenAI para utilizar el modelo de generación de embeddings de GPT 3.5 Turbo, *Llama Index* para la generación de la base de datos vectorial, módulos de tratamiento de distintas representaciones de

datos como JSON o YAML, y el resto de módulos del sistema de generación propia (LLM, FDD, Weather y PcCommand).

En primer lugar, encontramos el *snippet* de código que se muestra en la Figura 2. Este se encarga de la generación del índice vectorial. Previamente necesitamos haber establecido una variable de entorno `OPENAI_API_KEY` que contenga una API key válida poder interactuar con el modelo. A continuación, seleccionamos la codificación propia del modelo GPT 3.5 Turbo, para que la representación vectorial de los documentos sea compatible con el modelo que después utilizaremos para generar las respuestas. Después, se lee el directorio en el que se encuentran los documentos y se crea un índice vectorial a partir de los mismos. Por último, se guarda el índice de forma que persista en disco y no sea necesario volverlo a generar cada vez que se ejecute el programa, para ahorrar presupuesto y tiempo en cada ejecución. Este apartado del código permanece comentado a no ser que se añadan documentos, momento en el que se deberá volver a ejecutar una vez y puede volver a ser comentado.



```
1 service_context= Settings.llm=OpenAI(llm="gpt-3.5-turbo")
2 documents = SimpleDirectoryReader('./documents').load_data()
3 index = VectorStoreIndex.from_documents(documents, service_context=service_context)
4
5 # Save your index to a index.json file
6 index.storage_context.persist()
```

Figure 2: Creación del índice vectorial

Acto seguido, se crea una instancia de aplicación *Flask* y del módulo FDD, que crea la variable de entorno `OPENAI_API_KEY` en caso de que no haya sido creada en el paso anterior. Esta aplicación *Flask* se conecta con el adaptador de eventos de la API de Slack, y esta API con el token correspondiente a nuestro bot.

El grosor del programa se incluye en una función que se lanza únicamente cuando detecta un mensaje en Slack. Esta función recibe los datos del evento detectado, y solo actúa si el mensaje ha sido enviado por un usuario (para evitar contestarse a sí mismo en bucle o a otro que pueda existir en el mismo equipo de Slack). Además, almacena el ID del último mensaje recibido para evitar contestar por duplicado al mismo mensaje, ya que la API de Slack algunas veces vuelve a enviar el mismo evento por duplicado.

En el momento en el que se recibe un mensaje al que el chatbot debe responder, en primer lugar es analizado por el módulo LLM para decidir si el cliente pretende hacer uso de una de las funciones disponibles (por ejemplo, consultar información sobre el equipo) o si desea una interacción común con el agente conversacional. En caso de haber hecho una consulta relacionada con una de las funciones disponibles, se encarga de extraer de la consulta los argumentos necesarios para llevarla a cabo (por ejemplo, la dirección de correo a la que enviar el mensaje).

Una vez se conoce la función a ejecutar, el bot procede a llevarla a cabo y responder. Las distintas funciones posibles se muestran en las siguientes subsecciones.

2.2 Weather

El módulo **Weather** se utiliza para realizar consultas sobre el tiempo atmosférico actual en una ciudad dada. Emplea la WeatherAPI para realizar la consulta y devuelve el la temperatura y las condiciones atmosféricas en las que se encuentra. Acto seguido, el LLM recibe un JSON con esta información y ejecuta una función para interpretarla y convertirla en una respuesta en lenguaje natural, más amigable para el usuario.

2.3 PcCommand

Este módulo se encarga de demostrar cómo el agente podría ejecutar un comando del propio ordenador. En este caso, el agente detecta que el usuario pretende realizar una consulta a una página web, extrae del mensaje lo que se desea buscar y ejecuta Google Chrome para buscarlo.

Además, este módulo incluye la función de *look_for_pictures* que se encarga de buscar un número dado de imágenes (en adelante k) que tengan la mejor coincidencia con una frase (en adelante *positive*) y la menor posible con otra cadena de texto (en adelante *negative*).

Para conseguirlo, se emplea un modelo denominado CLIP cuyo objetivo es crear representaciones, o *embeddings*, similares para una imagen y su descripción. Por ello, todas las imágenes deben procesarse para generar una base de datos de embeddings (para esta demo, se han utilizado mis fotos personales extraídas mediante la API de Google Photos). Una vez almacenadas, se codifican tanto *positive* como *negative* y se calcula su similitud con cada una de las imágenes (producto vectorial). Al haber conseguido un score para cada imagen, procedemos a calcular el score final: $score_{final} = score_{positive} - score_{negative}$. Finalmente, se seleccionan las k imágenes mejor rankeadas con esta métrica y se le muestran al usuario.

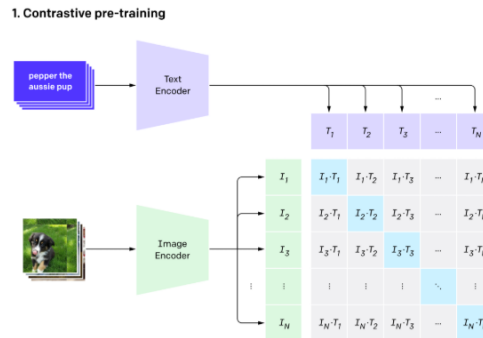


Figure 3: Diagrama oficial de CLIP, por OpenAI

Esta función tendría grandes aplicaciones. Por ejemplo, el equipo creativo podría recibir una tarea para una publicación en redes sociales, y desde la propia aplicación de Slack podría solicitar al chatbot fotos que se adecúen con su tarea. Todo ello expresado por completo en lenguaje natural, y con mucha mayor adaptabilidad que una búsqueda basada, por ejemplo, en etiquetas.

2.4 LLM

Este es uno de los módulos principales del sistema. Es el encargado de la interacción con el propio modelo de lenguaje subyacente, para detectar la función que el usuario

busca y generar las respuestas apropiadas a sus consultas. Consta de dos funciones principales, una para procesar la respuesta y decidir qué función lanzar, y otra para generar una respuesta.

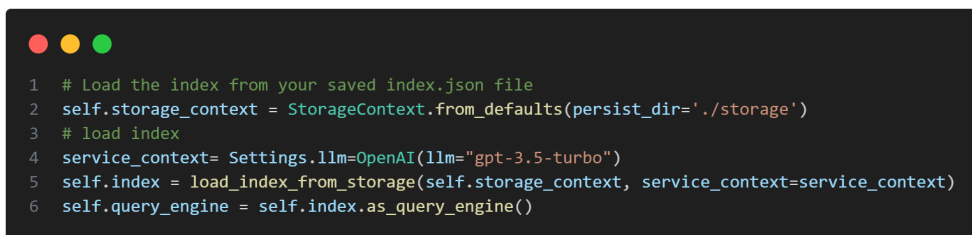
En la primera función, se llama a la función de *chat completion* de la API de OpenAI, a la que se le proporciona la pregunta del usuario y una serie de funciones entre las que decidir. Cada función posee un nombre, que será el que devuelva el modelo, una descripción en lenguaje natural, que utilizará para decidir qué función elegir, y una serie de argumentos. Estos argumentos vienen descritos y deben incluir el tipo de dato que se busca. Una vez analizada la pregunta, si el modelo considera que el usuario buscar aprovechar alguna de las funciones, devuelve el nombre de la función a ejecutar y sus argumentos, que nuestro módulo de generación propia se encarga de formatear en un JSON para ser devuelto junto con el propio mensaje del usuario.

La segunda función genera una respuesta final en lenguaje natural. Tienen en cuenta dos contextos distintos, uno en el que se ha ejecutado una de las funciones y otro en el que no. En el caso de no haber ejecutado ninguna de las funciones, el mensaje del usuario es transmitido al modelo de lenguaje para generar una respuesta. En caso contrario, el modelo de lenguaje recibe también la información proporcionada por la función que se ha ejecutado (como por ejemplo, la temperatura actual en Valencia) y la utiliza para generar la respuesta.

2.5 FDD

Este es el otro módulo clave de la aplicación, el encargado de llevar a cabo las funciones RAG para adquirir información veraz sobre el equipo de Hyperloop UPV. Sus siglas vienen de *Final Demonstration Documentation*, el documento empleado para la demo del sistema. Este documento incluye información sobre el último vehículo de Hyperloop UPV, *Vèspèr*, en detalle, ya que es la documentación adjuntada para los jueces de la pasada competición en la que el equipo se proclamó campeón.

En primer lugar, como se muestra en el *snippet* de código de la Figura 4, establecemos el directorio en el que se encuentra nuestro índice vectorial de forma persistente. A continuación, establecemos el modelo cuya codificación utilizaremos y posteriormente cargamos el índice y lo transformamos en un motor de búsqueda.



```
1 # Load the index from your saved index.json file
2 self.storage_context = StorageContext.from_defaults(persist_dir='./storage')
3 # load index
4 service_context= Settings.llm=OpenAI(llm="gpt-3.5-turbo")
5 self.index = load_index_from_storage(self.storage_context, service_context=service_context)
6 self.query_engine = self.index.as_query_engine()
```

Figure 4: Creación del motor de búsqueda RAG

Como se puede observar, es imprescindible decodificar la información almacenada en el índice con el mismo modelo con el que se generaron los embeddings para obtener resultados válidos.

Una vez hemos instanciado el motor de búsqueda, el principal objetivo de este

módulo es realizar una consula o query al sistema y devolver su respuesta.

3 Guía de Uso

Esta sección detalla los pasos a seguir para poder ejecutar el chatbot como desarrollador y una demostración de su uso a nivel usuario. Antes de comenzar, debemos crear nuestro bot desde la API de Slack, asignarlo a un canal y dotarle de los permisos necesarios para nuestra app (en nuestro caso, acceso al historial de mensajes).

En primer lugar, debemos abrir la consola de comandos y ejecutar el comando `ngrok http puerto`, donde *puerto* debe ser el puerto establecido en la aplicación Flask, en nuestro caso el 5000. Nos aparecerá la siguiente información:

```
Version      3.19.0
Region      Europe (eu)
Latency      34ms
Web Interface http://127.0.0.1:4040
Forwarding   https://eb42-88-17-177-213.ngrok-free.app -> http://localhost:5000

Connections  ttl    opn    rt1    rt5    p50    p90
              0      0      0.00   0.00   0.00   0.00
```

Figure 5: Ejecución de Ngrok

Podemos observar que en el apartado *Forwarding* nos proporciona una URL que deberemos copiar. A continuación, después de haber ejecutado nuestro programa `welcomebot.py`, deberemos entrar en el apartado de *Event Subscriptions* y cambiar la URL por la que acabamos de copiar, añadiendo al final `/slack/events`, que es la dirección en la que nuestro bot está escuchando. Una vez se verifique la url, estamos preparados para utilizar la aplicación.

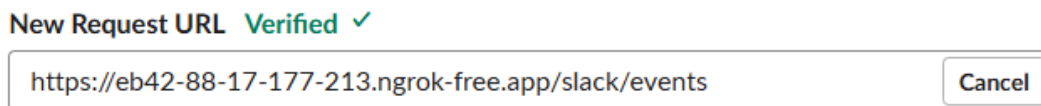


Figure 6: URL verificada

Una vez verificado, deberemos encontrar tanto en nuestra consola de comandos de ejecución de Ngrok como en la que hayamos ejecutado el fichero Python el código 200 señalando una correcta conexión. En este momento, nuestro bot está preparado para ayudarnos en lo que necesitemos. Abriremos Slack en el entorno en el que se encuentre nuestro bot. Debemos crear un canal y permitir la interacción con nuestro chatbot desde el apartado *Integraciones*.

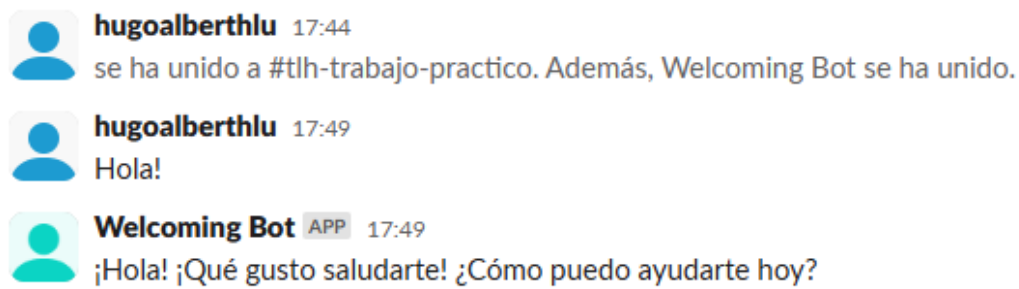


Figure 7: Primera contestación del chatbot

Ahora que ya es capaz de contestarnos a mensajes que no están relacionados con ninguna función, vamos a comprobar el correcto funcionamiento de sus funcionalidades. En primer lugar, le preguntaremos por el tiempo en la ciudad de Valencia. Cabe destacar que, pese a que la descripción viene dada en inglés por la API, el agente traduce al idioma en el que le estamos hablando.

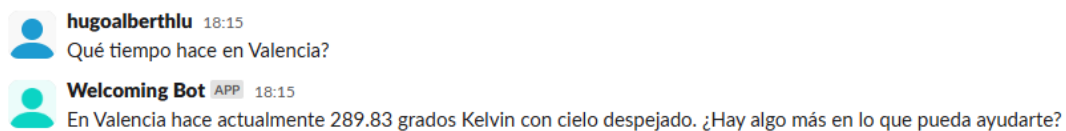


Figure 8: Pregunta sobre el clima

La siguiente funcionalidad a evaluar es la búsqueda a través de Chrome. Para ello, preguntaremos por un motor de inducción, el que se utiliza en el equipo de Hyperloop UPV.

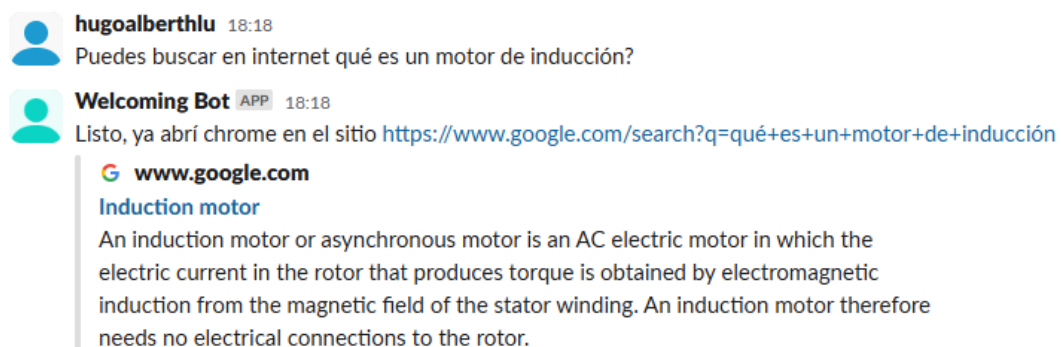


Figure 9: Búsqueda en Chrome

Además del enlace mostrado en la imagen, el bot ha abierto automáticamente la pestaña en la aplicación de Chrome en la que estaba usando Slack.

También pondremos a prueba la búsqueda de imágenes (*image retrieval*) mediante el sistema implementado basado en CLIP.

hugoalberthlu 01:04
Quiero encontrar tres imágenes de gente trabajando en tecnología punta, pero quiero evitar que sean memes ni capturas de pantalla

Welcoming Bot APP 01:05
Estoy buscando las imágenes, ¡en un momento las verás!

Figure 10: Búsqueda de imágenes

Muestra tres imágenes que se corresponden correctamente con la descripción, una de ellas mostrada a continuación:



Figure 11: Imagen mejor rankeada a partir de la query

La última funcionalidad es la principal, hacer preguntas sobre el equipo de Hyperloop UPV, el propio sistema RAG basado en documentos de la organización. A continuación se muestra un ejemplo.

hugoalberthlu 18:30
Qué modelos comparó Hyperloop UPV en su investigación socioeconómica?

Welcoming Bot APP 18:30
Hyperloop UPV comparó su modelo con GPT 3.5 Turbo y GPT 4 en su investigación socioeconómica.

hugoalberthlu 18:30
Qué datos analizó el equipo en esta investigación?

Welcoming Bot APP 18:30
El equipo analizó comentarios, artículos de noticias y videos de diferentes plataformas de redes sociales como Google News, Bing News, Reddit y YouTube. Se recopilaban noticias en inglés y español, comentarios relacionados con Hyperloop en Reddit, y videos en inglés en YouTube. También se menciona que se descartaron plataformas como Instagram y TikTok debido a la falta de datos o a métodos no éticos para obtener la información.

Figure 12: Preguntas acerca de la investigación de Hyperloop UPV

4 Conclusiones

En conclusión, los modelos conversacionales basados en LLMs presentan un mayor potencial que los chatbots RASA observados en clase, con una mayor generalización y capacidad de investigación. Hemos podido observar cómo su integración con la selección de funciones y detección de argumentos para las mismas les permiten con una de las funciones principales de los chatbots clásicos, la capacidad de preguntar por datos concretos para llevar a cabo su tarea.

Además, la tecnología RAG nos permite consultar múltiples fuentes de información diferente para obtener unas respuestas más precisas. Hemos podido descubrir cómo el modelo es capaz de dar respuestas concretas a preguntas específicas sobre la investigación llevada a cabo en el equipo, o incluso consultar en línea la temperatura actual de la ciudad que se solicite, y no es más que el principio. En concreto, las bases de datos de embeddings vectoriales permiten comprender el contenido semántico de las *queries* y las *keys* para adquirir el contexto más adecuado a cada pregunta.

4.1 Trabajo Futuro

Este chatbot todavía tiene muchos puntos de mejora, entre los que considero interesantes:

- Integración otras herramientas proporcionadas por el framework de *Llama Index*, como por ejemplo el enlace con el buscador DuckDuckGo para dotar al agente con la capacidad de buscar en la red.
- Integración de más documentos del equipo para ampliar la información disponible.
- Registro del contexto de la conversación para dar memoria al agente y darle la capacidad de comprender el sentido de la conversación.

References

- [1] 1, A. W. S. ¿Qué es la RAG (generación aumentada por recuperación)?