RAG de la Coppermind, una Wikipedia del Cosmere

Ferrán Plana Caminero

Repositorio en *Github*: https://github.com/FerranPlanaCoEIA/TFB.git

Introducción

La técnica *RAG* (*Retrieval-Augmented Generation*) es una de las principales aplicaciones de la Inteligencia Artificial Generativa, siendo de las primeras en aplicarse en entornos productivos. Esta es una técnica que permite a los modelos generativos acceder dinámicamente a grandes fuentes de datos, generando respuestas precisas y rápidas a las preguntas de los usuarios. Esto, sumado a la posibilidad de poder *chatear* con el modelo e iterar hasta que se encuentre la respuesta, hace de los *RAG* la técnica abanderada de los modelos generativos.

Un *RAG* está compuesto por dos partes: *retrieval* y generación de la respuesta. Durante la parte del *retrieval* se encuentran los fragmentos de la base de datos que más coinciden semánticamente con la pregunta del usuario, y en la parte de la respuesta se utiliza un modelo generativo para que, con los fragmenos encontrados, responda a la pregunta.

Para poder abordar el funcionamiento de los *RAGs* debe entenderse primero qué son los *embeddings*. Un *embedding* es una representación vectorial de una palabra, frase o fragmento de texto. Es, al fin y al cabo, la transformación de un fragmento de texto a un vector de N dimensiones. Esta transformación guarda información acerca del significado de ese fragmento. De esta forma, dos palabras con significados similares se representarán con vectores similares, muy cercanos entre sí, que forman un ángulo muy pequeño. El coseno de este ángulo es la similitud; dos vectores muy similares tendrán una similitud cercana a 1, si no tienen relación será cercana a 0 y si tienen relación opuesta tenderá a -1, aunque esto último no es muy habitual. Recapitulando, el *embedding* de rey y el de reina tendrán una similitud cercana a 1, mientras que el de rey y mesa cercana a 0. Pero no solo eso, sino que la similitud del de rey y reina será un valor cercano al de hombre y mujer, ya que la relación semántica es la misma (ver figura 1).

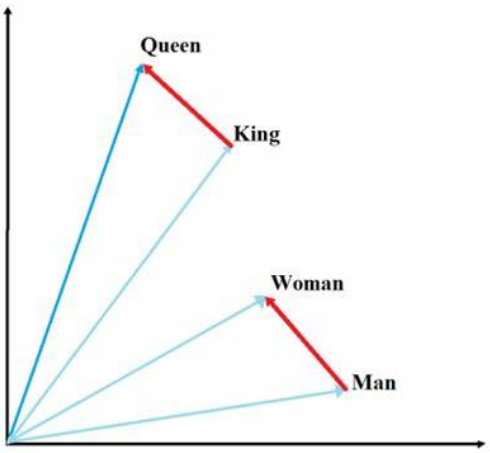

Figure 1: Representación en 2D de los *embeddings* de las palabras rey, reina, hombre y mujer.

Entendido esto, el primer paso para crear un *RAG* es dividir el texto de la base de datos en fragmentos que se conocen como *chunks*. La longitud de estos fragmentos es uno de los hiperparámetros de estas soluciones, cuyo valor óptimo depende de la base de datos y se obtiene empíricamente. Además del tamaño de los *chunks* se juega con otro hiperparámetro: el solapamiento. El solapamiento es la cantidad final de un *chunk* que está presente al principio del siguiente, y se aplica para preservar algo de contexto y evitar perder información crítica. Una vez se ha dividido todo el contenido de la base de datos en *chunks* se calculan los *embeddings* de cada *chunk* y se almacenan junto a su contenido en lo que se llama índice de la base de datos.

A continuación, cuando se recibe la pregunta del usuario se calcula el *embedding* y su similitud con el de cada *chunk* de la base de datos. Los *chunks* con mayor similitud con la pregunta serán los que más se parezcan semánticamente y los que probablemente contengan la respuesta a la pregunta. El número de *chunks* con mayor similitud que nos quedamos es otro de los hiperparámetros que podemos modificar, y se conoce como *top-n*. De nuevo, el valor óptimo depende de la base de datos.

Por último, teniendo esos *top-n chunks* con mayor similitud, se le pasa su contenido (¡el texto, no el vector!) junto con la pregunta del usuario a un *LLM* al que se le pide que responda a la pregunta con el contexto proporcionado. ¡Y *voilà*! Ya tienes un sistema *RAG* completamente funcional. Tanto el modelo de generación de la respuesta como sus parámetros (temperatura, *top-p*, *top-k*, *presence penalty*, etc.) son más hiperparámetros con los que se puede jugar.

Objetivos

A continuación detallo los objetivos básicos de este trabajo. Más allá de estos se ha perseguido algún otro y muchos se dejan como trabajo a futuro.

- Diseñar un *RAG* funcional a modo de pregunta-respuesta, no *chatbot*.
  - Elaborar un test con el que poder elegir los valores óptimos de los hiperparámetros.
  - Definir métricas para poder comparar los resultados.
  - Seleccionar los hiperparámetros más óptimos: *chunk size* , *overlap*, modelo de *embeddings*, *top-n*, modelo de generación de la respuesta, etc.
- Elaborar una interfaz sencilla en la que poder hacer las preguntas.

Desarrollo

I. Creación de la base de datos

En primer lugar, he elegido la lista de entradas de la *wikipedia* que quiero utilizar como base de datos. Me he decantado por centrarme no en una saga de libros en concreto, sino en los conceptos que son comunes a toda saga, a todo el universo del *Cosmere*.

Adjunto la *lista* con las urls de las entradas elegidas. He hecho un script que descarga el html a partir de la lista de urls. La descarga de cada entrada, al ser de una web, ha sido en formato html.

I.II. Tratamiento de los htmls

Para optimizar el RAG he convertido los htmls en markdown. Además, he hecho cierto procesamiento para eliminar varias cosas:

- Vínculos a otras entradas de la *wikipedia*.
- Citas o referencias.
- Imágenes.
- Sección de notas en adelante (sección donde se ponen todas las referencias, no aporta valor).

De momento no voy a hacer más tratamiento al texto. Más adelante, con la ayuda del Test Automático, estudiaré si sería conveniente.

I.III. Embeddings

Una vez procesado el contenido de cada entrada, he dividido el texto completo de la base de datos en *chunks* con un *overlap* de un 20% del *chunk size*(este último está por ver, a estudiar con el Test Automático). He hecho los *embeddings* con el modelo *paraphrase-MiniLM-L6-v2*, un modelo de transformers relativamente compacto y eficiente cuya principal aplicación es la búsqueda semántica. He obtenido el modelo de *Hugging Face*.

El código divide el texto en *chunks*, hace los *embeddings* de cada *chunk* y guarda en local cada *chunk* y *embedding* con varios metadatos asociados: el número de *chunk* (entre todos los *chunks* del documento), el número de documento o *DOCID* (número de entrada de las totales) y su nombre (la *url* asociada, o algo similar).

II. Funcionamiento del *RAG*

Como hemos dicho antes, un modelo *RAG* tiene dos partes principales: búsqueda semántica y generación de la respuesta. La búsqueda semántica consiste en hallar un cierto número de *chunks* de la base de datos que tienen mayor similitud con la pregunta del usuario. Una vez hallados, se le pide a un *LLM* componer una respuesta en base a esa pregunta y los chunks encontrados.

II.I. Búsqueda semántica

Hechos y guardados los *embeddings*, podemos acceder a ellos en cualquier momento. Al hacer una pregunta, se hace el *embedding* de esa pregunta y se calcula la similitud entre esa pregunta y todos los *embeddings* de la base de datos. Otro de los hiperparámetros, junto con el tamaño de *chunk* y el *overlap*, es *top-n*, el número de chunks con más similitud que nos quedamos. El valor óptimo de este hiperparámetro también lo decidiremos con la ayuda del Test Automático.

Al hacer una pregunta, obtenemos los *chunks* con mayor similitud, el número de *chunk*, el *DOCID*, su nombre y la similitud entre ese *chunk* y la pregunta. Quedaría esto:

> Question: ¿Se puede leer algún libro del Cosmere sin haber leído otras sagas? The 3 chunks most similar to your question are:
>
> 1. DOCID: 12 | Document Name: https://es.coppermind.net/wiki/Cosmere | Chunk number: 3 | Similarity: 0.6855
> subyacentes, apareciendo algunos personajes en otros mundos ajenos al suyo. A pesar de las conexiones, Brandon ha dejado claro que uno no necesita ningún conocimiento del Cosmere en general para leer, entender, o disfrutar de los libros que tienen lugar en él. Las secuencia principal del Cosmere consistirá en la saga *Dragonsteel*"), la trilogía de *Elantris*, al menos cuatro eras de la saga *Nacidos de la bruma*") y *El archivo de las tormentas*. La historia del Cosmere no incluye ningún libro que haga referencia a la Tierra, puesto que la tierra no está en el Cosmere. Para una lista completa
> 2. DOCID: 19 | Document Name: https://es.coppermind.net/wiki/Esquirla del Amanecer | Chunk number: 49 | Similarity: 0.6744
> no se refiera a Sigzil. En *Viento y verdad*, se confirmó que Hoid tuvo en su poder la Esquirla del Amanecer Existe durante los sucesos de los libros 1-5 de *El archivo de las tormentas*. ## Notes 1. ↑ a b c d e f g h i j k l m n o p q r s t u *Esquirla del Amanecer (novella)* capítulo 19")Summary: Esquirla del Amanecer (novella)/Chapter_19/Chapter 19 (la página no existe)")#/Chapter 19") 2. ↑ a b General Reddit 2022 — Arcanum - 2022-12-02Cite: Arcanum-15961# 3. ↑ a b c Dawnshard Annotations Reddit Q&A — Arcanum -
> 3. DOCID: 24 | Document Name: https://es.coppermind.net/wiki/Hoid | Chunk number: 184 | Similarity: 0.6562

qué libro fue eso respondió *Brazales de Duelo").* * La misión de Hoid quizás sea: «hacer aquello que una vez fue». * Hoid no está impresionado por los Sangre Espectral. * Hoid detesta al Grupo, y los miembros de este último que le conocen también le detestan. * Aunque una vez le dijo a Kaladin (bastante acertado) que su vida comenzó como palabras en una página, este hecho no tenía la intención de romper la cuarta pared. * Hoid se ha travestido en el pasado, «muchas veces». * Antes de los eventos de *Palabras Radiantes*, a Hoid no le habían

Además, se puede obtener una gráfica interesante: el histograma de la similitud entre cada *chunk* de la base de datos y la pregunta (ver figura 2).

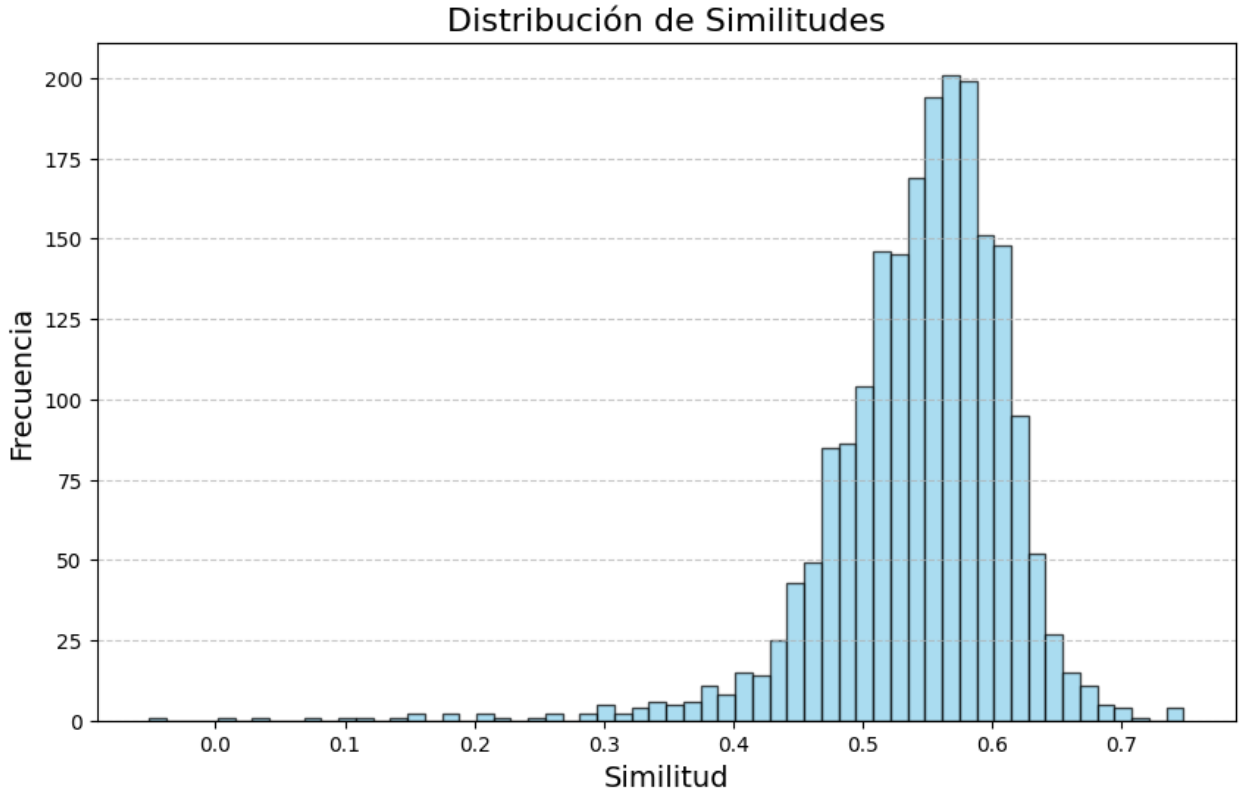

Figure 2: Distribución de *chunks* con mayor similitud respecto a una pregunta dada.

II.II. Generación de la respuesta

Esos *chunks* con mayor similitud con la pregunta se le pasan, junto con la pregunta del usuario, a un *LLM*, pidiéndole que responda a la pregunta del usuario en base a los *chunks* encontrados.

El modelo seleecionado se decidirá más adelante con el Test Automático. Para esta y posteriores llamadas a *LLMs* se utilizarán varios proveedores que permiten hacer llamadas gratis vía API (ver *LLMs_free_API_keys.ipynb* en la carpeta "Apoyo" del repositorio de *Github* para más detalles).

El *system prompt* es el siguiente:

> Eres un asistente virtual experto en responder preguntas. A continuación vas a recibir la pregunta de un usuario y el conocimiento que debes utilizar para responderla en el siguiente formato:
>
> Pregunta: Pregunta del usuario Conocimiento: Nombre del documento 1: contenido del documento 1 Nombre del documento 2: contenido del documento 2 . . . Nombre del documento N: contenido del documento N

> Responde como si fueras un chatbot de una wikipedia. Después de dar tu respuesta completa, di el nombre de los documentos en los que te has basado para elaborarla. Si te has basado dos veces en el mismo documento, no lo repitas al referenciarlo. Hazlo en este formato:
>
> Pon aquí tu respuesta [[Pon aquí únicamente el nombre del primer documento en el que te has basado]] [[Pon aquí únicamente el nombre del segundo documento en el que te has basado, siempre y cuando no hayas puesto el mismo nombre antes]] ...
>
> No utilices conocimiento propio de tu entrenamiento, utiliza solo el que se te proporciona. Si parte del conocimiento que se te proporciona no te sirve para responder a la pregunta, no lo utilices. Cita únicamente los documentos en los que te has basado para elaborar la respuesta. Si con el conocimiento que se te proporciona no puedes responder a la pregunta, responde únicamente "Lo siento, no puedo responderte a esa pregunta" y no cites ningún documento.

El *user prompt* es el siguiente:

> Pregunta: {Pregunta del usuario}
>
> Conocimiento:
> {Nombre del documento del chunk}: {contenido del chunk}

Como se puede observar, no solo se le pide al *LLM* que componga la respuesta, sino que cite sus fuentes. De esta forma el usuario puede, con solo pinchar en el nombre de la referencia, acceder a dicha entrada de la *wikipedia* mediante la *url*.

III. Test Automático

Con el objetivo de poder valorar si los cambios tienen un impacto positivo en el modelo, he creado un test de regresión, al que llamaré Test Automático. Este test consiste en 137 preguntas de las que sé la respuesta correcta, a la que llamaré respuesta *best*, y la entrada (o entradas) de la wikipedia donde se responde a esa pregunta, que llamaré documento *best*. Las métricas que utilizaré son las siguientes:

- OK RAG: Porcentaje de preguntas en las que el documento/s *best* se encuentra entre los encontrados por la búsqueda semántica.

- OK FUENTES: Porcentaje de preguntas en las que el documento/s *best* se encuentra entre los elegidos y referenciados por el *LLM* para redactar la respuesta.

- OK RESPUESTA: Porcentaje de preguntas en las que un segundo *LLM* valora si la respuesta del primero se ajusta a la respuesta *best*. El *LLM* elegido para el test de *LLM as a judge* ha sido el *Llama 3.3 70B versatile*, ya que es un *LLM* grande, la útima versión de los modelos *LLama* (hasta el lanzamiento del *Llama 4*) y apto para gran variedad de tareas.

El *system prompt* es el siguiente:

> Vas a recibir una pregunta de un usuario (Pregunta), la respuesta correcta a esta pregunta (Respuesta_Best) y una respuesta generada (Respuesta_Generada). Tus objetivos son los siguientes: 1. Determinar si la Respuesta_Generada responde a la Pregunta. 2. Determinar si la Respuesta_Generada concuerda con la Respuesta_Best y no la contradice. Tienes que hacer una valoración en detalle y razonando sobre tu valoración. Si la Respuesta_Generada no responde a la Pregunta, valorar como KO. Si la Respuesta_Generada concuerda con la Respuesta_Best y no la contradice, valorarla como OK. Si la Respuesta_Generada no concuerda con la Respuesta_Best y la contradice, valorarla como KO. No tener en cuenta posibles detalles adicionales que puedan estar incluidos en la Respuesta_Generada, siempre y cuando no contradigan la Respuesta_Best.
>
> Una vez hayas hecho tu razonamiento, cúentalo, y al final pon tu valoración en este formato: [[Valoración: OK/KO]]

El *user prompt* es el siguiente:

> Pregunta:
> {Pregunta del usuario}
>
> Respuesta_Best:
> {Respuesta marcada como correcta}
>
> Respuesta_Generada:
> {Respuesta generada por el *LLM*}

Las preguntas, documento/s *best* y respuestas *best* se pueden ver aquí: *Input Test Automático.xlsx*. El registro de todos los tests se puede ver aquí: *Registro de pruebas.xlsx*

III.I. *Chunk size*, *overlap* y *top n*

El primer objetivo de este test es determinar el tamaño óptimo de los *chunks* de la base de datos. Para eso se ha ejecutado el test con varios *chunk size* distintos, así como para varios *top n* (el número de *chunks* pasados al *LLM* para redactar la respuesta). En este caso se ha fijado el *LLM* de elaboración de la respuesta y varios de sus parámetros. El *LLM* ha sido *Google Gemini 2.0 pro experimental*, aunque más adelante se comaprarán varios modelos y se eligirá el mejor. La temperatura se ha fijado a 0 para aumentar la reproducibilidad de los tests y reducir su variabilidad. Además, se ha fijado la *repetition penalty* a 0 para intentar producir respuestas concisas. Por último, el *chunk overlap* se ha fijado por defecto al 20%.

Se han hecho estas pruebas para un *chunk size* de 50, 75, 100 y 200 tokens, y *top n* de 3, 5, 10 y 15 *chunks* (ver figura 3).

Chunk_size (tokens)	top_n (chunks)	OK RAG (%)	OK FUENTES (%)	OK RESPUESTA (%)
50	3	68,38	55,15	47,06
	5	78,68	61,76	55,15
	10	87,50	72,79	66,18
	15	90,44	74,26	69,85
75	3	63,97	48,53	42,22
	5	72,79	55,88	55,88
	10	83,09	63,97	63,97
	15	86,76	69,12	69,85
100	3	66,18	51,47	44,85
	5	75,74	60,29	52,94
	10	**84,56**	**66,91**	**66,18**
	15	88,24	69,85	72,79
200	3	61,03	47,06	44,12
	5	65,44	53,68	52,21
	10	74,26	53,68	56,62
	15	79,41	58,09	59,56

A raíz de estos resultados se utilizará un *chunk size* de 100 *tokens* (por tanto, un *chunk overlap* de 20 *tokens*) y un *top n* de 10 *chunks*. Se escoge esto por varias razones. En primer lugar, es el que mayor porcentaje de OK arroja en OK de la respuesta para *top n* de 10, junto a un *chunk size* de 50. Se elige sobre este porque, para resultados iguales, un *chunk size* de 100 ofrece más contexto. No se escoge *chunk size* de 100 y *top n* de 15 porque la mejora en esta métrica no es demasiada. Además, hay que tener en cuenta el tiempo de respuesta del modelo, una métrica que en este caso no se ha evaluadao pero que es fundamental para este tipo de soluciones. En resumen, los parámetros elegidos han sido ***chunk size* = 100 *tokens*, *chunk overlap* = 20 *tokens*, *top n* = 10 *chunks***.

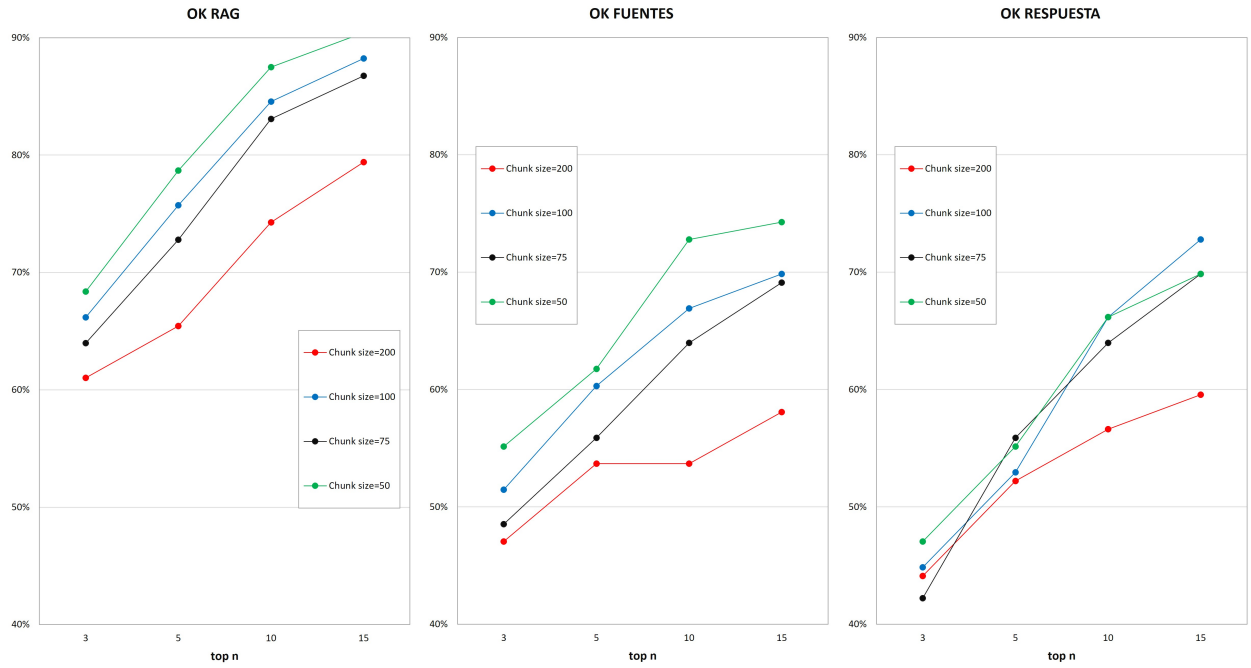

Figure 3: Valor de las métricas OR RAG, FUENTES y RESPUESTA en función del *chunk size* y *top n*, con *chunk overlap* del 20% del *chunk size*, modelo de *embeddings paraphrase-MiniLM-L6-v2* y modelo de elaboración de la respuesta *Google-Gemini 2.0 pro exp 02/05* (temperatura y *repetition penalty* a 0).

Por otro lado, como más adelante se va a analizar el *LLM* a utilizar para responder a las preguntas, cabría preguntarse si este análisis fijando el *LLM* es válido para otros. La realidad es que no, pero se ha hecho así para reducir el número de pruebas a hacer. Aunque los resultados del análisis del *chunk size*, *overlap* y *top n* probablemente cambien de un *LLM* a otro, se ha supuesto que no serán cambios significativos.

III.II. Modelo de *embeddings*

En el momento en que se hicieron los tests III.I pensaba que el modelo de *embeddings* que estaba utilizando, *paraphrase-MiniLM-L6-v2*, era el mejor. Sin embargo, en una de las clases me hicieron saber que este modelo no está entrenado en español, por lo que es fundamental encontrar uno entrenado específicamente en español.

Lo ideal sería hacer primero este test y después el III.I, ya que es más determinante el modelo de *embeddings* utilizado. Sin embargo, como el test III.I consume mucho tiempo, asumiremos el error producido por hacerlo en este orden.

Los modelos de *embeddings* que vamos a comparar son los siguientes:

- *paraphrase-MiniLM-L6-v2* (as-is, entrenado solo en inglés)
- *paraphrase-multilingual-MiniLM-L12-v2*
- *paraphrase-multilingual-mpnet-base-v2*
- *distiluse-base-multilingual-cased-v2*
- *stsb-xlm-r-multilingual*
- *Shaharyar6/finetuned_sentence_similarity_spanish*

Por desgracia, y debido a las limitaciones de tener que usar llamadas gratis via API a *LLMs* ofrecidos por distintos proveedores, el *LLM* de elaboración de la respuesta que usé en el test III.I, *Google: Gemini Pro 2.0 Experimental (free)*, ya no está disponible. Debido a esto voy a tener que utilizar otro, *Google: Gemini 2.0 Flash Thinking Experimental 01-21 (free)*. Aún así, esto no invalida las conclusiones de este test.

Los resultados de este test son los siguientes (ver figura 4):

Modelo de embeddings	OK RAG (%)	OK FUENTES (%)	OK RESPUESTA (%)
paraphrase-MiniLM-L6-v2 (inglés)	84,56	66,18	61,03
paraphrase-multilingual-MiniLM-L12-v2	88,97	77,21	80,74
paraphrase-multilingual-mpnet-base-v2	89,71	76,47	78,68
distiluse-base-multilingual-cased-v2	**88,24**	**77,94**	**82,35**
stsb-xlm-r-multilingual	67,65	47,06	56,62
Shaharyar6/finetuned_sentence_similarity_spanish	92,65	82,35	83,82

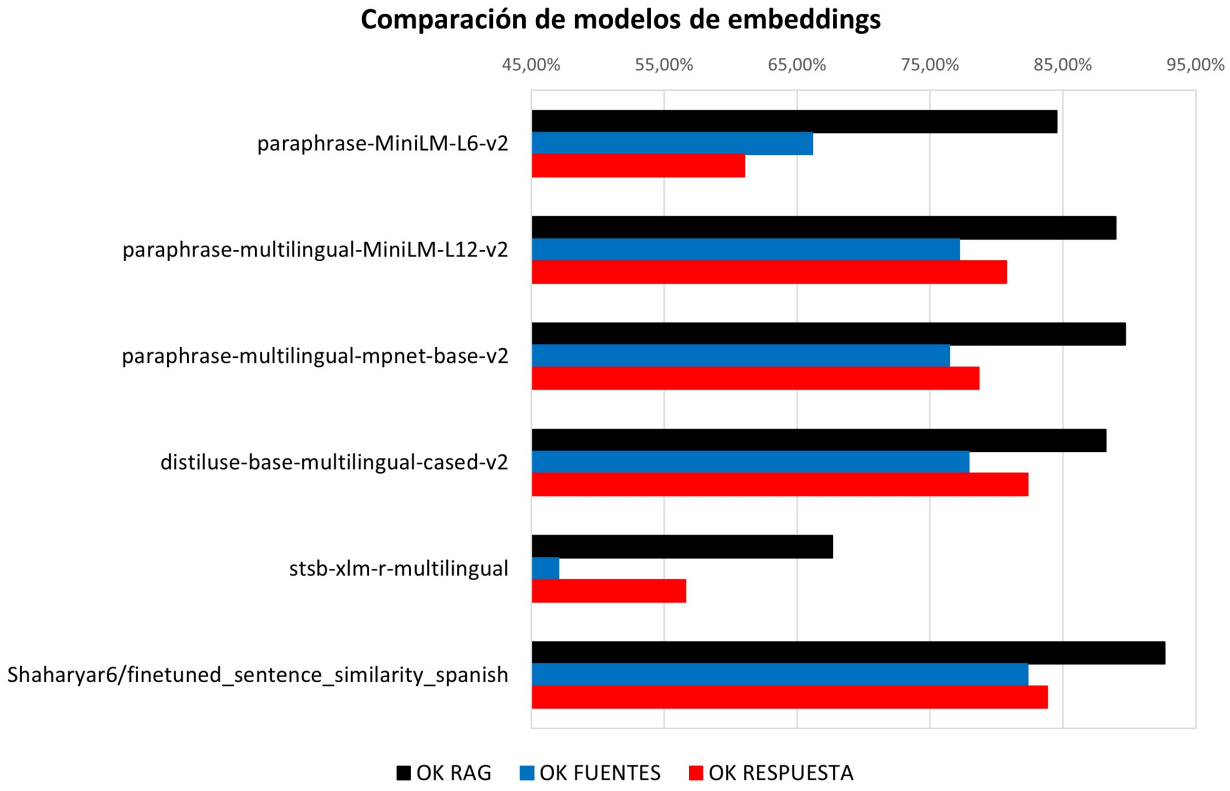

Figure 4: Valor de las métricas OR RAG, FUENTES y RESPUESTA en función del modelo de *embeddings*, con *chunk size* de 100 *tokens*, *chunk overlap* de 20 *tokens*, *top n* de 10 *chunks* y modelo de elaboración de la respuesta *Google: Gemini 2.0 Flash Thinking Experimental 01-21* (temperatura y *repetition penalty* a 0).

Podemos observar que, aunque OK RAG del modelo entrenado en inglés es similar al resto (entrenados en español), en las otras dos métricas es muy inferior. Es decir, los modelos de *embeddings* entrenados en español están encontrando chunks mucho más útiles para elaborar la respuesta.

Que haya más OK RESPUESTA que OK FUENTES se explica por la naturaleza de la base de datos: como es una *wikipedia* tiene mucha información redundante, así que es probable que para alguna pregunta haya más documento/s *best* de los que he puesto en el test automático.

Sin embargo, lo que más llama la atención son los pésimos resultados del modelo *stsb-xlm-r-multilingual*. Este es un resultado soprendente, porque en otros trabajos que he hecho este modelo ha dado buenos resultados. Una posible explicación es que, aunque en la documentación oficial se dice que este modelo está entrenado con unos 40 idiomas, el español no sea uno de ellos.

Como los resultados de los modelos entrenados en español (quitando el mencionado anteriormente) son similares, vamos a elegir basándonos también en el tiempo de inferencia de cada modelo. Aunque no está implementado el cálculo del tiempo de inferencia de cada pregunta del test, podemos estimarlo otra forma. Como el número de *chunks* y *tokens* de la base de datos es fijo, podemos medir el tiempo que tarda cada modelo en crear el índice. Esto nos dará una idea de qué modelos tardan menos en hacer una inferencia (ver figura 5).

Modelo de embeddings	Tiempo de creación del índice (s)
paraphrase-MiniLM-L6-v2	85
paraphrase-multilingual-MiniLM-L12-v2	154
paraphrase-multilingual-mpnet-base-v2	531
distiluse-base-multilingual-cased-v2	**260**
stsb-xlm-r-multilingual	365
Shaharyar6/finetuned_sentence_similarity_spanish	447

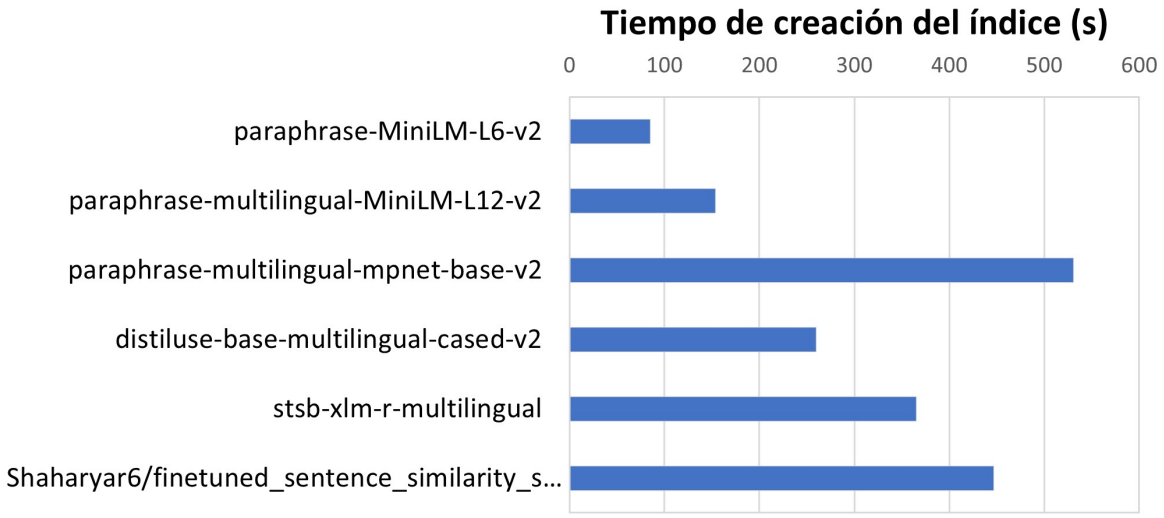

Figure 5: Tiempo de creación del índice de la base de documentos en función del modelo de *embeddings*, con *chunk size* y *overlap* de 100 y 20 *tokens* respectivamente.

Con estos resultados, el modelo de *embeddings* que vamos a utilizar es ***distiluse-base-multilingual-cased-v2***, el segundo que mejor OK RESPUESTA da y el tercero más rápido, además del más rápido de los modelos por encima del 70% de OK RESPUESTA.

III.III. *LLM* de generación de la respuesta

A continuación, se evaluará qué *LLM* se utilizará para generar la respuesta a partir del conocimiento encontrado.

Los modelos que vamos a evaluar son:

- *google/gemini-2.0-flash-thinking-exp:free*
- *google/gemini-2.5-pro-exp-03-25:free*
- *meta-llama/llama-3.3-70b-instruct:free*
- *deepseek/deepseek-chat:free (V3)*
- *deepseek/deepseek-r1-zero:free*
- *qwen/qwen-2.5-72b-instruct:free*

- *microsoft/phi-3-medium-128k-instruct:free*
- *mistralai/mistral-7b-instruct:free*
- *gpt-4o-mini*
- *meta-llama/llama-4-maverick:free*
- *meta-llama/llama-4-scout:free*

Los resultados han sido los siguientes (ver figura 6). Recordemos que la métrica OK RAG tiene un valor de 88,24%, que no cambiará con el *LLM* de elaboración de la respuesta, por lo que no incluiremos esta métrica aquí:

Modelo de respuesta	OK FUENTES (%)	OK RESPUESTA (%)
google/gemini-2.0-flash-thinking-exp:free	77,94	82,35
google/gemini-2.5-pro-exp-03-25:free	80,88	83,09
meta-llama/llama-3.3-70b-instruct:free	82,35	80,15
deepseek/deepseek-chat:free (V3)	79,41	82,35
deepseek/deepseek-r1-zero:free	75,00	83,09
qwen/qwen-2.5-72b-instruct:free	77,21	79,41
microsoft/phi-3-medium-128k-instruct:free	0,00	66,91
mistralai/mistral-7b-instruct:free	73,53	73,53
gpt-4o-mini	79,41	76,47
meta-llama/llama-4-maverick:free	**86,76**	**83,82**
meta-llama/llama-4-scout:free	83,09	77,94

A la vista de estos resultados se aprecia que el mejor modelo de elaboración de la respuesta es ***meta-llama/llama-4-maverick:free***, de reciente lanzamiento el 05/04/2025. Se puede apreciar que en la métrica OK RESPUESTA está cerca de otros, como los *gemini* o *deepseek*, pero parece que hay más diferencia al referenciar las fuentes.

Puesta en producción

Para poder analizar la puesta en producción de esta solución primero debemos estudiar su *ROI* (*Return of Investment*). Este cálculo se hace difícil, más que nada por el cálculo de los ingresos; el de los gastos lo abordaremos más adelante. Las fuentes de ingresos serían básicamente las visitas a la página del *RAG* y los ingresos que estas pudieran producir, por ejemplo con la inclusión de anuncios. Sin embargo, atendiendo al proyecto del que nace este, la *Coppermind*, vemos que se sostiene principalmente gracias al trabajo desinteresado y gratis de muchos de sus colaboradores. Por tanto, la viabilidad económica de este proyecto puede ser difícil de alcanzar. Aún así no está todo perdido, y habría que hacer un análisis más extenso de los ingresos que podrían obtenerse; además, con la automatización de los procesos los gastos pueden reducirse mucho. Otra alternativa podría ser ponerse en contacto con el propio autor del mundo en el que la *Coppermind* se basa, Brandon Sanderson. Este es un autor muy cercano al público y puede que estuviera dispuesto a invertir en él.

El valor de negocio es claro, y es que esto no deja de ser análogo a la batalla entre *Wikipedia* y *chatgpt*. Valoraciones personales aparte, es evidente quién la está ganando, ya que los usuarios prefieren chatear y encontrar respuestas concretas a sus preguntas con solo un par de *clicks* a bucear por decenas de páginas.

Los gastos del proyecto son varios. En primer lugar, el almacenamiento de la base de datos y el índice creado a partir de ella. Se podría almacenar en la nube en *Azure Storage*. Por otro lado, los gastos derivados de tener desplegado un código ejecutándose permanentemente, que podría hacerse también con *Azure Function*. A su vez, los gastos derivados de tener un dominio o una página web pueden no ser pocos. Por útlimo, los gastos que considero más importantes son lo de las llamadas a los modelos generativos, que podrían hacerse con *OpenAI* o *Azure AI Studio*. En este sentido, en soluciones cuyo tiempo de inferencia es crítico suele hacerse una reserva de capacidad (*PTUs*), en que se te reservan una serie de *GPUs* para que tus inferencias

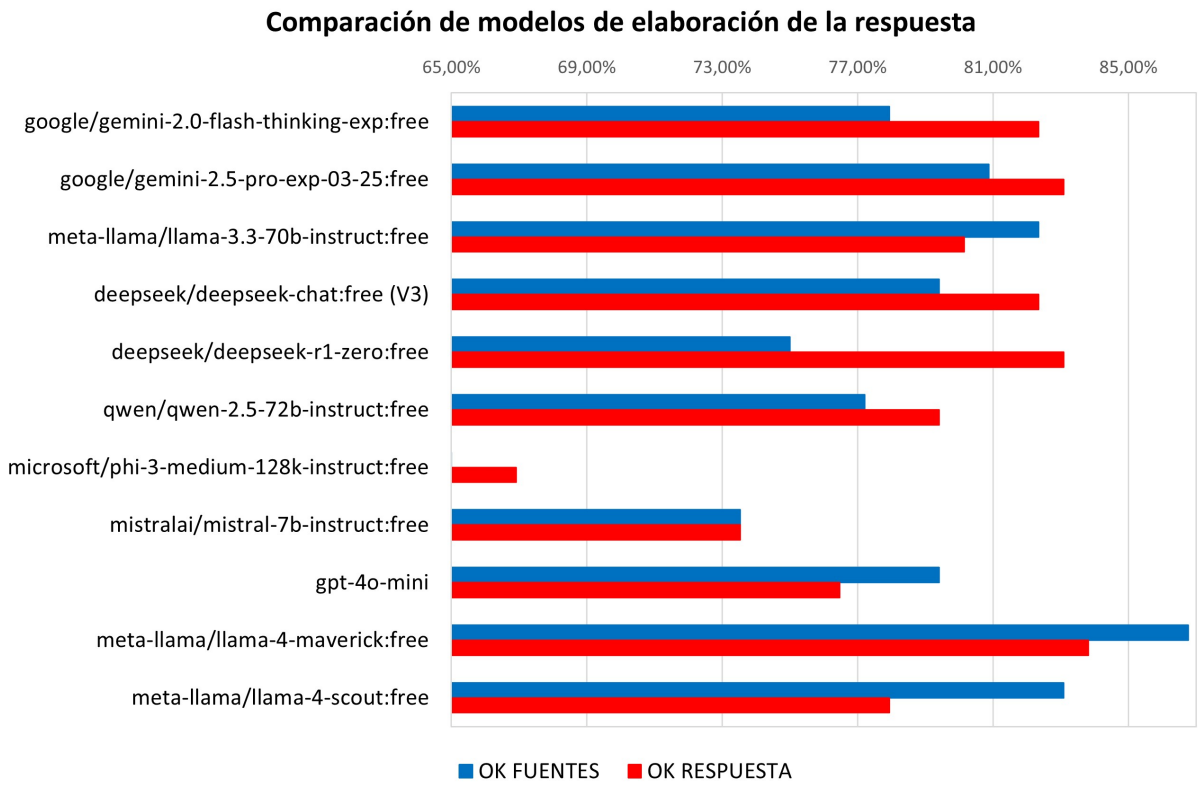

Figure 6: Valor de las métricas OK FUENTES y RESPUESTA en función del modelo de elaboración de la respuesta (temperatura y *repetition penalty* a 0), con *distiluse-base-multilingual-cased-v2* como modelo de *embeddings*, *chunk size* de 100 *tokens*, *chunk overlap* de 20 *tokens* y *top n* de 10 *chunks*.

no compitan con la del resto de usuarios. Este es un gasto que puede ser muy alto y que considero que se podría ahorrar. Y no solo habría que pagar las llamadas a los modelos geenrativos, sino también las llamadas a un modelo de *embeddings* grande, como el de *OpenAI*, un gasto fundamental, ya que reducirá los tiempos y mejorará los resultados respecto a los modelos que hemos probado en este trabajo. Otros gastos a tener en cuenta son los de mantenimiento, y es que la *Coppermind* es una página web viva, a la que se añaden o modifican entradas cada semana, por lo que habría que diseñar un automatismo que detectara cuando se hacen cambios y volviera a hacer el índice de la base de datos.

El criterio de aceptación podría ser ciertos valores de las métricas del Test Automático de este trabajo. Habría que diseñar un Test Automático que abarcara toda la página, ya que el que se ha diseñado en este trabajo recoge solo las entradas seleccionadas al principio. Este test podría realizarse con la ayuda del *fandom* o con la de las personas que publican entradas en la página web; estas últimas podrían utilizarse a modo de usuarios alfa. Otras métricas interesantes para el *Feedback Loop* podrían ser el *feedback* de usuarios, *Faithfulness* de *Ragas* o *Gorundedness* de *Microsoft*, este último con un coste asociado.

La estrategia que usaría para la puesta en producción es el *Canary deployment*, donde antes de ponerla en producción se le daría acceso a la solución a las personas que publican en la propia página. Además de que estas son figuras de autoridad en el tema y serían capaces de discernir si las respuestas son buenas o no, el *RAG* podría serles de utilidad en su labor de publicación ayudándoles a resolver preguntas puntuales de forma rápida. Una vez puesta en producción esta solución, cada nueva versión o mejora podría volver a aplicarse a este conjunto de usuarios antes de pasar a público general.

Conclusiones

Los objetivos principales de este trabajo se han conseguido, además de haber ido un poco más allá, con la creación del *RAG* general. Durante la elaboración de este trabajo he adquirido muchos conocimientos sobre la creación de sistemas *RAG* locales y completamente gratis, así como sus limitaciones. Con esto he aprendido a descargar modelos de *huggingface* y utilizar distintos proveedores para hacer llamadas via *API* a *LLMs*. Y, no sin dificultad, he aprendido a sortear los problemas de certificados con el ordenador del trabajo al conectarse a estas páginas. He aprendido a crear frontales simples con la librería de *Python streamlit*, además de descubrir que no me apasiona. He profundizado en cómo cambiar el formato de documentos para el mejor funcionamiento del *RAG*, para lo que he aprendido sobre *pandoc* y *markitdown*, librerías de código abierto que permiten la conversión de pdfs, words y htmls entre otros a markdown. He hecho mucha investigación acerca de métricas con las que poder medir el rendimiento de los sistemas *RAG*, así como desarrollado un test automático para evaluarlos.

En resumen, he adquirido muchos conocimientos en sistemas *RAG*. He creado una herramienta que bien podría servir para valorar si tiene sentido implementar un *RAG* productivo en una base de datos, o de uso personal si se quiere hacer preguntas sobre una serie de documentos relativamente grandes. Por último, he podido plantearme cómo llevaría a producción el *RAG* de la *Coppermind*. Ha sido un trabajo muy interesante y que seguirá evolucionando con la implementación de las posibles mejoras o trabajos a futuro que detallo a continuación.

Posibles mejoras

En esta sección voy a detallar las posibles mejoras que hacer a este trabajo. Son ideas que han salido durante la realización del mismo o gracias a las clases recibidas, que no se alejan demasiado de los objetivos del trabajo, pero que o bien son demasiado ambiciosas o bien no ha dado tiempo a hacerlas.

1. *LLms* de pago

Si ha habido algo que haya lastrado este trabajo es la limitación que teníamos de utilizar llamdas gratis via *API* a *LLMs* ofrecidos por distintos proveedores. Esto ha hecho que estemos restringidos en cuanto a los modelos que utilizar, tengamos que crear varias cuentas por proveedor para poder sortear esos *rate limit*, no podamos automatizar del todo los test automáticos, etc. Además, los mejores *LLMs* no se ofrecen gratis, por

lo que contratar alguno puede aumentar también el desempeño del *RAG*, además de probablemente reducir los tiempos de inferencia.

Usar *LLMs* de pago me permitiría además aplicar *Structured Outputs* para que en la elaboración de la respuesta y el *LLM as a judge* den, respectivamente, las referencias separadas de la respuesta y la valoración del razonamiento.

Por otro lado, utilizar el modelo de *embeddings* de *OpenAI* mejoraría significativamente los resultados de la parte del *restrieval*, ya que es un modelo grande pero que no necesita ser alojado en local, por lo que además es rápido. En clases y prácticas anteriores hemos discutido y comprobado la mejora notable por utilizar este modelo.

2. *Prompts*

En cuanto a los *prompts*, el prompt que utilizo en este trabajo, tanto para la elaboración de la respuesta como para la parte del *LLM as a judge* del test automático, son los que han funcionado para el modelo *Google: Gemini Pro 2.0 Experimental (free)*. Que funcionaran bien con ese modelo no implica que funcionen también con el resto, por lo que una posible mejora podría ser encontrar el *prompt* ideal para cada *LLM* utilizado. Además, se podría haber aplicado la técnica *Few-Shot Prompting* para incluir algún ejemplo de cómo elaborar la respuesta y referenciar los documentos adecuados.

Otra cosa que me gustaría haber hecho mejor es la gestión de los prompts. En el repositorio del código están incluidas en un *script* de *python*, pero deben poder guardarse y tratar las versiones con alguna herramienta externa que sea más idónea.

3. Test Automático

Una mejora clara en esta parte es incluir el tiempo de inferencia medio de cada test. Esta es una métrica funcamental para soluciones tipo *RAG*. Esta es además una métrica que, de poner esta solución en producción, mejoraría mucho, ya que los *LLMs* y modelos de *embeddings* de pago son mucho más rápidos.

Por otro lado, el test puede no ser todo lo representativo que pretende, ya que el número de preguntas de cada documento no lo he decidido de forma rigurosa. Podría hacerse que el porcentaje de preguntas sobre cada documento dependa de la longitud de cada documento de la base de datos. Además, por supuesto, las preguntas pueden estar hechas de forma sesgada, ya que he sido yo mismo el que las ha diseñado. Una manera de hacerlo menos sesgado es quizás pasarle fragmentos del documento a un *LLM* y pedirle que elaborara una pregunta que fuera respondida con algo de ese fragmento.

A su vez, los resultados de cada test se han enviado a un *Excel*, cosa que no es ni muy limpia ni muy escalable. Lo que podría hacerse es enviar los resultados y parámetros de cada test a *MLflow*.

4. Métricas de *Ragas* y *Groundedness* de *Microsoft*

Una mejora que sería muy buena es utilizar *Ragas*, una herramienta de código abierto diseñada para evaluar sistemas *RAG*. En la carpeta de Apoyo dejo un notebook donde hago un análisis de las distintas métricas que ofrece, además de poder usarse de soporte para elaborar métricas propias.

Además de estas métricas, podría calcularse la métrica *Groundedness* de *Microsoft*. Esta métrica mide lo desviada que está la respuesta de un sistema *RAG* respecto del contexto que se le pasa. Es al fin y al cabo una forma de medir las alucinaciones, o lo que añade el *LLM* que elabora la respuesta al contexto recibido. Esta es una métrica que se calcula via *API* y en la que no se utiliza un *LLM* para calcularla, por lo que es muy rápida (unos 300 ms). Es por esto que podría incluso utilizarse para avisar al usuario de lo fiable que puede ser la respuesta, pintándola por ejemplo en una escala de color del rojo al verde.

5. Llamadas a *LLMs* con *LiteLLM*

En el código de este trabajo hago las llamadas a los *LLMs* de los distintos proveedores de forma algo sucia; cada uno necesita una estructura diferente. *LiteLLM* es una librería de código abierto que actúa como una

interfaz unificada para hacer estas llamadas, de forma que lo hace mucho más escalable (es más fácil añadir otros proveedores) y limpia. En la carpeta de Apoyo dejo un pequeño tutorial de cómo se haría.

Esto finalmente me ha dado tiempo a incluirlo.

6. Prevención del *prompt injection* y mal uso de la herramienta.

Tal y como está diseñada esta solución es muy vulnerable al prompt injection y al mal uso de la misma. Para evitarlo, podría añadirse una pieza previa que detectara si la *query* del usuaria es adecuada o no. Otra opción es modificar el *system prompt* de la pieza de generación de la respuesta para que esa misma valore si cada pregunta es adecuada o no.

Líneas a futuro

En esta sección voy a detallar las líneas a futuro que surgen de este trabajo. Son ideas que han salido durante la realización del mismo o gracias a las clases recibidas, como las de la sección de posibles mejoras, pero estas suponen cambios grandes en el funcionamiento de la solución actual y podrían formar parte de un nuevo trabajo por sí mismas.

1. Implementación de un *chatbot*

Actualmente este *RAG* funciona sin poder hacer varias iteraciones sobre las mismas preguntas; únicamente recibe una *query* de entrada a la que da respuesta. En un primer momento se pretendía que se pudiera *chatear*, pero esto supone un gran cambio en la lógica, como el manejo del historial de la conversación, una búsqueda más dinámica en el *RAG*, etc. Una de las piezas nuevas que habría que pensar en incluir es un detector de *chit-chat* que, si uno de los mensajes del usuario no tiene intención de realizar una búsqueda en el *RAG*, sepa manejarlo y no haga la búsqueda.

2. Implementación de un agente

La solución actual es muy rígida por construcción: para cada *query* obtiene un número fijo de *chunks*, con los que tiene que elaborar la respuesta. Si ahí no está la respuesta no puede hacerse nada. Todos los hiperparámetros de esta solución son fijos, aunque puedan no ser los más idóneos para alguna pregunta. Un ejemplo de pregunta para la que esta solución no es idónea es si se le pidiera hacer una lista de características de varios elementos: lo ideal sería que buscara cada elemento por separado, pero tal y como está planteada la solución actualmente hace una única búsqueda, pudiendo no encontrar toda la información.

En este sentido, sería muy interesante elaborar una solución en la que sea un agente el que gestione las búsquedas en la base de datos y la elaboración de la respuesta. De esta forma, el agente puede adaptar la *query* del usuario y hacer búsquedas con *querys* más adecuadas, no hacer solamente una, decidir si tiene que hacer más porque aún no tiene la respuesta o modificar hiperparámetros como *top_n*.

3. Mejoras en las *querys* del *RAG*

Una técnica muy habitual en *RAGs* productivos es *RAG-fusión*. *RAG-fusión* consiste en, con la ayuda de un *LLM*, generar más *querys* a partir de la inicial, hacer varias búsquedas para cada *query*, reordenar los *chunks* encontrados y generar la respuesta con todos ellos. Esta solución aporta mayor cobertura, precisión y respuestas más robustas a las soluciones de *RAG*.

4. Mejoras en la creación del índice

Una de las limitaciones de este *RAG* es que en la creación del índice se hacen *embeddings* directamente sobre el contenido de la base de datos, pero este contenido no tiene por qué ser muy similar a las preguntas que hacen los usuarios. Otra opción es hacer los *embeddings* sobre las preguntas que podría responder cada *chunk* de la base de datos, generando estas preguntas con la ayuda de un *LLM*.

Otra de las posibles mejoras en la cración del índice de este tipo de sooluciones es incluir un pequeño resumen de cada documento como metadato de los *chunks* que lo conforman. De esta forma, el *LLM* que elabora la respuesta tiene más contexto y puede responder mejor. Estos resúmenes pueden crearse con otro *LLM*.