UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Estudio, análisis y experimentación del uso de técnicas Retrieval-Augmented Generation para grandes modelos del lenguaje

Trabajo Fin de Grado

Grado en Ciencia de Datos

AUTOR/A: Ferrando Muñoz, Guillermo

Tutor/a: Sánchez Anguix, Víctor

Cotutor/a: Alberola Oltra, Juan Miguel

Cotutor/a externo: Hussein Galindo, Raul

CURSO ACADÉMICO: 2023/2024

Resumen

El presente trabajo aborda el estudio y la aplicación de técnicas específicas en IA Generativa, con el objetivo de mejorar el rendimiento de un RAG (*Retrieval-Augmented Generation*). El trabajo se centra especialmente en las técnicas de recuperación y generación, que son estrategias fundamentales para guiar y mejorar la generación de texto por parte del modelo RAG. Además, se desarrollará un experimento enfocado en evaluar el rendimiento del modelo en la generación de documentos. Este experimento incluye la utilización del gran modelo de lenguaje Llama 3-8B y modelos de vectorización del idioma español, junto a las técnicas recuperación y generación mencionadas, con el fin de identificar las configuraciones óptimas para mejorar la calidad y la coherencia del texto generado.

Palabras clave: RAG, *retrieval*, *generation*, LLM, *chunks*, *embeddings*.

Abstract

This paper deals with the study and application of specific techniques in Generative AI, with the aim of improving the performance of a RAG (Retrieval-Augmented Generation). The work focuses especially on retrieval and generation techniques, which are fundamental strategies to guide and improve text generation by the RAG model. In addition, an experiment focused on evaluating the performance of the model in document generation will be developed. This experiment includes the use of the Llama 3-8B large language model and Spanish language vectorization models, together with the aforementioned retrieval and generation techniques, in order to identify the optimal configurations to improve the quality and consistency of the generated text.

Keywords: *RAG, retrieval, generation, LLM, chunks, embeddings.*

Tabla de contenidos

1. Introducción

El campo de la ciencia de datos o más popularmente conocido como *Big Data* ha sido un gran foco de atención durante los últimos años. Por su naturaleza del trato la información, la ciencia de datos es aplicable a todo sector de negocio que pueda ser medido. Esto significa que puede aplicarse al deporte, a la industria, economía, medicina, cinematografía y más.

El procedimiento en un proyecto de ciencia de datos siempre ha estado delimitado por una serie de fases definidas. En primer lugar, el científico/a de datos, en contacto con la entidad o empresa interesada en desarrollar el proyecto, establecerá los objetivos a desarrollar que mejor se adecúen a los intereses de la empresa. Una vez definido el problema a resolver y todos sus procesos, se entrará en fase de recolección de datos o información. Junto a la anterior, puede esperarse que esta segunda fase abarque semanas o meses, ya que un mayor volumen de información llevará a un proyecto más pulido y confiable, además de que suele ser una tarea tediosa que suele toparse con imprevistos durante su desarrollo. Una vez los datos han sido recolectados, el científico/a los revisará y tratará para que puedan ser interpretados por lenguajes de programación. En otras palabras, los datos tendrán que tener una forma específica e interpretable por un ordenador para poder realizar operaciones matemáticas con los mismos. Dicho aspecto es relevante respecto a la resolución de este proyecto. Finalmente, tras esa limpieza de la información, se procederá a la resolución del problema definido al comienzo del proyecto [1, 2].

Como hemos mencionado, hasta hace poco un proyecto de ciencia de datos tendría aseguarada, para un momento u otro, una fase de limpieza de información para su correcta interpretación por parte de un computador. Sin embargo, los avances en la ciencia de datos han llevado al desarrollo de herramientas que simulan el aprendizaje humano. Dichas herramientas han sido nombradas como “Inteligencia Artificial Generativa” o, para abreviar, “IA” o “IA generativa” [3], han alcanzado una vez más una sorprendente popularidad en un corto plazo. La IA es capaz de procesar información denominada como cruda. Sin ir más lejos, la información cruda hace referencia a cómo los humanos procesamos la información: con texto en lenguaje natural, es decir, el lenguaje mediante el cual los humanos nos comunicamos, imágenes o audio.

Respecto a la IA generativa en el ámbito empresarial, es evidente que tiene un enorme potencial. En este proyecto, trataremos de profundizar teórica y prácticamente en las utilidades de la IA generativa, en concreto en las herramientas disponibles para la comunidad dentro del campo del procesamiento de lenguaje natural, así como las prácticas comunes para explotarlas, con el objetivo de beneficiar a la empresa en la que se desarrolla este proyecto. Esas herramientas de procesamiento de lenguaje natural son principalmente los llamados *Large Language Models* (LLMs) o Grandes Modelos de Lenguaje, que a su vez son empleados en una práctica llamada *Retrieval-Augmented Generation* (RAG) o Generación con Recuperación Aumentada. Para ambas herramientas entraremos en detalle en la siguiente sección.

1.1 Motivación

Este trabajo de fin de grado forma parte de un proyecto mayor de línea de investigación sobre la inteligencia artificial generativa dentro del Instituto Tecnológico de Informática (ITI). Esta línea de investigación tiene el fin de estar al día con los avances en el campo de la inteligencia artificial y ser así competentes en el mismo, tanto en el aspecto teórico como la implementación práctica. Una vez completado el proyecto, el ITI pretende usar los resultados para ofrecer servicios de I+D a empresas de la Comunidad Valenciana, ya que el ITI se creó para formar a dichas las empresas para crecer en el ámbito tecnológico.

Además, el uso de las herramientas de procesamiento de lenguaje natural mencionadas, en concreto la antes mencionada RAG, busca la agilización de comunicación interna entre los diferentes departamentos de la propia empresa. Lo desarrollado en este trabajo tratará refinar la calidad de los resultados obtenidos hasta el momento.

Contando con las preferencias personales del desarrollador del trabajo de fin de grado por la inteligencia artificial, el equipo de la empresa propuso la idea.

1.2 Objetivos

El objetivo general de este trabajo consiste en investigar sobre el estado del arte de la práctica *Retrieval-Augmented Generation* para su posterior implementación práctica con recursos de código abierto, enfocándose en el idioma español. Todo ello en el lenguaje de programación Python.

Asimismo, podemos desglosar el objetivo global en una retahíla de objetivos específicos:

1. Lograr un entendimiento y asentamiento de los principales conceptos sobre los *Large Language Models* y el *Retrieval-Augmented Generation*.
2. Investigar sobre las distintas prácticas que se pueden llevar a cabo para mejorar el rendimiento del *Retrieval-Augmented Generation*.
3. Implementar un *Retrieval-Augmented Generation* en entorno local, con el fin de mantener la privacidad de los clientes de la empresa. En otras palabras, todos los recursos utilizados deberán ser de código abierto, sin depender de suscripciones a servicios externos.
4. Adaptar la implementación a las metodologías y tipos de datos que se usan dentro de la empresa, además de ser reutilizable a largo plazo.
5. Encontrar la mejor combinación de técnicas estudiadas en el objetivo 2. En concreto, enfocándose para el idioma español.

1.3 Estructura

A continuación, proporcionaremos una concisa explicación de las distintas secciones que forman la memoria de este trabajo. Estas secciones son:

1. **Introducción**

Esta sección introductoria da un breve contexto de la utilidad del proyecto "*Retrieval-Augmented Generation* (RAG) para grandes modelos de lenguaje (LLM)". Por otro lado, se exponen las motivaciones, tanto de la empresa como para el

realizador, para la realización del proyecto, los objetivos a alcanzar en el mismo y una breve explicación de la estructura de la memoria.

2. Estado del arte

La segunda sección de la memoria tratará los conceptos técnicos necesarios para un correcto entendimiento del trasfondo que guardan, por un lado, los *Large Language Models* y, por otro, el *Retrieval-Augmented Generation*. Respecto al segundo, se profundizará en las distintas técnicas que han surgido y siguen surgiendo para mejorar su rendimiento. Finalmente, se comparará el trabajo realizado con otros anteriores, resaltando las fallas de los mismos y, posteriormente, la mejora que propone este respecto a ellos.

3. Análisis del problema

Este capítulo narra sobre el hecho de forzar una implementación de un *Retrieval-Augmented Generation* con recursos de código abierto, que deriva en un gran número de pequeñas trabas que a su vez conllevan a un gran esfuerzo para llevar a cabo la implementación. Por otra parte, se presenta el marco legal y ético.

4. Diseño de la solución

Aquí se presenta el esquema conceptual con el que se ha llevado a cabo la implementación del RAG. Se explican los tipos de documentos de información cruda usados y cómo fueron tratados, cómo se estableció el entorno de ejecución del RAG ideado para adaptarse a los recursos de computación disponibles en la empresa, a la necesidad del uso de *Large Language Models* de código abierto y para ser reutilizable en el futuro a largo plazo.

5. Experimentación

Esta sección estará enfocada en mostrar las distintas formas implementadas sobre el tratamiento de la información para un conjunto de documentos públicos. Después, se mostrarán los resultados de la mejor combinación de técnicas probadas y se razonará cual sería la que mejor se adaptaría a la extracción de los documentos privados del Instituto Tecnológico de Informática.

6. Conclusiones

La sección de conclusiones servirá para establecer la mejor metodología probada y para razonar cual sería la que mejor se adaptaría a la extracción de los documentos privados del Instituto Tecnológico de Informática.

7. Trabajos futuros

En este capítulo final explicaremos qué procedimientos deberían llevarse a cabo para mejorar el rendimiento del RAG de código abierto implementado y que el realizador del proyecto no pudo poner en práctica.

2. Estado del arte

Esta segunda sección de la memoria del proyecto contiene la base teórica necesaria para comprender el funcionamiento interno de los LLMs y el cómo son un pilar clave en el surgimiento del proceso RAG. Por otra parte, haremos un estudio sobre propuestas que han ido surgiendo para esbozar los primeros paradigmas de los RAG según su complejidad, además de las distintas técnicas específicas que marcan la diferencia entre estos paradigmas.

Cabe recalcar la muy reciente creación del proceso RAG. Esto supone que todavía se encuentre en espera de un mayor número de investigaciones formales o publicaciones por parte de editoriales reconocidas.

2.1 *Large Language Models*

En 2017, la empresa Google lanzó productos que suponían una gran mejora en cuanto a la capacidad de entendimiento de las preguntas de los usuarios y de su buscador web. No obstante, fue otra empresa emergente llamada OpenAI [1] que decidió llevar los llamados *Large Language Models* al siguiente nivel. Fue por dicha empresa que los *Large Language Models* obtuvieron fama mundial, además de aplicarse en diversos campos de negocio, a finales de 2022. Desde entonces, el campo del procesamiento de lenguaje natural ha aumentado y sigue aumentando drásticamente.

Cuando hablamos de un gran modelo de lenguaje o LLM, nos referimos a un modelo, es decir, una función que intenta replicar el comportamiento de un sistema [4]. Un ejemplo que cualquiera puede tener en mente es el modelo climático. El modelo climático, técnicamente hablando, sería un programa, software, o tipo de sistema que permite predecir el clima de la semana que viene basándose en datos históricos. De forma que, por ejemplo, con el clima de las últimas semanas, el viento, las nubes y la humedad, el modelo predice el clima de la próxima semana. Todo esto no ocurre en la realidad, sino que hay un software que tiene una serie de instrucciones que logran simular el resultado. Además, un LLM es un modelo de lenguaje generativo [5, 6]. Los modelos de lenguaje generativos son capaces de conocer e identificar partes de nuestro idioma y, dado un texto, generar palabras. Por tanto, al igual que el modelo climático predice el clima, un LLM predecirá palabras. De forma simplificada, un gran modelo de lenguaje predecirá cuál es la palabra más probable que va a haber a continuación en un texto, haciéndolo tan bien que parece que entienda perfectamente de lo que se está hablando y pueda responder casi a cualquier cosa de forma coherente.

Unos de los pensamientos que suelen surgir al interactuar con un LLM por primera vez es que las respuestas hayan sido copiadas de alguna fuente como una página web, pero no, se tratan de textos únicos. Entonces, ¿en qué se basan los LLMs para predecir esas palabras?

Como sabemos, los humanos hablamos con lo denominado como lenguaje natural. El lenguaje natural nos permite rectificar nuestras palabras, algo que de lo

[1] https://www.forbes.com/sites/bernardmarr/2023/05/19/a-short-history-of-chatgpt-how-we-got-to-where-we-are-today/

normal no será un problema para que se nos entienda, no obstante, un ordenador es incapaz entender el lenguaje humano. Normalmente, cuando los humanos quieren comunicarse con un ordenador, usarán la programación, pero hay algunas cosas que son difíciles de programar. Por ejemplo, reconocer un número escrito a mano o el dibujo de un ave. Para todo aquello que fuese muy difícil de programar se inventó el Aprendizaje Automático o *Machine Learning*. Los *Large Language Models* se basan en un modelo de *Machine Learning* denominado como red neuronal [7], inspirado en cómo funcionan las redes neuronales del cerebro humano. En realidad, los modelos de redes neuronales entran dentro de una categoría "mayor" del Aprendizaje Automático llamada *Deep Learning* o Aprendizaje Profundo, debido a su gran complejidad y tamaño y, a medida que incremente esa complejidad y tamaño, la red neuronal podrá denominarse como Inteligencia Artificial Generativa [3].

Dada una estrategia, la red neuronal recibirá una ingente cantidad de datos y casos de los que irá sacando conclusiones, encontrando patrones, detalles que los humanos no seamos capaces de percibir. Este proceso en el que la Inteligencia Artificial estudia miles y miles de casos se llama entrenamiento [2].

2.1.1 Entrenamiento de un *Large Language Model*

Como hemos mencionado, para que un ordenador pueda entender el lenguaje natural, se necesita algo más que la programación. Es más, los ordenadores solo entienden números: ceros y unos. Por ello, para un ordenador, el texto será un conjunto de números con los que hacer operaciones matemáticas. Por tanto, el texto tendrá forma de grupos de números que las redes neuronales recibirán y en los que reconocerán tantos patrones como puedan. Dichos patrones de números representando texto son un punto clave, y les damos el nombre de *token* [4]. Un *token* puede ser una sola palabra o varios *tokens* pueden representar una sola palabra si es, por ejemplo, compuesta. Este proceso de extracción de *tokens* o "tokenización" permitirá al ordenador conocer las palabras, pero, como decíamos antes, nunca sabrá qué significan. Lo que sí puede hacer si presta mucha atención es saber qué términos están relacionados con cuáles. O sea, poner un marcador numérico cuando detectamos que dos *tokens* son similares, de forma que este marcador indique si unos *tokens* están relacionados entre sí dado un contexto. De este modo, cada *token* tendrá un gran número de marcadores que le dirán cuáles son las cosas que tienen común con otras palabras. Una forma de representar con una imagen esta idea es la que vemos en la Figura 1, donde, por ejemplo, para la palabra "soundtrack" y con la distancia euclídea elegida como medida de similitud, las palabras más cercanas serían "album" y "starring", entre otras.

[2] https://www.youtube.com/watch?v=Sz4qacFBHLk

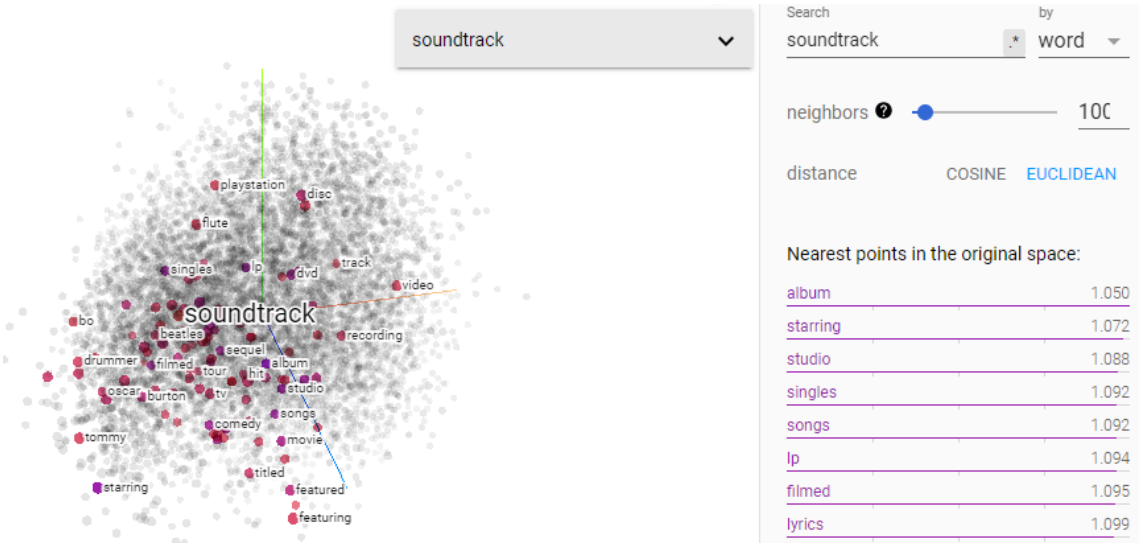

Figura 1: representación tridimensional de marcadores de tokens. [3]

En realidad es bastante más complejo, ya que en la Figura 1 solo hay tres dimensiones, pero un *token* de lo normal tendrá muchos más marcadores y por tanto muchas más dimensiones en la nube.

Pensando en detalle, con este proceso de marcadores se está definiendo un sistema de codificación, sistema de codificación que se conoce como *embedding* [9, 10]. Como acabamos de ver, el sistema de *embedding* permite agrupar palabras similares e incluso sus tendencias en cuanto a significado, lo que nos sirve para muchas cosas. Pero, un gran modelo de lenguaje también recuerda conjuntos de *tokens*, o sea, frases. Si bien la idea original de envolvimiento era buscar qué relación tienen las palabras entre ellas y crear un sistema complejo de etiquetado para saber por dónde se orienta cada una, cuáles están relacionadas y qué tanto están relacionadas, un LLM lo lleva a otro nivel usando secuencias de varios *tokens*, es decir, frases enteras, y hace lo mismo: obtiene frases y las organiza con sus respectivas relaciones dentro de un espacio de hasta más de mil marcadores para cada frase.

Una vez creado el sistema de *embedding*, se añade un proceso de limpieza de textos que consiste en eliminar aquellas palabras con poco valor a las frases, tales como determinantes, artículos, adverbios y signos de puntuación que aporten poco. Tras dicho proceso, todas las frases que dice un LLM se sacan de su respectivo sistema de *embeddings*. Esto es clave para entender por qué los LLMs generan textos únicos: al hacer consultas a sus *embeddings*, sacan estas secuencias de *tokens* que son frases comprimidas, que después tienen que convertir en algo aceptable por parte de los humanos. Lo que hacen es convertir estos *tokens* en las palabras que están más cercanas entre ellas en los *embeddings*, por lo que a veces las palabras son ligeramente distintas a las que estaban en la frase original y, sobre todo, usa nuestras reglas de sintaxis y gramática para volver a poner esas palabras que había quitado al principio, convirtiendo esa secuencia de números en una frase que sea adecuada a la lectura de un ser humano. Todo esto añadido a la utilización de la técnica de *sampling*, que consiste en que el sistema genere un número aleatorio y, en base a ese número aleatorio, se

[3] https://projector.tensorflow.org/

mueve ligeramente dentro del espacio del *embedding*, para un lado y para otro, hacia una frase y hacia otra. Y como se puede intuir, esa frase estará lo suficientemente cerca como para que sea parte de la misma conversación y sea coherente con lo que estamos hablando. Así es como los *Large Language Models* inventan o crean historias que parecen coherentes, pero que en realidad son combinaciones de otras historias que han leído antes en su fase de entrenamiento.

2.1.2 Tamaño de un *Large Languaje Model*

Como hemos dicho, hay una fase en la que la red neuronal realiza operaciones matemáticas para poder entender el lenguaje natural, estas operaciones tienen el fin de obtener unas variables o parámetros que se usarán para tratar los *tokens* de las palabras, frases y oraciones que se le den. Un aspecto clave que determinará la capacidad y el rendimiento de un LLM es el número de esos parámetros del que disponga, en general, cuantos más sean, mejor [11]. En la Figura 2 podemos observar una gráfica comparativa sobre el ratio del número de parámetros entre el número de *tokens* a procesar por esos parámetros.

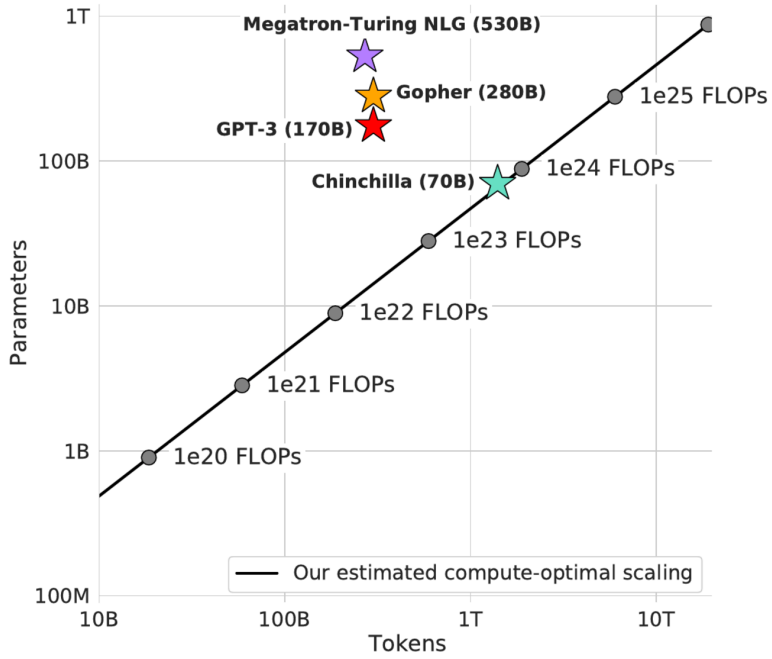

Figura 2: comparativa gráfica de LLMs según sus parámetros y sus tokens. [4]

Como se puede observar en la Figura 2, el número de parámetros de los LLMs se ubica, a día de hoy, entre los miles de millones (billones en inglés). Por ejemplo, el LLM GPT-3 cuenta con 170 miles de millones de parámetros. Como se muestra en la imagen, la línea gris marca del punto óptimo de la relación parámetros/*tokens* en cuanto a coste computacional durante la fase de entrenamiento, por lo que, de nuevo el ejemplo de GPT-3, dicho modelo podría haber soportado más *tokens* sin suponer demasiada computación. En caso de que uno de los modelos se encontrase al lado derecho de la línea gris, significaría que el coste computacional habría sido muy elevado.

[4] https://gpt3demo.com/apps/chinchilla-deepmind

Cabe mencionar que los grandes modelos de lenguaje de código abierto suelen contar con un número pequeño de parámetros en comparación a los mostrados en la Figura 2, que son de pago. Esto se hace así ya que los LLMs de código abierto suelen enfocarse a un público que no cuente con ordenadores muy potentes, por lo que se podrá esperar que el rendimiento de estos modelos de lenguaje sea peor que los de pago. Sin embargo, con el tiempo han ido publicándose LLMs de código abierto con números de parámetros más cercanos a los modelos de pago.

2.1.3 *Transformers*

Llegados a este punto, ya entendemos la gran mayoría del funcionamiento de las IAs, pero nos estamos dejando una parte muy importante: el gran problema que ha existido durante mucho tiempo y para el que hasta hace muy poco no había solución, y que afecta tanto a la fase de entrenamiento de una red neuronal como a la fase de inferencia, es decir, la fase de cuando hablamos con una inteligencia artificial.

Durante la fase de entrenamiento, cuando damos letras y luego *tokens* a la red neuronal, la red neuronal tiene una entrada de un determinado tamaño. Esa entrada la recoge, la procesa y saca conclusiones. El problema llega al estar dando por hecho que el ordenador puede lidiar con cualquier tipo de frase de cualquier tamaño. No obstante, esto no es así, y es un problema mucho más grande de lo que parece. La red neuronal no recuerda lo que pasa entre paso y paso cada vez, llevando a que simplemente se le olvide. En la fase de inferencia, cuando hablamos con la inteligencia artificial y esperamos que genere texto, es importante que entienda toda la frase que le damos, independientemente de cuántas cosas hayamos mencionado. Además, debe recordar todo lo que vamos hablando a lo largo de la conversación, que tenga algún tipo de memoria, algo que le permita recordar.

Hasta hace poco, las opciones que se tenían para tratar este problema no eran eficientes a la hora de entrenarlas, no podían desarrollar tareas muy largas, hacer resúmenes o cualquier tipo de tarea que requiriera analizar un texto muy grande y entender cosas complejas. Este era el caso de las redes neuronales recurrentes [12] y más adelante de las redes *Long-Short Term Memory* [13].

Fue en 2017 cuando llegó la solución a este problema, cuando se publicó "*Attention is all you need*" [14], un artículo de la mano del equipo de la empresa Google donde se plantea una solución que a día de hoy persiste como el estado del arte de para dar forma a la memoria de las inteligencias artificiales. En otras palabras, una nueva forma de organizar las redes neuronales llamada *Transformers*.

En los *Transformers* [5, 6], lo que se hace es añadir una nueva capa a la red neuronal, una capa dedicada a lo que los investigadores llamaron atención, que hace referencia a cómo los humanos vemos mucha información, pero solo nos centramos en ciertos objetos, ignorando el resto. La idea es que nuestra red neuronal haga exactamente lo mismo, que analice una gran cantidad de texto de golpe y lo use para extraer el contexto de lo que se está hablando, prestando atención solo a los detalles

[5] https://www.youtube.com/watch?v=aL-EmKuB078&t=331s

[6] https://www.youtube.com/watch?v=xi94v_jl26U

más importantes. Para hacer esto, se escoge una parte del texto de entrada, se normaliza, se tokeniza y, de esos *tokens*, se sacan para cada uno los vectores de *embedding*, para después mirar cuánto se parecen a el resto de vectores de *embedding* o palabras que hay dentro de la frase. Con esa comparación, se obtendrán los términos o palabras más relevantes de la frase, aquellas que aporten la mayor información semántica. Se trata de que la información sobre dichos términos relevantes no solo se vaya enriqueciendo, sino que también que se vaya manteniendo a lo largo de todo el proceso, todo eso recogido mediante un "vector de atención". El vector de atención ayudará mucho al ordenador, ya sea para saber el contexto de cada una de las frases, como para saber el significado de palabras polisémicas.

En la Figura 3 se puede observar la arquitectura de un transformer resumida en un diagrama. En la parte baja izquierda de ese diagrama podemos ver los *inputs*, que son las palabras o frases que queremos que el *Transformer* entienda. Por ello, en las siguientes fases de la parte izquierda, lo que se hace es generar los vectores de atención que hemos mencionado, para ello pasando por la capa de atención o, como se muestra en la Figura 3, *Multi-Head Attention*. Luego, en el lado de la derecha, tenemos los *outputs*, que son las palabras o frases que el *Transformer* genera como respuesta. El proceso es similar a al llevado a cabo en el lado izquierdo, pero, en este caso, la capa de atención contará con una "máscara" que impida ver las palabras futuras, ayudando a generar el texto en el orden correcto. Después, tras fases de refinado de los vectores de atención, se aplicarán las funciones matemáticas *Linear* y *Softmax* para convertir los vectores en palabras de nuevo y recoger las más probables para formar la frase final a responder.

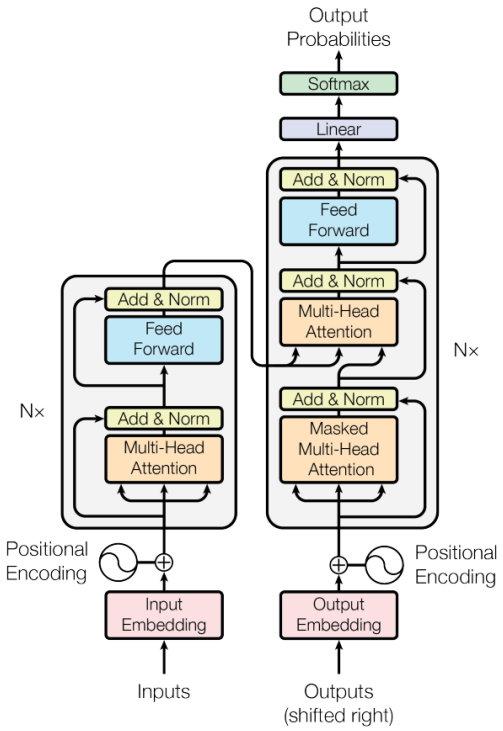

Figura 3: arquitectura de un transformer. [14]

Por tanto, ahora podemos entender que la atención es la que permite a los LLMs extraer los conceptos clave de un texto escrito por nosotros, saber qué términos tienen

más peso, qué es lo más importante de nuestra pregunta, incluso hacer un resumen de un texto o responder tema por tema a todo lo que le hemos dicho. Todo esto añadido a que permite entrenar redes neuronales de forma mucho más eficiente y eficaz que las anteriores opciones.

Sin embargo, los LLMs tienen limitaciones, ya que tienen un "corte" en su conocimiento. Esto se refiere a que la fase de entrenamiento de la red neuronal se debe hacer con datos recogidos hasta una cierta fecha, además de que la fase de entrenamiento toma tiempo. Por ejemplo, el famoso modelo de ChatGPT gratuito que se usaba antes de la actualización del 15 de mayo de 2024, tenía conocimiento actualizado hasta septiembre de 2021, por lo que no podía responder a eventos que hubiesen ocurrido tras esa fecha. Por otra parte, otro gran problema es el de las llamadas alucinaciones [5]. Una alucinación hace referencia a cuando una inteligencia artificial nos da un resultado inesperado o, más bien, se lo inventa. Las alucinaciones existen porque las redes neuronales son imperfectas y se basan en aproximaciones y estadística, pero con el tiempo fueron y siguen saliendo muchas soluciones a este problema. Un ejemplo de estas son los mencionados RAG o *Retrieval-Augmented Generation*, y será el principal enfoque del siguiente apartado de esta sección.

2.2 *Retrieval-Augmented Generation*

Este apartado de la sección del estado del arte estará enfocado en el estudio de los paradigmas y técnicas más populares y efectivas que han ido surgiendo para la creación del proceso llamado *Retrieval-Augmented Generation* y la mejora de su rendimiento, detallando también pros y contras de las mismas.

Retrieval-Augmented Generation (RAG) se trata de una técnica para dar, al LLM, acceso a nuevo contenido específico de un determinado tema, de modo que se amplíe el propio conocimiento del modelo sobre ese tema o que se adquiera nuevo directamente [16]. No obstante, cabe mencionar que debido a la constante mejora de los grandes modelos de lenguaje [17], [18], para un futuro es probable que RAG solo se use para dar al modelo conocimiento nuevo en entornos en los que se quiera mantener la privacidad de la información dada al modelo. En la Figura 4 podemos observar un esquema de la estructura típica del funcionamiento de un RAG.

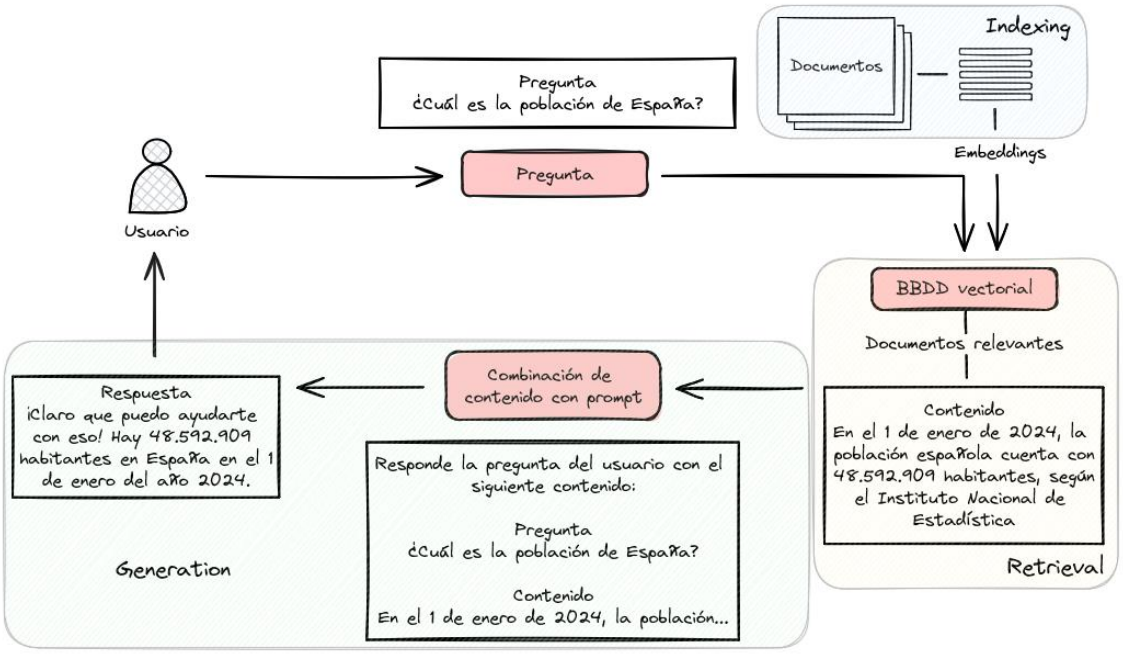

Figura 4: estructura clásica de un RAG.

Normalmente, cuando hacemos un RAG buscamos usar el contenido de una base de datos para responder preguntas, una práctica muy común representada en la Figura 4 y que haremos más adelante en la sección de implementación del RAG, es usar documentos de texto. El texto e información de estos documentos se divide en un número K de segmentos o *chunks* a elegir, de los que se extraerán los vectores de *embedding* y se almacenarán en una base de datos vectorial, es decir, una base de datos que contiene información en forma de vectores, haciéndola ideal para el procesamiento de *embeddings*, además de ser más eficiente y que, por tanto, reduzca la latencia de respuestas del RAG. Luego, cuando un usuario haga uso del LLM haciéndole una pregunta, como, por ejemplo, "¿Cuál es la población de España?" En vez de pasar esa pregunta directamente al LLM, se enviará a la base de datos vectorial, donde se llevará a cabo algún tipo de búsqueda por similitud de *embeddings*. Tras esta búsqueda por similitud, se debería encontrar algún contenido que mencione cuál es la población española, contenido que se combinará con la pregunta o *prompt* del usuario. Por último, enviaremos la pregunta y el contenido al LLM, esperando que devuelva la respuesta correcta al usuario [7].

Como puede observarse, RAG es una práctica constituida por diferentes pasos que tratan de mejorar o aumentar el conocimiento de la red neuronal que forma a un LLM, pero son una serie de pasos independientes entre sí. Muchos hacen una comparativa del RAG con Frankenstein, ya que está formado por distintas piezas, tratando de hacer funcionar algo y, según sea la combinación de piezas, los resultados pueden ser mejores o peores. Sabiendo esto, narraremos en este apartado aquellas técnicas de estos pasos que han ido surgiendo durante los últimos meses, haciendo principal hincapié en las que están siendo más generalmente aceptadas. Ahora en adelante nos referirenos, con la nomenclatura que se ha estado usando en la

[7] https://www.youtube.com/watch?v=ahnGLM-RC1Y

comunidad, a los pasos del RAG que hemos descrito. Una vez más, podemos observar la Figura 4 para ver los siguientes términos: fase de división de la información de los documentos en K segmentos e indexarlos o *indexing*, fase de recopilación de *embeddings* o *retrieval* y fase de generación de respuesta o *generation*, entre las que surgirán pasos intermedios que harán uso ingeniería de entradas del usuario o *prompt engineering* o la reevaluación de combinaciones de contenido con *prompts* o *reranking* [18]. Dichos pasos intermedios también suelen recibir los nombres de fase de *pre-retrieval* y *post-retrieval*, respectivamente. Cabe mencionar que, tal y como se ha mencionado antes, los resultados de las combinaciones de las técnicas de los distintos pasos pueden ser mejores o peores, pero, ¿cómo podemos medir eso? Hasta ahora, la medición de resultados también es un campo que está siendo descubierto, pero hay técnicas ampliamente aceptadas de las que haremos uso.

2.2.1 Estructuras de *Retrieval-Augmented Generation*

La estructura clásica de un RAG, representada en la Figura 4, al tratarse de una práctica tan poco precisa, ha recibido el nombre de *Naive* RAG [19], pero, entre cada paso de los marcados con casillas rosas en la mencionada Figura, han ido surgiendo nuevas variantes o ampliaciones, dando lugar a nuevos subtipos de RAG más desarrollados a los que se les ha llamado *Advanced* RAG [19] y *Modular* RAG [20, 21], en orden correspondiente a su nivel de mejora o desarrollo. Obviamente, hay unas razones claras que han llevado a hacer dichas mejoras del *Naive* RAG. Estas razones eran:

1. La dependencia de que el usuario proporcionase un *prompt* adecuado al LLM.
2. La segmentación de documentos o *chunking* subóptima, que llevaba a respuestas imprecisas o alucinaciones, como comentábamos al final del apartado 2.1.
3. La ausencia de reevaluación de *chunks* en función de la pregunta o *prompt* del usuario.

A continuación, detallaremos cada una de las mejoras del RAG clásico que lo llevan a la categoría de RAG avanzado y del avanzado al modular. Obsérvese la Figura 5, se trata de una comparación de las tres estructuras mencionadas.

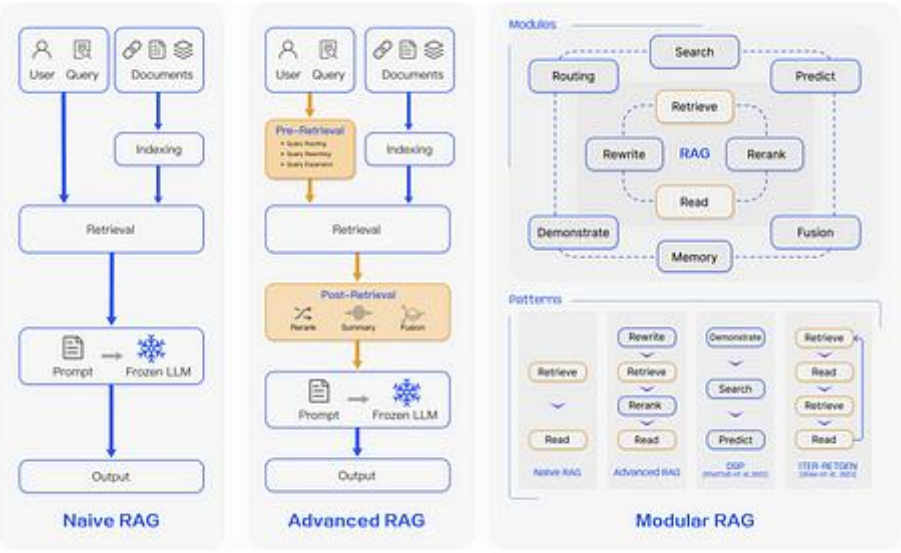

Figura 5: comparación de paradigmas de RAG. [21]

En la Figura 5 se muestra la diferencia entre *Naive* y *Advanced* RAG, que es la incorporación de las fases de *pre-retrieval* y *post-retrieval* mencionadas antes, donde se incluyen los pasos de *prompt engineering* y *reranking*. Asimismo, el cambio de *Advanced* a *Modular* se delinea con la inclusión de automatizaciones, búsquedas a nuevas fuentes de información e iteraciones de los pasos que solo se hacen una vez en las arquitecturas predecesoras. Por tanto, entendemos que la estructura modular es la estructura avanzada en la que el gran modelo de lenguaje ha pasado por un entrenamiento en base a los documentos a procesar por el mismo, entendiendo por entrenamiento el proceso similar al que explicábamos sobre las redes neuronales en la sección de introducción de esta memoria, añadido a las automátizaciones ya mencionadas.

Antes de continuar, es conveniente introducir un término de mucha influencia de cara al entrenamiento, en el ámbito del RAG, de un gran modelo de lenguaje. Hablamos del *Fine-tuning* [22]. Como sabemos, un LLM genera textos únicos, por lo que, por ejemplo, si le preguntamos una misma pregunta en ocasiones diferentes, nos sabrá responder siempre, pero las palabras y la forma del mensaje probablemente serán diferentes. Este hecho tiene sus ventajas y desventajas, pero en lo que se refiere a la calidad de respuestas en el RAG, es más negativo que positivo. Para brindar una solución a esto, llegó *Fine-tuning*, una práctica para enseñar al modelo a devolver respuestas siguiendo siempre un mismo patrón. En la Figura 6 podemos apreciar un ejemplo de esta práctica.

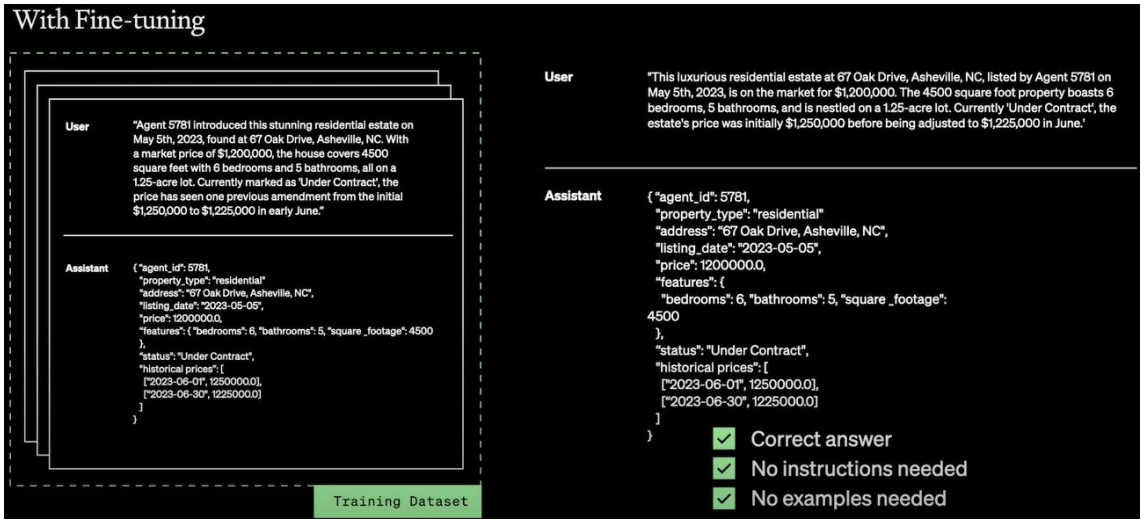

Figura 6: ejemplo de respuesta de LLM con Fine-tuning. [7]

La Figura 6 muestra cómo entrenar un LLM indicando una forma fija de salida dado un tipo de entrada definido. Consistirá en procesar muchos textos similares para obtener resultados como los que se ven al lado derecho de la Figura 6.

2.2.2 Componentes

Una vez hemos entendido cómo diferenciar las estructuras *Naive*, *Advanced* y *Modular*, podemos entrar en detalle sobre las variantes de las técnicas que los componen.

2.2.2.1 Indexing

Como sabemos, la fase de indexación o *indexing* es en la que obtenemos los *embeddings* del texto, y está principalmente constituida por el *chunking*.

Chunking consiste en dividir el texto de los documentos en segmentos o *chunks* de los que extraeremos los *embeddings* y pasaremos al LLM, esto se hace para que el LLM pueda encontrar un segmento conciso con la información que necesitamos que utilice para responder correctamente. Y es que, en un principio puede parecer que cuantos más documentos e información le pasemos al LLM, mejor podrá responder, pero ocurre más bien lo contrario. Al igual que los humanos, los LLMs centran mejor su atención al principio y al final de la información dada, por lo que la información en el medio tiende a perderse. Este problema recibe el nombre de *"Lost in the middle"* [23]. Luego, conseguir *chunks* que tengan la información clave de una forma concisa es importante. Veamos qué tipos de técnicas se suelen aplicar:

a) Separación de caracteres: suponiendo que la información dada está constituida únicamente por texto, se trata de la forma más básica de dividirlo, haciendo *chunks* de N caracteres estáticos (p. ej. 50, 100, 1000, etc.). Este es un método sencillo y útil para entender la práctica, pero no es recomendable usarlo, ya que no se tiene en cuenta la estructura del texto del documento. Para ayudar a visualizar el porqué de su pobre utilidad, podemos observar la Figura 7. En este caso, se escogen 453 caracteres.

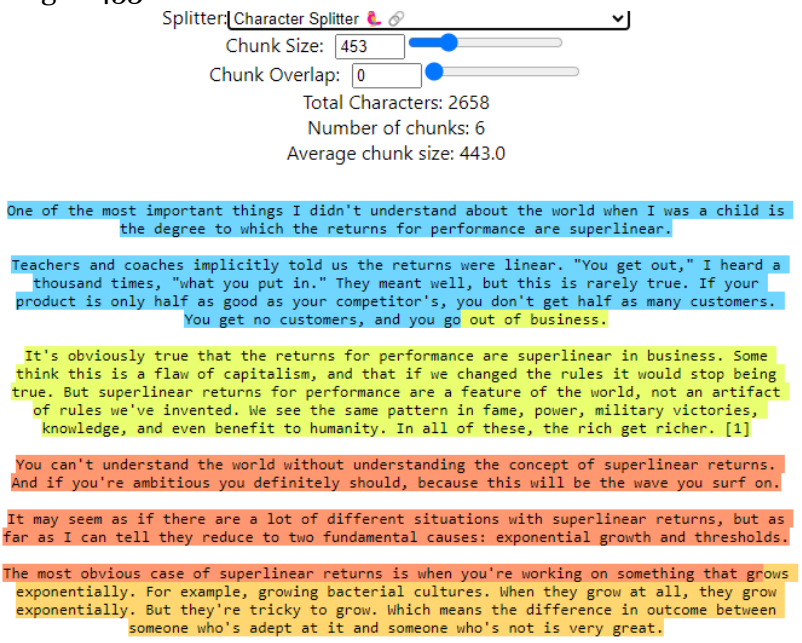

Figura 7: representación visual de la separación de caracteres. [8]

Los *chunks* dividen los párrafos de forma no óptima en la Figura 7.

b) Separación de caracteres recursiva: en este caso sí tendremos en cuenta la estructura del documento, ya que haremos uso de determinados separadores como los saltos de línea, nuevos párrafos, espacios o comillas. Añadido a eso,

[8] https://chunkviz.up.railway.app/

mantendremos la idea de los N caracteres, pero en este caso para limitar el tamaño de los *chunks*. De nuevo, podemos ver en la Figura 8 cómo se dividirían los parráfos del texto de la imagen usando este método con un número de 453 caracteres.

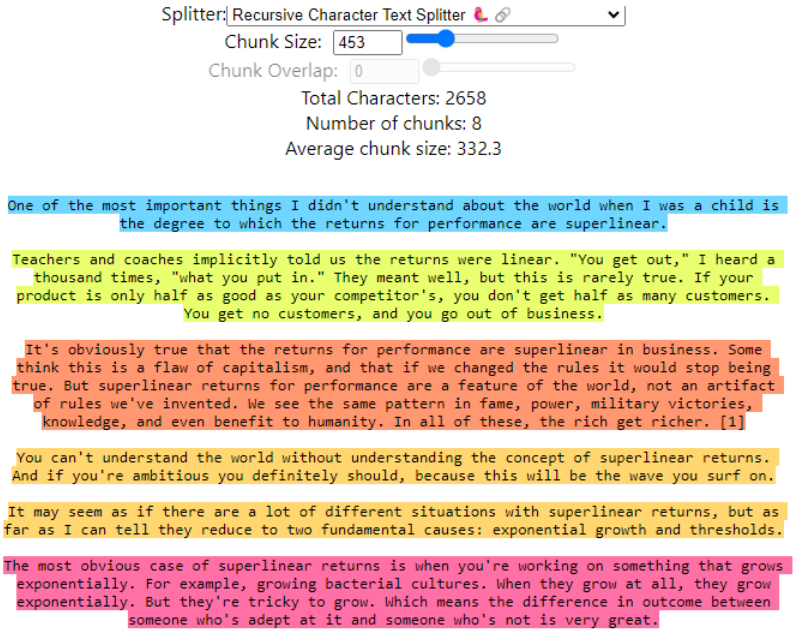

Figura 8: representación visual de la separación de caracteres recursiva. [8]

En comparación al anterior método, la separación en la Figura 8 es más coherente y por tanto de mayor utilidad para un LLM.

c) Separación de documentos específicos: pensado para cuando tenemos distintos formatos de documentos, que es lo más frecuente. La idea es la misma que la separación recursiva, pero ahora incluiremos separadores para tener en cuenta imágenes, tablas, código...

d) Separación semántica: hasta ahora hemos hablado de crear *chunks* con un número estático de caracteres, es decir, un parámetro que no estamos adaptando plenamente. Esta idea trata de crear nuevos *chunks* mediante la agrupación por similitud semántica de *chunks* creados con los métodos de anteriores. En la práctica se resume en la agrupación de *chunks* mediante la similitud de sus *embeddings*, por lo que conviene utilizar un número N de caracteres no muy grande para los *chunks* iniciales. Un ejemplo sería el de usar oraciones como *chunks* iniciales.

e) Separación con agente: se trata de hacer que el propio LLM haga *chunks* por sí mismo en vez de pasar por el proceso de extracción y medida de similitud de *embeddings*. Es una buena idea si el caso permite usar los LLMs más potentes, sin necesidad de ser *open source*. Una forma de separación basada en esta idea es la de las propuestas (proposiciones) [24], que consiste en obtener pequeñas oraciones que contienen medias verdades sobre el texto. Por ejemplo, el fragmento de texto “Jorge ha ido a la piscina. Le gusta nadar.” Se convertiría en las propuestas “Jorge ha ido a la piscina” y “A Jorge le gusta nadar”.

f) Técnica *Small-to-Big*: [25] esta serie técnicas usan *chunks* más pequeños para usarlos de atajo o índice hacia un *chunk* más grande que contiene la información deseada. Estos *chunks* pequeños se crean, por tanto, a partir de los grandes, de nuevo mediante la ayuda de los LLMs, y normalmente suelen ser resúmenes o fragmentos de los *chunks* grandes.

g) Adición de metadatos: esta práctica no entraría exactamente en el subgrupo de técnicas de *chunking*, sino que directamente como técnica de *indexing*. Consiste en enriquecer los *chunks* con metadatos como: número de página, nombre de archivo, autor, categoría y fecha. Por ejemplo, asignando diferentes pesos a las fechas de los documentos podemos conseguir que nuestro RAG sea consciente del tiempo, asegurando la actualidad de la información. Además, podemos aplicar la técnica de los LLMs aquí también para extraer metados artificiales, como resúmenes de párrafos o preguntas hipotéticas. Por otro lado, podemos seguir esta lógica de las técnicas de *Small-to-Big* con la adición de metadatos, pero esta vez con los documentos en sí en vez de con *chunks*.

h) *Knowledge graphs*: por último, se puede optar por reorganizar la información de los documentos, que generalmente es información desestructurada, en una reorganización de la información llamada grafo de conocimiento. Como se observa en la Figura 9, un grafo está formado por nodos y aristas. Los nodos corresponderán a identidades, personas o cosas que se unen semánticamente con las aristas. Centrado en el contexto del RAG, tenemos la técnica de *Knowledge Graph Prompting* [26], que trata de unir párrafos, páginas o tablas representados como nodos mediante similitud léxica o semántica, representada con las aristas. El enfoque del grafo de conocimiento ayuda a reducir las alucinaciones y mejorar la fase de *retrieval*, permitiendo a los LLMs generar respuestas más coherentes e incrementando la eficacia del RAG. Esta práctica correspondería al RAG Modular.

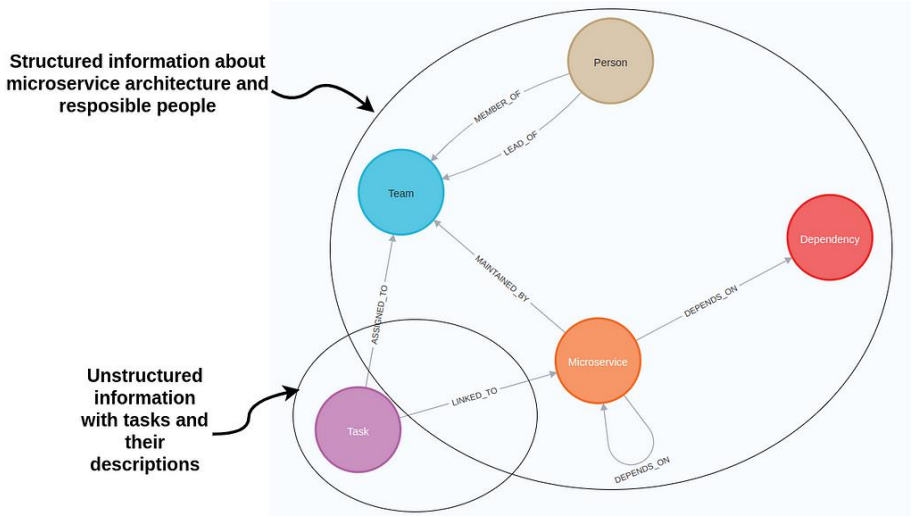

Figura 9: ejemplo de un grafo de conocimiento. [9]

[9] https://neo4j.com/developer-blog/knowledge-graph-rag-application/

2.2.2.2 Prompt Engineering

Prompt engineering o ingeniería de *prompts* hace referencia a la creación de *prompts*, es decir, consultas en forma de texto que un usuario escribe a un LLM, con la intención de que sean precisos y eficaces para explotar las capacidades de la IA [5]. Como vimos antes, en el contexto del RAG de la estructura del *Advanced* RAG, la ingeniería de *prompts* se aplica en una fase llamada *pre-retrieval*, y se llevaría a cabo a la vez que se hace la fase de *indexing*. Como sabemos, dichas capacidades incluyen una amplia gama de tareas comunes y complejas, como responder preguntas y razonamiento aritmético. Veamos detalladamente cómo se suele componer un *prompt*:

1. Instrucción: una tarea o instrucción específica a realizar por el LLM.
2. Contexto: puede involucrar información externa o contexto adicional que guíe mejor al modelo.
3. Datos de entrada: una pregunta para la que nos interesa encontrar una respuesta.
4. Indicador de salida: indica el tipo o formato de la salida.

Cabe mencionar que no todos los componentes son necesarios para un *prompt* y el formato depende de la tarea en cuestión.

Como es de esperar, distintas técnicas han ido surgiendo en los últimos años y meses. Y, tal y como se ha expuesto, un *prompt* lo proporciona un usuario, pero en el ámbito del RAG, también es común utilizar a los LLMs para expandir o mejorar estos *prompts*. A continuación, veremos las 3 técnicas de ingeniería de *prompts* clásicas [5, 18], aquellas que cualquier usuario usará, por ejemplo, cuando pregunta algo a ChatGPT en su día a día. Después, veremos cómo se combinan estas técnicas con los modelos de lenguaje:

a. *Zero-shot prompting*: también llamada *prompt* sin entrenamiento previo, es la técnica más básica. Consiste en dar una instrucción al modelo para generar una respuesta sin proporcionar ningún ejemplo ni entrenamiento previo.

   Para mejorar la calidad de un *prompt zero-shot*, tenemos que: primero, proporcionar instrucciones claras y directas utilizando verbos en imperativo como "Escribe", "Clasifica", "Resume", "Traduce" u "Ordena". Segundo, aportar contexto específico sobre la tarea deseada, detallando y describiendo el *prompt* de manera exhaustiva para mejorar los resultados. Tercero, utilizar la técnica de "*role-playing*", comunicando al modelo que asuma un rol o personaje concreto, de modo que oriente las respuestas al conocimiento de ese ámbito específico. Cuarto, hacer uso de "*emotion prompting*", cuando usamos palabras con énfasis emocional (p. ej. “es crucial”, “es inaceptable”), el modelo reflexiona más. Finalmente, se puede considerar el uso de formatos de texto como markdown para hacer la respuestas más legibles y vistosas.

b. *Few-shot prompting*: esta técnica consiste en proporcionar al modelo un pequeño número de ejemplos (*shots*) para que pueda aprender de ellos. Por lo que, en función de la complejidad de la tarea, habrá que proporcionar uno o más ejemplos. La parte negativa de *few-shot prompting* es que tiene limitaciones para tareas como resolver problemas de aritmética. Veamos un ejemplo:

Prompt:

Clasifica las frases según si son de Harry Potter (HP) o El Señor de los Anillos (SA). Por ejemplo:

¡50 puntos para Griffindor!
Clasificación: HP

¡Cuenta con mi hacha!
Clasificación: SA

Atrapa la snitch dorada
Clasificación: HP

Atrapa mi tesoro
Salida*:*

Clasificación: SA

c. *Chain-of-Thought (CoT) Prompting*: *Chain-of-Thought* consiste en abordar tareas complejas dividiendo el problema en una cadena lógica de pasos. Se combina con *Zero-shot* y *Few-shot*, recibiendo respectivamente los nombres de *Zero-shot-CoT* y *Manual-CoT*. En el primero, simplemente se añade "Piensa paso a paso" en el *prompt* para que el LLM tenga más facilidad para razonar. En el segundo, que en general proporciona mejores resultados, se dan ejemplos de demostraciones que muestran una pregunta y su respectiva respuesta con los pasos lógicos para llegar hasta ella. La Figura 10 ilustra lo descrito.

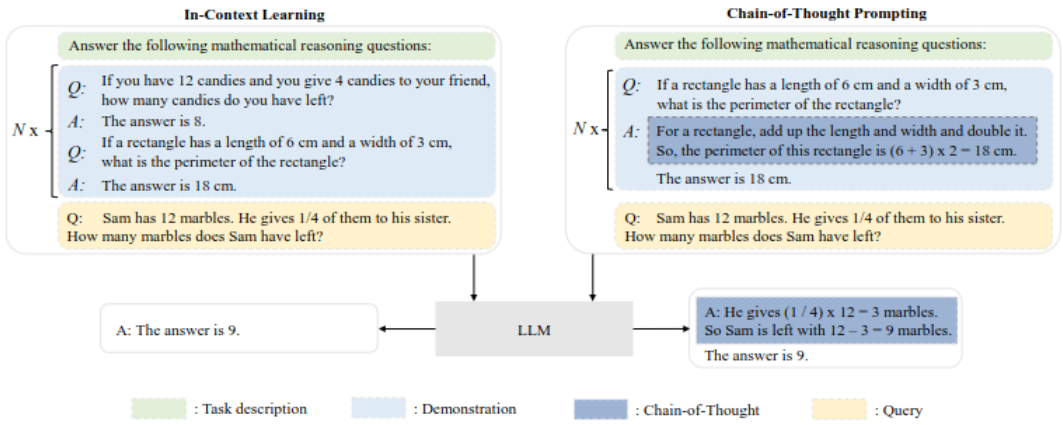

Figura 10: diferencia entre In-Context Learning (o sea, Zero-shot y Few-shot) y Chain-of-Thought. [5]

Los métodos explicados hasta ahora son los métodos clásicos o básicos. A continuación, expondremos otros más elaborados.

d. *Multi-query (Multi-prompting)* y *Query transformation (Prompt transformation)*: centrada principalmente para el uso de un RAG, *Multi-query* es una técnica simple pero efectiva. Consiste en pedir al modelo de lenguaje que genere N preguntas similares a la que ha proporcionado el usuario, que de lo normal será del estilo *Zero-shot*, para así tener más probabilidad de emparejar el *prompt* con un *chunk* con la respuesta buscada.

Por otro lado, incluimos en este mismo apartado la práctica de la transformación del *prompt*, que no es más que pedir al LLM que reescriba la pregunta que ha proporcionado el usuario para que sea más fácil de emparejar con los *chunks*. Como se puede ver, es muy similar a la anterior [27].

e. *Automatic Chain-of-Thought* (CoT)**:** una desventaja de usar CoT manualmente es el esfuerzo que supone diseñar preguntas y respuestas con cadenas de razonamientos. Por ello, existe una variación de esta técnica llamada Auto-CoT [28, 29]**,** que trata reducir los esfuerzos manuales de la creación de ejemplos, creándolos automáticamente. Auto-CoT consta de dos etapas principales: por un lado, la creación de las preguntas sobre un conjunto de datos dado y la división de esas preguntas en grupos o *clusters*. Luego, una vez formados esos *clusters*, escoger una pregunta representativa de cada uno y desarrollar la respuesta con la cadena de razonamientos mediante un LLM y *Zero-Shot-CoT*. Una vez más, observando la Figura 11 entenderemos mejor esta técnica.

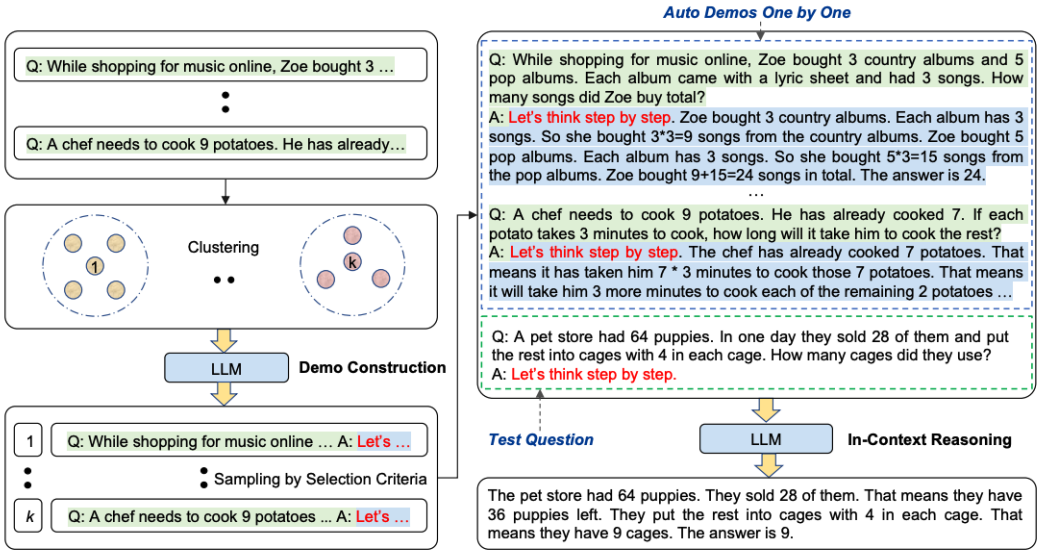

Figura 11: ejemplo de implementacón de Auto-CoT. [28]

f. *Self-Consistency*: la técnica de *Self-Consistency [30]* plantea una mejora a un problema que da *Chain-of-Thought*: la llamada decodificación *naive greedy* o codiciosa ingenua. "Decodificación codiciosa ingenua" hace referencia a las respuestas que proporciona CoT, que son como tratar de escribir una historia con oraciones sueltas. Cada oración se basa únicamente en la oración anterior sin pensar demasiado en cómo encaja esta oración en la historia general.

Dicho esto, la solución de *Self-Consistency* se basa en cómo las personas resuelven problemas en la vida real. Los humanos normalmente consultamos múltiples fuentes o partimos de distintas ideas para llegar a una decisión bien informada, esto es lo que se muestra en el ejemplo de la Figura 12. *Self-Consistency* hace que el modelo de lenguaje genere distintas respuestas con razonamientos del estilo CoT para una sola pregunta, para después alentarlo a evaluar críticamente su propio razonamiento. Al hacerlo, aumenta la probabilidad de generar una respuesta precisa e imparcial.

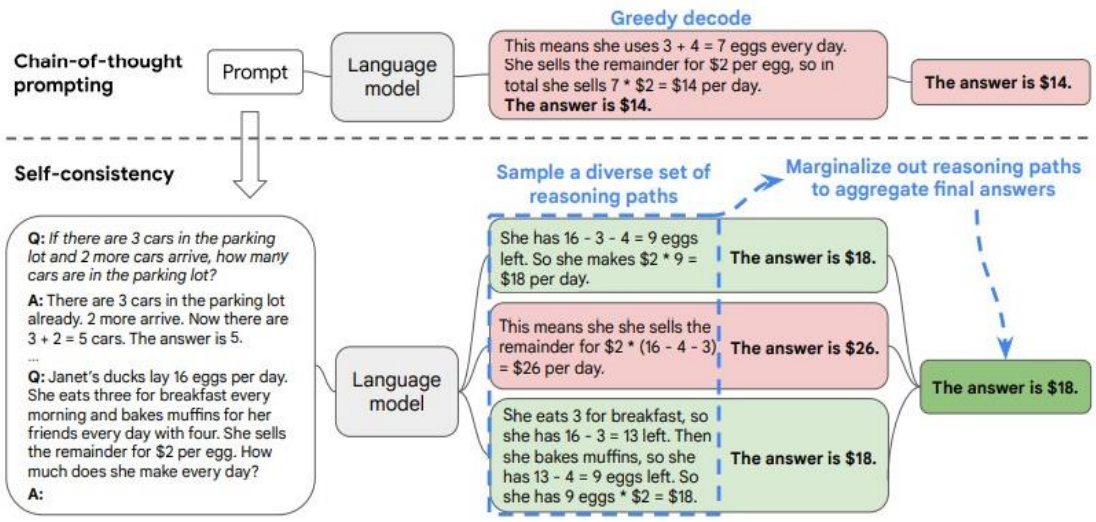

Figura 12: ejemplo de Self-Consistency. [30]

En pocas palabras, *Self-Consistency* es un enfoque que llama al modelo de lenguaje varias veces con el mismo *prompt* y toma el resultado más consistente como la respuesta final. La idea es muestrear múltiples cadenas de razonamiento diferentes a través de CoT y usarlas para seleccionar la respuesta más consistente. Esto ayuda a mejorar el rendimiento de CoT en tareas complejas.

g. *Tree-of-Thoughts* (ToT): esta técnica, inspirada también en CoT, adopta una estructura más ramificada y exploratoria, permitiendo al modelo considerar múltiples caminos y posibilidades en paralelo antes de tomar una decisión final. Cuando hablamos de caminos, hablamos de generar pensamientos (o partes de cadenas de razonamiento) que sirven de pasos intermedios. Este enfoque permite a los LLMs autoevaluar el progreso de los pensamientos intermedios hacia la resolución del problema. Además, todo esto se puede combinar entonces con algoritmos de búsqueda, permitiendo anticipar pasos o pensamientos futuros o volver a anteriores. Podemos visualizar esta técnica en la Figura 13.

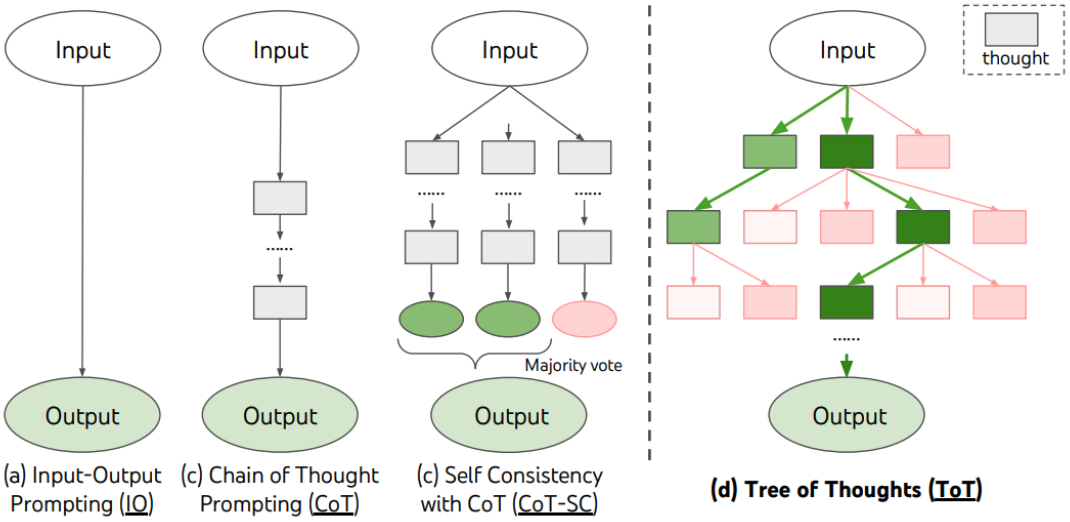

Figura 13: diagrama de ToT comparado con los de Zero-Shot y Few-Shot, CoT y Self-Consistency. [31]

h. *Least-to-Most*: aquí tenemos otra práctica de *prompting* que trata de dar arreglo a la decodificación de *Chain-of-Thought*. Como se ve en la Figura 14, *Least-to-Most prompting* se compone de dos fases: primero, la descomposición de un problema complejo en una lista de subproblemas más sencillos y, segundo la resolución secuencial estos subproblemas. Por tanto, las resoluciones a los nuevos subproblemas dependerán de las de los anteriores. Ambas etapas se ejecutan con *Few-shot*.

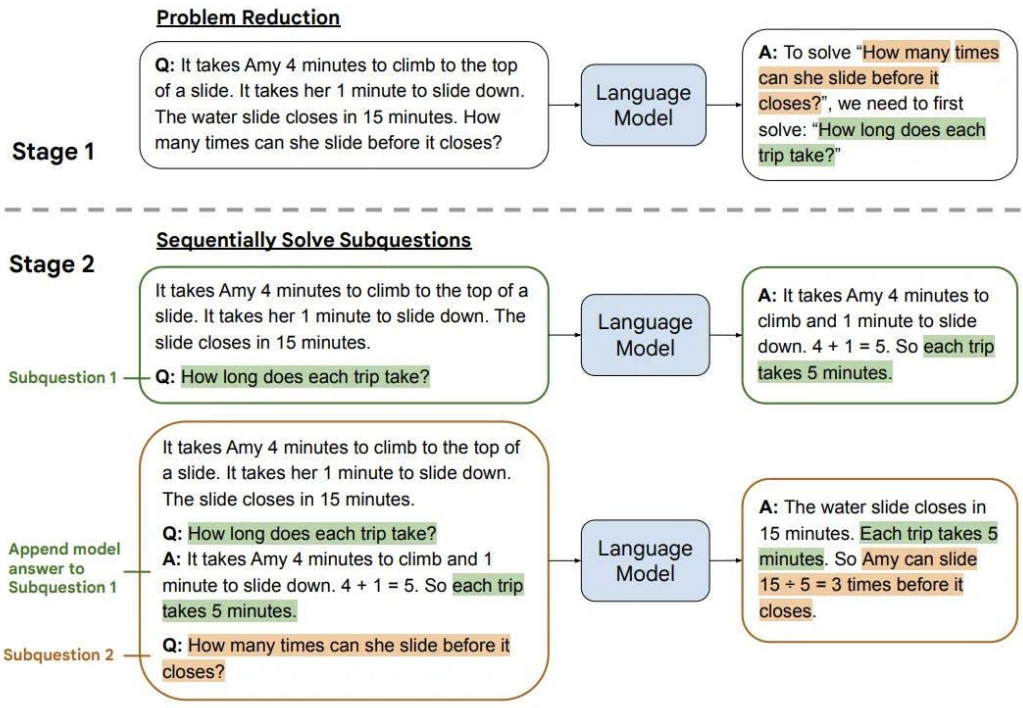

Figura 14: ejemplo de Least-to-Most. [32]

i. *Chain-of-Verification* (CoVe): CoVe es una idea propuesta para intentar reducir las posibles alucinaciones del LLM. En el paper que se presenta [33], se divide en cuatro pasos: primero, hacer una pregunta al LLM, segundo, generar un listado de subpreguntas para cada afirmación que hace el modelo en su respuesta a la pregunta inicial, tercero, preguntar por separado esas subpreguntas al modelo, ya que las respuestas suelen sufrir menos alucinaciones de esta manera, y cuarto, generar una última respuesta revisando las posibles inconsistencias entre la respuesta inicial y las subrespuestas posteriores. Consultemos la Figura 15 para ver un caso de uso de lo explicado.

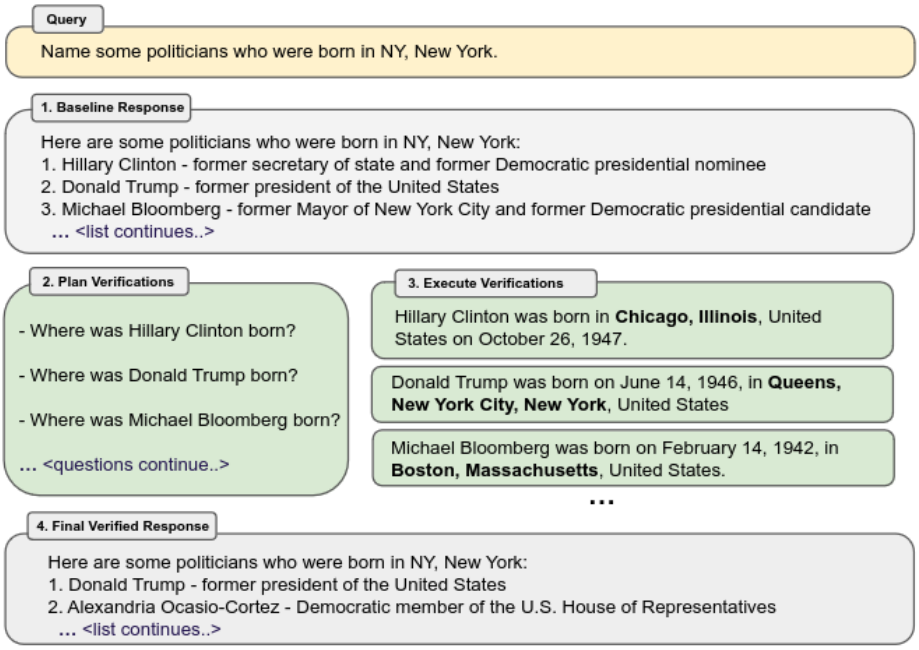

Figura 15: ejemplo de Chain-of-Verification sobre políticos nacidos en Nueva York. [33]

j. ReAct: este método correspondría más a la estructura de *Modular RAG* por el hecho de que se añade un módulo de acceso a internet, algo que va más allá del *Advanced RAG*. Este caso también parte de *Chain-of-Thought*, pero parte de su falta de acceso al mundo externo o su incapacidad para actualizar sus conocimientos, que puede dar lugar a alucinaciones de hechos y la propagación de errores.

   Dicho esto, ReAct es un paradigma general que combina el razonamiento y la actuación con los LLMs. ReAct pide a los LLMs que generen razonamientos y acciones para una tarea. Esto permite realizar un razonamiento dinámico para crear, mantener y ajustar planes para cómo actuar al tiempo que permite la interacción con entornos externos (p. ej., Wikipedia) para incorporar información adicional al razonamiento. Veamos un ejemplo en la Figura 16.

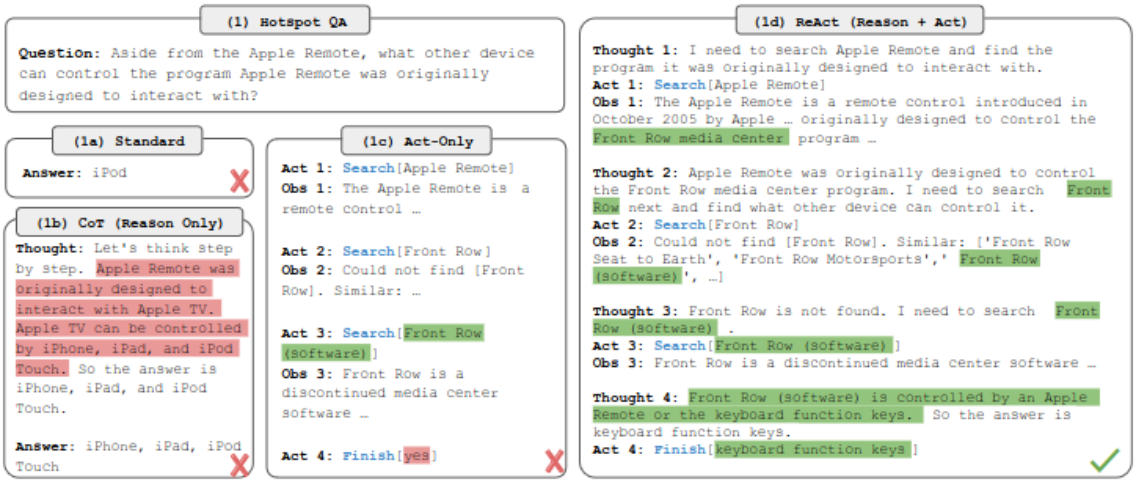

Figura 16: comparado con otros métodos, se muestra cómo ReAct contesta a una pregunta sobre Apple Remote. Primero genera un pensamiento sobre qué tiene que buscar en internet y después lo busca, para finalmente proporcionar la respuesta. [34]

k. Adición de metadatos: tal y como veíamos en el apartado de técnicas de *chunking*, también podemos enriquecer los *prompts* mediante la inclusión de metadatos que ayuden a emparejarlos con los *chunks* con los que más coincidan. Los metadatos pueden tener forma de palabras clave, por ejemplo.

Como hemos podido observar, muchas técnicas comparten conceptos muy similares, ya que todas parten de las tres primeras mostradas. Muchas otras ideas han sido publicadas, pero las expuestas han sido de las más notorias.

2.2.2.3 Modelos de embeddings

Dedicamos un apartado para mostrar la relevancia de la elección de técnicas o modelos de *embeddings*, algo tan relevante como el LLM a usar. Se añade una dificultad a esta elección, y es la del uso del idioma español, del que hay menos opciones y que no suelen estar tan actualizadas o ser tan potentes como para otros idiomas como el inglés o el chino. Principalmente se distinguen dos tipos de modelos de *embeddings* o *retrievers*, junto con dos técnicas para sacarles más partido [18]:

1. *Sparse retriever*: esta categoría es más simple y en un primer momento puede parecer que esté anticuada, pero aún así se sigue usado con frecuencia por eficiencia y estabilidad. Es muy común leer en documentos el uso del algoritmo BM25 [35] y también hay otros como TF-IDF [36], aunque no tan populares.

2. *Dense retriever*: Se trata del tipo más usado. Cuando se habla del paradigma del RAG y la fase de extracción de *embeddings*, se da por hecho que es este tipo de *retriever* lo que se usa. Esto se debe a su mayor capacidad de codificación, aunque requiere de gran capacidad de computación en algunos casos.

   En la página de Hugging Face [10], una gran comunidad involucrada en los grandes modelos de lenguaje, podemos consultar una tabla en tiempo real con aquellos *dense retrievers* publicados que mejor funcionan para los idiomas inglés, chino, francés y polaco, a fecha en la que se escribe este trabajo. Para

[10] https://huggingface.co/spaces/mteb/leaderboard

encontrar modelos que funcionen bien con el español, tenemos que indagar más. Podemos optar por modelos mostrados en esta tabla que abarcan un gran número de idiomas o recurrir a otros que se usaban hace pocos años y que siguen dando buenos resultados. Estos modelos relativamente desactualizados pero potentes son los de la arquitectura BERT [37], propuesta por el equipo de Google y pionera en el campo. Dentro de la arquitectura BERT, disponemos una vez más de modelos que abarcan múltiples idiomas y también BETO [11], que es esta arquitectura pero entrenada únicamente en español.

3. Híbrido: tanto los *sparse* como los *dense retrievers* capturan diferentes variables complementarias y se pueden beneficiar mutuamente. Los *sparse retrievers* pueden proporcionar resultados iniciales para entrenar a los *dense retrievers*, mejorando la extracción de *embeddings* para los *prompts* del usuario (principalmente cuando son *Zero-shot*) o el entendimiento de *prompts* con nombres o entidades que no son muy conocidos.

4. *Fine-Tuning* para el *retriever*: similar a lo visto en el apartado 2.2, se podría entrenar al recuperador para generar *embeddings* dadas unas preguntas con unos mismos contextos y palabras frecuentes. Esta práctica correspondería al paradigma del RAG modular.

2.2.2.4 Reranking

Reranking es la parte final de la fase de *generation* y por tanto el último paso a dar en el RAG antes de generar la respuesta para el usuario. Una vez se hayan obtenido los *chunks*, el *prompt* y sus respectivos *embeddings*, evaluaremos los *chunks* para seleccionar un número N que mejor emparejen con el *prompt*. Después, esos *chunks* evaluados serán los que proporcionen el contexto de la respuesta. Podemos distinguir tres formas generales de aplicar este *ranking* [38, 39]:

1. Modelos específicos de *reranking*: son modelos diseñados para esta tarea específica, optimizados con entropía cruzada.
2. Uso de los LLMs: Como se puede intuir, los LLMs, al comprender detalladamente tanto el *chunk* como el *prompt*, permiten capturar la información semántica.
   Una vez los modelos extraen los *chunks*, podemos evaluar la calidad de emparejamiento mediante métricas como Hit Rate y MRR (*Mean Reciprocal Rank*) [12].

2.2.2.5 Evaluación de RAG

Diversas formas de evaluar el funcionamiento de un RAG han sido propuestas, tales como *Benchmarks* o herramientas que calculan medidas de calidad. Entre esas herramientas están RAGAS (*Retrieval Augmented Generation Assessment*) [40], ARES [41] o TruLens [13], pero RAGAS ha sido la más aceptada con diferencia hasta el

[11] https://github.com/dccuchile/beto

[12] https://www.llamaindex.ai/blog/boosting-rag-picking-the-best-embedding-reranker-models-42d079022e83

[13] https://www.trulens.org/trulens/getting_started/core_concepts/rag_triad/

momento, por lo que RAGAS será la herramienta que usemos para la evaluación de nuestro RAG en la sección de experimentación.

Entrando en detalle, RAGAS calcula 4 medidas de calidad, 2 para la fase de *retrieval* y 2 para la de *generation* como se puede apreciar en la Figura 17. Estas medidas son:

1. *Faithfulness* o fidelidad: hasta qué punto es exacta la respuesta generada.
2. *Answer relevancy* o relevancia de la respuesta: mide cómo de relevante es la respuesta respecto a la pregunta.
3. *Context precision*: se trata de un ratio señal/ruido del contexto obtenido.
4. *Context recall* o recuerdo del contexto: mide si se ha podido obtener o recuperar toda la información necesaria para responder a la pregunta.

Para la extracción de esas afirmaciones se hace uso de un LLM y un modelo de *embeddings* a elegir, este es un aspecto crucial.

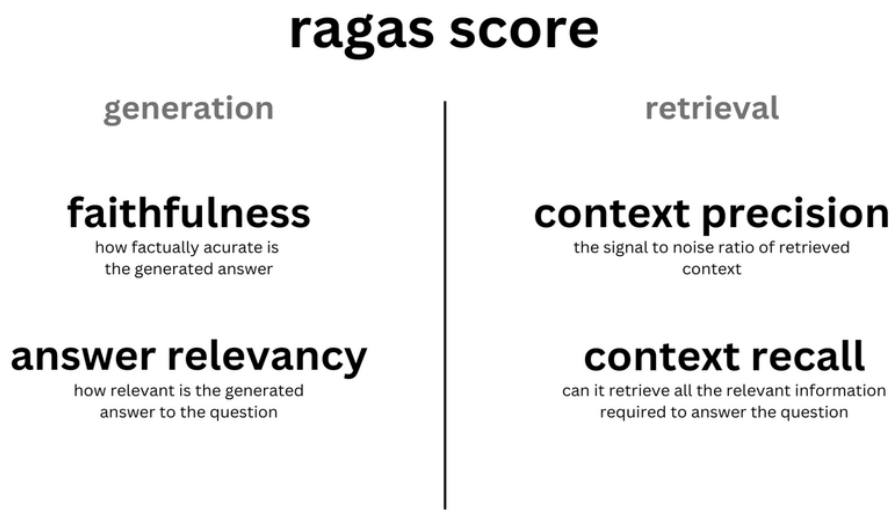

Figura 17: medidas de calidad de RAGAS. [14]

Una vez proporcionada la explicación teórica, pasemos a ver cómo se calculan exactamente estas medidas [40]:

1. $Faithfulness = \frac{[Número\ de\ afirmaciones\ en\ la\ respuesta\ generada\ que\ se\ pueden\ inferir\ del\ contexto\ dado]}{[Número\ total\ de\ afirmaciones\ en\ la\ respuesta\ generada]}$

Para la extracción de esas afirmaciones se hará uso del LLM y el modelo de *embeddings* elegidos. En concreto, se hace uso de dos plantillas de *prompts*:

- *"Given a question and answer, create one or more statements from each sentence in the given answer.*

  question: [question]

  answer: [answer]"

[14] https://docs.ragas.io/en/stable/concepts/metrics/index.html

- *"Consider the given context and following statements, then determine whether they are supported by the information present in the context. Provide a brief explanation for each statement before arriving at the verdict (Yes/No). Provide a final verdict for each statement in order at the end in the given format. Do not deviate from the specified format.*

  statement: [statement 1]

  ...

  statement: [statement n]"

Con la primera, se pretende descomponer oraciones largas en afirmaciones más cortas y centradas. Después, para las afirmaciones extraídas, se usa la segunda plantilla, con el fin de verificar si cada afirmación puede ser inferida por el contexto (*chunks*) dado.

2. $answer\ relevancy = \frac{1}{N}\sum_{i=1}^{N}\cos\left(E_{g_i}, E_o\right)$

   $$answer\ relevancy = \frac{1}{N}\sum_{i=1}^{N}\frac{E_{g_i}\cdot E_o}{\left\|E_{g_i}\right\|\|E_o\|}$$

   Donde:

   - E_{g_i} es el embedding de la pregunta generada i.
   - E_o es el embedding de la pregunta original.
   - N es el número de preguntas generadas, que por defecto es 3.

Esas preguntas generadas se obtienen nuevamente con el LLM y usando el siguiente *prompt*:

- *"Generate a question for the given answer.*

  answer: [answer]"

3. $Context\ Precision@K = \frac{\sum_{k=1}^{N}(\text{Precision@k} \times v_k)}{Número\ total\ de\ ítems\ relevantes\ en\ los\ top\ K\ resultados}$

   $$Precision@k = \frac{verdaderos\ positivos@k}{(verdaderos\ positivos@k + \ falsos\ positivos@k)}$$

Donde K es el número total de *chunks* que se seleccionan para responder y $v_k \in \{0,1\}$ es un indicador de relevancia de ítems en el rango k.

4. $context\ recall = \frac{|Oraciones\ de\ la\ respuesta\ correcta\ que\ pueden\ atribuirse\ al\ contexto|}{|Número\ de\ oraciones\ en\ la\ respuesta\ correcta|}$

Como se ha mencionado, estas medidas dependerán de un LLM y un modelo de *embeddings* a elegir para ser extraídas. En el caso de los desarrolladores de RAGAS, usaron el gran modelo de lenguaje *gpt-3.5-turbo-16k* y el modelo de *embeddings* text-*embedding-ada-002* de OpenAI. Para las librerías de ARES y Trulens, dichos modelos coinciden.

Finalizado este subapartado sobre la evaluación, damos por concluido el apartado del *Retrieval-Augmented Generation*.

2.3 Crítica al estado del arte

Entre los proyectos con temáticas similares al de este mismo, tenemos:

- The Chronicles of RAG: The Retriever, the Chunk and the Generator [42]:

  En este paper se hace un trabajo enfocado al portugués de Brasil y a LLMs de OpenAI y de Google. El uso de modelos de lenguaje que no son de código abierto es una buena opción en los casos que se trabaje con documentos públicos, pero no con documentos privados, ya que no se podrá garantizar que se mantenga la privacidad de los mismos. Por otro lado, el paper se centra en probar distintos LLMs, pero únicamente usando una técnica de *retrieval* y de *generation*. Nuestro estudio innova respecto a este en el aspecto de que se prueban diferentes técnicas de *retrieval* y *generation*. Finalmente, el paper usa una métrica propia para evaluar los resultados, mientras que en este trabajo se usa RAGAS. Al ser un paper con 10 citas en Google Scholar, su aportación es innovadora y puede considerarse su uso, pero RAGAS es la métrica número uno de evaluación de RAG.

En general, los papers con trabajos similares al de este proyecto están enfocados al idioma inglés y a LLMs de pago como los de OpenAI. Además, el código está desactualizado a los cambios de las librerías como Langchain. Es por esto que, como se ha mencionado antes en este documento, este estudio se diferencia al enfocarse en el idioma español, en dar código ejecutable a largo plazo y con modelos de lenguaje de código abierto.

3.Análisis del problema

Desde la perspectiva de la empresa, se intenta mejorar los resultados de un RAG local ya implementado, sin embargo, ese RAG local previo se hizo mediante código que no permite adaptarse a actualizaciones, de modo que el nuevo debía implementarse desde cero. Por tanto, la mayor complicación que presenta el tener un RAG local es la ideación del mismo para que pueda ser compatible a la hora de incluir nuevas formas de *chunking*, *prompt engineering*, *reranking* o distintos grandes modelos de lenguaje o de *embeddings*.

Añadido a las incompatibilidades, durante el desarrollo de este proyecto, la gran mayoría de las documentaciones o guías de código consultadas, todas ellas en forma de blogs o repositorios de GitHub, contienen una pobre organización y claridad en la explicación que facilite el aprendizaje de las mismas. Por otro lado, añadido a la mejorable documentación, el constante y rápido avance en el desarrollo de librerías diseñadas para la implementación y/o evaluación de un RAG deriva en una desactualización de las dudas resueltas por la comunidad acerca de posibles fallos de dichas librerías.

La principal librería utilizada para la implementación del RAG, Langchain [15], actúa como base para el desarrollo de aplicaciones de grandes modelos de lenguaje. Aunque fuese la principal herramienta, Langchain también fue el mayor inconveniente, ya que se mantiene en un constante y fugaz desarrollo y actualización. Si bien se mantiene a la última en el campo de la inteligencia artificial y los LLMs, no es así en cuanto a la documentación de la compatibilidad con lo desarrollado en el pasado. En 2023, cuando se lanzó Langchain y se hizo popular, constaba únicamente de una sección principal que servía para trabajar con los servicios de OpenAI [43]. Con el paso del tiempo, Langchain ha crecido, añadiendo muchas otras subsecciones que integran desde otras librerías hasta contribuciones de la comunidad, pero que son volátiles. Todo este problema hace efecto de bola de nieve en los casos de librerías que son completamente dependientes de Langchain, como es RAGAS, cuya documentación oficial está desactualizada desde hace meses a fecha de la realización de este trabajo de fin de grado.

Tras intentos y estudios del funcionamiento de las librerías utilizadas, se decidió mantener aquellos métodos de Langchain que pertenecen a la rama principal de su estructura y que se han mantenido estáticos desde la época de su comienzo (langchain-core). Para los casos como el de RAGAS, la solución que se logró fue una encontrar versión compatible con el código desarrollado con los métodos de la rama principal de LangChain y dejarla fija, todo mediante experimentación.

En conjunto, los problemas de incompatibilidades y de desactualizaciones de las librerías mencionados supusieron una reorganización del tiempo a invertir en el desarrollo de las tareas de este proyecto, llevando a una reducción de la sección de experimentación.

[15] https://js.langchain.com/v0.1/docs/get_started/introduction

3.1 Análisis del marco legal y ético

A mediados de 2024, el Consejo de la UE ha aprobado el *AI Act* [16] o Reglamento Europeo de Inteligencia Artificial, que supone el primer marco regulatorio de la inteligencia artificial. Fundamentalmente, el mencionado reglamento clasifica la inteligencia artificial en función de su riesgo: riesgos inaceptables o prohibidos (por ejemplo, la IA manipuladora o la puntuación social), riesgos altos y riesgos limitados. La mayoría de las regulaciones se centran en los riesgos altos, que principalmente son aquellos que utilizan y automatizan datos personales de individuos.

Por tanto, el objetivo del desarrollo del RAG local de empresa de este trabajo no entraría en riesgo alto, sino en riesgo limitado. Como riesgo limitado, se debe garantizar que los usuarios finales sean conscientes de que están interactuando con IA, y ya que esos usuarios finales serán los mismos miembros de la empresa con la que se desarrolla este proyecto, eso ya se garantiza.

[16] https://artificialintelligenceact.eu/ai-act-explorer/

4. Propuesta

En línea con los objetivos de este proyecto, las herramientas comúnmente usadas y los recursos de la empresa en la que se desarrolla, como se ha ido mencionando a lo largo de esta memoria, la implementación del RAG será con LLMs de código abierto. Esta decisión permite mantener la privacidad de los documentos que el modelo de lenguaje procesará, pero con la contra de que el modelo de lo normal no será tan capaz como otros de servicios de contrato mediante una suscripción de pago. También se debe tener constancia de que cuando se mencionan los recursos de los que dispone la empresa, se incluye el tener a disposición un entorno local de GPU que soporte el tamaño de un gran modelo de lenguaje junto al de un modelo de vectores de *embedding*, ya que, sin un entorno de este tipo, el tiempo de ejecución del proceso RAG puede ser inviable.

Es por esta razón que se pondrán en práctica técnicas de *retrieval* y *generation* que traten de optimizar el rendimiento de los grandes modelos de lenguaje de código abierto y tratar así de aproximarse a aquello que los de pago puedan ofrecer. Por ende, este proyecto marca la diferencia con el estado del arte en que respetará la privacidad de la información que se le proporcione al proceso *Retrieval-Augmented Generation* a costa de una quizás peor capacidad de respuesta a corto plazo, ya que cada vez son más los modelos ejecutables en entornos locales que se acercan a aquellos de pago.

Entrando en detalle sobre la implementación, respecto al almacenamiento de código y el entorno de programación, se creó un repositorio de GitHub y un contenedor de Docker mediante terminales de Linux y Visual Studio Code. Todo el código se escribió en el lenguaje de programación Python. Mediante el repositorio de GitHub, se pudo dar acceso a otros miembros de la empresa al código escrito, mientras que el contenedor de Docker facilitaría y agilizaría la instalación de librerías y dependencias necesarias para la ejecución del RAG. Luego, también por medio de Linux y Visual Studio Code, se hacía una conexión a los servidores de la empresa para tener acceso a los grandes modelos de lenguaje descargados en los mismos y a la ejecución con GPU.

Una vez establecida toda la base en la que ejecutar código, se comenzó con la programación. Como se ha explicado en anteriores secciones, se recurrió a librerías de Python que facilitan la tarea de la implementación del RAG, salvando los detalles expuestos anteriormente. Entre ellas, ya mencionadas, LangChain y RAGAS, aunque faltarían añadir *Hugging Face* [17], Chroma [18] y Ollama [19]. Si bien también son dependientes de LangChain y dieron ligeros problemas de compatibilidad, para cuando ocurrió ya se tenía un mayor conocimiento de LangChain y se pudieron solucionar con rapidez. Hugging Face y Ollama son librerías similares que proporcionan acceso a modelos de *embedding* y a *Large Language Models*. En este caso, Hugging Face se usó para llamar a el modelo de *embeddings intfloat/multilingual-e5-large* [44], capaz de extraer vectores de *embedding* para textos de varios idiomas, entre ellos el español. Este modelo fue elegido ya que ha demostrado una notable capacidad al posicionarse

[17] https://huggingface.co/

[18] https://www.trychroma.com/

[19] https://ollama.com/

en un buen puesto en el *ranking* MTEB [20] de Hugging Face. Para cuando fue elegido, ocupaba el puesto número 21. Luego, en cuanto al LLM usado, se recurrió al modelo Llama 3 [21] a través a la librería Ollama, que permite descargar estos modelos en local y a diferencia de Hugging Face, que solo permite hacerlo con los modelos de *embedding*. Llama 3 es un LLM proporcionado por la empresa Meta el día 15 de mayo de 2024. El modelo ha resultado ser muy capaz al aparecer en el *ranking* LMSYS *Chatbot Arena Leaderboard* [22]. Cabe aclarar que Llama 3 ofrece dos versiones del modelo, una más potente que otra, pero que a su vez una es menos pesada que la otra. En este proyecto, por razones de ahorro de computación, se ha usado la versión ligera.

4.1 Flujo

En la Figura 18 podemos observar un diagrama con todo el proceso llevado a cabo para establecer lo mencionado hasta ahora sobre el entorno de ejecución, llamadas a librerías y descargas de modelos, además del orden de pruebas de técnicas de *retrieval* y de *generation*.

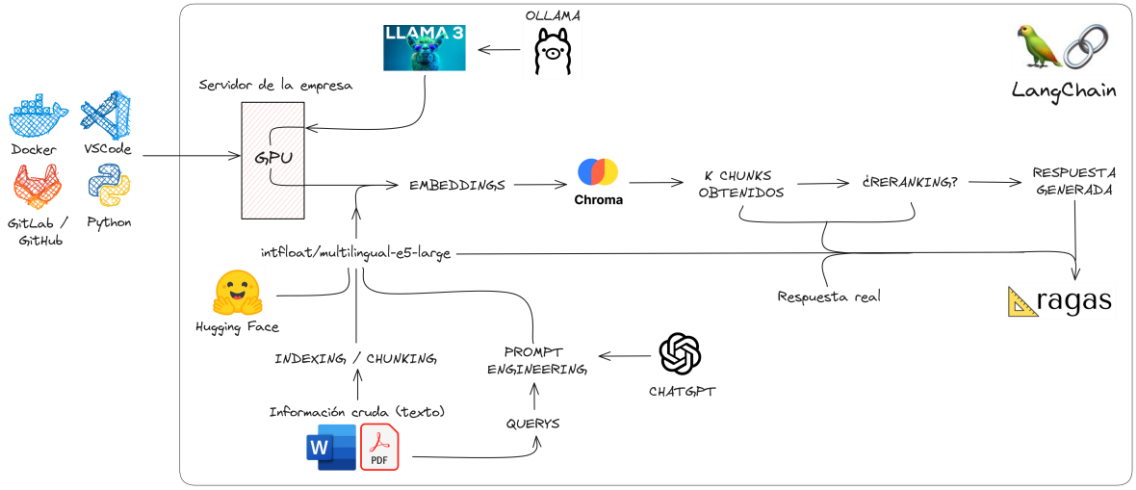

Figura 18: diagrama del proceso RAG implementado.

Observando la Figura 18, lo que corresponde al entorno de ejecución sería la parte izquierda y comienzo del diagrama. Siguiendo el flujo del diagrama, una vez establecido el entorno, nos conectaríamos al servicio de GPU dentro del servidor de la empresa, donde el gran modelo de lenguaje Llama 3 estaría descargado mediante Ollama y listo para usarse. Por otro lado, observando en la parte más baja de la Figura 18, veremos la representación de la elección de los documentos usados para hacer pruebas, por una parte , tenemos un .docx y por otra, un .pdf. En concreto, fueron elegidos esos ya que son los formatos que la empresa usa con sus documentos privados. Como debían ser públicos para el desarrollo de este proyecto de fin de grado, el .pdf elegido fue el de la explicación de la estructura de la memoria de un TFG de la ETSINF [23]. Luego, para el .docx se utilizó el texto de un blog del IBM sobre aplicaciones de la

[20] https://huggingface.co/spaces/mteb/leaderboard

[21] https://ai.meta.com/blog/meta-llama-3/

[22] https://chat.lmsys.org/

[23] https://www.inf.upv.es/www/etsinf/wp-content/uploads/2018/05/estructura.pdf

IA en medicina [24], del que a propósito se copió y pegó manualmente el texto dando el formato correcto a cada título, subtítulo y texto simple, ya que es algo que se hará en la empresa con este tipo de documentos.

Por dejar constancia, debe saberse que los temas que tratan los documentos son irrelevantes, aunque si ambos documentos trataban sobre temas diferentes, las pruebas serían más sencillas y realistas.

4.1.1 Preparación de datos

Volviendo a la Figura 18 en los iconos de Word y PDF, vemos que la flecha en dirección arriba lleva a la fase de *indexing*. Distintos métodos de los explicados en el apartado 2.2.2.1 han sido probados, en concreto, la separación recursiva, semántica, con agente y *Small-to-Big*, todas ellas combinadas con la separación de documentos específicos. Y en cuanto a método de la separación con agente y la *Small-to-Big*, respectivamente fueron probados el mencionado método de las proposiciones (mencionado en 2.2.2.1) y el de indexación de *chunks* de mayor tamaño mediante un resumen de los mismos.

Una vez la información cruda era tratada con cada una de las metodologías mencionadas, los resultados de la fase de *chunking/indexing* se entregaban al modelo *intfloat/multilingual-e5-large* de Hugging Face, que devolvería vectores de 1024 componentes. Toda ejecución de este modelo de *embeddings* también se llevaba a cabo por medio de la GPU de la empresa, esto se representa en la Figura 18 con la unión de las flechas de salida del servidor y del modelo de *embeddings*. Continuando por el diagrama, una vez obtenidas las representaciones vectoriales de los textos a tratar, se almacenaron en la base de datos vectorial Chroma. Esta base de datos de código abierto fue elegida, además de por su gran aceptación general en la comunidad, por recomendación en la empresa. Chroma permite crear instancias para conjuntos de *embeddings* que se almacenarán localmente, con la ventaja de mayor rapidez de procesamiento y de eficaces métodos de búsqueda de *chunks* ya implementados. Por tanto, en el siguiente paso en el diagrama, a la hora de obtener un número K de *chunks* a leer por el LLM, se usaba el método que proporcionaba Chroma. Más adelante en el flujo podemos ver escrito “¿RERANKING?”, esto se debe a que el método de Chroma de lo normal ya devolvía los mejores trozos de infomación deseada, haciendo que el uso de una reevaluación fuese innecesario. No obstante, la implementación de un método de *reranking* fue completada para tenerla a disposición en caso de ser necesario. Finalmente, tras la obtención de la información, Llama 3 se encargaría de procesarla y devolver una respuesta en base a ella.

4.1.2 Evaluación

Regresando a la Figura 18, falta hablar sobre el flujo que va por debajo de lo comentado hasta ahora. Volviendo a los documentos utilizados, con el fin de disponer de algo que permitiese de la evaluación de las respuestas generadas, se prepararó una batería de preguntas acerca de los documentos a responder. Esto significa cualquier otra pregunta habría sido válida. Las preguntas son:

[24] https://www.ibm.com/es-es/topics/artificial-intelligence-medicine

- *"Describe qué deben tener los agradecimientos"*
- *"Describe qué puede hacer la IA con imágenes médicas"*
- *"¿Qué debo hacer si el TFG se ha realizado en el ámbito de una práctica en empresa?"*
- *"Resume qué debe tener el diseño detallado"*

Y sus respuestas correctas serían:

- *"Opcional. Puede colocarse al principio o al final de la obra justo antes de los puntos complementarios finales como anexos o bibliografía. Normalmente los agradecimientos son a compañeros que pueden haber ayudado en la redacción o desarrollo del TFG, al tutor del TFG, el tutor en la empresa en la que se ha realizado el TFG, a la propia empresa en la que se han realizado las prácticas. Si procede, a compañeros que hayan desarrollado contenidos que no forman parte del TFG pero que le ayudan en suaspecto profesional, etc. Si el trabajo ha sido financiado bajo algún proyecto, beca o similar deberá incluirse de forma obligatoria."*
- *"La IA ya está desempeñando un papel destacado en el área de imágenes médicas. La investigación ha indicado que la IA impulsada por redes neuronales artificiales puede ser tan eficaz como los radiólogos humanos para detectar signos de cáncer de mama y otras afecciones. Además de ayudar a los médicos a detectar los primeros signos de la enfermedad, la IA también puede ayudar a que la asombrosa cantidad de imágenes médicas que los médicos deben controlar sea más manejable al detectar partes vitales del historial de un paciente y presentarles las imágenes relevantes."*
- *"Si se ha realizado el TFG en el ámbito de una práctica en empresa, y con el consentimiento del tutor de empresa, es esta sección o en otra similar (con un título más adecuado) se puede presentar la empresa, su trayectoria, historia, productos que desarrolla, trabajos más representativos, marco en el que se engloban las prácticas realizadas, resaltar la importancia del trabajo realizado para la estrategia de la empresa, del proyecto en el que se enmarca, etc. Si la empresa ya comercializa un producto software o hardware al cual complementa el TFG, habría que describirlo ampliamente dado que dicho producto es el que determinará posteriormente los requisitos a cumplir por el TFG. Tanto la empresa como el producto podrían aparecer en un par de apéndices si no se quiere hacer la memoria muy farragosa. En este último caso, podemos referenciar al producto/empresa sucintamente y referir a los apéndices para ampliar la información."*
- *"Obligatorio. En este segundo nivel de diseño se detalla el nivel propuesto previamente. Por ejemplo, en muchos TFG es aquí donde se presenta la lista de clases que describirán los diferentes elementos que interactuarán durante la ejecución del programa. Estas clases mantienen relaciones entre ellas incluyen unas a otras como atributos. Hay que indicar estas relaciones, la estructura de módulos que las contendrán, cómo se organizan en la estructura de directorios de la solución. Básicamente se trata de detallar la arquitectura software propuesta, el diseño de la base de datos, toda aquella información que aporte luz a la forma en la que se ha diseñado de forma detallada la solución del problema a resolver con el TFG."*

La razón de que no se prepararan más preguntas fue, de nuevo, el consumo de recursos de ejecución. Por otro lado, en el diagrama se puede observar la primera pregunta de la lista.

Una vez tenemos una pregunta que hacer al RAG, ya podemos trabajar con ella directamente. Sin embargo, como sabemos, la calidad de la respuesta puede mejorar si aplicamos ingeniería de *prompts*. Por ello, se puso en práctica la técnica de *query expansion* expuesta en el estado del arte. Como veremos en el siguiente capítulo, no se aplicaron más técnicas porque esa ya fue efectiva. Observando el diagrama, salta a la vista cómo el logo de OpenAI y el nombre de ChatGPT debajo de él, apuntan al apartado de *prompt engineering*. Como hemos ido repitiendo a lo largo de esta memoria, pretendemos usar solo LLMs de código abierto para mantener la privacidad de los documentos de una empresa, pero este breve uso de modelos externos se hizo siendo fiel a esa condición. El objetivo era obtener plantillas de preguntas a procesar por un LLM de código abierto para tener nuevas opciones en caso de que una pregunta no sea efectiva. Por ejemplo, para la consulta *"Describe qué deben tener los agradecimientos"* obtuvimos alternativas como: *"¿Cuáles son los elementos esenciales que deben incluirse en los agradecimientos?"* o *"¿Qué componentes son importantes en una sección de agradecimientos?"*, por lo que, en el ámbito privado de la empresa, solo habría que cambiar la palabra "agradecimientos" por la de interés. Para aplicar el *query expansion*, se usó el *prompt Zero-shot* siguiente:

> *Eres un asistente de modelos lingüísticos de inteligencia artificial. Tu tarea consiste en generar *N* versiones diferentes de la pregunta dada por el usuario para recuperar documentos relevantes de una base de datos vectorial. Al generar múltiples perspectivas de la pregunta dada por el usuario, tu objetivo es ayudarle a superar algunas de las limitaciones de la búsqueda por similitud basada en la distancia. Proporciona estas alternativas a la pregunta en diferentes líneas. Pregunta original: *Pregunta original**

El uso del servicio de OpenAI para trabajar la fase de ingeniería de *prompts* se debe a la ventaja de la reducción en el gasto de computación, además de una considerable facilidad de implementación en comparación a la de LangChain por causa de las desactualizaciones de sus métodos.

Para acabar la explicación de la Figura 18, cuando la *query*/*prompt* está lista, obtenemos sus *embeddings* con *intfloat/multilingual-e5-large*, para después pasarlos a la base de datos de Chroma elegida, que los comparará con sus *embeddings* de *chunks* para devolver un número K de ellos. Como ya sabemos, dependerá del caso si el método de *reranking* se llega a usar. Y, finalmente, se obtendrán los *embeddings* de la respuesta generada por Llama 3.

Para realizar la evaluación con RAGAS, necesitaremos los *embeddings* de cada una de las partes mencionadas: la pregunta, los *chunks*, la respuesta generada y, de añadido, los *embeddings* de la respuesta correcta para cada pregunta. Para lo último, deberemos buscar manualmente la respuesta, como hemos visto. Una vez obtenido el formato correcto para que la librería RAGAS pueda procesar cada uno de estos conjuntos de *embeddings*, podremos evaluar.

5.Experimentación

Esta sección está dedicada a la narración de las diferentes técnicas de mejora de RAG implementadas, principalmente de la fase de *retrieval*. Como veremos a lo largo de la sección, el RAG local desarrollado correspondería a un *Advanced* RAG, debido a la limitación que suponen los documentos internos hacia una mejora hasta un *Modular* RAG, además de que hubiese resultado innecesario, al menos, para este caso de uso.

Por otro lado, la evaluación RAGAS no ha resultado ser completamente fiable, aunque sí en parte. Como se mencionó, este método de evaluación (y otros) fueron desarrollados con modelos potentes de OpenAI, por lo que puede que el uso de modelos más débiles como el caso de la versión pequeña de Llama 3 influyan en la precisión de evaluación. No se ha podido encontrar estudios que evidencien esta idea, pero por las razones que iremos viendo en esta sección, es posible que así sea. Cuando decimos que RAGAS no ha sido fiable, se debe a los resultados que ha ido devolviendo para distintas pruebas de *retrieval*, que fueron la separación recursiva, semántica, con agente y *Small-to-Big*, como decíamos en 4.1.1. Estos resultados eran valores NaN o *Not a Number*, y, según dudas publicadas en el foro de GitHub de RAGAS [25], esto se debe a las ocasiones en las que el LLM no sabe responder a una pregunta dada. Sin embargo, en nuestro caso, durante el desarrollo de las distintas técnicas de recuperación, los valores NaN aparecían en diferentes ocasiones también. Además, para los casos en los que no se devolvían valores NaN, se obtenían valores inconsistentes a las respuestas otorgadas por el modelo. Como se puede ir intuyendo, estas inconsistencias eran encontradas mediante evaluación humana [45].

A pesar de la necesidad de la evaluación humana, no se dejó de lado RAGAS y, mediante las implementaciones de *retrieval* y su estudio de calidad de rendimiento, se llevó un estudio paralelo sobre su comportamiento en cada variante de *retrieval*. De este modo, se consiguió la identificación de los casos en los que se devolvían valores no numéricos y, por tanto, dar una mayor fiabilidad de sus evaluaciones.

Durante el desarrollo de proyecto, debido a los problemas expuestos en la sección 3, el plazo de realización de prácticas en la empresa expiró. Esto implicó una migración de los servicios de GPU de la empresa a otra alternativa independiente de la misma, en este caso se recurrió a los servicios de suscripción de GPU de Google Colab Pro+. Dicho cambio no supondría ningún problema respecto al código y la compatibilidad del mismo con la empresa, pero sí en cuanto al tiempo de computación de GPU disponible. Los 4 métodos de *indexing* usados, además de crecer en la escala de complejidad de ideación, sobre todo lo hacen la necesidad de recursos de computación, por lo que hubo que gestionarlos para llevar a cabo las pruebas.

5.1 Técnicas de *retrieval* y estudio del caso

Este subapartado está dedicado a la explicación de las técnicas de *retrieval* usadas, así como a el entendimiento del comportamiento del LLM Llama 3 pequeño durante el uso de las mismas.

[25] https://github.com/explodinggradients/ragas/issues/733

1. Separación recursiva: distintos números de caracteres fueron probados, pero el que se consideró adecuado en base a las respuestas devueltas fue 300. Aunque sea simple y pueda aumentar el riesgo de que la información necesaria se divida en diferentes *chunks*, resulta efectivo a la hora de combinar la información necesaria con el *prompt* del usuario. Es más, resulta tan efectivo, al menos para los documentos y preguntas usadas, que no necesita el uso de *reranking*, la búsqueda por similitud que proporciona Chroma ya es efectiva.

2. Separación semántica: adquirida por contribuciones de la comunidad de GitHub [26], la idea de este método de *chunking* consistirá en agrupar oraciones en función de su similitud semántica, medida con la similitud de coseno. El procedimiento es el siguiente:
   - Se obtienen todas las oraciones por separado.
   - Con el objetivo de iterar sobre cada una de las oraciones, se elegirá primero un número *n* de oraciones que se combinarán y compararán con la correspondiente oración de la iteración *i*. Por ejemplo, si *n*=2, la primera iteración comenzará en la oración número 2 y será comparada con la agrupación de las 2 anteriores oraciones. En el caso de uso, el número elegido fue 1.
   - Tras combinar las oraciones, se recurrirá al modelo *intfloat/multilingual-e5-large* para obtener sus vectores de *embedding* y, seguidamente, sus distancias de similitud por coseno.
   - Cuando se comparan las distancias de las oraciones combinadas, si superan un determinado umbral, se separarán y se crearán *chunks*. Dicho umbral se establece con un percentil de las distancias a elegir. En el caso de uso, el que mejor resultados daba era el percentil 80, dando un total de 72 *chunks*.

   De esta manera, los *chunks* creados deberían estar marcados por oraciones que hablen de un mismo tema. Observemos la Figura 19, en ella se puede observar un gráfico que marca la separación de chunks, si la distancia entre dos oraciones combinadas supera el umbral marcado con la línea roja, se creará un nuevo *chunk*.

[26] https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb

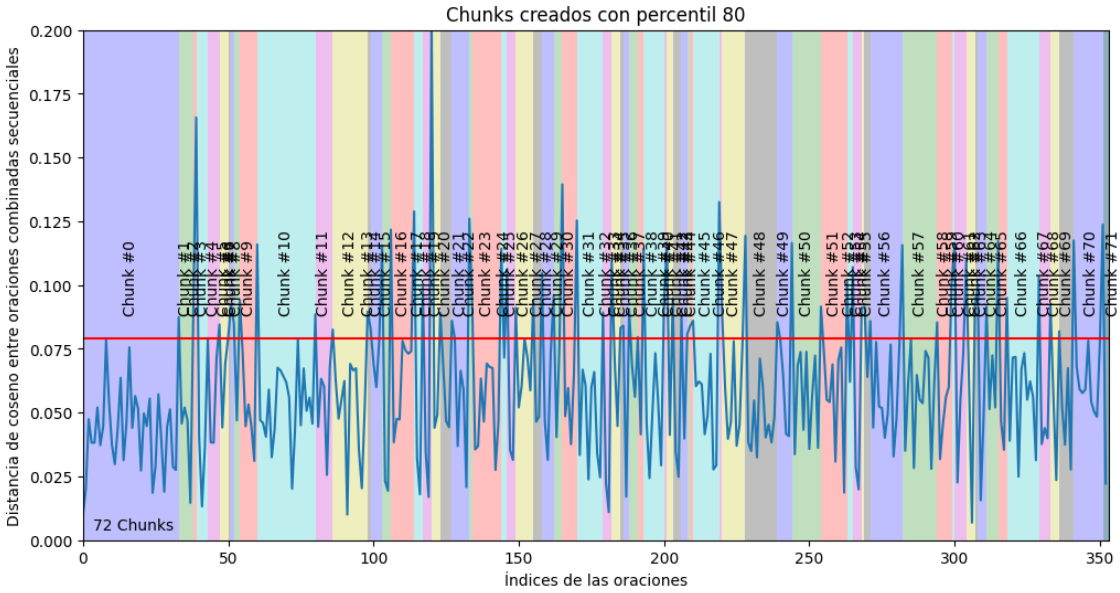

Figura 19: separación de chunks por distancia de coseno.

De nuevo, si miramos la Figura 19, se puede ver cómo hay *chunks* más grandes que otros. En varios casos esto se refleja en que los *chunks* pequeños son frases sueltas que han quedado separadas por fijar el percentil como 80. Como ejemplo, en la Figura 20, podemos ver cómo quedaron los *chunks* 12, 13 y 14.

```
Chunk #12
Las palabras clave ayudan a tipificar y clasificar el TFG en las bases de datos,
facilitando su búsqueda. Las palabras clave generalmente serán de 3 a 10 y podrán ser
compuestas. Prefacio o prólogo
Opcional.El prefacio o prólogo suele ser una introducción que acostumbra a destacar los
méritos, el valor del trabajo, o también a situar la obra en un contexto y unas circunstancias
determinantes. Esta parte es opcional y se incluirá en el caso de que se desee ampliar el
resumen. Índices y glosarios
Recomendable.Son colecciones de entradas que aparecen a lo largo de la memoria y que tienen
como misión facilitar la localización de información relevante para el lector, así como poner de
manifiesto la estructura general de la obra. Normalmente los índices suelen ir al principio de la
obra, justo antes del primer capítulo de introducción y después de los agradecimientos (si estos
se colocan al principio de la memoria). Índice de contenido
Obligatorio. Tabla de contenido donde se pueden observar las partes en que está dividido el
TFG. También se denomina índice de títulos. Como regla general, se recomienda mantener el
contenido de los índices de títulos en un nivel 3 y excepcionalmente ampliar al 4 si la extensión
y estructura de la obra lo requiere. Más allá del nivel 4 no tiene mucho sentido porque los índices
se hacen muy extensos y el nivel de detalle hace perder al lector la estructura del TFG. Índices generales, tablas e imágenes
Opcionales.


Chunk #13
Hay de varios tipos.


Chunk #14
Índice o glosario de términos. Lista ordenada de conceptos, nombres propios, etc.; que
aparecen en la obra, con las indicaciones necesarias para su localización. Ejemplo: Índice
alfabético. Índice onomástico o un índice de materias.
```

Figura 20: ejemplos de chunks de separación semántica de diferentes tamaños.

Pese a no aparentar ser un problema, al pasar cualquier pregunta por la base de datos de Chroma, los *chunks* de frases sueltas como el 13 siempre eran recuperados antes que los adecuados. Por ello, se programó una función que uniese a los *chunks* pequeños con el *chunk* más pequeño entre sus *chunks* anterior y posterior. Tras este prepoceso, el método de búsqueda por similitud de Chroma no era tan efectivo como en la separación recursiva, por lo que se añadió el paso de reranking por MRR. Finalmente, los contextos proporcionados de las preguntas a resolver eran adecuados y los resultados,

hasta el momento evaluados por comprensión humana, eran similares a los del método anterior.

Queda claro que es preferible usar la separación recursiva a la semántica si las respuestas dadas de la segunda dan resultados comparables. Sin embargo, puede que solo ocurra para los documentos utilizados en este caso de uso, por lo que no debería descartarse en pruebas distintas.

3. Separación con agente: este tipo de separación de texto fue la más exigente computacionalmente. El método elegido fue el de las propuestas o proposiciones mencionado en el estado del arte. Para ello, para separar inicialmente el texto de los dos documentos en párrafos, se usó separación recursiva en función de símbolos separadores. Estos son: "\n\n", "\n", ".", ",", entre otros. Obtenidos los párrafos, tras un largo proceso de ejecuciones e ingeniería de *prompts*, se obtuvieron las proposiciones para cada uno de los párrafos. El *prompt* usado fue el siguiente:

Descompón la "Entrada" en proposiciones claras y sencillas, asegurándose de que sean interpretables fuera de contexto.

1. Divide la frase compuesta en frases simples. Mantén la formulación original de la entrada siempre que sea posible.
2. Para cualquier entidad con nombre que vaya acompañada de información descriptiva adicional, separa esta información en su propia proposición distinta.
3. Descontextualiza la proposición añadiendo el modificador necesario a los sustantivos o frases enteras y sustituyendo los pronombres (por ejemplo, "eso", "él", "ella", "ellos", "esto", "aquello") por el nombre completo de las entidades a las que se refieren.
4. Presenta los resultados como una lista python de cadenas, es decir, cada grupo de cadenas debe ir entre los caracteres [ ], no uses asteriscos, ni flechas ni enumeraciones.
5. Responde únicamente con la lista de python, no incluyas ningún otro mensaje.
6. Las respuestas deben estar siempre en idioma español.

Te muestro un ejemplo para que entiendas cómo debes descomponer la "Entrada", pero no debes incluir nada del texto del ejemplo en tus respuestas:

Entrada:
Los primeros indicios de la existencia de la liebre de Pascua (Osterhase) fueron registrados en el suroeste de Alemania en 1678 por el profesor de medicina Georg Franck von Franckenau, pero fue desconocida en otras partes de Alemania hasta el siglo XVIII.
El erudito Richard Sermon escribe que "las liebres estaban en los jardines en primavera, lo que podría explicar el origen de los huevos de colores escondidos allí para los niños".

También existe una tradición europea en que las liebres ponían huevos, ya que el nido de una liebre y el de una avefría son muy parecidos, ambos se encuentran en praderas y se ven por primera vez en primavera.
En el siglo XIX, la influencia de las tarjetas, juguetes y libros de Pascua hizo popular a la liebre/conejo de Pascua en toda Europa.
Los inmigrantes alemanes exportaron la costumbre a Gran Bretaña y América, donde evolucionó hasta convertirse en el Conejo de Pascua.

Salida: ["La evidencia más temprana de la Liebre de Pascua fue registrada en el suroeste de Alemania en 1678 por Georg Franck von Franckenau", "Georg Franck von Franckenau era profesor de medicina", "Las pruebas de la Liebre de Pascua permanecieron desconocidas en otras partes de Alemania hasta el siglo XVIII",
"Richard Sermon era un erudito", "Richard Sermon escribe una hipótesis sobre la posible explicación de la conexión entre las liebres y la tradición durante la Pascua", "Las liebres se veían con frecuencia en los jardines en primavera",
"Las liebres pueden haber servido como una explicación conveniente para el origen de los huevos de colores escondidos en los jardines para los niños",
"Existe una tradición europea en que las liebres ponían huevos.", "El nido de una liebre y el nido de una avefría se parecen mucho.", "Tanto los nidos de liebre y de avefría se encuentran en los prados y se ven por primera vez en primavera.",
"En el siglo XIX, la influencia de las tarjetas de Pascua, los juguetes y los libros popularizaron la liebre y el conejo de Pascua en toda Europa.",
"Los inmigrantes alemanes exportaron la costumbre de la Liebre/Conejo de Pascua a Gran Bretaña y América.",
"La costumbre de la Liebre/Conejo de Pascua evolucionó hasta convertirse en el Conejo de Pascua en Gran Bretaña y América."]

Se trata de un ejemplo de *Few-shot Chain-of-Thought* y la plantilla inicial fue obtenida de un *hub* de LangChain [27] donde se publican *prompts* que pueda usar la comunidad. Dicha plantilla estaba en inglés, por lo que tuvo que traducirse al español y modificarse para que siempre respondiese como era deseado. Entre otros fallos, a veces Llama 3 respondía en inglés a pesar de estar el *prompt* en español, esto puede deberse a la mayor influencia del inglés en el entrenamiento de dicho LLM.

El resultado fueron *chunks* muy diferentes a los anteriores, siendo una versión reducida de los párrafos iniciales. Pese a que esto derivase en peores respuestas y en alucinaciones, sirvió para entender el porqué de que RAGAS devolviese valores NaN, al menos usando el LLM Llama 3 pequeño. Y es que, a pesar de que los contextos recuperados no fuesen buenos para generar respuestas parecidas a las respuestas correctas, RAGAS no devolvía valores no numéricos. Esto se debe a que los *chunks* de las proposiciones eran cortos y con afirmaciones concisas.

[27] https://smith.langchain.com/hub

Hasta ahora, con los anteriores métodos siempre se estaba intentado recuperar varios *chunks* de contexto para cada pregunta y, como sabemos, RAGAS usa *prompts* para obtener afirmaciones de los contextos. En otras palabras, RAGAS hace un proceso similar al que hicimos nosotros con este método de proposiciones. Por lo que, al pasar varios *chunks* de los métodos anteriores, Llama 3 tendría problemas para procesar todo el texto y daría problemas como el mencionado de las respuestas en otro idioma, llevando a afectar al funcionamiento de RAGAS. En el caso de los *chunks* de las proposiciones, aunque se obtuviesen varios *chunks*, sí eran manipulables debido a su menor tamaño.

4. *Small-to-Big*: implementado antes de percatarse de lo explicado en el anterior método, la técnica elegida de *Small-to-Big* elegida fue la de la creación de resúmenes de *chunks* grandes a partir del gran modelo de lenguaje. Una vez más, los resultados mantenían la línea de la separación recursiva y la semántica, por lo que se descartó al ser exigente a nivel de computación.

Tras esta explicación, podemos intuir que la técnica de separación de texto escogida para este caso de uso es la recursiva. Teniendo en cuenta lo aprendido durante el desarrollo de la separación con agente, se redujo el número de *chunks* a devolver por Chroma a 1. Puede sonar arriesgado, pero es rara la vez que el *chunk* devuelto no es el indicado, como veremos más adelante. Por otra parte, que la separación recursiva haya funcionado mejor en esta ocasión no significa que las demás sean peores, pero quizás sean más eficaces con otros LLMs más potentes o simplemente con otros casos de uso.

5.1.1 Inclusión de *prompt engineering*

Como se explicó en la sección de Propuesta, se añadió la práctica de *query expansion* al proceso RAG por medio de ChatGPT, pero con el fin de encontrar plantillas de preguntas reutilizables. Como veremos es este subapartado, en ocasiones, Llama 3 devuelve respuestas indicando que no sabe responder a pesar de que el contexto recuperado sea el correcto, pero al dar una misma pregunta con distinta formulación, sí responde correctamente.

Respecto a las pruebas de la ingeniería de *prompts*, para cada una de las preguntas de la batería de preguntas mostrada también en la anterior sección, se generaron 3 preguntas adicionales, dando un total de 16 preguntas sobre 4 respuestas correctas. Como paso adicional de depuración de respuestas, por si una de las preguntas era del tipo “No sé la respuesta” se marcó un mínimo de palabras por respuesta de 15 para no incluirla en la evaluación de RAGAS. Esto queda abierto a una clara mejora, pero para este caso fue útil y suficiente.

Tras ese proceso, el RAG siempre devolvía al menos una vez una buena respuesta. Veamos la Tabla 1, donde se enseña un ejemplo de ejecución con las preguntas, los contextos obtenidos, las respuestas generadas y las respuestas correctas. Por facilitar la visualización de la Tabla 1, se han seleccionado los casos en las que las respuestas reflejan la variabilidad de las respuestas, sean cercanas a la correcta o no.

Preguntas	Contextos	Respuestas generadas	Respuestas correctas
Describe qué deben tener los agradecimientos	'Autor/a y año de creación del TFG\nObligatorio. Incluye al autor con su nombre completo. (...) Agradecimientos\nOpcional.Puede colocarse al principio o al final de la obra justo antes de los puntos\ncomplementarios finales como anexos o bibliografía. Normalmente los agradecimientos son a compañeros que pueden haber ayudado en la\nredacción o desarrollo del TFG, al tutor del TFG, el tutor en la empresa en la que se ha realizado\nel TFG, a la propia empresa en la que se han realizado las prácticas. Si procede, a compañeros\nque hayan desarrollado contenidos que no forman parte del TFG pero que le ayudan en suaspecto profesional, etc. Si el trabajo ha sido financiado bajo algún proyecto, beca o similar\ndeberá incluirse de forma obligatoria. Resumen\nObligatorio. Esta parte será pública y seguramente la más leída de todo el TFG. Es mejor\nescribirlo cuando el TFG esté prácticamente finalizado, pero no en el último minuto. Debe ser\nuna síntesis del trabajo y, por lo tanto, deben aparecer aquí todos los elementos fundamentales\ndel TFG.'	'Pueden colocarse al principio o al final de la obra justo antes de los puntos complementarios finales como anexos o bibliografía. Normalmente los agradecimientos son a compañeros que pueden haber ayudado en la redacción o desarrollo del TFG, al tutor del TFG, el tutor en la empresa en la que se ha realizado el TFG, a la propia empresa en la que se han realizado las prácticas. Si procede, a compañeros que hayan desarrollado contenidos que no forman parte del TFG pero que le ayudan en su aspecto profesional, etc.'	'Opcional. Puede colocarse al principio o al final de la obra justo antes de los puntos complementarios finales como anexos o bibliografía. Normalmente los agradecimientos son a compañeros que pueden haber ayudado en la redacción o desarrollo del TFG, al tutor del TFG, el tutor en la empresa en la que se ha realizado el TFG, a la propia empresa en la que se han realizado las prácticas. Si procede, a compañeros que hayan desarrollado contenidos que no forman parte del TFG pero que le ayudan en suaspecto profesional, etc. Si el trabajo ha sido financiado bajo algún proyecto, beca o similar deberá incluirse de forma obligatoria.'
¿Cuáles son los elementos esenciales que deben incluirse en los agradecimientos?		'Pueden colocarse al principio o al final de la obra justo antes de los puntos complementarios finales como anexos o bibliografía. Normalmente los agradecimientos son a compañeros que pueden haber ayudado en la redacción o desarrollo del TFG, al tutor del TFG, el tutor en la empresa en la que se ha realizado el TFG, a la propia empresa en la que se han realizado las prácticas. Si procede, a compañeros que hayan desarrollado contenidos que no forman parte del TFG pero que le ayudan en su aspecto profesional, etc. Si el trabajo ha sido financiado bajo algún proyecto, beca o similar debe incluirse de forma obligatoria.'	
¿Qué componentes son importantes en una sección de agradecimientos?	'No es\nnecesario detallar el contenido más allá del segundo nivel de índice por no ser tedioso ni repetir\naspectos que se verán más adelante. Básicamente es un párrafo grande que justifica y describe\nlos capítulos para que el lector sepa qué va a encontrar en cada capítulo del TFG. En esta sección se debe incluir información del contenido de los anexos (...)	'Colaboraciones Obligatorio (si se da el caso). En el caso de que el trabajo sea realizado junto con otros compañeros, hay que mencionar al equipo completo del trabajo indicando cada uno lo que ha realizado y en concreto, destacar qué parte ha realizado el alumno.'	
Explique las aplicaciones de la inteligencia artificial en el análisis de imágenes médicas	'¿Qué es la inteligencia artificial en la medicina? La inteligencia artificial en medicina es el uso de modelos de aprendizaje automático para buscar datos médicos y descubrir conocimientos que ayuden a mejorar los resultados de salud y las experiencias de los pacientes. (...) En las imágenes médicas, las herramientas de inteligencia artificial se están utilizando para analizar tomografías computarizadas, rayos X, resonancias magnéticas y otras imágenes en busca de lesiones u otros hallazgos que un radiólogo humano podría pasar por alto. (...)	'Las herramientas de inteligencia artificial se están utilizando para analizar tomografías computarizadas, rayos X, resonancias magnéticas y otras imágenes en busca de lesiones u otros hallazgos que un radiólogo humano podría pasar por alto.'	'La IA ya está desempeñando un papel destacado en el área de imágenes médicas. La investigación ha indicado que la IA impulsada por redes neuronales artificiales puede ser tan eficaz como los radiólogos humanos para detectar signos de cáncer de mama y otras afecciones. Además de ayudar a los médicos a detectar los primeros signos de la enfermedad, la IA también puede ayudar a que la asombrosa cantidad de imágenes médicas que los médicos deben controlar sea más manejable al detectar partes vitales del historial de un paciente y presentarles las imágenes relevantes.',
¿Qué pasos debo seguir si el TFG está relacionado con una práctica en empresa?	'La validación de las afirmaciones que vierte el alumno en esta parte mediante citas\nexternas garantiza que el alumno no se está inventando nada de lo que está diciendo y que, en\nla realidad, las cosas funcionan como se está indicando. (...) Si se ha realizado el TFG en el ámbito de unapráctica en empresa, y con el consentimiento del\ntutor de empresa, es esta sección o en otra similar (con un título más adecuado) se puede\npresentar la empresa, su trayectoria, historia, productos que desarrolla, trabajos más\nrepresentativos, marco en el que se engloban las prácticas realizadas, resaltar la importancia del\ntrabajo realizado para la estrategia de la empresa, del proyecto	Si el TFG está relacionado con una práctica en empresa, es importante dar indicios no solo propios sino fundamentalmente externos de que lo que se dice está validado y es real. Puede presentar la empresa, su trayectoria, historia, productos que desarrolla, trabajos más representativos, marco en el que se engloban las prácticas realizadas, resaltar la importancia del trabajo realizado para la estrategia de la empresa, del proyecto en el que se enmarca, etc.'	'Si se ha realizado el TFG en el ámbito de una práctica en empresa, y con el consentimiento del tutor de empresa, es esta sección o en otra similar (con un título más adecuado) se puede presentar la empresa, su trayectoria, historia, productos que desarrolla, trabajos más representativos, marco en el que se engloban las prácticas realizadas, resaltar la importancia del trabajo realizado para la estrategia de la

¿Qué procedimientos se recomiendan para un TFG basado en prácticas empresariales?	en el que se enmarca, etc. Si la empresa ya comercializa un producto software o hardware al cual complementa el TFG,\nhabría que describirlo ampliamente dado que dicho producto es el que determinará\nposteriormente los requisitos a cumplir por el TFG. Tanto la empresa como el producto podrían aparecer en un par de apéndices si no se quiere\nhacer la memoria muy farragosa. En este último caso, podemos referenciar al\nproducto/empresa sucintamente y referir a los apéndices para ampliar la información (...).	'La validación de las afirmaciones que vierte el alumno en esta parte mediante citas externas garantiza que el alumno no se está inventando nada de lo que está diciendo y que, en la realidad, las cosas funcionan como se está indicando. Es importante dar indicios no sólo propios sino fundamentalmente externos de que lo que se dice está validado y es real.'	empresa, del proyecto en el que se enmarca, etc. Si la empresa ya comercializa un producto software o hardware al cual complementa el TFG, habría que describirlo ampliamente dado que dicho producto es el que determinará posteriormente los requisitos a cumplir por el TFG. Tanto la empresa como el producto podrían aparecer en un par de apéndices si no se quiere hacer la memoria muy farragosa. En este último caso, podemos referenciar al producto/empresa sucintamente y referir a los apéndices para ampliar la información.'
¿Cuáles son las acciones a tomar si el TFG se ha desarrollado durante una práctica profesional en una empresa?		'Si el TFG se ha desarrollado durante una práctica profesional en una empresa, es esta sección o en otra similar (con un título más adecuado) se puede presentar la empresa, su trayectoria, historia, productos que desarrolla, trabajos más representativos, marco en el que se engloban las prácticas realizadas, resaltar la importancia del trabajo realizado para la estrategia de la empresa, del proyecto en el que se enmarca, etc.'	
Resume qué debe tener el diseño detallado.	'Para muchos TFG es aquí donde se elige y explica la\narquitectura software o patrón arquitectónico elegido, se identificarán los componentes\nesenciales de la arquitectura y cómo se relacionan o interactúan unos con otros. (...) Diseño Detallado\nObligatorio. En este segundo nivel de diseño se detalla el nivel propuesto previamente. Por\nejemplo, en muchos TFG es aquí donde se presenta la lista de clases que describirán los\ndiferentes elementos que interactuarán durante la ejecución del programa. Estas clases\nmantienen relaciones entre ellas incluyen unas a otras como atributos. Hay que indicar estas\nrelaciones, la estructura de módulos que las contendrán, cómo se organizan en la estructura de\ndirectorios de la solución. Básicamente se trata de detallar la arquitectura software propuesta,\nel diseño de la base de datos, toda aquella información que aporte luz a la forma en la que se\nha diseñado de forma detallada la solución del problema a resolver con el TFG. (...)	'En este nivel de diseño se detalla la arquitectura software propuesta, incluyendo la lista de clases que describirán los diferentes elementos que interactuarán durante la ejecución del programa. Se deben indicar las relaciones entre ellas, incluyendo atras como atributos. También se debe mostrar la estructura de módulos que las contendrán y cómo se organizan en la estructura de directorios de la solución.'	'Obligatorio. En este segundo nivel de diseño se detalla el nivel propuesto previamente. Por ejemplo, en muchos TFG es aquí donde se presenta la lista de clases que describirán los diferentes elementos que interactuarán durante la ejecución del programa. Estas clases mantienen relaciones entre ellas incluyen unas a otras como atributos. Hay que indicar estas relaciones, la estructura de módulos que las contendrán, cómo se organizan en la estructura de directorios de la solución. Básicamente se trata de detallar la arquitectura software propuesta, el diseño de la base de datos, toda aquella información que aporte luz a la forma en la que se ha diseñado de forma detallada la solución del problema a resolver con el TFG.',
Describe los aspectos clave que deben estar presentes en el diseño detallado.		'Para muchos TFG es aquí donde se elige y explica la arquitectura software o patrón arquitectónico elegido, se identificarán los componentes esenciales de la arquitectura y cómo se relacionan o interactúan unos con otros. Se presentará mediante un gráfico o diagrama de bloques las piezas o componentes esenciales de la arquitectura.'	

Tabla 1: ejemplos de ejecuciones del RAG para las distintas preguntas. Las celdas con distintos colores corresponden a cada grupo de preguntas similares. Aquellas celdas de contextos o respuestas que recuperaron o generaron el mismo texto han sido combinadas. Por último, aquellas celdas con texto morado indican fallo.

En la Tabla 1, si observamos la respuesta correcta de la IA generativa en la medicina y la respuesta generada, se aprecia que no son similares, pero el contexto recuperado podría estar dando una respuesta buena. Por esta razón no se ha coloreado de morado ningún elemento, ya que la respuesta correcta elegida manualmente podría ser modificada.

Pasemos ahora a las evaluaciones de RAGAS. Como se ha ido adelantando, a pesar de encontrar el motivo de que se devolvieran valores NaN y solucionarlo, en parte las evaluaciones no son fiables del todo; para los casos en los que las respuestas son claramente incorrectas, las medidas suelen ser bajas, por debajo de 0.5, aunque para aquellos casos en los que las respuestas son buenas y parecidas, en ocasiones el valor de alguna de las variables se diferencia mucho. En la Figura 21 podemos observar las medidas de RAGAS obtenidas para las 3 primeras y las 2 últimas preguntas mostradas en la Tabla 1.

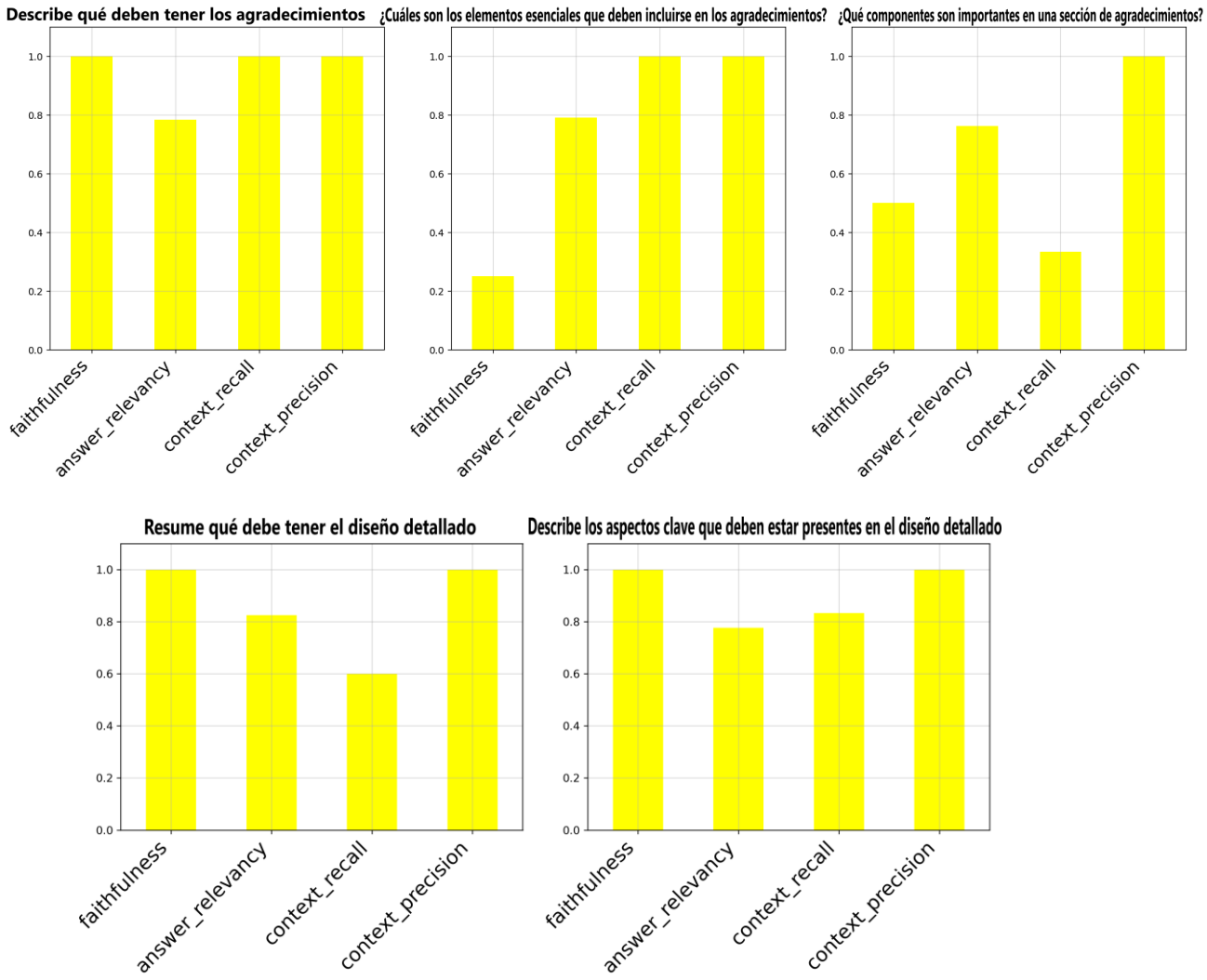

Figura 21: resultados de RAGAS en dos grupos de preguntas

En la fila superior de la Figura 21 podemos observar que se da peor puntuación a la respuesta mala, por lo que es un resultado aceptable. Sin embargo, en la fila de abajo hay un problema: en la pregunta de la izquierda, Llama 3 supo reescribir la información del contexto de forma que el mensaje es el mismo, de hecho, está incluso mejor adaptado para responder a la respuesta, el problema de haber reescrito la información es que afecta a la puntuación de RAGAS. Por otra parte, la pregunta de la derecha da una mala respuesta, pero mantiene exacto el contexto y da buena puntuación.

Por tanto, tras esta sección de experimentación hemos visto que no han sido necesarias usar técnicas complejas para hacerse con resultados buenos. Por otra parte, la automatización la evaluación de un RAG en español, al menos para uno con un LLM del tamaño de Llama 3-8B, necesitaría un propio proyecto para conseguirse.

6. Conclusiones

En este proyecto de fin de grado presentamos la teoría necesaria para lograr un entendimiento del funcionamiento de los *Large Language Models* y la práctica de *Retrieval-Augmented Generation*, además de cómo escalar la segunda mediante técnicas de *retrieval* y *generation*. Al tratarse de un proyecto empresarial surgido como una rama dentro de otro proyecto mayor de investigación sobre la inteligencia artificial generativa, el propio diseño del proyecto y sus correspondientes problemas se fueron hallando durante su desarrollo. La reciente aparición del sistema RAG y el constante crecimiento del campo de la inteligencia artificial generativa hacen difícil, en general para la comunidad, la creación de documentación adecuada y que vaya a la par en cuanto a ritmo de desarrollo. En otras palabras, la falta y desactualización de información sobre las implantaciones de RAGs locales fueron el principal problema a resolver en este trabajo. Para dar una solución, se decidió no ser completamente dependientes de la librería Langchain, sino hacer uso de los elementos no volátiles y esenciales de la misma, para combinarlos con los recursos descargables que proporcionan Hugging Face y Ollama, todas librerías de código abierto, además de la programación métodos propios para las técnicas de *retrieval* probadas. De esta forma, se ha completado una implementación de un RAG que no sufrirá errores por actualizaciones de librerías en el futuro y que podrá ser utilizado en la empresa.

Asimismo, un posible error durante la realización del trabajo sería el no haber comenzado a dar pinceladas a la implementación práctica desde antes, ya que en un inicio el foco fue la obtención de toda la teoría posible, por lo que no se tenía en cuenta las complicaciones respecto al código. Por ello, si bien por limitaciones de tiempo durante el periodo de prácticas de empresa en el que se desarrollaba el proyecto no se pudo llevar a cabo una experimentación con aún más distintas técnicas de *retrieval* y *generation*, sí se ha conseguido una combinación que brinda buenos resultados para el gran modelo de lenguaje local Llama 3, todo ello focalizado en el idioma español.

La realización de este trabajo ha brindado al desarrollador conocimiento sobre las aplicaciones que pueden tener los LLMs a un nivel técnico, más allá de las simples preguntas que se pueden hacer a un chatbot en el día a día. Además, ha supuesto una ventaja para ser competente y estar actualizado sobre el campo de la inteligencia artificial, del que se tiene interés. Por otro lado, se ha obtenido experiencia con casos de la vida real en servidores con GPUs y en cómo conectarse a ellos mediante Linux y Python.

6.1 Relación del trabajo desarrollado con los estudios cursados

La relación de este trabajo con los estudios del grado es total. Sobre todo, con las asignaturas: Programación (14003), Infraestructura para el procesamiento de datos (14016), Modelos descriptivos y predictivos II (14011) y Lenguaje natural y recuperación de la información (14029). En concreto, a nivel personal, este proyecto se ha sentido como una expansión de la última asignatura mencionada, ya que todos lo conceptos teóricos de los que parten los LLMs se dieron en ella, pero por aquel entonces no se habían popularizado; estaban a punto de hacerlo.

Otro punto son las competencias transversales empleadas en a lo largo de este proyecto, estas son, principalmente: Análisis y resolución de problemas (CT-03), para

dar soluciones a las trabas en la fase práctica, Conocimiento de problemas contemporáneos (CT-11) junto con Aprendizaje permanente (CT-12), para contar con ideas premeditadas sobre las posibles debilidades de Llama 3 y Responsabilidad y toma de decisiones (CT-18) para encontrar la razón de que la evaluación con RAGAS fuese nefasta en un inicio.

En otras palabras, la realización de este proyecto se ha beneficiado no solo de las *hard skills* adquiridas a lo largo del grado, sino también de las *soft*.

7.Trabajos futuros

Como se ha mencionado a lo largo de este trabajo, hubo que reducir la cantidad de técnicas *retrieval* y *generation* que trataban de probarse en un inicio debido a la complejidad de la implementación del RAG. Es por esto que en el futuro podrían ponerse en práctica esas pruebas adicionales.

Por los resultados mostrados en la sección de experimentación, hemos entendido que el RAG todavía necesita de evaluación humana para medir sus respuestas, ya que RAGAS no llega a ser confiable al completo. Es por esto que, unido a una mayor experimentación, se podrían adaptar las plantillas de *prompts* de las medidas de RAGAS (apartado 2.2.2.5) al español y al LLM que se esté usando localmente para tratar de mejorar las evaluaciones, ya que recordemos que esas plantillas están diseñadas para el modelo GPT 3.5 de OpenAI. Obteniendo una forma de evaluación fiable, se eliminaría la dependencia humana de la evaluación del RAG y, por tanto, las experimentaciones se agilizarían. Cabe mencionar que dicha adaptación necesitaría programarse desde cero, casi con total seguridad.

8. Bibliografía

1. Peña, D. (2014). *Fundamentos de estadística*. Alianza editorial.
2. Peña, D. (2013). Análisis de datos multivariantes. Cambridge: McGraw-Hill España.
3. Banh, L., Strobel, G. Generative artificial intelligence. Electron Markets 33, 63 (2023). https://doi.org/10.1007/s12525-023-00680-1
4. Bruce Croft, Donald Metzler, and Trevor Strohman. 2009. Search Engines: Information Retrieval in Practice (1st. ed.). Addison-Wesley Publishing Company, USA.
5. Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., ... & Wen, J. R. (2023). A survey of large language models. *arXiv preprint arXiv:2303.18223*.
6. Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A Survey on Evaluation of Large Language Models. ACM Trans. Intell. Syst. Technol. 15, 3, Article 39 (June 2024), 45 pages. https://doi.org/10.1145/3641289
7. Michael A. Nielsen, "Neural Networks and Deep Learning". Determination Press, 2015.
8. Li, J., Chen, X., Hovy, E., & Jurafsky, D. (2015). Visualizing and understanding neural models in NLP. arXiv preprint arXiv:1506.01066.
9. Jurafsky, D. (2000). Speech and language processing. https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf
10. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
11. Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., ... & Sifre, L. (2022). Training compute-optimal large language models. arXiv preprint arXiv:2203.15556.
12. Medsker, L., & Jain, L. C. (Eds.). (1999). Recurrent neural networks: design and applications. CRC press.
13. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
14. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
15. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

16. Li, H., Su, Y., Cai, D., Wang, Y., & Liu, L. (2022). A survey on retrieval-augmented text generation. arXiv preprint arXiv:2202.01110.

17. Wu, T., Luo, L., Li, Y. F., Pan, S., Vu, T. T., & Haffari, G. (2024). Continual learning for large language models: A survey. arXiv preprint arXiv:2402.01364.

18. Huang, Y., & Huang, J. (2024). A Survey on Retrieval-Augmented Text Generation for Large Language Models. arXiv preprint arXiv:2404.10981.

19. I. ILIN, "Advanced rag techniques: an ilustrated overview," https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6

20. Yunfan Gao, "Modular RAG and RAG Flow: Part I," https://medium.com/@yufan1602/modular-rag-and-rag-flow-part-%E2%85%B0-e69b32dc13a3

21. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... & Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997.

22. Eldan, R., & Russinovich, M. (2023). Who's Harry Potter? Approximate Unlearning in LLMs. arXiv preprint arXiv:2310.02238.

23. Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the middle: How language models use long contexts. Transactions of the Association for Computational Linguistics, 12, 157-173.

24. Chen, T., Wang, H., Chen, S., Yu, W., Ma, K., Zhao, X., ... & Zhang, H. (2023). Dense x retrieval: What retrieval granularity should we use?. arXiv preprint arXiv:2312.06648.

25. Sophia Yang, Ph.D., "Advanced RAG 01: Small-to-Big Retrieval. Child-Parent RecursiveRetriever and Sentence Window Retrieval with LlamaIndex" https://towardsdatascience.com/advanced-rag-01-small-to-big-retrieval-172181b396d4

26. Wang, Y., Lipka, N., Rossi, R. A., Siu, A., Zhang, R., & Derr, T. (2024, March). Knowledge graph prompting for multi-document question answering. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 38, No. 17, pp. 19206-19214).

27. Ma, X., Gong, Y., He, P., Zhao, H., & Duan, N. (2023). Query rewriting for retrieval-augmented large language models. arXiv preprint arXiv:2305.14283.

28. Zhang, Z., Zhang, A., Li, M., & Smola, A. (2022). Automatic chain of thought prompting in large language models. arXiv preprint arXiv:2210.03493.

29. Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. arXiv preprint arXiv:2402.07927.

30. Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., ... & Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171.

31. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., & Narasimhan, K. (2024). Tree of thoughts: Deliberate problem solving with large language models. Advances in Neural Information Processing Systems, 36.

32. Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., ... & Chi, E. (2022). Least-to-most prompting enables complex reasoning in large language models. arXiv preprint arXiv:2205.10625.

33. Dhuliawala, S., Komeili, M., Xu, J., Raileanu, R., Li, X., Celikyilmaz, A., & Weston, J. (2023). Chain-of-verification reduces hallucination in large language models. arXiv preprint arXiv:2309.11495.

34. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629.

35. Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. Foundations and Trends® in Information Retrieval, 3(4), 333-389.

36. Aizawa, A. (2003). An information-theoretic perspective of tf–idf measures. Information Processing & Management, 39(1), 45-65.

37. Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

38. Zhuang, S., Liu, B., Koopman, B., & Zuccon, G. (2023). Open-source large language models are strong zero-shot query likelihood models for document ranking. arXiv preprint arXiv: Advanced RAG 04: Re-ranking

39. Florian June, “Advanced RAG 04: Re-ranking. From Principles to Two Mainstream Implementation Methods” https://pub.towardsai.net/advanced-rag-04-re-ranking-85f6ae8170b1

40. Es, S., James, J., Espinosa-Anke, L., & Schockaert, S. (2023). Ragas: Automated evaluation of retrieval augmented generation. arXiv preprint arXiv:2309.15217.

41. Saad-Falcon, J., Khattab, O., Potts, C., & Zaharia, M. (2023). Ares: An automated evaluation framework for retrieval-augmented generation systems. arXiv preprint arXiv:2311.09476.

42. Finardi, P., Avila, L., Castaldoni, R., Gengo, P., Larcher, C., Piau, M., ... & Caridá, V. (2024). The Chronicles of RAG: The Retriever, the Chunk and the Generator. arXiv preprint arXiv:2401.07883.

43. Topsakal, O., & Akinci, T. C. (2023, July). Creating large language model applications utilizing langchain: A primer on developing llm apps fast. In

International Conference on Applied Engineering and Natural Sciences (Vol. 1, No. 1, pp. 1050-1056).

44. Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., & Wei, F. (2024). Multilingual e5 text embeddings: A technical report. arXiv preprint arXiv:2402.05672.

45. Abhinav Kimothi, (2024). A Simple Guide to Retrieval Augmented Generation. Manning.

9.Anexo

OBJETIVOS DE DESARROLLO SOSTENIBLE

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. **Fin de la pobreza.**			X	
ODS 2. **Hambre cero.**			X	
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**	X			
ODS 5. **Igualdad de género.**		X		
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía sostenible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructura.**	X			
ODS 10. **Reducción de las desigualdades.**		X		
ODS 11. **Ciudades y comunidades sostenibles.**			X	
ODS 12. **Producción y consumo responsables.**		X		
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**	X			
ODS 17. **Alianzas para lograr los objetivos.**	X			

Depende de cómo se enfoque, el presente trabajo puede tener una considerable aportación a los ODS. Considerando que el proyecto pertenece a una línea de investigación dentro de una empresa, el RAG local tendría una aportación a los objetivos si su implementación se llegase a comercializar con otras empresas. Ese

último aspecto es importante para razonar las aportaciones que tendría en los ODS. Por ello, consideramos que tendría altas contribuciones en los objetivos 3, 4, 9, 16 y 17.

- **ODS 3. Salud y bienestar:**
  La implementación de un RAG puede ser muy beneficiosa en la salud, al ofrecer respuestas precisas y actualizadas basadas en datos médicos, investigaciones científicas y políticas de salud pública. Esto podría ayudar a los profesionales médicos y al público a acceder a información actualizada sobre tratamientos, prevención de enfermedades y gestión de la salud, mejorando la atención y los resultados sanitarios.

- **ODS 4. Educación de calidad:**
  Un sistema RAG puede proporcionar acceso a contenido educativo actualizado y relevante para estudiantes, maestros y comunidades en regiones con menos recursos educativos. Esto puede mejorar significativamente la calidad y accesibilidad de la educación al facilitar el aprendizaje personalizado, basado en información confiable y diversa.

- **ODS 9. Industria, innovación e infraestructura:**
  Este proyecto puede tener un alto impacto en la innovación, al proporcionar información clave a investigadores, ingenieros y empresarios para el desarrollo de nuevas tecnologías y soluciones industriales. Además, al ser de código abierto, puede facilitar la colaboración y el acceso a recursos en comunidades tecnológicas globales.

- **ODS 16. Paz, justicia e instituciones sólidas:**
  Un RAG puede ser usado para mejorar el acceso a información legal, fomentar la transparencia y apoyar la justicia, lo que podría fortalecer las instituciones. Por otra parte, en un futuro, un RAG potente entrenado en derecho y un caso legal específico podría funcionar como herramienta de neutralidad en juzgados.

- **ODS 17. Alianzas para lograr los objetivos:**
  Como sistema de código abierto, un RAG podría facilitar la colaboración y el intercambio de conocimiento entre diferentes actores a nivel global, desde investigadores hasta organizaciones no gubernamentales y gobiernos, similar a lo que decíamos en ODS 9. Esto puede ser clave para impulsar alianzas en la consecución de los ODS.

En definitiva, el desarrollo de un RAG local con fin de uso en empresas busca agilizar la comunicación y aprendizaje entre trabajadores. Con la organización y entrenamiento del sistema adecuados, puede tratarse de una herramienta muy potente respecto a los Objetivos de Desarrollo Sostenible.