Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Introducción 1

En los últimos años, los grandes modelos del lenguaje (LLMs) han experimentado un avance significativo, revolucionando diversos campos relacionados con el procesamiento del lenguaje natural (NLP). Estos modelos, entrenados con grandes cantidades de datos, han demostrado una capacidad impresionante para generar texto coherente y relevante en una variedad de contextos. Sin embargo, uno de los desafíos persistentes en el uso de LLMs es su capacidad para manejar información específica y detallada de manera eficiente y precisa.

Una metodología emergente que aborda este desafío es la generación aumentada por recuperación (retrieval augmented generation, RAG). Los sistemas RAG combinan las capacidades generativas de los LLMs con técnicas de recuperación de información, permitiendo a los modelos acceder a bases de datos externas para mejorar la precisión y relevancia de las respuestas generadas. Este enfoque es particularmente útil ya que enriquece el contexto de los LLMs de manera que son capaces de contextualizar mejor el texto que se desea generar.

En este trabajo, se exploran las metodologías de chunkerización y retrieval en el contexto de la Constitución Española. La chunkerización es un proceso crítico que implica dividir un documento en fragmentos manejables, o chunks, que pueden ser más fácilmente procesados por los modelos de lenguaje. Se ha desarrollado un splitter especializado para chunkerizar los artículos de la Constitución Española de manera precisa y coherente. Este trabajo incluye una comparación detallada de diversas metodologías de chunkerización, evaluadas mediante métricas específicas para determinar la eficacia y eficiencia de cada una.

Además, se explicarán brevemente los conceptos fundamentales relacionados con los LLMs y los sistemas RAG, incluyendo embeddings y almacenes vectoriales (vector stores). Estos conceptos son esenciales para comprender cómo se integran las técnicas de recuperación de información con los modelos generativos, mejorando significativamente su rendimiento en tareas específicas.

En resumen, este trabajo busca proporcionar una visión comprensiva de cómo los sistemas RAG pueden ser aplicados de manera efectiva. Para ello, se compararán diferentes metodologías de chunkerización y se culminará con la creación de un agente con la capacidad de contextualizar las preguntas usando la constitución española.

Objetivos

2

El principal objetivo de este trabajo es desarrollar un sistema RAG específicamente diseñado para interactuar con la Constitución Española. Este sistema se centrará en la creación de un agente basado en LLMs, capaz de responder a cualquier pregunta relacionada con la Constitución Española. Para lograr este objetivo principal, se llevarán a cabo las siguientes tareas específicas:

1. **Comparación de métodos de chunkerización**: Se analizarán y compararán diferentes técnicas de chunkerización para determinar cuál de ellas resulta más eficiente y precisa en la segmentación de los textos legales de la Constitución Española.
2. **Desarrollo de un agente**: Se diseñará y se implementará un agente basado en LLMs capaz de responder a cualquier pregunta relacionada con la Constitución Española. Este agente deberá ser capaz de comprender y procesar consultas en lenguaje natural, proporcionando respuestas precisas y contextualizadas.
3. **Marco teórico del estado del arte**: Proporcionar un marco teórico del estado del arte de los sistemas RAG utilizando LLMs, incluyendo una revisión de la literatura existente y las metodologías actuales en el campo de los sistemas RAG.

Large Language Models (LLMs)

3

3.1. Introducción

Los Modelos de Lenguaje de Gran Escala (LLMs) han demostrado capacidades notables en tareas de procesamiento de lenguaje natural y más allá. Estos modelos han revolucionado el campo del NLP y se han postulado como los modelos a resolver el problema de la generación de texto. Para este capítulo se ha usado la revisión *A Comprehensive Overview of Large Language Models* (Naveed et al., 2023)

3.2. Generación de Texto

La generación de texto es una de las tareas fundamentales que los LLMs intentan resolver. Este problema consiste en producir secuencias de texto coherente y relevante a partir de una entrada dada. La tarea puede implicar varios subproblemas, como la continuación de un texto, la respuesta a preguntas, la traducción automática, entre otros.

El enfoque principal para resolver la generación de texto en LLMs es la predicción del siguiente token. Esto implica modelar la probabilidad condicional de cada token dado el contexto de los tokens anteriores. Formalmente, si tenemos una secuencia de tokens $t_1, t_2, ..., t_n$, el modelo aprende a predecir la probabilidad del token t_{n+1} basándose en los tokens $t_1, t_2, ..., t_n$. Este proceso se puede expresar como:

$$P(t_{n+1}|t_1, t_2, ..., t_n)$$

Para entrenar este tipo de modelos, se utilizan corpus grandes de texto en el cual el modelo ajusta sus parámetros para maximizar la probabilidad de los tokens siguientes.

3.3. Historia y Evolución de los LLMs

La evolución de los modelos de lenguaje natural ha sido marcada por avances significativos en las técnicas y arquitecturas utilizadas. Desde los primeros modelos estadísticos hasta los avanzados modelos de lenguaje de gran escala (LLMs), el campo ha experimentado una transformación profunda.

3.3.1. Modelos de Lenguaje Estadísticos

Los primeros enfoques para el procesamiento de lenguaje natural se basaban en modelos estadísticos. Estos modelos utilizaban probabilidades para predecir la aparición de palabras en un contexto dado. Un ejemplo típico es el modelo de n-gramas, que calcula la probabilidad de una palabra basándose en las n-1 palabras anteriores. Aunque efectivos en ciertos contextos, estos modelos tenían limitaciones significativas en su capacidad para capturar dependencias a largo plazo y gestionar vocabularios extensos.

3.3.2. Modelos de Lenguaje Neurales

Con el advenimiento de las redes neuronales, los modelos de lenguaje experimentaron un cambio paradigmático. Los modelos de lenguaje neurales, como los basados en redes neuronales recurrentes (RNNs), ofrecieron mejoras en la capacidad de modelado y generalización. Sin embargo, a pesar de estas mejoras, las RNNs enfrentaban desafíos en la captura de dependencias a largo plazo debido al problema del gradiente desvaneciente. Las arquitecturas como LSTM (Long Short-Term Memory) y GRU (Gated Recurrent Unit) fueron desarrolladas para mitigar estos problemas.

3.3.3. Transformers y su Impacto

El verdadero avance en los modelos de lenguaje vino con la introducción de los Transformers, presentados por Vaswani et al. en 2017 (Vaswani et al., 2017). Los Transformers se basan en un mecanismo de autoatención que permite al modelo considerar todas las posiciones en la secuencia de entrada simultáneamente, superando las limitaciones de las RNNs en cuanto a la captura de dependencias a largo plazo. Esta arquitectura revolucionó el campo y llevó al desarrollo de modelos preentrenados como BERT (Bidirectional Encoder Representations from Transformers) y GPT (Generative Pre-trained Transformer). La Figura 3.1 muestra la arquitectura básica de un Transformer.

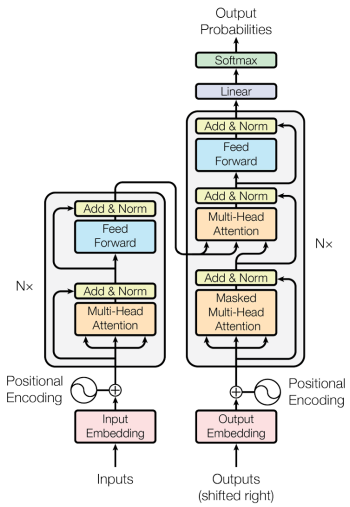

Figura 3.1: Arquitectura básica de un Transformer.

3.3.4. Transición de PLMs a LLMs

La transición de los modelos de lenguaje preentrenados (PLMs) a los modelos grandes del lenguaje (LLMs) se caracterizó por un aumento significativo en el tamaño de los modelos y la cantidad de datos de entrenamiento. Modelos como GPT-3, con 175 mil millones de parámetros, demostraron que el escalado de los modelos y los datos de entrenamiento puede llevar a mejoras drásticas en el rendimiento en una amplia gama de tareas de NLP. Los LLMs han mostrado capacidades sorprendentes en tareas de zero-shot y few-shot learning, donde pueden realizar tareas no vistas durante el entrenamiento con poca o ninguna adaptación adicional. La Figura 3.4 muestra la evolución cronológica de los principales LLMs.

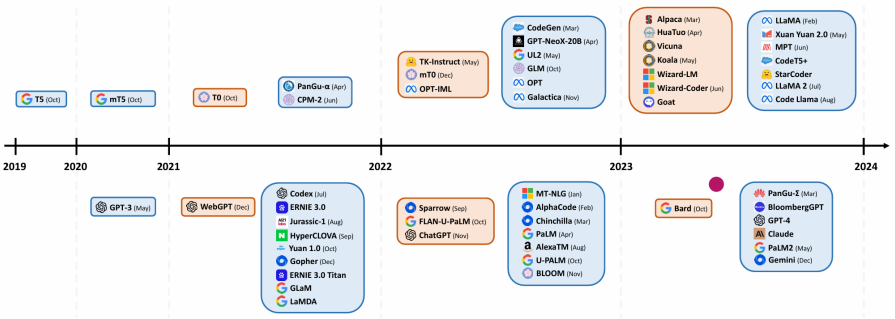

Figura 3.2: Evolución cronológica de los principales LLMs.

3.4. Arquitecturas y Modelos Importantes

En los últimos años, los Modelos grandes del lenguaje (LLMs) han experimentado avances significativos, lo que ha resultado en la creación de arquitecturas y modelos innovadores. Estos modelos no solo han mejorado en términos de capacidad y precisión, sino que también han introducido nuevas formas de abordar problemas complejos en el procesamiento de lenguaje natural. A continuación, se presentan algunas de las arquitecturas históricas y modelos más importantes en el ámbito de los LLMs.

3.4.1. T5 (Text-to-Text Transfer Transformer)

El modelo T5, desarrollado por Google, es un modelo de codificador-decodificador que trata todas las tareas de procesamiento de lenguaje natural (NLP) como problemas de generación de texto. Esto significa que convierte todas las tareas de entrada en una estructura de texto a texto, lo que permite un enfoque unificado para diversas tareas de NLP.

3.4.2. BERT (Bidirectional Encoder Representations from Transformers)

BERT, desarrollado por Google (Devlin et al., 2018), es un modelo basado en transformers que se entrena utilizando un objetivo de modelado de lenguaje enmascarado, donde se predicen tokens enmascarados dentro de una secuencia utilizando el contexto bidireccional de

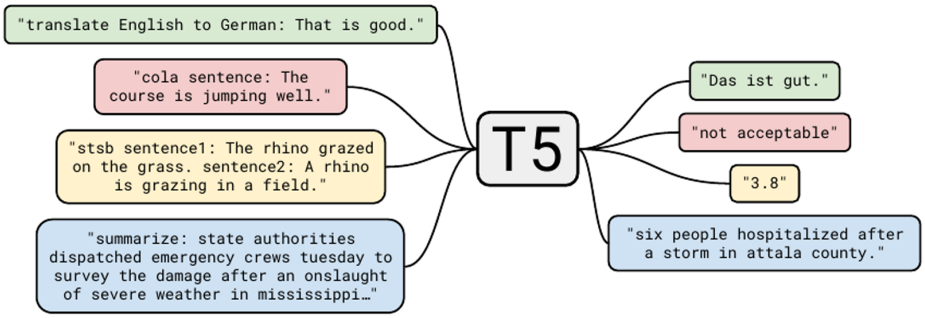

Figura 3.3: Ejemplo unificado de formación texto a texto, imagen origen de (Raffel et al., 2020)

la secuencia completa. Este enfoque permite a BERT capturar relaciones más complejas y contextuales en el texto en comparación con los modelos unidireccionales. La capacidad de BERT para ser finetuneado de manera eficiente sin requerir modificaciones sustanciales en la arquitectura específica de cada tarea ha contribuido significativamente a su adopción y éxito en múltiples dominios de NLP.

3.4.3. mT5 (Multilingual T5)

mT5 es una variante multilingüe del modelo T5, entrenada en el dataset mC4 que abarca 101 idiomas. Este modelo utiliza un vocabulario más amplio de 250,000 tokens para cubrir múltiples lenguajes. Para evitar el sobreajuste o subajuste en un idioma específico, mT5 emplea un procedimiento de muestreo de datos que selecciona muestras de todos los idiomas. Además, durante el afinamiento para tareas específicas utilizando datos en inglés, el modelo puede generar salidas correctas en otros idiomas.

3.4.4. GPT-3 (Generative Pre-trained Transformer 3)

GPT-3, desarrollado por OpenAI, es uno de los modelos de lenguaje más grandes y avanzados hasta la fecha, con 175 mil millones de parámetros. GPT-3 utiliza una arquitectura de transformers similar a GPT-2 pero con atención densa y dispersa en las capas del transformer. Este modelo demostró que el escalado masivo del tamaño del modelo y los datos de entrenamiento puede llevar a mejoras significativas en el rendimiento en una amplia gama de tareas de NLP. GPT-3 es especialmente conocido por su capacidad de realizar tareas en configuraciones de zero-shot, few-shot y one-shot learning, proporcionando respuestas coherentes y contextualmente relevantes.

Entre todos estos avances han llegado infinidad de modelos y arquitecturas que han presentado avances significativos en el ámbito del NLP creando un antes y un después en la figura de los LLMs, véase en la figura 3.4. Desde modelos abiertos como Llama (meta) y Mistral hasta modelos propietarios como GPT4 (OpenAI) o Claude (Anthropic) en sus diferentes versiones.

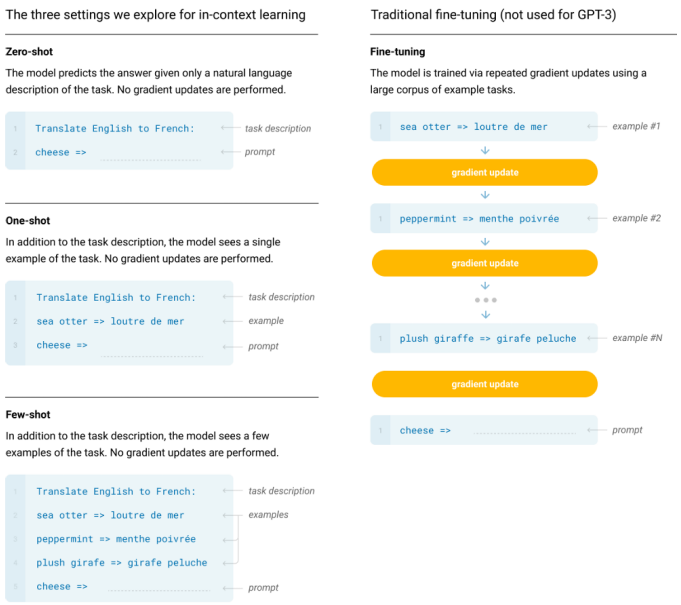

Figura 3.4: Zero-shot, one-shot y few-shot contrastado con finetunning tradicional. Imagen origen de (Brown et al., 2020)

3.5. Aplicaciones de los LLMs

Los LLMs han demostrado capacidades notables en diversas tareas de procesamiento del lenguaje natural. En esta sección, se describen algunas de las aplicaciones más relevantes de los LLMs.

3.5.1. Generación de Texto

La generación de texto es una de las aplicaciones más prominentes de los LLMs. Estos modelos pueden generar texto coherente y contextualizado en una variedad de estilos y formatos, desde artículos periodísticos hasta poesía y ficción. La capacidad de los LLMs para entender y replicar patrones de lenguaje humano permite su uso en la creación de contenido, asistiendo a escritores y periodistas, y automatizando la redacción de informes y documentos.

3.5.2. Resumen Automático

Los LLMs también se utilizan para la generación de resúmenes automáticos de documentos extensos. Mediante técnicas de comprensión y extracción de información, los modelos pueden condensar textos largos en versiones más breves que retienen los puntos clave. Esto es especialmente útil en campos como el periodismo, la investigación científica y la gestión de información empresarial, donde es necesario procesar grandes cantidades de texto de manera eficiente.

3.5.3. Traducción Automática

Otra aplicación importante de los LLMs es la traducción automática de textos entre diferentes idiomas. Los modelos entrenados en textos multilingües pueden traducir con alta precisión, preservando el contexto y el significado original. Esto facilita la comunicación global y el acceso a información en múltiples idiomas, apoyando tanto a usuarios individuales como a organizaciones en la superación de barreras lingüísticas.

3.5.4. Interacción Conversacional

Los LLMs se utilizan extensamente en la creación de agentes conversacionales y chatbots. Estos agentes pueden participar en diálogos naturales y brindar respuestas relevantes a las consultas de los usuarios. Aplicaciones prácticas incluyen asistentes virtuales, atención al cliente automatizada y soporte técnico. Los avances en los LLMs han permitido que estos sistemas sean más intuitivos y efectivos, mejorando la experiencia del usuario.

3.5.5. Respuesta a Preguntas

Los LLMs son capaces de responder preguntas de manera precisa y contextualizada, utilizando información extraída de grandes bases de datos y textos. Esta capacidad se emplea en sistemas de búsqueda avanzada, asistentes personales y herramientas de aprendizaje, proporcionando respuestas rápidas y precisas a consultas específicas. En el contexto de este trabajo, se desarrollará un agente basado en LLM que pueda responder a preguntas relacionadas con la Constitución Española, demostrando así el potencial de los LLMs en aplicaciones jurídicas y educativas.

3.6. Desafíos y Futuras Direcciones

El desarrollo y la implementación de LLMs presentan una serie de desafíos significativos que deben abordarse para maximizar su eficiencia y eficacia. En esta sección, se analizan algunos de los desafíos más críticos.

3.6.1. Costos Computacionales y Energéticos

Los LLMs requieren enormes recursos computacionales tanto para su entrenamiento como para su implementación. Este alto costo computacional no solo limita su accesibilidad sino que también tiene un impacto ambiental considerable debido al consumo de energía. (Strubell et al., 2020) destacaron que los modelos de aprendizaje profundo en NLP pueden tener un impacto energético significativo, comparable a la huella de carbono de automóviles en su ciclo de vida completo . Por lo tanto, es crucial investigar métodos más eficientes en términos de energía y cómputo, como la cuantización, el pruning y el uso de hardware especializado para reducir estos costos.

3.6.2. Privacidad y Datos Sensibles

El uso de grandes cantidades de datos para entrenar LLMs plantea serias preocupaciones sobre la privacidad. Los modelos pueden inadvertidamente memorizar y reproducir datos sensibles, lo que pone en riesgo la privacidad de los individuos. (Brown et al., 2022) discutieron la importancia de definir claramente qué significa preservar la privacidad en el contexto de los modelos de lenguaje y propusieron técnicas para minimizar estos riesgos . La investigación en técnicas de anonimización de datos y el desarrollo de modelos que puedan ser entrenados con datos sintéticos o protegidos son direcciones prometedoras para abordar estas preocupaciones.

3.6.3. Alucinaciones

Los LLMs exhiben *alucinaciones*, donde generan respuestas que, aunque suenen plausibles, son incorrectas o no se alinean con la información proporcionada (Zhang et al., 2023). Las alucinaciones pueden categorizarse en tres tipos:

- **Alucinación en conflicto con la entrada**: Se produce cuando los LLMs generan contenido que diverge de la información proporcionada por los usuarios.
- **Alucinación en conflicto con el contexto**: Se refiere a cuando los LLMs generan contenido que contradice la información que ellos mismos han generado anteriormente.
- **Alucinación en conflicto con los hechos**: Ocurre cuando los LLMs generan contenido que no se alinea con el conocimiento establecido del mundo.

Las alucinaciones representan un desafío significativo para la confiabilidad y precisión de los LLMs. Abordar este problema implica desarrollar técnicas de verificación y validación que permitan a los modelos identificar y corregir estas desviaciones antes de presentar la información al usuario.

Retrieval Augmented Generation (RAGs)

4

Para este capítulo se ha usado el artículo *Retrieval-augmented generation for large language models: A survey* (Gao et al., 2023)

4.1. Introducción

El *Retrieval-Augmented Generation* (RAG) ha emergido como una solución efectiva para superar algunas de las limitaciones de los LLMs, como las alucinaciones y la incapacidad de manejar información actualizada o específica de un dominio. RAG integra la generación de respuestas de los LLMs con la recuperación de información de bases de datos externas, permitiendo que los modelos de lenguaje accedan a información más precisa y reciente.

4.2. Motivación y Aplicaciones

Los LLMs son conocidos por generar contenido no siempre veraz, especialmente cuando se enfrentan a preguntas fuera de su conjunto de datos de entrenamiento. Aquí es donde el RAG entra en juego: este método permite a los LLMs consultar bases de datos externas para recuperar documentos relevantes y aumentar su conocimiento con información específica y precisa. Esta combinación ha demostrado ser particularmente útil en tareas intensivas en conocimiento, como la resolución de preguntas complejas y en la integración de datos especializados.

4.3. Componentes Principales

El marco de RAG se compone principalmente de tres etapas: recuperación, generación y aumento (retrieval, generation y augmentation, respectivamente). Cada una de estas etapas desempeña un papel fundamental en la mejora del rendimiento del LLM:

- **Recuperación**: Esta etapa implica la búsqueda de fragmentos de documentos relevantes a partir de bases de datos externas, utilizando medidas de similitud semántica. Esto asegura que la generación de texto se base en información actual y precisa.
- **Generación**: Una vez recuperados los fragmentos, el modelo genera respuestas utilizando tanto la consulta inicial como la información recuperada. Este proceso ayuda a

evitar la generación de contenido incorrecto o alucinado.

- **Aumento**: Finalmente, el proceso de aumento integra la información recuperada con el contexto de la tarea, mejorando así la calidad y relevancia de la salida del modelo.

Una ilustración visual del flujo de trabajo en un sistema RAG se muestra en la Figura 4.1. En esta imagen, se puede observar cómo el usuario realiza una consulta inicial y, se muestra como el LLM podría responder directamente a la pregunta de manera descontextualizada o cómo puede mediante el uso de un RAG contextualizar la pregunta. Para ello, como primer paso los documentos son fragmentados y convertidos en vectores para su comparación semántica. Luego, el LLM utiliza estos fragmentos recuperados para generar una respuesta más precisa y contextualizada.

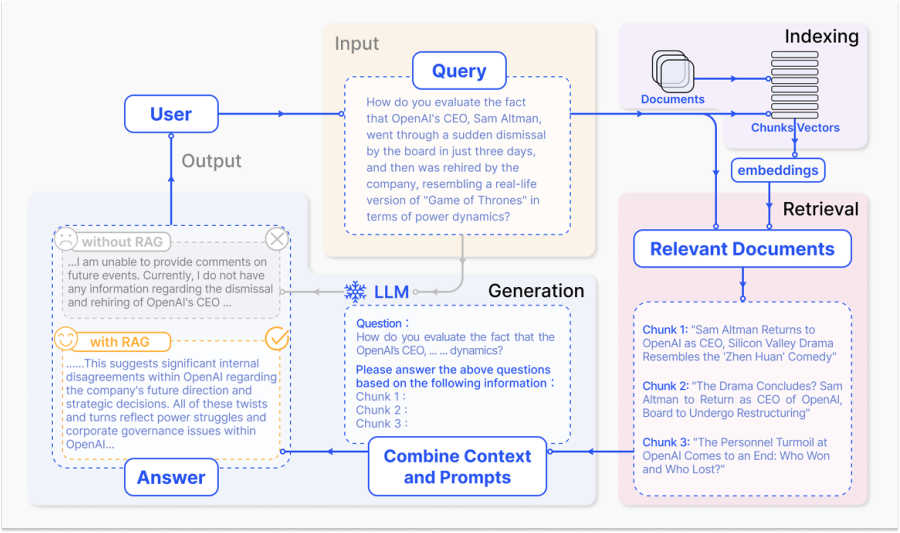

Figura 4.1: Flujo de trabajo de RAG mostrando la integración de documentos recuperados para mejorar la generación de texto.

Como se muestra en la Figura 4.1, el uso de RAG mejora significativamente la calidad de las respuestas generadas por el LLM al proporcionar información adicional relevante. Sin RAG, el modelo puede verse limitado a sus conocimientos previos y no ser capaz de manejar eventos recientes o información específica. Con RAG, el modelo puede acceder a documentos actualizados que enriquecen sus respuestas, como es el caso del ejemplo de la figura en donde el LLM es capaz de dar un análisis detallado del despido y la recontratación del CEO de OpenAI, basado en noticias recuperadas.

4.3.1. Embeddings en Retrieval-Augmented Generation

En el contexto de Retrieval-Augmented Generation (RAG), los *embeddings* juegan un papel crucial para conectar de manera efectiva los sistemas de recuperación y generación de texto. Un *embedding* es una representación numérica de un fragmento de texto, que captura su significado semántico en un espacio de alta dimensión, permitiendo que los algoritmos de

recuperación encuentren información relevante mediante la comparación de similitudes semánticas en lugar de simples coincidencias de palabras clave datastax (2023); Vergadia (2023).

Proceso de Vectorización: El proceso de creación de *embeddings* comienza dividiendo el texto en fragmentos manejables, a menudo denominados *chunks*. Estos fragmentos son transformados en vectores utilizando modelos de aprendizaje automático especializados, como *word2vec* o *sentence transformers*. Estos modelos de *embeddings* están diseñados para representar el contexto y el significado de los fragmentos de texto de manera que los vectores de fragmentos similares estén más cerca unos de otros en el espacio vectorial Nvidia (2023).

Una vez creados, estos vectores se almacenan en bases de datos vectoriales especializadas, como Pinecone, Milvus o Faiss, que están optimizadas para la recuperación rápida de información basada en similitudes vectoriales. Esto permite que el sistema de recuperación encuentre rápidamente fragmentos de texto que sean relevantes para una consulta dada datastax (2023).

Función en RAG: Los *embeddings* son esenciales para el funcionamiento del sistema RAG. Cuando se presenta una consulta, el sistema genera un *embedding* para dicha consulta y lo compara con los *embeddings* almacenados en la base de datos vectorial. Este proceso de búsqueda vectorial permite que el sistema identifique los fragmentos de texto más relevantes en función de su similitud semántica con la consulta Vergadia (2023). Estos fragmentos recuperados luego se integran en la generación de respuestas del modelo de lenguaje, mejorando la precisión y relevancia de las respuestas al proporcionar información actualizada y específica del contexto.

Almacenamiento y Procesamiento de los Embeddings: Para maximizar la eficiencia, los *embeddings* se almacenan en bases de datos vectoriales optimizadas para la búsqueda y recuperación de datos en espacios de alta dimensión. Esto permite que el sistema de RAG maneje grandes volúmenes de información de manera eficiente, ofreciendo respuestas precisas basadas en datos externos relevantes que enriquecen la salida generada por los LLMs Nanonets (2023).

Figura 4.2: Ejemplos de embeddings. (OpenAI, 2024)

En resumen, los *embeddings* son la columna vertebral del sistema de recuperación en RAG, permitiendo que los modelos de lenguaje accedan a información relevante de manera eficiente, lo que resulta en respuestas más precisas y contextualmente ricas Nvidia (2023); Vergadia (2023).

4.4. Desafíos y Futuras Direcciones

A pesar de su éxito, el RAG enfrenta varios desafíos. Uno de los principales es la necesidad de manejar información ruidosa o contradictoria, lo que puede afectar la calidad de la generación de texto. Además, los desarrollos futuros buscan mejorar la integración de datos de múltiples fuentes y mejorar la capacidad de los modelos para adaptarse a tareas más complejas.

Estrategias de chunkerización para LLMs

5

5.1. Introducción

La chunkerización en el ámbito de los modelos de lenguaje de gran escala (LLM) se refiere al proceso de dividir textos extensos en segmentos más manejables, conocidos como *chunks*. Esta técnica es fundamental para mejorar la gestión y el procesamiento de grandes volúmenes de información en aplicaciones de inteligencia artificial que utilizan modelos de lenguaje para entender y generar texto humano.

Necesidad de la Chunkerización: Los modelos de lenguaje, son inherentemente limitados por la cantidad de texto que pueden procesar en una sola instancia debido a restricciones de memoria y capacidad de cálculo. Al descomponer los textos en chunks más pequeños, se facilita que el modelo maneje datos extensos de manera eficiente, permitiendo una evaluación más rápida y precisa.

Objetivos de la Chunkerización: El principal objetivo de la chunkerización es maximizar la relevancia y la precisión del texto procesado. Al dividir el contenido en partes significativas y manejables, se busca preservar la coherencia semántica sin sobrecargar el modelo. Además, esta estrategia es crucial para la indexación efectiva de contenidos en bases de datos vectoriales, lo que mejora la recuperación y relevancia de los resultados en consultas específicas.

Retos de la Chunkerización: A pesar de sus beneficios, la chunkerización presenta varios desafíos. El más significativo es determinar el tamaño óptimo de cada chunk. Si los chunks son demasiado pequeños, pueden perder contexto necesario para una comprensión completa; si son demasiado grandes, pueden exceder las capacidades de procesamiento del modelo y diluir la relevancia del contenido. Otro reto es la selección de los puntos de corte dentro del texto, que debe hacerse de manera que se preserve la integridad del contenido semántico.

La chunkerización es un proceso esencial que requiere un cuidadoso equilibrio entre tamaño de chunk, preservación del contexto y capacidades del modelo. En las siguientes secciones, exploraremos diferentes estrategias y métodos de chunkerización, evaluando sus ventajas y limitaciones en diversos contextos de aplicación.

5.2. Consideraciones sobre la Chunkerización

Al desarrollar una estrategia de chunkerización eficaz, es crucial considerar varios aspectos que influirán en el rendimiento y la eficacia del modelo de lenguaje. A continuación, se detallan algunas de las consideraciones más importantes:

5.2.1. Naturaleza del Contenido

El tipo de contenido que se está indexando es determinante en la selección de la estrategia de chunkerización. Por ejemplo, documentos largos como artículos o libros pueden requerir un enfoque diferente en comparación con contenidos más breves como tuits o mensajes instantáneos. Esta distinción afecta tanto la elección del modelo para crear los embeddings como la estrategia de chunkerización a aplicar.

5.2.2. Modelo y Tamaño Óptimo del Chunk

Dependiendo del modelo utilizado para crear los embeddings, existirán tamaños de chunk en los que el modelo desempeñará mejor. Por ejemplo, algunos modelos están optimizados para trabajar con frases individuales, mientras que otros pueden manejar mejor segmentos de texto más largos. Identificar el tamaño de chunk que maximiza la calidad de los embeddings es crucial para el éxito de la estrategia de chunkerización.

5.2.3. Complejidad y Longitud de las Consultas de Usuario

Las expectativas sobre la longitud y complejidad de las consultas de los usuarios deben guiar la manera en que se chunkeriza el contenido. Si las consultas suelen ser cortas y específicas, es posible que se prefiera chunkerizar el contenido en segmentos más pequeños para reflejar esta especificidad. En cambio, consultas más largas y complejas podrían beneficiarse de chunks más grandes que proporcionen un contexto más amplio.

5.2.4. Uso de los Resultados Recuperados

El propósito final de los chunks recuperados también juega un papel crucial. Dependiendo de si los resultados serán utilizados para búsqueda semántica, respuesta a preguntas, resumen, o cualquier otro fin, la estrategia de chunkerización podría variar para optimizar la relevancia y utilidad de la información recuperada.

5.2.5. Limitaciones Técnicas

Las limitaciones técnicas, como el número máximo de tokens que el modelo puede procesar en una sola instancia o las restricciones de memoria, son críticas para definir el tamaño de los chunks. Estas limitaciones no solo afectan cómo se chunkeriza el contenido sino también cómo se procesa posteriormente en el modelo.

Estas consideraciones son fundamentales para desarrollar una estrategia de chunkerización que no solo sea eficiente sino también efectiva en términos de mejorar la relevancia y precisión de las respuestas del modelo.

5.3. Estrategias de Chunkerización

La chunkerización puede implementarse de diversas maneras, cada una con sus propios beneficios y desafíos. A continuación, exploramos algunas de las estrategias más comunes y cómo pueden optimizarse según diferentes necesidades y contextos.

5.3.1. Chunkerización de Tamaño Fijo

Esta estrategia consiste en dividir el texto en segmentos de un tamaño predeterminado, medido en número de tokens o carácteres. La principal ventaja de este método es su simplicidad y facilidad de implementación, ya que no requiere un análisis profundo del contenido. Sin embargo, un desafío significativo es que puede cortar frases a la mitad, perdiendo contexto o generando chunks que carecen de sentido por sí solos. Para mitigar esto, se puede optar por incluir una superposición entre los chunks, donde el final de un chunk se superpone con el inicio del siguiente, ayudando a preservar el contexto (overload).

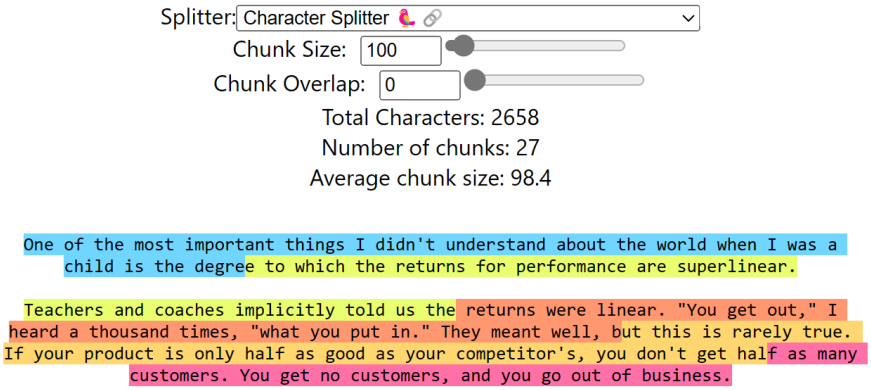

Figura 5.1: Ejemplo de chunkerización tamaño fijo sin overload

5.3.2. Chunkerización Consciente del Contenido

A diferencia de la chunkerización de tamaño fijo, este método tiene en cuenta el contenido textual para determinar los puntos de corte. La chunkerización consciente del contenido puede basarse en la detección de fronteras naturales en el texto, como finales de oraciones o párrafos, asegurando que cada segmento contenga una unidad completa de información. Esto ayuda a mantener la coherencia semántica de los chunks, haciendo que cada uno sea significativo por sí mismo y mejore la calidad de las incrustaciones generadas por el modelo. Este método es particularmente útil en documentos con una estructura clara y bien definida, como artículos académicos o reportes técnicos.

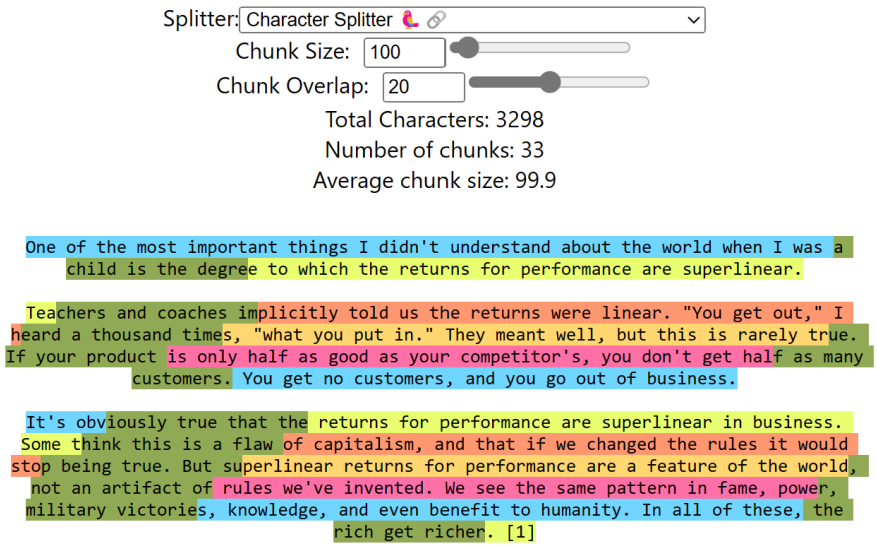

Figura 5.2: Ejemplo de chunkerización tamaño fijo con overload

5.3.3. Chunkerización Recursiva

Este enfoque implica un proceso iterativo y jerárquico de dividir el texto en chunks. Inicialmente, el texto se divide utilizando un criterio de separación amplio, y si los chunks resultantes aún son demasiado grandes o no cumplen con ciertos criterios, el proceso se repite en cada chunk hasta alcanzar el tamaño o la estructura deseada. La chunkerización recursiva es útil en textos largos y complejos donde los niveles múltiples de división permiten manejar mejor la diversidad y complejidad del contenido.

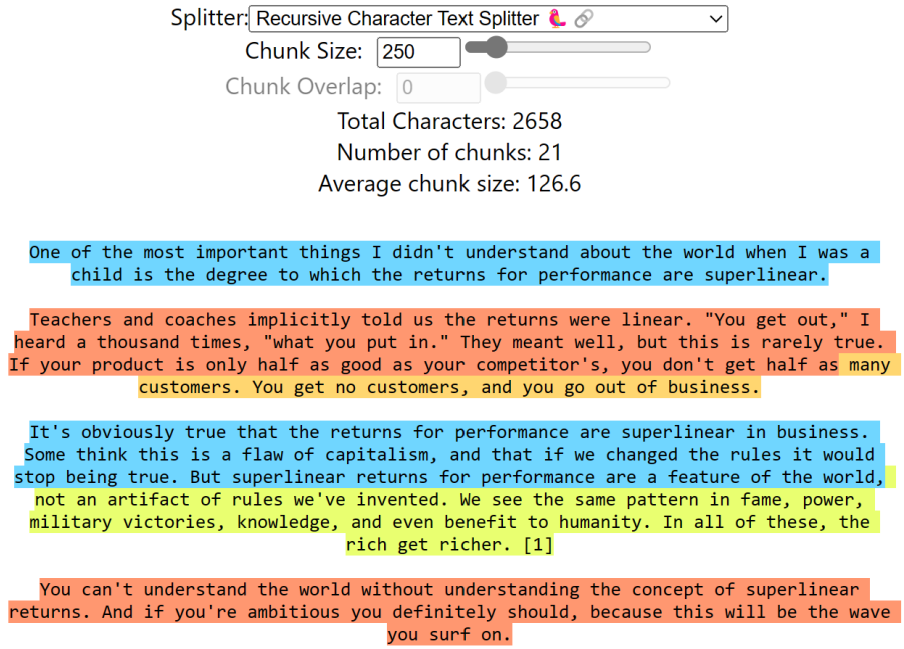

Figura 5.3: Ejemplo de chunkerización recursiva

5.3.4. Chunkerización Especializada

Esta estrategia se adapta a formatos de contenido específicos que requieren un tratamiento particular, como puede ser el caso de textos en Markdown o LaTeX. La chunkerización especializada reconoce y respeta la estructura y sintaxis propias de estos formatos, asegurando que los chunks resultantes conserven la integridad y funcionalidad del texto original. Por ejemplo, en un documento LaTeX, los chunks podrían definirse para encapsular secciones completas o subsecciones, preservando las etiquetas y comandos propios del formato.

Cada una de estas estrategias tiene sus propias fortalezas y puede ser más adecuada para diferentes tipos de textos y aplicaciones. La elección de una estrategia de chunkerización debe basarse en una evaluación cuidadosa de los requisitos del proyecto, la naturaleza del contenido y los objetivos específicos del sistema de procesamiento de lenguaje natural.

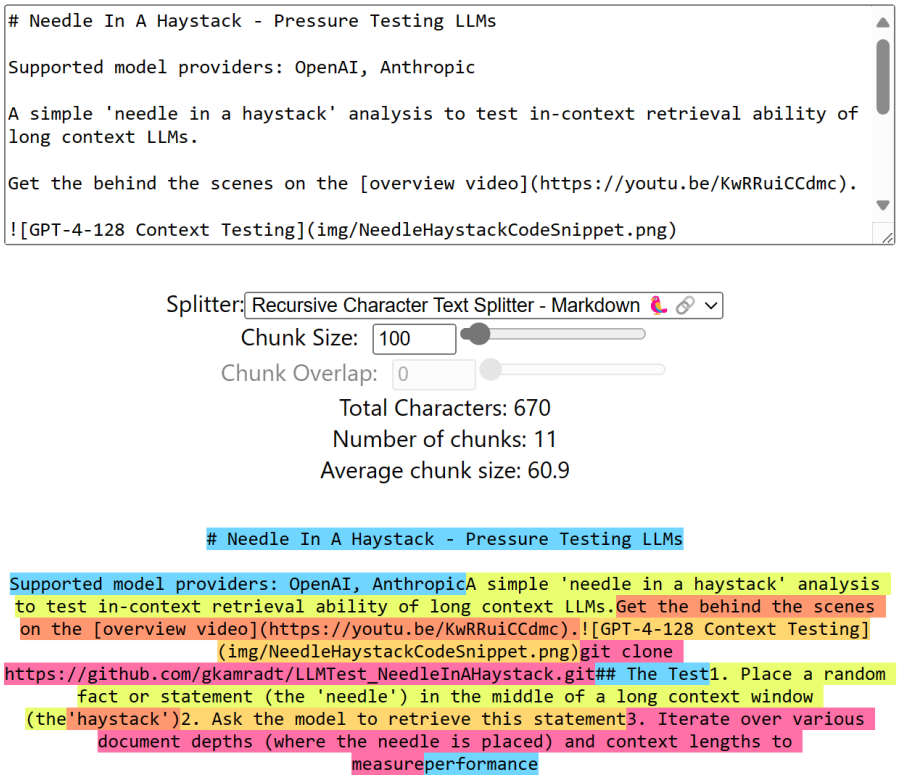

Figura 5.4: Ejemplo de chunkerización Especializada

5.3.5. Chunkerización Semántica

La chunkerización semántica es una técnica avanzada que divide el texto en unidades basadas en la relevancia semántica, utilizando modelos de creación de embeddings para evaluar la similitud entre segmentos de texto.

5.3.5.1. Implementación y Ventajas

Esta estrategia se implementa mediante la transformación de texto en representaciones vectoriales y el cálculo de su similitud, agrupando segmentos similares. Es especialmente útil para mantener la coherencia del contenido en tareas de recuperación de información y análisis de texto.

5.3.5.2. Desafíos y Aplicaciones

A pesar de su efectividad, enfrenta desafíos como la alta demanda computacional y la dificultad en determinar umbrales de similitud óptimos. Su aplicación es ideal en sistemas de búsqueda semántica y en el análisis de textos complejos en contextos académicos o de investigación.

5.4. Evaluación de los chunks

La selección del tamaño de chunk adecuado es un paso fundamental para optimizar el rendimiento de los sistemas de Recuperación Aumentada por Generación (RAG). El tamaño del chunk influye directamente en la calidad de la información recuperada y, por lo tanto, en la precisión y relevancia de las respuestas generadas por el modelo. En esta sección, explicaremos los aspectos clave de la evaluación del tamaño de los chunks y cómo afecta al rendimiento de un sistema RAG utilizando LlamaIndex, una librería diseñada para optimizar este tipo de sistemas.

5.4.1. Impacto del Tamaño del Chunk en el Rendimiento del Sistema

El tamaño del chunk determina la granularidad de la información capturada durante el proceso de recuperación, lo que afecta la precisión y relevancia de las respuestas generadas. Chunk sizes pequeños, como 256 tokens, tienden a ser más detallados y contextualmente relevantes, lo que puede mejorar la alineación de las respuestas con las consultas del usuario. Sin embargo, si el chunk es demasiado pequeño, puede ocurrir que información crítica no esté contenida en los segmentos recuperados, lo que afectaría la completitud de la respuesta (LlamaIndex, 2023).

Por otro lado, tamaños de chunks más grandes, como 512 o 1024 tokens, proporcionan un contexto más amplio pero corren el riesgo de incluir información irrelevante, lo que puede sobrecargar el sistema y disminuir la precisión de las respuestas. Este balance entre granularidad y contexto es crucial para garantizar que el sistema RAG recupere la información más relevante de manera eficiente (DataCamp, 2023).

5.4.2. Evaluación Cuantitativa y Cualitativa del Tamaño de los Chunks

Para determinar el tamaño óptimo del chunk, se utilizan tanto técnicas de evaluación cuantitativa como cualitativa. Las técnicas cuantitativas consisten en variar sistemáticamente el ta-

maño de los chunks y medir métricas como la precisión, la relevancia y la calidad de las respuestas. En experimentos típicos, se calculan métricas como el tiempo de respuesta promedio, la fe en la fidelidad de las respuestas (*faithfulness*) y la relevancia (*relevancy*) con respecto a las consultas (Vectorize.io, 2023). Estos datos permiten obtener medidas objetivas sobre cómo diferentes tamaños de chunks impactan en el rendimiento del sistema.

Por otro lado, la evaluación cualitativa implica la participación de jueces humanos que califican la calidad de las respuestas generadas. Factores como la coherencia, la naturalidad y la adecuación al contexto son considerados.

5.4.3. Balance entre Precisión y Eficiencia Computacional

Uno de los desafíos clave en la selección del tamaño de chunk es encontrar un equilibrio entre precisión y eficiencia computacional. Chunks más grandes pueden requerir más recursos computacionales y tiempo de procesamiento, lo que podría afectar la velocidad de respuesta del sistema, especialmente en aplicaciones en tiempo real, como chatbots o asistentes virtuales. Por el contrario, chunks más pequeños pueden ser más eficientes pero requerirán consultas de recuperación más frecuentes, lo que aumenta la sobrecarga computacional (Vectorize.io, 2023).

Este balance entre precisión y eficiencia es esencial para garantizar que el sistema RAG funcione de manera óptima dentro de las limitaciones computacionales. La selección cuidadosa del tamaño de chunk y su evaluación en función de los requisitos específicos del sistema y el dominio del problema son pasos críticos para lograr un rendimiento adecuado.

La evaluación del tamaño de los chunks es un proceso clave para optimizar los sistemas RAG. Mediante el uso de técnicas de evaluación cuantitativa y cualitativa, es posible determinar el tamaño óptimo que maximiza la relevancia y precisión de las respuestas, al tiempo que minimiza los costos computacionales. Adaptar el tamaño de chunk a las necesidades específicas del sistema y del dominio asegura un mejor rendimiento general del sistema (DataCamp, 2023; LlamaIndex, 2023).

5.5. Conclusiones

La chunkerización es fundamental para mejorar la eficiencia y precisión en aplicaciones relacionadas con LLM. No existe una solución única para todos los casos, por lo que es crucial evaluar cada situación individualmente para encontrar la estrategia más efectiva.

Retrievers

6

6.1. Introducción

Habiendo introducido los conceptos fundamentales de los Modelos de Lenguaje a Gran Escala (LLMs) y los RAG en los capítulos anteriores, ahora nos centramos en uno de los componentes críticos que potencian los sistemas RAG: los *retrievers*. Este concepto es esencial para la creación de un RAG sobre un LLM, permitiéndole acceder a información actualizada y relevante más allá de su entrenamiento inicial.

En este capítulo, exploraremos los diferentes tipos de *retrievers*, cómo se integran en los sistemas RAG y las distintas metodologías para su implementación. Discutiremos desde los simples *retrievers* basados en bases de datos vectoriales hasta complejas configuraciones que involucran múltiples métodos de recuperación de información.

6.2. Fundamentos de los Retrievers

6.2.1. Definición y Función

Un *retriever* es una herramienta que busca y recupera información relevante de un conjunto de datos grande y posiblemente no estructurado. En el contexto de RAG, un *retriever* actúa como el puente entre la pregunta del usuario y la base de conocimientos o información almacenada. La eficacia de un *retriever* es crucial, ya que determina la calidad y la relevancia de la información que se utiliza para generar respuestas.

6.2.2. Rol en RAG

Como se puede ver en la figura 6.1, en un sistema RAG, el retriever selecciona fragmentos de texto o documentos que son potencialmente útiles para responder a una consulta específica. Estos documentos se pasan luego a un modelo de lenguaje, que integra esta información para generar una respuesta coherente y contextualmente rica. Esta capacidad de incorporar información dinámica de diversas fuentes externas distingue a los RAGs de los modelos de lenguaje tradicionales que dependen únicamente de lo que han aprendido durante su entrenamiento inicial.

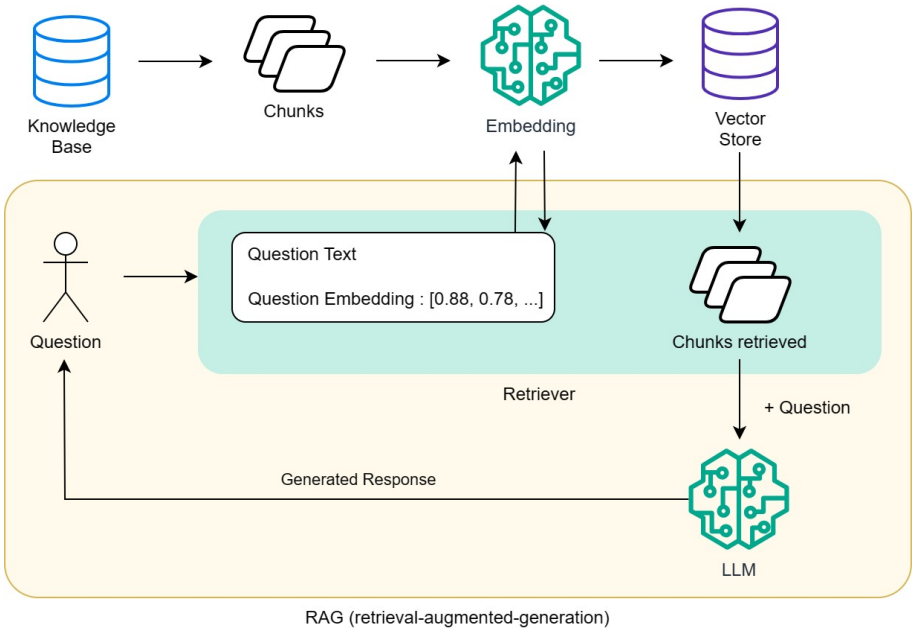

Figura 6.1: Esquema de un RAG

6.3. Tipos de Retrievers Tradicionales

Los retrievers se clasifican principalmente en tres tipos, cada uno con sus propias técnicas y aplicaciones preferidas en el procesamiento de consultas y la recuperación de información. Estos son los retrievers *sparse*, *dense* e *hybrid*.

6.3.1. Retrievers Sparse

Los retrievers *sparse* utilizan métodos basados en coincidencia de términos para recuperar documentos. Estos métodos, como TF-IDF (Frequency-Inverse Document Frequency) y BM25, analizan la frecuencia de las palabras en documentos para determinar su relevancia con respecto a una consulta. La principal ventaja de los retrievers sparse es su eficiencia y rapidez, lo que los hace adecuados para grandes volúmenes de datos donde la relevancia se puede medir por la presencia de palabras clave específicas. (Lewis et al., 2020)

Estos retrievers son particularmente útiles en situaciones donde las consultas son directas y las palabras clave bien definidas son suficientes para recuperar información relevante. Sin embargo, pueden no ser efectivos en contextos donde la semántica de la consulta es más importante que las palabras específicas utilizadas.

6.3.2. Retrievers Dense

Por otro lado, los retrievers *dense* emplean modelos de vectorización (embeddings) para representar tanto las consultas como los documentos como vectores densos en un espacio vectorial continuo. Estos modelos capturan la semántica de las palabras y las frases, permitiendo una recuperación más efectiva en casos donde las consultas y los documentos no comparten

términos exactos pero están relacionados en contexto y significado. (Lewis et al., 2020)

Los retrievers dense son esenciales para aplicaciones donde la relevancia semántica entre la consulta y los documentos es más crítica que la coincidencia exacta de palabras. Su capacidad para entender y procesar el lenguaje de manera más natural los hace muy valiosos en sistemas RAG modernos. Estos retrievers son muy dependientes del modelo de vectorización usado.

6.3.3. Retrievers Hybrid

Finalmente, los retrievers *hybrid* combinan las técnicas de los retrievers sparse y dense para aprovechar las ventajas de ambos. Este enfoque permite una recuperación inicial rápida de documentos potencialmente relevantes utilizando métodos sparse, seguida de un refinamiento más detallado y semánticamente rico con técnicas dense.

Los retrievers hybrid ofrecen un equilibrio entre eficiencia y profundidad semántica, lo que los hace ideales para sistemas RAG que requieren tanto precisión en la recuperación de información relevante como la capacidad de manejar grandes volúmenes de datos.

Cada tipo de retriever tiene su lugar en el ecosistema de los RAG, y la elección de uno sobre otro depende en gran medida de las necesidades específicas del sistema y las características de las consultas y los conjuntos de datos utilizados.

6.4. Retrievers Avanzados

Dada la evolución constante de los sistemas RAG y la necesidad de integrar información más específica y contextual, la comunidad ha estado desarrollando diferentes retrievers que tienen sentido en determinados contextos. Estos retrievers utilizan técnicas sofisticadas para mejorar la precisión y la relevancia de la información recuperada. Aquí exploramos varios enfoques que se recopilan en la web de langchain (LangChain, 2024).

6.4.1. Documento Matriz (Parent Document)

Este tipo de retriever es ideal cuando los documentos contienen numerosos fragmentos de información distintos que se benefician de ser indexados individualmente, pero que es preferible recuperar juntos. El proceso involucra la indexación de múltiples fragmentos o trozos de cada documento. Luego, estos fragmentos se buscan en el espacio de vectores (embeddings) para identificar aquellos que son más similares entre sí, pero en lugar de recuperar solo los fragmentos individuales, se recupera y devuelve el documento completo al que pertenecen. Esta metodología asegura que toda la información contextual relacionada esté disponible para generar respuestas más completas y detalladas.

6.4.1.1. Multi Vector

Este retriever es especialmente útil cuando es posible extraer de los documentos información que se considera relevante para indexar (a parte del texto en sí). El proceso implica

la creación de múltiples vectores para cada documento. Cada vector puede ser generado de diversas maneras, incluyendo, por ejemplo, resúmenes del texto, preguntas hipotéticas, descripciones de imágenes o tablas, etc. Esta técnica permite una representación más rica de los documentos, facilitando una recuperación más precisa y alineada con las necesidades específicas del retriever.

6.4.2. Consulta Autónoma (Self Query)

Utiliza un LLM para transformar la entrada del usuario en dos componentes principales: una cadena que se busca semánticamente y un filtro de metadatos que acompaña la consulta. Este método es particularmente valioso porque muchas veces las preguntas están relacionadas con los metadatos de los documentos, no con su contenido textual directo. La capacidad de enfocar la búsqueda en metadatos permite recuperar información que es más relevante para la intención específica del usuario, mejorando así la precisión y utilidad de las respuestas generadas. Este tipo de retriever es especialmente útil cuando las preguntas de los usuarios se responden mejor recuperando documentos basados en metadatos en lugar de en similitudes textuales directas

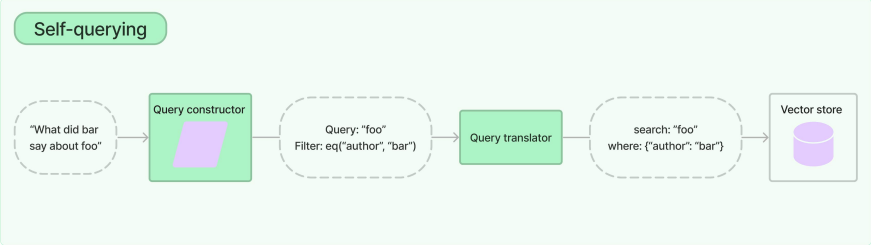

Figura 6.2: Esquema de un Self Query retriever (LangChain, 2024)

6.4.2.1. Compresión Contextual

Uno de los desafíos en la recuperación de documentos es que generalmente no se conocen las consultas específicas que enfrentará el sistema de almacenamiento de documentos cuando se ingieren datos en el sistema. Esto significa que la información más relevante para una consulta puede estar enterrada en un documento con mucho texto irrelevante. Pasar ese documento completo a través de la aplicación puede llevar a llamadas más costosas al LLM y a respuestas de menor calidad.

La compresión contextual está diseñada para solucionar esto. En lugar de devolver inmediatamente los documentos recuperados tal como están, se pueden comprimir utilizando el contexto de la consulta dada, de modo que solo se devuelva la información relevante.

6.4.3. Time-Weighted Vectorstore

Optimiza la recuperación basándose tanto en la similitud semántica como en la fechateams del documento. Es ideal para aplicaciones donde la información más actual es crucial, como

en el seguimiento de noticias o tendencias del mercado.

6.4.4. Multi-Query Retriever

Genera múltiples consultas a partir de una inicial para abordar consultas complejas que requieren información sobre varios temas. Luego, recupera documentos para cada una de estas consultas, lo que garantiza una respuesta exhaustiva y detallada. Es muy útil cuando las preguntas van a tratar sobre varios temas y temas no relacionados.

6.4.5. Long-Context Reorder

Es especialmente útil en modelos que enfrentan degradaciones de rendimiento cuando deben acceder a información relevante en medio de contextos extensos. Este tipo de retriever reordena los documentos recuperados para optimizar la atención del modelo a la información más pertinente, colocando los documentos más relevantes al principio y al final, mientras que los menos relevantes quedan en el medio. El "Long Context Reorderütiliza un enfoque específico conocido como "Lost in the middle"para contrarrestar el problema de que los modelos ignoren documentos importantes simplemente porque aparecen en posiciones menos destacadas dentro de un conjunto de datos grande. (Liu et al., 2023)

Estos retrievers avanzados representan un conjunto diverso de herramientas diseñadas para mejorar la eficiencia y efectividad de los sistemas RAG, facilitando respuestas más precisas y contextuales en una amplia variedad de aplicaciones.

6.5. Métricas para evaluar el sistema RAG

En los sistemas RAG, evaluar la calidad de las respuestas generadas es crucial para garantizar que la información recuperada y generada sea relevante, precisa y coherente con el contexto. En este apartado, describimos las principales métricas que se utilizan para evaluar los sistemas RAG, basándonos en el marco propuesto en RAGAS (Es et al., 2023).

6.5.1. Faithfulness (Fidelidad)

La fidelidad mide hasta qué punto la respuesta generada está basada en la información proporcionada por el contexto recuperado. En otras palabras, se asegura de que las afirmaciones hechas en la respuesta puedan inferirse directamente del contexto proporcionado y no se generen .ªlucinaciones". Para medir la fidelidad, se divide la respuesta en varias afirmaciones y se verifica si cada una de ellas está respaldada por el contexto. El puntaje de fidelidad se calcula como la proporción de afirmaciones correctas sobre el total de afirmaciones generadas. Esta métrica es especialmente importante en dominios donde la precisión factual es fundamental, como en el derecho y la medicina (Es et al., 2023).

6.5.2. Answer Relevance (Relevancia de la Respuesta)

La relevancia de la respuesta evalúa si la respuesta generada aborda directamente la pregunta planteada, independientemente de si es factualmente correcta o no. Una respuesta relevante debe centrarse en responder la pregunta de manera adecuada, evitando información redundante o innecesaria. Para medir esta métrica, se generan preguntas alternativas a partir de la respuesta y se calcula la similitud entre estas preguntas y la pregunta original mediante técnicas de embeddings. Cuanto mayor sea la similitud, mayor será el puntaje de relevancia de la respuesta (Es et al., 2023).

6.5.3. Context Relevance (Relevancia del Contexto)

La relevancia del contexto mide hasta qué punto el contexto recuperado es relevante para responder la pregunta. Un buen contexto debe proporcionar la información justa y necesaria, sin añadir datos irrelevantes que puedan confundir al modelo de generación. Para evaluar esta métrica, se extraen oraciones clave del contexto que son relevantes para la pregunta y se calcula la proporción de oraciones relevantes sobre el total del contexto. Esta métrica es crítica en sistemas que trabajan con contextos largos, ya que los modelos de lenguaje grandes tienden a ser menos efectivos cuando el contexto contiene demasiada información innecesaria (Es et al., 2023).

Las métricas de fidelidad, relevancia de la respuesta y relevancia del contexto son fundamentales para evaluar de manera efectiva los sistemas RAG. Estas métricas permiten detectar alucinaciones, asegurar que las respuestas se ajusten a las preguntas planteadas y optimizar la calidad de la información recuperada y utilizada en la generación de respuestas. El uso de estas métricas en la evaluación de sistemas RAG garantiza que se mantenga un alto nivel de precisión y relevancia en las respuestas generadas.

Es importante señalar la necesidad de un LLM que actúe como **modelo juez** para poder automatizar estas métricas. Este modelo debe tener capacidad suficiente de razonamiento para llevar acabo esta evaluación. Además, hay que añadir que los resultados de las evaluaciones dependerán también de la fiabilidad del modelo juez elegido.

6.6. Conclusión

A lo largo de este capítulo, hemos explorado en profundidad los diversos tipos de *retrievers* que potencian los sistemas de RAG. Hemos visto cómo los *retrievers* no solo facilitan la recuperación de información relevante y actualizada, sino que también optimizan la eficiencia del procesamiento y la precisión de las respuestas generadas por los LLMs. La implementación adecuada de estos *retrievers* puede marcar una diferencia significativa en la capacidad de un sistema RAG para proporcionar respuestas precisas, contextuales y de alta calidad.

Este capítulo también ha subrayado la importancia de elegir el tipo correcto de *retriever* según las necesidades específicas del sistema y del dominio de aplicación. Con las tecnologías

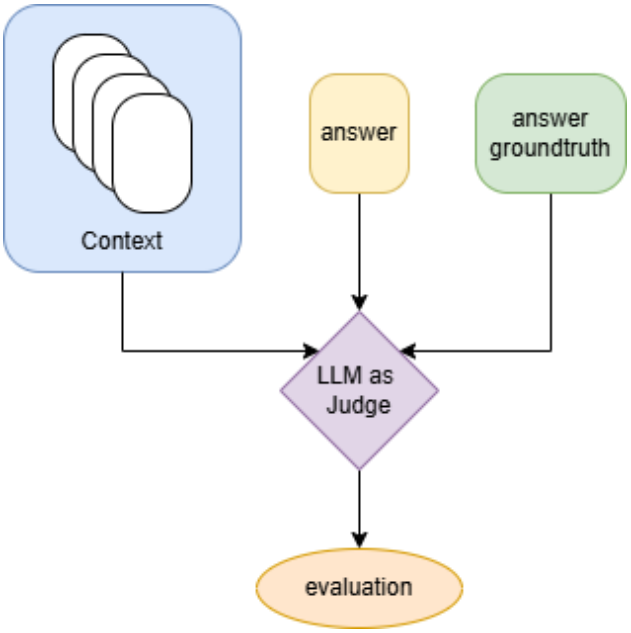

Figura 6.3: Esquema de un LLM como modelo juez

emergentes y los desarrollos continuos en el campo de la inteligencia artificial, es probable que veamos aún más innovaciones en los métodos de recuperación, lo que a su vez podría ampliar aún más las capacidades y aplicaciones de los RAGs.

En resumen, los *retrievers* son componentes cruciales que no solo enriquecen la funcionalidad de los sistemas basados en LLMs mediante la incorporación de conocimientos actualizados y contextualmente relevantes, sino que también representan un área de investigación activa y en evolución que continuará influyendo en el futuro de la generación de lenguaje y la recuperación de información.

Sistema RAG aplicado a la Constitución Española

7

7.1. Introducción

En este capítulo se presenta un caso práctico de laboratorio en el que se diseña un sistema RAG orientado a la recuperación de información de la Constitución Española para responder preguntas relacionadas con dicho documento.

El contenido de este capítulo incluye una justificación de los modelos empleados en el proceso, un análisis detallado del documento (Constitución Española), la definición de las estrategias de chunkerización utilizadas, justificación mediante métricas de la estrategia escogida, y finalmente, la implementación del agente capaz de responder a preguntas sobre el documento.

7.2. Modelos usados

Para el sistema RAG creado han sido necesarios 3 modelos. Por simplicidad para el hardware necesario se van a usar los modelos via API. Para este trabajo se usará la nube de Microsoft Azure y el recurso Azure OpenAI sobre el que se consumirán mediante API los modelos a continuación mencionados.

7.2.1. Modelo Juez

Modelo LLM encargado de evaluar las respuestas aportadas por el agente. Para esta tarea es necesario un modelo con alta calidad y razonamiento en sus respuestas.

En este caso se ha usado el modelo GPT-4o debido a que en el momento de la creación de este trabajo era el modelo más capaz según todos los benchmarks como puede verse en la figura A.1 (no tiene puntuación en Chatbot Arena debido a que no ha sido evaluado en esa métrica). Como otros puntos fuertes podemos ver en la figura 7.1 que tiene un precio contenido con respecto a sus competidores directos en calidad y además como se ve en la figura 7.2 es más rápido en sus respuestas.

Finalmente, la característica clave para su elección es la relación calidad/precio donde en el momento de el desarrollo de este trabajo no tenía rival (véase la figura 7.3).

7.2.2. Modelo del Agente

Modelo LLM encargado de dar respuesta a una pregunta en base al contexto aportado. Este modelo no es necesario que sea de la calidad del modelo anterior ya que es de suponer que tendrá el contexto suficiente para resolver la pregunta del usuario.

Para el modelo del agente se ha elegido GPT-3.5-Turbo ya que muestra la mejor relación velocidad/precio. Hay que tener en cuenta que este modelo será el que podrá ser consultado en masa para responder preguntas sobre la constitución española y por tanto tanto la velocidad de respuesta como el precio forman un valor diferencial (véase las figuras 7.1 y 7.2.

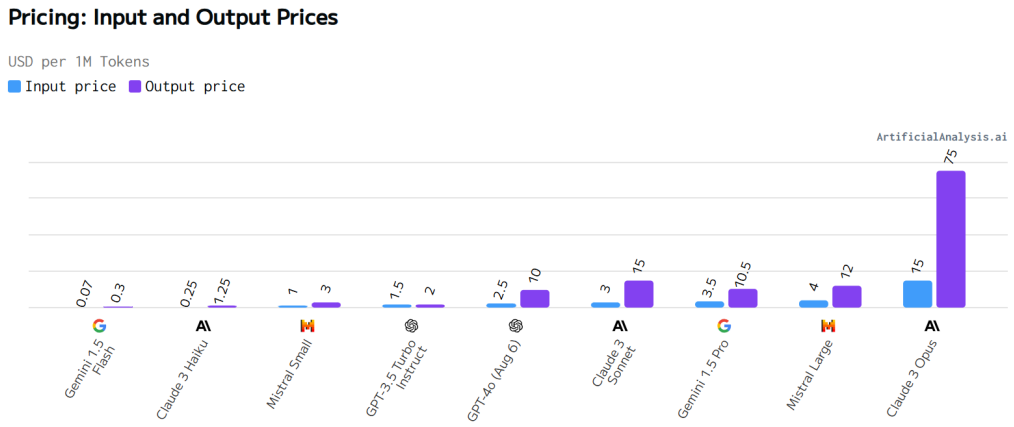

Figura 7.1: Benchmark sobre el precio de los LLMs evaluados. (Artificialanalysis, 2024)

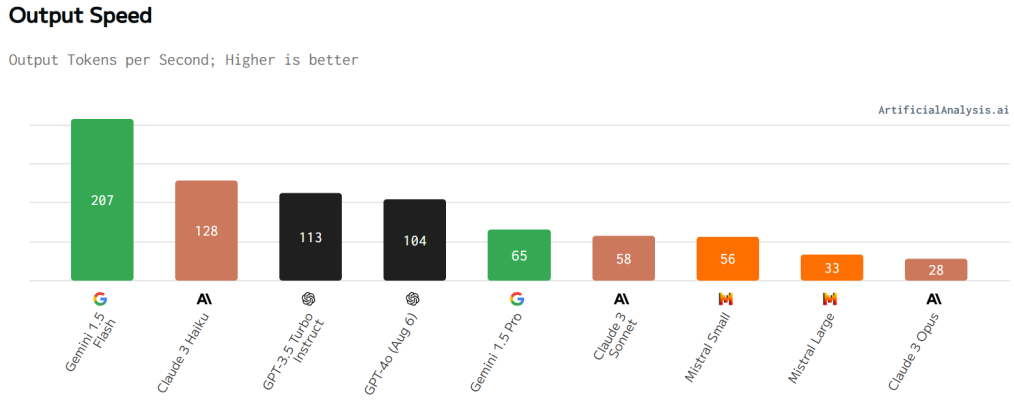

Figura 7.2: Benchmark sobre la velocidad de respuesta de los LLMs evaluados. (Artificialanalysis, 2024)

7.2.3. Modelo Embedder

Este modelo es el encargado de transformar los fragmentos del documento en vectores o embeddings. Este modelo es de alta importancia en el proceso ya que será el que defina la similitud entre los textos. Por tanto es crucial para la recuperación de la información necesaria para dar una respuesta.

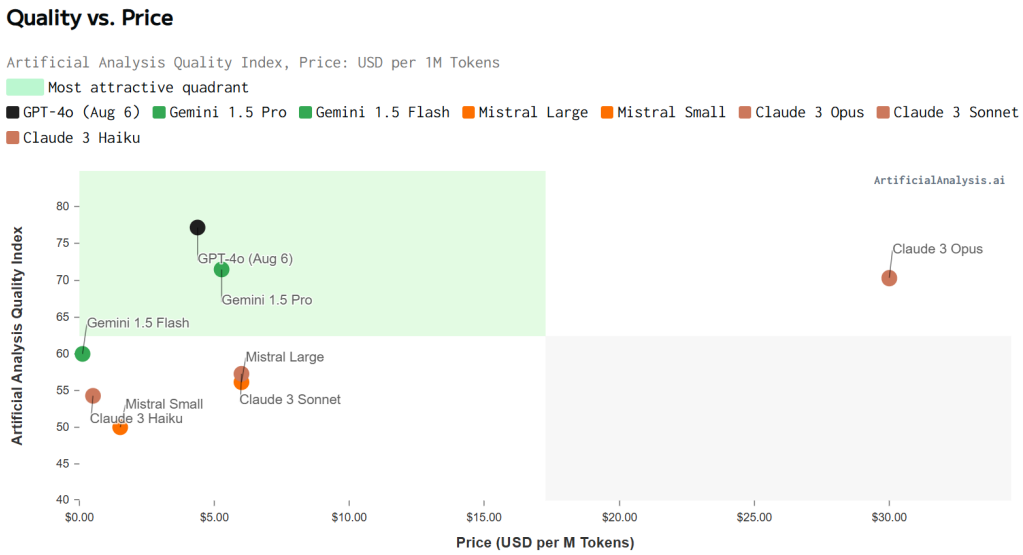

Figura 7.3: Gráfico comparativa calidad contra precio. (Artificialanalysis, 2024)

El modelo embedder ha sido text-embedding-3-small con posición 59 en el leaderboard de huggingface (lea, 2024). Se ha usado este modelo por ser el mejor calidad/precio disponible en Azure OpenAI. Como se puede ver en la figura 7.4 mejora al anterior model ada v2 y se puede ver en la tabla 7.1 como reduce su precio. Por estas dos razones es el modelo seleccionado para el sistema RAG creado.

Eval benchmark	ada v2	text-embedding-3-small	text-embedding-3-large
MIRACL average	31.4	44.0	54.9
MTEB average	61.0	62.3	64.6

Figura 7.4: Gráfico calidad embeddings OpenAI (OpenAI, 2024)

Model	Pricing
text-embedding-3-small	$0.020 / 1M tokens
text-embedding-3-large	$0.130 / 1M tokens
ada v2	$0.100 / 1M tokens

Tabla 7.1: Precio de los embeddings de OpenAI (OpenAI, 2024)

7.3. Análisis del Documento: Constitución Española

7.3.1. Estructura del Documento

La **Constitución Española** es el documento fundamental que establece el marco legal y político de España. Fue ratificada en 1978 y es el texto supremo que rige los derechos, deberes

y la organización de las instituciones del país. El documento está dividido en varios títulos que abordan aspectos clave del sistema político, económico y social de España. Se compone de una introducción (Preámbulo) y 10 títulos, además de disposiciones adicionales, transitorias, derogatorias y finales.

- **Preámbulo**: Establece los principios y objetivos fundamentales de la Constitución, como la justicia, la libertad, la democracia, la protección de los derechos humanos, el progreso social y la promoción de la paz y la cooperación internacional.
- **Títulos**:
  - **Título Preliminar**: Define a España como un Estado social y democrático de derecho y establece la soberanía nacional y la forma política del Estado, que es la monarquía parlamentaria.
  - **Título I**: Regula los derechos y deberes fundamentales, dividiéndose en varios capítulos que incluyen los derechos de los españoles y los extranjeros, así como las libertades y derechos fundamentales.
  - **Título II**: Describe la Corona y las funciones del Rey como jefe de Estado.
  - **Títulos III a V**: Tratan sobre las Cortes Generales, el Gobierno y las relaciones entre ambos poderes.
  - **Título VI**: Regula el poder judicial.
  - **Títulos VII a X**: Abordan la economía y hacienda, la organización territorial del Estado, el Tribunal Constitucional y la reforma constitucional.

7.3.2. Relevancia para el Sistema RAG

El **sistema RAG (Retrieval-Augmented Generation)** que se propone desarrollar utilizará este documento como base de conocimiento para responder preguntas relacionadas con la Constitución Española. Debido a su estructura bien definida y la clara división de temas, es posible fragmentar el documento en secciones más manejables que puedan ser recuperadas de manera eficiente.

7.3.3. Formato del documento

El documento se encuentra tanto en formato PDF como en formato XML. En el formato XML se puede acceder a cada uno de los artículos de la constitución por separado. Se muestra un ejemplo a continuación:

```
</div>
<p class="linkSubir"><a href="https://www.boe.es/buscar/act.php?
id=BOE-A-1978-31229&amp;p=20240217&amp;tn=1#top">Subir</a></p>
<hr class="bloque">
<div class="bloque" id="a3">
```

```
  <p class="bloque">[Bloque 5: #a3]</p>
  <form action="https://www.boe.es/buscar/act.php?id=BOE-A-1978-31229&amp;
  p=20240217&amp;tn=1#" method="get">
    <input type="hidden" value="BOE-A-1978-31229" name="id">
    <input type="hidden" value="5" name="tn">
    <input type="hidden" value="a3" name="bj">
    <input id="btn_jur_a3" type="submit">
    <label for="btn_jur_a3" title="Jurisprudencia">
      <span class="fuera">Jurisprudencia</span>
    </label>
  </form>
  <h5 class="articulo">Artículo 3</h5>
  <p class="parrafo">1. El castellano es la lengua española oficial del Estado.
  Todos los españoles tienen el deber de conocerla y el derecho a usarla.</p>
  <p class="parrafo">2. Las demás lenguas españolas serán también oficiales en las
  respectivas Comunidades Autónomas de acuerdo con sus Estatutos.</p>
  <p class="parrafo">3. La riqueza de las distintas modalidades lingüísticas de
  España es un patrimonio cultural que será objeto de especial respeto y protección.</p>
</div>
```

7.4. Chunkerización del Documento

La chunkerización es un proceso crucial en la creación de sistemas de Recuperación Aumentada por Generación (RAG, por sus siglas en inglés). En este capítulo, se describe cómo se realiza la chunkerización del documento de la Constitución Española para mejorar la eficiencia en la recuperación de la información. Utilizamos diferentes estrategias de chunkerización que se adaptan a la naturaleza del texto y el caso de uso. Para implementar estas estrategias, utilizamos la librería `LangChain`, que ofrece una serie de herramientas avanzadas para manejar documentos de gran tamaño.

7.5. Estrategias de Chunkerización

La chunkerización del documento implica dividir el texto en fragmentos más pequeños y manejables (chunks). Esto mejora la precisión de la recuperación de información al asegurar que los fragmentos estén centrados en temas específicos. A continuación, se explican las tres estrategias de chunkerización empleadas:

7.5.1. Character Text Splitter

El *Character Text Splitter* es una de las técnicas básicas de chunkerización. Divide el texto en fragmentos basándose en un número fijo de caracteres, lo que asegura que los fragmentos

resultantes tengan un tamaño consistente. Esta técnica es útil cuando se requiere dividir grandes bloques de texto de manera uniforme, pero no considera la semántica ni la estructura del texto.

Para esta estrategia, definimos tamaños de chunk de 200, 300 y 400 caracteres, con un pequeño solapamiento entre los fragmentos (10 % del tamaño del chunk), lo que permite preservar el contexto cuando se fragmenta el documento. Este método es sencillo y eficiente para manejar grandes volúmenes de datos.

7.5.2. Recursive Character Text Splitter

El *Recursive Character Text Splitter* es una versión más avanzada del *Character Text Splitter*. Además de dividir el texto basándose en el número de caracteres, este enfoque aplica un análisis recursivo, respetando la estructura del documento al intentar evitar cortar palabras o frases importantes. De esta manera, los fragmentos generados tienen mayor coherencia interna y preservan mejor el contexto semántico, lo que mejora las respuestas generadas en los sistemas RAG.

Al igual que con el splitter básico, se utilizan tamaños de chunk de 200, 300 y 400 caracteres, pero con el beneficio añadido de que los fragmentos resultan más naturales y mejor alineados con las divisiones lógicas del texto.

7.5.3. Chunkerización personalizada (Spanish Article Splitter)

El *Spanish Article Splitter* ha sido desarrollado específicamente para chunkerizar documentos legales en español, como la Constitución Española. Este splitter aprovecha la estructura inherente del documento en XML, basada en artículos, para realizar la chunkerización. Utilizando técnicas de procesamiento de HTML, el splitter identifica los artículos individuales y los divide en fragmentos de texto correspondientes. Cada artículo es tratado como un fragmento separado, preservando su integridad semántica.

Este splitter es particularmente útil en textos legales, donde cada artículo puede ser considerado una unidad independiente y completa de información. Para documentos como la Constitución Española, esta técnica asegura que los fragmentos de texto sean coherentes y estén alineados con la estructura legal del documento.

El pseudocódigo del algoritmo desarrollado se encuentra en el algoritmo 1.

7.5.4. Librería LangChain

La librería usada para esta tarea ha sido `LangChain`. Proporciona una plataforma robusta para la manipulación y chunkerización de documentos. Permite el uso de diferentes tipos de splitters para dividir el texto según las necesidades del usuario. En este caso, se emplearon tanto splitters estándar como personalizados. La flexibilidad de `LangChain` facilita la integración de diferentes enfoques, desde la división básica de texto por caracteres hasta estrategias más complejas como el *Spanish Article Splitter*.

Algoritmo 1: SpanishArticleSplitter

Entrada: Ruta del archivo HTML (*file_path*) ;
Salida: Lista de documentos divididos por artículos (*documents*) ;
Abrir el archivo en la ruta especificada (*file_path*) y leer su contenido como HTML;
Parsear el contenido HTML utilizando un parser compatible con HTML;
Inicializar una lista vacía para almacenar los documentos (*documents*);
Encontrar todos los elementos de encabezado <h5> con la clase articulo;
for *cada <h5> encontrado* **do**
    Encontrar el que contiene el artículo asociado;
    Extraer el nombre del artículo desde el texto del encabezado <h5>;
    Inicializar una lista vacía para los párrafos del artículo;
    for *cada párrafo <p> dentro del * **do**
        if *el texto del párrafo no contiene "Bloque"* **then**
            Agregar el texto del párrafo a la lista;
    Unir todos los párrafos para formar el texto completo del artículo;
    Crear un diccionario de metadatos con nombre del artículo, nombre del archivo y fuente del archivo;
    Crear un documento con el texto del artículo y los metadatos;
    Agregar el documento a la lista *documents*;
Retornar la lista de documentos *documents*;

LangChain también facilita el procesamiento de diferentes formatos de documentos (como PDF y HTML) y su posterior transformación en fragmentos manejables para mejorar la eficiencia en los procesos de recuperación y generación de texto.

En total se obtienen 7 formas de fragmentar el documento que deberán ser evaluadas.

7.6. Evaluación del sistema RAG

Para evaluar la efectividad de la chunkerización en el sistema RAG creado, se utilizó un proceso exhaustivo que involucró la generación y evaluación de preguntas basadas en los fragmentos generados de la Constitución Española. Este proceso de evaluación fue realizado utilizando la librería `LlamaIndex`, que permitió automatizar tanto la generación de preguntas como la medición de métricas clave para determinar la calidad de las respuestas generadas.

Es importante puntualizar que el número de chunks devueltos por el sistema RAG de cara a su evaluación ha sido de 2 chunks.

7.6.1. Generación de Preguntas

El primer paso consistió en generar un conjunto de preguntas para evaluar la precisión y relevancia de las respuestas proporcionadas por el sistema. Para ello, se utilizó el modelo GPT-3.5-turbo, que empleó un *prompt* diseñado para generar preguntas en español a partir de fragmentos de texto extraídos de los documentos chunkerizados. Este conjunto de preguntas

se almacenó en un archivo CSV para su posterior uso en la evaluación.

Se usó el siguiente prompt para la generación de preguntas:

```
Context information is below.
---------------------
{context_str}
---------------------
Given the context information and not prior knowledge,
generate only questions based on the below query.
Generate the questions in Spanish.
{query_str}
```

Questions
¿Cuáles son los valores superiores del ordenamiento jurídico en España?
¿En quién reside la soberanía nacional en España?
¿Cuál es la forma política del Estado español?
¿Cómo se define a España en términos de su Estado?
¿Cuáles son los principios fundamentales de España como Estado?
¿Cuál es el sistema político de España?
¿Qué poderes emanan del pueblo español?
¿Qué tipo de monarquía tiene España?
...

Tabla 7.2: Ejemplo de preguntas generadas

7.6.2. Métricas de Evaluación

Las métricas de evaluación clave empleadas en este proceso se centran en tres aspectos fundamentales para sistemas RAG: fidelidad, relevancia de la respuesta, relevancia del contexto y tiempo de respuesta. Estas métricas se calcularon utilizando el modelo GPT-4o (**modelo juez**), que permitió evaluar de manera automática la calidad de las respuestas generadas para cada pregunta.

- **Fidelidad (FP)**: Mide hasta qué punto las respuestas generadas están respaldadas por el contexto proporcionado. Es decir, se asegura de que las afirmaciones hechas en las respuestas sean consistentes con la información del documento original, minimizando la aparición de alucinaciones. En la implementación de LlamaIndex el resultado será True o False.

- **Relevancia de la Respuesta (RP)**: Evalúa si la respuesta aborda directamente la pregunta formulada. Este criterio no solo considera la factualidad, sino también si la respuesta es completa y libre de información redundante o innecesaria. En la implementación de LlamaIndex el resultado será True o False.

- **Relevancia del Contexto (RCP)**: Mide cuán relevante es el contexto recuperado para responder la pregunta. Un buen contexto debe contener únicamente la información necesaria para proporcionar una respuesta precisa, sin añadir detalles irrelevantes que puedan afectar la calidad de la generación de texto.

- **Tiempo de respuesta (TRP)**: El tiempo en el que el sistema RAG completo da una respuesta a la pregunta del usuario.

Cabe recalcar que ninguna de las métricas usadas necesita la respuesta verdadera a la pregunta ya que analizan la respuesta en base al contexto (los chunks) aportado.

En la tabla 7.3 puede verse un ejemplo de error con la respuesta, sin embargo todas las métricas cumplen su función.

- **Fidelidad**: El resultado es True ya que no hay alucinación en la respuesta. El Agente LLM no tiene en el contexto la información que el usuario solicita por lo que la respuesta es correcta al responder que no puede responder.

- **Relevancia de la Respuesta**: Esta métrica mide la relevancia de la respuesta a la pregunta. En otras palabras si responde la pregunta del usuario. El resultado es True, ya que responde que no puede responder con la información que tiene, pero responde a la pregunta formulada.

- **Relevancia del Contexto**: En esta métrica si podemos comprobar que el contexto no es el adecuado para responder a la pregunta y el modelo juez aporta la explicación de esta métrica.

7.6.3. Evaluación

Se creó una base de datos vectorial sobre chroma utilizando los grupos de documentos chunkerizados anteriormente, y para cada pregunta y grupo de chunks (cada metodología de chunkerización descrita), el sistema calculó las métricas de fidelidad, relevancia de la respuesta, relevancia del contexto y tiempo de respuesta.

El proceso de evaluación se llevó a cabo mediante la generación de respuestas automáticas a las preguntas formuladas. Se calculó el promedio de las métricas para cada metodología de chunks. Finalmente, los resultados de las métricas se almacenaron en un archivo JSON para su análisis posterior.

7.6.4. Resultados

Se presentan los resultados de la evaluación de diferentes técnicas de chunkerización aplicadas al sistema RAG. A continuación, se describen los resultados obtenidos para cada estrategia de chunkerización (véase tabla 7.5 y la figura 7.5):

Field	Content
Question	¿Cuál es el contenido del artículo 28 de la Constitución Española?
Response	El contenido del artículo 28 de la Constitución Española no se encuentra proporcionado en el contexto dado.
Context 1	**Article Name:** Artículo 53 **Text:** Los derechos y libertades reconocidos en el Capítulo segundo del presente Título vinculan a todos los poderes públicos... **Source:** ../data/Constitución Española/BOE-A-1978-31229 Constitución Española.html
Context 2	**Article Name:** Artículo 20 **Text:** Se reconocen y protegen los derechos: a) A expresar y difundir libremente los pensamientos, ideas y opiniones mediante la palabra, el escrito o cualquier otro medio... **Source:** ../data/Constitución Española/BOE-A-1978-31229 Constitución Española.html
Faithfulness	True
Relevancy	True
Context Relevancy Score	0.125
Context Relevancy Feedback	**Feedback:** - The context is related to the Spanish Constitution but does not specifically match Article 28. - **Score:** 0.5/2 - The context does not provide the specific content needed to answer the query. - **Final Score:** 0.5
Response Time	1.253065586090088 seconds

Tabla 7.3: Ejemplo evaluación pregunta 1

- **Spanish Article Splitter**: Esta estrategia produjo un tiempo de respuesta promedio de 1.56 segundos, con una fidelidad promedio de 0.94, una relevancia promedio de 0.94 y una relevancia del contexto de 0.86. Estos resultados muestran que este splitter mantiene un alto nivel de precisión en las respuestas y una relevancia adecuada del contexto, aunque el tiempo de respuesta es ligeramente superior a otras estrategias.

- **Character Splitter (400 caracteres)**: El tiempo de respuesta promedio fue de 1.52 segundos, con una fidelidad de 0.95, relevancia de 0.91 y relevancia del contexto de 0.72. Esta estrategia proporcionó un excelente balance entre fidelidad y relevancia, aunque la relevancia del contexto es menor en comparación con el splitter basado en artículos.

- **Recursive Character Splitter (400 caracteres)**: Este splitter alcanzó un tiempo de respuesta promedio de 1.50 segundos, con una fidelidad de 0.93, relevancia de 0.87 y relevancia del contexto de 0.69. Aunque la relevancia del contexto es menor, esta técnica ofrece un buen equilibrio entre tiempos de respuesta y precisión.

- **Recursive Character Splitter (200 caracteres)**: El tiempo de respuesta promedio fue

Field	Content
Question	¿Cuál es el objetivo de la educación según los principios constitucionales?
Response	El objetivo de la educación según los principios constitucionales es el pleno desarrollo de la personalidad humana en el respeto a los principios democráticos de convivencia y a los derechos y libertades fundamentales.
Context 1	**Article Name:** Artículo 27 **Text:** Todos tienen el derecho a la educación. Se reconoce la libertad de enseñanza... **Source:** ../data/Constitución Española/BOE-A-1978-31229 Constitución Española.html
Context 2	**Article Name:** Artículo 9 **Text:** Los ciudadanos y los poderes públicos están sujetos a la Constitución... **Source:** ../data/Constitución Española/BOE-A-1978-31229 Constitución Española.html
Faithfulness	True
Relevancy	True
Context Relevancy Score	1.0
Context Relevancy Feedback	**Feedback:** - The retrieved context directly addresses the query about the objective of education. - **Score:** 2/2 - The context can be used exclusively to fully answer the query. - **Final Score:** 4.0
Response Time	2.0108368396759033 seconds

Tabla 7.4: Ejemplo evaluación pregunta 2

el más bajo con 1.43 segundos. La fidelidad promedio fue de 0.95, la relevancia de 0.85 y la relevancia del contexto de 0.65. Aunque presenta buenos tiempos de respuesta, la relevancia del contexto es la más baja entre las estrategias evaluadas.

- **Character Splitter (300 caracteres)**: Con un tiempo de respuesta promedio de 1.30 segundos, esta estrategia obtuvo una fidelidad de 0.91, relevancia de 0.83 y relevancia del contexto de 0.67. Aunque el tiempo de respuesta es rápido, las métricas de relevancia están ligeramente por debajo de otras técnicas.

- **Recursive Character Splitter (300 caracteres)**: Este splitter alcanzó un tiempo de respuesta promedio de 1.39 segundos, con una fidelidad de 0.94, relevancia de 0.90 y relevancia del contexto de 0.69. Aunque el tiempo de respuesta es mayor que otras técnicas, ofrece un buen compromiso entre precisión y relevancia de las respuestas.

- **Character Splitter (200 caracteres)**: El tiempo de respuesta promedio fue de 1.36 segundos, con una fidelidad de 0.91, relevancia de 0.82 y relevancia del contexto de 0.65.

Este splitter tuvo uno de los tiempos de respuesta más rápidos, pero mostró una ligera disminución en las métricas de relevancia y contexto.

En general, los resultados muestran que la estrategia *Spanish Article Splitter* ofrece las mejores métricas de fidelidad y relevancia, a pesar de un tiempo de respuesta ligeramente superior. Por otro lado, las estrategias basadas en *Character Splitter* y *Recursive Character Splitter* presentan una relación equilibrada entre tiempos de respuesta rápidos y métricas de calidad aceptables, con la versión de 400 caracteres proporcionando el mejor compromiso entre velocidad y precisión.

Chunkerización	**TRP (s)**	**FP**	**RP**	**RCP**
Spanish Article Splitter	1.56	0.94	0.94	0.86
Character Splitter (400 caracteres)	1.52	0.95	0.91	0.72
Recursive Character Splitter (400 caracteres)	1.50	0.93	0.87	0.69
Recursive Character Splitter (200 caracteres)	1.43	0.95	0.85	0.65
Character Splitter (300 caracteres)	1.30	0.91	0.83	0.67
Recursive Character Splitter (300 caracteres)	1.39	0.94	0.90	0.69
Character Splitter (200 caracteres)	1.36	0.91	0.82	0.65

Tabla 7.5: Resultados de la Evaluación de las Técnicas de Chunkerización

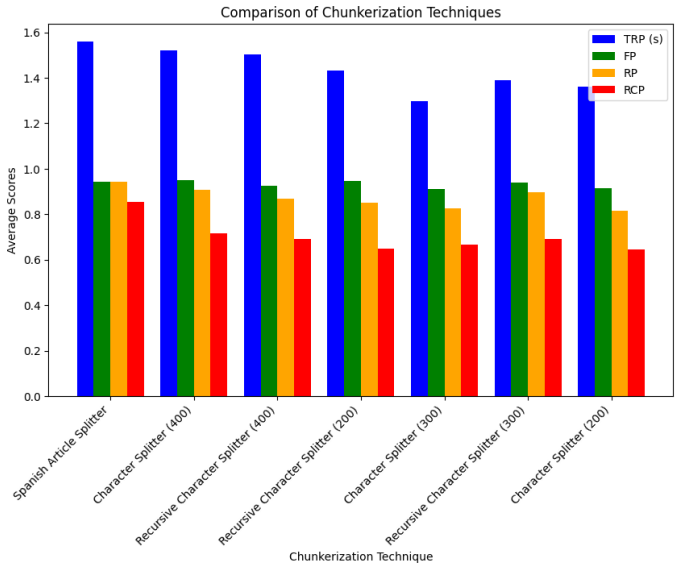

Figura 7.5: Gráfico comparativo de los splitters evaluados

7.7. Creación del agente

Por último, se ha creado el agente mendiante la librería Langchain usando el splitter personalizado para los artículos de la constitución española. Para ello se ha usado un retriever denso descrito en el capítulo 5. Se ha implementado la arquitectura de la figura 6.1.

Para que sea más visual (figura 7.7) y haya una interfaz con la que interactuar con el agente, se ha implementado una app usando la librería Streamlit. En esta interfaz web se pueden hacer preguntas de todo tipo. A continuación se muestran ejemplos de preguntas sobre la interfaz, donde a la vez de ser visualmente más agradable también se muestra un desplegable con el contexto (figura 7.6) utilizado para responder a la pregunta.

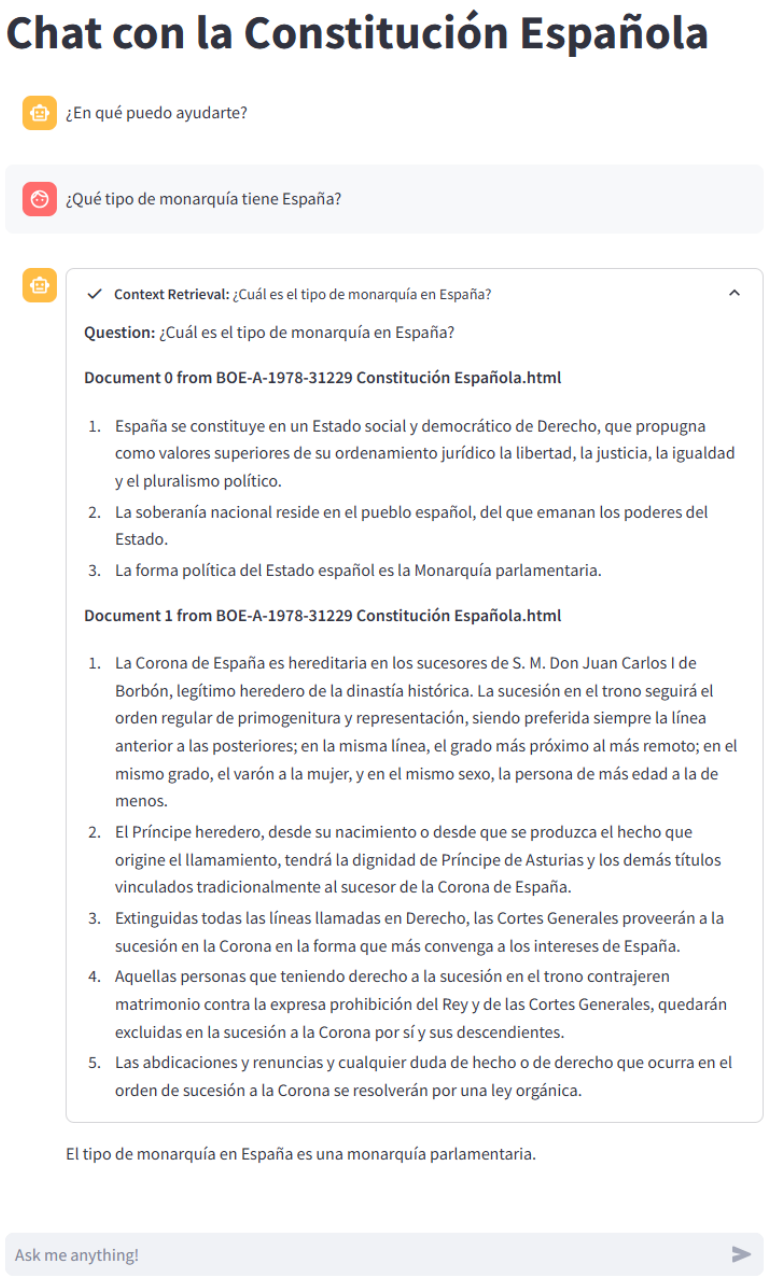

Figura 7.6: Ejemplo de pregunta en Chat con contexto expandido

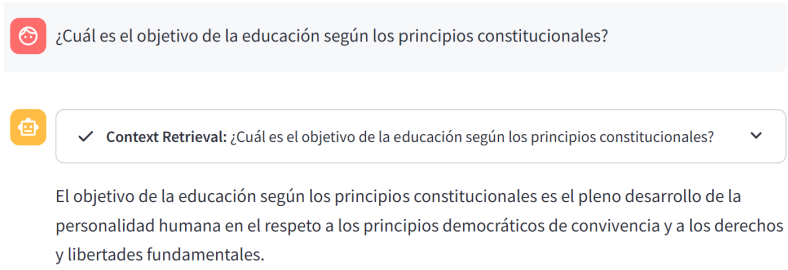

Figura 7.7: Ejemplo de pregunta en Chat

7.7.1. Diagrama Arquitectura

El diagrama de arquitectura de la app desarrollada es el siguiente:

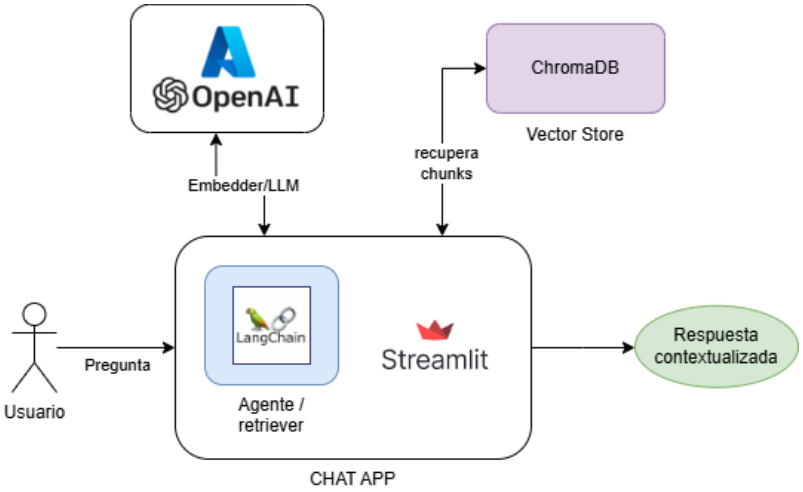

Figura 7.8: Arquitectura desarrollada

En el diagrama se puede ver como las consultas al LLM y al embedder se hacen a través de Azure OpenAI, el vector store utilizado es una instancia de ChromaDB, el agente (retriever) se desarrolla usando el framework Langchain y finalmente la interfaz se ha creado con Streamlit.

7.8. Conclusión

Este capítulo describe la implementación de un sistema RAG para responder preguntas sobre la Constitución Española. Se utilizan modelos LLM como GPT-4o para la evaluación y GPT-3.5-Turbo para generar respuestas, junto con un modelo de embeddings para procesar el texto. Se analizan y prueban varias estrategias de chunkerización del documento, como Character Text Splitter y Spanish Article Splitter, optimizando la recuperación de información.

La evaluación de las técnicas demuestra que el Spanish Article Splitter ofrece la mejor precisión y relevancia del contexto, aunque con un ligero aumento en el tiempo de respuesta. Finalmente, se implementa una interfaz web interactiva para facilitar el uso del sistema, mostrando cómo se puede automatizar la consulta legal de manera eficiente.

Conclusiones

8

Este trabajo ha explorado en profundidad la aplicación de sistemas de generación aumentada por recuperación (RAG) basados en grandes modelos del lenguaje (LLMs) en el contexto de la Constitución Española. Se ha logrado desarrollar un sistema capaz de responder preguntas relacionadas con dicho documento, integrando metodologías avanzadas de chunkerización y técnicas de recuperación de información.

Se compararon diversas estrategias de chunkerización, evaluando su eficiencia y relevancia en la recuperación de fragmentos textuales adecuados para responder preguntas sobre la Constitución Española. Además, se implementó un agente basado en LLMs que emplea técnicas de recuperación y generación para responder consultas de manera precisa y contextualizada.

En resumen, las principales contribuciones de este trabajo son:

1. El desarrollo de un splitter personalizado adaptado para fragmentar artículos de la Constitución Española, optimizando la recuperación de información relevante para su uso en sistemas RAG.

2. La comparación y evaluación detallada de diferentes técnicas de chunkerización aplicadas a textos legales, proporcionando una guía sobre las estrategias más eficaces para este tipo de documentos.

3. La implementación de un agente basado en LLMs que utiliza una arquitectura RAG, integrando recuperación de información y generación de texto para ofrecer respuestas precisas y contextualizadas sobre la Constitución Española.

Aunque existen limitaciones, las perspectivas futuras para la evolución de estos sistemas son prometedoras, especialmente a medida que se avanza en el desarrollo de modelos más potentes y se mejora la capacidad de recuperación de información relevante. Con mejoras continuas, este tipo de herramientas podrían convertirse en recursos fundamentales tanto en el ámbito jurídico como en otras disciplinas que requieran el manejo de grandes volúmenes de información técnica y precisa.

Limitaciones y Perspectivas de Futuro

9

9.1. Limitaciones

A pesar del éxito del sistema RAG desarrollado para la consulta de la Constitución Española, es importante destacar algunas limitaciones que podrían afectar su rendimiento y precisión. Estas limitaciones se dividen en aspectos relacionados con los modelos utilizados, la chunkerización del documento, y la implementación general del sistema.

9.1.1. Modelos LLM

El uso de modelos LLM como GPT-3.5-Turbo y GPT-4o implica una dependencia significativa de la calidad de los modelos subyacentes. Aunque se seleccionaron en base a su relación calidad-precio y rendimiento en benchmarks, estos modelos pueden generar respuestas erróneas o alucinaciones si el contexto proporcionado no es suficiente o si los fragmentos del documento no contienen la información necesaria. Además, su implementación vía API en plataformas en la nube, como Azure, introduce limitaciones relacionadas con la latencia y costos a gran escala.

9.1.2. Contexto y Complejidad Legal

La naturaleza compleja de los documentos legales representa un desafío adicional. La Constitución Española contiene un lenguaje técnico y referencias cruzadas a múltiples artículos y disposiciones. Aunque el sistema puede manejar preguntas directas, su capacidad para interpretar correctamente relaciones complejas o ambigüedades legales es limitada. El sistema RAG no sustituye el análisis detallado de expertos legales, y su uso debe restringirse a consultas básicas o de mediana complejidad.

9.1.3. Escalabilidad

El sistema desarrollado está orientado a responder preguntas sobre un documento específico, en este caso, la Constitución Española. Sin embargo, su capacidad de escalabilidad hacia otros textos legales, o incluso bases de conocimiento más amplias, está limitada por la estrategia de chunkerización aplicada y los recursos computacionales disponibles.

9.2. Perspectivas de Futuro

El desarrollo de este sistema abre numerosas posibilidades de mejora y expansión en el futuro. Las siguientes perspectivas pueden abordar las limitaciones actuales y ofrecer nuevas funcionalidades.

9.2.1. Mejora de Modelos LLM

A medida que evolucionen los modelos de lenguaje, se espera que surjan versiones más avanzadas con una mayor capacidad para comprender el contexto, reducir las alucinaciones y mejorar la precisión en la recuperación de información legal compleja. Incluir futuros modelos LLM con capacidades de razonamiento jurídico y procesamiento de texto legal específico será un paso importante para mejorar el sistema.

9.2.2. Chunkerización Dinámica

Una posible mejora es el desarrollo de un sistema de chunkerización dinámico, que pueda adaptar el tamaño de los fragmentos en función del tipo de consulta realizada. Este enfoque permitiría optimizar la relevancia de los chunks recuperados, asegurando que se proporcione el contexto más adecuado para cada tipo de pregunta, especialmente en textos legales complejos.

9.2.3. Integración con Bases de Conocimiento Jurídico

Ampliar el sistema a una base de conocimiento jurídica más extensa permitiría a los usuarios consultar múltiples normativas, códigos legales o jurisprudencias. Esto requeriría una integración eficiente con bases de datos legales, así como mejoras en la indexación y chunkerización de documentos para asegurar respuestas relevantes y completas en distintos ámbitos legales.

9.2.4. Optimización de la Infraestructura

Optimizar la infraestructura en la nube, o incluso migrar hacia entornos locales con capacidad para manejar modelos de lenguaje avanzados, podría mejorar la eficiencia del sistema. También se podría explorar el uso de soluciones híbridas que combinen el almacenamiento local y en la nube para reducir la latencia y los costos a gran escala.

9.2.5. Interfaz Multiplataforma

Una perspectiva de futuro interesante sería mejorar la interfaz para que sea accesible desde diversas plataformas (móvil, escritorio, web), ofreciendo una experiencia de usuario más fluida y versátil. Esto incluiría mejoras en la visualización del contexto, la capacidad de hacer seguimientos de preguntas anteriores, y la integración de funcionalidades como la búsqueda avanzada de documentos legales.

9.3. Conclusión

El sistema RAG implementado para la consulta de la Constitución Española representa un avance significativo en el uso de tecnologías de procesamiento del lenguaje natural para el ámbito legal.

Apéndize A

Figura relativa al benchmark de LLMs.

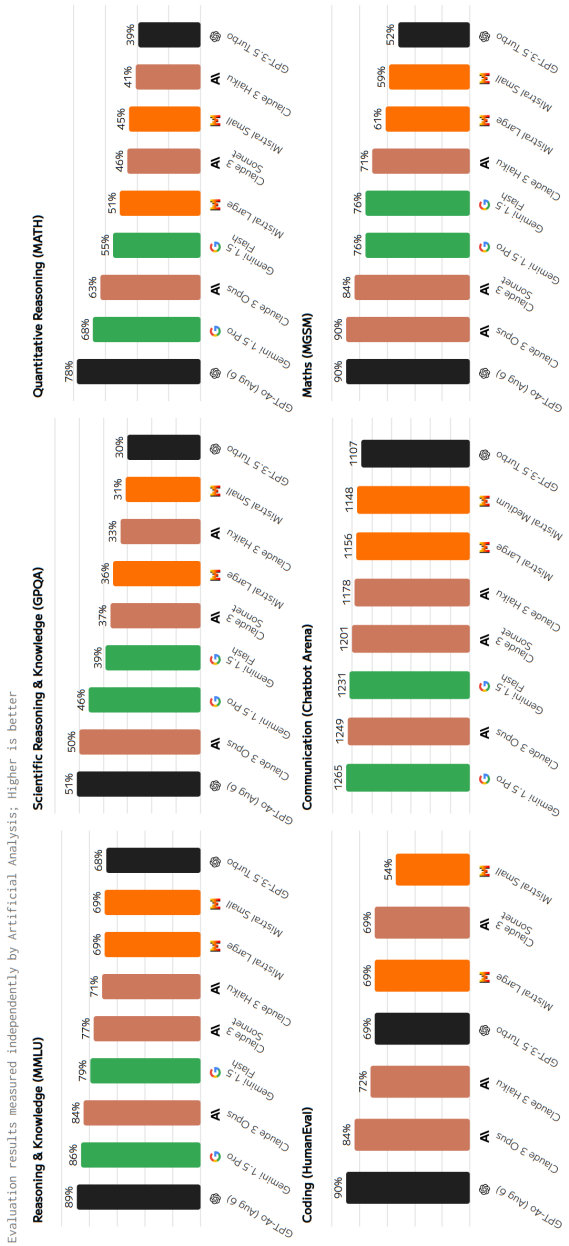

Figura A.1: Benchmark sobre distintas métricas para evaluar LLMs. (Artificialanalysis, 2024)

Bibliografía

(2024). Leaderboard mteb huggingface, https://huggingface.co/spaces/mteb/leaderboard.

Artificialanalysis (2024). Independent analysis of ai models and api providers, https://artificialanalysis.ai/.

Brown, H., Lee, K., Mireshghallah, F., Shokri, R., y Tramèr, F. (2022). What does it mean for a language model to preserve privacy? In *Proceedings of the 2022 ACM conference on fairness, accountability, and transparency*, pages 2280–2292.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

DataCamp (2023). How to improve rag performance: 5 key techniques with examples, https://www.datacamp.com/community/tutorials/improve-rag-performance-techniques.

datastax (2023). Retrieval-augmented generation (rag): A comprehensive guide, https://www.datastax.com/.

Devlin, J., Chang, M.-W., Lee, K., y Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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

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

LangChain (2024). Documentation for retriever modules https://python.langchain.com.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., y Liang, P. (2023). Lost in the middle: How language models use long contexts.

LlamaIndex (2023). Evaluating the ideal chunk size for a rag system using llamaindex, https://www.llamaindex.ai/blog/evaluating-the-ideal-chunk-size-for-a-rag-system-using-llamaindex-6207e5d3fec5.

Nanonets (2023). Building a retrieval-augmented generation (rag) app, https://nanonets.com/.

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Barnes, N., y Mian, A. (2023). A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.

Nvidia (2023). Rag 101: Retrieval-augmented generation questions answered, https://developer.nvidia.com/.

OpenAI (2024). Openai documentation, https://openai.com/.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., y Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Strubell, E., Ganesh, A., y McCallum, A. (2020). Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13693–13696.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., y Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Vectorize.io (2023). Evaluating the ideal chunk size for a rag system, https://www.vectorize.io/blog/evaluating-the-ideal-chunk-size-for-a-rag-system.

Vergadia, P. (2023). The secret sauce of rag: Vector search and embeddings, https://www.thecloudgirl.dev/.

Zhang, Y., Li, Y., Cui, L., Cai, D., Liu, L., Fu, T., Huang, X., Zhao, E., Zhang, Y., Chen, Y., et al. (2023). Siren's song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.