Universidad de Cantabria

Facultad de Ciencias

Análisis y evaluación de LLMs con técnicas RAG para el desarrollo y despliegue de un asistente virtual

(Analysis and evaluation of LLMs with RAG techniques for the development and deployment of a virtual assistant)

Trabajo de Fin de Grado
para acceder al

Grado en Ingeniería Informática

Autor: Ismael Negueruela Gómez
Director: Rafael Duque Medina
Co-Director: Raúl Uriel Vinagre

Julio 2024

Para Victor

Agradecimientos

Quiero comenzar agradeciendo a mi director Rafael Duque por la dedicación mostrada durante todo el periodo de desarrollo de este trabajo. Por cuidar cada detalle y asegurarse de que el resultado final es de alta calidad.

A mis compañeros del grupo de innovación en CIC, por acogerme como uno más desde el primer día y facilitarme cualquier cosa que he necesitado. En especial a Marcos, que se ha preocupado de hacerme entender todo lo necesario para la realización de este proyecto y de solucionarme todos los problemas que me han surgido por el camino.

A mis amigos, por ayudarme a desconectar de la carrera en los momentos en los que sentía que me superaba.

A mi familia, por apoyarme en todas las decisiones que tomé durante estos 4 años y por hacerme ver cuales son las cosas que debemos valorar realmente como personas.

A todos, muchas gracias.

Ismael Negueruela Gómez, Santander, 2024.

Resumen

Actualmente, la Inteligencia Artificial (IA) y los Grandes Modelos Lingüísticos, desempeñan un papel cada vez más importante a la hora de facilitar la interacción entre los usuarios y los sistemas informáticos. En el ámbito empresarial, estos se ofrecen habitualmente como un servicio a través de modelos de negocio como Platform as a Service (PaaS) o sistemas de uso por token. Sin embargo, algunas empresas ofrecen alternativas open-weights, liberando los pesos de sus modelos a la comunidad.

El Trabajo de Fin de Grado (TFG) comenzará haciendo un estudio de estas alternativas y de frameworks para el despliegue y desarrollo de dichos modelos. Fruto del estudio previo, en el que se alimentará un asistente virtual con técnicas de recuperación de información y agregación de conocimiento donde se desplegarán dichos modelos con una configuración común a todos, se realizará la selección de una de las alternativas en base al rendimiento. Posteriormente, se ajustará dicha configuración optimizando el rendimiento del modelo seleccionado. Para la evaluación de los modelos, se tendrá en cuenta tanto el coste de recursos y de tiempo en inferirlo como la precisión de sus predicciones. Estas predicciones serán comparadas con unas respuestas previamente planteadas, fruto del estudio y análisis de un software de la empresa Consulting Informático de Cantabria SL. (CIC), que sirve como caso de uso para este trabajo.

Palabras clave: Inteligencia Artificial, Grandes Modelos Lingüísticos, Generación Aumentada por Recuperación, Interfaz de Programación de Aplicaciones

Abstract

Today, Artificial Intelligence (AI) and Large Language Models (LLMs) are playing an increasingly important role in facilitating interaction between users and computer systems. In the enterprise environment, LLMs are typically offered as a service through business models such as Platform as a Service (PaaS) or tokenised systems. However, some companies offer open-weights alternatives, releasing the weights of their models to the community.

The TFG will start with a study of these alternatives and frameworks for the deployment and development of these models. As a result of the previous study, in which a virtual assistant will be fed with information retrieval and knowledge aggregation techniques where these models will be deployed with a common configuration for all of them, the selection of one of the alternatives will be made based on performance. Subsequently, the configuration will be adjusted to optimise the performance of the selected model. For the evaluation of LLMs, both the cost of resources and time to infer it and the accuracy of its predictions will be taken into account. These predictions will be compared with previously proposed answers, the result of the study and analysis of software from the company Consulting Informático de Cantabria SL (CIC), which serves as a use case for this work.

Keywords: Artificial Inteligence, Large Language Models, Retrieval-Augmented Generation, Application Programming Interface

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

En la era actual, la Inteligencia Artificial (IA) y los LLMs cada vez desempeñan un papel más significativo en la mejora de la interacción entre usuarios y sistemas informáticos. En este contexto, se propone el estudio y desarrollo de una prueba de concepto (PoC del inglés Proof of Concept) de un asistente virtual alimentado por un LLM, con el objetivo de poder integrarlo en el caso de estudio que se explica al final del apartado. Este proyecto aborda la necesidad de mejorar la experiencia del usuario al proporcionar respuestas precisas, inteligentes y relevantes a las preguntas que pueda tener sobre cómo utilizar las diferentes funcionalidades del software del caso de estudio. La meta es ofrecer una asistencia eficiente y efectiva, facilitando así la comprensión y el uso óptimo de la herramienta informática completa donde se integra este asistente.

Los LLMs están ganando cada vez más popularidad tanto en el mundo académico como en la industria, debido a su rendimiento sin precedentes en diversas aplicaciones. A medida que los LLMs siguen desempeñando un papel vital tanto en la investigación como en el uso diario, su evaluación se hace cada vez más crítica, no sólo a nivel de tarea, sino también a nivel de sociedad para comprender mejor sus riesgos potenciales. En los últimos años, se han realizado importantes esfuerzos para examinar los LLMs desde diversas perspectivas [1]. El campo de los asistentes basados en dichos modelos, como los ya conocidos ChatGPT (desarrollado por OpenAI) o Gemini (desarrollado por Google), está experimentando un crecimiento exponencial en los últimos años. Esta evolución ha llevado a una mayor integración de chatbots en diversas aplicaciones y servicios, desde atención al cliente hasta asistentes virtuales personales. Consecuentemente, hay una gran cantidad de investigación en curso sobre cómo mejorar la capacidad de estos asistentes para entender y responder al lenguaje humano. Algunas de las áreas clave incluyen las técnicas de entrenamiento y la integración de conocimiento para conseguir inteligencias eficientes y útiles, y la personalización para adaptarse a las necesidades individuales de los usuarios.

En este TFG, el proyecto se desarrollará en la empresa CIC, dentro de uno de sus equipos de innovación. El objetivo es crear la mencionada PoC para dar pie a una infraestructura de IA que se integrará dentro de sus sistemas en el futuro. Dado que el trabajo se desarrolla dentro del ámbito empresarial, cierto contenido o ciertos detalles sobre el software en el que se ha integrado no pueden ser revelados debido a la confidencialidad. Esto implica que algunas explicaciones no se puedan dar con todo detalle o que haya código desarrollado que no pueda ser mostrado.

1.2. Motivación

El desarrollo de este proyecto está motivado por algunos conceptos que convergen en el objetivo general de mejorar la interacción entre usuarios y sistemas empresariales a través de la implementación de un asistente virtual impulsado por LLMs. Del lado empresarial, algunos de ellos son:

- Mejora de la interacción Usuario-Sistema: El desarrollo de un sistema de chat impulsado por LLMs tiene como objetivo mejorar la interacción entre los usuarios y el software de la empresa. Al dotar al sistema con capacidades avanzadas de Procesamiento del Lenguaje Natural (NLP del inglés Natural Language Processing), se busca facilitar la comunicación, comprensión y respuesta a las consultas de los usuarios de manera más eficiente y efectiva.
- Eficiencia operativa: La implementación de un asistente virtual basado en LLMs puede contribuir a la eficiencia operativa al reducir la carga de trabajo del personal de soporte técnico. Automatizando respuestas a consultas comunes, se liberan recursos humanos para abordar cuestiones más complejas y estratégicas.
- Innovación tecnológica: La integración de tecnologías de vanguardia, como los LLMs, en el entorno de software empresarial representa un paso significativo hacia la innovación.
- Adaptación a las demandas del mercado: La creciente demanda de sistemas inteligentes que simplifiquen la interacción usuario-sistema resalta la importancia estratégica de este proyecto. La capacidad de proporcionar respuestas precisas y contextuales a través de un sistema de chat alimentado por LLMs puede situar a la empresa a la vanguardia de la satisfacción del cliente y la competitividad en el mercado.

La realización de este trabajo también se fundamenta en el propósito de expandir los conocimientos adquiridos a lo largo del grado, centrándose específicamente en el campo del Natural Language Processing (NLP). En un mundo cada vez más digitalizado, la comprensión y mejora de las interacciones entre humanos y máquinas se han vuelto esenciales.

1.3. Asistente virtual

Ante la realidad expuesta en la anterior sección, se expone ahora un visión muy general y sin entrar en detalles sobre el estudio al que se va a hacer frente y el trabajo que se va a desarrollar.

Este proyecto tiene como objetivo principal implementar una Interfaz de Programación de Aplicaciones (API del inglés Application Programming Interface) de preguntas y respuestas utilizando un LLM preentrenado de pesos abiertos (open-weight en inglés). En lugar de entrenar un modelo desde cero, se explorará una variedad de modelos preentrenados disponibles públicamente y se seleccionará aquel que mejor se adapte a las necesidades del proyecto. A lo largo de la evolución de los LLMs, la comunidad opensource ha desempeñado un papel fundamental al proporcionar acceso libre y equitativo a estos modelos. A través de esta liberación, los estudiantes (como es el caso de este proyecto) tienen la oportunidad de acceder a herramientas de última generación, ampliando sus habilidades y preparándose para enfrentar los desafíos tecnológicos del futuro con una base sólida y diversa.

El primer paso será la preparación del conjunto de datos específico para el proyecto. Se recopilará y procesará la guía de usuario del software de CIC para el que se realiza esta Proof of Concept (PoC). Esta guía se utilizará para ajustar el LLM, de modo que pueda comprender y responder preguntas relacionadas con la herramienta en cuestión.

Comenzará entonces la mencionada exploración de las ofertas de modelos y, una vez seleccionado un grupo de ellos, se pasará a su evaluación para determinar cual de todos resulta más beneficioso utilizar. Para ello, se preparará una base de preguntas junto a las correspondientes respuestas que la empresa considera que tiene que dar el modelo.

Una vez seleccionado el modelo adecuado, será el momento de optimizar sus hiperparámetros para sacar su máximo potencial teniendo en cuenta los objetivos que se tiene y la información extra de la que el modelo debe disponer (en este caso, como ya se ha mencionado, la guía de usuario proporcionada por la empresa).

Cuando el modelo esté optimizado, se expondrá a través de una API. Esta permitirá a los usuarios, aunque de una manera muy alejada a un producto final, realizar consultas sobre la guía de usuario del software y recibir respuestas precisas y relevantes generadas por el LLM.

En resumen, este proyecto combina el uso de tecnologías de vanguardia en el campo del NLP y la informática en la nube para desarrollar una herramienta innovadora que facilitará el acceso a la información contenida en la guía de usuario de un software a través de consultas intuitivas y conversacionales.

1.4. Metodología de trabajo

Tras haber expuesto los pasos a seguir en la sección 1.3, se presenta la metodología seguida para el cumplimiento de los mismos. La planificación temporal del trabajo se ha llevado a cabo mediante un diagrama de Gantt, mientras que la creación de tareas específicas para cada objetivo ha sido estructurada mediante la creación de distintos bloques de trabajo.

1.4.1. Organización temporal

A continuación, se presenta el diagrama de Gantt seguido durante el proyecto 1.1. En él se detallan las distintas tareas a realizar durante las distintas semanas de trabajo. También se muestran los hitos del proyecto.

La duración del trabajo ha sido dividida en 18 semanas que abarcan desde el 26 de febrero de 2024, día en el que se elaboró la presente planificación y se comenzó con el análisis, hasta el 28 de junio de 2024. Es necesario aclarar que se usan las semanas del año para la numeración, por lo que el TFG comenzó en la semana 9 de 2024. Las fases del trabajo han sido determinadas en función de los distintos hitos. De este modo, se habrá finalizado todo el proceso de análisis para la semana 10 y toda la fase de desarrollo y evaluación de los resultados se habrá completado para la semana 22. De forma que para la fecha del depósito (finalización del diagrama) solo sean necesario llevar a cabo la revisión del documento y aplicar los cambios pertinentes. En el bloque de trabajo 3 el diagrama se extiende un par de semanas (puede observarse en un tono más claro) debido a que la entrega de la memoria se ha llevado a cabo más tarde de lo inicialmente planteado.

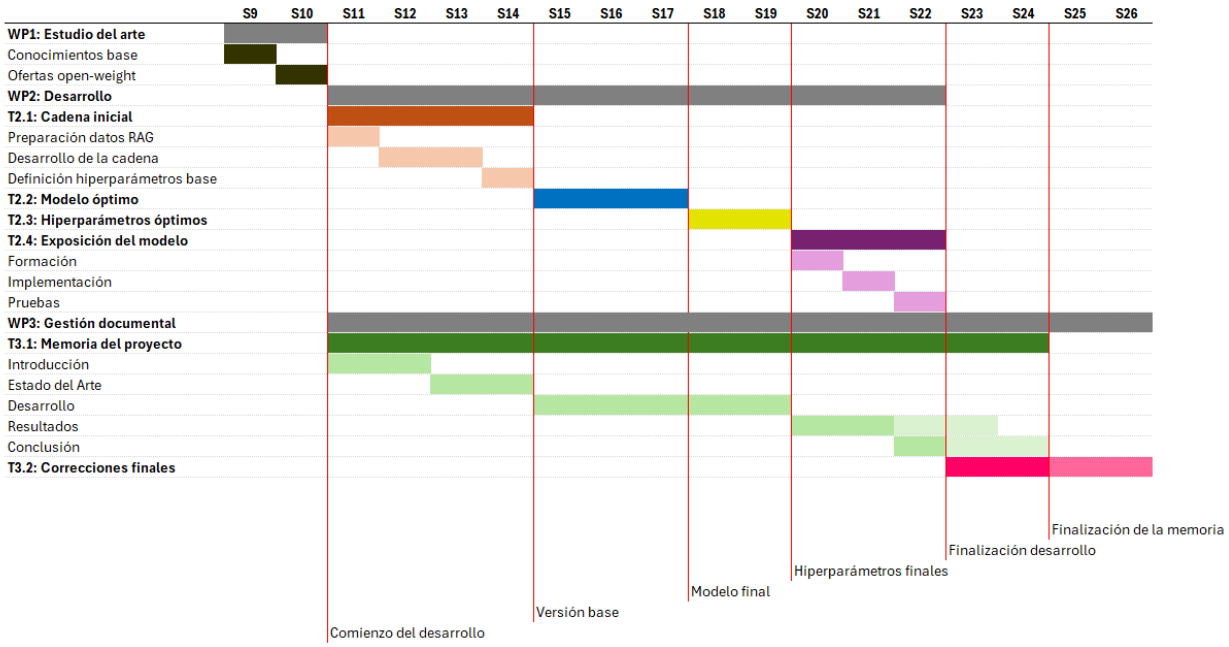

Figura 1.1: Diagrama Gantt

1.4.2. Metodología de tareas

Para el cumplimiento de los objetivos se han dispuesto distintos bloques de trabajo donde se especifican las tareas para alcanzar el objetivo especificado. A continuación, se presentan todos los bloques que se han creado para el desarrollo del proyecto, desglosados en las tablas 1.1, 1.2 y 1.3.

ID	Nombre	Fecha Inicio	Fecha Fin	Duración (Días)
WP1	**Estudio del Arte**: Tareas relacionadas con el estudio general sore el tema a desarrollar y los modelos disponibles.	26/02/2024	10/03/2024	14
T1.1	Conocimientos base: Recopilación y revisión de la literatura existente y materiales relevantes para adquirir una comprensión fundamental del tema.	29/02/2024	03/03/2024	7
T1.2	Ofertas open-weight: Evaluación de las soluciones y tecnologías disponibles que puedan ser aplicadas al proyecto.	04/03/2024	10/03/2024	7

Tabla 1.1: WP1: Estudio del Arte.

ID	Nombre	Fecha Inicio	Fecha Fin	Duración (Días)
WP2	**Desarrollo**: Tareas relacionadas con el desarrollo de la PoC.	11/03/2024	02/06/2024	84
T2.1	Cadena inicial: Preparación datos RAG, desarrollo de la cadena y definición de hiperparámetros base	11/03/2024	07/04/2024	28
T2.2	Modelo óptimo: Ejecución de pruebas iniciales para seleccionar el modelo más adecuado.	08/04/2024	28/04/2024	21
T2.3	Hiperparámetros óptimos: Ajuste de los hiperparámetros para mejorar la precisión y eficiencia del modelo.	29/05/2024	12/05/2024	14
T2.4	Exposición del modelo: Formación en APIs e implementación de una con sus respectivas pruebas.	13/05/2024	02/06/2024	21

Tabla 1.2: WP2: Desarrollo.

ID	Nombre	Fecha Inicio	Fecha Fin	Duración (Días)
WP3	**Gestión Documental**: Tareas relacionadas con la redacción de la memoria y su correcciones.	11/03/2024	30/06/2024	112
T3.1	Memoria del proyecto: Escritura de todos los capítulos que conforman la memoria.	11/03/2024	16/06/2024	98
T3.2	Correcciones finales: Ajustes finales y correcciones al documento de la memoria antes de su entrega.	03/06/2024	30/06/2024	28

Tabla 1.3: WP3: Gestión Documental.

1.5. Estructura de la memoria

Además del capítulo actual, esta memoria tiene otros cuatro capítulos adicionales:

- **Capítulo 2**. El Estado del Arte examina el conocimiento actual en el campo de la IA, centrándose en la relevancia de los modelos fundacionales y su impacto en el desarrollo de LLMs. Además, se analizan los modelos corporativos y los modelos open-weights, destacando sus ventajas y limitaciones. Esta sección proporciona el contexto teórico necesario para comprender las decisiones tomadas durante el desarrollo del proyecto.

- **Capítulo 3**. En el capítulo de Desarrollo, se detalla el entorno de desarrollo utilizado, explicando su integración y utilidad en el proyecto. Se describe la arquitectura general del sistema implementado, especificando cada componente y su función. Además, se exploran las diferentes fases del desarrollo del proyecto: Generación Aumentada por Recuperación (RAG del inglés Retrieval-Augmented Generation), creación y gestión del pipeline, selección del LLM, ajuste de hiperparámetros y la implementación de la API.

- **Capítulo 4**. El capítulo de Resultados y Análisis presenta los resultados obtenidos durante el desarrollo del proyecto. Se incluye un análisis comparativo de los diferentes modelos evaluados, mostrando sus rendimientos y características. Además, se presentan los resultados del ajuste de hiperparámetros y del prompt engineering, destacando su impacto en el rendimiento del modelo. Este capítulo proporciona una evaluación crítica de los logros del proyecto, basada en datos y análisis objetivos.

- **Capítulo 5**. En Conclusiones y Futuros Avances, se resumen los hallazgos del proyecto, destacando su relevancia e impacto potencial en la empresa CIC. Se discuten las lecciones aprendidas y se proponen planes para el desarrollo futuro del sistema de IA. Este capítulo cierra la memoria, ofreciendo una perspectiva sobre cómo los resultados del TFG pueden influir en futuros proyectos y desarrollos en la empresa.

Capítulo 2

Estado del arte

2.1. Grandes Modelos Fundacionales

Los grandes modelos fundacionales (LFM del inglés Large Foundational Model), también conocidos como modelos de IA a gran escala, representan un avance significativo en la evolución en el campo de la IA. Estos sistemas se entrenan con cantidades masivas de datos, estructurados o no, como texto, imágenes y vídeos, lo que les permite realizar una amplia gama de tareas con un alto grado de precisión. Los LFM no se limitan a automatizar tareas repetitivas, sino que aprenden a comprender y generar contenidos, adaptándose a diferentes contextos y dominios. Esta capacidad revoluciona la forma en que construimos sistemas de IA, permitiéndonos abordar problemas intrincados que antes eran inconcebibles. Y lo que es más importante, allana el camino a la IA generativa, un campo con un enorme potencial para revolucionar ámbitos como la educación, la medicina, el marketing o el entretenimiento.

2.1.1. IA generativa

Los Large Foundational Model (LFM) han sentado las bases para una nueva era en la IA, permitiendo una comprensión profunda y una generación avanzada de contenido en una amplia gama de dominios. Entre las innovaciones más destacadas que han surgido se encuentra el gran avance dentro de la IA generativa. Esta rama destaca por su capacidad para no solo comprender y analizar datos, sino también para crear contenido original y coherente dentro de los parámetros aprendidos durante el entrenamiento.

El funcionamiento de la IA generativa se basa en el concepto de aprendizaje e inferencia. Durante el entrenamiento, el modelo se expone a grandes cantidades de datos, estructurados y no estructurados, para aprender las características estadísticas y estructurales que subyacen en ellos. Esto implica ajustar los parámetros del modelo a través de algoritmos de optimización para minimizar la discrepancia entre las predicciones del modelo y los datos reales.

Una vez entrenado, el modelo puede realizar inferencias, es decir, generar nuevo contenido basado en lo que ha aprendido. Esto se logra utilizando las distribuciones de probabilidad aprendidas durante el entrenamiento para generar muestras que sean coherentes y realistas dentro del espacio de datos observado.

Es importante destacar que la IA generativa no simplemente replica los datos de entrenamiento, sino que tiene la capacidad de imaginar y crear nuevas instancias que se ajusten a las características

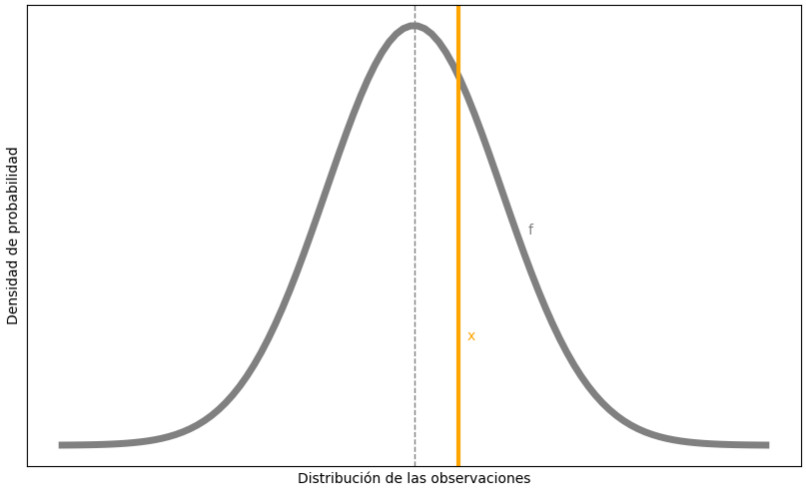

Figura 2.1: Inferencia de los modelos [18].

aprendidas. Con la figura 2.1 como apoyo, podemos entender que una predicción o generación de un modelo no es más que un valor próximo a la media de la distribución aprendida, y por definición de probabilidad, dada una distribución de probabilidad continua f, la probabilidad de obtener un valor $x \in \mathbb{R}$ es $f(x) = 0$, por lo que asumiendo x como un valor observado en la muestra de entrenamiento, nunca tendremos una predicción, o generación, exactamente igual a x, aunque si puede ser muy cercana.

Dentro del campo de la IA generativa diversas arquitecturas han marcado hitos significativos en su evolución, pero ha sido la arquitectura Transformer la que ha supuesto una revolución en este campo, sobre todo en el área del NLP. Esta arquitectura se caracteriza por su innovador mecanismo de atención, demostrando un rendimiento excepcional en una variedad de aplicaciones, consolidando así su posición como una de las arquitecturas más influyentes en la IA generativa.

2.1.1.1. Transformers

Hasta 2017, la arquitectura principalmente usada para el entrenamiento de modelos de NLP eran las Redes Neuronales Recurrentes (RNN del inglés Recurrent Neural Network). Para una oración, por ejemplo, estas redes procesan cada palabra secuencialmente, combinando la nueva palabra de entrada con la información almacenada sobre las anteriores. Esta idea tiene una gran desventaja: al finalizar el procesamiento de la oración completa, las palabras del comienzo han perdido peso en relación con las últimas. Fue con la publicación de Vaswani et al.[2] donde se introdujo por primera vez la arquitectura Transformer y el panorama de aquel momento cambió por completo.

A diferencia de las Redes Neuronales Convolucionales (CNN del inglés Convolutional Neural Network) o las ya mencionadas RNN, las Transformers no dependen de una estructura secuencial. En cambio, utilizan un proceso de autoatención para procesar secuencias de entrada. La idea central detrás de ellas es la capacidad de modelar relaciones a largo plazo entre palabras en una oración sin perder la información contextual. Esto las hace adecuadas para tareas de NLP como traducción automática, resumen de texto y generación de lenguaje natural.

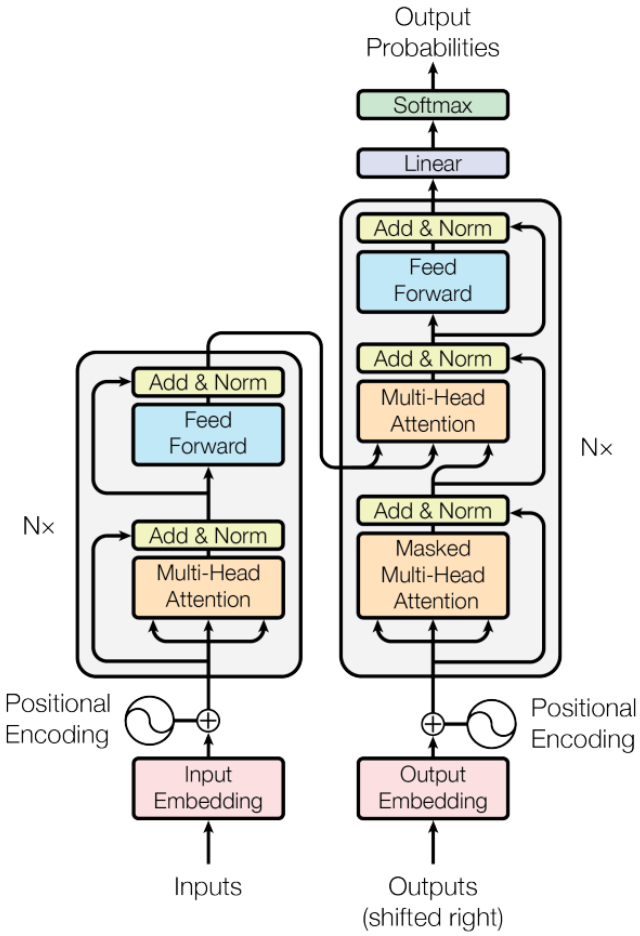

Figura 2.2: Transformer Architecture [2]

El componente central de una Transformer, como se puede observar en la figura 2.2, son las capas de atención. Estas capas calculan una matriz de puntuaciones de atención que pondera la importancia de cada palabra en función de todas las demás palabras en la secuencia. Son Convolutional Neural Network (CNN)s que se aplican al vector de contexto para obtener una representación más profunda y abstracta. Las capas de atención y las capas de alimentación hacia adelante se repiten en múltiples capas para capturar relaciones a diferentes niveles de abstracción. Como podemos ver en la figura 2.3, para una sentencia, los mecanismos de atención calculan las relaciones de cada una de las palabras con el resto dando mayor peso a las más relevantes en su contexto.

En este proceso, los vectores *key* y *query* juegan un papel crucial. Los vectores *query* representan las consultas sobre la información de entrada, indicando qué partes de la secuencia son relevantes para la tarea en cuestión. Por otro lado, los vectores *key* actúan como claves que permiten recuperar información relevante en la secuencia de entrada. Durante el cálculo de la atención, los vectores *query* se comparan con los vectores *key* para determinar su similitud y, por lo tanto, su importancia relativa para la consulta actual. Esta interacción entre vectores *query* y *key* es esencial para que el modelo pueda enfocarse en partes específicas de la secuencia de entrada que son relevantes para la tarea en cuestión.

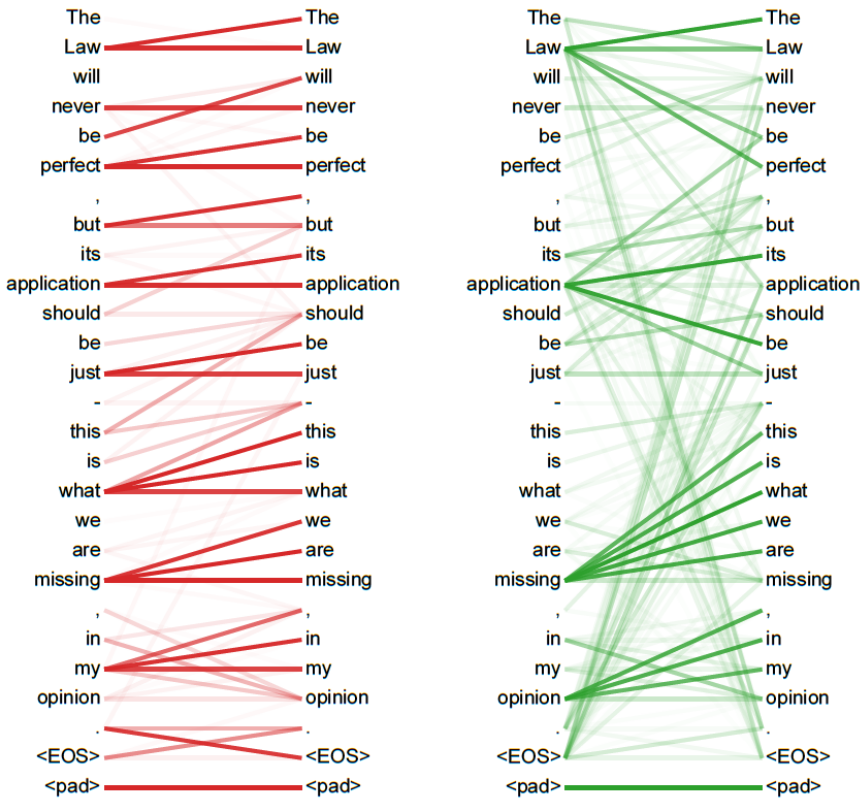

Figura 2.3: Atención en diferentes capas [2]

Un último componente fundamental de las redes Transformers es el *positional encoding* o codificación posicional. Dado que las capas de atención en las Transformers no tienen en cuenta la posición absoluta de las palabras en la secuencia, es necesario incorporar información sobre la posición de cada palabra de manera explícita. El *positional encoding* se suma al embedding de cada palabra para proporcionar al modelo información sobre su posición relativa en la secuencia. Usualmente, se utilizan funciones trigonométricas como senos y cosenos para codificar la posición de manera continua. Esto permite al modelo capturar patrones de posición a lo largo de la secuencia y comprender mejor la estructura y el contexto del texto. Podemos ver un ejemplo en la figura 2.4. En esencia, los positional encoders son fundamentales para que el modelo pueda capturar no solo el significado de las palabras, sino también su posición dentro de la secuencia.

La capacidad de procesar simultáneamente todas las palabras de una sentencia sin recurrencia entre ellas que tienen las redes Transformer permite el entrenamiento paralelizado de los modelos basados en esta arquitectura sobre Unidades de Procesamiento Gráfico (Graphics Processing Unit (GPU) por sus siglas en inglés). Esta característica ha sido fundamental para los notables avances en el campo del procesamiento del lenguaje natural en los últimos años, ya que hemos podido aprovechar plenamente el potencial del hardware disponible.

2.2. Grandes Modelos Lingüísticos (LLMs)

Los LLMs son una categoría de LFM entrenados sobre inmensas cantidades de datos que los hacen capaces de comprender y generar lenguaje natural y otros tipos de contenidos para realizar una amplia gama de tareas [16].

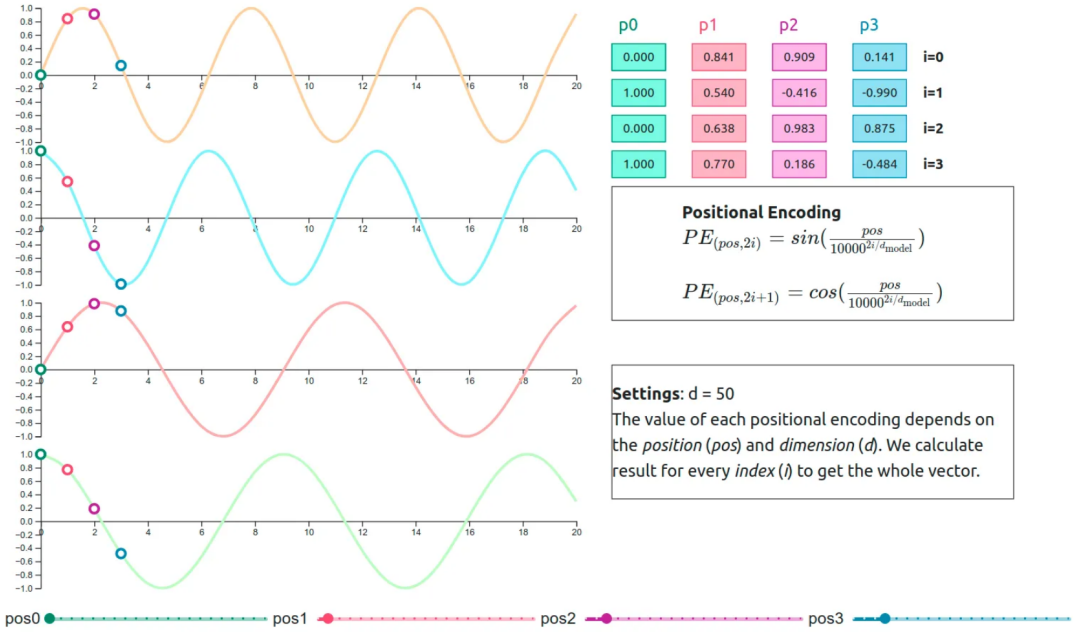

Figura 2.4: Positional- Encoding [3]

En el ámbito empresarial, los LLM se ofrecen como un servicio a través de modelos de negocio como PaaS o sistemas de uso por token. Esto implica que las empresas pueden acceder a la capacidad de estos modelos mediante suscripciones o pagos por uso, lo que les permite integrar capacidades de procesamiento de lenguaje natural en sus productos y servicios de manera eficiente y escalable.

Sin embargo, existen alternativas de pesos abiertos igualmente atractivas en el mundo de los LLMs. Estas alternativas permiten a los usuarios desplegar estos modelos en su propia infraestructura, lo que les brinda un mayor control sobre la privacidad de los datos y la configuración del sistema. Además, tienen la libertad de modificar y adaptar el modelo según sus necesidades específicas, lo que ofrece una opción más transparente y adaptable a las necesidades individuales o empresariales. Al optar por soluciones de este formato, las organizaciones pueden aprovechar al máximo la potencia de los LLM mientras mantienen un control más directo sobre sus datos y procesos. Esto puede ser especialmente relevante en sectores donde la privacidad y la seguridad de la información son de suma importancia, como la salud o las finanzas.

En las siguientes secciones, se plasmarán los conocimientos adquiridos a través del estudio de varios modelos propuestos desde CIC, siendo los modelos open-weights posibles alternativas a utilizar. Antes, es necesario explicar algunas de las características técnicas que se van a mostrar:

- **Parámetros**: Pesos (valores que se multiplican con las entradas en cada neurona) y sesgos (valores adicionales que se suman después de aplicar los pesos) en las capas de la red neuronal.
- **Ventana de contexto**: Cantidad de tokens (palabras, subpalabras, o caracteres, dependiendo del tokenizador) que el modelo puede considerar simultáneamente al generar respuestas o hacer predicciones.
- **Precisión**: Cantidad de bits utilizados para representar los números en los cálculos del modelo. Es común utilizar float32 para la fase de entrenamiento y float16 para la fase de inferencia, la que ocupa en este caso. Alguno de los modelos que se van a comentar usan bfloat16, un tipo

de precisión que usa los mismos bits para el exponente que float32 (8) pero solo 7 bits para la mantisa en lugar de los 10 de float16. En definitiva, esto supone ampliar el rango de valores perdiendo precisión decimal, lo que en ciertos casos en Machine Learning (ML) es beneficioso. Se puede observar mejor esta comparación en la figura 2.5.

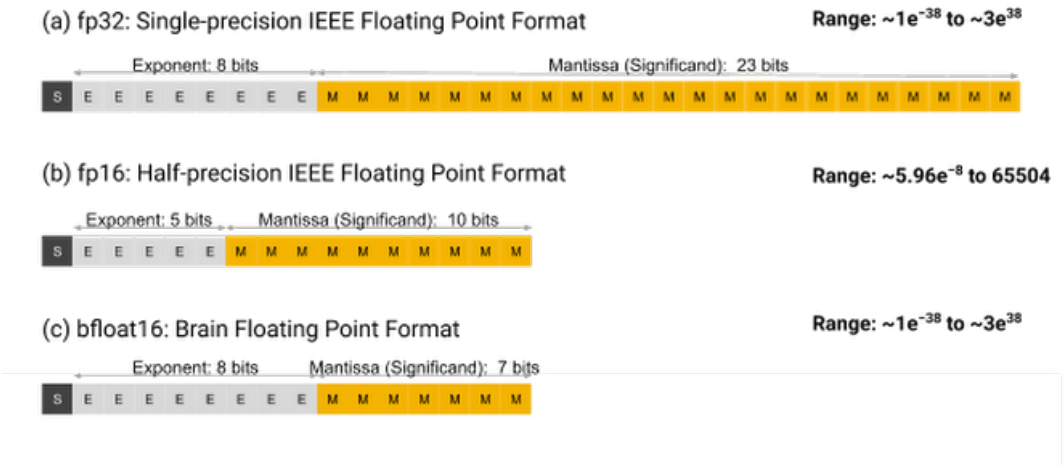

Figura 2.5: Formatos de precisión [4]

2.2.1. Modelos corporativos

Comenzaremos centrándonos en los modelos corporativos, que son desarrollados por empresas privadas y están disponibles a través de plataformas en la nube a cambio de una tarifa. Estos modelos, entre los cuales se incluyen notables exponentes como GPT-4, GPT-3.5 y Gemini, son representativos de la última vanguardia en el desarrollo de LLMs. A lo largo de esta sección, vamos a ver en que destaca cada uno de ellos, además de sus capacidades y contribuciones específicas en el ámbito de los modelos de lenguaje.

2.2.1.1. Generative Pre-trained Transformer (GPT)

GPT, desarrollado por OpenAI, es el modelo de lenguaje basado en Transformers por excelencia. Dado que las primeras versiones, tanto GPT como GPT-2, tenían dificultades para mantener coherencia y generar texto similar a los humanos, no fue hasta la salida de GPT-3 y su integración en ChatGPT cuando realmente esta serie de modelos se asentó, ya no sólo en el nicho del NLP, sino también como complemento al desarrollo en todas las ramas de la informática y otros campos como las finanzas o la educación. Destaca por su enorme escala, con 175 mil millones de parámetros, aproximadamente 116 veces mayor a su predecesor. Otros detalles técnicos son una ventana de contexto de 2048 tokens y precisión de 16 bits en sus parámetros. Teniendo en cuenta que cada parámetro ocupa 2B de espacio, este modelo requiere aproximadamente 350GB de espacio en memoria.

Con su sucesor, GPT-3.5, se han logrado mejoras significativas en términos de rendimiento y precisión. Un gran avance para ello fue duplicar la ventana de contexto a 4096 tokens. Además, GPT-3.5 ha demostrado una mayor versatilidad al abordar tareas específicas, mostrando

una comprensión más profunda de diversos dominios y contextos, a pesar de seguir sufriendo alucinaciones ocasionales y respuestas problemáticas.

Más sorprendente es la capacidad del siguiente modelo, GPT-4, que asciende a 1760 millones de parámetros de aprendizaje, a lo que hay que sumar una ventana de contexto de 32768 tokens. Modelos de este tamaño implican un coste computacional en cada inferencia que suponen más gasto a la empresa y más espera al usuario. Por este motivo, OpenAI basa de este modelo en lo que se conoce como Mixture of Experts (MoE), una arquitectura de ML propuesta hace más de 20 años que combina múltiples modelos especializados (expertos) para manejar diferentes aspectos de una tarea. Estos expertos son ponderados y combinados por una red de control para producir una salida final [5]. De esta manera, GPT-4 se conforma por 16 expertos de 110 millones de hiperparámetos cada uno, activando los dos más relevantes en cada inferencia.

Otra característica que diferencia a GPT-4 de su predecesor es la multimodalidad. Esto significa que no solo procesa palabras, sino que también comprende imágenes identificando objetos en ellas. Todos estos avances provocan mejoras realmente sustanciales en numerosos benchmark con respecto al modelo precursor, como se puede observar en la figura 2.6.

El último modelo de esta gama, GPT-4o, acepta como entrada cualquier combinación de texto, audio, imagen y vídeo y genera cualquier combinación de salidas de texto, audio e imagen, yendo un paso mas allá en la multimodalidad. En lugar de entrenar un modelo para cada tarea, se entrenó un único modelo nuevo de extremo a extremo para texto, visión y audio, lo que significa que todas las entradas y salidas son procesadas por la misma red neuronal [6].

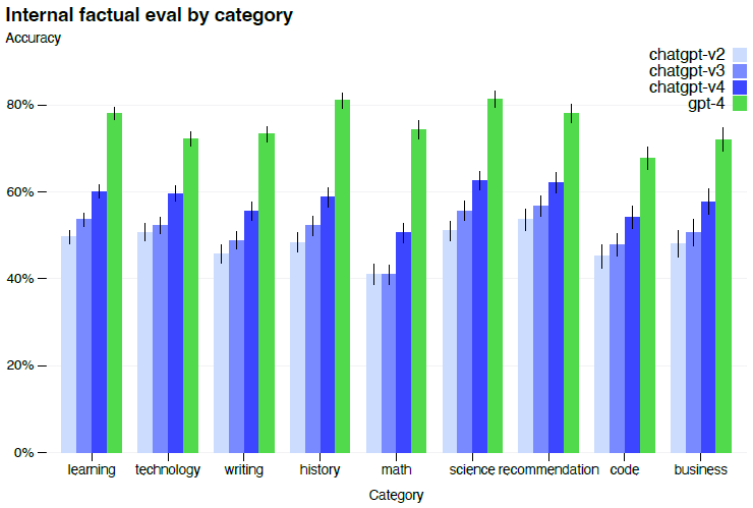

Figura 2.6: Comparación de GPT-4 con GPT-3.5 en diferentes versiones de ChatGPT [7]

2.2.1.2. Gemini

Gemini 1.0 es un modelo de lenguaje desarrollado por Google AI, que se destaca por su capacidad para acceder y procesar información en tiempo real a través de la Búsqueda de Google. A diferencia de otros LLM, que se entrenan con conjuntos de datos estáticos, Gemini está en constante evolución, aprendiendo y adaptándose a las nuevas tendencias y al conocimiento en constante cambio del mundo real. Además, Gemini aprende de forma continua a partir de las interacciones con los usuarios, lo que le permite mejorar su rendimiento y ofrecer una experiencia más personalizada.

Gemini 1.5 Pro es un modelo disperso basado en MoE de Transformers construido con los avances

de investigación y capacidades multimodales de Gemini 1.0. En su technical report, Google presume de lograr una recuperación casi perfecta (>99,7 %) en hasta 1M de tokens en todas las modalidades, es decir, texto, vídeo y audio. Y aún más, consigue mantener este rendimiento cuando se amplía a 10 millones de tokens en la modalidad de texto [8]. La figura 2.7 pone en contexto las desorbitadas cifras que se están contemplando, apuntando no solo los 10 millones de tokens de texto, sino que también incluye 2.8 millones de video y 2 millones de audio.

Este modelo dispone de una versión open-weight, Gemma, que sera objeto de evaluación en este TFG.

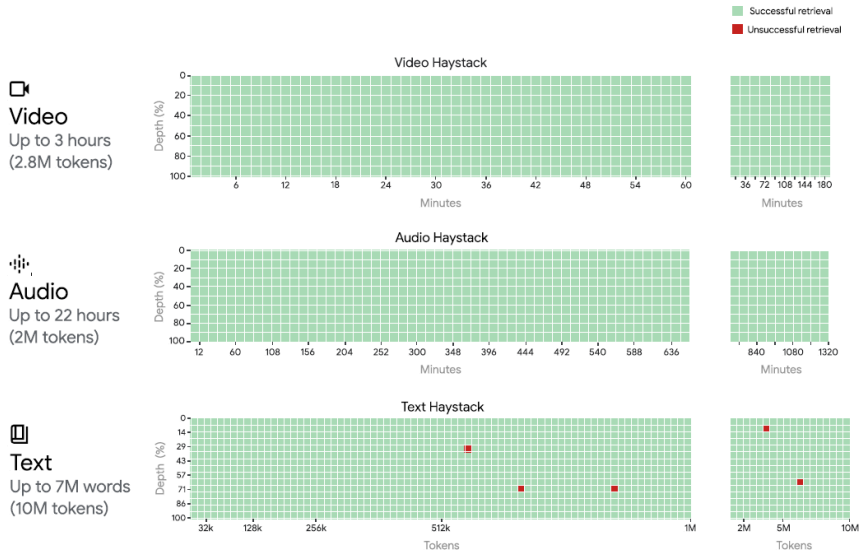

Figura 2.7: Éxito de Gemini 1.5 Pro con diferentes ventanas de contexto [8]

2.2.1.3. PaLM-2

PaLM (Pathways Language Model) es un modelo de lenguaje desarrollado por Google, con un legado de investigaciones innovadoras en aprendizaje automático e IA responsable. Debe su nombre al sistema de ML que se usó para entrenarlo, Pathways, que permite un entrenamiento altamente eficiente a través de múltiples TPU Pods (un material polimérico conectados por interfaces de red dedicadas de alta velocidad) [10]. Entre sus características técnicas 540 mil millones de parámetros, precisión de 16 bits y una ventana de contexto de 2048 tokens. Existen dos versiones notables: una llamada Med-PaLM, que fue el primer modelo en obtener una calificación aprobatoria en las preguntas sobre licencias médicas de los Estados Unidos respondiendo con precisión tanto a las preguntas abiertas como de opción múltiple, y PaLM-E, un modelo de lenguaje de visión de última generación que se puede usar para la manipulación robótica.

Un último modelo de este tipo a destacar es PaLM-2, un modelo lingüístico con mejores capacidades multilingües y de razonamiento, y más eficiente desde el punto de vista computacional que su predecesor PaLM [9], que se se fundamenta en tres avances cruciales en modelos de lenguaje. En primer lugar, destaca por su enfoque de Escalamiento Óptimo de Cálculos, una técnica que ajusta de manera proporcional tanto el tamaño del modelo como el conjunto de datos de entrenamiento. Además, PaLM-2 ha optimizado su capacidad de aprendizaje mediante una Mezcla de Conjuntos de Datos Mejorada, incorporando una gama más amplia y diversa de datos, que incluyen cientos de idiomas humanos y de programación, ecuaciones matemáticas, artículos científicos y páginas web.

Por último, la actualización de la Arquitectura y Objetivo de PaLM-2 lo distingue al presentar una estructura mejorada y ser entrenado en diversas tareas, permitiéndole abordar y comprender diferentes aspectos del lenguaje de manera más precisa y versátil. Hay un versión experta en aplicaciones de seguridad llamada Sec-PaLM que utiliza conocimientos de amenazas de seguridad, incluida la visibilidad de Google en el panorama de amenazas y la inteligencia de primera línea de Mandiant sobre vulnerabilidades, malware, indicadores de amenazas y perfiles de actores maliciosos.

2.2.2. Modelos open-weights

En el contexto de IA, open-weights significa que los pesos del un modelo entrenado son accesibles para que otros los utilicen, analicen o ajusten, aunque el código fuente del modelo en sí no necesariamente sea abierto. No confundir con el concepto opensource, más generalizado en la rama de la informática y que se refiere a software cuyo código fuente está disponible públicamente y puede ser modificado y distribuido por cualquiera.

2.2.2.1. LLaMA

LLaMA es un modelo desarrollado por Meta que ha sido entrenado en un conjunto diverso de datos de diálogo y se ha convertido en un modelo de lenguaje open-weight ampliamente utilizado. Su sucesor, LLaMA-2, presenta mejoras significativas con la primera versión. Fue entrenado con un 40 % más de datos y tiene el doble de longitud de contexto. Existen varios modelos con distintos tamaños, desde 7 mil millones a 70 mil millones de parámetros, con una precisión de 16 bits (ocupando desde 14GB en su versión más ligera hasta 140GB en su versión completa) y todos ellos con una ventana de contexto de 4096 tokens [11]. Se ajustó utilizando más de un millón de anotaciones de preferencias humanas para garantizar su utilidad y seguridad. LlaMA-2 supera a otros modelos de código abierto tanto en conjuntos de datos de comprensión del lenguaje natural como en enfrentamientos directos. Está disponible tanto para investigación como para casos comerciales. Su última versión, LLaMA-3, ha sido uno de los modelos sujetos a evaluación en este trabajo, aunque debido a ser un LLM que se ha publicado recientemente su implantación todavía genera problemas para llegar a los resultados que se le presuponen.

2.2.2.2. Mistral

Mistral es un modelo de lenguaje corporativo que fusiona elementos de LLaMA y LLaMA-2. Tiene una versión llamada Mixtral 8x7B, cuyo diseño se basa en la combinación de múltiples enfoques lingüísticos, lo que le permite abordar una amplia gama de tareas. Se trata de de un modelo basado en una red dispersa de MoE y de sólo decodificación en el que el bloque feedforward elige entre un conjunto de 8 grupos distintos de parámetros, cada uno de 7 mil millones y una precisión de 16 bits, por lo que el modelo ocupará aproximadamente 112GB de memoria, además de una ventana de contexto de 32,768 tokens [12].

Desarrollado por la empresa europea Mistral AI, se caracteriza por su eficiencia y accesibilidad. Al ser un modelo abierto, permite que cualquier persona pueda acceder a su funcionamiento interno y adaptarlo a sus necesidades específicas. Además, su tamaño relativamente pequeño lo hace ideal para implementaciones en dispositivos con recursos limitados. Destaca principalmente en tareas de programación y matemáticas en relación con otros modelos.

2.2.2.3. Falcon

Falcon es un LLM creado por el Instituto de Innovación Tecnológica en Abu Dhabi y se presenta como una nueva familia de modelos de lenguaje de vanguardia. Una característica destacada de su variante Falcon-40B es que cuenta con 40 mil millones de parámetros. Aunque es relativamente más pequeña que LLaMA, que posee 65 mil millones de parámetros, Falcon-40B se posiciona como el primer modelo "verdaderamente abierto" con capacidades que rivalizan con muchos modelos de código cerrado actuales.

El modelo más grande, Falcon-180B, se ha entrenado con más de 3,5 billones de tokens de texto, la mayor prueba de preentrenamiento documentada abiertamente hasta el momento de su salida [13]. Además cuenta con una venta de contexto de 2048 tokens y una precisión de 16 bits, ocupando aproximadamente 360GB en memoria.

En términos de rendimiento, Falcon ofrece un equilibrio entre tamaño y potencia, lo que lo convierte en una opción atractiva para investigadores y desarrolladores interesados en modelos de lenguaje de alto nivel.

2.2.2.4. Otros modelos

Además de los modelos mencionados anteriormente, existe una gran variedad de alternativas también open-weight. Un ejemplo es BERT (Bidirectional Encoder Representations from Transformers), el cual es un modelo de lenguaje preentrenado en grandes cantidades de texto y luego se ajusta para tareas específicas mediante afinado. Un caso de esto es OPT, un modelo basado en BERT diseñado específicamente para tareas de NLP en el dominio biomédico. También podemos destacar MPT (Multimodal Pretrained Transformer), un modelo que puede procesar imágenes y texto simultáneamente, lo que lo hace ideal para tareas como descripción de imágenes y generación de subtítulos; o BLOOM, que se centra en la eficiencia y la escalabilidad, utilizando técnicas de compresión y representación de texto para lograr un equilibrio entre precisión y velocidad, lo que lo hace adecuado para grandes volúmenes de texto.

En la tabla 2.1, se pueden observar los resultados para distintas pruebas o benchmarks para versiones afinadas de los modelos tratados.

Model	Average	ARC	MMLU	TruthfulQA	Winogrande
LLaMA 2-70B	**72.91**	73.12	70.77	64.65	83.11
Mixtral-8x7B	**73.39**	74.91	70.08	66.88	81.69
Falcon-180B	**67.97**	69.45	70.05	45.47	86.9
OPT-30B	**49.21**	43.86	41.09	38.16	73.72
MPT-7B	**49.42**	47.35	42.58	36.65	71.11
BLOOM-176B	**48.28**	50.43	30.85	39.76	72.06

Tabla 2.1: Comparación de modelos open-weight en diferentes benchmark [14].

Estos benchmark evalúan con distinto criterio el rendimiento de los LLMs, puntuándolo en un rango de valores entre 0 y 100 (un valor mayor indica mejor rendimiento). A continuación, una breve explicación de cada uno:

- Corpus de Razonamiento Analógico (ARC del inglés Analogical Reasoning Corpus): Consiste en un conjunto de problemas de analogía donde se le presenta al LLM una analogía de origen

y una palabra objetivo. Su tarea es identificar la palabra objetivo que completa la analogía de la misma manera que la cuarta palabra completa la analogía de origen.

- Comprensión Multitarea Masiva del Lenguaje (MMLU del inglés Massive Multitask Language Understanding: Conjunto de pruebas que abarcan 57 temas diferentes abarcando desde conocimientos básicos hasta niveles profesionales avanzados, diseñado para evaluar la comprensión general del lenguaje.
- TruthfulQA: Evalúa la eficacia de un LLM para evita generar respuestas falsas que haya aprendido de los datos de entrenamiento.
- Winogrande: Conjunto de datos a gran escala con miles de oraciones donde se le presenta al LLM una oración con dos posibles finales ambiguos. La ambigüedad surge debido a la existencia de un pronombre cuya referencia no es clara sin la comprensión del contexto o el uso del sentido común característico de los humanos.

Capítulo 3

Desarrollo

3.1. Entorno de desarrollo

Un entorno de desarrollo es un conjunto integrado de herramientas, aplicaciones y recursos que proporciona a los desarrolladores las funcionalidades necesarias para escribir, probar, depurar y mantener software de manera eficiente y efectiva. A continuación, apoyándose en la figura 3.1, se muestra el entorno que se ha dispuesto por parte de CIC para el desarrollo de este TFG:

- La máquina IA - MLOPS está compuesta por los siguientes servicios:
  - **MLflow** MLflow es una herramienta utilizada para gestionar el ciclo de vida del aprendizaje automático, incluyendo el seguimiento de experimentos, la gestión de modelos y la implementación.
  - **PostgreSQL** Base de datos relacional que se utiliza probablemente para almacenar metadatos de los experimentos gestionados por MLflow.
  - **Docker** Asegura que los servicios sean consistentes y fáciles de desplegar mediante contenedores.
  - **Recursos** 8GB de RAM, 2 vCores y 50GB de almacenamiento NVMe, lo cual es adecuado para las necesidades de MLOps.
- La máquina IA - LLM (GPU) está compuesta por los siguientes servicios:
  - **Jupyter** Ambiente interactivo para desarrollo y pruebas de modelos, particularmente útiles en investigaciones y prototipos.
  - **FastAPI** Framework para construir APIs rápidas y eficientes, utilizado para servir los modelos de aprendizaje automático.
  - **Docker** Garantiza que las aplicaciones se desplieguen de manera uniforme y predecible.
  - **Recursos** 32GB de VRAM, 45GB de RAM, 15 vCores y 300GB de SSD, necesarios para entrenamiento y despliegue de modelos grandes (LLMs).
- La máquina IA - LLM (CPU) está compuesta por los siguientes servicios:
  - Similar al entorno GPU, pero con menos recursos, lo cual es adecuado para tareas menos intensivas en cálculo y evitar sobrecostes en las primeras pruebas.

- **Recursos**: 16GB de RAM, 2 vCores y 50GB de almacenamiento NVMe.

- El servicio IA - S3, usado para almacenamiento en la nube, utiliza buckets S3 para almacenar artefactos de MLflow y DVC (Data Version Control), facilitando el versionado y el acceso a datos y modelos.

- El entorno de desarrollo local (Developer) se usa para las siguientes funcionalidades:

  - **Visual Studio Code y SourceTree** IDEs que ofrecen herramientas avanzadas para el desarrollo, mantenimiento y versionado de código.
  - **DVC** Herramienta para el control de versiones de datos, crucial para reproducibilidad en proyectos de IA.
  - **Anaconda** Distribución que facilita la gestión de paquetes y entornos en Python.
  - **Git** Sistema de control de versiones utilizado para el control y colaboración del código fuente.
  - **GitLab** Plataforma para la gestión del código fuente y CI/CD, que integra repositorios Git, pipelines de integración continua y gestión de proyectos.

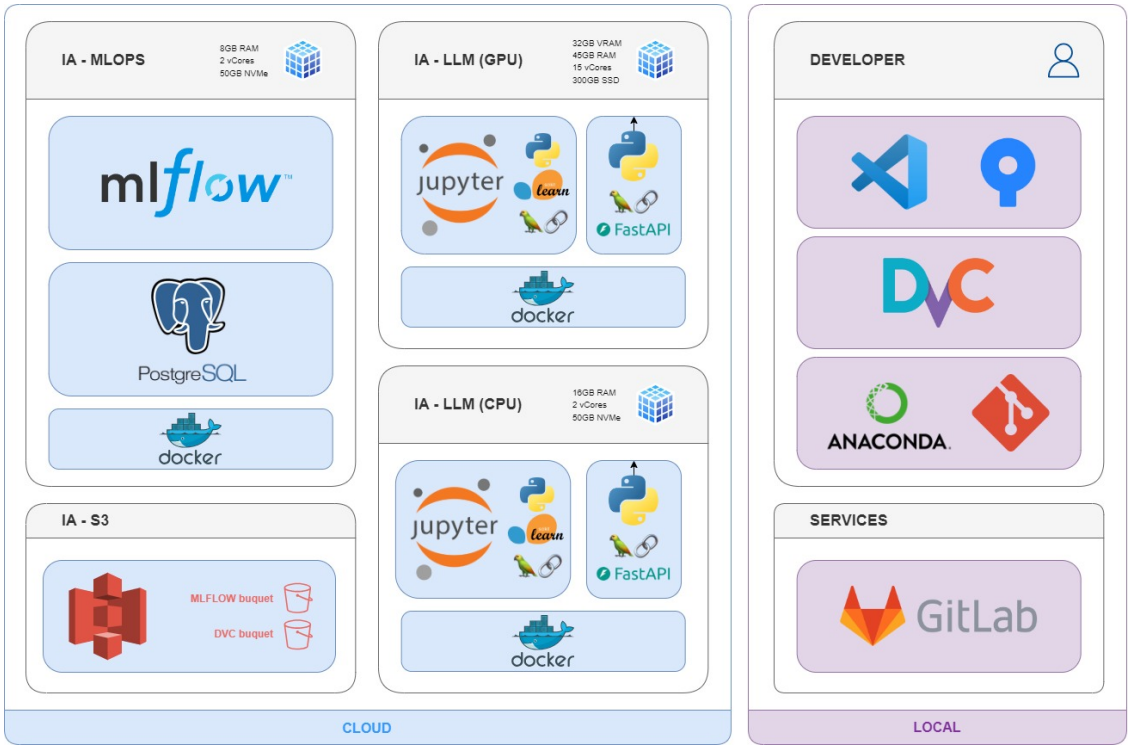

Figura 3.1: Infraestructura del entorno de desarrollo.

Para aprovechar al máximo el potencial de estos modelos, es necesario contar con las herramientas y frameworks adecuados para su desarrollo, monitorización y despliegue. Las empresas que ofrecen LLMs de pago generalmente proporcionan una API propia con librerías en Python para facilitar su integración en aplicaciones. Sin embargo, si buscamos una solución más unificada, podemos recurrir al framework LangChain. Esta herramienta permite trabajar con diferentes modelos de pago bajo una misma sintaxis, simplificando considerablemente el desarrollo.

Los modelos open-weight, por otro lado, se pueden descargar directamente desde la página del autor o autores del modelo. Sin embargo, una opción más cómoda es utilizar el portal Hugging Face. Este sitio web ofrece una amplia colección de modelos Open Source, con la posibilidad de descargarlos e incluso probarlos directamente en la web.

En el siguiente apartado, nos centraremos en el desarrollo de una aplicación en Python utilizando LangChain como framework de desarrollo. Para los modelos, utilizaremos Hugging Face como fuente de descarga. Además, para la monitorización y el versionado de los modelos, emplearemos las herramientas MLflow y DVC, respectivamente, y para exponer el modelo en una API emplearemos la librería FastAPI de python.

3.2. Arquitectura del asistente virtual

A continuación, nos adentraremos en la arquitectura que sustenta el proyecto de la PoC del asistente virtual que se está desarrollando. Para este propósito, utilizaremos el framework LangChain ya explicado, para crear una cadena de interacciones. Esta cadena toma la guía de usuario del software que estamos creando para el asistente y selecciona los fragmentos más relevantes en cada pregunta, los cuales se utilizan como contexto (RAG) junto con la pregunta del usuario. Posteriormente, esta cadena envía la información al LLM principal. Una vez recibida la pregunta, el LLM genera la respuesta correspondiente. Esta respuesta es entonces enviada de vuelta a la cadena, que a su vez la dirige hacia la API. En resumen, esta arquitectura permite una interacción fluida entre el usuario y el asistente virtual, donde la cadena de LangChain actúa como intermediario entre el usuario y el modelo de lenguaje principal, facilitando así una respuesta precisa y oportuna a las consultas del usuario.

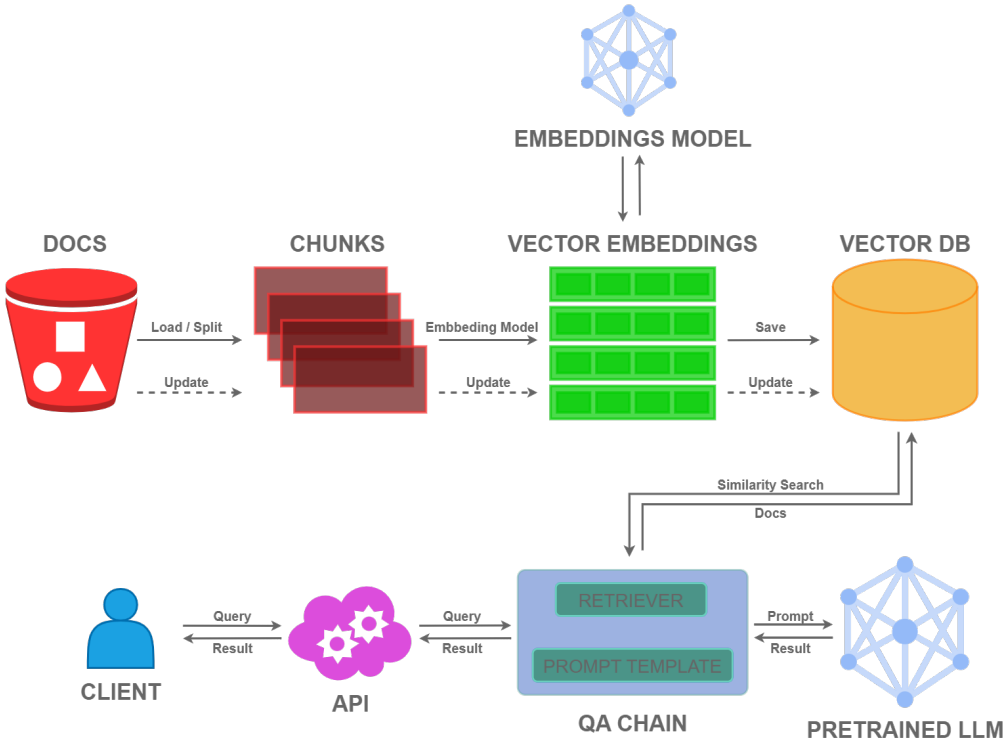

Figura 3.2: Arquitectura del asistente virtual

3.3. Ejecución del proyecto

Una vez que hemos desglosado la arquitectura subyacente del proyecto, es momento de sumergirnos en el proceso de desarrollo que implica cada una de sus etapas. Este análisis detallado nos permitirá comprender a fondo cómo cada componente se entrelaza y contribuye al funcionamiento integral del asistente virtual que estamos construyendo. A lo largo de esta exploración, se expondrán las decisiones de diseño, los desafíos técnicos enfrentados y las soluciones implementadas en cada fase del desarrollo.

3.3.1. RAG

Se ha demostrado que los LLMs preentrenados almacenan conocimientos factuales en sus parámetros y obtienen resultados de vanguardia cuando se ajustan con precisión a tareas de NLP posteriores. Sin embargo, su capacidad para acceder al conocimiento y manipularlo con precisión sigue siendo limitada y, por lo tanto, en tareas intensivas en conocimiento, su rendimiento es inferior al de las arquitecturas específicas de tareas. Además, proporcionar procedencia a sus decisiones y actualizar su conocimiento del mundo siguen siendo problemas de investigación pendientes. Esto se debe a que los datos de entrenamiento del LLM son estáticos e introducen una fecha límite en los conocimientos que tienen. Por otra parte, es muy común todavía encontrarse con que estos modelos responden información falsa cuando no conocen la respuesta a la pregunta formulada por el usuario o crear respuestas inexactas debido a una confusión terminológica, en la que diferentes fuentes de entrenamiento utilizan la misma terminología para hablar de cosas diferentes.

La técnica RAG es un proceso de optimización de la salida de un LLM, de modo que haga referencia a una base de conocimientos autorizada fuera de los orígenes de datos de entrenamiento antes de generar una respuesta. Como ya se ha explicado, los LLMs se entrenan con grandes volúmenes de datos y usan miles de millones de parámetros para generar resultados originales en tareas como responder preguntas, traducir idiomas y completar frases. RAG extiende las ya poderosas capacidades de los LLMs a dominios específicos o a la base de conocimientos interna de una organización, todo ello sin la necesidad de volver a entrenar el modelo. Se trata de un método rentable para mejorar los resultados de los LLM de modo que sigan siendo relevantes, precisos y útiles en diversos contextos. Sin el RAG, el LLM toma la información del usuario y crea una respuesta basada en la información en la que se entrenó o en lo que ya sabe. En cambio, haciendo uso de esta técnica, se introduce un componente de recuperación de información que utiliza la entrada del usuario para extraer primero la información de un nuevo origen de datos. La consulta del usuario y la información relevante se proporcionan al LLM. El LLM utiliza los nuevos conocimientos y sus datos de entrenamiento para crear mejores respuestas. A continuación, se explicará los pasos que se siguen durante el proceso para después adentrarse en cómo se ha aplicado esta técnica en este proyecto.

Es necesario mencionar que los nuevos datos fuera del conjunto de datos del entrenamiento original del LLM se denominan datos externos. Pueden provenir de varios orígenes de datos, como API, bases de datos o repositorios de documentos. Los datos pueden existir en varios formatos, como archivos, registros de bases de datos o texto plano. A través de otra técnica de IA, llamada incrustación de modelos de lenguaje, convierte los datos en representaciones numéricas y los almacena en una base de datos vectorial.

El siguiente paso es realizar una búsqueda de relevancia. La consulta del usuario se convierte en una representación vectorial y se compara con las bases de datos vectoriales. La relevancia se calculó y estableció mediante cálculos y representaciones vectoriales matemáticas, conocidos como embeddings. Los embeddings son una técnica de procesamiento de lenguaje natural que convierte el lenguaje humano en vectores matemáticos. Estos vectores son una representación del significado subyacente de las palabras, lo que permite que las computadoras procesen el lenguaje de manera más efectiva. Los vectores generados por embeddings tienen varias propiedades útiles que los hacen especialmente efectivos en aplicaciones de procesamiento de lenguaje natural. En primer lugar, los vectores son densos, lo que significa que cada una de sus dimensiones contiene información importante. En segundo lugar, los vectores son similares para las palabras que se utilizan en contextos similares. Esto significa que los vectores de embeddings pueden utilizarse para determinar la similitud semántica entre las palabras.

A continuación, el modelo RAG aumenta la entrada (o las indicaciones) del usuario al agregar los datos recuperados relevantes como contexto. Este paso utiliza técnicas de ingeniería de peticiones para comunicarse de manera efectiva con el LLM. El indicador aumentado permite que los modelos de lenguajes de gran tamaño generen una respuesta precisa a las consultas de los usuarios. En la figura 3.3 se puede ver un diagrama de como funciona este mecanismo.

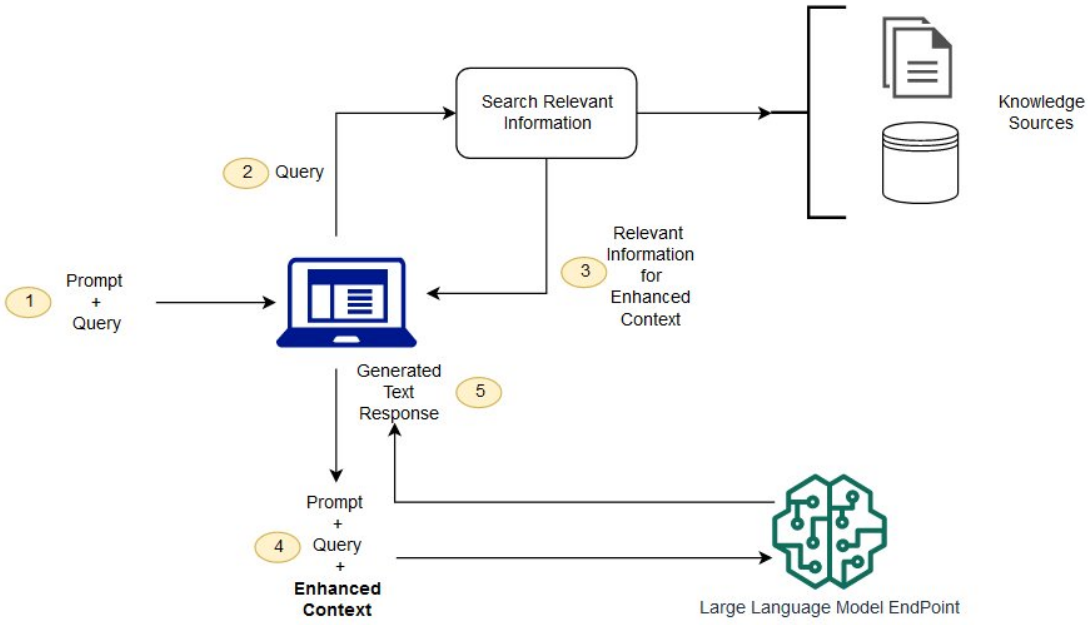

Figura 3.3: Diagrama de flujo del proceso RAG
[17]

Una vez se ha introducido RAG y como funciona, es momento de explicar de que manera se ha aplicado a este proyecto. En este caso, los datos externos se corresponden al manual de usuario del software que este trabajo tiene como caso de estudio.

Inicialmente, dicho manual es demasiado extenso y con información a veces completamente independiente entre sí, por lo que se aplicará división en fragmentos más pequeños o "chunks". El formato de archivo es de tipo Markdown (.md), un lenguaje de marcado que te permite dar formato a textos de manera sencilla y legible. La comunidad de LangChain ofrece una librería

para cargar documentos dentro de un proyecto Python llamada "document_loader". Y, dentro de esta, gracias a la clase UnstructuredMarkdownLoader() podemos comenzar a manipular un archivo de tipo Markdown. Un vez cargado, gracias a LangChain podemos disponer de la clase CharacterTextSplitter para realizar dicha división del archivo. Primero, es necesario crear una instancia de la clase, pasándole como parámetros el tamaño de cada trozo, su overlap (superposición, número de caracteres que compartirán un documento y su contiguo, una técnica que permite no perder a relación de contigüidad que mantienen) y el carácter de separación que se usará como criterio para decidir donde hacer el corte entre documentos, en principio el salto de línea. Los dos primeros forman parte del grupo de parámetros que se van a optimizar con las pruebas del apartado de elección del LLM, por lo que su valor definitivo a estas alturas todavía es desconocido. Una vez creada la instancia, con la función "split_document" de la clase, se hace la división y se almacena en memoria como una lista de documentos.

Después comienza el ya mencionado proceso de cálculo de embeddings, para el cual se necesita un LLM especializado en esta tarea. Concretamente, los modelos de extracción de características (feature extraction) y similitud de frases (sentence similarity) son los que han mostrado mayor acierto en esta tarea. Que modelo usar es uno de los desafíos que se tiene en este TFG y se decidirá durante el desarrollo. Como también será cargado en GPU durante el proceso, la limitada memoria de la que se dispone es un detalle fundamental a lo hora de tomar esta decisión, ya que interesa dedicar la mayor cantidad de espacio posible al modelo principal. Para ayudar con la selección, HuggingFace dispone de un espacio llamado "mteb/leaderboard" (Massive Text Embedding Benchmark Leaderboard (MTEB)). La mayoría de modelos que nos encontramos a la cabeza de este ranking son demasiado pesados y han sido inicialmente descartados.

Una vez que se han obtenido los embeddings de los chunks, estos son almacenados en una base de datos vectorial. Esta base de datos posee la ventaja de permitir una recuperación rápida y eficiente de la información relevante en respuesta a las consultas realizadas por el usuario. En otras palabras, cuando el usuario formula una pregunta o realiza una solicitud de información específica, el sistema puede buscar los trozos más relevantes en la base de datos vectorial y proporcionárselos al LLM principal para dar una respuesta precisa y pertinente.

La combinación de la técnica RAG con LLMs especializados en la generación de embeddings ofrece una solución innovadora para la gestión de documentos extensos y la recuperación de información relevante de manera precisa y oportuna. Esta aproximación representa un avance significativo en el campo de la recuperación de información, ya que permite superar algunas de las limitaciones tradicionales asociadas con los métodos de búsqueda de texto convencionales.

En resumen, este proyecto demuestra cómo la combinación de la técnica RAG con LLMs especializados en la generación de embeddings puede mejorar significativamente la capacidad de búsqueda y recuperación de información en documentos extensos.

3.3.2. Pipeline

En el contexto del desarrollo de sistemas conversacionales basados en modelos de lenguaje de gran escala, la construcción de un pipeline eficiente y efectivo es fundamental para el éxito del proyecto. En este apartado, se describe el diseño de un pipeline inicial, destacando la carga del modelo principal, la configuración de hiperparámetros por defecto y la aplicación de técnicas como la cuantización para optimizar el rendimiento.

El corazón de nuestro sistema conversacional reside en el modelo principal de lenguaje, que en esta etapa de desarrollo es uno de prueba. En nuestro pipeline, comenzamos por cargar este modelo, el cual será responsable de comprender y generar respuestas coherentes a partir de las preguntas planteadas por los usuarios. La selección de este modelo de prueba se basa en criterios de disponibilidad, capacidad computacional y la posibilidad de iterar rápidamente en el desarrollo del pipeline.

Para simplificar la fase inicial de desarrollo, hemos definido un conjunto de hiperparámetros por defecto que serán utilizados en el funcionamiento del modelo. Estos hiperparámetros se ajustan según las necesidades y particularidades del modelo seleccionado, y serán detallados exhaustivamente en el siguiente apartado de este trabajo. La elección de valores por defecto proporciona una base sólida para comenzar el proceso de experimentación y ajuste fino del sistema.

La cuantización es una técnica clave en la optimización de modelos de lenguaje, especialmente cuando se enfrentan restricciones de recursos computacionales, como es común en sistemas conversacionales desplegados en entornos reales. En nuestro pipeline, aplicamos la cuantización a aquellos modelos que lo requieran, con el objetivo de reducir el tamaño del modelo y mejorar la eficiencia computacional, sin comprometer significativamente su rendimiento en tareas de comprensión y generación de lenguaje.

Además de cargar el modelo principal y configurar los hiperparámetros, nuestro pipeline incluye la integración con una base de datos vectorial. Esta base de datos contiene el contexto relevante que enriquecerá las respuestas generadas por el modelo, permitiendo una interacción más contextualizada y precisa con los usuarios. La información extraída de la base de datos se utilizará para enriquecer el contexto de las preguntas realizadas, mejorando así la calidad y relevancia de las respuestas generadas por el sistema.

Una vez que el pipeline está configurado y el modelo principal cargado, el sistema está listo para interactuar con los usuarios. Esto se logra a través de un prompt, donde los usuarios pueden introducir sus preguntas. Estas preguntas se procesan mediante el modelo cargado y se generan respuestas relevantes basadas en el contexto proporcionado por la base de datos vectorial. Este proceso de interacción proporciona una experiencia conversacional fluida y natural para los usuarios, facilitando la comunicación efectiva entre humanos y sistemas de IA.

3.3.3. Elección del LLM

En este estudio, se llevó a cabo una comparación exhaustiva entre diez modelos de lenguaje utilizando parámetros fijos que se explican a continuación.

Para empezar, para cargar en memoria un modelo LangChain pone a disposición de los desarrolladores la librería transformers, que ofrece una gran variedad de opciones para manipular estos modelos de forma local. De esta librería se van a usar dos clases: AutoTokenizer y AutoModelForCausalLM, que son las clases por defecto que se encargan de asignar las clases adecuadas para tokenizar y cargar dependiendo del modelo que se este usando.

Estos modelos fueron seleccionados por su relevancia y popularidad en la comunidad de investigación de procesamiento del lenguaje natural. Los modelos evaluados incluyeron:

- Meta-Llama-3-8B-Instruct
- Mistral-7B-Instruct-v0.2

- Gemma-1.1-2b-it
- Gemma-1.1-7b-it
- Llama-2-7b-hf
- Llama-2-7b-chat-hf
- Falcon-7b-instruct
- GritLM-7B
- Bloom-7b1
- Mixtral-8x7B-Instruct-v0.1

Para llevar a cabo esta evaluación, se utilizó una variedad de métricas que se explicarán en el capítulo 4. Además, se empleó MLflow (3.1), una plataforma de seguimiento y gestión de experimentos de código abierto, para almacenar y analizar los resultados de manera eficiente.

Después de realizar numerosos experimentos y analizar detalladamente los resultados (capítulo 4), se llegó a la conclusión de que el modelo más prometedor para este proyecto es GritLM-7B. Este modelo se destacó significativamente en varias métricas importantes, lo que lo convierte en la elección ideal para nuestras necesidades específicas.

Uno de los aspectos más destacados de GritLM-7B es su capacidad para generar texto coherente y relevante en una amplia variedad de contextos. Esto se evidenció en nuestras pruebas, donde GritLM-7B superó consistentemente a los otros modelos en la capacidad de mantener la coherencia del texto a lo largo de largas secuencias. Además, su capacidad para capturar y utilizar eficazmente la información contextual demostró ser superior en comparación con los otros modelos evaluados.

Otro factor crucial que influyó en nuestra decisión fue la escalabilidad y eficiencia computacional de GritLM-7B. A pesar de su tamaño y complejidad, este modelo demostró un rendimiento impresionante en términos de velocidad de entrenamiento y generación de texto. Esto es especialmente importante para nuestro proyecto, ya que nos permite manejar grandes volúmenes de datos de manera eficiente y generar resultados en tiempo real.

Además, GritLM-7B ofrece una arquitectura flexible y fácilmente adaptable, lo que permite ajustar y personalizar el modelo según necesidades específicas. Esta capacidad de personalización es muy importante en entornos donde se requiere un alto grado de precisión y adaptabilidad a contextos variables.

3.3.4. Ajuste de hiperparámetros

En el desarrollo y optimización de modelos de lenguaje de gran tamaño, el ajuste de hiperparámetros es una etapa crucial para maximizar el rendimiento y la precisión de las respuestas generadas. En este apartado, se analizarán los principales hiperparámetros considerados para ajustar un LLM: temperature, max_new_tokens, search_k, chunk_size y chunk_overlap. Cada uno de estos hiperparámetros se describe en detalle a continuación, explicando su función y contribución al rendimiento del modelo.

El hiperparámetro *temperature* controla la aleatoriedad de las respuestas generadas por el modelo. Este valor influye en la distribución de probabilidad de las palabras siguientes en una secuencia.

- **Función**: Modificar la *temperature* afecta la diversidad de las respuestas del modelo. Un valor bajo de *temperature* (cercano a 0) hace que el modelo sea más determinista, eligiendo las palabras con mayor probabilidad. En contraste, un valor alto de *temperature* (cercano a 1) aumenta la aleatoriedad, permitiendo respuestas más variadas y creativas. Aunque al modelo se le puede especificar cualquier valor de *temperature* el rango efectivo de valores está comprendido entre 0 y 1.
- **Contribución**: Ajustar la *temperature* permite equilibrar entre respuestas precisas y variadas, dependiendo del contexto de uso. Para aplicaciones que requieren consistencia y precisión, una *temperature* baja es preferible. Para tareas que se benefician de la creatividad y la diversidad, se puede usar una *temperature* más alta.

El hiperparámetro *max_new_tokens* define el número máximo de tokens que el modelo puede generar en una respuesta.

- **Función**: Este parámetro limita la longitud de la salida generada, previniendo respuestas excesivamente largas que puedan ser irrelevantes o redundantes.
- **Contribución**: Ajustar *max_new_tokens* ayuda a controlar el tamaño de las respuestas, lo que es útil para mantener la relevancia y la concisión. Además, es importante para aplicaciones con restricciones de longitud de texto.

El hiperparámetro *search_k* se refiere al número de trayectorias consideradas durante el proceso de búsqueda del siguiente token.

- **Función**: Aumentar *search_k* permite al modelo evaluar más trayectorias posibles, mejorando la calidad de la respuesta al considerar más opciones antes de elegir la más adecuada.
- **Contribución**: Un mayor valor de *search_k* puede mejorar la precisión de las respuestas, pero a costa de un mayor tiempo de cómputo. Este parámetro es crucial para encontrar un balance entre precisión y eficiencia computacional.

El hiperparámetro *chunk_size* se refiere al tamaño de los segmentos en los que se divide el texto de entrada para ser procesado por el modelo.

- **Función**: Dividir el texto en chunks permite al modelo manejar entradas largas de manera más eficiente, procesando cada segmento por separado antes de combinar los resultados.
- **Contribución**: Ajustar *chunk_size* puede influir en la coherencia y el contexto de las respuestas generadas. Segmentos más grandes mantienen más contexto, mientras que segmentos más pequeños pueden procesarse más rápidamente.

El hiperparámetro *chunk_overlap* define el número de tokens que se solapan entre los chunks consecutivos.

- **Función**: El solapamiento entre chunks asegura que el contexto relevante se mantenga entre segmentos, mejorando la coherencia de la salida generada.
- **Contribución**: Ajustar *chunk_ overlap* es importante para mantener el flujo de información entre chunks. Un solapamiento adecuado puede mejorar la calidad de las respuestas, especialmente en textos largos y complejos.

El *prompt* es el parámetro mediante el cuál se le indica al modelo cualquier instrucción que debe seguir a la hora de responder a la consulta del usuario.

- **Función**: El *prompt* implica estructurar la pregunta o solicitud de manera clara y precisa, proporcionando suficiente contexto para guiar al modelo hacia una respuesta adecuada.
- **Contribución**: Un prompt bien diseñado puede mejorar la precisión y relevancia de las respuestas, reducir la ambigüedad y evitar respuestas irrelevantes o fuera de contexto.
- **Técnicas comunes**: Algunas técnicas incluyen la especificación clara de la tarea, el uso de ejemplos, y la incorporación de instrucciones explícitas dentro del prompt.

El ajuste de hiperparámetros y el *prompt engineering* son componentes esenciales en la optimización de modelos de lenguaje de gran tamaño. Estos procesos permiten personalizar y mejorar el rendimiento del modelo, adaptándolo a diferentes aplicaciones y necesidades específicas.

3.3.5. API

Para acabar el desarrollo de este proyecto, se ha desarrollado una API a través de la librería FastAPI mencionada anteriormente. Una API (Interfaz de Programación de Aplicaciones, por sus siglas en inglés) es un conjunto de reglas y protocolos que permite que diferentes aplicaciones se comuniquen entre sí. Básicamente, una API define cómo interactuar con un sistema o servicio específico, proporcionando una forma estandarizada de acceder a sus funcionalidades y datos. Se ha optado por hacer uso del método GET de HTTP, uno de los dos métodos principales utilizados para enviar datos desde un formulario HTML a un servidor para su procesamiento. El método GET envía los datos como parte de la URL de la solicitud HTTP. Cuando un usuario envía un formulario utilizando el método GET, los valores de los campos del formulario se adjuntan a la URL como una cadena de consulta (query string).

Por otra parte, se ha generado una clase llamada Chatbot, para interactuar desde la API con el modelo. Esta clase contiene un método `__init__` con todos los parámetros del modelo a usar, así como un valor por defecto para cada parámetro por si el usuario no lo especifica en la interacción con la API. Después, se incluyen los métodos que permiten construir toda la cadena de inferencia. Para empezar, un método `construct_main_llm` para obtener el modelo principal con sus parámetros identificador, temperatura y máximo número de tokens nuevos. El método `construct_embeddings_llm` genera el modelo de embeddings a partir de su identificador e indicando si se le quiere ejecutar en CPU o GPU y si se quieren normalizar los embeddings. Con la función `generate_vector_db` genera la base de datos vectorial donde se almacenan los embeddings del manual de usuario. Finalmente, se puede construir la cadena completa con la función `construct_chain` y hacer una pregunta al modelo con `query_model`.

Capítulo 4

Resultados y análisis

En este capítulo se mostrará los resultados para su análisis y así dar paso a sacar conclusiones en el capítulo 5.

4.1. Resultados de comparación de modelos

En la figura 4.1, se comparan las 10 ejecuciones de distintos modelos. El tiempo total de ejecución es un indicador crucial de la eficiencia de cada modelo. Su análisis muestra una clara variabilidad en la eficiencia de los modelos evaluados. Los modelos gemma-2b-it y GritLM-7B son los más eficientes, mientras que el modelo gemma-7b-it es el menos eficiente.

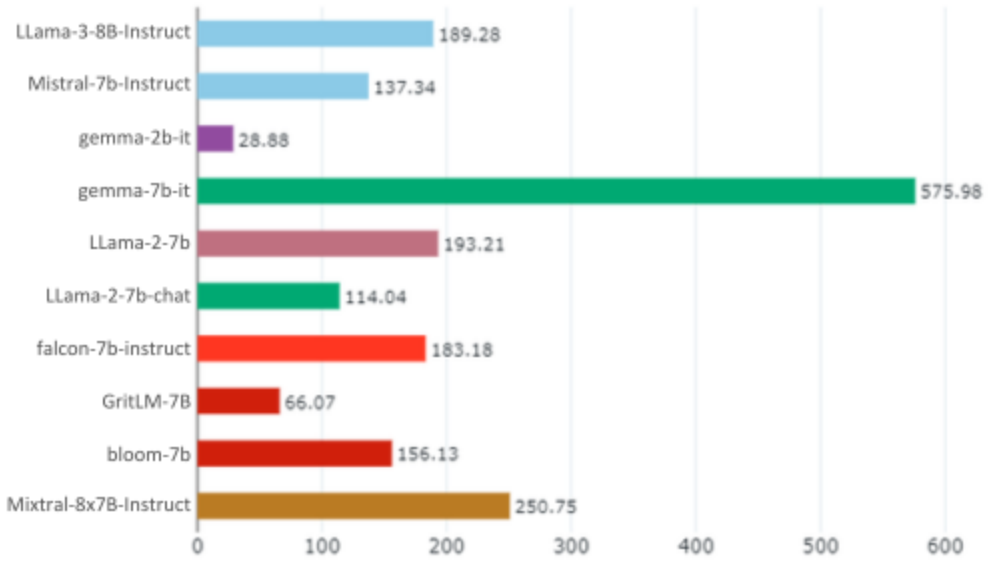

Figura 4.1: Tiempo de inferencia (en segundos)

La figura 4.2, presenta una comparación del tamaño de los LLMs evaluados. El tamaño del modelo puede tener implicaciones importantes en términos de almacenamiento y manejo de datos, especialmente en sistemas con recursos limitados. El tamaño, por lo general, no varía demasiado exceptuando los Gemma, ya que uno es más pequeño y el otro esta cuantizado.

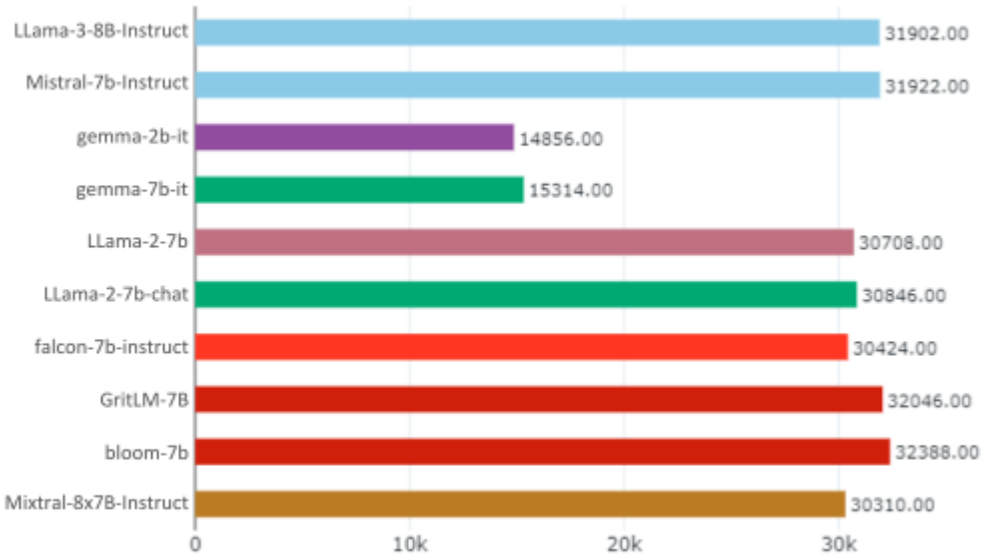

Figura 4.2: Memoria usada (en MB)

El análisis de los tiempos de ejecución y tamaños de los modelos proporciona una visión clara de sus eficiencias y requisitos de recursos. Mientras que los modelos gemma-2b-it y GritLM-7b destacan por su rapidez, el gemma-7b-it claramente requiere optimización. En términos de tamaño, los modelos gemma son los más eficientes, ofreciendo una combinación equilibrada de rendimiento y gestión de recursos. Este análisis es crucial para la selección del modelo adecuado según las necesidades específicas de almacenamiento y rendimiento del sistema en el que se vayan a implementar. El modelo gemma-2b-it, al ser ejecutado, no mostró casi respuestas completas, por lo que fue descartasdo

En global, los LLM mejor valorados en este primer análisis forman parte de la familia open-weight lanzada por la empresa francesa Mistral. Vamos a pasar ahora a analizarlos más en profundidad.

La figura 4.3 muestra una comparación detallada entre tres LLMs de Mistral: Mixtral-8x7B, Mistral-7B y GritLM-7B (siendo este último un fine-tuning de Mistral-7B). Se evalúan utilizando parámetros fijos explicados en el anterior capítulo (capítulo 3). A continuación, se analizan cada uno de estos aspectos y se sacan conclusiones a partir de los datos resultantes.

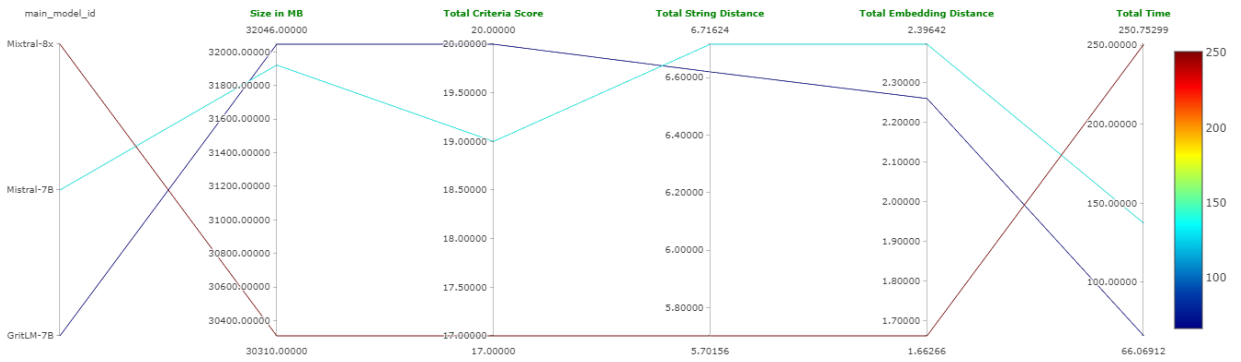

Figura 4.3: Comparación entre varios modelos Mistral

El primer parámetro a considerar es el tamaño en Megabytes (MB) de cada modelo. Mixtral-8x7B es el más grande, con un tamaño de 32046 MB. Esto indica que este modelo puede tener una mayor capacidad para almacenar información y características complejas, lo cual podría traducirse en un rendimiento superior en ciertas tareas. Sin embargo, ha tenido que ser cuantizado a 4 bits de precisión, lo que puede equilibrar el rendimiento con modelos más pequeños. En comparación, Mistral-7B y GritLM-7B tienen tamaños de 31800 MB y 30310 MB respectivamente. Aunque más pequeños, estos modelos pueden ser más eficientes en términos de almacenamiento y potencialmente más rápidos en procesamiento debido a su menor tamaño.

La puntuación total (suma de las 20 preguntas) de criterios es una métrica que combina varias evaluaciones cualitativas y cuantitativas del rendimiento de los modelos. Realmente se trata de autoevaluaciones que hace el modelo de sus propias respuestas, en lo que el desarrollador solo puede especificar el criterio de evaluación, que en este caso se optó por la *relevancia*. GritLM-7B lidera con una puntuación de 20, seguido por Mistral-7B con 19 y Mixtral-8x7B con 17. Este resultado sugiere que Mixtral-8x7B, a pesar de ser el que más capas tiene, también es el menos preciso debido a la cuantización.

La distancia total de strings mide la similitud entre las salidas generadas por los modelos y un conjunto de referencias. Mistral-7B tiene una distancia de 6.71624, mientras que GritLM-7B y Mixtral-8x7B tienen distancias de 6.6 y 5.70156 respectivamente. Una menor distancia sugiere una mayor similitud con las referencias, indicando que Mixtral-8x7B genera resultados más precisos en comparación con los otros dos modelos.

La distancia de embeddings es una métrica que evalúa la calidad de las representaciones internas generadas por los modelos. Mistral-7B tiene la mayor distancia de embeddings con 2.39642, lo que indica una mayor variabilidad en sus representaciones. Por otro lado, GritLM-7B y Mistral-8x7B tienen distancias de 2.27 y 1.66266 respectivamente. Una menor distancia se interpreta como una mejor coherencia interna y una mayor calidad en las representaciones semánticas.

El tiempo total es una medida crucial, especialmente para aplicaciones en tiempo real. Mixtral-8x7B, siendo el más grande, también tiene el mayor tiempo total con 250.75299 unidades. Esto se debe principalmente a que, al tratarse de un modelo con más parámetros, requiere más accesos a memoria, lo que eleva el tiempo de inferencia. Aunque por lo visto antes es el más potente en alguna métrica, es el menos eficiente en términos de tiempo. GritLM-7B,presenta un tiempo total significativamente menor de 66.06912, lo que lo hace más adecuado para aplicaciones que requieren respuestas rápidas.

El análisis de la figura 4.3 proporciona una visión clara de las fortalezas y debilidades de cada uno de los modelos de Mistral. Mixtral-8x7B se destaca por su capacidad y robustez general, reflejada en su tamaño y sus distancias más pequeñas. Sin embargo, su mayor tamaño y tiempo total indican que puede no ser la opción más eficiente para todas las aplicaciones. Mistral-7B, con un equilibrio entre tamaño, puntuación de criterios y tiempo total, parece ser una opción sólida para una variedad de aplicaciones, ofreciendo un buen rendimiento sin comprometer demasiado la eficiencia. GritLM-7B obtiene métricas más positivas respecto a Mistral-7B con una variación de recursos de almacenamiento necesario muy baja Esto lo hace una opción atractiva para aplicaciones que requieren precisión y eficiencia en términos de almacenamiento y procesamiento.

Desde CIC se ha indicado que en el ámbito empresarial interesa ofrecer a los usuarios una inferencia lo más rápida posible sin ofrecer respuestas inútiles. De acuerdo con las métricas

observadas y dado que los tres modelos ofrecían respuestas coherentes, se puede concluir que GritLM-7B ha sido el LLM que mejores resultados ha dado fruto de los experimentos realizados. Este será el que se usará de aquí en adelante para las pruebas de ajuste de hiperparámetos.

4.2. Resultados de ajuste de hiperparámetros

La figura 4.4 muestra una gráfica que resume los resultados de una serie de experimentos realizados con el LLM GritLM-7B. Los experimentos se llevaron a cabo con diferentes configuraciones de hiperparámetros, y la gráfica muestra el efecto de estas configuraciones en el rendimiento del modelo en cuatro de las métricas previamente utilizadas.

El único experimento en el que no se obtiene la máxima puntuación es aquel en el que se utilizando 8 documentos como contexto, demasiado para una buena respuesta. En general, la puntuación total de criterios tiende a ser la máxima y variar los parámetros no afecta demasiado.

Como se puede observar, la distancia total de incrustación varía significativamente entre las diferentes configuraciones de hiperparámetros. La distancia total de incrustación más baja se obtiene con un tamaño de bloque de 200, la menor de las configuraciones probadas. Vuelve a ser notable los malos resultados al aplicar un gran número de documentos al contexto.

En cuanto a la distancia total de cadenas va de la mano, en cuanto a resultados se refiere, con la distancia total de incrustación. Se observan los valores más altos para los experimentos con muchos documentos distintos y es notable el resultado el especificar un tamaño de bloque de 500 pero con más documentos.

Los resultados de los experimentos muestran que el rendimiento del modelo GritLM-7B se ve afectado significativamente por la configuración de los hiperparámetros. La configuración óptima de los hiperparámetros depende de la tarea específica que se esté realizando. Sin embargo, en general, los resultados sugieren que limitar lo máximo posible los tokens que se pueden generar y un tamaño de bloque grande con pocos documentos son propensos a dar respuestas más precisas y, como resultado, un mejor rendimiento.

Es importante tener en cuenta que los resultados de este análisis se basan en un único conjunto de experimentos. Es posible que se obtengan resultados diferentes con otros conjuntos de datos o tareas.

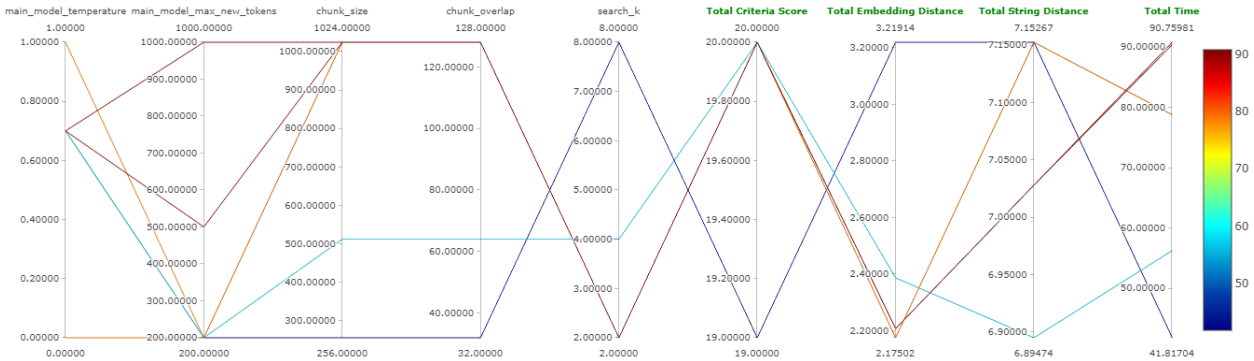

Figura 4.4: Experimentos de hiperparámetros

La imagen 4.5 muestra los resultados de un experimento de prompt engineering con LLMs. Se probaron 5 prompts diferentes, y la imagen muestra el rendimiento de cada prompt con las mismas

métricas que para los hiperparámetros.

Lo primero que se puede observar es que para todos los prompts se ha obtenido la máxima puntuación de criterio.

- Prompt A: *DOCUMENT: {} QUESTION: {} INSTRUCTIONS: Answer the users QUESTION using the DOCUMENT text above. Keep your answer ground in the facts of the DOCUMENT. If the DOCUMENT doesn't contain the facts to answer the QUESTION, say that you don't know the answer.*

  La distancia total de incrustación relativamente baja de este prompt sugiere que las respuestas generadas fueron similares a las respuestas en el conjunto de datos de validación. Sin embargo, la distancia entre cadenas y el tiempo fueron los más elevados por lo tanto hay que descartar este ejemplo como el más indicado.

- Prompt B: *Given the context: {}; answer this question: {}. Answer only with the information that you really think is relevant. Do not include unnecessary information from the context. Always say «thanks for asking!» at the end of the answer.*

  La distancia total de incrustación relativamente baja de este prompt sugiere que las respuestas generadas fueron similares a las respuestas en el conjunto de datos de validación. En comparación con el primer prompt, destaca especialmente la reducción de coste computacional lo que en la práctica lo convierte en un prompt mucho más interesante para aplicar.

- Prompt C: *You are an expert in telecommunications and cybersecurity. Answer the question {} given the context {}.*

  A pesar de tener un coste total aceptable este prompt obtiene malos resultados tanto en distancia total de incrustación como de cadenas, lo que indica mal rendimiento y hace que se descarte como una opción viable.

- Prompt D: *Given the context: {}; answer this question: {}. Answer only with the information that you really think is relevant. Always say «thanks for asking!» at the end of the answer.*

  Nos encontramos ahora con un prompt que se puede relacionar directamente con el anterior debido a que han obtenido prácticamente el mismo tiempo total. A pesar de ello, en este nos encontramos con que las métricas son bastante mejores en ambos casos, lo que lo convierten en un firme candidato a ser seleccionado.

- Prompt E: *DOCUMENT: {} QUESTION: {} INSTRUCTIONS: Answer the users QUESTION using the DOCUMENT text above. Keep your answer ground in the facts of the DOCUMENT.*

  Con este último prompt tenemos al que con diferencia consigue sacar mayor rendimiento al LLM y conseguir una recuperación de información óptima, a pesar de que lo hace a costa de aumentar el tiempo de inferencia.

En conclusión, el experimento de prompt engineering con LLMs demostró resultados variados en función de las métricas evaluadas para cada prompt. Todos los prompts alcanzaron la máxima puntuación de criterio, sin embargo, su desempeño difirió significativamente en otros aspectos clave.

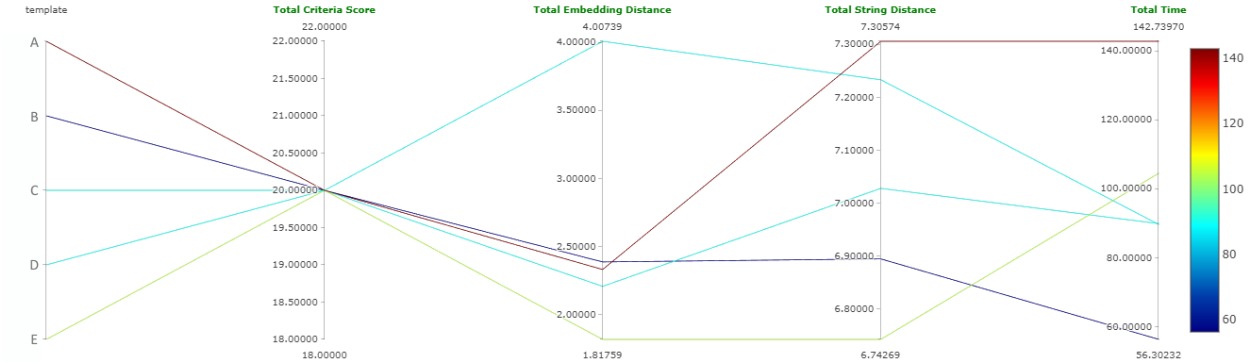

Figura 4.5: Experimentos de prompt engineering

El Prompt A, aunque generó respuestas similares al conjunto de datos de validación, fue el menos eficiente en términos de distancia entre cadenas y tiempo, lo cual lo descarta como una opción óptima. Por otro lado, el Prompt B, manteniendo una baja distancia de incrustación, sobresalió al reducir significativamente el coste computacional, posicionándose como una alternativa atractiva.

El Prompt C mostró un rendimiento deficiente en las métricas de distancia, lo que lo descalifica como viable. En contraste, el Prompt D, aunque similar en tiempo total al anterior, ofreció mejores métricas, consolidándose como un fuerte candidato. Finalmente, el Prompt E destacó por maximizar el rendimiento del LLM y optimizar la recuperación de información, aunque a expensas de un mayor tiempo de inferencia.

Estos hallazgos subrayan la importancia de equilibrar diversas métricas al seleccionar el prompt adecuado, teniendo en cuenta no solo la precisión y similitud de las respuestas, sino también la eficiencia computacional y el tiempo de procesamiento.

Capítulo 5

Conclusiones y futuros avances

5.1. Conclusiones

El desarrollo y análisis de modelos de lenguaje generativo ha revelado varias conclusiones importantes en relación con la integración de técnicas avanzadas y la optimización del rendimiento de estos modelos en casos de uso específicos. En particular, este trabajo se ha centrado en la técnica RAG, el prompt engineering, y la sensibilidad del modelo a estas técnicas en comparación con el ajuste de hiperparámetros. Además, se ha considerado la importancia de contar con una infraestructura robusta para realizar investigaciones en IA generativa.

Un hallazgo notable de este desarrollo es que los modelos no tan pesados o densos pueden ser una opción viable en comparación con los modelos que, aunque teóricamente más precisos, requieren significativamente más tiempo de cómputo. Estos modelos más ligeros ofrecen un equilibrio entre precisión y eficiencia, permitiendo su uso en aplicaciones donde los recursos computacionales son limitados o donde el tiempo de respuesta es crítico. Esta conclusión sugiere que, en muchos casos, la elección de un modelo debe considerar no solo la precisión teórica, sino también la viabilidad práctica de su implementación y uso continuo.

La técnica RAG ha demostrado ser extremadamente eficiente para añadir conocimiento específico al modelo en un caso de uso concreto. Esta técnica combina la recuperación de información con la generación de texto, permitiendo que el modelo acceda a datos relevantes y actualizados que no están directamente presentes en su entrenamiento original. Este enfoque no solo mejora la precisión de las respuestas del modelo, sino que también permite una mayor adaptabilidad a contextos específicos. En situaciones donde la información actualizada y precisa es crítica, como en el ámbito médico o financiero, RAG se posiciona como una herramienta invaluable para enriquecer las capacidades del modelo.

El prompt engineering se ha destacado como una parte clave para que el modelo utilice efectivamente la información adicional proporcionada. Diseñar prompts efectivos requiere una comprensión profunda de cómo el modelo interpreta y responde a diferentes tipos de entradas. Los resultados del experimento indican que un buen diseño de prompt puede guiar al modelo a generar respuestas más precisas y contextualmente relevantes. Esto sugiere que, a medida que los modelos de lenguaje se vuelven más complejos, la habilidad para crear y ajustar prompts se convierte en una competencia esencial para maximizar el rendimiento del modelo.

El análisis comparativo ha revelado que este LLM ha sido más sensible al prompt engineering

que al ajuste de hiperparámetros. Mientras que los hiperparámetros son fundamentales para definir la estructura y el comportamiento básico del modelo, el prompt engineering parece tener un impacto más inmediato y significativo en la calidad de las respuestas generadas. Esta observación resalta la importancia de centrarse en el diseño de prompts como una estrategia efectiva para mejorar el rendimiento del modelo sin necesidad de recurrir a modificaciones costosas y complejas en los hiperparámetros.

El estudio y desarrollo en IA generativa todavía requiere una cantidad considerable de recursos computacionales. Contar con un equipo y una infraestructura potente es fundamental para explorar y desarrollar estas tecnologías de manera efectiva. Los modelos actuales, especialmente los más avanzados, demandan hardware especializado y una infraestructura robusta para su entrenamiento y operación. Esta necesidad de recursos no solo se traduce en equipos de alto rendimiento, sino también en la disponibilidad de soporte técnico y software optimizado para gestionar grandes volúmenes de datos y realizar cálculos intensivos.

En conclusión, la combinación de técnicas como RAG y el prompt engineering puede transformar significativamente el rendimiento y la aplicabilidad de los modelos de lenguaje generativo. La capacidad de añadir conocimiento específico y relevante a través de RAG, junto con un diseño cuidadoso de prompts, permite que los modelos no solo sean más precisos sino también más adaptables a diversos contextos. La sensibilidad del modelo al prompt engineering destaca la necesidad de invertir en el desarrollo de habilidades y estrategias para optimizar esta área.

Asimismo, la importancia de una infraestructura computacional robusta no puede subestimarse. El estudio y el desarrollo en IA generativa son tareas que demandan recursos significativos, y contar con el equipo adecuado es esencial para avanzar en este campo. Finalmente, la consideración de modelos menos densos y más eficientes sugiere un camino hacia soluciones prácticas y escalables que pueden ser implementadas de manera más amplia y accesible.

El resultado final de la PoC del asistente virtual se ha considerado satisfactorio y desde CIC se ha usado como punto de partida para un proyecto de innovación en IA.

5.2. Futuros avances

El campo de la IA generativa está en constante evolución, y los descubrimientos realizados hasta ahora abren numerosas vías para futuras investigaciones y mejoras. En esta sección, exploraremos varios caminos prometedores que pueden mejorar significativamente el rendimiento y la aplicabilidad de los modelos de lenguaje generativo.

El fine-tuning es un proceso de ajuste fino del modelo entrenado, adaptándolo a un dominio específico o a un conjunto de datos más reducido. Este enfoque puede mejorar drásticamente la precisión y relevancia de las respuestas generadas por el modelo, especialmente en contextos especializados. En lugar de entrenar un modelo desde cero, el fine-tuning permite refinar un modelo preentrenado con datos adicionales específicos, lo cual es más eficiente tanto en términos de tiempo como de recursos computacionales.

El procesamiento exhaustivo de los datos previos es otro aspecto crucial para mejorar el rendimiento de los LLMs. Este proceso implica la limpieza, normalización y enriquecimiento de los datos utilizados para entrenar y evaluar el modelo. Datos de alta calidad son fundamentales para asegurar que el modelo aprenda patrones relevantes y generalizables.

Ampliar el conjunto de preguntas y respuestas utilizadas para validar y normalizar los resultados del modelo es una estrategia eficaz para mejorar la robustez y fiabilidad de las respuestas generadas. Un mayor número de ejemplos permite al modelo aprender y generalizar mejor a partir de una base de datos más amplia y diversa. La validación exhaustiva implica no solo aumentar la cantidad de datos, sino también asegurar que estos datos sean representativos de los diferentes escenarios y contextos en los que se espera que el modelo opere. Además, la normalización de resultados a través de múltiples iteraciones y escenarios de prueba permite identificar patrones de error y áreas de mejora, facilitando ajustes más precisos y efectivos en el modelo.

Las técnicas avanzadas de regularización son fundamentales para mejorar la robustez y generalización de los modelos de lenguaje generativo. Estas técnicas ayudan a prevenir el sobreajuste y a mantener el equilibrio entre la precisión y la capacidad de generalización del modelo. Además, la implementación de enfoques de aprendizaje activo, donde el modelo se entrena iterativamente con nuevos datos seleccionados estratégicamente, puede mejorar la capacidad del modelo para adaptarse a nuevos contextos y datos.

Finalmente, la optimización del uso de recursos computacionales sigue siendo un área crucial para el avance de la IA generativa. A medida que los modelos se vuelven más grandes y complejos, encontrar formas de entrenarlos y operarlos de manera más eficiente es esencial. El desarrollo de algoritmos más eficientes, la implementación de técnicas de compresión de modelos, y el uso de hardware especializado, como las GPU y TPU, pueden contribuir significativamente a reducir los costos y tiempos de cómputo. Además, la investigación en enfoques distribuidos y paralelos para el entrenamiento de modelos puede abrir nuevas oportunidades para manejar modelos de gran escala de manera más efectiva.

En resumen, el futuro de la IA generativa se perfila lleno de oportunidades y desafíos. El fine-tuning del modelo, el procesamiento exhaustivo de datos previos o la validación con un mayor número de preguntas y respuestas son solo algunas de las estrategias que pueden conducir a mejoras significativas en el rendimiento y aplicabilidad de estos modelos. La continua investigación en nuevas arquitecturas de modelos, técnicas avanzadas de regularización, y la optimización del uso de recursos computacionales serán cruciales para avanzar en este campo. A medida que las tecnologías y metodologías evolucionen, es esencial mantenerse al día con las últimas innovaciones para aprovechar al máximo el potencial de la IA generativa y sus aplicaciones en el mundo real.

Referencias

[1] CHANG, Yupeng, et al. A survey on evaluation of large language models. ACM Transactions on Intelligent Systems and Technology, 2024, vol. 15, no 3, p. 1-45. Disponible en https://dl.acm.org/doi/abs/10.1145/3641289. Visto por última vez el 04/07/2024.

[2] VASWANI, Ashish, et al. Attention is all you need. Advances in neural information processing systems, 2017, vol. 30. Disponible en https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. Visto por última vez el 04/07/2024.

[3] Kemal Erdem (2021), *Understanding Positional Encoding in Transformers. Disponible en* https://towardsdatascience.com/understanding-positional-encoding-in-transformers-dc6bafc021ab. Visto por última vez el 04/07/2024.

[4] Shibo Wang, Pankaj Kanwar (2019), https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus. Visto por última vez el 30/06/2024.

[5] JACOBS, Robert A., et al. Adaptive mixtures of local experts. Neural computation, 1991, vol. 3, no 1, p. 79-87. Disponible en https://ieeexplore.ieee.org/abstract/document/6797059. Visto por última vez el 04/07/2024.

[6] OpenAI (2024), https://openai.com/index/hello-gpt-4o/. Visto por última vez el 04/07/2024.

[7] ACHIAM, Josh, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023. Disponible en https://arxiv.org/pdf/2303.08774. Visto por última vez el 04/07/2024.

[8] REID, Machel, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv preprint arXiv:2403.05530, 2024. Disponible en https://arxiv.org/pdf/2403.05530. Visto por última vez el 04/07/2024.

[9] ANIL, Rohan, et al. Palm 2 technical report. arXiv preprint arXiv:2305.10403, 2023. Disponible en https://arxiv.org/pdf/2305.10403. Visto por última vez el 04/07/2024.

[10] CHOWDHERY, Aakanksha, et al. Palm: Scaling language modeling with pathways. Journal of Machine Learning Research, 2023, vol. 24, no 240, p. 1-113. Disponible en https://www.jmlr.org/papers/v24/22-1144.html. Visto por última vez el 04/07/2024.

[11] TOUVRON, Hugo, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023. Disponible en https://arxiv.org/pdf/2307.09288. Visto por última vez el 04/07/2024.

[12] JIANG, Albert Q., et al. Mixtral of experts. arXiv preprint arXiv:2401.04088, 2024. Disponible en https://arxiv.org/pdf/2401.04088. Visto por última vez el 04/07/2024.

[13] ALMAZROUEI, Ebtesam, et al. The falcon series of open language models. arXiv preprint arXiv:2311.16867, 2023. Disponible en https://arxiv.org/pdf/2311.16867. Visto por última vez el 04/07/2024.

[14] Hugging Face, https://huggingface.co/. Visto por última vez el 30/06/2024.

[15] LangChain, https://www.langchain.com/. Visto por última vez el 30/06/2024.

[16] IBM, https://www.ibm.com/topics/large-language-models. Visto por último vez el 14/04/2024.

[17] AWS, https://aws.amazon.com/es/what-is/retrieval-augmented-generation/. Visto por última vez el 15/05/2024.

[18] https://github.com/ismaelngmez/VirtualAssistant-LLM-RAG. Modificado por última vez el 01/07/2024.

Apéndice A

Lista de Acrónimos

API Application Programming Interface

ARC Analogical Reasoning Corpus

CIC Consulting Informático de Cantabria SL.

CNN Convolutional Neural Network

GPT Generative Pre-trained Transformer

GPU Graphics Processing Unit

IA Inteligencia Artificial

LFM Large Foundational Model

LLM Large Language Model

MB Megabytes

ML Machine Learning

MMLU Massive Multitask Language Understanding

MoE Mixture of Experts

MTEB Massive Text Embedding Benchmark Leaderboard

NLP Natural Language Processing

PaaS Platform as a Service

PoC Proof of Concept

RAG Retrieval-Augmented Generation

RNN Recurrent Neural Network

TFG Trabajo de Fin de Grado