Facultad
de
Ciencias

Entrevistador virtual utilizando técnicas RAG en LLMs

(Virtual interviewer using RAG techniques in LLMs)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Raúl Elizalde Roldán

Director: Inés González Rodríguez

Co-Director: Sergio Martínez Prieto (HP SCDS)

Junio - 2024

Resumen

En un mundo cada vez más digitalizado y competitivo parece no solo aconsejable sino necesario optimizar las prácticas de selección de personal con las herramientas tecnológicas adecuadas. En concreto, incorporar técnicas de Inteligencia Artificial al ámbito de las entrevistas laborales presenta a la vez un desafío y la oportunidad de automatizar y mejorar significativamente el proceso de selección de personal.

El objetivo del proyecto es obtener un Sistema de Aprendizaje Automático (SAA) para entrevistas laborales utilizando un enfoque basado en modelos de lenguaje de gran tamaño (LLMs o Large Language Models). Durante el desarrollo se han elegido diferentes modelos a los que se le aplican técnicas RAG. Para aplicar RAG, se ha seleccionado un corpus de entrenamiento relacionado con las entrevistas laborales. Finalmente, se realiza un análisis y evaluación de las pruebas realizadas.

Entre las herramientas y tecnologías a utilizar se incluyen Python (incorporando Pytorch) y Google Colab y, como modelos LLM, Llama2 o Mistral entre otros.

El TFG se desarrolla en el programa observatorio de la empresa HP con sede en León bajo la supervisión de Sergio Martínez Prieto como tutor de la entidad y de Inés González Rodríguez como tutora académica.

Palabras clave: IA Generativa, NLP, RAG, LLM

Abstract

In an increasingly digitalized and competitive world, it seems not only advisable but necessary to optimize personnel selection practices with appropriate technological tools. Specifically, incorporating Artificial Intelligence techniques into job interviews presents both a challenge and an opportunity to automate and significantly improve the recruitment process.

The objective of the project is to develop a Machine Learning System (MLS) for job interviews using an approach based on Large Language Models (LLMs). During the development process, different models have been chosen to which RAG techniques will be applied. To apply RAG, a training corpus related to job interviews has been selected. Finally, an analysis and evaluation of the tests conducted will be performed.

Among the tools and technologies to be used are Python (including Pytorch) and Google Colab, and for LLMs, Llama2 or Mistral, among others.

The Final Degree Project (TFG) is being developed in the HP Observatory program based in León, under the supervision of Sergio Martínez Prieto as the entity tutor and Inés González Rodríguez as the academic tutor.

Palabras clave: Generative AI, NLP, RAG, LLM

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Propuesta

El objetivo de este trabajo es obtener un Sistema de Aprendizaje Automático (SAA) en el contexto de las entrevistas laborales. Se propone un enfoque centrado en el uso de modelos de lenguaje, más concretamente en los modelos de lenguaje de gran tamaño, en inglés Large Language Models (LLM). Se aplican técnicas de generación aumentada por recuperación, conocidas como Retrieval Augmented Generation (RAG), para entrenar al modelo y conseguir, a partir de un LLM básico de características espaciales limitadas, un modelo que sepa actuar en el contexto de la entrevista virtual. Finalmente, se realiza un análisis con herramientas de inteligencia artificial para comparar los diferentes modelos y el uso de las técnicas RAG en cada uno de ellos.

El proceso comienza con la elección de diferentes modelos de lenguaje. Esta elección viene dada por las limitaciones físicas que presentan las diferentes tecnologías utilizadas. Posteriormente, se selecciona un corpus de entrenamiento adecuado que contenga variedad de entrevistas e información relevante como pautas a seguir como entrevistador y calidad de respuestas del sujeto entrevistado. A continuación, se aplican técnicas RAG al LLM para adaptarlo a la tarea de entrevistar, consiguiendo una capacidad para, en el contexto dado, generar respuestas pertinentes y coherentes.

Finalmente, la evaluación consiste en la comparación de diferentes configuraciones en los modelos empleados, así como la influencia de las técnicas RAG en el desempeño final del entrevistador virtual. Las respuestas de los modelos que se analizan se obtienen utilizando un banco de preguntas que se ejecutará en todos los modelos. Con los resultados se identifican los mejores modelos y las técnicas más efectivas para la generación de entrevistas automáticas, que proporcionarán información para futuras investigaciones en el campo.

Para el entrenamiento de los modelos se van utilizar diferentes tecnologías en cada área. Para la generación de código se utiliza el lenguaje Python en cua-

dernos de Jupyter. Se utiliza como editor de texto Visual Studio Code para las consultas locales, y Google Colab desde la nube para el uso de elementos físicos necesarios para llevar a cabo el entrenamiento. El uso de estas tecnologías supone limitaciones físicas que se tratarán en este trabajo.

1.2. Motivación

Implementar un sistema como el que proponemos en este trabajo presenta una oportunidad para abordar diversos desafíos y mejorar significativamente el proceso de selección de personal. La motivación de esta propuesta aparece en la creciente necesidad de optimizar las prácticas de selección de individuos para puestos laborales en un mundo cada vez más digitalizado y competitivo.

La selección de personal sigue diferentes procesos dependiendo del puesto o empresa a la que se quiera acceder; sin embargo, la entrevista es un punto crucial que puede suponer desafíos tanto para entrevistadores como para los propios entrevistados. La subjetividad en la interpretación de las respuestas o los sesgos inconscientes son unas de las principales preocupaciones. La aparición de prejuicios personales o culturales pueden afectar a la evaluación de las habilidades de los entrevistados, resultando una discriminación hacia ciertos grupos. La falta de un criterio evaluador objetivo da como resultado una interpretación subjetiva de las respuestas, haciendo imposible un análisis firme de las competencias expuestas por la persona entrevistada.

Utilizar modelos de lenguaje de gran tamaño como entrevistador virtual ofrece diferentes ventajas. La primera a destacar es que los modelos pueden proporcionar estructuras firmes y objetivas en las entrevistas al seguir pautas predefinidas. Además, son capaces de adaptar su contexto en función de la situación del candidato. Al poder adaptarse, se pueden crear entrevistas más personalizadas y efectivas para los entrevistados. Los modelos pueden analizar a fondo las respuestas, consiguiendo resaltar aspectos que para el humano puedan pasar desapercibidos, como la repetición del lenguaje en las respuestas o la tendencia a esquivar parte de las preguntas realizadas.

Otra motivación reside en superar las limitaciones físicas y geográficas del proceso de selección tradicional. Con la reciente pandemia, han aumentado el número de entrevistas a través de plataformas online de videollamada, planteando nuevos desafíos a la hora de evaluar la validez para el puesto laboral. Con la utilización de tecnologías basadas en la nube y la utilización de LLMs, se puede conseguir aumentar el alcance de las entrevistas, haciendo así posible la participación de candidatos de diversas ubicaciones más eficientemente.

Con la creciente presencia de la tecnología en el mundo laboral es crucial que los procesos de selección se adapten a las nuevas herramientas disponibles, consi-

guiendo una respuesta a la demanda del mercado laboral actual. En un mercado en el que la reducción tiene una gran importancia, la accesibilidad a un reclutamiento online objetivo y automático permite reducir los costes en el personal y el tiempo utilizado.

Finalmente, gracias al uso de las nuevas tecnologías mencionadas, se pueden diseñar pruebas específicas para el puesto, consiguiendo realizar una selección de candidatos en función de sus capacidades frente a un problema designado y no frente a la subjetividad de la respuesta ante una pregunta.

1.3. Contenido de la memoria

El resto de la memoria está organizado como sigue. En el Capítulo 2 se introducen los conceptos necesarios sobre los LLMs y técnicas RAG. A continuación, los capítulos 3 y 4 se ocupan de exponer los aspectos principales del diseño y desarrollo del software. En los capítulos 5 y 6 encontramos las pruebas realizadas y las conclusiones y posibles líneas futuras de investigación.

Capítulo 2

Grandes modelos de lenguaje (LLMs)

2.1. ¿Qué es un LLM?

Como define Murray Shanahan en su articulo "Talking about large language models", *son modelos generativos matemáticos que analizan la distribución estadística de fragmentos en un vasto conjunto de textos accesibles públicamente, donde estos fragmentos incluyen palabras completas, partes de palabras o caracteres individuales como signos de puntuación, letras o símbolos.* [1]

Los modelos calculan la probabilidad de que una palabra siga a una cadena de palabras o símbolos procesada previamente, conocida como *prompt*. Con cada adición de una nueva palabra, se crea un nuevo *token* (palabras, juegos de caracteres o combinaciones de palabras y puntuación que han sido descompuestos por el LLM) que es verificado y se realiza una generación de una nueva cadena de texto final, permitiendo el proceso y generación de texto en un contexto dado.

Los LLMs son modelos preentrenados con grandes bancos de datos textuales que luego se ajustan a tareas concretas. Para operar, lo hacen como una red neuronal profunda con distintos parámetros. La red está formada por un número finito de capas de neuronas artificiales con las que realizar diferentes cálculos. Cada una de estas neuronas está conectada a otras diferentes, donde cada una de estas conexiones tiene un peso. La red neuronal profunda se basa en una estructura especial denominada transformer, que está preentrenada con datos e intenta maximizar la probabilidad de que la salida de texto predicha coincida con la de los datos de entrenamiento.

El modelo es entrenado para poder analizar y aprender patrones del lenguaje a partir de millones de corpus de texto con el uso de *machine learning*, en español, aprendizaje automático. Con este análisis y aprendizaje se consigue generar información relevante y coherente a la consulta realizada. El entrenamiento se basa en el proceso por el que se asignan unas salidas a unas entradas que han

sido proporcionadas como ejemplo. Los grandes modelos de lenguaje se engloban dentro del procesamiento de lenguaje natural (NLP, del inglés Natural Language Processing).

El procesamiento de lenguaje natural es un área de la inteligencia artificial que se centra en la conexión entre el lenguaje humano y el computador [2]. Entre sus aplicaciones destacan la traducción o la comprensión lectora. Hay que destacar que los modelos de lenguaje de gran tamaño no tienen una tarea principal designada. Sino que tienen un propósito más general, que puede ser adaptado según la necesidad de las tareas que hayan sido asignadas al LLM.

2.2. Modelos neuronales del lenguaje

EL funcionamiento de los LLMs se basa en las mencionadas redes neuronales, sistemas computacionales que estan basados en el comportamiento del cerebro humano. Una red neuronal esta compuesta por diferentes capas de neuronas conectadas entre sí, donde cada capa adquiere la información de la capa anterior y la procesa para enviar a la siguiente capa.

La primera capa es la capa de *embedding*, cuya función es convertir palabras o símbolos en vectores de números reales. Los vectores de embedding son actualizados con los otros parámetros del modelo y aprendidos durante el entrenamiento.

Un aspecto importante del funcionamiento del LLM es la asignación de pesos a diferentes partes de la entrada, conocido como mecanismo de atención del LLM. Con el mecanismo se le asignan niveles de importancia a cada entrada para de esta manera, comprender mejor el contexto , y generar una salida de texto más acorde al *prompt* propuesto como entrada. La estructura consiste en la composición de múltiples capas de atención que proporcionan al modelo la capacidad de enfocarse en más de un aspecto de la entrada.

El texto final se genera con la capa de salida, otro componente crucial en la arquitectura de los modelos. Esta capa depende de la tarea propuesta y puede ser de regresión, para generación de texto libre, *softmax*, si queremos clasificar símbolos, o la combinación de ambas capas para objetivos más complejos como la traducción de frases o palabras.

Cada red neuronal sigue diferentes arquitecturas de las que podemos destacar *Transformers*, redes neuronales recurrentes o convulacionales. A continuación se describen estas arquitecturas:

- **Transformers**

  La red neuronal predominante en muchos modelos es la red transformers

o transformador, que destaca por su procesamiento de secuencias de datos como frases o párrafos de una manera eficiente. En función del tamaño del modelo, existen capas apiladas de transformadores, cada una de estas más abstracta que la anterior, permitiendo comprender las estructuras lingüísticas de una manera jerárquica y compleja. Si se utiliza para modelación del lenguaje causal, la entrada que recibe es una secuencia de palabras y la salida es una predicción de cuál será la siguiente palabra.

Las capas mencionadas se utilizan para construir representaciones contextuales de los significados de las palabras o tokens de entrada. En cada capa de un transformer, para calcular la representación de una palabra i, combinamos la información de la representación de i en la capa anterior con la información de las representaciones de las palabras vecinas [3]. Los ejemplos de transformers más famosos son BERT, GPT y T5.

- **Redes neuronales recurrentes (RNNs)**

  Estas redes son útiles para las tareas relacionadas con las secuencias de texto y sirven para la generación de texto, modelado de lenguaje o traducción automática. Tienen conexiones retroalimentadas para poder capturar dependencias temporales y procesar las secuencias de los escritos. Sus variantes populares son LSTM y GRU.

- **Redes neuronales convolucionales (CNNs)**

  Las CNNs tienen como uso principal procesar imágenes pero también son aplicables al lenguaje natural. Existen CNNs multicapas y pre-entrenadas para clasificar texto. Su característica clave es la extracción de características locales. Se utilizan para clasificar texto, clasificar noticias o detectar SPAM.

También existen los modelos de atención, modelos basados en memoria o modelos generativos adversarios. Se describen a continuación:

- **Modelos de atención**

  La idea de estas redes es el uso de una red codificadora que toma una secuencia de entrada y crea una representación contextualizada de la misma, a menudo llamada el contexto. Se combinan con otras arquitecturas como transformers o Seq2Seq (una arquitectura basada en las RNNs). Sus aplicaciones son las respuestas a preguntas o resumen automático.

- **Modelos basados en memoria**

  Estos modelos basan su funcionamiento en la utilización de una memoria explícita para almacenar información relevante y poder acceder a ella durante el procesamiento del lenguaje. Destacan en en tareas donde se requiere

un razonamiento basado en el conocimiento como la inferencia de relaciones entre entidades.

Dentro de los modelos basados en memoria podemos destacar que esta arquitectura utiliza los principios de las máquinas de Turing en las redes neuronales. Su funcionamiento se basa en una memoria externa, donde la red puede acceder y modificar información, consiguiendo aprender algoritmos y realizar tareas algorítmicas con mayor eficiencia que una red neuronal tradicional. Las NTM se utilizan en la navegación visual o comprensión del lenguaje.

- **Modelos generativos adversarios (GANs)**

  Los GANs tienen como característica el uso de dos redes neuronales que crean una competencia entre ellas, cada una de las redes se entrena de manera adversial, una actúa como generador y la otra como discriminador. Estos modelos son característicos en la mejora de imágenes o síntesis de datos.

2.3. Ejemplos de LLMs

Los modelos se basan en diferentes arquitecturas y dependiendo de la tarea se utiliza un transformador, un modelo generativo adversario u otro tipo de arquitectura. Existen miles de modelos diferentes entrenados para tareas como transformar imagen en texto, convertir imágenes en vídeo , clasificar texto o generarlo. De todas las posibilidades, vamos a destacar los modelos que se utilizaran posteriormente en el programa con el objetivo de simular un entrevistador virtual.

2.3.1. GPT2

Según [4], *un modelo de transformadores preentrenado en un corpus muy grande de datos en inglés de forma autosupervisada. Esto significa que fue entrenado previamente solo con los textos sin procesar, sin que ningún ser humano los etiquete de ninguna manera (razón por la cual puede usar una gran cantidad de datos disponibles públicamente) con un proceso automático para generar entradas y etiquetas a partir de esos textos. Más precisamente, fue entrenado para adivinar la siguiente palabra en oraciones.*

Aunque en el trabajo solo hagamos uso de GPT2 por las limitaciones espaciales, es importante mencionar cómo este modelo ha ido evolucionando posteriormente. El modelo creado por OpenAI vió la luz en 2018 con GPT1 contando con alrededor de 117 millones de parámetros. Evolucionó posteriormente a GPT2 para contener 1.5 millones de parámetros, reflejando el progreso de los modelos

de procesamiento de lenguaje natural. En 2020 se lanzó GPT3, conteniendo 175 mil millones de parámetros, lo que supuso un cambio gigante en el rendimiento, creando debates sobre la ética de la utilización de estos modelos. La última versión que se puede utilizar es GPT4, que maneja más de 1 billón de parámetros.

Como se puede observar en la figura 3.1 [5], el porcentaje de las respuestas acertadas con estos modelos ha ido creciendo enormemente desde el 2.8 hasta casi el 45 por ciento.

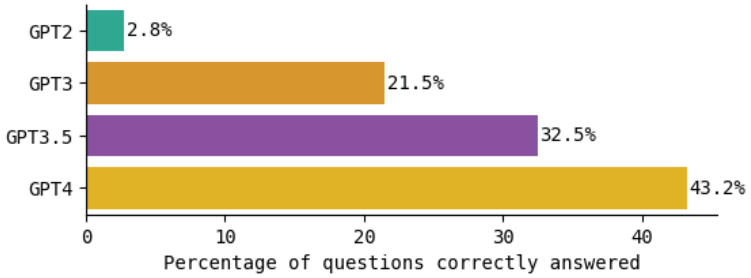

Figura 2.1: Porcentaje de preguntas acertadas por los diferentes modelos de GPT

2.3.2. Llama2

Como se define en [6], *en este trabajo, desarrollamos y lanzamos Llama 2, una colección de modelos de lenguaje grande (LLM) previamente entrenados y ajustados que varían en escala de 7 mil millones a 70 mil millones de parámetros. Nuestros LLM perfeccionados, llamados Llama 2-Chat, están optimizados para casos de uso de diálogo. Nuestros modelos superan a los modelos de chat de código abierto en la mayoría de los puntos de referencia que probamos y, según nuestras evaluaciones humanas de utilidad y seguridad, pueden ser un sustituto adecuado de los modelos de código cerrado. Proporcionamos una descripción detallada de nuestro enfoque para el ajuste y las mejoras de seguridad de Llama 2-Chat para permitir que la comunidad aproveche nuestro trabajo y contribuya al desarrollo responsable de los LLM.*.

La figura 2.2 [7] muestra como se desenvuelve Llama2 en sus diferentes versiones frente a GPT4 y un humano. El modelo utilizado en este trabajo es Llama2.7B (7 billones de parámetros) por las limitaciones físicas, existiendo una gran diferencia con sus modelos evolucionados.

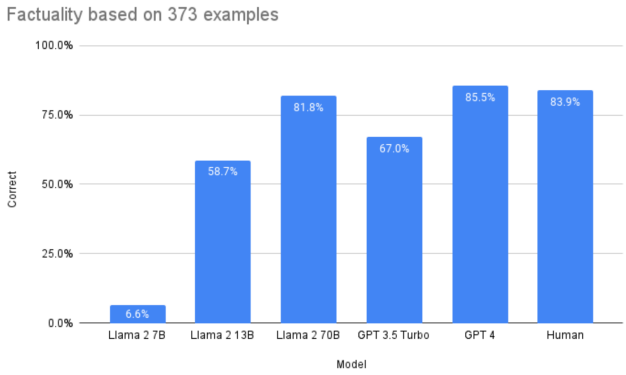

Figura 2.2: Comparativa entre Llama2, GPT4 y Humano

2.3.3. Mistral

Como indican los autores, *presentamos Mistral 7B v0.1, un modelo de lenguaje de 7 mil millones de parámetros diseñado para un rendimiento y eficiencia superiores. Mistral 7B supera a Llama 2 13B en todos los puntos de referencia evaluados y a Llama 1 34B en razonamiento, matemáticas y generación de código. Nuestro modelo aprovecha la atención de consultas agrupadas (GQA) para una inferencia más rápida, junto con la atención de ventana deslizante (SWA) para manejar de manera efectiva secuencias de longitud arbitraria con un costo de inferencia reducido. También proporcionamos un modelo ajustado para seguir instrucciones, Mistral 7B - Instruct, que supera al modelo Llama 2 13B - Chat tanto en puntos de referencia humanos como automatizados. Nuestros modelos se lanzan bajo la licencia Apache 2.0.* [8]

Se utiliza Mistral en su versión 7B para el desarrollo del proyecto.

2.3.4. Gemma

Los modelos Gemma demuestran un sólido desempeño en los puntos de referencia académicos para la comprensión, el razonamiento y la seguridad del lenguaje. Se han lanzado dos tamaños de modelos (2 mil millones y 7 mil millones de parámetros) y proporcionado puntos de control previamente entrenados y ajustados. Gemma supera a los modelos abiertos de tamaño similar en 11 de 18 tareas basadas en texto, y presenta evaluaciones integrales de los aspectos de seguridad y responsabilidad de los modelos. [9]

Se utiliza la versión de Gemma1.1 en la versión 7B, que es una pequeña actualización frente al modelo original de Gemma.

2.4. Técnicas RAG

Retrieval Augmented Generation o RAG es una arquitectura enfocada a mejorar la eficacia de los grandes modelos de lenguaje en diferentes tareas aprovechando información personalizada. El comienzo del uso de las técnicas RAG surge

en 2020, cuando Patrick Lewis publica junto a Ethan Perez, un articulo titulado "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", donde explican una nueva técnica para hacer más precisos los modelos. *Exploramos un ajuste fino de propósito general para la generación aumentada por recuperación (RAG), modelos que combinan memoria paramétrica y no paramétrica preentrenada para la generación de lenguaje. Introducimos modelos RAG donde la memoria paramétrica es un modelo seq2seq preentrenado y la memoria no paramétrica es un índice de vectores densos de Wikipedia, accesado con un recuperador neuronal preentrenado.*[10]

Muchos modelos han sido entrenados con millones de parámetros para responder a la generación de texto en diferentes áreas. Sin embargo, un gran número de ellos, una vez han sido entrenados, no pueden acceder a nueva información, convirtiéndose en una figura de información estática sin posibilidad de actualización. La imposibilidad de aceptar nueva información puede crear consultas erróneas, ya sea porque la documentación es antigua y se ha actualizado más tarde que el entrenamiento del modelo, o por la malinterpretación de datos. El uso de técnicas RAG busca solventar estas preocupaciones, entrenando al modelo y logrando recuperar información relevante para la tarea propuesta, e incorporándola para darle un contexto más preciso. Este contexto se consigue concatenando los nuevos documentos con la solicitud original, alimentando al generador de texto para producir la respuesta final.

La arquitectura RAG combina la recuperación de información con la ampliación del contexto y la generación de nuevas cadenas de símbolos más coherentes a la tarea impuesta. El uso de esta técnica sigue la siguiente estructura:

- **Recuperación (Retrieval)**

  Para llevar a cabo el proceso de recuperación se identifican fragmentos de texto que pueden ayudar en el contexto para posteriormente ser recuperados, apareciendo en la generación de información posterior. Esta búsqueda se puede dar en datos encontrados en bases de datos, archivos descargados de la nube o corpus específicos.

- **Ampliación (Augmentation)**

  Con la información relevante recuperada, se pasa a la fase de ampliación, que busca aumentar la información del fragmento recuperado. Las técnicas RAG corrigen errores o adaptan el contenido a la situación.

- **Generación (Generation)**

  La última etapa consiste en la generación de nuevo texto a partir del contenido que ha sido previamente recuperado y ampliado. Dependiendo de

la tarea encomendada, se podrán generar nuevas respuestas o textos descriptivos adaptados a la situación propuesta, entre otras múltiples opciones.

En la figura 2.3, adaptación de [11], se observa el funcionamiento de la generación de recuperación aumentada donde se realiza una petición. En la primera fase se busca la información más relevante y se recupera, para poder aumentar el contexto en la siguiente fase. Finalmente, se genera la nueva cuestión, añadiendo la información recuperada y aumentada.

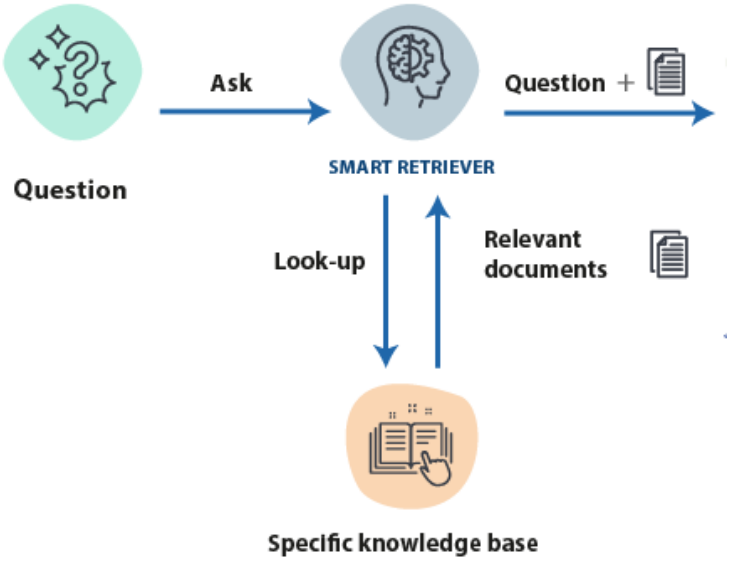

Figura 2.3: Fases de RAG

2.5. Entrenamiento LLMs con técnicas RAG

Mejorar la generación de texto en LLMs es posible implementando el uso de técnicas RAG. La incorporación de estas técnicas se lleva a cabo aplicando diferentes pautas a cada una de las fases mencionadas en el punto anterior.

- Para llevar a cabo la búsqueda en la fase de recuperación, se adapta el modelo, transformando la documentación y/o ejemplos a vectores. Esta transformación se logra con el uso de técnicas de representación de texto, donde encontramos modelos de incrustación de palabras o documentos (word embeddings o document embeddings). Posteriormente, se crea el llamado índice vectorial, que es la parte integral de la fase, donde se organizan los vectores como una estructura de datos de fácil acceso mediante la búsqueda por similitud. Las estructuras de datos predominantes para este tipo de consultas son los árboles KD, las tablas hash o las estructuras basadas en grafos, entre otras.

- A continuación, cuando los vectores más relevantes para la consulta han sido encontrados, se recupera la información asociada a estos. Esta información

se amplía en la segunda fase y se concatena junto con la entrada original del modelo, para en última estancia, generar un texto más coherente y contextualmente relevante en la fase de generación.

- El último paso es enviar la nueva consulta generada al modelo que hayamos escogido para trabajar, para que la procese y envié la respuesta final. Existen dos tipos de procesado:

  - Procesado estándar, tratando la consulta de manera similar a otras entradas de texto.
  - Procesado adaptable, donde el LLM puede ajustar los pesos y/o capas del modelo durante la generación del texto en la última fase, para tener en cuenta la información proporcionada por las técnicas RAG utilizadas de manera más directa.

La figura 2.4 [12], muestra el proceso que sigue un *query* o consulta con el uso de técnicas RAG. La consulta se envía y realiza la fase recuperación, donde se buscan vectores indexados con la información relevante (esta información ha sido transformada en vectores previamente por *embeddings*), para posteriormente aumentarla y enviarla como nuevo contexto para la primera consulta. El resultado final es enviado al LLM para generar la respuesta final.

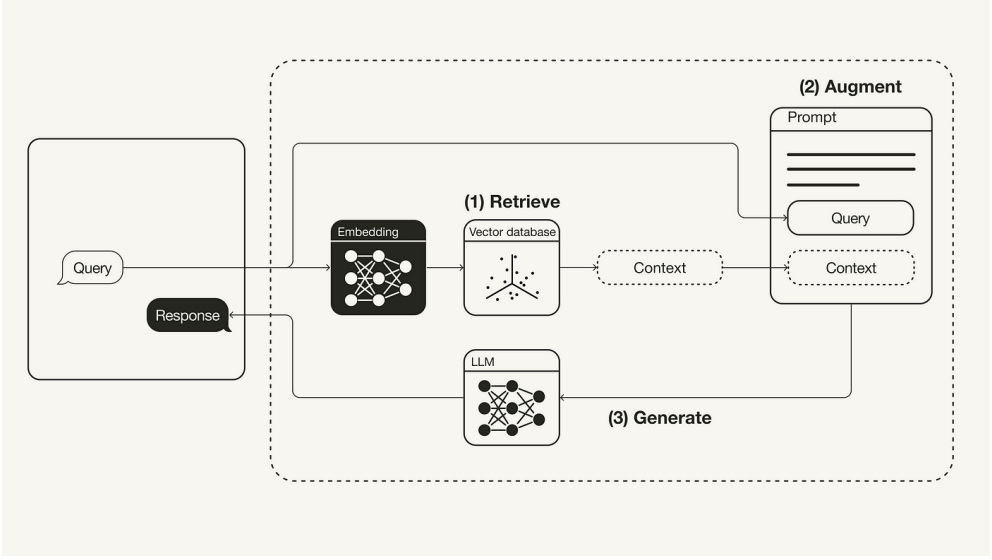

Figura 2.4: Generación aumentada por recuperación en LLM

Capítulo 3

Diseño del software

El diseño del software es un aspecto crucial en el desarrollo del proyecto. En este capítulo se tratan los aspectos esenciales del diseño, entre los que se incluyen las diferentes tecnologías empleadas y sus limitaciones y la metodología seguida para la realización del proyecto.

3.1. Tecnologías empleadas

En este apartado se detallan las tecnologías clave para desarrollar el sistema propuesto, haciendo especial énfasis en aquellas que no se han estudiado a lo largo del grado. Las tecnologías abarcan desde el lenguaje de programación utilizado, hasta los elementos hardware necesarios. Además, se detallan los editores de código o las librerías y sistemas utilizados para desarrollar el código.

El desarrollo de la propuesta inicial ha ido cambiando en función de las limitaciones de las siguientes tecnologías. La aparición de las limitaciones ha resultado en el uso de más tecnologías. En el siguiente apartado (3.2) se encuentran con más detalles las limitaciones, tanto físicas como temporales.

3.1.1. Python

El lenguaje seleccionado para desarrollar el código ha sido Python, no solo por la simplicidad que ofrece su escritura, sino también por la gama de librerías que ofrece para crear código relacionado con los LLM.

Python es un lenguaje de alto nivel de pogramación interpretado, que hace especial incapié en la legibilidad de su código. Posee una licencia de código abierto y es uno de los lenguajes de programación más populares en la actualidad.

La versión de Python utilizada es la 3.10.12, publicada el 6 de Junio de 2023 [13].

3.1.2. Jupyter Notebooks

Para la ejecución del código creado se hace uso de los cuadernos de Jupyter, por sus ventajas a la hora de analizar el código ejecutado y de crear una documentación enriquecida de manera simple. Además, la herramienta soporta el lenguaje de Python, por lo que es idónea.

El cuaderno Jupyter es un software de código abierto basado en una herramienta de navegador que funciona de tal manera que se pueden apoyar flujos de trabajo, código, datos y visualizaciones que detallan el proceso de investigación [14].

Los cuadernos de Jupyter permiten ejecutar código por bloques, de tal manera que no es necesario ejecutar todo el código de nuevo en cada nueva ejecución de prueba. El funcionamiento consiste en la creación de un entorno de ejecución en el que las variables o librerías importadas en los bloques se van guardando, para poder usarlas en los siguientes bloques sin la necesidad de volver a cargarlas.

Esto es muy útil en nuestro proyecto, ya que los modelos utilizados necesitan descargarse en cada ejecución del proceso, por lo que al crear un bloque independiente para su descarga, solo se descargarán una vez. Una vez operado el bloque del modelo, nos permitirá acceder a él desde nuevo código sin la necesidad de una nueva carga. De esta manera, conseguimos aislar el código para realizar las pruebas que queramos independientemente.

3.1.3. Hugging Face

La elección de la plataforma Hugging Face en el desarrollo del trabajo se justifica con que es una de las plataformas líderes en inteligencia artificial, centrada en el procesamiento de lenguaje natural y aprendizaje automático.

Hugging Face es una empresa estadounidense de almacenamiento global de módulos de *deep learning*. Los modelos de lenguaje se ubican en un servicio web centralizado, *Hugging Face Hub*, creado por la propia empresa. Existen más de 120000 modelos y han participado mas de 50000 organizaciones, de las que podemos destacar empresas reconocidas como Google o Microsoft entre otras [15]. En la página web se publican los diferentes modelos en repositorios basados en Git, incluyendo un control de versiones con funciones similares a GitHub, para facilitar el uso o aprendizaje de los mismos a personas externas a las empresas creadoras de los propios modelos. Además, esta empresa ha incorporado la librería transformadores para aplicaciones de NLP (Natural Language Processing).

En su página web encontramos todos los modelos que han sido entrenados, clasificados por la tarea que van a realizar, ya sea procesamiento del lenguaje natural (para clasificación de texto), multimodales (como asistentes de respuesta de

documentos) o modelos de tareas relacionadas con el audio. Además, se pueden encontrar diferentes bases de datos que no utilizaremos en el proyecto. Existe un apartado de búsqueda rápida donde podemos buscar por modelo o autor.

Todos los modelos mencionados en el apartado 2.3 del proyecto han sido utilizados con la ayuda de esta plataforma. En el apartado de modelos de generación de texto en la página web podemos encontrarlos [16].

El funcionamiento para el uso de la página es el siguiente;

- Acceso a la plataforma

  Para utilizar tanto los modelos como las bases de datos debemos crear una cuenta asociada a Hugging Face, que nos permitirá tener asociado un Token de acceso de usuario con el que seremos identificados.

- Búsqueda y permisos del Modelo

  Una vez accedidos, se busca el modelo a utilizar. Cuando se ha encontrado se requerirá guardar el nombre del modelo, para cargarlo posteriormente en el código. Además, muchos de ellos, aunque tengan una licencia libre, requieren de aceptar una serie de condiciones y términos. Asimismo, en algunos modelos se requiere de permiso para su uso. De los utilizados en el proyecto, Llama2, mencionado en el punto 2.3.2, requiere de un permiso de acceso por los autores del repositorio.

- Carga y utilización del modelo

  Finalmente, podemos hacer uso del modelo que hemos seleccionado cargándolo en el código. Para la incorporación del mismo se hace uso de otra tecnología que encontramos en el apartado 3.1.6.

Figura 3.1: Hugging Face Logo

3.1.4. Google Colab

Debido a que ejecutar LLMs en modo local resulta imposible por la demanda de recursos computacionales de este tipo de modelos, se decidió utilizar hardware

accesible en la nube. Se barajaron diferentes opciones, entre ellas Azure y Google Colab.

- Azure nos permitía crear máquinas virtuales a cambió de un crédito valorado en dólares. Al ser estudiante, la plataforma añadía un valor de cien dólares para gastar anualmente gratuitamente. Sin embargo, la creación de las máquinas con el hardware necesario para llevar a cabo la carga de los modelos de lenguaje y la posterior utilización de las técnicas RAG en ellos, superaba el crédito aportado, por lo que se decidió no utilizar la plataforma.

- De entre las otras opciones que se barajaron, Google Colab fue la solución al problema de las limitaciones físicas que encontramos, ya que aportaba el hardware necesario para hacer uso de las técnicas RAG en los grandes modelos de lenguaje. Utlizando Google Colab tenemos la ventaja de permitir mayor capacidad de cálculo, ya que se utiliza una GPU.

  La GPU esta diseñada para realizar cálculos en paralelo, por lo que se convierte en un recurso necesario para operaciones paralelas. Esto se logra distribuyendo las tareas entre los múltiples núcleos de procesamiento de la GPU, consiguiendo reducir drásticamente el tiempo necesario para crear el índice vectorial o utilizar los modelos comparado con el uso de la CPU.

  Google Colab es un servicio gratuito de nube alojado en Cuadernos de Jupyter que no requiere configuración y ofrece acceso gratuito a recursos de computación como CPUs y GPUs, lo que hace que sea una solución adecuada para el aprendizaje automático y la ciencia de datos.

  La plataforma esta asociada a una cuenta de Google, que posteriormente nos será útil en la carga de datos. El funcionamiento es simple, al no requerir de ninguna configuración porque la ejecución se da en una máquina de Google, se crean los ficheros necesarios y se comienza un entorno de ejecución. El entorno permite una ejecución mediante CPU, GPU y TPU limitadas, estos valores limitados se tratarán detenidamente en el apartado 3.2.

  A pesar de las limitaciones que sigue teniendo utilizar esta plataforma, era la que más satisfacía nuestras necesidades, por esto, se utilizó Google Colab como editor de texto en modo GPU T4, resolviendo los siguientes problemas previamente comentados;

  - Necesidad de un entorno de ejecución con una unidad de procesamiento gráfico.

- Entorno que soporte Cuadernos de Jupyter para realizar la descarga del LLM una sola vez y poder crear código referente al modelo después sin necesidad de volver a cargarlo.
- Entorno desde el que podamos acceder a diferentes archivos ubicados localmente o en la nube. Este punto se trata en el siguiente apartado, 3.1.5.

Cabe recalcar que en la carga de los modelos de Hugging Face en el código, el hardware era insuficiente, y las capacidades de la GPU y la RAM de Google Colab se veían superadas. Para solventar el problema se decidió hacer uso de la cuantización de los modelos, que queda explicada en detalle en el punto 3.1.7.

3.1.5. GoogleDrive

Para poder acceder a la información que posteriormente se indexará con las técnicas RAG para generar un contexto más apropiado para crear el entrevistador virtual, se ha decidido hacer uso de Google Drive.

Google Drive es un servicio de Google asociado a la cuenta de Google en el que podemos almacenar documentos y la información a la que queremos acceder. En el caso de este proyecto, es necesario el acceso desde Google Colab (apartado 3.1.4) a algún medio de almacenamiento para hacer uso de la generación aumentada por recuperación. Al estar asociado este a una cuenta de Google, se puede utilizar Google Drive con una instrucción de código para que se permita el acceso al mismo para operar con él en lo que resta de código.

3.1.6. Transformers

La primera librería utilizada para el desarrollo del proyecto es *Transformers* de HuggingFace. Esta herramienta es un marco de código abierto que ofrece una API y herramientas para descargar modelos previamente entrenados de última generación. Esto convierte la librería en una de las claves para hacer uso de los modelos de HuggingFace, a los que hemos accedido previamente como se ha mencionado en el punto 3.1.3. La librería es compatible con los cuatro modelos que se van a utilizar: GPT2, Mistral, Gemma y Llama2 [17].

Además, se pueden adaptar los modelos para optimizarlos y maximizar su rendimiento. Esto se logra con el ajuste de diferentes parámetros, entre los que destacamos:

- *Model_name*: Con este parámetro especificamos el nombre del modelo preentrenado de HuggingFace que se utilizará.
- *Tokenizer_name*: Se selecciona el tokenizer asociado al modelo, para convertir el texto en tokens. Generalmente coincide con el nombre del modelo.

- *Max_new_tokens*: Este parámetro especifica el número máximo de nuevos tokens que el modelo puede generar como respuesta a la consulta.
- *Model_kwargs*: Permite especificar configuraciones adicionales relacionadas con la cuantificación del modelo. Con este ajuste logramos solventar el problema de superar las capacidades de Google Colab, se trata en profundidad en el siguiente apartado, cuantización.
- *Generate_Kwargs*: Contiene un diccionario adicional al que se le pasan métodos de generación de texto del modelo, como *temperature*, *top_k* o *top_p*, para controlar la creatividad y fluidez de las respuestas.

Los parámetros seleccionados para maximizar el funcionamiento del modelos se tratarán en el capítulo referido a las pruebas realizadas.

3.1.7. Cuantización con Pytorch

La cuantización de modelos es una técnica basada en conseguir utilizar grandes modelos de lenguaje con recursos limitados, como es el caso de nuestro proyecto, ya que solo contamos con los recursos disponibles en Google Colab. Con la cuantización se reduce la precisión de los modelo utilizados. Gracias a esto, se logra un modelo más eficiente respecto al uso de la memoria sin reducir significativamente su rendimiento.

Para lograr cuantizar los diferentes modelos utilizados en el proyecto se hace uso de la librería PyTorch. Esta librería ofrece funciones para llevar a cabo la cuantización de los modelos de manera eficiente mediante el uso de GPUs y CPUs [18]. Una de las características de PyTorch es que permite adaptar la cuantización según las limitaciones específicas del proyecto ajustando sus parámetros. De forma predeterminada, los modelos PyTorch utilizan puntos flotantes de 32 bits, lo que significa que un solo parámetro ocupa 32 bits en la memoria de la GPU. Durante la cuantización se pueden ajustar los diferentes tipos de precisión en 32, 16, 8 o 4 bits.

En el proyecto se ha utilizado la configuración de cuantización basada en la clase *BitsAndBytesConfig*, que es parte de las herramientas proporcionadas por Pytorch y permite especificar diversos parámetros de los que destacamos:

- *load_in_4bit*: Este parámetro indica si cargarán los parámetros del modelo en un formato de 4 bits. Si esta establecido a TRUE se cargarán de esta manera y se logrará un modelo más compacto y eficiente en cuanto al uso de la memoria.
- *bnb_4bit_compute_dtype*: Especifica el tipo de datos de coma flotante utilizado para los cálculos durante la cuantificación. Estos valores pueden ser de 16, 32 o 64 bits.

- *bnb_4bit_quant_type*: Indica el tipo de cuantización que se aplicará, de entre los que podemos destacar Non-Linear-4-bit, que es un método de cuantización no lineal que puede proporcionar una mejor precisión en comparación con la cuantización lineal.

- *bnb_4bit_use_double_quant*: Este parámetro especifica si se utilizará la doble cuantización. Cuando está establecido en True, se aplicará la cuantización de doble precisión, lo que puede ayudar a mejorar la precisión del modelo.

Teniendo en cuenta el contexto del proyecto en el que tenemos unas limitaciones físicas importantes en el uso de los modelos, se ha decidido hacer uso de las facilidades que nos aporta la cuantización de PyTorch a la hora de manipular un modelo de lenguaje de grandes dimensiones.

3.1.8. Llamaindex

Otra de las librerías de Python que vamos a utilizar es Llama Index. Esta librería es una pieza clave para la implementación de técnicas RAG en modelos de lenguaje de gran tamaño. LlamaIndex es un marco de datos simple y flexible para conectar fuentes de datos personalizadas a modelos de lenguaje grandes (LLMs)[19]. Al combinar la capacidad de indexación eficiente de Llama Index con la potencia de los LLMs, podemos crear sistemas de generación de texto que superan las limitaciones de aquellos con enfoques más tradicionales.

EL uso de esta librería en conjunto con los modelos de lenguaje y las técnicas de generación por recuperación aumentada permite recuperar información relevante de grandes conjuntos de datos (recuperación),organizarla de manera coherente y aumentarla (aumento) y generar las respuestas contextualmente adecuadas (generación).

En el desarrollo de nuestro proyecto, se utilizan funciones asociadas a Llama Index para:

- Cargar documentos de un directorio específico y crear nuestro servicio de contexto.

- Facilitar una indexación rápida de los elementos que vamos a utilizar en las técnicas RAG, consiguiendo que los documentos se representen como vectores en un espacio de características específico.

- Configurar el índice de almacenamiento de los vectores para que actúe como un motor de consulta, estableciendo las forma en la que se devolverán las respuestas de las consultas al índice.

Por lo comentado anteriormente, se decidió utilizar llama-index como librería para manejar los índices vectoriales.

Cabe destacar que el uso de esta librería comenzó con la versión 0.9.4, publicada el 1 de Enero de 2024. Se creó el código y el 12 de febrero de 2024, se publicó la versión 0.10.0 donde se introdujeron múltiples cambios, convirtiéndose en la mayor actualización del paquete hasta la fecha, donde destacan entre otros cambios;

- Creación de un paquete llama-index-core y división de todas las integraciones y plantillas en paquetes separados. Cientos de integraciones (LLMs, incrustaciones, almacenes de vectores, cargadores de datos, devoluciones de llamada, herramientas de agente, y más) ahora están versionadas y empaquetadas como paquetes separados de PyPI.
- LlamaHub será el hub central para todas las integraciones: el antiguo repositorio llama-hub se consolida en el repositorio principal llama-index. En lugar de que las integraciones se dividan entre la biblioteca principal y LlamaHub, todas las integraciones se listarán en LlamaHub[20].

Tras la gran actualización, el código del proyecto sufrió diferentes cambios, destacando la manera de importar desde la librería las funciones a utilizar. A pesar de que la documentación existente en la fecha sobre la nueva actualización era nula, se decidió invertir tiempo en aprender a utilizar la nueva versión y no forzar una anterior, consiguiendo aprovechar al máximo la librería.

3.1.9. Langchain

Para realizar la evaluación de la calidad de las respuestas generadas por los grandes modelos de lenguaje una vez aplicadas las técnicas RAG, se ha hecho uso de la librería Langchain. Esta librería facilita el proceso de análisis de respuestas ya que proporciona herramientas avanzadas que pueden ser utilizadas como evaluador.

Dentro de las diferentes herramientas que ofrece Langchain, se hace uso de *String evaluator*, un componente diseñado para evaluar el rendimiento de un modelo de lenguaje comparando los resultados generados con una cadena de texto utilizada como referencia, proporcionando una medida de la precisión o calidad del texto generado.

Se ha valorado el uso de diferentes implementaciones existentes en los *String evaluators*[21], pero algunas de ellas han dado problemas de compatibilidad y utilización en el programa del proyecto. Las opciones que se han barajado son las siguientes:

- Criteria Evaluation: Para evaluar el resultado de un modelo utilizando una rúbrica o un conjunto de criterios específicos, *Criteria Evaluation* es una herramienta útil. Con su uso, se verifica si el resultado de un LLM o el la respuesta generada por el LLM cumple con un conjunto definido de criterios

- Scoring Evaluator: Esta evaluación utiliza un modelo de lenguaje(por defecto GPT de OpenAI) para asignar a las predicciones del modelo utilizado en el proyecto, un valor en la escala del 1 al 10 basado en la rúbrica o criterio establecido.

- Embedding distance: Este evaluador se usa para medir la similitud (o disimilitud) semántica entre una predicción y una cadena de etiqueta de referencia. Retorna un valor correspondiente a la distancia, por lo que un número pequeño se asocia a una respuesta similar.

- String Distance: La evaluación consiste en asignar una distancia entre la cadena de salida del LLM y una cadena que introduce el usuario. Utiliza las métricas de distancia de la librería *rapidfuzz*. Como en la anterior opción, a menor distancia, mejor resultado.

Mientras que tanto *Scoring Evaluator* como *Embedding Distance* generan problemas de compatibilidad con el modelo que se utiliza(Utiliza GPT3 y se requieren licencias), *Criteria Evaluation* y *String Distance* se presentan como los mejores evaluadores. Al ser el primero un evaluador que aporta una opinión a partir del criterio que se indica y no una puntuación respecto de una referencia, se ha decidido no utilizar. Finalmente se utiliza *String Distance*, debido a que aporta un valor a las respuestas utilizando una referencia sin problemas de compatibilidad.

3.2. Limitaciones

Durante el desarrollo del proyecto, el uso de tecnologías avanzadas como los grandes modelos de lenguaje o la aplicación de técnicas RAG a ellos ha supuesto diversas limitaciones físicas y temporales. Las limitaciones han afectado al progreso del proyecto y han condicionado la implementación de las tecnologías mencionadas en el punto anterior.

3.2.1. Físicas

Las principales limitaciones físicas han estado relacionadas con la ausencia de hardware local, de un hardware en los servicios de le nube limitados y con el almacenamiento de los datos. En el inicio del proyecto se quiso realizar el programa donde cargar los LLM localmente, con la ayuda del editor de código Visual Studio Code, pero apareció la necesidad de GPU en el ordenador, limitando el proceso y pudiendo realizar solo parte del código. Tras la migración del programa al editor con acceso a herramientas hardware en la nube, Google Colab, aparecieron nuevas limitaciones dentro del programa. Google Colab cuenta con un espacio limitado de almacenamiento, y una RAM y GPU limitadas como aceleradores hardware.

- El espacio de almacenamiento máximo por entorno de ejecución es 78.2 gigabytes, donde se tienen que incluir tanto las librerías descargadas para el funcionamiento del código como los modelos utilizados. Siendo las librerías

estrictamente necesarias para el uso del código, el espacio restante nos limita a utilizar modelos de un tamaño menor a 30 gigabytes, siendo esta una de las razones por las que se han implementado los modelos mencionados en el apartado 2.3.

- La RAM que ofrece Google Colab consta de 12.7 GB, y con ella se realiza el índice vectorial para el uso de las técnicas RAG. Dado que es un hardware limitado, se requiere del uso de la GPU que ofrece Google Colab para la creación del índice.

- La GPU RAM disponible es la Tesla T4 de NVIDIA y consta de 15 gigabytes utilizables con los que se crea el índice vectorial y se realizan las consultas al modelo. Teniendo en cuenta este hardware, los modelos que se han utilizado son demasiado grandes y precisos para llevar a cabo las tareas propuestas, lo que supone la introducción de la cuantización como solución a esta limitación.

3.2.2. Temporales

Las limitaciones temporales han aparecido con la curva de aprendizaje, el uso del hardware en la nube cuando se han realizado las pruebas en Google Colab y en la integración de las tecnologías utilizadas.

- Curva de aprendizaje

  - El uso de todas las tecnologías que no se habían estudiado previamente en el grado ha supuesto tiempo adicional, no solo para investigar su uso, si no para su posible implementación en el proyecto. Hay que destacar el aprendizaje de Hugging Face, Google Colab y las librerías Pytorch, LlamaIndex y Langchain. Además, el proceso de selección y descarte de las tecnologías se ha llevado a cabo en base a prueba y error, lo que ha supuesto tiempo extra.
  - Las actualizaciones constantes de las librerías, como LlamaIndex 0.10.0, han implicado tiempo adicional para entender las novedades y adecuar el código existente.

- Pruebas en Google Colab

  Además de las ya mencionadas limitaciones físicas que ofrece Google Colab, existen limitaciones temporales asociadas al servicio. La plataforma permite el uso temporal de la misma, estando este uso definido por el número de demandas de la aplicación. Esto supone que diariamente Google Colab solo se pueda utilizar un tiempo concreto, oscilando este entre una hora y media y tres horas en los mejores casos.

- Configuración de entornos

La garantía de que las versiones de las librerías y herramientas fueran compatibles y no existiesen conflictos entre ellas fue una limitación temporal que supuso una tarea continua en el desarrollo del proyecto.

3.3. Metodología

Para desarrollar el proyecto, la metodología empleada ha sido uno de los aspectos clave para la evolución del trabajo. Se ha utilizado una adaptación de metodología *scrum* [22], *scrum* divide el trabajo en periodos cortos llamados *sprints*, que pueden tener una duración de 1 a 4 semanas, y promueve la visibilidad de los diferentes aspectos que aparecen en el proyecto. Además, esta metodología crea diferentes roles para cada ocupante del proyecto (*Product Owner*, *Scrum Master*, o *Development Team*, entre otros).

No obstante, el proyecto no se ha basado estrictamente en la metodología *scrum*, sino que se ha adaptado para utilizar únicamente los sprints, facilitando los cambios debidos a los problemas que han ido surgiendo. En cada uno de los sprints se ha incluido una planificación de las tareas a realizar y su posterior ejecución y evaluación.

Para el control de versiones se ha utilizado GitLab, un servicio en la nube que permite a los desarrolladores y equipos de trabajo almacenar y gestionar el código fuente de sus proyectos [23]. Como se ha utilizado Google Colab, no se ha podido seguir un control de versiones estricto ya que se ha tenido que descargar el código y subir a la plataforma cada vez que existiese un cambio significativo.

A continuación, se muestran los contenidos tratados en cada uno de los sprints del proyecto.

- Semanas 1-4: Búsqueda de información y pruebas

  Durante las primeras semanas se realizó una búsqueda intensiva de la información relevante al proyecto. Se estudiaron los puntos tratados en el capítulo 2 del proyecto, incluyendo qué es un modelo de lenguaje de gran tamaño, sus aplicaciones y modelos relevantes, y qué son las técnicas RAG y cómo aplicarlas. Además, se realizaron pruebas simples de manejo de los LLM para comprobar su funcionamiento.

- Semana 5: Búsqueda de modelos

  Esta semana estuvo enfocada en la búsqueda de modelos interesantes que pudiesen ser útiles para el desarrollo en Hugging Face. Se decidió utilizar aquellos que cuadraban tanto con el desarrollo y tarea, como con las limitaciones físicas que fuesen apareciendo, siendo estos los mostrados en el capítulo 2, apartado 2.3.

- Semana 6: Implementación del modelo localmente

Este sprint consistió en la implementación del modelo Llama2 localmente con el uso de las técnicas RAG, se realizaron diversas pruebas para ver su funcionamiento. Sin embargo, apareció el primer problema importante en el desarrollo, la necesidad de una GPU para crear el índice vectorial y las consultas al modelo. Este problema se trata en el apartado 3.2.

- Semanas 7-8: Implementación del modelo en la nube

  Para solventar el problema se implementó el modelo en Google Colab (punto 3.1.4). Se modificó y creó código para el correcto funcionamiento del programa, y también se realizaron pruebas para ver el funcionamiento de las técnicas RAG accediendo a información creada por el usuario, exclusivamente para responder preguntas sencillas (nombre y domicilio de un nombre guardado en un archivo en Google Drive para su acceso).

- Semana 9: Resolver problemas de actualización de software

  Durante esta semana llamaindex sufrió una actualización de software muy importante, la versión 0.10.0, y se dedicó el tiempo a la modificación del código para su funcionamiento. Debido a la falta de documentación online, llevó más tiempo del esperado.

- Semana 10: Implementación de información relevante en las técnicas RAG

  La información relevante que se utiliza en el proyecto para aprovechar las técnicas RAG en los grandes modelos de lenguaje se buscó durante este sprint. Esta información abarca PDFs acerca de la entrevista virtual o puntos positivos y negativos de una entrevista.

- Semanas 11-12: Pruebas

  Para comprobar el funcionamiento del programa y su utilidad, se llevaron a cabo pruebas que se especifican en el capítulo 5.

- Semana 13: Evaluación de resultados

  Finalmente, se llevó a cabo una evaluación de los resultados que se obtuvieron en las pruebas, para realizar la evaluación se utilizó la librería Langchain(punto 3.1.9).

- Semanas 14-20: Escritura de la memoria

  En las últimas semanas se realizó el escrito de la memoria, recabando la información y desarrollos documentados durante las anteriores semanas.

Las semanas son orientativas y existe variación de 1-3 días en algunos sprints, pero nunca la solapación de uno con otro. Además, durante las semanas 6-10, se aprendió a utilizar las librerías transformers, PyTorch y llamaindex, correspondientes a los puntos 3.1.6, 3.1.7 y 3.1.8.

Capítulo 4

Desarrollo del software

En este capítulo se detalla como se ha estructurado el programa, tanto su arquitectura y caso de uso como el desarrollo del código.

4.1. Arquitectura y caso de uso

La arquitectura del sistema se basa en un modelo cliente-servidor, donde el frontend actúa como interfaz de usuario (cliente), enviando las consultas al backend (servidor). La interfaz referida al cliente realiza la interacción directa con el usuario desde el chatbot o la interfaz pregunta-respuesta. En cuanto al servidor, recibe las consultas del cliente, maneja la comunicación con el almacenamiento y los modelos de lenguaje, y las devuelve al frontend para ser presentadas al usuario.

El servidor cuenta con componentes de almacenamiento, procesamiento del lenguaje natural y el uso de técnicas RAG. El almacenamiento consta de los archivos de Google Drive que posteriormente son indexados con un índice vectorial. Los modelos de lenguaje de Hugging Face son cuantizados para el correcto funcionamiento y utilizados para generar las respuestas a las consultas del cliente. Las técnicas RAG se aplican en el proceso de generar la respuesta final.

A continuación, se detalla el caso de uso del programa. Como se observa en la figura 4.1, el funcionamiento es similar al explicado en la utilización de técnicas RAG en los grandes modelos de lenguaje, detallado en el capítulo 2. El flujo del proceso es el siguiente:

- 1. El usuario ingresa la consulta a través del frontend.
- 2. El frontend envía la información al backend.
- 3. El backend consulta el almacenamiento para recuperar (R) información relevante al contexto dado por el usuario (ficheros en Google Drive).
- 4. Se enriquece el contexto con la información recuperada (A) y se genera (G) la nueva consulta que se envía al LLM.

- 5. El LLM procesa la información y genera la respuesta.
- 6. La respuesta se envía al backend, para ser enviada al frontend y ser finalmente presentada al usuario.

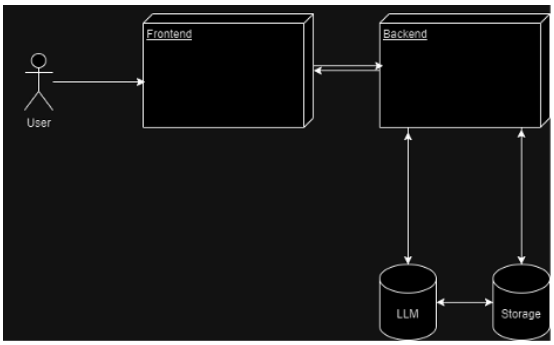

Figura 4.1: Caso de uso

4.2. Desarrollo del código

En este apartado se presenta el código utilizado para desarrollar el programa, se hace especial énfasis en diferenciar el codigo desarrrollado del adaptado de otras fuentes. A continuación se muestran las secciones en las que se ha dividido el código en formato Jupyter Notebooks utilizado en Google Colab.

4.2.1. Preparación del programa

Para acceder a los documentos que utilizaremos en la implementación de las técnicas RAG, se conecta el programa con el almacenamiento en Google Drive. Se inicia sesión en la cuenta de huggigface para la posterior carga y utilización de los modelos.

Listing 4.1: Google Drive y Hugging Face

```
from google.colab import drive
drive.mount('/content/drive')

!huggingface-cli login
```

Se descargan en el entorno las librerías que se van a utilizar (no se fija una versión porque son compatibles entre todas en la última versión) y se importa lo necesario para el desarrollo.

Listing 4.2: Librerias e imports

```
!pip install llama-index transformers accelerate bitsandbytes
    pypdf langchain llama_index.embeddings.huggingface

from llama_index.core import VectorStoreIndex,
    SimpleDirectoryReader
from llama_index.llms.huggingface import HuggingFaceLLM
```

```
from llama_index.core import Settings
import torch
from transformers import BitsAndBytesConfig
from llama_index.core.prompts import PromptTemplate
from llama_index.core import ServiceContext
from llama_index.core.response.notebook_utils import
    display_response
import sys
```

4.2.2. Cuantización y prompts

Se crea la cuantización que se utilizará posteriormente en el modelo. El código ha sido adaptado del artículo "retrieval augmented generation using open source llms"[24], para satisfacer las limitaciones físicas del proyecto. También se define el modo en que se convierten los mensajes a prompt con el código del anterior articulo.

Listing 4.3: Cuantización

```
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
)
```

Listing 4.4: Mensajes a Prompt

```
def messages_to_prompt(messages):
  prompt = ""
  for message in messages:
    if message.role == 'system':
      prompt += f"<|system|>\n{message.content}\n"
    elif message.role == 'user':
      prompt += f"<|user|>\n{message.content}\n"
    elif message.role == 'assistant':
      prompt += f"<|assistant|>\n{message.content}\n"
# ensure we start with a system prompt, insert blank if needed
  if not prompt.startswith("<|system|>\n"):
    prompt = "<|system|>\n\n" + prompt

  # add final assistant prompt
  prompt = prompt + "<|assistant|>\n"

  return prompt
```

4.2.3. Carga del modelo

Para cargar el modelo, se adapta el código de HuggingFaceLLM [25]. Las limitaciones físicas hacen obligatorio el uso de la cuantización definida anteriormente y de la siguiente selección de parámetros.

Listing 4.5: Carga de modelo

```
llm = HuggingFaceLLM(
    model_name="model_name",
    tokenizer_name="model_name",
    query_wrapper_prompt=PromptTemplate("<|system|>\n\n<|user|>\n
       {query_str}\n<|assistant|>\n"),
    context_window=3900,
    max_new_tokens=256,
    model_kwargs={"quantization_config": quantization_config},
    # tokenizer_kwargs={},
    generate_kwargs={"temperature": 0.3, "top_k": 50, "top_p":
       0.95},
    messages_to_prompt=messages_to_prompt,
    device_map="auto"
)
```

4.2.4. Aplicación de técnicas RAG

Para aplicar las técnicas RAG, se accede a la carpeta en Google Drive y se crean tanto el servicio como el índice vectorial. Además, se fija la respuesta a las preguntas en modo compacto. Código del artículo "retrieval augmented generation using open source llms"[24].

Listing 4.6: Ténicas RAG

```
service_context = ServiceContext.from_defaults(llm=llmMistral,
    embed_model="local:BAAI/bge-small-en-v1.5")
documents = SimpleDirectoryReader("/content/drive/MyDrive/data").
    load_data()
vector_index = VectorStoreIndex.from_documents(documents,
    service_context=service_context)
query_engine = vector_index.as_query_engine(response_mode="
    compact")
```

4.2.5. Pruebas

Para realizar las pruebas se crea código tanto para una pregunta simple como para un pequeño chatbot funcional en el que se intercambian preguntas y respuestas con un código para la salida del programa.

Listing 4.7: Pregunta Simple

```
response = query_engine.query("Question") # give your question
    here
display_response(response)
```

Listing 4.8: Chatbot

```
query_engine = vector_index.as_query_engine(response_mode="
    compact")
response = query_engine.query("Fixed first question") # give your
     question here
```

```
display_response(response)
print("If you want to exit the chat type 'exit', 'quit', 'q' or ,
    'f'")
while True:
  query2=input(f"Prompt:")
  if query2 == "exit":
    print('Exiting')
    sys.exit()
  if query2 == '':
    continue
  query_engine2 = vector_index.as_query_engine(response_mode="
     compact")
  response2 = query_engine2.query(query2) # give your question
     here
  display_response(response2)
```

Capítulo 5

Pruebas

Durante el desarrollo del proyecto se han realizado diferentes pruebas para comprobar tanto el funcionamiento de los grandes modelos de lenguaje como de las técnicas RAG aplicadas en ellos. Se han creado diferentes bloques de prueba.

- La primera sección de pruebas se realizó las primeras semanas y estaba referida a la carga de modelos e intento de utilización de los mismos. Al no ser específicamente el objetivo del trabajo, es suficiente mencionarla y no profundizar en ella. No obstante, existe una sección en este capítulo que explica con más detalle esta prueba.

- La siguiente sección de pruebas esta referida a la influencia de los parámetros en la carga del modelo. De nuevo, al estar relacionada con RAG, pero no ser determinante, no se hará especial hincapié.

- La última sección de pruebas es la referida a la aplicación directa de técnicas RAG en los modelos cargados y configurados correctamente. Las pruebas siguen el siguiente orden:

    - Se crea un banco de preguntas relacionadas con la entrevista de trabajo. Utilizar estas preguntas consigue abarcar desde el inicio de la entrevista hasta el final con la contratación, pasando por preguntas sobre la compañía y definiendo cualidades del entrevistado, consiguiendo así una descripción de las habilidades y preguntas directas al entrevistador. Son las siguientes;
      - Hello, I want you to act as an interviewer, first question?
      - How does your company work?
      - I have been studying for 5 years and have 10 years of experience.
      - My strong points are effectiveness and teamwork.
      - I want to learn from the comany and help them.
      - What do I need to get hired?

      Se realizan en inglés por ayudar al modelo a una compresión más directa.
    - Se fija un tamaño de información (en bytes) que se pasará al modelo como datos que, con la ayuda de las técnicas RAG, se procesan y

generan un contexto adecuado como se ha explicado en el capítulo 2. Se emplean las cantidades 0 bytes, 100 KB y 5 MB para ver si existe un cambio significativo en el uso de la cantidad de información.

- Se prueban en cada uno de los modelos todas las combinaciones de preguntas y tamaños de contexto y se procesan las respuestas con la ayuda de *string evaluator* de Langchain. Este evaluador asigna una puntuación a la respuesta. Finalmente, se comparan los resultados.

5.1. Pruebas y evaluación de modelos

Las pruebas referentes a los modelos han consistido en la carga de los mismos y la selección de un conjunto representativo de preguntas para analizar su capacidad. Sin embargo, se han encontrado desafíos al comparar los Modelos de Lenguaje Grande (LLM).

Uno de los principales problemas es la subjetividad en el análisis de las respuestas. El texto de referencia utilizado para comparar las respuestas de los modelos no presenta diferencias suficientemente significativas con respecto a otras referencias existentes. Dado que la evaluación se basa en una *string distance*, una respuesta podría ser correcta, pero al no coincidir exactamente con las palabras de la referencia, podría aparecer como errónea.

Considerando que el objetivo principal del proyecto no es comparar los modelos en sí, sino aplicar técnicas de Recuperación y Generación (RAG), se ha decidido utilizar la página de rankings de Hugging Face. Esta página asigna una puntuación a cada modelo, permitiendo así una comparación objetiva de los modelos utilizados en las pruebas [26].

El leaderboard mide los modelos en diferentes aspectos con ayuda de otros modelos y les asigna una puntuación final. HellaSwag asigna una puntuación a la compresión de texto, ARC se utiliza para evaluar las capacidades de razonamiento y lógica y MMLU evalúa exhaustivamente como un lenguaje maneja variedad de tareas complejas y diversificadas. Además, existen otras como TruthfulQA, Winograde o GSM8K que son menos interesantes para el contexto que se plantea en el proyecto:

- Mistral-7B: Una puntuación media de 65.71. ARC: 63.14, HellaSwag: 84.88, MMLU: 60.78.
- Gemma11-7B: Su puntuación media es de 60.09. ARC: 60.07, HellaSwag: 76.14, MMLU: 60.92.
- Llama2-7B: Obtiene 50.97 como puntuación media. ARC: 53.07, HellaSwag: 78.59, MMLU: 46.87.

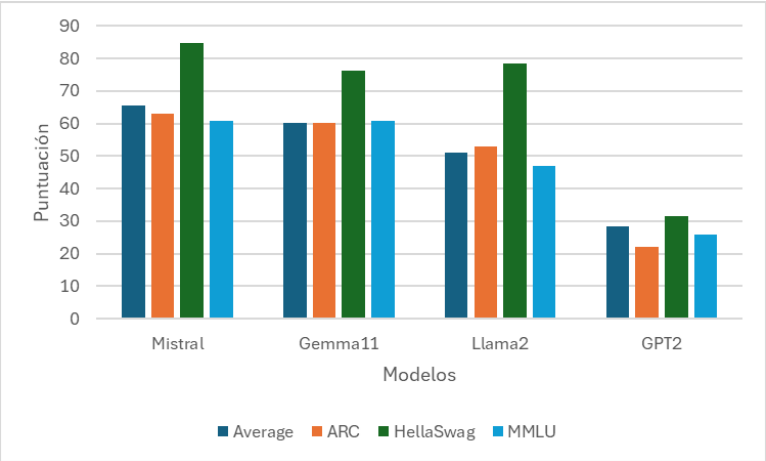

Figura 5.1: Comparación modelos en diferentes aspectos

- GPT2: Puntuación media 28,53. ARC: 22.01, HellaSwag: 31.53, MMLU: 25.83.

Como podemos observar en la figura 5.1, Mistral-7B se destaca como el modelo con mejor rendimiento general, destacando especialmente en la comprensión de texto, lo que significa que tiene una capacidad sólida para genera texto coherente y relevante. Este aspecto es vital en nuestro proyecto, ya que la gran dificultad reside en la generación de texto relevante. Gemma11-7B obtiene una puntuación cercana a la primera, haciendo que su rendimiento sea similar a Mistral, no obstante, presenta una mejoría en la capacidad para manejar tareas complejas y diversificadas, pero una menor en la generación de texto coherente. Llama2, a pesar de tener una puntuación inferior media, obtiene una alta en HellaSwag, lo que hace que sea importante en el desarrollo de las pruebas RAG. Finalmente, GPT2 tiene un rendimiento muy bajo en todas las métricas, lo que indica que está limitado en diferentes aspectos.

Estos resultados no son solo interesantes para analizar los modelos de una manera más objetiva, sino que también nos ayudarán a contextualizar las pruebas finales realizadas en el uso de las técnicas RAG para crear un entrevistador virtual.

5.1.1. Parámetros utilizados

Dado que las pruebas respectivas a la influencia de los parámetros no constituyen el enfoque principal del proyecto, se ha optado por no profundizar en ellas de manera extensa. A pesar de que la selección de los mismos afecta en la configuración de los modelos, no existe un impacto significativo en la aplicación de las técnicas RAG.

Los parámetros a tener en cuenta se dividen en tres:

- Cuantización. Como se ha explicado en otros apartados del proyecto, la selección de los parámetros de la cuantización no está basada en la optimi-

zación de las respuestas si no en la superación de las limitaciones físicas. Por eso, se han fijado los más útiles.

- Nuevos tokens y ventana máxima de contexto. En la carga de modelos con la librería HuggingFace se pueden especificar el nuevo número de tokens en la respuesta y la cantidad máxima de tokens que el modelo puede procesar cada vez con la ventana. Los únicos cambios significativos en las respuestas aparecen cuando se fijan valores muy bajos(1-10 y 1-100 respectivamente), ya que impiden al modelo trabajar correctamente. En cuanto a la mejoría de respuestas con RAG, es muy pequeña y se fijan los que mejores resultados han dado experimentalmente(256 y 4900).

- Generate kwargs. Con los ajustes en los parámetros del kwargs conseguimos cambiar la aleatoriedad y los tokens a tener en cuenta. Existe un cambio cuando se toman valores erróneos no acordes al proyecto, como una alta aleatoriedad. Se fijan 0.3 como aleatoriedad(baja), se consideran solo los 50 tokens más probables(más compacto) y tokens con probabilidad acumulada menor a 0.95(más coherencia en las respuestas). El cambio en estos valores en un 15 por ciento no suponen cambios significativos en el uso de RAG, por lo que no se tratan más a fondo.

5.2. Pruebas y evaluación RAG

Las pruebas se han realizado conforme a lo comentado en la introducción del capítulo, comparando las respuestas dadas por el modelo frente a una referencia, y asignando una puntuación posterior. Para realizar la comparación se ha hecho uso de la distancia de string de langchain (comentado en el apartado 3.1.9).

La principal dificultad ha sido escoger una referencia adecuada para la comparación. Ya que estamos haciendo uso de grandes modelos de lenguaje, se ha llevado a cabo una simulación de entrevista con el banco de preguntas en uno de ellos. Como se comentó en el capítulo 2, GPT4 está disponible y es uno de los modelos con la mayor cantidad de parámetros, por tanto, se ha escogido el desarrollo de la entrevista en él. Para cada pregunta se ha recogido la respuesta del mismo y se ha comparado con las respuestas de los modelos utilizados.

Cabe mencionar que GPT-2 se ha descartado del análisis por las siguientes razones:

- Como modelo es muy inferior a los otros utilizados, como se ha demostrado con las puntuaciones del ranking de Hugging Face.

- La incorporación de las técnicas RAG en el modelo se dificulta, no consiguiendo utilizar la información almacenada para el uso de técnicas RAG en el modelo.

- Las puntuaciones que obtiene son de alrededor a 0.5 al utilizar *string distance*, siendo estas muy bajas en comparación con los otros modelos, descartándose así para una mejor comparación entre los modelos restantes.

5.2.1. Comparación sin información de referencia

Las primeras pruebas realizadas se basan en la utilización de técnicas RAG sin información de referencia en los modelos, esto significa que los documentos que se utilizan para la generación de un nuevo contexto están vacíos, por lo que el modelo se basa en su propia información. Los resultados han sido los siguientes:

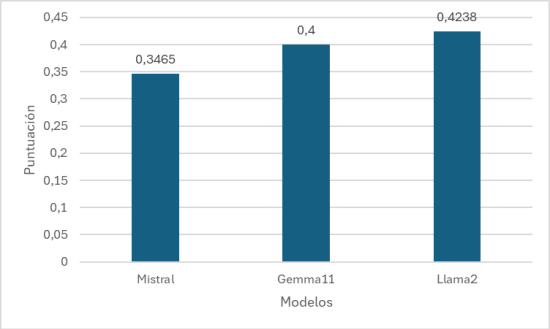

Figura 5.2: Comparación modelos con 0 Bytes de información para RAG

Es necesario recalcar que a menor puntuación obtenida, mejor resultado, y que una puntuación entre 0.4 y 0.5 es muy negativa. Teniendo en cuenta esto observamos en la figura 5.2 que Mistral obtiene la mejor puntuación, seguido de Gemma1.1 y Llama2 obteniendo la peor puntuación, todos ellos en el modelo 7B.

Estos resultados indican que, sin un contexto adicional, Mistral y Gemma1.1 tienen una capacidad superior para generar respuestas que se alinean estrechamente con la referencia, lo que es una ventaja en escenarios donde la información previa no está disponible. Llama2; sin embargo, es menos efectivo sin información lo que sugiere de un contexto mayor para obtener mejores resultados.

La respuesta con la mejor puntuación la encontramos en el modelo Mistral a la respuesta a 'I want to learn from the company end help them', con 0.2004 con la salida:

- 'To learn from a company or contribute to it, you may want to look for opportunities such as training programs, workshops, or projects that align with your skills and interests. You could also offer your expertise to help with specific tasks or initiatives.'.

Las peores respuestas son aquellas en las que el modelo detecta que no hay contexto pero no añade información adicional, obteniendo:

- 'I cannot directly answer your query as it does not relate to the information given'

En la conclusión se detallará más este problema recurrente.

5.2.2. Evaluación con 100 KB de información

La siguiente prueba consistió en el uso de 100 KB de información correspondiente a un manual de entrevistas y una entrevista obtenidos en internet [27], y se obtuvieron los siguientes resultados:

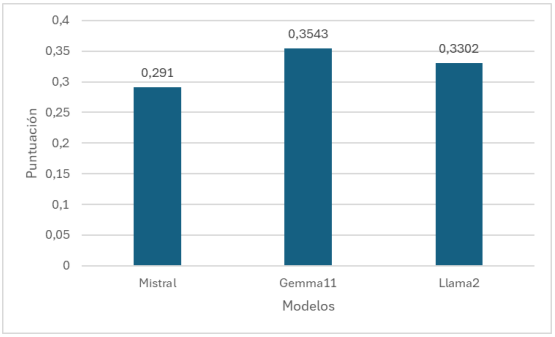

Figura 5.3: Comparación modelos con 100 KB de información para RAG

Como observamos en la figura 5.3, Mistral sigue siendo el modelo con mejor resultado frente a los otros. Le sigue Llama2 con una puntuación de 0.33, superando a Gemma1.1 que obtiene 0.3543. Podemos afirmar que Llama2 necesita poca información para generar el nuevo contexto y mejorar su puntuación, mientras que Gemma1.1 no aprovecha la información adicional tan eficazmente como en los otros modelos. No obstante, existe una mejoría notable en ellos.

Las peores respuestas son de nuevo aquellas que no encuentran un contexto y no son capaces de, a pesar de ello, crear información relevante.

5.2.3. Evaluación con 5 MB de información

La última prueba consiste en utilizar 5 MB de información referente a entrevistas de trabajo, tanto buenas como malas entrevistas, y dos manuales de entrevistas, obtenido de internet [27][28][29]. Se incluye también la información utilizada en el apartado anterior para ver la mejoría con nueva. Se han obtenido las siguientes puntuaciones al utilizar la distancia de string:

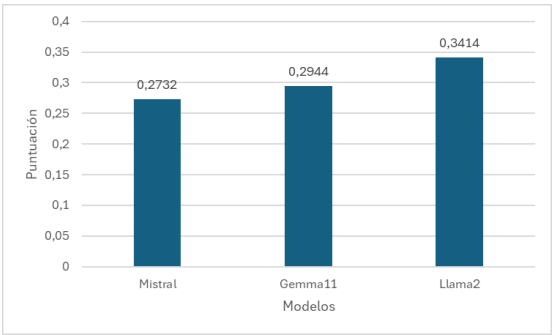

Figura 5.4: Comparación modelos con 5 MB de información para RAG

Como observamos en la figura 5.4, Mistral obtiene de nuevo la menor puntuación con 0.2732, consiguiendo generar respuestas que se aproximan más a la referencia en comparación con Gemma1.1, que le sigue con 0.2944, mejorando significativamente frente a su prueba anterior y obteniendo una puntuación positiva. Finalmente, Llama2 obtiene una puntuación negativa y similar a la anterior, desaprovechando la nueva información suministrada a los modelos. Los resultados siguen distando de una puntuación muy positiva para los modelos. No obstante, utilizar Mistral y Gemma1.1 para desempeñar entrevistas virtuales con técnicas RAG puede ser útil.

Al ser esta prueba la que mejores resultados ha obtenido, se muestra la puntuación que ha obtenido cada modelo en cada una de las preguntas del banco de preguntas.

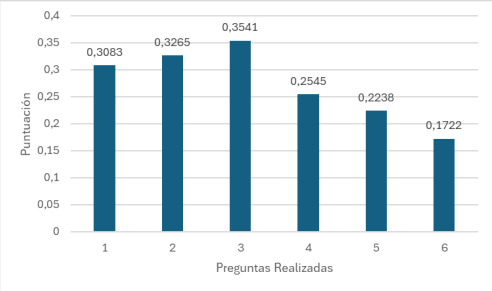

Figura 5.5: Puntuación Mistral-7B en RAG con 5MB

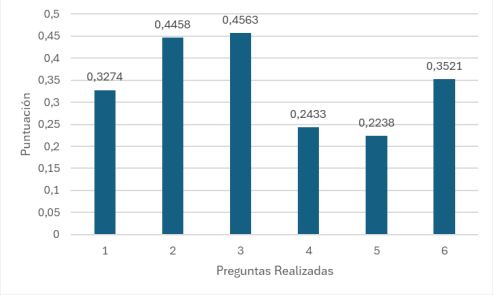

Figura 5.6: Puntuación Llama2-7B en RAG con 5MB

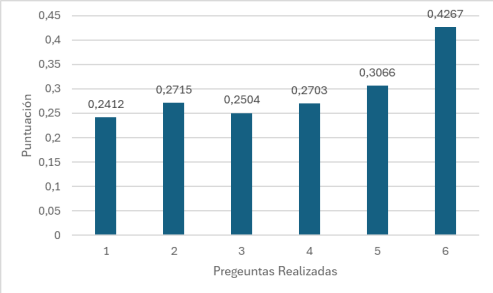

Figura 5.7: Puntuación Gemma1.1-7B en RAG con 5MB

La figura 5.5 y 5.6 nos muestran el desempeño de los modelos Mistral y LLama2 frente a las preguntas realizadas y vemos que encuentran sus mayores dificultades en la pregunta 2 y 3, confundiendo la experiencia del usuario y la función de la empresa, haciendo así que sus salidas no contengan respuestas adecuadas y obtengan puntuaciones alrededor de 0.45 puntos en Llama2 y alrededor de 0.34 en Mistral. Vemos en la figura 5.7 como la mayor dificultad que ha encontrado Gemma1.1 ha sido a la hora de interpretar el estar contratado o no, lo que supone que no ha sabido interpretar el contexto adecuadamente, creando de nuevo respuestas similares a:

- 'It is impossible to provide an accurate answer to your query.'

Donde aparece el mismo problema que en las otras pruebas.

La mejor respuesta a todas ha sido en la última pregunta en Mistral, obteniendo 0.1722 con el siguiente texto:

- 'To get a role in the company, you should first understand the job and its key objectives. Make sure you fully understand the duties, skills, experience, and other requirements of the position.'

Mostrando que el modelo ha sido capaz de comprender perfectamente la pregunta y creando una respuesta acorde al contexto.

5.2.4. Evolución de los modelos con RAG

Finalmente, observamos la evolución que han tenido los modelos respecto a las pruebas realizadas.

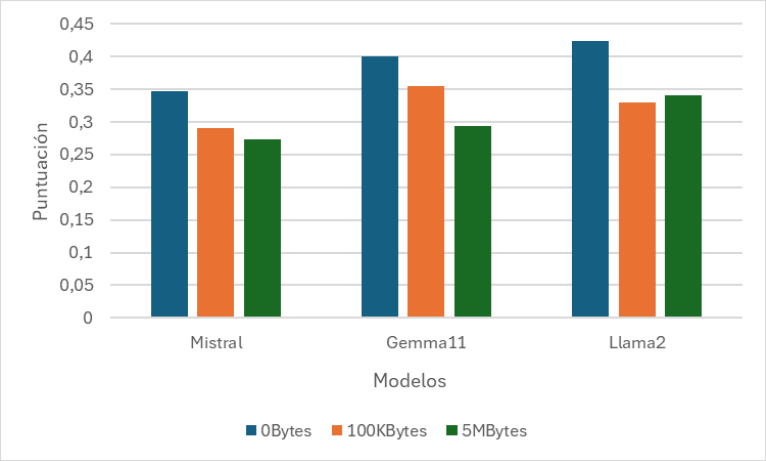

Figura 5.8: Evolución modelos según información en RAG

La figura 5.8 muestra los resultados en las puntuaciones en los diferentes escenarios. Mistral se destaca como el modelo que ha obtenido mejores puntuaciones y ha conseguido una evolución lineal según la información utilizada, explicando así que el modelo es capaz de manejar la información y generar un contexto adecuado para las consultas. Gemma1.1 obtiene también buenos resultados y con evolución a pesar de tener una puntuación más desfavorable, lo que indica que el modelo es capaz de interpretar la información para generar nuevos contextos y respuestas mejoradas.

De nuevo, podemos observar como el modelo Llama2 ha sufrido una desmejora en su versión de 5MB en comparación con la misma de 100KB de uso de almacenamiento en RAG, explicando así que la cantidad de información utilizada en el modelo Llama2 no es un factor determinante para que se procesen respuestas adecuadas al contexto, refiriendo las respuestas ya mencionadas en las que el modelo no es capaz de encontrar información ni tampoco de generar nueva propio

a pesar de ello.

Finalmente, cabe recalcar cómo las puntuaciones que han recibido los modelos se han asemejado a las de Hugging Face Leaderboard, mostradas en el apartado 5.1. Observamos cómo Mistral obtuvo la mejor puntuación, seguido de Gemma1.1 y con Llama2 como último modelo, tal y como se detalla en el ranking.

A pesar de que el impacto de las RAG ha sido significativo en los modelos, tampoco se han obtenido resultado idílicos, ya que muchas de las respuestas han obtenido puntuaciones lejos de 0.15, que se considera óptima.

Capítulo 6

Conclusiones

En el desarrollo de este trabajo de fin de grado se ha realizado un análisis exhaustivo y una evaluación objetiva de las técnicas de Recuperación y Generación Aumentada (RAG) aplicadas a grandes modelos de lenguaje en el contexto de las entrevistas laborales. La implementación de las técnicas tiene como fin mejorar la capacidad de los modelos para actuar como entrevistadores virtuales efectivos.

Entre los principales hallazgos se encuentran el rendimiento de los modelos en el contexto de la entevista, la influencia de los parámetros, la efectividad de las técnicas RAG y la importancia de las limitaciones.

En cuanto al rendimiento de los modelos con la utilización de técnicas RAG para actuar como entrevistador virtual, Mistral-7B se ha destacado como el mejor modelo, mostrando una capacidad sólida para generar texto coherente y relevante para las consultas, encontrando una facilidad mayor en escenarios sin información relacionada con entrevistas de trabajo adicional (0 bytes) y con información considerable (5MB). Gemma1.1-7B también consiguió un rendimiento notable, aunque inferior a Mistral. Llama2-7B; sin embargo, presentó una mejora significativa al incluir información adicional (100KB), pero obtuvo una puntuación menor que los anteriores modelos. Finalmente, GPT2 quedó descartado debido a su bajo rendimiento e incapacidad para integrar eficazmente técnicas RAG.

La influencia de parámetros como ajustes en la ventana máxima de contexto y parámetros de generación tienen un impacto en el rendimiento de los modelos, aunque no determinante en el uso de técnicas RAG.

Implementar RAG en los modelos ha mejorado la relevancia y coherencia de las respuestas generadas por el entrevistador virtual. No obstante, todavía existen limitaciones en la capacidad de los modelos para utilizar la información de forma óptima. Mediante el uso de la distancia de string, se ha descubierto que las mejores respuestas se generan cuando el modelo tiene acceso a una cantidad notable de información. También se ha descubierto el problema cuando no existe información

de referencia, en el que se crean respuestas erróneas indicando que no existe información de referencia. Podemos afirmar que, a pesar de que utilizar estas técnicas obtiene buenos resultados, no es una técnica de aprendizaje idónea para el caso de un entrevistador, ya que crea respuestas sobre la documentación pero no utiliza la documentación para aprender a crear nuevas respuestas sin basarse en ella.

Con el uso e implementación de todas las tecnologías, han surgido desafíos y limitaciones, destacando la capacidad de almacenamiento y uso en Google Colab como principal limitación, y la subjetividad en el análisis de las respuestas y la dependencia en herramientas de evaluación automática como desafío, además del aprendizaje y constante actualización de las diferentes librerías.

6.1. Líneas de trabajo futuro

Los resultados ofrecen una comprensión de como los modelos utilizan las técnicas RAG, proporcionando una base sólida para futuras investigaciones en el campo de la inteligencia artificial y la generación de texto con nueva información.

Para trabajos futuros, se recomienda:

- Explorar nuevos modelos que no estén tan limitados y sean más recientes, dado el rápido avance en los LLM.
- Con las próximas evoluciones hardware, crear un modelo específico para cada puesto de trabajo.
- Tratar de optimizar y tratar en profundidad los parámetros de los modelos y técnicas RAG para mejorar las respuestas generadas.
- Realizar un *finetuning* del modelo con el uso de *reinforcement learning*.

En conclusión, la integración de técnicas RAG en grandes modelos de lenguaje puede ofrecer soluciones efectivas para aplicaciones complejas como las entrevistas de trabajo, a pesar de las limitaciones y desafíos que la implementación y uso pueden suponer.

Bibliografía

[1] Murray Shanahan. Talking about large language models. *Communications of the ACM*, 67(2):68–79, 2024. doi: 10.1145/3624724.

[2] Daniel Jurafsky and James H. Martin. Speech and language processing. URL `https://web.stanford.edu/~jurafsky/slp3/`.

[3] Daniel Jurafsky and James H. Martin. Transformers and large language models, 2023. URL `https://web.stanford.edu/~jurafsky/slp3/10.pdf`.

[4] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya. Sutskever. Article: Language models are unsupervised multitask learners. huggingface. 2019.

[5] Yennie Jun. Exploring creativity in large language models: From gpt-2 to gpt-4, 2023. URL `https://towardsdatascience.com/exploring-creativity-in-large-language-models-from-gpt-2-to-gpt-4-1c2d1779be`

[6] Hugo Touvron, Louis Martin, and et al. Llama 2: Open foundation and fine-tuned chat models.arxiv preprint arxiv:2307.09288 (2023).

[7] Waleed Kadous. Llama 2 is about as factually accurate as gpt-4 for summaries and is 30x cheaper, 2023. URL `https://www.anyscale.com/blog/llama-2-is-about-as-factually-accurate-as-gpt-4-for-summaries-and-is-30x-che`

[8] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. arxiv: abs/2310.06825 (2023).

[9] Gemma Team, Thomas Mesnard, and et al. Gemma: Open models based on gemini research and technology. arxiv abs/2403.08295 (2024), 2024.

[10] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In H. Larochelle, M. Ranzato,

R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

[11] RAG - retrieval augmented generation: La llave que abre la puerta de la precisión a los modelos del lenguaje, 2024. URL https://datos.gob.es/es/blog/rag-retrieval-augmented-generation-la-llave-que-abre-la-puerta-de-la-precisi

[12] Leonie Monigatti. Retrieval-augmented generation (RAG): From theory to langchain implementation, 2023. URL https://towardsdatascience.com/retrieval-augmented-generation-rag-from-theory-to-langchain-implementation-4

[13] Python 3.10.12, 2023. URL https://docs.python.org/release/3.10.12/.

[14] B. M. Randles, I. V. Pasquetto, M. S. Golshan, and C. L. Borgman. Using the jupyter notebook as a tool for open science: An empirical study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 1–2, Toronto, ON, Canada, 2017. doi: 10.1109/JCDL.2017.7991618.

[15] Sandra Navarro. ¿qué es hugging face?, 2024. URL https://keepcoding.io/blog/que-es-hugging-face/.

[16] Hugging face, n.d. URL https://huggingface.co/.

[17] Transformers, n.d. URL https://huggingface.co/docs/transformers/index.

[18] Pytorch, n.d. URL https://pytorch.org/docs/stable/index.html.

[19] Llamaindex, n.d.. URL https://www.llamaindex.ai/.

[20] Llamaindex 0.10.0 release, n.d.. URL https://github.com/run-llama/llama_index/releases?page=4.

[21] LangChain. String evaluator, n.d. URL https://python.langchain.com/v0.1/docs/guides/productionization/evaluation/string/.

[22] Ken Schwaber and Jeff Sutherland. The scrum guide. Journal:. *Scrum Alliance*, 21(1):1–38, 2011.

[23] Gitlab, n.d. URL https://about.gitlab.com/.

[24] Kartheek Yakkala. Retrieval-augmented generation(rag) using open source llms., 2024. URL https://medium.com/@yakkalakartheek/retrieval-augmented-generation-rag-using-open-source-llms-805184b0fe58.

[25] Huggingface LLM - stablelm, n.d. URL https://docs.llamaindex.ai/en/stable/examples/customization/llms/SimpleIndexDemo-Huggingface_stablelm/.

[26] Open LLM leaderboard, n.d. URL https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.

[27] Preparing for an interview, 2021. URL https://www.goucher.edu/career-education-office/documents/Preparing-for-an-Interview.pdf.

[28] Rodney Apple and APICS. How to master the art of job interviewing, 2014. URL https://www.apics.org/docs/default-source/careers-white-papers-webianrs/14-2170_how_to_master_the_art_of_job_interviewing.pdf.

[29] Princeton University. Interview guide, n.d. URL https://careerdevelopment.princeton.edu/sites/g/files/toruqf1041/files/media/interview_guide_5.pdf.