UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADUADO EN INGENIERÍA INFORMÁTICA

Especialización de un modelo LLM en una disciplina específica
Specializing an LLM model in a specific field

Realizado por
JAVIER CARMONA GÁLVEZ

Tutorizado por
FRANCISCO EMILIO LÓPEZ VALVERDE
SERGIO GÁLVEZ ROJAS

Departamento
LENGUAJES Y CIENCIAS DE LA COMUNICACIÓN
UNIVERSIDAD DE MÁLAGA

MÁLAGA, (mes) de 2025

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA
CON MENCIÓN EN SISTEMAS DE INFORMACIÓN

Especialización de un modelo LLM en una disciplina específica

Specializing an LLM model in a specific field

Realizado por
Javier Carmona Gálvez

Tutorizado por
Francisco Emilio López Valverde
Y
Sergio Gálvez Rojas

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2025

Fecha defensa: XX de julio de 2025

Resumen

El presente proyecto aborda la especialización de un agente conversacional basado en modelos grandes del lenguaje (LLM) en disciplina específica, con el objetivo de mejorar su precisión y relevancia en contextos especializados. A pesar de la creciente utilización cotidiana de los LLM en diversas aplicaciones, estos modelos enfrentan desafíos significativos cuando se aplican a disciplinas técnicas debido a su entrenamiento generalista.

Para enfrentar esta limitación, este trabajo realiza una investigación exhaustiva del estado del arte en técnicas de reentrenamiento de LLM, incluyendo métodos como el *fine-tuning* completo, parcial, métodos PEFT, destacando especialmente el método LoRA. Asimismo, se exploran estrategias complementarias como la generación aumentada por recuperación (RAG).

Se desarrolla partiendo de Falcon3 7B Instruct, seleccionado por su óptimo balance entre rendimiento y eficiencia. El conjunto de datos, ha sido generado sintéticamente usando documentos técnicos relacionados con las comunicaciones por satélite, obtenidos de diversas fuentes especializadas, y procesado en un conjunto de datos de 14787 tuplas pregunta y respuesta, usando un modelo Mistral 24B.

Finalmente, se realiza una evaluación rigurosa del modelo resultante utilizando la metodología *LLM as a judge* para comparar diferentes estrategias de reentrenamiento por medio de diferentes métricas diseñadas, ofreciendo así conclusiones claras sobre la eficacia y eficiencia de cada método, además, esta metodología ha sido duplicada, utilizando 2 modelos diferentes por cada evaluación, descartando sesgos y preferencias. Obteniendo a través de esta metodología resultados coherentes con la literatura investigada, denotando la importancia de la calidad del conjunto de datos y el método de entrenamiento escogido según el caso de uso e información disponible.

Palabras clave: agente conversacional, modelo LLM, re-entreno, *fine-tuning*, comunicaciones satelitales

Abstract

This project addresses the specialization of a conversational agent based on Large Language Models (LLMs) in a specific discipline, aiming to enhance its accuracy and relevance within specialized contexts. Despite the growing daily use of LLMs across various applications, these models face significant challenges when applied to technical fields due to their generalist training.

To overcome this limitation, this work conducts a comprehensive review of state-of-the-art techniques for retraining LLMs, including methods such as complete fine-tuning, partial fine-tuning, and PEFT methods, notably highlighting the LoRA method. Complementary strategies, such as Retrieval-Augmented Generation (RAG), are also explored.

The development begins with Falcon3 7B Instruct, chosen for its optimal balance between performance and efficiency. The dataset, synthetically generated, with the model Mistral 24B, using technical documents related to satellite communications obtained from various specialized sources, comprises a total of 14,787 question-answer pairs.

Finally, a rigorous evaluation of the resulting model is performed using the *LLM as a judge* methodology to compare different retraining strategies through various specifically designed metrics, thereby providing clear conclusions about the effectiveness and efficiency of each method. Additionally, this methodology was duplicated using two different models per evaluation to avoid biases and preferences. The conclusions, consistent with existing literature, emphasize the importance of dataset quality and the training method selected, depending on the specific use case and available information.

Keywords: **conversational agent, large language model, re-training, fine-tuning, satellite communications**

Índice

1

Introducción

1.1 Motivación

El uso de LLMs (por sus siglas en inglés) o grandes modelos del lenguaje, como ChatGPT, Claude, Gemini o DeepSeek, entre otros, se ha convertido en una parte común y cotidiana de nuestro día a día, por medio de chats, asistentes virtuales y herramientas generativas varias. Además, los últimos modelos han conseguido un gran rendimiento, convirtiéndolos en potentes asistentes que millones de personas utilizan a diario tanto en el ámbito laboral como en el personal.

Sin embargo, a pesar de su versatilidad y capacidades, estos modelos presentan limitaciones en los campos técnicos y específicos, pues su entrenamiento generalista no garantiza la precisión necesaria para un amplio espectro de disciplinas especializadas, lo que supone un obstáculo para su adopción en aplicaciones profesionales y técnicas que exigen información precisa y contextualizada.

Vemos a diario cómo crece la oferta con nuevos modelos LLM, por medio de diferentes herramientas y plataformas, sin embargo, no ocurre lo mismo con los agentes especializados preentrenados, estos son la alternativa a los modelos generalistas, que consiguen un mejor rendimiento en tareas específicas y precisas, y sin embargo siguen siendo escasos. Esto obliga a investigadores y programadores a desarrollar alternativas personalizadas, lo que encarece y ralentiza la investigación y el desarrollo de sistemas multiagente, limitando la propagación de la tecnología LLM a campos donde aún no ha sido incorporada y en los que podría marcar una diferencia significativa en la toma de decisiones y aumentando la eficiencia operativa.

1.2 Objetivos

Este proyecto, ha sido definido con una intención investigadora y con el objetivo de generar un agente conversacional basado en un modelo LLM entrenado en disciplina específica. De manera que se pueda usar posteriormente en diferentes contextos, por ejemplo, para favorecer la investigación, para ser usado como un agente dentro de sistemas multiagente o asistente conversacional, entre otros.

Los entregables que componen este proyecto han sido:

1.2.1 Estudio del estado del arte

Estudiar el estado del arte de los métodos de reentrenamiento para LLMs en disciplina específica, ha sido el pilar fundamental de este proyecto, pues trivialmente, es necesario conocer los métodos de reentrenamiento para poder elegir uno, que sea apropiado para la disciplina que se escoja, tanto por términos de eficiencia con respecto al uso que se le quiera dar al modelo, como por eficiencia en términos de tiempo de entreno, como por eficiencia en tiempo de inferencia en el uso del modelo.

1.2.2 Análisis de modelos LLM

Antes de escoger un modelo específico, es necesario hacer una investigación previa en la que se revisan los distintos modelos de código abierto, actuales y relevantes. Esto es de mayor importancia de la que aparenta, pues es un campo activamente en desarrollo y de muy rápido movimiento.

1.2.3 Desarrollo y evaluación del modelo

Con las secciones anteriores finalizadas, se tendrá una vista fundamentada al estado del arte del reentreno de modelos LLM y la lista de modelos LLM actuales que se pueden utilizar. Así pudiendo de manera fundamentada, reentrenar el modelo LLM, en disciplina específica. Y de manera posterior, elaborar una evaluación del modelo obtenido, para que pueda ser comparado con el modelo del que se partió, así cuantificando el cambio obtenido, y extrayendo la conclusión del proceso que se ha acarreado.

1.3 Propuesta

En este proyecto, además de cumplir con los objetivos planteados en la sección anterior y en el anteproyecto correspondiente, se ha realizado una evaluación sistemática sobre los efectos de trece modalidades distintas de reentrenamiento aplicadas a un mismo modelo con un conjunto de datos constante. Este enfoque ha permitido obtener una cantidad significativa de datos numéricos, derivados de implementaciones prácticas de cada método de reentrenamiento. De esta manera, se proporciona una visión panorámica del estado actual del reentrenamiento en disciplina específica, particularmente en el contexto del uso de un conjunto de datos sintético y no revisado, situación realista y frecuente en contextos profesionales y académicos.

1.4 Estructura del documento

El documento está estructurado con la Introducción en el capítulo 1, donde se introduce la motivación para realizar este proyecto, los objetivos que se plantearon al definirlo, la estructura del documento, y por último la metodología que se ha seguido a lo largo del proyecto.

En el capítulo 2, se habla sobre el estudio del estado del arte que se ha llevado a cabo, mencionando la bibliografía revisada, siendo esta la que fundamentará las decisiones tomadas. El capítulo 3, desarrolla el *pipeline* desarrollado de generación sintética del conjunto de datos usado, donde incumbe desde la temática usada para el proyecto, la necesidad de este *pipeline*, y el desarrollo de la generación sintética hasta la obtención del conjunto de datos resultante. El capítulo 4, está dedicado a la fase del desarrollo, donde se desarrolla desde la elección del modelo LLM que se reentrenará, hasta los diferentes tipos de re-entrenos en disciplina específica que se han evaluado. El capítulo 5, se dedica a la evaluación y la ejecución de los tests necesarios, incluye desde la elección de la estrategia, hasta una discusión en profundidad de los datos obtenidos. Por último en el capítulo 6, se concluyen los resultados obtenidos, extrayendo conocimiento del proceso acarreado, y se finaliza con las líneas futuras de investigación posibles para este proyecto.

1.5 Metodología

Como marco metodológico se ha adoptado CRISP-ML(Q), una evolución del modelo CRISP-DM específicamente orientada al ciclo de vida de sistemas de aprendizaje automático e inte-

ligencia artificial. Esta metodología ha sido seleccionada debido a su enfoque estructurado y riguroso, que resulta especialmente adecuado para proyectos complejos basados en modelos de lenguaje de gran tamaño (LLM), donde la trazabilidad, la reproducibilidad y la alineación con los objetivos del dominio son factores críticos.

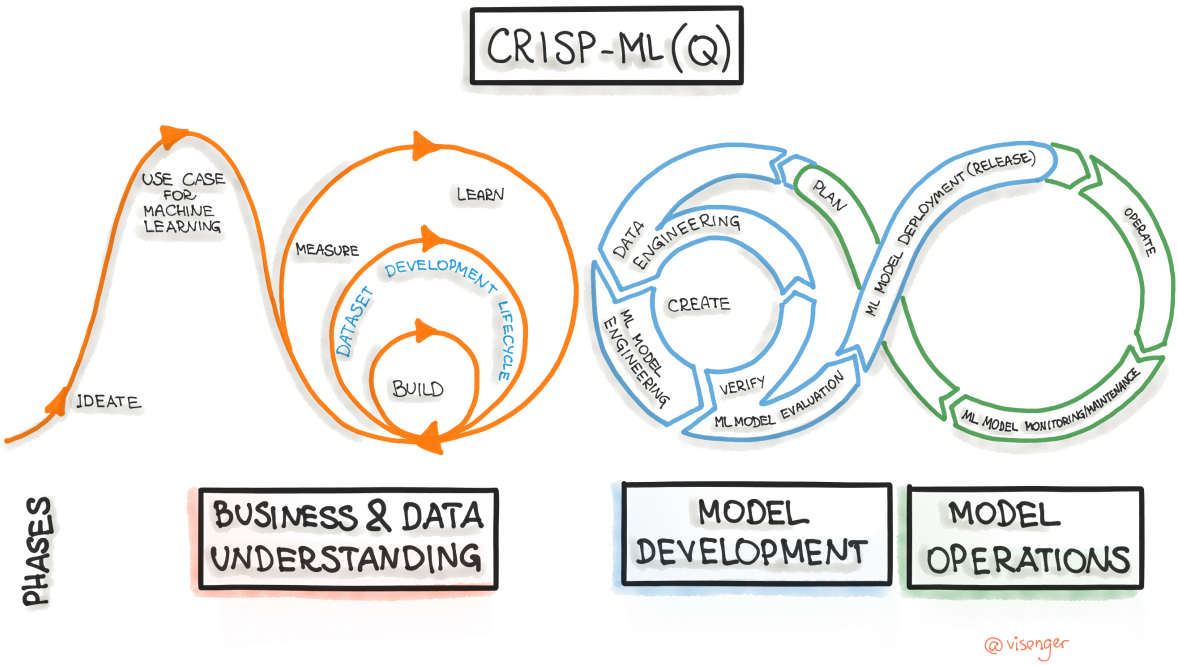

Figura 1: Ciclo de vida del desarrollo del aprendizaje automático basado en CRISP-ML(Q), imagen reproducida de [19].

CRISP-ML(Q) surge como una respuesta a las limitaciones detectadas en CRISP-DM en contextos actuales de IA, e introduce mejoras sustanciales, entre las que se incluyen: la integración de aseguramiento de la calidad en todas las fases del ciclo de vida(véase Figura 2); la adición de una fase explícita de monitorización y mantenimiento post-despliegue; la fusión de las fases de comprensión del negocio y de los datos, reconociendo su interdependencia; la definición formal de criterios de éxito en niveles de negocio, técnico y económico; y la incorporación sistemática de prácticas de gestión de riesgos técnicos y reproducibilidad. Estas ampliaciones refuerzan la robustez del proceso y permiten asegurar la utilidad del sistema desplegado en contextos reales y exigentes.

El proceso metodológico se estructura en seis fases principales, iterativas entre sí, como se puede ver en la Figura 1:

1. **Comprensión del negocio.** Se identifican y formalizan los requisitos funcionales y

no funcionales del sistema, alineando sus objetivos con las necesidades del dominio de aplicación.

2. **Comprensión de los datos.** Mediante análisis exploratorio se evalúan la calidad, completitud, representatividad y adecuación de los datos disponibles, aspectos clave para la fiabilidad del modelo.

3. **Preparación de los datos.** Se ejecutan procesos que pueden incluir limpieza, transformación, segmentación y, si es necesario, anotación manual, con el objetivo de adecuar los datos al modelo y tecnología definidos.

4. **Modelado.** En esta fase se configura el proceso de especialización del modelo, aplicando técnicas como *fine-tuning*, *prompt tuning*, LoRA o BitFit. Se definen los hiperparámetros y se diseña la estrategia experimental.

5. **Evaluación.** La validación del modelo se realiza mediante métricas cuantitativas (p. ej., exactitud, F1, *perplexity*) y cualitativas (p. ej., adecuación semántica al dominio), determinando así la necesidad de iterar fases previas.

6. **Despliegue.** Se analiza la viabilidad de integrar el modelo en un entorno operativo, considerando aspectos como mantenimiento, actualización, reproducibilidad y monitorización post-despliegue. En este proyecto no se ha implementado esta fase, pues no se pretende desplegar públicamente.

La adopción de CRISP-ML(Q) garantiza un enfoque sistemático, reproducible y orientado a la calidad, esenciales en el desarrollo de sistemas basados en LLM en entornos reales. La descripción formal de esta metodología se encuentra en [27, 19], donde también se recogen recomendaciones actualizadas para su implementación práctica.

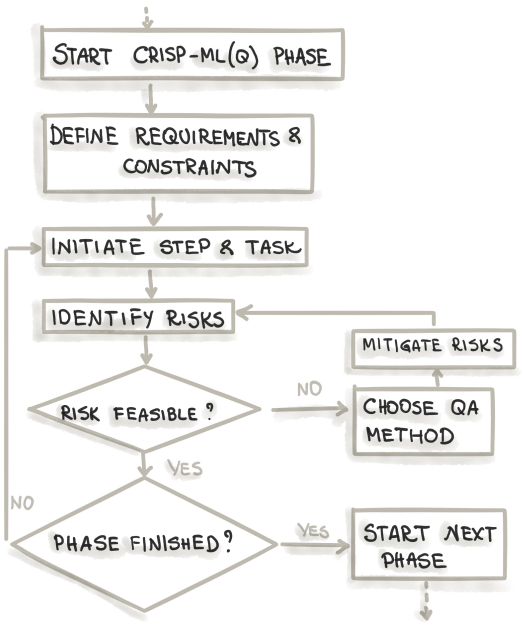

Figura 2: Control de calidad que aplica CRISP-ML(Q), imagen reproducida de [19].

2

Estado del arte

2.1 Introducción general al problema

En el ámbito de la inteligencia artificial, un agente se define como una entidad autónoma que percibe su entorno a través de sensores, procesa la información recibida y realiza acciones en dicho entorno mediante actuadores, con el objetivo de cumplir tareas específicas [26]. Estos agentes destacan por su capacidad para adaptarse dinámicamente al contexto en el que operan, tomando decisiones autónomas basadas en su percepción y conocimientos previos.

Específicamente, los agentes conversacionales constituyen una clase particular de agentes que interactúan con usuarios humanos mediante lenguaje natural. Su función es entender, procesar y generar respuestas coherentes y contextualmente relevantes ante entradas textuales o vocales proporcionadas por los usuarios[11]. El contexto en el que se manejan estos agentes conversacionales, se puede definir como la conversación misma, así este proyecto aborda precisamente la especialización de un agente conversacional basado en Modelos de Lenguaje Grandes (LLMs, por sus siglas en inglés) en una disciplina específica, con la finalidad de mejorar su rendimiento en contextos profesionales especializados.

Los modelos de lenguaje de gran tamaño (LLM, por sus siglas en inglés) han demostrado una capacidad sobresaliente para aprender patrones complejos, extraer representaciones semánticas y capturar relaciones contextuales en el lenguaje natural. Estas arquitecturas no solo generan texto con coherencia y fluidez comparable al generado por un humano, sino que también facilitan traducciones automáticas entre múltiples idiomas, responden a consultas, analizan el sentimiento de los enunciados y cubren un amplio espectro de tareas de procesamiento de lenguaje natural [14].

No obstante, cuando se aplican directamente a dominios verticales, es decir, ámbitos especializados que poseen vocabulario, convenciones y conocimientos muy específicos, su ren-

dimiento suele ser subóptimo. Esta deficiencia obliga a realizar un afinamiento o re-entreno utilizando datos y criterios propios de cada campo de aplicación para alcanzar niveles de precisión aceptables [17]. Además, estudios recientes han señalado que la falta de corpus especializados, las variaciones semánticas propias del dominio y la complejidad intrínseca de ciertos conocimientos pueden agravar estos problemas, limitando la eficacia de los modelos LLM en tareas muy concretas [15, 31].

En este contexto, la adaptación de LLM a dominios específicos implica más que el simple reentrenamiento con datos especializados; requiere el diseño de estrategias que incorporen nuevo conocimiento sin sacrificar las competencias previamente adquiridas. Así lo señala [16], quien advierte que la escasez de datos de entrenamiento originales y de muestras de alto valor informativo impone un límite superior al grado de mejora y asimilación de conocimientos alcanzable mediante afinamiento.

La selección del método de entrenamiento resulta, por tanto, determinante para el rendimiento final. En [6] se comparan diversas aproximaciones de adaptación de parámetros, incluyendo técnicas de aprendizaje eficiente (PEFT), prompt tuning, ajuste completo de parámetros (full fine-tuning) y adapter tuning, entre otras estrategias (véanse las secciones 2.2.3, 2.3.1 y 2.2.2 para una descripción detallada).

Dada la elevada demanda computacional y temporal asociada al entrenamiento de grandes modelos, los desarrolladores suelen optar por soluciones que equilibren coste y rendimiento. En respuesta a esta necesidad, han surgido métodos orientados a reducir los recursos empleados sin sacrificar significativamente la calidad de los resultados, como LoRA, PEFT y otras variantes de adaptación eficiente de parámetros [10, 1, 6]. Estos enfoques se describirán con mayor detalle en secciones posteriores.

Para ilustrar la magnitud del coste computacional, considérese el caso del modelo Llama 3.1 de 8 mil millones de parámetros. En un escenario de preentrenamiento y afinamiento completo, dicho modelo requiere aproximadamente 1,46 millones de horas de GPU, con un consumo estimado de 700 vatios por unidad de procesamiento [18].

Para concluir se muestra en la Figura 3 de manera esquemática la división que se ha usado para plantear esta sección.

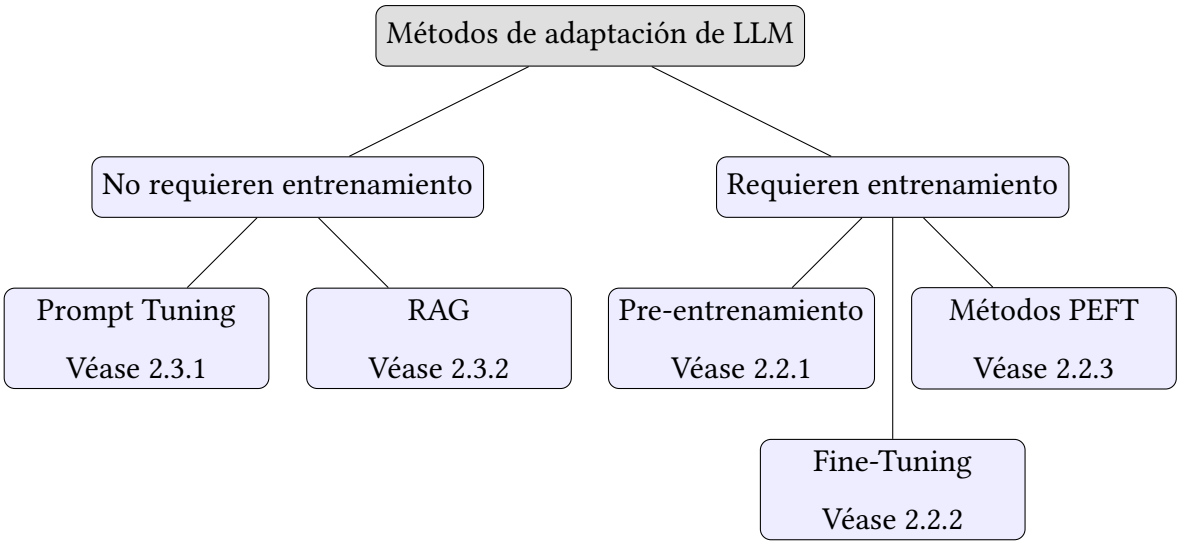

Figura 3: Clasificación de métodos de adaptación según necesidad de entrenamiento.

2.2 Técnicas de afinamiento con entrenamiento para LLMs

La Figura 3 ilustra las distintas estrategias de adaptación de modelos LLM, distinguiendo entre métodos que requieren reentrenamiento y aquellos que no. Dentro de las aproximaciones que implican un proceso de entrenamiento adicional, el entrenamiento completo o (fine-tuning completo) destaca como la alternativa de más eficaz, al ofrecer el máximo potencial de mejora en rendimiento [17, 6].

Estas técnicas consisten en actualizar los parámetros del modelo, desde un subconjunto de capas hasta la totalidad de ellas, con el objetivo de incorporar conocimiento especializado sin comprometer las capacidades adquiridas durante el preentrenamiento [14]. En esta sección se describirán los métodos más relevantes dentro de este grupo: en primer lugar, los métoods tradicionales con preentrenamiento y fine-tuning completo (Secciones 2.2.1 y 2.2.2); a continuación, las técnicas de afinamiento eficiente o métodos PEFT (Sección 2.2.3).

2.2.1 Preentrenamiento

El preentrenamiento constituye la fase inicial y más costosa en la construcción de modelos de lenguaje a gran escala, siendo la etapa que produce el modelo pre-entrenado que se usa como base, y del que se hace inferencia [25]. Durante esta etapa, el modelo se entrena sin supervisión sobre vastos volúmenes de texto genérico, lo que le permite aprender representaciones

profundas del lenguaje, así como extraer patrones semánticos y sintácticos relevantes para posteriores tareas de adaptación [5, 6]. Si bien este proceso requiere un elevado consumo de recursos computacionales y temporales, superior al del afinamiento completo y el mayor de todos los métodos de re-entreno hasta la fecha, sienta la base para obtener mejoras sustanciales en el rendimiento de modelos más grandes, cuyo rendimiento tiende a escalar positivamente con el tamaño y la diversidad del corpus utilizado [6].

En el contexto de los dominios específicos, el preentrenamiento se extiende frecuentemente mediante la técnica de preentrenamiento continuo (CPT, por sus siglas en inglés, Continued Pre-training), en la que el modelo previamente entrenado se reentrena adicionalmente con corpus propios del dominio de aplicación. Este enfoque permite inyectar conocimiento específico y adaptar de forma más efectiva las representaciones internas a vocabularios y convenciones particulares, mejorando así la ejecución de tareas complejas sin sacrificar las capacidades generales adquiridas en la fase inicial [16, 3].

A pesar de la eficacia demostrada, en numerosos casos no se dispone de información completa sobre los datos originales de preentrenamiento, especialmente en modelos open-source como Llama o Mistral, lo que dificulta reproducir con exactitud los procesos de alineación y ajuste empleados en las etapas previas [16]. No obstante, la evidencia empírica sugiere que extender el preentrenamiento con conjuntos de datos especializados de gran tamaño contribuye de manera significativa a la incorporación de nuevos conocimientos y a la mejora de la precisión en tareas concretas del dominio [3]. Concluyendo así, con el mayor problema existente con la estrategia del pre-entrenamiento, la falta de corpus textuales con tamaño y calidad suficiente para imbuir en los modelos el conocimiento necesario.

2.2.2 Fine-tuning

El *fine-tuning* (afinamiento o re-entrenamiento) constituye la técnica principal para especializar un modelo de lenguaje previamente pre-entrenado en tareas y dominios verticales, como el jurídico o el médico, donde su desempeño nativo resulta subóptimo [17]. Partiendo de un LLM pre-entrenado, se ajustan sus parámetros mediante aprendizaje por transferencia; este proceso puede complementarse con estrategias como la expansión de vocabulario y el refinamiento de instrucciones, lo que potencia su eficacia en escenarios especializados [3]. Los principales desafíos incluyen la selección de hiperparámetros adecuados, la prevención del sobreajuste y

la identificación de un modelo base con características afines a la tarea objetivo [14].

Según [16], la combinación de pre-entrenamiento continuado (CPT) y afinamiento supervisado (SFT), resulta especialmente efectiva: primero se expone al modelo a datos de dominio relevantes y, a continuación, se le entrena con conjuntos etiquetados en formato instrucción-respuesta o pregunta-respuesta, optimizando su capacidad para resolver casos de uso concretos.

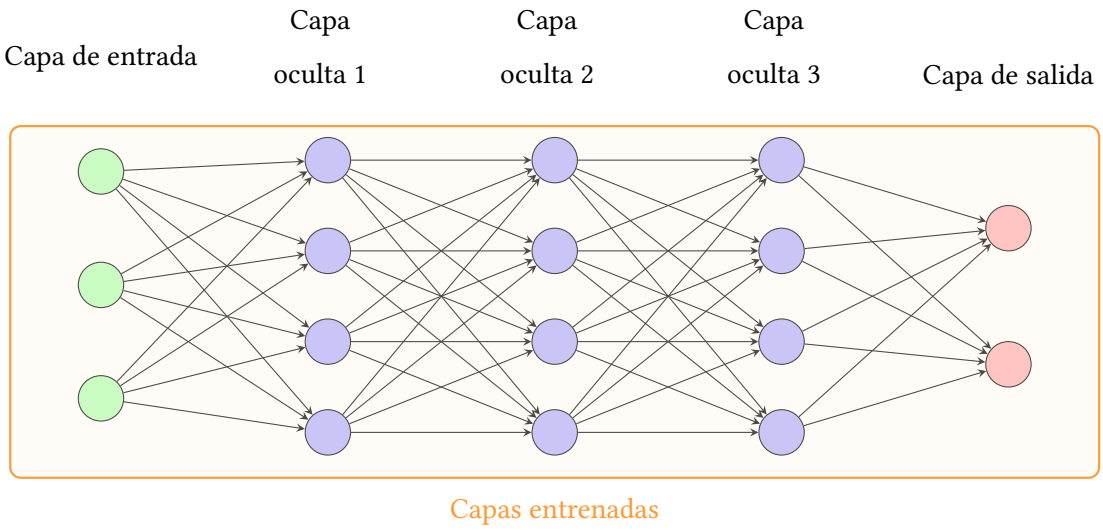

Figura 4: Representación simplificada del entrenamiento completo

El procedimiento de fine-tuning se articula generalmente en dos fases:

1. **Tokenización:** conversión del corpus de entrenamiento a secuencias de identificadores numéricos mediante el tokenizador asociado al LLM.

2. **Entrenamiento iterativo:** ajuste de los pesos de las capas seleccionadas empleando datos de entrenamiento y validación, de acuerdo con la arquitectura de la tarea [23].

Aunque los métodos PEFT (p. ej., prefix-tuning, LoRA y adapters; véase Sección 2.2.3) reducen significativamente el coste computacional, los estudios comparativos indican que el *fine-tuning* completo (ajuste de todos los parámetros) continúa superándolos en rendimiento promedio, a cambio de mayores exigencias computacionales [6].

La categorización de [24], se agrupa *fine-tuning* según sus datos en tres categorías, sin embargo por la estructura de este proyecto, no se ha incluido la tercera categoría de prompt como dato, pues se verá en la sección 2.3.1, así las dos categorías en las que se dividen son:

- **Datos sin estructurar:** empleados en *fine-tuning* no supervisado para adaptar el modelo a dominios amplios.

- **Datos estructurados:** utilizados en *fine-tuning* supervisado para tareas específicas, como clasificación o etiquetado de texto.

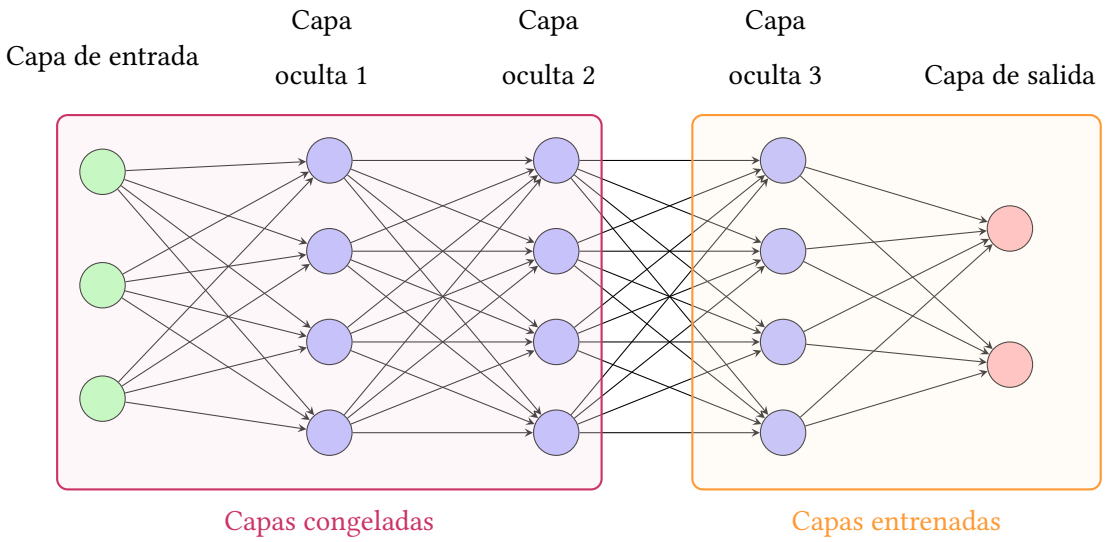

Figura 5: Representación simplificada del entrenamiento parcial

Según el alcance de la modificación de la red, el *fine-tuning* se distingue en:

- **Afinamiento completo:** como se ve en la figura 4, ajusta los pesos de todas las capas, maximizando el potencial de mejora a costa de elevados costes computacionales, mayor riesgo de sobreajuste y de olvidado catastrófico [23].

- **Afinamiento parcial:** como se visualiza en la figura 5, actualiza un subconjunto de capas (iniciales, intermedias o finales), lo que reduce la carga de memoria y de cálculo y mantiene un rendimiento cercano al completo [6].

En síntesis, el *fine-tuning* requiere no solo datos de alta calidad y alineados con el caso de uso, sino también una planificación cuidadosa de las capas a ajustar y una infraestructura computacional adecuada. Cuando los recursos son limitados, los métodos PEFT descritos en la sección siguiente ofrecen una alternativa viable.

2.2.3 Adaptación eficiente de parámetros (PEFT)

Los métodos PEFT *(Parameter Efficient Fine-tuning)* constituyen un conjunto de estrategias orientadas a maximizar el rendimiento de los modelos LLM modificando únicamente un número reducido de parámetros, lo que se traduce en menores requisitos computacionales durante el proceso de afinamiento. Asimismo, estos métodos facilitan la generalización cruzada entre tareas y contribuyen a preservar la privacidad del modelo al exponer mínimamente sus parámetros internos [17]. Estudios recientes indican que cada variante PEFT puede generalizar satisfactoriamente sin necesidad de un sobreajuste exhaustivo del conjunto de entrenamiento, y que la combinación de múltiples métodos amplía la brecha de generalización hasta niveles comparables con el *fine-tuning* completo, pese a ajustar muchos menos parámetros [6]. Además obtienen mejores resultados en escenarios de escasez de información, [24]

Según la localización de los parámetros entrenables, los métodos PEFT se agrupan en dos categorías principales:

- **Parámetros extra:** En este enfoque, como se ve en la figura 6, se congelan los pesos

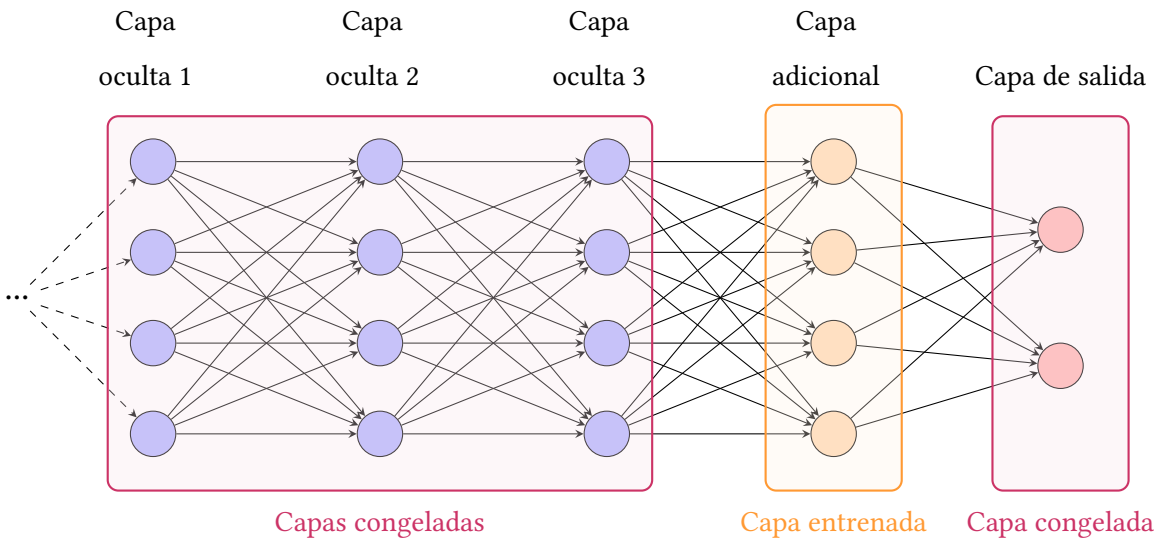

Figura 6: Representación simplificada de los métodos PEFT extra-parámetro

originales del modelo y se insertan parámetros adicionales siendo los únicos ajustados. Un ejemplo paradigmático son los *adapters*, aunque en algunos estudios se incluye también el *prompt tuning* dentro de esta categoría; sin embargo, en este trabajo el ajuste de *prompts* se analiza en la Sección 2.3.1.

- **Intra-parámetros:** Como se observa en la figura 7, se conserva la mayor parte de la

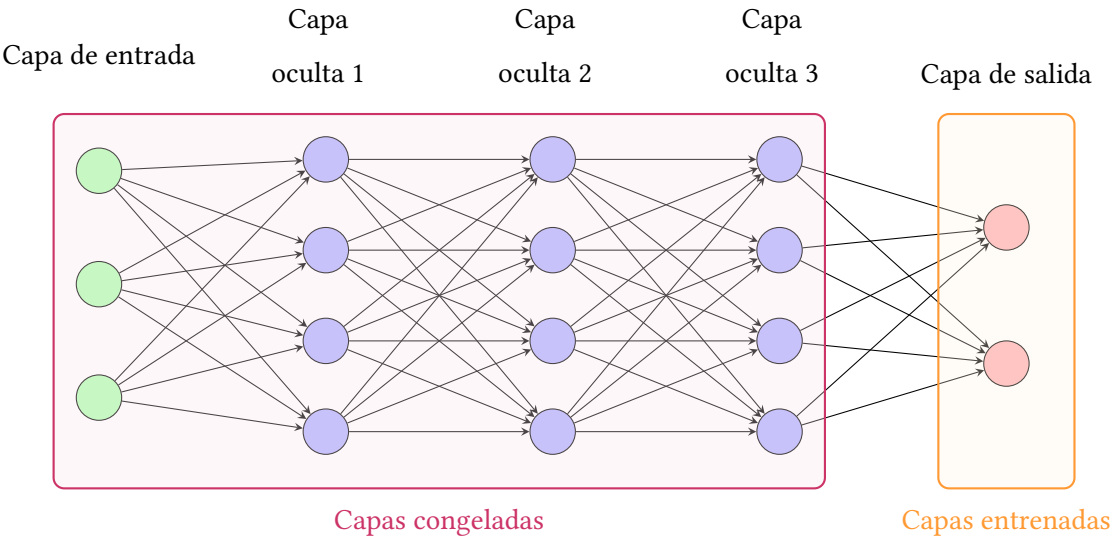

Figura 7: Representación simplificada de los métodos PEFT intra-parámetro

arquitectura intacta y solo se actualiza un subconjunto de sus parámetros. Entre las instancias más relevantes figuran BitFit, LISA y LoRA, siendo éste último el foco de la Sección siguiente 2.2.4.

Aunque ninguna variante PEFT iguala de forma absoluta el rendimiento del *fine-tuning* completo, la brecha de desempeño no resulta insuperable al combinar diferentes métodos PEFT; de hecho, el número bruto de parámetros entrenables no suele determinar la calidad final de la adaptación. Además, estas técnicas han demostrado una excelente capacidad para el aprendizaje multitarea y mantienen un coste de almacenamiento mínimo [6, 24].

En síntesis, los métodos PEFT ofrecen un compromiso óptimo entre coste y calidad de adaptación, resultando especialmente adecuados en entornos con recursos limitados y contribuyendo a la democratización del afinamiento de modelos a gran escala.

En particular, LoRA se erige como una de las metodologías PEFT con mayor desempeño, [17], por lo que se investigará en profundidad en la siguiente sección 2.2.4.

2.2.4 LoRA

La técnica de LoRA *(Low-Rank Adaptation)*, es una estrategia dentro de la categoría de métodos PEFT, en la sección 2.2.3, sin embargo, como será usado en este proyecto, y es de la estrategia

mas destacada en este campo por ser la que mayor rendimiento ofrece [10], se ha decidido darle su propia sección.

Esta técnica, está siendo ampliamente adoptada por su eficacia y eficiencia. Esto queda resaltado cuando es comparado con los métodos tradicionales como se ha indicado en la sección 2.2.2, sobre *fine-tuning*, pues entrena $\frac{1}{1000}$ de los parámetros del *fine-tuning* completo. Además con respecto a este, es 1.9 veces más rápido, no incrementa la latencia de inferencia y puede conseguir un rendimiento comparable o mejor en tareas específicas [17, 6].

Consigue esto, sin tener la necesidad de modificar el modelo, y ajustando solo una pequeña parte de los parámetros. Lo logra añadiendo unas matrices de pesos adicionales y entrenables de bajo rango, cuyos valores se suman a los del modelo, así alterando las capas por el entrenamiento. Como se puede ver en la figura 8, que representa esta ecuación adaptada del artículo [10]:

$$W_{nuevo} = W_{original} + \Delta W_{LoRA}; \text{donde } \Delta W_{LoRA} = \alpha BA, \tag{1}$$

donde [17]:

- $W_{original}$ son las matrices de pesos originales y sin alterar
- W_{nuevo} son las matrices resultantes, que representan al modelo entrenado.
- W_{LoRA} corresponde a las matrices adicionales que este método añade, a lo que se refiere con LoRA plugin
- α representa el factor de escalado, una constante numérica que controla la fuerza de las actualizaciones de valores.
- B y A son 2 pequeñas matrices. Se reparten en estas 2 matrices para optimizar la memoria necesaria para hacer cálculos.

Como se puede observar en la ecuación de arriba, es fácil de integrar sin rediseñar el modelo, lo que lo hace muy versátil. Además, como hemos visto que son matrices adicionales, hace que se puedan incorporar al modelo de manera sencilla y desactivarlas. Estas matrices que se pueden incorporar y retirar del modelo son LoRA plugins, son independientes del LLM y se pueden conectar y desconectar a antojo. Permitiendo esto almacenarlas y distribuirlas independientemente del modelo base, facilitando así su intercambio. Adicionalmente, los LoRA plugins, pueden ser combinados para conseguir una buena generalización entre tareas

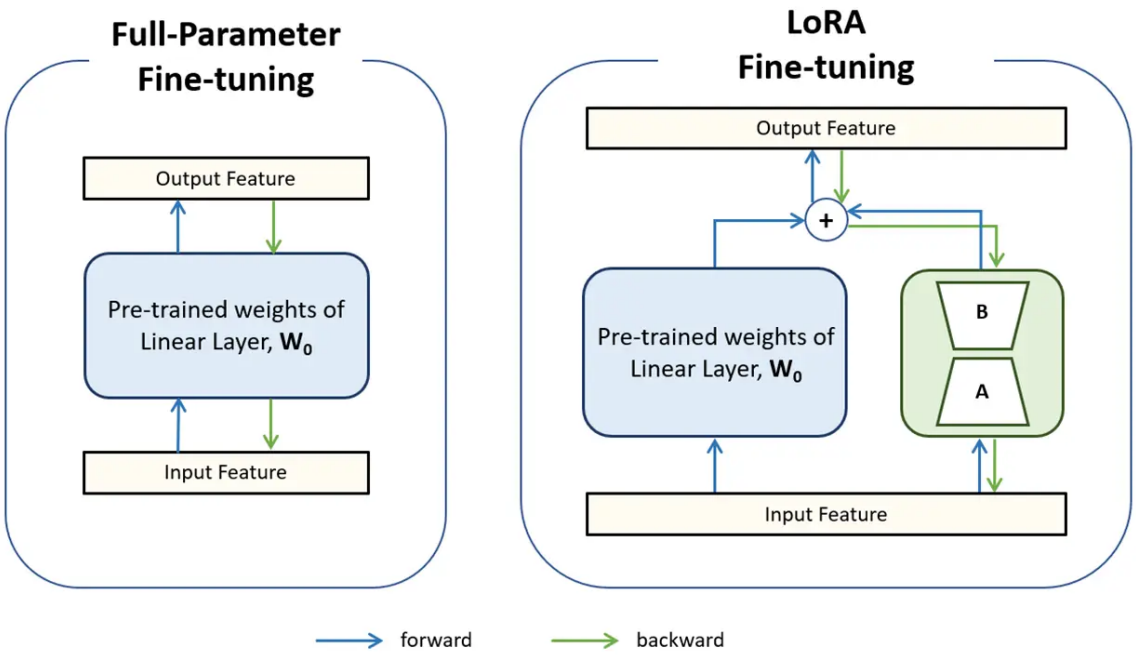

Figura 8: Comparación de afinamiento completo contra LoRA. Fuente: [12]

[10], a pesar de que incremente el coste computacional. También es de destacar, que se puede combinar la estrategia de LoRA, con otros métodos PEFT, o se puede hacer sobre modelos ya afinados, haciendo de LoRA una estrategia increiblemente versátil y eficiente.

LoRA se ha utilizado de manera exitosa en tareas como generación de texto, clasificación, respuesta a preguntas, y revisión de código. Su adopción ha sido especialmente significativa en contextos con restricciones de hardware, como implementaciones locales o entornos corporativos limitados[17].

Así, LoRA defiende de manera indiscutible su lugar en la vanguardia del entrenamiento o afinamiento de modelos LLM, aún no llegando al 100 % de rendimiento potencial que sí obtendríamos con afinamiento completo, en muchos casos puede ser un rendimiento suficiente considerando las ventajas y facilidades que provee.

2.3 Técnicas de afinamiento sin entrenamiento para LLMs

Las técnicas de afinamiento sin entrenamiento permiten adaptar un LLM a nuevas tareas o dominios sin necesidad de reentrenar sus parámetros. En determinados escenarios, estos métodos ofrecen rapidez de implementación, menores costes computacionales y resultados compara-

bles, o incluso superiores, a los obtenidos mediante métodos que requieren entrenamiento del modelo. Por ello, constituyen herramientas ampliamente adoptadas en la práctica de despliegue de LLMs.

Dentro de esta categoría distinguimos dos grupos principales:

2.3.1 Prompt tuning

El *prompt tuning* consiste en diseñar o aprender cadenas de texto (prompts) que, al suministrarse al modelo en fase de inferencia, guían su comportamiento hacia la tarea deseada [25]. Este enfoque aprovecha la capacidad del LLM para interpretar instrucciones y se implementa íntegramente mediante ingeniería de *prompts*, evitando cualquier proceso de reentrenamiento.

Diversas estrategias de *prompting* han sido propuestas en la literatura [25, 6]:

- ***Prompts* basados en instrucciones:** incluye *zero-shot*, *few-shot* y *persona prompting*. En el primero, el modelo recibe únicamente la orden de la tarea, sin ejemplos previos; en el segundo, se le proporcionan unos pocos ejemplos anotados; y en el tercero, se configura un perfil o personalidad que condiciona el estilo de la respuesta [23].
- **Prompts basados en razonamiento:** engloba técnicas como *chain-of-thought*, *tree-of-thought* y *self-consistency*, diseñadas para mejorar el razonamiento lógico y aritmético al inducir al modelo a exponer pasos intermedios antes de emitir la respuesta final [33].
- **Compresión de *Prompts*** propone sintetizar prompts extensos en secuencias más cortas e incluso ininteligibles para el humano, pero que mantienen o mejoran la precisión del modelo mediante un número reducido de tokens [20].

A pesar de su sencillez aparente, el *prompt tuning* presenta varios desafíos [32]:

- **Sensibilidad y robustez:** pequeñas variaciones en el texto del *prompt* pueden inducir cambios sustanciales en el comportamiento del modelo. Asimismo, un mismo sistema de *prompting* puede implicar interdependencias entre módulos que dificultan su diseño óptimo.
- **Tendencia a la alucinación:** la generación de contenido no factualmente respaldado o las llamadas alucinaciones, pueden reducirse mediante retroalimentación humana y refinamiento iterativo de los *prompts*.

Desde la perspectiva de adaptación eficiente de parámetros, el aprendizaje basado en *prompts* ha mostrado un rendimiento prometedor, especialmente en escenarios de pocos datos, si bien converge más lentamente que el *fine-tuning* completo o que otros métodos PEFT [6].

En conjunto, el *prompt tuning* constituye una primera línea de adaptación de LLMs, de rápida implementación y bajo coste computacional, que suele emplearse como paso previo a otras estrategias de afinamiento.

2.3.2 RAG

La generación aumentada por recuperación (RAG, Retrieval-Augmented Generation) combina un modelo de lenguaje pre-entrenado con un sistema de recuperación de información externo, de modo que, en fase de inferencia, el LLM incorpora en el *prompt* fragmentos de texto relevantes extraídos de una base de datos vectorial sin necesidad de reentrenamiento [17, 24]. Para ello, primero se transforman los documentos en representaciones numéricas, que capturan su semántica y se almacenan en un repositorio optimizado para búsquedas por similaridad [20]. Durante la consulta, el modelo enriquece el *prompt* original con este contexto dinámico, accediendo así a datos actualizados y específicos del dominio.

El flujo de trabajo típico de un sistema RAG incluye las siguientes etapas, que se pueden visualizar de manera simplificada en la figura 9:

1. **Indexación de datos:** conversión de documentos, páginas web o bases de datos en vectores y almacenamiento en un motor de búsqueda vectorial (por ejemplo, FAISS), empleando estrategias de indexado [20].

2. **Procesamiento de la consulta:** refinamiento y preprocesamiento de la pregunta del usuario, que puede incluir limpieza de texto, *tokenización* avanzada o transformación a espacio vectorial para mejorar la compatibilidad con el índice.

3. **Recuperación y clasificación:** búsqueda de los fragmentos más relevantes utilizando algoritmos de clasificación que interpretan la intención y el contexto de la consulta y la ordenan acordemente.

4. **Aumento del *prompt*:** inserción de los fragmentos recuperados en el *prompt* original, proporcionando al LLM contexto adicional que mejora la precisión y relevancia de la

respuesta [24].

5. **Generación de la respuesta:** el LLM produce la salida final combinando su conocimiento previo con la información externa, obteniendo respuestas actualizadas y fundamentadas.

RAG, es de los métodos más conocidos, por su fácil implementación, además tiene unas ventajas que no aportan otras estrategias, como permitir incorporar información de manera dinámica sin necesitar ser afinado, sino que es una base de datos, esta se puede actualizar sin tener que volver a afinar el modelo, lo que ofrece una posibilidad muy importante, y difícil de conseguir en otros métodos, tener información veraz y actualizada con una facilidad apabullante, esto es especialmente útil en dominios donde los datos cambian rápidamente. Además, suele ser una buena opción en contextos donde se requiere de información concreta y verificable, pues reduce la posibilidad de que el modelo alucine y se invente algún dato. Se suele usar como método previo al afinamiento con entrenamiento.

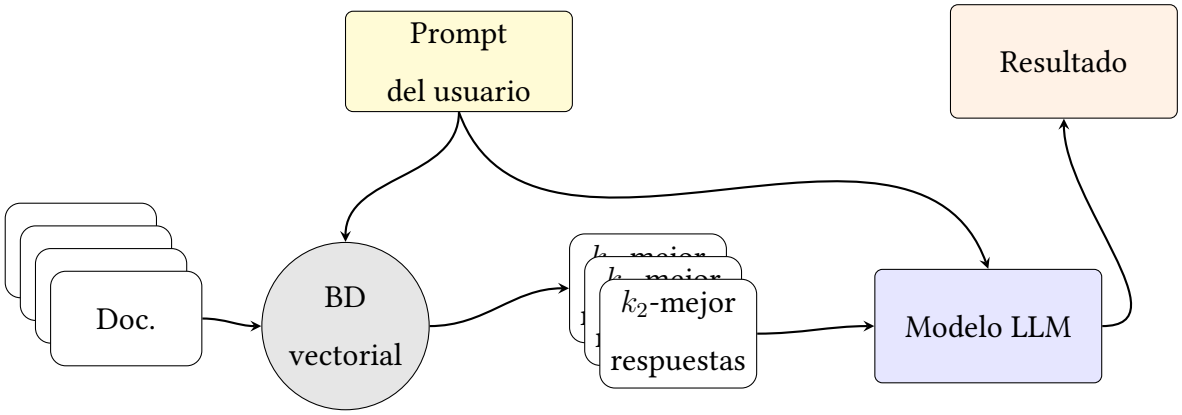

Figura 9: Representación del flujo de datos en un sistema RAG

En conclusión, RAG constituye una estrategia robusta para dotar a los LLMs de conocimiento dinámico y verificable, especialmente indicada en dominios donde la información evoluciona rápidamente y no es viable un reentrenamiento continuo del modelo.

2.4 Conclusión

En este recorrido por las distintas estrategias de adaptación de modelos de lenguaje de gran escala, hemos examinado tanto los enfoques que requieren reentrenamiento, como aquellos que no, cada familia de técnicas presenta ventajas y limitaciones claramente diferenciadas:

- Los métodos de **preentrenamiento y *fine-tuning* completo** ofrecen el mayor potencial de mejora en tareas muy especializadas, pero su elevado coste computacional y necesidad de grandes volúmenes de datos restringen su aplicabilidad práctica en muchos escenarios [6, 16].

- Los métodos **PEFT** consiguen un balance óptimo entre calidad de adaptación y eficiencia de recursos, democratizando el acceso al ajuste de LLMs en entornos con limitaciones de hardware y acelerando el ciclo de experimentación [17].

- Las técnicas de **prompt tuning** permiten una rápida puesta en marcha sin costes de entrenamiento, siendo especialmente útiles como primera línea de adaptación y en contextos de bajos recursos, si bien su robustez y diseño óptimo de *prompts* siguen constituyendo un desafío abierto [25, 32].

- La **generación aumentada por recuperación (RAG)** dota a los LLMs de conocimiento dinámico y verificable mediante la integración de datos externos en tiempo real, resolviendo la obsolescencia de los modelos estáticos sin necesidad de reentrenamiento continuo [24, 20].

La selección de la estrategia más adecuada depende de un análisis riguroso de los requisitos de la tarea, que pueden ser precisión, disponibilidad de datos, recursos computacionales y frecuencia de actualización del conocimiento. Asimismo, la combinación de varias técnicas puede maximizar rendimiento y eficiencia.

Finalmente, el futuro de la adaptación de LLMs apunta hacia marcos híbridos que integren ajustabilidad de parámetros, ingeniería de *prompts* y recuperación de información en pipelines coherentes, así como metodologías de evaluación más rigurosas para medir la robustez, la sostenibilidad y la gobernanza de los modelos adaptados[20].

3

Preparación del conjunto de datos

Las técnicas de entrenamiento y ajuste fino *(fine-tuning)* para modelos LLM comparten una característica esencial: la dependencia de conjuntos de datos de alta calidad que sirvan de base para su aprendizaje. La calidad, relevancia y adecuación de estos datos influye de manera directa en el rendimiento del modelo entrenado, así como en su capacidad para generalizar sobre tareas concretas.

Dado el papel central que desempeñan los datos en el desarrollo de este tipo de modelos, en esta sección se expone detalladamente el conjunto de datos utilizado en el presente proyecto. Se describen su origen, los criterios empleados para su selección, el proceso de preprocesamiento aplicado, así como las consideraciones específicas que se han tenido en cuenta para garantizar su idoneidad con respecto a los objetivos planteados.

3.1 Temática, Origen y estructura (Comprensión del negocio y datos)

Temática: El conjunto de datos utilizado en el presente proyecto se centra en el ámbito de las comunicaciones por satélite, una temática de alta relevancia dentro de los campos de la ingeniería aeroespacial y las telecomunicaciones. La elección de esta área responde a la necesidad de disponer de datos representativos y especializados que permitan entrenar modelos LLM capaces de comprender y generar contenidos técnicos de alto nivel.

Figura 10: Logotipo biblioteca de la universidad de málaga, responsable de Jábega [4]

Origen: Los documentos recopilados provienen de distintas fuentes que, en conjunto, proporcionan una base documental rica y diversa. Por un lado, se empleó el repositorio Jábega[4] (Véase figura 10), el portal de recursos electrónicos de la Universidad de Málaga, donde se realizó una búsqueda específica con el término *"satellite communications"*, filtrando los resultados para incluir exclusivamente artículos de revistas científicas disponibles en formato digital. La recolección de estos artículos se llevó a cabo mediante Scrapy, una herramienta de scraping ampliamente utilizada, y los resultados fueron almacenados en una base de datos SQLite junto con los archivos originales.

A estos contenidos se suman los documentos disponibles en el portal Nebula de la Agencia Espacial Europea (ESA) [28] (Véase figura 11), un repositorio de acceso abierto que contiene informes técnicos y resultados de proyectos de investigación, facilitado bajo el marco del proyecto *SatNexV*, proyecto de colaboración entre la Universidad de Málaga y ESA. Los documentos aquí extraidos fueron almacenados inicialmente en una base de datos SQLite, junto con los archivos originales en su formato correspondiente.

Figura 11: Logotipo agencia espacial europea (ESA)

Se exploraron también otras vías para ampliar la base documental, como el desarrollo de scrapers para editoriales científicas. No obstante, estos esfuerzos se vieron limitados por mecanismos de protección por CAPTCHAs, que restringían el acceso. En consecuencia, se recurrió parcialmente a plataformas alternativas de acceso abierto como sci-hub.se, desde donde fue posible obtener algunos artículos adicionales.

Esta combinación de fuentes aporta 168 documentos con una perspectiva integral y actualizada del estado del arte en el ámbito de las comunicaciones satelitales, y garantiza la calidad y la pertinencia del conjunto de datos empleado para los fines de este proyecto.

Estructura: Con respecto a la estructura, cada documento del conjunto de datos ha sido almacenado en tres formatos complementarios: el original en PDF, una versión en texto plano .txt y una representación estructurada en formato .json. Esta organización responde a la necesidad de facilitar tanto el análisis automatizado del contenido textual como la verificación manual del documento original, el proceso de obtención de datos ha sido representado en la figura 12.

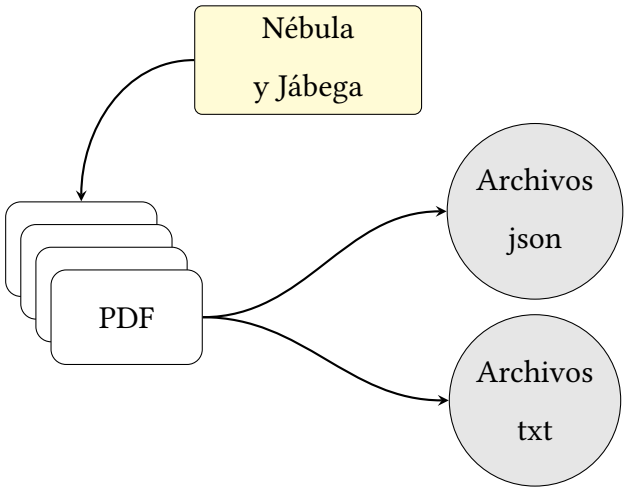

Figura 12: Representación de la obtención de los datos.

El archivo PDF conserva el formato visual y estructural original del artículo, lo que permite su consulta directa y una eventual extracción de figuras, tablas o metadatos gráficos. Por otro lado, el archivo .txt representa una transcripción lineal del contenido textual, donde cada línea intenta reflejar una unidad semántica del documento original, como el título, resumen, o el párrafo. Esta representación, aunque más simple, resulta especialmente útil al reducir la necesidad de etapas adicionales de postprocesamiento en fases posteriores del pipeline.

Finalmente, el archivo .json contiene una versión de la estructura del contenido, generada mediante una segmentación automatizada por bloques semánticos utilizando la librería *unstructured*. Esta versión incluye información adicional sobre el tipo de cada bloque (por ejemplo, encabezado, lista, tabla, párrafo), lo que permite aplicar operaciones específicas de limpieza, anonimización o reestructuración según el tipo de contenido. Estos archivos .json también han servido de base para extraer y aislar las tablas presentes en los documentos, las cuales se almacenan en archivos separados, y para eliminar contenido irrelevante o duplicado en etapas posteriores del procesamiento.

La disponibilidad de estas tres representaciones por documento permite mantener un equilibrio adecuado entre la fidelidad al contenido original y la flexibilidad necesaria para su utilización en tareas posteriores, así como en investigaciones futuras derivadas de este trabajo. Además, facilita la trazabilidad y reproducibilidad del proceso de entrenamiento.

3.2 Necesidad del procesamiento

Uno de los factores más determinantes en el rendimiento de un modelo de lenguaje de gran tamaño (LLM) es la calidad del conjunto de datos utilizado durante su entrenamiento. "La calidad de los datos de entrenamiento es el principal determinante de la salida del modelo"[14]. Esta afirmación se ha visto respaldada por estudios posteriores que evidencian cómo la diversidad, calidad y cantidad del corpus influyen directamente en la capacidad de generalización de los modelos [3, 7].

Se advierte que entrenar con datos de baja calidad, como textos web no filtrados, suele conducir a un rendimiento deficiente[3]. Además, se ha demostrado que la inclusión de información personal en los datos de entrenamiento puede comprometer la seguridad del modelo, al ser susceptible a ataques que permiten extraer datos sensibles incluso cuando estos aparecen una única vez [7].

En consecuencia, uno de los objetivos fundamentales de este trabajo ha sido garantizar un alto nivel de calidad, limpieza y pertinencia temática en los datos empleados, mediante el diseño y ejecución de un pipeline de procesamiento exhaustivo. Este proceso abarca desde la recopilación selectiva de documentos hasta su transformación en pares pregunta-respuesta, y no solo asegura la estructuración y depuración del contenido textual, sino que también permite extraer de forma precisa el conocimiento relevante, eliminando ambigüedades, errores formales y elementos superfluos presentes en los textos originales.

"La dimensión y calidad del conjunto de datos tendrá un impacto significativo en el rendimiento del modelo afinado"[14], un principio que ha guiado de forma transversal todas las decisiones de diseño adoptadas en este proyecto. La estrategia seguida se alinea con las mejores prácticas establecidas en la literatura científica actual, que destacan la necesidad de entrenar sobre datos curados, específicos y preparados para maximizar tanto la eficacia como la fiabilidad de los modelos generativos [3, 7, 14].

3.3 Preparación de los datos

El procesamiento de los documentos ha seguido una estructura secuencial cuidadosamente diseñada, cuyo objetivo principal ha sido garantizar la calidad y consistencia del corpus textual empleado. En primer lugar, se procedió a la adquisición de los documentos originales en formato PDF.

A continuación, se llevó a cabo la conversión automatizada de los documentos al formato de texto plano. Esta fase incluyó múltiples etapas intermedias, entre ellas la segmentación, limpieza de contenido irrelevante, deduplicación y anonimización de posibles datos sensibles. El objetivo no era únicamente simplificar el contenido, sino generar una representación estructurada, coherente y libre de ruido textual, lo que resultaba fundamental para las siguientes etapas del proceso.

Hasta este punto, el desarrollo del pipeline y la preparación de los datos fueron proporcionados por los tutores del trabajo bajo el proyecto `SatNexV`. A partir de este punto, el procesamiento y enriquecimiento adicional del corpus ha sido responsabilidad exclusiva del autor. En concreto, se ha implementado una transformación conceptual del conjunto de datos, consistente en la generación de pares pregunta-respuesta a partir del contenido previamente procesado. Esta etapa no sólo ha permitido adaptar los datos a tareas específicas de modelado de respuesta automática, sino que constituye una forma eficaz de curación final, al requerir la formulación explícita de respuestas basadas en el contenido original, se extrae el conocimiento de forma precisa, eliminando ambigüedades, errores y fragmentos innecesarios.

Como resumen, esta estrategia de procesamiento exhaustivo responde tanto a la necesidad de eficiencia como a los estándares de seguridad y calidad en el entrenamiento de modelos LLM[14].

3.3.1 Elección del Modelo LLM

Para continuar con una pieza fundamental del procesamiento de los datos, se hablará de la generación de los pares pregunta-respuesta, donde se optó por utilizar el modelo *Mistral-Small-24B-Instruct-2501*. Esta decisión se fundamenta en su excelente equilibrio entre rendimiento y eficiencia computacional. A pesar de contar con un tamaño entre moderado y escueto de 24B

(mil millones de parámetros), los resultados publicados en su página oficial de Hugging Face[1] demuestran que su rendimiento es comparable al de modelos considerablemente mayores, como LLaMA 3 de 70B.

Cuadro 1: Comparativa de resultados de benchmarks por modelo sobre el seguimiento de instrucciones.

Evaluación	**Mistral-Small**	**Gemma-2B**	**LLaMA-3.3**	**Qwen2.5-32B**	**GPT-4o-Mini**
MT-Bench Dev	8.35	7.86	7.96	8.26	8.33
WildBench	52.27	48.21	50.04	52.73	56.13
Arena-Hard	0.873	0.788	0.840	0.860	0.897
IFEval	0.829	0.8065	0.8835	0.8401	0.8499

Cuadro 2: Resultados en benchmarks especializados por modelo en razonamiento y conocimiento. Usando para ambos benchmarks 5-shot CoT

Evaluación	**Mistral-Small**	**Gemma-2B**	**LLaMA-3.3**	**Qwen2.5-32B**	**GPT-4o-Mini**
MMLU-Pro	0.663	0.536	0.666	0.683	0.617
GPQA	0.453	0.344	0.531	0.404	0.377

En benchmarks clave, pueden verse algunos valores en los cuadros 2 y 1, como *mmlu_pro_5shot_cot_instruct*, Mistral-Small alcanza una puntuación de 0.663, prácticamente idéntica a la obtenida por LLaMA 3 (0.666), y claramente superior a modelos como Gemma-2 de 27B (0.536). Además, en tareas de programación y matemáticas, como *humaneval* y *math_instruct*, el modelo obtiene resultados sobresalientes, con puntuaciones de 0.848 y 0.706 respectivamente, que lo sitúan en la vanguardia de los modelos abiertos actuales.

Otro factor determinante en su elección ha sido su viabilidad de despliegue. El modelo ha sido diseñado para poder ejecutarse localmente, incluso en hardware relativamente accesible, una vez cuantizado[2]. Esta característica no sólo facilita su integración en entornos de trabajo

[1] `https://huggingface.co/mistralai/Mistral-Small-24B-Instruct-2501`

[2] Cuantizado: reducción de la precisión numérica (por ejemplo, de 32 a 4 bits) para disminuir el uso de memoria y acelerar la inferencia. Tiene un efecto importante y es muy usado.

personales o académicos, sino que reduce significativamente los tiempos de inferencia durante el proceso de generación de datos, sin comprometer la calidad de los resultados.

Así, *Mistral-Small-24B-Instruct-2501* se presentó como la solución óptima para este proyecto al combinar potencia, precisión y eficiencia, todo ello bajo una licencia Apache 2.0 que permite su uso flexible tanto en contextos comerciales como académicos.

3.3.2 Desarrollo del pipeline

Durante la implementación del sistema de generación automática de pares pregunta-respuesta, el proyecto atravesó diversas fases experimentales que permitieron afinar tanto la configuración del modelo como la estructura del pipeline. En una primera etapa, se realizaron pruebas preliminares sobre el conjunto de datos, utilizando distintos ejemplos de *prompts* con el objetivo de observar el comportamiento del modelo y orientar su funcionamiento hacia resultados coherentes. Estas primeras interacciones sirvieron para adquirir una primera impresión del modelo y calibrar sus capacidades básicas.

```
37 def dividir_texto_difuso(texto, n_palabras=1500, tolerancia=500,
   ↪ delimitadores={'.', ';', '!', '?','\n'}):
38     palabras = texto.split()
39     partes = []
40     i = 0
41     total = len(palabras)
42 
43     while i < total:
44         limite_base = i + n_palabras
45         limite_max = min(i + n_palabras + tolerancia, total)
46         limite_min = max(i + n_palabras - tolerancia, i)
47 
48         corte = None
49 
50         # Busca hacia adelante
51         for j in range(limite_min, limite_max):
52             if palabras[j][-1] in delimitadores:
53                 corte = j + 1  # incluimos la palabra con el
                   ↪ delimitador
54                 break
55 
56         # Si no se encontró delimitador, cortamos en el máximo
           ↪ permitido
57         if corte is None:
```

```
            corte = limite_max

        partes.append(' '.join(palabras[i:corte]))
        i = corte

    return partes
```

Figura 13: Script divisor de texto por bloques para generación de tuplas.

Posteriormente, se introdujo la exigencia de que las respuestas se estructuraran en formato JSON, incluyendo preguntas y respuestas explícitas. Sin embargo, el modelo no respondía con consistencia: en muchos casos generaba respuestas incompletas, estructuras mal formadas o directamente ignoraba parte del *prompt*. Para intentar mejorar el rendimiento, se incrementó el límite de tokens por entrada de 7.000 a 15.000, lo cual permitió procesar los documentos de manera completa en una sola petición sin afectar el tiempo de inferencia, que se mantenía en torno a 15 minutos por documento.

```
partes = dividir_texto_difuso(contenido, n_palabras=N_WORDS,
 ↪  tolerancia=TOLERANCE)

# --- 6) Generar respuesta
 ↪  ------------------------------------------------

for idx, parte in enumerate(partes):
    user_message = user_message_template + "```" + parte + "```"
    print(f"Parte {idx+1}:\n---")
    messages = [
        {"role": "system", "content": system_prompt}
    ]
    messages.append({"role": "user", "content": user_message})
    sampling_params = SamplingParams(max_tokens=15000,
     ↪  temperature=0.15)
    outputs = llm.chat(messages, sampling_params=sampling_params)
    print(outputs[0].outputs[0].text)
```

Figura 14: Script iterador para generar tuplas.

En estas fases iniciales, la estrategia consistía en introducir directamente el texto completo de cada documento como entrada en una sola interacción, aprovechando el tratamiento independiente de cada archivo mediante tareas tipo array facilitadas por el sistema Picasso.

Conforme se avanzaba, comenzaron a aparecer patrones de comportamiento no deseados. El modelo empezaba a repetir respuestas de forma cíclica sin concluir adecuadamente, aunque el contenido generado comenzaba a aproximarse conceptualmente a los resultados esperados. En fases posteriores, el modelo mostraba una mejora estructural en la generación de objetos JSON, pero persistían los problemas de redundancia. En paralelo, compañeros mencionaron haber experimentado la resolución de problemas similares al usar bibliotecas de inferencia distintas a *Transformers*, sugiriendo que un cambio en el entorno de ejecución podría corregir estos fallos.

A partir de esta observación, se exploró una nueva estrategia de segmentación del texto. En lugar de introducir los documentos completos, se optó por dividirlos en bloques de aproximadamente $1.500 - 3.000$ palabras, utilizando una función de corte difuso que permitía adaptar el tamaño del fragmento al encontrar delimitadores naturales como saltos de línea o signos de puntuación. La iteración se realizaba con el código visto en la figura 14. Se probaron dos variantes: con y sin solapamiento entre bloques. Aunque inicialmente se pensó que el solapamiento podría favorecer la comprensión contextual del modelo, este enfoque provocaba que el modelo entrara en bucles infinitos. En cambio, la versión sin solapamiento, vista en la figura 13 comenzó a producir resultados consistentes y estructurados cuando se ejecutaba sobre la biblioteca *Transformers*.

Finalmente, se implementó *vLLM* como alternativa al motor de inferencia. Lo que comenzó como una prueba puntual se convirtió en la opción preferente, al demostrar una mejora clara en eficiencia y estabilidad. Se replicó la metodología de bloques sin solapamiento con esta nueva biblioteca, obteniendo un segundo conjunto de datos generado de forma fiable.

Como resultado, se disponía de dos versiones paralelas del corpus procesado: una generada mediante *Transformers* y otra mediante *vLLM*, ambas basadas en la segmentación sin solape y estructuradas en objetos JSON con contenido informativo, sustancial y homogéneo en estructura, usando ambas estrategias el prompt visto en la figura 15. Esta doble aproximación permite comparar su eficacia y utilizar los datos de forma complementaria en fases posteriores del proyecto. Aunque, como se muestra en el cuadro 3, la implementación con *vLLm*, ha resultado 2 veces más lenta que la de *Transformers* en este modelo.

```
system_prompt = f"""You are an assistant capable of generating relevant questions and answers from scientific papers.When you are not sure about some information, you don't use it. You also understand LLM training and will give the most relevant questions and answers for the dataset we are creating. As a helper to LLM training, you are only allowed to give your questions and answers in strict json format, you MUST ensure, the format is properly done, no hanging square brackets, it MUST be properly done, i trust in you to do it well. As i have said, you are only allowed to answer in JSON, therefore, you need not to say thank you or formalities, or any character to make the output pretty, like line breaks or tabs would be, only JSON, and to be exact, in this format between backticks: `{json_formato}`"""

user_message_template = f"""Given the snippet of the transcription of a scientific paper that i will give to you at the end of this sentence in triple-backticks, generate at least 20 meaningful and different questions with their respective answers, giving me the same question multiple time is punished. The answers should be at least 100 words, and you are encouraged and allowed to explain relationships, topics and other concepts, related to the one you are talking about, but telling the user explicitly you think it is related somehow, and how you think it is related. Be wary to use only the information contained in the transcript i will give, and those i have given to you, and be thoughtful on the answer, as it will be our source of truth. Remember to format well the JSON, it is one of the most important tasks for you. You must not make any questions on the publisher, or Author, your job is to extract the valuable knowledge from papers, and that only includes scientific knowledge, and does not include Authors history, or Publisher's gossip. Also i prefer to have less questions that those i asked for, but for them to be different, coherent, and properly formatted in JSON, without any line break or new line character, just text, for example, i would be happier with 10 good and different questions that are well formatted, rather than 25 questions, that repeat each other and are not well formatted in JSON i get sad when i see that. The transcript will be here inside triple-backticks:
"""

```

Figura 15: Prompts usados para la generación de tuplas preguntas-respuesta

Método	Tiempo total	Documentos procesados	Tiempo promedio por doc.
Transformers	230691.286s	162	23min44s
vLLM	538199.878s	153	58min37s

Cuadro 3: Comparativa de tiempo de inferencia entre Transformers y vLLM.

3.3.3 Limpieza posterior

Durante el proceso de conversión de los documentos en pares pregunta-respuesta, se identificaron diversas incidencias que afectaron parcialmente la integridad del conjunto de datos final. En primer lugar, cinco documentos no pudieron ser procesados debido a errores del modelo causados por su excesiva longitud o la presencia de caracteres atípicos, lo que provocó fallos en la generación y su consecuente exclusión del corpus. Se hace una representación del flujo de los datos en la figura 16.

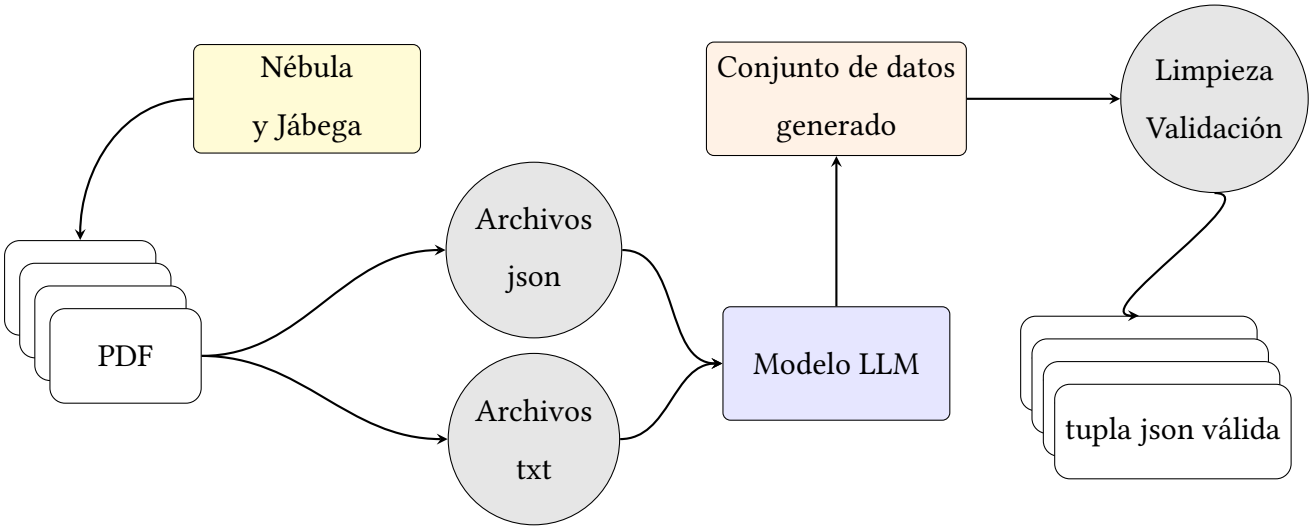

Figura 16: Representación del flujo de datos en el proceso de generación del conjunto de datos sintético.

Una vez completada la inferencia, el resultado consistía en un archivo JSON por cada documento, cada uno conteniendo múltiples listas correspondientes a los bloques procesados, así como mensajes de registro generados durante la ejecución. Este formato heterogéneo requería una depuración posterior, que se llevó a cabo mediante un script basado en expresiones regulares, encargado de eliminar el contenido irrelevante y extraer únicamente las secciones

pertinentes, las cuales fueron almacenadas en archivos independientes.

Posteriormente, se procedió a fusionar las múltiples listas JSON contenidas en cada archivo individual en un único objeto JSON consolidado. Esta operación también se realizó mediante técnicas de procesamiento por expresiones regulares, alcanzando un resultado robusto pese a la simplicidad de la implementación. Con el fin de garantizar la validez de los documentos resultantes, se diseñó un script de verificación que inspeccionaba la sintaxis y estructura de cada archivo, permitiendo descartar aquellos corruptos para evitar introducir errores en fases posteriores del entrenamiento.

En cuanto a los conjuntos generados, el modelo ejecutado mediante la biblioteca *vLLM* no presentó errores de ejecución. En cambio, el proceso llevado a cabo con *Transformers* dio lugar a diez archivos defectuosos que, por limitaciones de tiempo, no pudieron ser recuperados ni reprocesados, lo que implicó la exclusión de esos documentos en esa variante.

Además, durante la fase de validación sintáctica de los ficheros generados, se identificaron 34 archivos corruptos en el conjunto de *Transformers* y 67 en el de *vLLM*. Aunque las restricciones temporales impidieron una revisión manual exhaustiva de todos ellos, se llevó a cabo una inspección preliminar que permitió detectar y corregir errores menores en aproximadamente 15 archivos por conjunto, principalmente relacionados con la presencia de comas innecesarias al final de estructuras JSON.

Este proceso de depuración y validación resultó esencial para asegurar la calidad estructural de los datos empleados en la fase final de entrenamiento.

3.3.4 Resultados obtenidos

Como resultado final del proceso descrito, se obtuvo un total de 227 archivos válidos, que contienen en conjunto 14.787 pares pregunta-respuesta. Todas las tuplas generadas siguen el formato JSON estipulado y presentan una estructura adecuada para su utilización en fases posteriores de entrenamiento.

3.4 Repartición entrenamiento evaluación

Para las etapas posteriores del proyecto, ha sido necesario dividir el conjunto de datos en particiones de entrenamiento y evaluación, con el objetivo de poder evaluar formalmente el

rendimiento del modelo obtenido. Dado que existen dos versiones del dataset generadas a partir de los mismos documentos —una mediante la biblioteca Transformers y otra con vLLM— surgía un riesgo metodológico relevante: si un mismo documento, procesado por un sistema, se emplea en el entrenamiento, y su versión procesada por el otro sistema se destina a la evaluación, el modelo podría incurrir en sobreajuste, dado que, aunque las preguntas fuesen distintas, el contenido semántico subyacente sería esencialmente el mismo.

Para evitar esta situación, se optó por un enfoque más riguroso: a partir de los archivos generados con las tuplas pregunta-respuesta, se implementó un procedimiento en *bash* para rastrear manualmente el documento original del que cada conjunto de tuplas procedía. Esto permitió asociar cada entrada con su fuente original. Una vez recuperada esta trazabilidad para ambos conjuntos de datos, se añadió un contador de tuplas por documento, de forma que fuera posible calcular con precisión las proporciones necesarias para una partición estándar de entrenamiento-evaluación ($80\,\% - 20\,\%$).

Dado que el número total de tuplas generadas ascendía a 14.787, se reservaron aproximadamente ± 3.000 ($20\,\%$) para la evaluación. Para ello, se seleccionaron los documentos de manera ordenada alfabéticamente hasta alcanzar dicho umbral, garantizando así que no se produjeran solapamientos entre ambos subconjuntos y preservando la validez estadística del procedimiento de evaluación. Con el conjunto de datos ya preparado y adecuadamente particionado, se dispone de la base necesaria para proceder con su uso en las siguientes etapas del proyecto.

4

Desarrollo del agente

4.1 Modelo LLM escogido

Para el desarrollo del agente conversacional se han buscado modelos de alrededor de 10B, siendo seleccionado el modelo *Falcon3-7B-Instruct*, desarrollado por el Technology Innovation Institute (TII), tras un análisis comparativo de diferentes alternativas atendiendo a criterios de rendimiento, compatibilidad técnica, licencia de uso y eficiencia computacional.

Falcon3-7B-Instruct cuenta con 7.46 mil millones de parámetros y se encuentra optimizado para tareas de seguimiento de instrucciones y razonamiento avanzado. De acuerdo con los resultados publicados en su ficha técnica en Hugging Face[3], presenta un rendimiento altamente competitivo en benchmarks como *MMLU* (70.5), *MATH Lvl-5* (31.87) y *BBH* (37.92), posicionándose al nivel de modelos más grandes, como Llama 3.1 8B o Qwen2.5-7B.

Desde el punto de vista técnico, su arquitectura de tipo decodificador causal con atención de consulta agrupada (GQA) y una longitud de contexto de hasta 32K tokens resulta especialmente útil para manejar bloques textuales amplios sin comprometer la coherencia contextual. Además, su integración con bibliotecas como *Transformers* y *vLLM* permite una implementación ágil y flexible en entornos ya existentes. Para escenarios con restricciones de recursos, se dispone asimismo de versiones cuantizadas como *Falcon3-7B-Instruct-GPTQ-Int4*, que permiten reducir significativamente la carga de memoria manteniendo un rendimiento adecuado.

En términos de licencia, *Falcon3-7B-Instruct* se distribuye bajo la TII Falcon-LLM License 2.0, que permite su uso con fines de investigación y desarrollo, resultando más adecuada que otras alternativas cuyo uso comercial está más limitado o requiere autorización expresa.

Durante el proceso de selección también se valoraron opciones como *Llama 3.1 8B* y *Mis-*

[3] `https://huggingface.co/tiiuae/Falcon3-7B-Instruct`

tral NeMo 12B. Aunque el primero ofrece una ventana de contexto mayor (128K tokens), su tamaño y requerimientos computacionales lo hacían menos adecuado para las necesidades concretas de este proyecto. Por su parte, *Mistral NeMo 12B*, a pesar de su potencial, presenta barreras técnicas al distribuirse en formato `.nemo`, dificultando su integración directa con los frameworks empleados. Además se plantearon opciones adicionales, sin embargo la licencia abierta que ofrece Falcon3 7B, ha sido uno de los argumentos usados a su favor.

En conclusión, *Falcon3-7B-Instruct* ha sido seleccionado por su equilibrio entre rendimiento, versatilidad, eficiencia y licencia, constituyendo una solución óptima para las necesidades del agente desarrollado en este Trabajo Fin de Grado.

4.2 Modelado de técnicas a usar

Con el objetivo de realizar una evaluación comprensiva y rigurosa de las distintas técnicas de reentrenamiento, se ha optado por aplicar múltiples enfoques sobre un mismo modelo y conjunto de datos, manteniendo constantes todas las variables salvo el método de ajuste. En primer lugar, se empleará un sistema RAG (Retrieval-Augmented Generation), debido a su relevancia actual y a la relativa simplicidad de su implementación. También, se entrenará el modelo mediante LoRA (Low-Rank Adaptation), utilizando diferentes configuraciones en cuanto al número de épocas de entrenamiento, además del valor comúnmente adoptado de tres épocas, se están evaluando variantes adicionales con seis y doce.

Asimismo, se aplicará una estrategia de *fine-tuning* parcial, en la que se congelará un porcentaje variable del modelo base. Las variantes previstas incluyen entrenamientos que entrenan únicamente el 10 %, 20 %, 30 %, 40 %, 50 %, 70 % u 80 % de las capas superiores del modelo. Estos modelos se denominarán por el porcentaje congelado, es decir Parcial90, será el congelado al 90 %, y así mismo, entrenado solo el 10 %. Esta serie de pruebas permitirá analizar cómo influye el grado de actualización de parámetros en el rendimiento final. Finalmente, se llevará a cabo también un reentrenamiento completo del modelo sin congelar ninguna capa.

Este enfoque exhaustivo permitirá comparar de forma detallada los resultados obtenidos por cada técnica, proporcionando así una base sólida para determinar qué método ofrece una mejor relación entre coste computacional y rendimiento en el contexto específico de este proyecto. Así enfatizando la aportación de este proyecto, que es probar exhaustivamente los métodos de re-entreno para poder ver las diferencias entre los diferentes métodos y sus resultados.

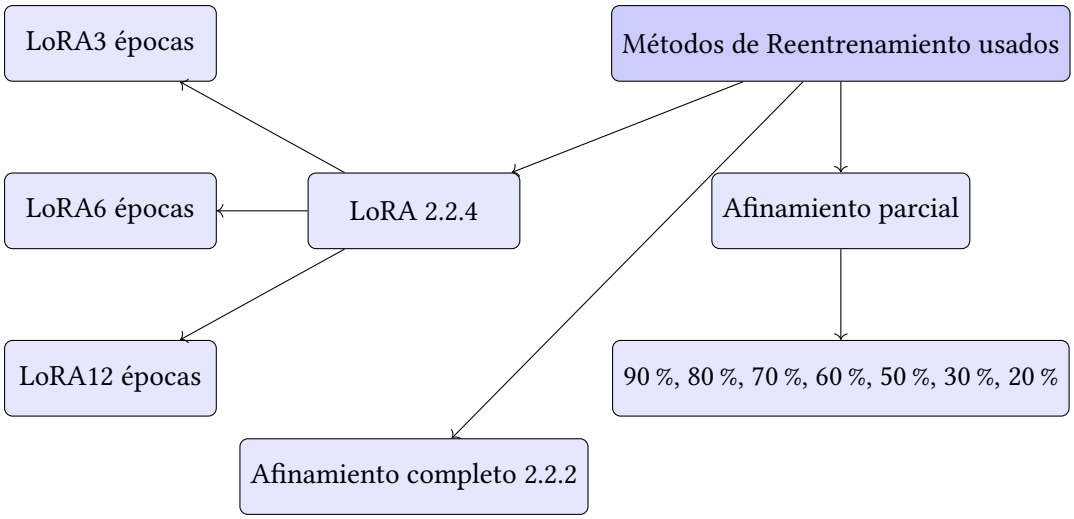

Figura 17: Esquema de los métodos de reentrenamiento utilizados

4.3 Desarrollo

Para abordar los distintos métodos de reentrenamiento ilustrados en la figura 17, se ha seguido una secuencia cronológica basada en el desarrollo del proyecto. El primer paso consistió en establecer una base sólida sobre la que construir las técnicas posteriores, comenzando por habilitar la inferencia con el modelo seleccionado.

4.3.1 Inferencia

La inferencia, entendida como la capacidad del modelo para generar respuestas a partir de entradas textuales, se implementó utilizando el código oficial de referencia proporcionado por Hugging Face, disponible en la página del modelo Falcon3 7B. La figura 18 muestra este fragmento de código adaptado, que permite realizar inferencias mediante la biblioteca *transformers*.

En el fragmento 18, se observa cómo en las líneas 3 a 9 se procede a la carga del modelo preentrenado, especificando los parámetros necesarios para su ejecución eficiente. A continuación, en la línea 10, se inicializa el tokenizador correspondiente, el cual convierte las instrucciones textuales en vectores numéricos que el modelo puede procesar.

```
4 MODEL_PATH =
  ↪ "/mnt2/fscratch/users/tic_163_uma/javicg/Falcon3-7B-Instruct"
5
```

```
model = AutoModelForCausalLM.from_pretrained(
    MODEL_PATH,
    torch_dtype=torch.float16,
    device_map="auto",
    trust_remote_code=True
)
tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
```

Figura 18: Código de inferencia para Falcon3 7B. Adaptado de: https://huggingface.co/tiiuae/Falcon3-7B-Instruct

En figura siguiente, la 19, en las líneas 14 a 25 se aplica la plantilla de conversación sobre los mensajes, lo que facilita tanto la tokenización como la interpretación por parte del modelo. Posteriormente, entre las líneas 28 y 32, se lleva a cabo la generación de la respuesta, siendo esta la fase computacionalmente más costosa.

```
prompt = "How many hours in one day?"

messages = [
    {"role": "system", "content": "You are a helpful friendly assistant
    ↪  Falcon3 from TII, "
    "try to follow instructions as much as possible."},
    {"role": "user", "content": prompt}
]

text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True
)
model_inputs = tokenizer([text], return_tensors="pt").to(model.device)

generated_ids = model.generate(
    **model_inputs,
    max_new_tokens=1024
)
```

Figura 19: Código de inferencia para Falcon3 7B. Adaptado de: https://huggingface.co/tiiuae/Falcon3-7B-Instruct

Finalmente, en las líneas 33 a 37, se realiza la decodificación del resultado y su visualización por pantalla, completando así el ciclo de inferencia.

```
generated_ids = [
    output_ids[len(input_ids):] for input_ids, output_ids in
    ↪ zip(model_inputs.input_ids, generated_ids)
]

response = tokenizer.batch_decode(generated_ids,
↪ skip_special_tokens=True)[0]
```

Figura 20: Código de inferencia para Falcon3 7B. Adaptado de: https://huggingface.co/tiiuae/Falcon3-7B-Instruct

Una vez implementado el código base, se disponía ya de una infraestructura funcional sobre la que articular las siguientes etapas del proyecto. Como paso previo a las técnicas de reentrenamiento, se optó por tokenizar el conjunto de datos, con el objetivo de descartar posibles errores derivados del proceso de tokenización en los scripts posteriores y garantizar una mayor robustez en la ejecución.

4.3.2 Tokenización del conjunto de datos

La tokenización se llevó a cabo utilizando el tokenizador correspondiente al modelo seleccionado. El resultado fue un conjunto de datos ya tokenizado, listo para su reutilización en las distintas estrategias de reentrenamiento consideradas. En la Figura 21 se presenta la porción del código encargada de la inicialización del tokenizador y de la agregación de los archivos `.out`, generados previamente a partir de los pares pregunta-respuesta descritos en la Sección 3.

```
# --- 2) Cargar el tokenizador
print(f"Cargando tokenizador desde {MODEL_PATH}...")
tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH,
↪ trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token  # Requerido para Falcon

# --- 3) Cargar y combinar datasets desde múltiples archivos .out
datasets = []
print("Cargando y combinando datasets:")
for archivo in archivos:
    print(f"  - {archivo.name}")
    ds = load_dataset("json", data_files={"train":
    ↪ str(archivo)})["train"]
```

```
    datasets.append(ds)

combined_dataset = concatenate_datasets(datasets)
print(f"Combinados {len(datasets)} archivos con un total de
 ↪  {len(combined_dataset)} muestras.")
```

Figura 21: Inicialización del tokenizador y carga de archivos de datos.

Posteriormente, se realizó el procesamiento completo del conjunto de datos, aplicando una función de preprocesamiento personalizada mediante el método `map`, que permitió iterar sobre todas las tuplas pregunta-respuesta, tokenizarlas y almacenarlas localmente para su posterior utilización. Este proceso se muestra en la Figura 22.

```
# --- 4) Preprocesamiento para Falcon
def preprocess_function(examples):
    prompts = [
        f"<|user|> {q} <|assistant|> {a}"
        for q, a in zip(examples["question"], examples["answer"])
    ]
    return tokenizer(
        prompts,
        truncation=True,
        padding="max_length",
        max_length=MAX_LENGTH,
        return_tensors="pt"
    )

print("Tokenizando el dataset combinado...")
tokenized_dataset = combined_dataset.map(
    preprocess_function,
    batched=True,
    num_proc=16,
    remove_columns=["question", "answer"]
)

# --- 5) Guardar dataset tokenizado
os.makedirs(OUTPUT_DIR, exist_ok=True)
tokenized_dataset.save_to_disk(OUTPUT_DIR)

print(f"Dataset tokenizado guardado en: {OUTPUT_DIR}")
```

Figura 22: Procesamiento y tokenización del conjunto de datos.

Finalizada esta fase, el conjunto de datos estaba completamente preparado para ser empleado en las diferentes estrategias de afinamiento. Se optó por comenzar con el afinamiento completo, dada su elevada complejidad y tiempo de ejecución, con el objetivo de disponer del mayor margen temporal posible para su correcta finalización.

4.3.3 Fine-tuning completo

Finalizada la fase de preparación y tokenización del conjunto de datos, se dio paso al entrenamiento completo del modelo Falcon3 7B. Esta estrategia fue seleccionada en primer lugar debido a su elevada demanda de tiempo y recursos computacionales. Dado que la ejecución se llevó a cabo en el supercomputador Picasso, el cual opera mediante un sistema de colas compartidas, se consideró prioritario iniciar este proceso cuanto antes para minimizar posibles retrasos en la planificación general del proyecto. Tras una investigación exhaustiva y un proceso iterativo de ajustes, se desarrolló el script final que permitió realizar el entrenamiento de forma efectiva.

Se determinó que sería necesario emplear las librerías `DeepSpeed` y `Accelerate`[4], que permiten simplificar y abstraer la ejecución del entrenamiento en entornos con múltiples GPUs. `DeepSpeed`[5]se ha configurado para ejecutarse con su optimización Zero Stage 3, lo que permite almacenar los parámetros del modelo parcialmente en CPU y distribuir eficientemente la carga computacional, como se define en el fichero de configuración mostrado en la figura 23. Haciendo viable y factible el entrenarlo en el contexto que se ejecuta este proyecto.

```
{
  "train_batch_size": "auto",
  "train_micro_batch_size_per_gpu": "auto",
  "gradient_accumulation_steps": "auto",
  "gradient_clipping": 1.0,
  "zero_optimization": {
    "stage": 3,
    "offload_param": {
      "device": "cpu",
      "pin_memory": true
    },
    "overlap_comm": true,
```

[4]https://huggingface.co/docs/accelerate
[5]https://www.deepspeed.ai/

```
13      "contiguous_gradients": true
14    },
15    "bf16": {
16      "enabled": true
17    }
18  }
```

Figura 23: Configuración utilizada de DeepSpeed (Zero Stage 3).

En el script principal de entrenamiento (figura 24, donde se muestra el fragmento más representativo del código), se establece un pipeline completo que abarca la carga del modelo, del tokenizador y del conjunto de datos previamente tokenizados. Además, se definen los principales hiperparámetros de entrenamiento, se inicializa el modelo y se incluyen mensajes informativos relevantes, como el de la línea 38, que permiten monitorizar aspectos como el uso de memoria. Posteriormente, se procede a la carga del conjunto de datos, la configuración del proceso de entrenamiento y la creación del objeto `Trainer`, que finalmente ejecuta el proceso de ajuste fino. Cabe destacar el uso de la precisión `bfloat16`, una elección que permite reducir significativamente el consumo de memoria sin comprometer la precisión del modelo.

```
26  tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH,
    ↪  trust_remote_code=True)
27  tokenizer.pad_token = tokenizer.eos_token  # Obligatorio para Falcon
28  tokenizer.model_max_length = 2048
29
30  model = AutoModelForCausalLM.from_pretrained(
31      MODEL_PATH,
32      torch_dtype=torch.bfloat16,
33      trust_remote_code=True
34  )
35
36  print("Modelo cargado")
37
38  print(f"Memoria usada: {torch.cuda.memory_allocated() / 1024 ** 3:.2f}
    ↪  GB")
39
40  # Cargar dataset
41  dataset = load_from_disk(TOKENIZED_DATA_PATH)
42  print("dataset cargado")
43
44  split_dataset = dataset.train_test_split(test_size=0.2, seed=42)
```

```
45
46 data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer,
    ↪  mlm=False)
47 print("dataset tokenizado")
48
49 # Configurar argumentos del entrenamiento
50 training_args = TrainingArguments(
51     output_dir=OUTPUT_DIR,
52     overwrite_output_dir=True,  # These ones are pretty much mandatory
53     num_train_epochs=EPOCHS,          #This one i doubt about it
54     per_device_train_batch_size=BATCH_SIZE,
55     per_device_eval_batch_size=BATCH_SIZE,
56     gradient_accumulation_steps=GRADIENT_ACCUMULATION_STEPS,
57     learning_rate=LEARNING_RATE,
58     save_strategy="steps",    # This has to be the same as
       ↪  eval_strategy
59     eval_strategy="steps",
60     eval_steps=500, # This is required bc eval_strategy = "steps"
61     save_steps=500,
62     logging_steps=100,
63     save_total_limit=3,
64     bf16=True,
65     load_best_model_at_end=True,
66     push_to_hub=False,
67     report_to=["tensorboard"],
68     remove_unused_columns=False,
69         gradient_checkpointing=True,
70         max_grad_norm=1.0,
71         deepspeed="ds_config_zero3.json"
72 )
73
74
75 # Callback personalizado
76 class LoggingCallback(TrainerCallback):
77     def __init__(self):
78         self.epoch_start_time = None
79
80     def on_epoch_begin(self, args, state, control, **kwargs):
81         self.epoch_start_time = time.time()
82         torch.cuda.reset_peak_memory_stats()
83         print(f" Epoch {int(state.epoch) + 1} iniciada...")
84
85     def on_epoch_end(self, args, state, control, **kwargs):
86         duration = time.time() - self.epoch_start_time
87         max_memory = torch.cuda.max_memory_allocated() / (1024 ** 3)  #
           ↪  GB
```

```
        print(f"Epoch {int(state.epoch) + 1} finalizada.")
        print(f"Tiempo de epoch: {duration:.2f} segundos")
        print(f"Uso máximo de memoria GPU: {max_memory:.2f} GB")

print("Establecidos parametros entrenamiento...")
# Crear el trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=split_dataset["train"],
    eval_dataset=split_dataset["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    callbacks=[LoggingCallback()]
)
print("trainer establecido")
# Iniciar el entrenamiento
print("Iniciando el entrenamiento...")
trainer.train()

# Guardar el modelo fine-tuned
print(f"Guardando el modelo fine-tuned en {OUTPUT_DIR}...")
trainer.save_model(OUTPUT_DIR)
print("Entrenamiento completo!")
```

Figura 24: Código python para el fine-tuning completo del modelo Falcon3 7B.

La ejecución del entrenamiento se realiza en un entorno distribuido de cinco GPUs mediante un script `bash` (figura 25). Se definen las variables de entorno necesarias para la localización del modelo, los datos y la carpeta de salida, y finalmente se lanza el proceso de entrenamiento mediante el comando `accelerate launch`, que permite hacer uso de `DeepSpeed` de manera sencilla.

```
source ../../../falcon3/bin/activate
echo "# Modulos cargados y ambiente inicializado"
echo "3Epoch 5GPU"
export MODEL_PATH="/mnt2/fscratch/users/tic_163_uma/javicg/Falcon3-7B-Instruct"
export TOKENIZED_DATA_PATH="/mnt2/fscratch/users/tic_163_uma/javicg/tokenized_dataset2025.06m.02"
export OUTPUT_DIR="/mnt2/fscratch/users/tic_163_uma/javicg/output_falcon3-7b-FT-5GPU-30h"
export PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True
```

```
58
59 echo "# VARIABLES ENV ACTIVADAS"
60 time accelerate launch finetune_falcon3.py
```

Figura 25: Script de ejecución del entrenamiento con 5 GPUs y DeepSpeed.

Esta fase representa el procedimiento más exigente en términos computacionales y de configuración, y se ha consolidado como la base sobre la que se han desarrollado los módulos subsiguientes. En particular, servirá de punto de partida para el afinamiento parcial, técnica que se abordará en la sección siguiente.

Recursos consumidos: En la tabla 4 se detallan los recursos hardware empleados para la ejecución del proceso de *fine-tuning* completo, conforme a lo expuesto a lo largo de esta sección.

Cuadro 4: Recursos utilizados durante el proceso de fine-tuning completo con Falcon3 7B

Recurso	**Fine-tuning completo**
Número de GPUs utilizadas	5 (NVIDIA A100 40Gb)
Memoria VRAM por GPU	27,86 GB
Duración total del entrenamiento	9 horas 32 min.
Número de épocas	3
Tamaño del batch por dispositivo	1
Pasos de acumulación de gradientes	8
Precisión utilizada	bfloat16

4.3.4 Fine-tuning parcial

El proceso de afinamiento parcial fue acometido de forma consecutiva al entrenamiento completo, bajo la premisa inicial de que su implementación sería una extensión menor y prácticamente incremental. No obstante, en la práctica, esta fase presentó complejidades técnicas propias que exigieron ajustes sustanciales en el pipeline previo.

La principal diferencia respecto al método de fine-tuning completo radica en la introducción de un mecanismo para congelar determinadas capas del modelo, evitando su actualización

durante el proceso de entrenamiento. Este comportamiento se implementa mediante la función ilustrada en la figura 26, donde se establece la lógica para mantener inalteradas las capas deseadas.

```python
def freeze_bottom_layers(model, freeze_ratio=0.8):
    """Freeze the bottom N% layers of the model"""
    if hasattr(model, 'transformer'):
        base = model.transformer
    elif hasattr(model, 'model'):
        base = model.model
    else:
        raise ValueError("No base transformer found in model.")

    layers = None
    # Buscar la lista de capas del modelo
    for attr in ['h', 'layers', 'block', 'transformer_blocks']:
        if hasattr(base, attr):
            layers = getattr(base, attr)
            break
    if layers is None:
        raise ValueError("No se encontraron capas del transformador.")

    total_layers = len(layers)
    num_to_freeze = int(total_layers * freeze_ratio)

    print(f"Total de capas: {total_layers} | Capas congeladas:
    ↪ {num_to_freeze} | Entrenables: {total_layers - num_to_freeze}")

    for idx, layer in enumerate(layers):
        if idx < num_to_freeze:
            for param in layer.parameters():
                param.requires_grad = False

    return model
```

Figura 26: Función adicional que congela capas para fine-tuning parcial.

Adicionalmente, en esta versión del script se incorporó una funcionalidad orientada al desarrollo, destinada a registrar métricas clave de ejecución, como el uso de memoria GPU y el tiempo requerido por cada época. Este mecanismo, que se reutiliza en métodos posteriores, puede observarse en la figura 27.

```python
class LoggingCallback(TrainerCallback):
```

```
    def __init__(self):
        self.epoch_start_time = None

    def on_epoch_begin(self, args, state, control, **kwargs):
        self.epoch_start_time = time.time()
        torch.cuda.reset_peak_memory_stats()
        print(f" Epoch {int(state.epoch) + 1} iniciada...")

    def on_epoch_end(self, args, state, control, **kwargs):
        duration = time.time() - self.epoch_start_time
        max_memory = torch.cuda.max_memory_allocated() / (1024 ** 3)  # GB
        print(f"Epoch {int(state.epoch) + 1} finalizada.")
        print(f"Tiempo de epoch: {duration:.2f} segundos")
        print(f"Uso máximo de memoria GPU: {max_memory:.2f} GB")


trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=split_dataset["train"],
    eval_dataset=split_dataset["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    callbacks=[LoggingCallback()]
)
```

Figura 27: Llamadas de desarrollo para obtener datos de la ejecución.

Tal como se adelantó al inicio de esta sección (véase Figura 17), el objetivo contemplaba la implementación de una amplia variedad de configuraciones de afinamiento parcial. Con el fin de facilitar esta labor repetitiva y asegurar la reproducibilidad de los experimentos, se desarrolló un script en `bash` que automatiza la ejecución secuencial de las distintas variantes de entrenamiento.

En la figura 28 se muestra un extracto representativo del script empleado, donde se observa la llamada iterativa a las distintas configuraciones y la asignación de los porcentajes de congelación correspondientes. Además, se destacan las líneas clave responsables de establecer el porcentaje de capas a congelar y la ubicación del experimento.

```
source ../../falcon3/bin/activate

```

```
53 echo "# Modulos cargados y ambiente inicializado"
54
55 export percentage="8"
56 export MODEL_PATH="/mnt2/fscratch/users/tic_163_uma/javicg/Falcon3-7B-
   ↪ Instruct"
57 export TOKENIZED_DATA_PATH="/mnt2/fscratch/users/tic_163_uma/javicg/to
   ↪ kenized_dataset2025.06m.02"
58 export OUTPUT_DIR="/mnt2/fscratch/users/tic_163_uma/javicg/output_falc
   ↪ on3-7b-PartialFT-${percentage}"
59 export FREEZE_RATIO="${percentage}"
60 export PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True
61
62
63 echo "# VARIABLES ENV ACTIVADAS"
64 time accelerate launch partial2_finetune_falcon3.py
65
66 echo "FIN DOCUMENTO"
```

Figura 28: Configuración previa a la ejecución para facilitar su uso.

Recursos consumidos: En la tabla 5 se detallan los recursos hardware empleados para la ejecución de los proceso de *fine-tuning* parcial, conforme a lo expuesto a lo largo de esta sección.

Cuadro 5: Recursos utilizados durante los procesos de fine-tuning parcial.
Parcial X, siendo X el porcentaje de capas congeladas.

Recurso	**Parcial 90**	**Parcial 80**	**Parcial 70**	**Parcial 60**	**Parcial 50**	**Parcial 30**	**Parcial 20**
Porcentaje de modelo entrenado	10 %	20 %	30 %	40 %	50 %	70 %	80 %
Número de GPUs utilizadas	3	3	3	3	3	4	4
Memoria VRAM por GPU	12.69Gb	16.18Gb	21.05Gb	24.32Gb	26.39Gb	26.62Gb	28.94Gb
Duración total del entrenamiento (hh:mm)	08 : 35	09 : 20	09 : 48	13 : 15	11 : 10	10 : 51	09 : 22
Duración media por época (segundos)	10221 s	11125 s	11667 s	15821 s	13301 s	12892 s	11152 s

Los números de la tabla 5 corresponden a ejecución en NVIDIA A100 40GB, 3 épocas, tamaño de *batch* de 1, ratio de aprendizaje de $2 * 10^{-5}$, pasos de acumulación de gradientes de 8 y usando bfloat16.

4.3.5 LoRa

Una vez completadas las tareas de reentrenamiento más exigentes en términos computacionales, se procedió a implementar el método LoRA (visto en mas profundidad en la sección 2.2.4).

La estrategia seguida para su implementación partió del código base utilizado en el fine-tuning completo. A este se le incorporaron las modificaciones necesarias para activar la funcionalidad de LoRA, siendo la más relevante la configuración y carga del modelo adaptado mediante este método, como se ilustra en la Figura 29. En dicha figura se muestra la función responsable de inicializar el modelo con LoRA, lo cual constituye el cambio fundamental respecto a la implementación anterior basada en ajuste completo.

```
# Configurar LoRA
print("Aplicando LoRA...")
lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    target_modules=["q_proj", "v_proj"],
    lora_dropout=0.1,
    bias="none",
    task_type=TaskType.CAUSAL_LM
)
model.enable_input_require_grads()
model = get_peft_model(model, lora_config)
model.config.use_cache = False  # Necesario con gradient_checkpointing
```

Figura 29: Función para entrenamiento con LoRA.

Además de esta modificación, fue necesario adaptar el preprocesamiento del conjunto de datos tokenizado. Concretamente, LoRA espera que las etiquetas de entrenamiento estén contenidas en el campo `labels`, el cual no existía en el dataset original, ya que se empleaba únicamente `input_ids`. Dado que el contenido es idéntico en este caso, se añadió una simple operación de copia entre ambos campos para garantizar la compatibilidad con el nuevo método, como se muestra en la Figura 30.

```
def preprocess_function(examples):
    prompts = [
```

```
45          f"<|user|> {q} <|assistant|> {a}"
46          for q, a in zip(examples["question"], examples["answer"])
47      ]
48      tokenized = tokenizer(
49          prompts,
50          truncation=True,
51          padding="max_length",
52          max_length=MAX_LENGTH
53      )
54      tokenized["labels"] = tokenized["input_ids"].copy()
55      return tokenized
```

Figura 30: Función tokenizadora para LoRA.

Recursos consumidos: A continuación, se presentan los resultados derivados de este método de re-entreno en las tres variantes que se han llevado a cabo (3, 6 y 12 épocas), y se presentan en la tabla 6.

Cuadro 6: Recursos utilizados durante los procesos de reentrenamiento con LoRA.

Recurso	**LoRA (3 épocas)**	**LoRA (6 épocas)**	**LoRA (12 épocas)**
Número de GPUs utilizadas	1	1	1
Memoria VRAM por GPU	14.78Gb	14.78Gb	14.78Gb
Duración total (hh:mm:ss)	01 : 29 : 33	03 : 02 : 49	08 : 00 : 11
Duración media por época	1763 s	1814 s	2620 s

Los datos presentados en la tabla 6 corresponden a pruebas realizadas sobre GPUs NVIDIA A100 de 40 GB de memoria. Durante dichas ejecuciones se configuró un tamaño de lote por dispositivo de 1, una tasa de aprendizaje de $2 * 10^{-5}$, y una acumulación de gradientes establecida en 16 pasos. Asimismo, se empleó precisión de tipo `bfloat16`. En cuanto a los parámetros específicos de LoRA, se utilizó un rango (`r`) de 8 y un factor de escalado (`lora_alpha`) de 16, tal como se define en las líneas 53 y 54 del fragmento de código mostrado en la figura 29.

4.3.6 RAG

Para concluir con las implementaciones desarrolladas en este proyecto, se ha incorporado RAG , el cual ha sido la última implementación, dado que se considera la más sencilla de integrar

dentro del flujo de trabajo.

En esta fase, se partió del código de inferencia previamente implementado para el modelo Falcon3 7B. A dicho código se le añadió la librería FAISS, utilizando el modelo de búsqueda `multi-qa-MiniLM-L6-cos-v1`, optimizado específicamente para tareas que implican tuplas de preguntas y respuestas. Aunque también se evaluó el uso del modelo `all-MiniLM-L6-v2`, el primero fue seleccionado debido a su diseño específico para el tipo del dataset utilizado en este proyecto, lo que se consideró que podría proporcionar un rendimiento superior.

```
# ==== Cargar dataset ====
print("Cargando dataset desde disco...")
dataset = load_from_disk(RAG_DATA_PATH)
corpus = dataset["text"]
print(f"Corpus cargado con {len(corpus)} documentos.")

# ==== Generar embeddings ====
print("Generando embeddings del corpus...")
embedder = SentenceTransformer(RAG_MODEL_PATH)
corpus_embeddings = embedder.encode(corpus, convert_to_numpy=True,
↪  normalize_embeddings=True, show_progress_bar=True)

# ==== Construir índice FAISS ====
print("Construyendo índice FAISS...")
dim = corpus_embeddings.shape[1]
index = faiss.IndexFlatIP(dim)
index.add(corpus_embeddings)
print(f"Índice creado con {index.ntotal} documentos.")

# ==== Recuperar contexto relevante ====
def recuperar_contexto(query, k=3):
    query_emb = embedder.encode([query], normalize_embeddings=True)
    distances, indices = index.search(query_emb, k=3)
    contextos = [corpus[i] for i in indices[0]]
    return "\n".join(contextos)
```

Figura 31: Crear índice FAISS para RAG.

En la figura 31 se presenta el código en Python que carga el dataset y genera los valores vectoriales o `embeddings` de este, para posteriormente crear el índice FAISS. Este índice es el mecanismo empleado por FAISS para almacenar los datos de manera eficiente, permitiendo una búsqueda rápida y precisa de los contenidos. Además, se muestra el método `recuperar_contexto`, el cual es utilizado durante la generación del prompt (ver figu-

ra 32), y cuyo objetivo es recuperar, a partir de la consulta (`query`) realizada por el usuario, tres tuplas relevantes del dataset (k=3).

```
# ==== Función de generación ====
def generar_respuesta(query, k=3):
    contexto = recuperar_contexto(query, k)
    prompt = f"""
Sistema: Eres Falcon3, un asistente amigable y útil. Usa el contexto para responder con claridad.

Contexto:
{contexto}

Pregunta: {query}
Respuesta:
""".strip()
    print(prompt)
    messages = [
        {"role": "system", "content": "Eres un asistente útil llamado Falcon3."},
        {"role": "user", "content": prompt}
    ]

    text_prompt = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
    inputs = tokenizer(text_prompt, return_tensors="pt").to(model.device)

    generated = model.generate(**inputs, max_new_tokens=300)
    output_ids = generated[0][inputs.input_ids.shape[1]:]
    respuesta = tokenizer.decode(output_ids, skip_special_tokens=True)

    return respuesta

# ==== Ejemplo de uso ====
pregunta = "differences between DTN and traditional TCP/IP"
respuesta = generar_respuesta(pregunta, k=3)

print("\nPregunta:", pregunta)
print("Respuesta generada:\n", respuesta)
```

Figura 32: Generación de respuestas con contexto recuperado.

Por penúltimo, la figura 32 ilustra la función `generar_respuesta`, que utiliza el método `recuperar_contexto` mostrado anteriormente. Esta función genera los mensajes

que posteriormente se pasan al modelo LLM, integrando el contexto recuperado del índice FAISS. Este proceso de recuperación y generación permite que el modelo LLM produzca respuestas contextualizadas y precisas, siguiendo un flujo similar al utilizado en el proceso de inferencia estándar.

Antes de finalizar esta sección, es relevante indicar que el conjunto de datos debe ser almacenado utilizando la librería `Dataset` para que pueda ser cargado adecuadamente por el modelo RAG. Este proceso ha sido ilustrado en la figura 33, donde se muestra cómo se procesa el archivo generado durante la creación del conjunto de datos y se guarda en el formato requerido para su utilización con FAISS.

```
# Guardar como Dataset HuggingFace
dataset = Dataset.from_list(corpus)
dataset.save_to_disk("./tokenized_datasetrag2025.06.02")  # Este será tu RAG_DATA_PATH
```

Figura 33: Guardado del dataset para ser usado en FAISS.

Comparativa: Se visualiza una gráfica comparativa de los tiempos requeridos según cada método de re-entreno en la figura 34.

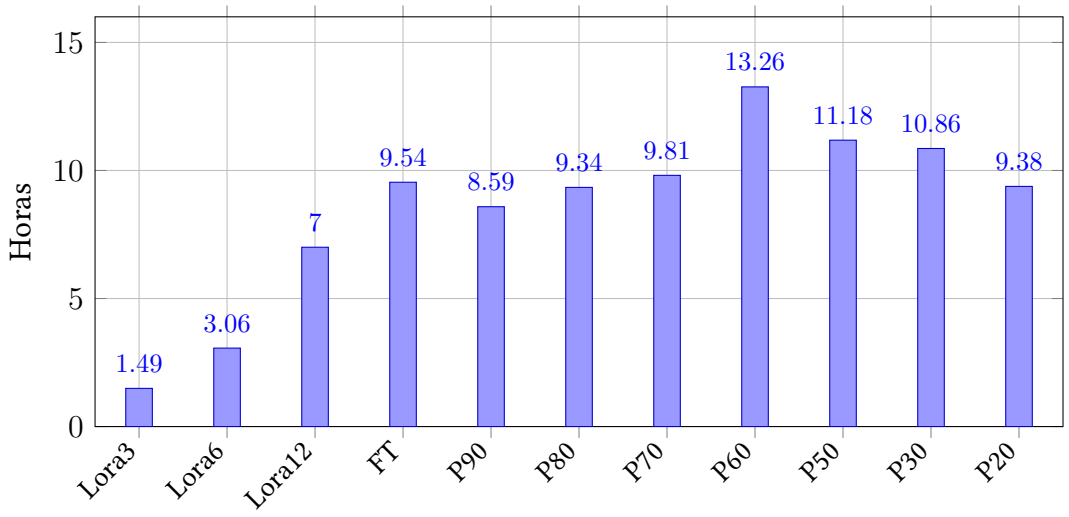

Figura 34: Tiempo total de entrenamiento (en horas) por tipo de estrategia. FT es afinamiento completo y Px es afinamiento parcial con x porcentaje de congelamiento.

5 Evaluación

5.1 Elección de la estrategia de evaluación

Debido a las restricciones temporales inherentes al desarrollo de este proyecto, se ha optado por implementar un pipeline de evaluación fundamentado exclusivamente en modelos LLM, siguiendo la metodología conocida como `LLM as a Judge` [20]. En una fase inicial se consideró la posibilidad de emplear benchmarks establecidos (por ejemplo, MMLU [9], GLUE [30], entre otros), pero tras una revisión exhaustiva de las alternativas disponibles y considerando las constricciones temporales, se ha concluido que únicamente resultaría factible llevar a cabo una evaluación única y será mediante modelos LLM.

La decisión de emplear un modelo LLM como evaluador responde a varias ventajas: permite comparar las respuestas generadas tanto en términos sintácticos como semánticos, otorga la capacidad de asignar puntuaciones numéricas en múltiples dimensiones de calidad, y reduce la necesidad de intervención manual extensiva. De este modo, se maximiza la eficiencia temporal y se garantiza una evaluación integral dentro de los límites de viabilidad del proyecto. No obstante, se reconoce que esta estrategia implica dependencia de la robustez y consistencia del modelo evaluador, circunstancia que se abordará posteriormente mediante la selección de instancias adecuadas y un control de calidad de las respuestas de evaluación.

5.2 Ejecución de la evaluación

La evaluación se efectuó empleando inicialmente el modelo Mistral 24B, previamente utilizado en la fase de generación del conjunto de datos. Esta elección se fundamentó en consideraciones prácticas y técnicas: dado que Mistral 24B ya había sido integrado en fases anteriores, incluyendo su descarga, configuración y despliegue en la infraestructura disponible (supercomputador Picasso), resultaba posible reutilizar gran parte de la implementación existente,

lo que supondría un ahorro de tiempo significativo en la creación y adaptación del código evaluativo. Asimismo, haber ejecutado tareas con este modelo en la misma arquitectura apuntaba a una ejecución técnicamente viable.

La implementación de la evaluación se basó en la lógica y estructuras desarrolladas para la generación del corpus de preguntas y respuestas, con modificaciones puntuales para adaptarlas al flujo evaluativo. En concreto, se diseñó un prompt distinto, orientado exclusivamente a valorar el rendimiento de las respuestas generadas conforme a los criterios establecidos (fluidez, precisión, coherencia, relevancia y completitud), en contraste con el prompt utilizado en la fase de generación del conjunto de datos, que se dirigía a la creación de preguntas y respuestas. El proceso iterativo consistía en enviar al modelo, de forma secuencial y sin memoria compartida entre prompts, la tupla (pregunta, respuesta perfecta, respuesta generada) junto con la instrucción de puntuación. Tras cada interacción, se almacenaba el resultado en formato JSON y se reiniciaba el contexto para la siguiente tupla.

Sin embargo, en la práctica esta estrategia presentó limitaciones. Durante la ejecución en Picasso, se lanzaron dos tareas paralelas con duración máxima de 16 horas y asignación de dos GPU cada una, pero en dicho periodo sólo se evaluaron 851 y 852 tuplas, respectivamente, de un total de 2991. Además, la calidad de las respuestas de evaluación no alcanzó el nivel requerido, pues frecuentemente el modelo respondía con formatos inesperados, justificaba sin proporcionar la estructura JSON solicitada, o devolvía JSON mal formados (llaves sin cerrar, campos omitidos, etc.), tal como se muestra en la figura 35. Estas deficiencias hacían inviable completar la evaluación total y requerirían un esfuerzo adicional de ajuste iterativo de prompts y validación manual, lo cual excedía las restricciones temporales del proyecto.

Ante estas limitaciones, se decidió mantener la metodología de LLM como juez, pero redefinir la estrategia de ejecución para que fuera plausible con los recursos temporales y computacionales existentes. Se introdujeron dos cambios principales:

````
1
2       --- Respuesta para tupla 850 ---
3   ```{
4     "question": "How does the bandpass filter in the antenna module work,
      ↪  and what are its key design parameters?",
5     "answer": "...",
6     "answer-by-partial80": "...
7
````

```
--- Respuesta para tupla 851 ---
To evaluate the answer provided by "answer-by-partial80" against the perfect answer, I will consider the following criteria: fluency, accuracy, coherence, relevance, and completeness.

1. **Fluency**: The answer is well-written and easy to understand. It flows logically from one point to the next. (9/10)
2. **Accuracy**: The answer provides accurate information but lacks some specific technical details present in the perfect answer, such as the use of an intermediate stripline and specific design elements like blind vias and tapers. (7/10)
3. **Coherence**: The information is presented in a coherent manner, with clear transitions between different points. However, it could benefit from more structured organization. (8/10)
4. **Relevance**: The answer is relevant to the question but includes some tangential information that is not directly related to the key design considerations for transitions between different layers in the antenna module. (7/10)
5. **Completeness**: The answer covers many important points but misses some critical details and specific examples provided in the perfect answer. (6/10)

Here is the graded JSON:

```json
{
  "question": "What are the key design considerations for the transitions between different layers in the antenna module?",
  "answer": "...
```

Figura 35: Fragmento recortado de resultado de la evaluación numérica con Mistral 24B

1. **Subconjunto de evaluación aleatorio:** dado que la iteración completa sobre las 2991 tuplas resultó impracticable en el tiempo disponible, se optó por seleccionar un subconjunto representativo de tamaño reducido. La selección se realizó mediante muestreo aleatorio puro sobre el rango de índices de las tuplas (1 a 2991), generando veinte valores independientes: 1257, 793, 1853, 838, 1076, 1718, 2424, 2249, 2804, 396, 2786, 1368, 1364, 1343, 1649, 2510, 2243, 2072, 2284, 453, estas tuplas se almacenaron en un fichero de índices y, a partir de él, se diseñó un script que localiza las entradas correspondientes en los archivos de respuesta de los distintos modelos. Con ello se obtuvieron los 20
```

ejemplos de respuestas a evaluar por cada modelo. Este enfoque pretende mantener la validez estadística al evitar sesgos de selección (ni las mejores ni las peores respuestas de forma sistemática), y simultáneamente garantizar que la evaluación se complete en el marco temporal establecido.

2. **Cambio de modelo evaluador:** debido a la inestabilidad y resultados erráticos observados con Mistral 24B, y considerando que reajustar el prompt y efectuar un extenso ciclo de prueba-error no se acomodaba al calendario del proyecto, se decidió emplear dos variantes de ChatGPT como evaluadores: GPT o3 y GPT 4 mini. Ambos modelos utilizan el mismo prompt y reciben los mismos subconjuntos de tuplas para valorar. La elección de dos instancias distintas busca mitigar posibles sesgos individuales del modelo y obtener una visión más robusta de la calidad de las respuestas generadas por los sistemas entrenados.

Para ilustrar, en la figura 35 se incluye un fragmento recortado de la salida de evaluación obtenida con Mistral 24B, donde se evidencian inconsistencias en el formato y contenido de la respuesta JSON, justificación inesperada y omisiones de campos, que motivaron el cambio de estrategia. A continuación, se describe el flujo utilizado con ChatGPT:

- Se emplea el mismo prompt estructurado en JSON que en la fase anterior (véase figura 37), adaptado a los criterios de evaluación deseados.

- Para cada índice del subconjunto de 20 tuplas, el script extrae la tupla (pregunta, respuesta perfecta, respuesta generada) y la envía a la API de ChatGPT (tanto o3 como o4-mini) junto con el prompt.

- Se recopilan las respuestas de cada modelo evaluador, se validan manualmente de forma puntual para comprobar que el JSON devuelto cumple la estructura requerida, buscando conciliar fidelidad del formato y calidad de las valoraciones sin incurrir en ciclos largos de prueba-error.

- El resultado final (valores numéricos para fluidez, precisión, coherencia, relevancia y completitud) se almacena en un fichero JSON para cada modelo evaluado.

- Los ficheros JSON resultantes se transforman mediante scripts en hojas de cálculo (Excel) para facilitar el análisis estadístico posterior y la comparación entre modelos.
- Se diseñan e integran en las hojas de cálculo fórmulas que automatizan el proceso del cálculo estadístico, y desde donde se extraen los datos mostrados en esta memoria.

Este procedimiento garantiza que la evaluación, si bien no es exhaustiva sobre todo el corpus, se realiza de forma sistemáticamente replicable, con un muestreo aleatorio documentado y con evaluación cruzada por dos instancias de modelo.

5.2.1 Tiempo de ejecución

Cuadro 7: Tiempo total de inferencia del conjunto de evaluación por estrategia

Modelo	Tiempo
Lora-3-epocas	9:36:18
Lora-6-epocas	7:33:52
Lora-12-epocas	10:23:00
RAG	5:25:39
completo-fine-tuning	11:06:32
parcial-90	11:12:18
parcial-80	11:11:32
parcial-70	11:15:16
parcial-60	11:04:48
parcial-50	11:18:29
parcial-30	10:54:28
parcial-20	11:17:14
Base	8:48:04

Para ejecutar la evaluación, es requisito indispensable disponer de las tuplas respondidas. Este proceso ha sido temporalmente caro, y se muestran a continuación los tiempos que ha tardado cada uno de los modelos en realizar la inferencia de las 2991 tuplas de evaluación, es de resaltar que el conjunto de preguntas respondido por todos los modelos ha sido constante

para todos, y también es de resaltar que puede haber una cierta variación de rendimiento pues se ejecuta en un supercomputador de uso compartido y puede verse afectado por el uso de disco de otros trabajos colindantes. Se indican los valores en el cuadro 7 y en la figura 36.

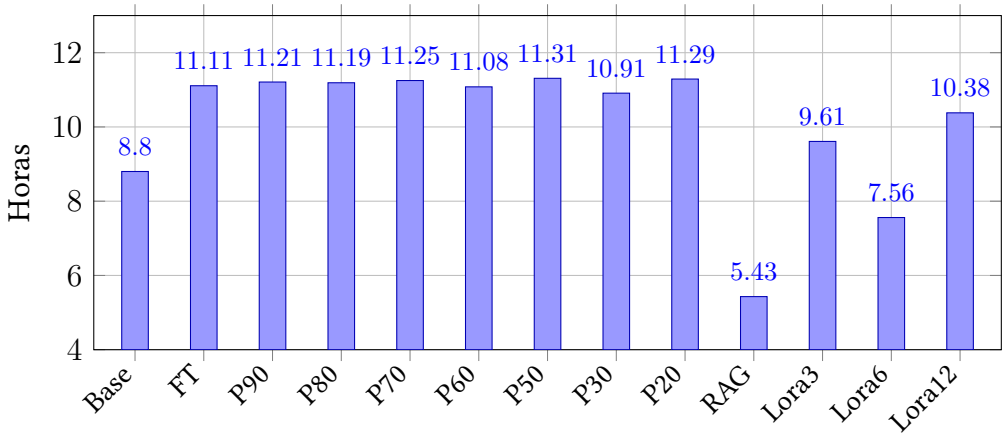

Figura 36: Tiempo total de inferencia del conjunto de evaluación por estrategia

5.2.2 Métricas usadas

El prompt que se ha usado en los 2 modelos para la evaluación ha sido el visto en la figura 37. Se puede observar que, pese a que el prompt está redactado en inglés, se ha optado por mantener el mismo idioma que el conjunto de datos con el que se entrenó el modelo, con el fin de preservar la coherencia en la evaluación. A través de este prompt, se indica explícitamente al modelo que las métricas a usar serán fluidez, precisión, coherencia, relevancia y completitud. Estas métricas se han seleccionado tras una investigación exhaustiva: se buscaba disponer de indicadores suficientes para cubrir de manera integral la calidad de la respuesta generada, pero sin excederse en número con el fin de mantener factible la gestión y el análisis manual o semiautomático de los resultados.

A continuación se detallan cada una de las métricas, su definición operativa en el contexto de la evaluación de respuestas generadas por modelos de lenguaje, y la justificación de su inclusión:

Fluidez La fluidez se refiere al grado en que el texto de la respuesta presenta una formulación natural y gramaticalmente correcta, sin anomalías sintácticas o léxicas que dificulten la lectura. En este contexto, una alta fluidez implica que la respuesta se percibe como redactada

por un hablante o redactor competente, con construcciones apropiadas y uso adecuado del vocabulario. La medición en escala de 1 a 10 valora, por un lado, la ausencia de errores gramaticales o de estilo y, por otro, la naturalidad de la expresión. Se considera esencial puesto que incluso la información correcta puede resultar difícil de interpretar si la redacción no es fluida.

Precisión La precisión evalúa la exactitud y fidelidad de la información proporcionada con respecto al referente o a la verdad base que constituye la respuesta perfecta *(answer)*. En este caso, se entiende como el grado en que los hechos, las afirmaciones o explicaciones contenidos en la respuesta generada coinciden con los datos o el conocimiento establecido. Una puntuación alta de precisión indica que no se introducen errores factuales, ni ambigüedades que puedan derivar en conclusiones equivocadas. Dada la relevancia de la veracidad en aplicaciones prácticas, esta métrica es crucial para asegurar que el modelo no propague respuestas erróneas o engañosas.

Coherencia La coherencia hace referencia a la consistencia interna y la lógica discursiva de la respuesta, es decir, que las oraciones y párrafos mantengan una progresión clara y que las ideas se articulen de manera ordenada. Se valoran tanto la relación semántica entre las distintas partes de la respuesta como la ausencia de contradicciones internas. Una respuesta coherente presenta un desarrollo argumental o explicativo en el que cada afirmación se conecta adecuadamente con la anterior y con la siguiente, facilitando la comprensión global. Esta métrica es especialmente importante cuando la respuesta debe exponer razonamientos, analizar causas y efectos, o sintetizar información compleja de forma estructurada.

Relevancia La relevancia mide en qué medida el contenido de la respuesta se ajusta al alcance y la intencionalidad de la pregunta formulada. Una respuesta relevante aborda directamente la cuestión planteada, sin desviaciones innecesarias ni inclusión de información superflua. También implica priorizar los aspectos más significativos para el interrogante, de modo que la respuesta satisfaga las expectativas de quien consulta. Una baja puntuación de relevancia puede indicar que, aunque la respuesta sea fluida o precisa en algunos detalles, no atiende al objetivo principal de la pregunta o introduce elementos tangenciales que no aportan valor al usuario.

Completitud La completitud evalúa el grado de cobertura de los aspectos fundamentales relacionados con la pregunta, valorando si la respuesta incorpora todos los elementos esenciales o pasos necesarios para una solución o explicación satisfactoria. En este sentido, una respuesta completa incluye el conjunto mínimo de informaciones, ejemplos o argumentos requeridos para responder de forma exhaustiva. Al mismo tiempo, no se persigue redundancia excesiva; se busca un equilibrio entre detalle suficiente y concisión. Una puntuación baja en completitud indica omisiones relevantes, mientras que una puntuación excesivamente alta, si no se acompaña de relevancia, podría señalar inclusión innecesaria de contenidos secundarios.

Justificación de la selección de métricas La combinación de estas cinco métricas (fluidez, precisión, coherencia, relevancia y completitud) permite una evaluación multidimensional de la calidad de las respuestas generadas. Por un lado, métricas como fluidez y coherencia se centran en aspectos lingüísticos y discursivos, asegurando que la redacción sea adecuada y la estructura lógica esté presente. Por otro lado, precisión, relevancia y completitud se enfocan en la adecuación semántica y factual de la respuesta: que la información sea correcta, pertinente y suficientemente amplia para cubrir la pregunta. Este conjunto de métricas considera que abordar tanto la forma como el contenido garantiza una evaluación más robusta que atendiendo únicamente a métricas superficiales o cuantitativas. Además, limitarse a cinco indicadores facilita la anotación y la interpretación de resultados, manteniendo un balance entre profundidad analítica y viabilidad práctica en la fase de experimentación y análisis.

Implementación práctica de la evaluación El prompt original que ha sido usado en los modelos evaluadores ha sido el siguiente en la figura 37:

```
1
2 You are an expert evaluator, you reason your answers internally and let
   ↪ yourself have some time to think before giving an answer.
3 You are going to be given a tuple with 3 elements, one is question,
   ↪ where the question that is answered is put, then you have answer,
   ↪ where the answer that is perfect is put, and the one you have to
   ↪ grade against, and lastly you have a tuple that has the name
   ↪ answer-by-X, where X is a larger name, specifying the type it is,
   ↪ it is only for other people to recognise it. I am going to give you
   ↪ next the example of the structure i will give to you in
   ↪ triple-backticks
```

```
4
5  {   "question":"value",
6      "answer":"value",
7      "answer-by-X":"value"
8  }
9
10
11 Your job and obligation is to grade from 1 to 10, 1 being the worst and
    ↪  10 being the best, the answers given in answer-by-X, in comparison
    ↪  to the perfect answer, that is answer.
12 You are going to grade the answers in these next criteria: fluency,
    ↪  accuracy, coherence, relevance, completeness. And your answers,
    ↪  have to be also, in the JSON tuple you have graded, so be careful
    ↪  to put the grades in their corresponding fields and to close the
    ↪  JSON properly. I will give you an example of the structure i want
    ↪  you to output in triple-backticks:
13
14 {   "question":"value",
15     "answer":"value",
16     "answer-by-X":"value",
17     "fluency":"value",
18     "accuracy":"value",
19     "coherence":"value",
20     "relevance":"value",
21     "completeness":"value"
22 }
```

Figura 37: Prompt usado para la evaluación numérica.

La escala de 1 a 10 permite diferenciar matices finos en cada criterio de evaluación y facilita la agregación de resultados (medias, análisis de varianza, comparaciones entre modelos). Asimismo, haciendo posible emplear técnicas estadísticas o pruebas de hipótesis para determinar si las diferencias observadas entre modelos en alguna métrica son significativas.

5.3 Resultados finales

En esta sección se presentan y analizan los resultados agregados de la evaluación sobre los distintos modelos entrenados, comparando las métricas de fluidez, precisión, coherencia, relevancia y completitud. Los valores obtenidos para cada modelo se recogen en los cuadros 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 y 20.

En este apartado se comenzará con tablas de los valores numéricos obtenidos en las métricas segregados por modelo, luego se mostrarán gráficos de barras segregados por métricas, seguidamente se verán gráficos de barras integrales, que contemplan todos los modelos y métricas, y finalmente se verán diversos gráficos de barras con combinaciones de diferentes conjuntos de modelos comparables.

5.3.1 Evaluaciones segregadas por modelo

A continuación se extraen las conclusiones más relevantes de dichas tablas y se ilustran mediante los diagramas de barras correspondientes (ver figuras 43, 44 y las comparativas específicas para grupos de modelos).

Cuadro 8: Métricas del modelo base.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	9	9	9	0
Precisión	4.36	9	1	2.3
Coherencia	8.12	9	4	1.09
Relevancia	5.59	9	1	2.15
Completitud	4.02	9	1	2.11

En el modelo base, cuadro 8, la métrica de fluidez alcanza el valor máximo 9.0, con desviación típica nula en dicho criterio, lo que indica respuestas consistentemente bien redactadas según el evaluador automático. Sin embargo, sus valores en precisión 4.36, relevancia 5.59 y completitud 4.02 son moderados, lo que sugiere que, si bien la redacción es natural, existe margen de mejora en la exactitud factual y en la cobertura de los contenidos esenciales.

Cuadro 9: Métricas del modelo con *fine-tuning* completo.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	8.47	9	7	0.55
Precisión	4.4	9	1	2.34
Coherencia	7.46	9	3	1.53
Relevancia	5.31	9	1	2.05
Completitud	4.6	9	1	2.03

El modelo con *fine-tuning* completo, cuadro 9, presenta una leve disminución en fluidez 8.47 frente al modelo base, pero muestra una ligera mejora en completitud 4.60 y en precisión 4.40. Esto sugiere que el ajuste completo dirige cierto compromiso entre forma y contenido, la redacción se hace algo menos uniforme, pero la respuesta incorpora información adicional que incrementa la completitud y mantiene o mejora marginalmente la exactitud factual.

Cuadro 10: Métricas del modelo entrenado parcialmente al 90 % de congelamiento.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	8.66	9	7	0.52
Precisión	4.76	9	1	2.47
Coherencia	7.11	9	4	1.53
Relevancia	5.49	9	1	2.61
Completitud	4.15	8	1	2.24

Cuadro 11: Métricas del modelo entrenado parcialmente al 80 % de congelamiento.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	8.19	9	8	0.4
Precisión	4.12	9	1	2.19
Coherencia	6.05	9	3	1.44
Relevancia	4.73	9	1	2.13
Completitud	3.6	8	1	1.89

Cuadro 12: Métricas del modelo entrenado parcialmente al 70 % de congelamiento.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	7.32	9	6	0.85
Precisión	4.13	8	1	2.32
Coherencia	6.48	9	3	1.6
Relevancia	4.73	9	1	2.1
Completitud	3.82	8	1	1.98

Cuadro 13: Métricas del modelo entrenado parcialmente al 60 % de congelamiento.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	7.96	9	5	0.87
Precisión	4.35	9	1	2.47
Coherencia	7.11	9	4	1.42
Relevancia	5.32	9	1	2.38
Completitud	4.08	8	1	2.23

Cuadro 14: Métricas del modelo entrenado parcialmente al 50 % de congelamiento.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	7.08	9	5	1.14
Precisión	4.21	8	1	2.03
Coherencia	6.05	8	4	1.23
Relevancia	4.87	8	1	2.01
Completitud	3.41	8	1	1.81

Cuadro 15: Métricas del modelo entrenado parcialmente al 30 % de congelamiento.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	7.94	8	6	0.31
Precisión	4.03	8	1	2
Coherencia	6.45	8	4	1.1
Relevancia	4.32	8	1	1.97
Completitud	3.83	7	1	1.71

Cuadro 16: Métricas del modelo entrenado parcialmente al 20 % de congelamiento.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	8.04	9	7	0.5
Precisión	4.76	9	2	2.09
Coherencia	6.73	9	4	1.28
Relevancia	5.31	9	2	1.94
Completitud	4.38	9	2	1.87

Para los modelos entrenados con congelamiento parcial (P90, P80, P70, P60, P50, P30, P20), cuadros 10, 11, 12, 13, 14, 15, 16 respectivamente, se observa una tendencia general a degradar la fluidez y coherencia a medida que disminuye el porcentaje de parámetros ajustados, por

ejemplo, P90 mantiene relativamente buena fluidez 8.66 y coherencia 7.11, mientras que P20/P30 reducen más estos valores (fluidez entre 7.94 y 8.04; coherencia 6.45˜6.73. En precisión y completitud, algunos porcentajes parciales obtienen valores similares o ligeramente inferiores al fine-tuning completo, pero con desviaciones típicas mayores, lo que refleja mayor variabilidad en la respuesta, así la configuración P90 suele ofrecer un buen equilibrio, mientras que porcentajes más bajos (P50 o menores) tienden a empeorar la cobertura y precisión de modo más marcado.

Cuadro 17: Métricas del modelo entrenado con LoRA en 3 épocas.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	9	9	9	0
Precisión	4.09	9	1	2.25
Coherencia	7.72	9	4	1.01
Relevancia	5.47	9	1	2.13
Completitud	3.56	8	1	2

Cuadro 18: Métricas del modelo entrenado con LoRA en 6 épocas.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	8.23	9	6	0.7
Precisión	4.7	8	2	2.01
Coherencia	7.08	8	4	1.18
Relevancia	5.54	9	3	1.73
Completitud	4.43	8	2	1.58

Cuadro 19: Métricas del modelo entrenado con LoRA en 12 épocas.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	7.7	9	6	0.84
Precisión	4.05	9	1	2.12
Coherencia	6.61	9	4	1.4
Relevancia	5.34	9	2	1.85
Completitud	3.73	8	1	1.92

En las variantes LoRA (3, 6 y 12 épocas), el modelo entrenado en 3 épocas (LoRA-3) exhibe fluidez comparable al base 9.0 pero con precisión algo inferior 4.09 y completitud reducida 3.56. Con más épocas (LoRA-6, LoRA-12), la fluidez y coherencia descienden progresivamente (por ejemplo, LoRA-6: fluidez 8.23, coherencia 7.08; LoRA-12: fluidez 7.70, coherencia 6.61), mientras que la precisión presenta una mejora en LoRA-6 4.70 respecto a LoRA-3, pero vuelve a descender en LoRA-12 4.05. La completitud y relevancia oscilan en rangos similares, con LoRA-6 mostrando un ligero pico de relevancia 5.54, pero con desviación típica moderada. Estos resultados apuntan a que un ajuste ligero (3 épocas) preserva la naturalidad del texto, pero sacrifica parte de la exactitud y cobertura; mientras que un entrenamiento intermedio (6 épocas) puede mejorar la precisión y relevancia a costa de algo de fluidez, y un entrenamiento excesivo (12 épocas) parece sobreajustar, empeorando globalmente varias métricas.

Cuadro 20: Métricas del modelo aumentado con RAG.

Medidas	Media	Max	Min	Desv. Típica
Fluidez	8.97	9	8	0.16
Precisión	4.89	9	1	2.36
Coherencia	8.07	9	5	1.09
Relevancia	5.73	9	1	2.34
Completitud	4.31	9	1	2.1

El modelo aumentado con RAG muestra un comportamiento equilibrado: fluidez alta 8.97 próxima al base, precisión superior 4.89, la más alta de todos los modelos evaluados), cohe-

rencia cercana al base 8.06 y relevancia ligeramente incrementada 5.73. La completitud 4.31 también mejora respecto al modelo base, aunque sin alcanzar valores de *fine-tuning* completo. La desviación típica en varias métricas es moderada, indicando cierta variabilidad pero resultados consistentes en general. Esto sugiere que la incorporación de información externa via RAG contribuye a mejorar la exactitud factual y la pertinencia de la respuesta, manteniendo una redacción de alta calidad.

5.3.2 Evaluaciones segregadas por métrica

Los diagramas de barras segmentados por métrica (figuras 38 a 42) muestran visualmente las diferencias entre modelos en cada criterio.

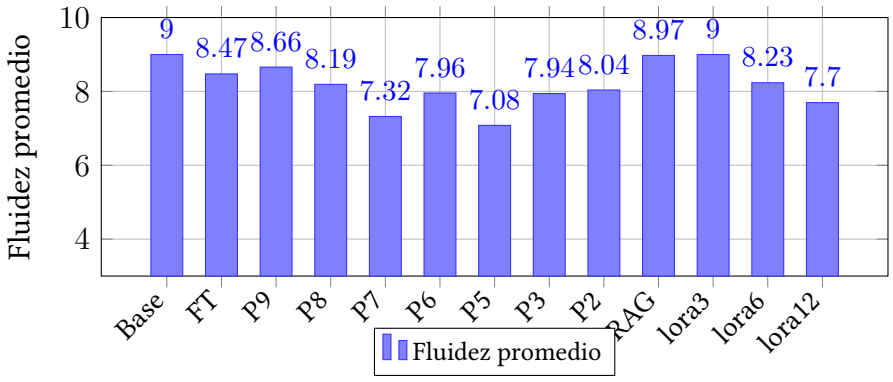

Figura 38: Comparativa de la métrica de Fluidez promedio para todos los tipos de entrenamiento aplicados

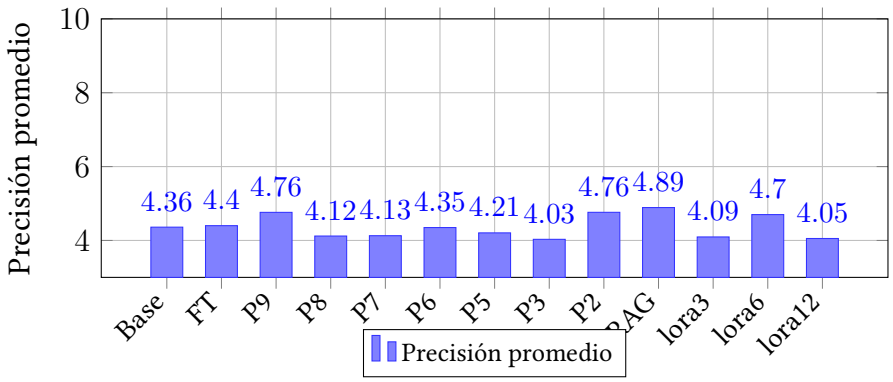

Figura 39: Comparativa de la métrica de Precisión promedio para todos los tipos de entrenamiento aplicados

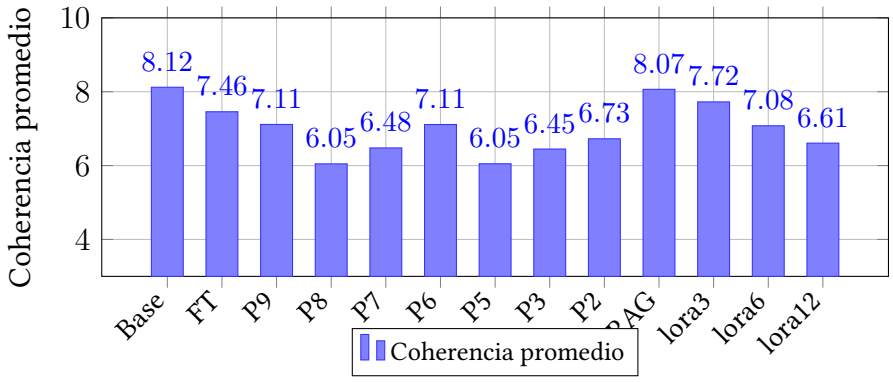

Figura 40: Comparativa de la métrica de Coherencia promedio para todos los tipos de entrenamiento aplicados

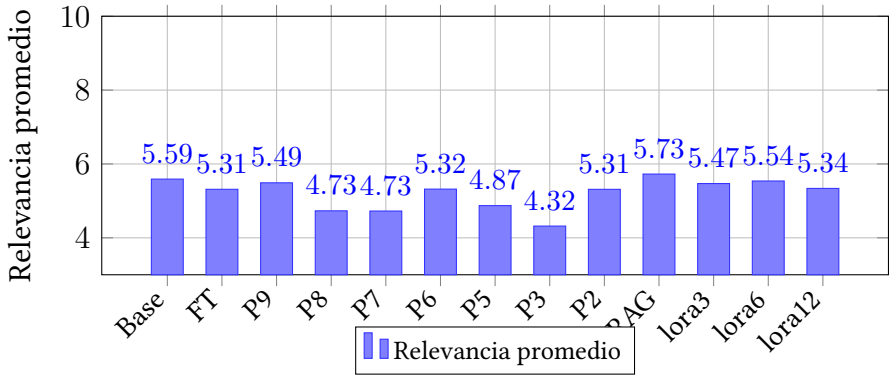

Figura 41: Comparativa de la métrica de Relevancia promedio para todos los tipos de entrenamiento aplicados

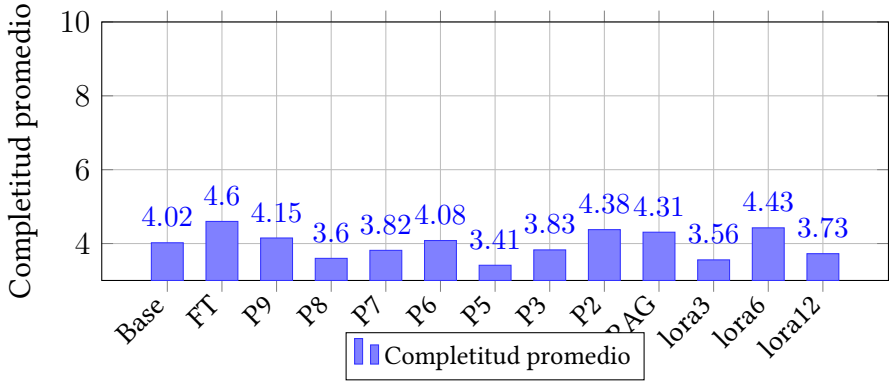

Figura 42: Comparativa de la métrica de Completitud promedio para todos los tipos de entrenamiento aplicados

Destacan:

- **Fluidez:** el modelo base y LoRA-3 ocupan el nivel más alto, seguidos por RAG y fine-tuning completo. Los modelos parciales y LoRA con más épocas manifiestan descensos progresivos.
- **Precisión:** RAG lidera, seguido de cerca por P90 y LoRA-6, con fine-tuning completo en posición intermedia. El modelo base y LoRA-3 muestran valores más bajos, indicando que la simple fluidez no garantiza precisión.
- **Coherencia:** similar a la fluidez, el base y RAG presentan los mayores valores, mientras que los parciales y LoRA-12 se sitúan en niveles inferiores.
- **Relevancia:** RAG y LoRA-6 muestran picos relativos, lo que indica que estas estrategias permiten abordar mejor el núcleo de la pregunta. Los parciales extremos y LoRA-12 tienden a menores puntuaciones.
- **Completitud:** Fine-tuning completo y RAG mejoran la cobertura de contenidos respecto al base, mientras que los modelos parciales con mayor congelamiento o LoRA-12 presentan valores más reducidos.

5.3.3 Evaluación de todos los modelos

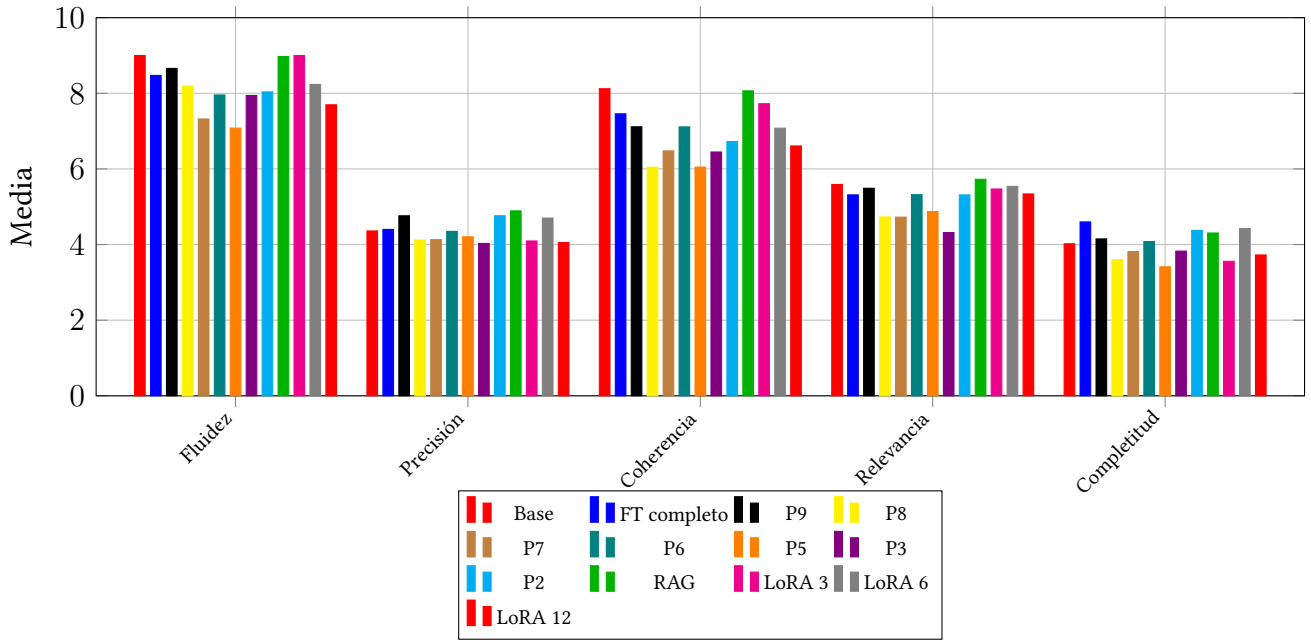

Figura 43: Comparativa de la métrica promedio para todos los modelos.

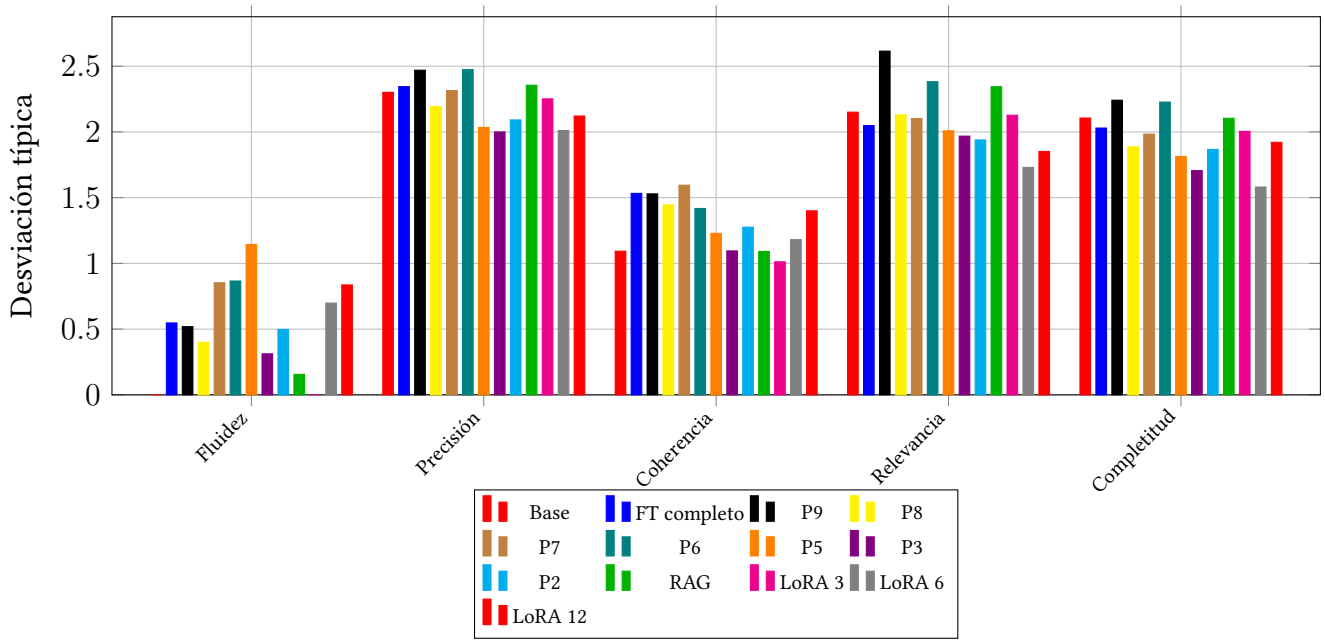

Figura 44: Comparativa de la desviación típica para todos los modelos.

La figura 43 condensa el promedio de cada métrica para todos los modelos, evidenciando el intercambio entre fluidez y precisión/completitud según la estrategia de entrenamiento. La

figura 44 muestra la variabilidad interna: modelos como base y RAG presentan desviaciones típicas relativamente bajas en fluidez y coherencia, mientras que los parciales y ciertas variantes LoRA exhiben mayor dispersión, reflejando respuestas menos uniformes.

5.3.4 Evaluaciones específicas

Se incluyen además comparativas focalizadas:

- **Modelos parciales vs. base vs. fine-tuning completo**

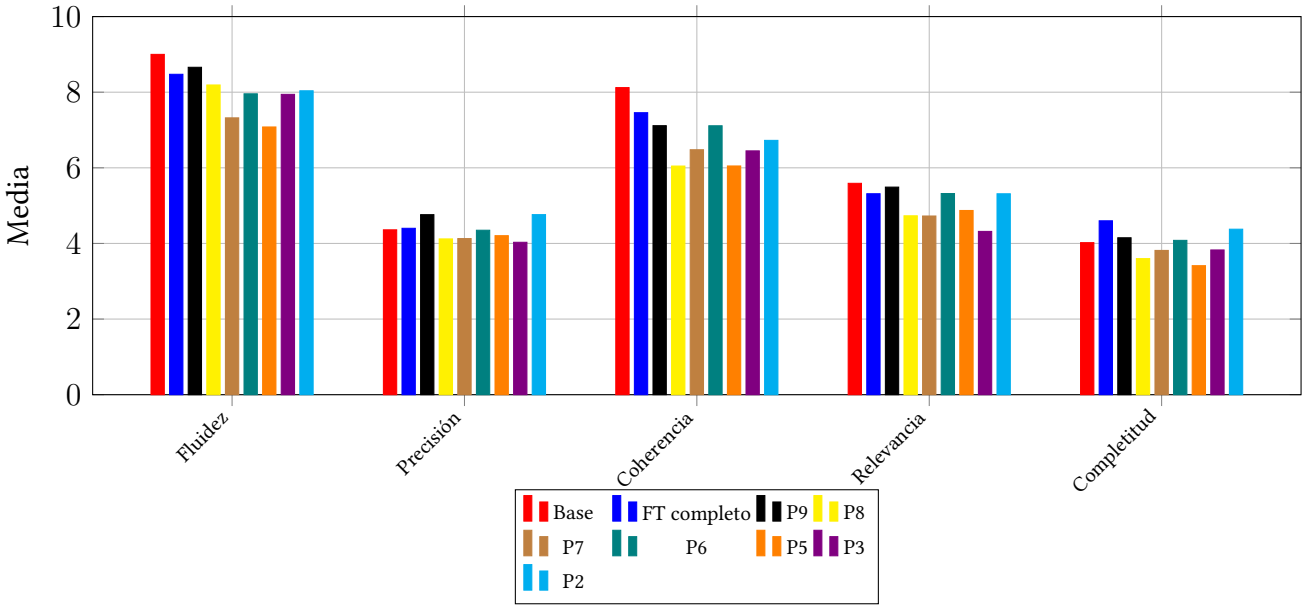

Figura 45: Comparativa de la métrica promedio para los modelos parciales, completo y base.

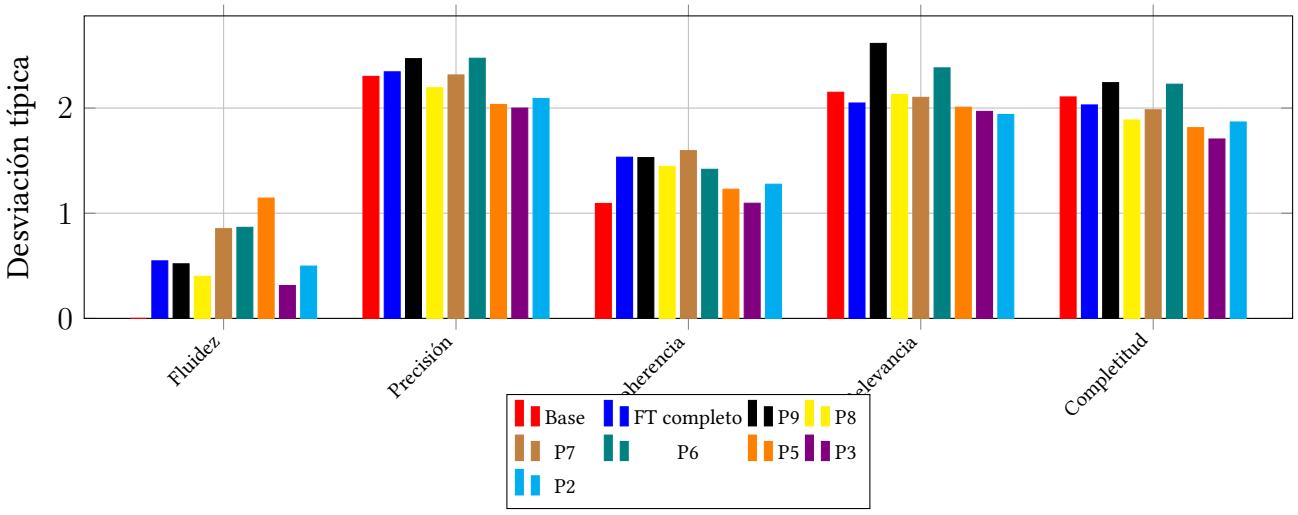

Figura 46: Comparativa de la desviación típica para los modelos parciales, completo y base.

(figuras 45, 46): estas gráficas ilustran cómo el congelamiento parcial afecta todas las métricas respecto al base y al fine-tuning completo, confirmando que P90 ofrece un equilibrio aceptable, mientras que porcentajes menores provocan deterioros más acusados en coherencia y completitud.

- **Modelos LoRA, RAG y base**

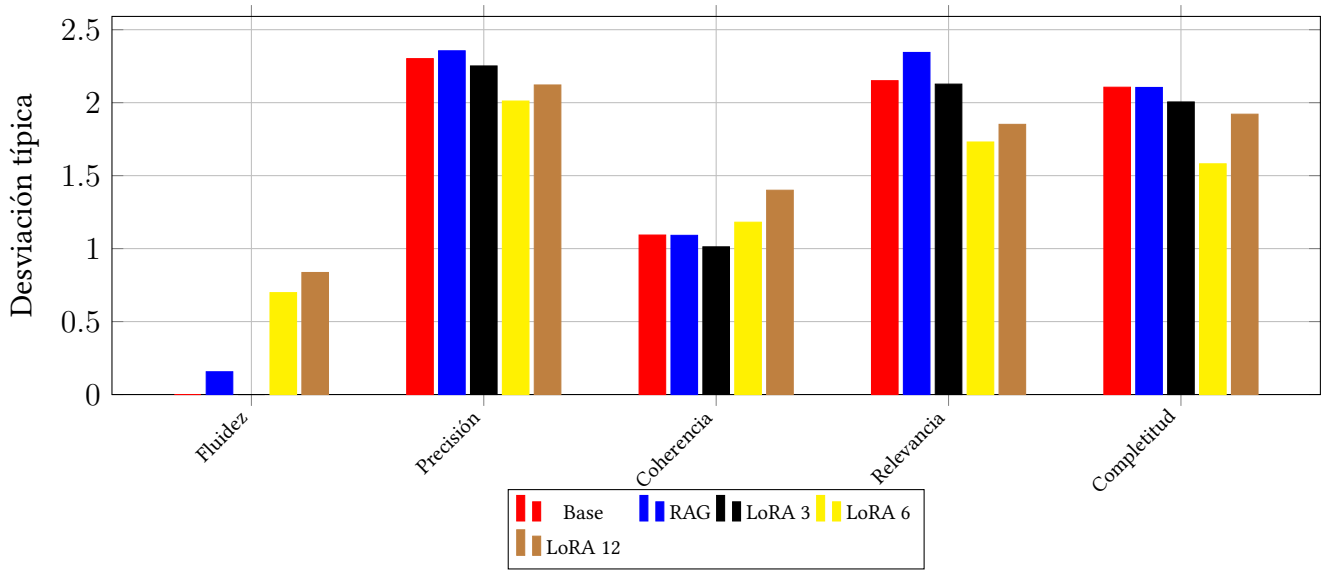

Figura 47: Comparativa de la desviación típica para los modelos LoRA, RAG y base.

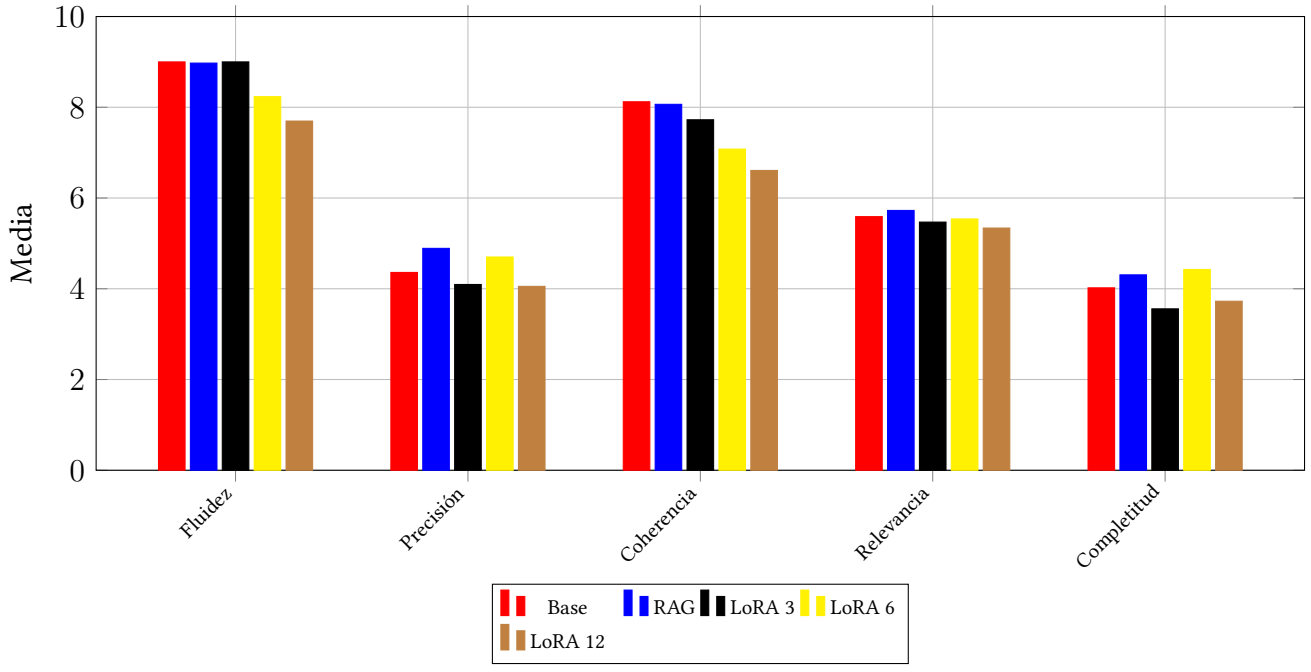

Figura 48: Comparativa de la métrica promedio para los modelos LoRA, RAG y base.

(figuras 47, 48): aquí se observa que la estrategia RAG supera o iguala al base en la mayoría de métricas clave, mientras que LoRA-3 conserva fluidez pero pierde exactitud y cobertura, y LoRA-6 representa un punto intermedio con mejora en precisión/relevancia a costa de leve sacrificio en fluidez. LoRA-12, en cambio, tiende a sobreajuste, reflejándose en caídas en varias métricas.

5.4 Discusión

En este proyecto se ha llevado a cabo una evaluación comparativa exhaustiva de diversas estrategias de especialización de un modelo LLM en un dominio técnico concreto, mediante un conjunto de datos de pares pregunta-respuesta generado y procesado específicamente para el ámbito de comunicaciones por satélite. Las estrategias incluyeron *fine-tuning* completo, afinamiento parcial en distintos porcentajes de congelamiento (desde un 10 % hasta un 90 %), adaptaciones LoRA con diferentes números de épocas (3, 6 y 12), y la incorporación de recuperación de información externa mediante RAG. La comparación se basó en métricas de fluidez, precisión , coherencia, relevancia respecto a la pregunta y completitud de la respuesta, evaluadas automáticamente mediante instancias de ChatGPT (GPT o3 y GPT 4 mini) sobre un muestreo inicialmente aleatorio de 20 tuplas, que se usó en todos los modelos.

Los hallazgos fundamentales pueden sintetizarse en:

Valor de la fluidez y la calidad lingüística El modelo base y la adaptación LoRA con un ajuste breve (3 épocas) mantuvieron consistentemente la fluidez más alta (media cercana a 9 sobre 10, con desviaciones prácticamente nulas), lo que evidencia que sin alterar en exceso los parámetros originales del modelo pre-entrenado se preserva su capacidad de generación de texto natural y elegante. Asimismo, la estrategia RAG, a pesar de incorporar fragmentos contextuales externos, apenas penalizó la fluidez (media 8.97, desviación reducida), demostrando que enriquecer el contexto no compromete la calidad estilística. Por el contrario, tanto ajustes parciales excesivos como afinamientos prolongados (por ejemplo, LoRA-12 épocas o congelamientos muy bajos) introdujeron descensos perceptibles en la fluidez y aumentaron la variabilidad en la forma de redactar, sugiriendo que alterar en exceso los parámetros del modelo puede desestabilizar su estilo lingüístico. Se sospecha que este descenso se debe a carencias en el conjunto de datos. Esta preservación de la fluidez resulta especialmente relevante

en aplicaciones donde la experiencia de lectura es prioritaria (agentes conversacionales generales, explicaciones divulgativas), pues garantiza respuestas homogéneas y sin incoherencias formales.

Mejora de la precisión factual mediante recuperación externa y ajustes moderados La métrica de precisión mostró una clara ganancia cuando se incorporó RAG (media 4.89 vs. 4.36 del modelo base) y, en menor medida, con *fine-tuning* completo (4.40) o afinamiento parcial alto (P90, 4.76). Esta tendencia apoya la hipótesis de que la recuperación de información actualizada y específica del dominio refuerza la exactitud factual de las respuestas, siempre que las fuentes recuperadas sean relevantes y de calidad. Ajustes moderados de parámetros, bien mediante *fine-tuning* parcial cuidadoso o LoRA en un número controlado de épocas (por ejemplo, 6 épocas mostró un pico en precisión 4.70 en este muestreo), también pueden mejorar la exactitud, aunque con mayor variabilidad. Sin embargo, al prolongarse demasiado el ajuste (LoRA-12), o al congelar muy pocos parámetros sin un control adecuado, se observó un retroceso en precisión, posiblemente por sobreajuste o pérdida de la capacidad generalista del modelo. Esto indica que para maximizar la precisión factual conviene combinar mecanismos de recuperación externa con ajustes moderados, evitando sobreentrenar el modelo o modificarlo drásticamente sin validaciones adicionales.

Coherencia interna como indicador de adaptación equilibrada Los resultados de coherencia (media 8.12 para el modelo base, 8.07 para RAG) sugieren que tanto el modelo sin ajuste como la estrategia de recuperación externa mantienen respuestas estructuralmente consistentes y bien organizadas. En cambio, *fine-tuning* completo y ajustes parciales menos agresivos introdujeron reducciones moderadas en coherencia (p. ej., 7.46 en *fine-tuning* completo, 7.11 en P90), y estrategias más agresivas o mal calibradas (LoRA-12, porcentajes de congelamiento muy bajos) provocaron descensos más acusados (valores en torno a 6.6˘6.7), que reflejan cierta fragmentación o contradicciones internas. La coherencia actúa así como un barómetro de la salud discursiva, así, preservar gran parte de la estructura original del modelo y añadir contexto relevante de forma controlada favorece la consistencia, mientras que cambios desmedidos en los parámetros requieren escrutinio adicional para evitar respuestas inconexas o redundantes.

Relevancia y completitud: equilibrio entre focalización y cobertura La evaluación de relevancia mostró que RAG y LoRA-6 épocas tendieron a mejorar ligeramente la adecuación al núcleo de la pregunta (medias cercanas a 5.7 y 5.5, respectivamente), superando al modelo base (5.59) y al *fine-tuning* completo (5.31) en el muestreo. Ajustes parciales extremos (P80, P70) o LoRA-3/LoRA-12 no evidenciaron mejoras significativas y presentaron desviaciones altas, lo que revela que la focalización adecuada exige un nivel de adaptación equilibrado. En cuanto a completitud, **fine-tuning** completo obtuvo la ganancia más clara (4.60 vs. 4.02 del base), seguido de LoRA-6 (4.43) y RAG (4.31). Ajustes parciales en torno a P90 mostraron un ligero beneficio (4.15), mientras que porcentajes de congelamiento menores y LoRA-3 o LoRA-12 tendieron a respuestas menos exhaustivas. Esto sugiere que, para maximizar la cobertura de aspectos esenciales, es recomendable un *fine-tuning* global o adaptaciones moderadas (LoRA intermedio) o la adición de contexto externo, evitando tanto subajuste (pocas épocas o congelamiento excesivo) como sobreajuste (épocas excesivas o ajuste sin validación).

Variabilidad y robustez estadística Las desviaciones típicas relativamente elevadas en todas las métricas (con frecuencias de α entre 1.0 y 2.5 en escala $1\lor10$) indican que, incluso en un muestreo de 20 tuplas, la variabilidad de los resultados es considerable, condicionada tanto por la heterogeneidad de las preguntas como por las limitaciones del evaluador automático. Esto limita la fiabilidad de diferencias de medias pequeñas y exige cautela en la interpretación: los patrones observados manifiestan tendencias claras (por ejemplo, RAG mejora precisión y relevancia; ajustes moderados de LoRA pueden ofrecer compromisos útiles), pero la magnitud exacta de estas mejoras o degradaciones requiere confirmación mediante muestras de evaluación más amplias y validación humana. La alta dispersión también pone de relieve la dependencia del contenido específico de cada pregunta y la necesidad de contrafactuales o análisis estratificados por tipo de pregunta.

Limitaciones y consideraciones metodológicas El uso de un evaluador LLM como juez proporcionó agilidad y permitó evaluar múltiples configuraciones en tiempo limitado, pero introduce incertidumbres asociadas a la consistencia de las valoraciones automáticas, especialmente en métricas subjetivas como coherencia y completitud. El reducido tamaño de la muestra (20 tuplas) se vio condicionado por restricciones de tiempo e interacción manual con

un modelo online. Asimismo, la selección aleatoria, aunque estadísticamente válida, podría beneficiar o penalizar ciertas estrategias dependiendo de la distribución de dificultad o temática en el subconjunto. Además, el haber usado un conjunto de datos generado sintéticamente por un modelo LLM, sin previa supervisión, ni revisión importante por parte de humanos, puede haber introducido ruido o frases incoherentes. Por tanto, las conclusiones, aunque orientativas y alineadas con la literatura, deben interpretarse considerando estas limitaciones y complementarse con evaluación humana y análisis más extensos en trabajos futuros.

Contribución y valor práctico A pesar de las restricciones, este estudio aporta una visión integrada sobre el comportamiento de diferentes técnicas de adaptación de LLM en un dominio técnico especializado en un caso común, siendo este el de usar un conjunto de datos sintético sin revisión humana, un caso perfectamente común en ámbitos profesionales y académicos, proporcionando en este caso criterios concretos para seleccionar estrategias según objetivos dentro de este contexto: mantener fluidez o maximizar precisión y cobertura, gestionar recursos computacionales o incorporar contexto actualizado. En particular, la constatación de que RAG equilibra favorablemente fluidez y precisión, y que adaptaciones LoRA intermedias o *fine-tuning* parcial alto (P90) ofrecen compromisos útiles con menor coste computacional, puede orientar implementaciones reales en entornos con limitaciones de computacionales o temporales. Asimismo, la variabilidad de los resultados subraya la importancia de diseñar pipelines de evaluación robustos y estratificados, integrando valoración automática y manual, resultados coherentes con lo visto en la literatura.

En conjunto, las conclusiones destacan que no existe una única estrategia óptima universal, sino que la elección adecuada depende del caso de uso, para tareas donde la naturalidad del lenguaje y la experiencia de usuario son primordiales, conviene preservar la fluidez del modelo base o mediante ajustes muy leves, para aplicaciones en las que la exactitud factual y la cobertura exhaustiva son críticas, conviene combinar mecanismos de recuperación externa (RAG) con ajustes moderados o fine-tuning completo cuando los recursos lo permitan, y en escenarios con restricción de tiempo o cómputo, adaptaciones parciales cuidadosas (P90) o LoRA con número moderado de épocas brindan un punto intermedio razonable. Estas orientaciones, cimentadas en datos empíricos y alineadas con la literatura previa, aportan una guía práctica y fundamentada para proyectos de especialización de LLMs en disciplinas específicas.

6

Conclusiones

A nivel de objetivos alcanzados, este trabajo ha permitido:

- **Profundizar en el estado del arte sobre el reentrenamiento de modelos LLM en contextos disciplinares específicos.** A lo largo del desarrollo del proyecto se ha realizado un estudio exhaustivo de la literatura científica actual, lo que no solo ha consolidado conocimientos técnicos relevantes, sino que también ha supuesto la primera toma de contacto del autor con la investigación académica. Este proceso ha facilitado la adquisición de competencias en la lectura crítica de artículos científicos, comprensión estructural de publicaciones, y familiarización con metodologías propias del ámbito académico, sentando así una base sólida para una futura carrera investigadora.

- **Analizar, comparar y seleccionar modelos LLM apropiados para tareas diferenciadas.** El diseño del sistema requirió la evaluación de múltiples alternativas, con especial atención a la relación entre capacidad del modelo y coste computacional. Se optó por una arquitectura dual: un modelo de menor tamaño, reentrenado para desempeñar el rol de agente, y otro de mayor capacidad dedicado a la generación sintética de datos (tuplas de pregunta-respuesta). Esta decisión respondió tanto a criterios técnicos como pragmáticos, priorizando eficiencia computacional en una etapa y cobertura semántica en otra.

- **Desarrollar e implementar el proceso completo de entrenamiento, conversión y evaluación de modelos.** Se han escrito y ejecutado los scripts necesarios para el reentrenamiento de modelos, muchos de los cuales requirieron largas jornadas de ejecución (superiores a diez horas). Posteriormente, los modelos fueron convertidos a formatos compatibles con bibliotecas estándar como `transformers` y evaluados cuantitativamente con base en métricas definidas. Los resultados obtenidos, si bien positivos y

funcionales, no fueron los cambios drásticos que el autor esperaba, lo que se atribuye en parte a la calidad del conjunto de datos utilizado, que no pasó por una revisión manual exhaustiva.

- **Realizar una evaluación comparativa de métodos actuales de reentrenamiento.** Se diseñó y ejecutó un experimento riguroso comparando 13 configuraciones distintas: desde el *fine-tuning* completo, pasando por siete variantes de entrenamiento parcial con distintas capas congeladas, hasta tres configuraciones de LoRA diferenciadas por número de épocas. Además, se compararon estos métodos con una versión base del modelo y una versión mejorada mediante RAG con FAISS. La evaluación incluyó análisis cuantitativo con dos LLM actuando como jueces, lo que ha permitido validar empíricamente el impacto de cada técnica sobre el rendimiento del modelo.

A nivel personal y profesional, este proyecto ha supuesto:

- **El inicio de una vocación investigadora.** La realización de este trabajo ha sido determinante para orientar el perfil profesional del autor hacia la investigación en inteligencia artificial. El contacto directo con bibliografía especializada, metodologías experimentales y problemáticas reales de la disciplina ha despertado un fuerte interés por seguir desarrollándose en este ámbito. Como consecuencia directa de este proyecto, el autor tiene la intención de continuar su formación investigadora.

- **El afrontamiento de problemas técnicos complejos y el fortalecimiento de la capacidad de resolución.** Durante el desarrollo del proyecto se presentaron diversos retos significativos, como la configuración de entrenamientos de alta carga computacional o la afinación de optimizadores, concretamente para el *fine-tuning* completo y parcial. También resultó especialmente desafiante la creación del pipeline automático para la generación de tuplas, que exigió múltiples iteraciones hasta cumplir con las restricciones planteadas. A pesar de su complejidad, estos obstáculos fueron superados con éxito, reforzando las competencias técnicas y la resiliencia del autor.

- **El descubrimiento de nuevas áreas de interés y disfrute personal.** El trabajo con modelos LLM, aunque inicialmente percibido como denso y técnicamente exigente, resultó ser profundamente gratificante cuando se alcanzaron resultados satisfactorios. De

igual forma, el diseño de pipelines de procesamiento de datos, y su integración con herramientas como Excel para visualización y análisis, abrió nuevas perspectivas sobre la importancia de la ingeniería de datos, un área que el autor no esperaba disfrutar tanto y que ha pasado a formar parte de sus intereses profesionales.

En definitiva, este Trabajo de Fin de Grado ha supuesto no solo la adquisición y consolidación de conocimientos técnicos avanzados en inteligencia artificial, sino también una experiencia transformadora a nivel académico y personal. Ha confirmado el interés del autor por la investigación aplicada, reforzado su vocación profesional en el ámbito de la IA, y proporcionado un espacio de descubrimiento en áreas como la evaluación de modelos y la automatización de procesos. Y empujando al autor a continuar su carrera por esta línea.

6.1 Líneas futuras

La presente investigación establece un punto de partida sólido para profundizar en la especialización de LLMs mediante enfoques más rigurosos y variados, orientados a reforzar la validez de los hallazgos y explorar nuevas vías de mejora. En primer lugar, se recomienda:

Ampliar la fase de evaluación: con un muestreo más extenso y estratificado, lo que implica revisar anticipadamente las preguntas, definir criterios de agrupación según temáticas o subdominios específicos del campo de comunicaciones por satélite, y asegurar que el subconjunto seleccionado cubra adecuadamente la diversidad de complejidad y contenido del corpus. Un muestreo así diseñado permitiría aplicar análisis estadísticos más robustos y comprobar la consistencia de las tendencias observadas en distintos segmentos de preguntas.

Incorporar evaluaciones anotadas por evaluadores humanos: lo que resulta esencial para calibrar y validar las métricas automáticas. Un protocolo de validación cruzada, en el que anotadores expertos revisen una muestra representativa de respuestas generadas y comparen sus juicios con los del LLM evaluador, ayudará a identificar sesgos o discrepancias en la interpretación de criterios como coherencia y completitud, así como a ajustar prompts y escalas de valoración hacia una mayor fidelidad a criterios humanos.

Diseño de prompts evaluativos: conviene iterar varias veces sobre su formulación, explorando variantes que incluyan meta-evaluación, es decir, solicitar al LLM no solo la puntuación numérica para cada métrica, sino también una breve justificación de dicho valor. Esto puede mejorar la transparencia del proceso y facilitar la detección de malentendidos o ambigüedades en las instrucciones.

Incorporar métricas automáticas: complementarias y benchmarks especializados como MMLU[9] y GLUE[30], para detectar cambios en el conocimiento inherente del modelo y fenómenos como el olvido catastrófico, por ejemplo, comparar la similitud de *embeddings* entre la respuesta generada y la verdad base, o emplear pruebas automatizadas que midan la pérdida de conocimiento adquirido a lo largo de diferentes fases de entrenamiento.

Exploración detallada distintas estrategias de adaptación de LLM: como combinar técnicas como LoRA y RAG para aprovechar sinergias, emplear congelamiento parcial en capas específicas identificadas como críticas mediante estudios de interpretación de parámetros, hacer una evaluación sobre el pre-entrenamiento, experimentar con *instruction tuning* dedicado. Este último resulta especialmente relevante ante la posibilidad de que las puntuaciones bajas en ciertas métricas se deban a insuficiente alineación del modelo con la tarea. Por lo que profundizar en *instruction tuning* podría mejorar la capacidad del modelo para seguir pautas de respuesta más exigentes.

Evaluar la especialización en otros dominios: técnicos o transversales, comparando patrones de comportamiento y replicabilidad de resultados, esto permitirá verificar si las conclusiones obtenidas se generalizan o requieren ajustes según la naturaleza de cada área del conocimiento. Permitiendo generar de esto un sistema fundamentado de especialización de agentes basados en modelos LLM.

Desde el punto de vista operacional:

Automatizar el pipeline de evaluación: proporcionará mayor eficiencia y reproducibilidad. Un sistema automatizado que gestione la selección estratificada de ejemplos, el envío de prompts a la API o al modelo local (incluyendo, en caso de usar Mistral u otros LLMs, introducir identificadores únicos en el prompt para evitar retransmisiones innecesarias de contexto) y

la recolección de resultados en formatos estructurados, permitirá iterar con rapidez sobre nuevas configuraciones y reducir el error manual. En particular, como se ha visto en el desarrollo de este proyecto al reevaluar con Mistral (u otros modelos locales) con un prompt optimizado, incluir un identificador de tupla evitaría reenvíos completos de texto en cada consulta, reduciendo tiempos de ejecución y mejorando la fiabilidad de la respuesta.

Automatización de la generación de tuplas de pregunta-respuesta: de igual manera que el anterior, un sistema automatizado, ocasionaría mayor eficiencia y reproducibilidad. Además de ahorro de tiempo considerable por parte del usuario. La gestión automática del envío de prompts a la API o modelo local, reduciendo la cantidad de contexto transmitido, usando identificadores únicos, automatizando la recolección de resultados, su verificación y procesado, proporcionando de manera directa el conjunto de datos de forma estructurada, reduciendo el error manual y aumentando la facilidad al generar diferentes tipos de estructuras o resultados.

Usar modelos cuantizados: sería una buena ampliación a investigar y analizar cómo la pérdida de precisión numérica afecta a las métricas de fluidez, precisión factual, coherencia y completitud. Este estudio contribuiría a determinar si es factible desplegar versiones más ligeras del modelo sin sacrificar excesivamente la calidad de las respuestas, lo que es relevante en escenarios con restricciones de hardware o latencia.

Ampliar los experimentos: comprobando el efecto en extender los experimentos de *fine-tuning* usando un mayor número de épocas, evaluando en qué medida ajustes prolongados, ya sea con LoRA o *fine-tuning* completo, ofrecen mejoras o inducen sobreajuste, y cómo equilibrar óptimamente la duración de entrenamiento con la calidad y robustez de los resultados.

Uso de herramientas: hacer la transición de un agente LLM conversacional a un agente inteligente capaz de usar herramientas externas de manera autónoma. En este sentido, se investigaría cómo dotar al sistema de interfaces para invocar buscadores especializados, APIs de bases de datos, módulos de cálculo o bibliotecas de análisis de datos en tiempo real, de modo que el modelo no solo genere texto, sino que también pueda ejecutar acciones concretas para enriquecer o verificar la información. Esta capacidad de orquestar herramientas permitiría,

por ejemplo, comprobar actualizaciones en fuentes externas, realizar cálculos precisos, llamar a entornos de simulación o extraer datos de repositorios especializados, elevando sustancialmente su potencia y fiabilidad. En la práctica, esto implicaría diseñar mecanismos seguros de invocación de herramientas (por ejemplo, a través de prompts estructurados o de un sistema de control de permisos) y evaluar el impacto en métricas de precisión factual, coherencia y completitud. Asimismo, se analizarían los requisitos de infraestructura y las implicaciones de seguridad y gobernanza de permitir al agente acceder a recursos externos, garantizando que las respuestas sigan siendo reproducibles y auditables. De esta manera, se avanzaría hacia agentes LLM más sofisticados, capaces de combinar generación de texto con ejecución de tareas especializadas, lo que abre nuevas perspectivas en aplicaciones técnicas avanzadas y entornos de toma de decisiones asistida.

Agradecimientos

El autor agradece sinceramente los recursos informáticos (superordenador Picasso), la experiencia técnica y la asistencia proporcionados por el centro SCBI (Supercomputación y Bioinformática) de la Universidad de Málaga.

Referencias

[1] Samuel Lima Braz. Peft: Parameter-efficient fine-tuning methods for llms. https://huggingface.co/blog/samuellimabraz/peft-methods, 2025.

[2] Nhat Bui, Giang Nguyen, Nguyen Nguyen, Bao Vo, Luan Vo, Tom Huynh, Arthur Tang, Van Nhiem Tran, Tuyen Huynh, Huy Quang Nguyen, and Minh Dinh. Fine-tuning large language models for improved health communication in low-resource languages. *Computer Methods and Programs in Biomedicine*, 263:108655, 5 2025.

[3] Zhou Chen, Ming Lin, Zimeng Wang, Mingrun Zang, and Yuqi Bai. Preparedllm: effective pre -pretraining framework for domain-specific large language models. *Big Earth Data*, 8:649–672, 10 2024.

[4] Biblioteca Universitaria de la Universidad de Málaga. Jábega: Catálogo de la biblioteca universitaria de la universidad de málaga. https://jabega.uma.es/.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, Mar 2022.

[7] Dimitrios Doumanas, Andreas Soularidis, Dimitris Spiliotopoulos, Costas Vassilakis, and Konstantinos Kotis. Fine-tuning large language models for ontology engineering: A comparative analysis of gpt-4 and mistral. *Applied Sciences*, 15:2146, 2 2025.

[8] Prompt Engineering Guide. Chain-of-thought prompting. https://www.promptingguide.ai/techniques/cot, 2025.

[9] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[10] Edward Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. Technical Report arXiv:2106.09685, Microsoft Research, 2021.

[11] Shafquat Hussain, Omid Ameri Sianaki, and Nedal Ababneh. *A Survey on Conversational Agents/Chatbots Classification and Design Techniques*, volume 927, pages 946–956. Springer, 2019.

[12] Intel Corporation. Fine-Tune Llama 2 70B Model with DeepSpeed ZeRO-3 and LoRA* on Intel® Gaudi® 2 Accelerators. `https://www.intel.com/content/www/us/en/developer/articles/llm/fine-tuning-llama2-70b-and-lora-on-gaudi2.html`.

[13] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[14] Akshay Kulkarni, Adarsha Shivananda, Anoosh Kulkarni, and Dilip Gudivada. *Applied Generative AI for Beginners: Practical Knowledge on Diffusion Models, ChatGPT, and Other LLMs.* Professional and Applied Computing. Apress, Berkeley, CA, 1 edition, November 2023.

[15] Ming Lin, Meng Jin, Jian Li, and Yanhua Bai. GEOSatDB: Global civil earth observation satellite semantic database. *Big Earth Data*, 8(3):522–539, 2024.

[16] Wei Lu, Rachel K. Luu, and Markus J. Buehler. Fine-tuning large language models for domain adaptation: exploration of training strategies, scaling, model merging and synergistic capabilities. *npj Computational Materials*, 11:84, 3 2025.

[17] Yuren Mao, Yuhang Ge, Yijiang Fan, Wenyi Xu, Yu Mi, Zhonghao Hu, and Yunjun Gao. A survey on lora of large language models. *Frontiers of Computer Science*, 19:197605, 7 2025.

[18] Meta AI. Llama 3.1 8b instruct - hugging face, 2024.

[19] ml-ops.org. Crisp-ml(q): Cross industry standard process model for machine learning with quality assurance, 2024.

[20] Kurt Muehmel. *THE LLM MESH Efficient and Governed Generative AI With Dataiku LEARN MORE ABOUT THE LLM MESH.* O'Reilly Media, Inc., early release edition, 8 2025.

[21] David OBrien, Sumon Biswas, Sayem Mohammad Imtiaz, Rabe Abdalkareem, Emad Shihab, and Hridesh Rajan. Are prompt engineering and todo comments friends or foes? an evaluation on github copilot. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13. ACM, 4 2024.

[22] OpenAI. Chatgpt, modelo de lenguaje conversacional basado en gpt-4. `https://chat.openai.com`, 2025.

[23] Ronghao Pan, José Antonio García-Díaz, and Rafael Valencia-García. Comparing fine-tuning, zero and few-shot strategies with large language models in hate speech detection in english. *Computer Modeling in Engineering & Sciences*, 140:2849–2868, 2024.

[24] Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities. *arXiv preprint arXiv:2408.13296*, 2024.

[25] C. Pornprasit and C. Tantithamthavorn. Fine-tuning and prompt engineering for large language models-based code review automation. *Information and Software Technology*, 175, 2024.

[26] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition, 2009.

[27] Stefan Studer, Thanh Binh Bui, Christian Drescher, Alexander Hanuschkin, Ludwig Winkler, Steven Peters, and Klaus-Robert Müller. Towards CRISP-ML(Q): A machine learning process model with quality assurance methodology. *CoRR*, abs/2003.05155, 2021.

[28] The European Space Agency. Nebula. `https://nebula.esa.int/`.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[30] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[31] Dong Wang, Xiaohui Tong, Cheng Dai, Chang Guo, Ying Lei, Cheng Qiu, Haoran Li, and Yahui Sun. Voxel modeling and association of ubiquitous spatiotemporal information in natural language texts. *International Journal of Digital Earth*, 16(1):868–890, 2023.

[32] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18:186345, 12 2024.

[33] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Apéndice A
Fuentes entregados

En los ficheros fuentes entregados, se ve la siguiente estructura. De manera general estas

- **Códigos re-entreno** en esta carpeta se encuentra la siguiente estructura:
  - **Fine-tuning completo:**
  - **Fine-tuning parcial:** dentro se incluyen todas las variantes implementadas en este proyecto.
  - **LoRA:** incluyendo las tres variantes de época.
  - **conversormodelo:** este script es el que debe ser ejecutado tras entrenarlo con *fine-tuning* completo o parcial, para convertir el modelo a un formato usable por la librería transformers.
- **Evaluación:** dentro incluye las 13 carpetas con los scripts para hacer inferencia en los distintos modelos usados. Además se incluye una carpeta llamada prueba nota numerica con mistral, referente a la prueba fallida de evaluación de preguntas.
- **Generación tuplas:** aquí se contienen dos carpetas:
  - **Transformers**
  - **VLLM**

  En estas se contienen los scripts que han ejecutado la generación de tuplas, con transformers y vllm, ambos usando el mismo modelo Mistral 24B.
- **Inferencia:** esta carpeta contiene otras dos carpetas:
  - **inferencia normal** esta es la plantilla original ofrecida, adaptada para funcionar en Picasso.
  - **inferencia con lora** esta es la plantilla que usa un plugin lora para hacer inferencia, se aporta por ser la versión reducida de evaluación con lora.

- **RAG** en esta carpeta se ubican los ficheros necesarios para hacer inferencia con RAG, y el script usado para preprocesar el dataset usado para FAISS.
- **Tokenizadores** en esta carpeta están los tokenizadores usados para tokenizar el dataset para *fine-tuning* y LoRA, además se incluye el script usado para comprobar que los resultados de la generación están en formato json válido.

Además, se indica que en gran parte de las carpetas se ubicarán ficheros .sh, normalmente conteniendo en el nombre script, estos ficheros han sido usados, para cargar ambientes python, y demás valores de valores necesarios. Además se ve en las primeras 40 líneas, texto diferente, este está ahí para Picasso, que usa el sistema de colas SLURM, y requiere esos campos para determinar, entre otros, el tiempo máximo de la tarea, la cantidad de RAM necesaria, las gráficas que se piden y demás configuraciones, se incluyen pues pueden servir como base para definir variables de ambiente y entender el flujo completo del código.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA