MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

AJUSTE FINO DE UN LLM ORIENTADO AL MANEJO EFICIENTE DE INFORMACIÓN PRIVADA

Autor
Sofía Amores Parra

Director
Álvaro Jesús López López

Madrid
Julio de 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título ……..…… AJUSTE FINO DE UN LLM ORIENTADO AL MANEJO EFICIENTE DE INFORMACIÓN PRIVADA……..……

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico …2024/2025……. es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Sofía Amores Parra Fecha: 13/ 7/ 2025

AMORES PARRA SOFIA - 21007927A

Firmado digitalmente por AMORES PARRA SOFIA - 21007927A
Fecha: 2025.07.17 15:55:36 +02'00'

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Álvaro Jesús López López Fecha: 16…/ 07…/ 25…..

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

AJUSTE FINO DE UN LLM ORIENTADO AL MANEJO EFICIENTE DE INFORMACIÓN PRIVADA

Autor
Sofía Amores Parra

Director
Álvaro Jesús López López

Madrid
Julio de 2025

AJUSTE FINO DE UN LLM ORIENTADO AL MANEJO EFICIENTE DE INFORMACIÓN PRIVADA

Autor: Amores Parra, Sofía
Director: López López, Álvaro Jesús
Entidad Colaboradora: ICAI - Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

1. Introducción

En los últimos años, los modelos de lenguaje de gran tamaño (LLM, por sus siglas en inglés, *Large Language Models*) han transformado el procesamiento de lenguaje natural, aunque su despliegue sigue siendo complejo en entornos con limitaciones de hardware, privacidad o coste. Este proyecto aborda este reto mediante el ajuste fino de un LLM en local con la técnica *Low-Rank Adaptation* (LoRA), aplicada a una tarea de Preguntas y Respuestas (Q&A) sobre datos meteorológicos estructurados de estaciones españolas.

Cuando no se dispone de suficientes datos para entrenar desde cero, el aprendizaje por transferencia permite adaptar modelos preentrenados mediante ajuste fino [1]. Las técnicas de *Parameter-Efficient Fine-Tuning* (PEFT), como LoRA y QLoRA (utilizada en este trabajo), reducen el coste al actualizar solo una pequeña fracción de parámetros [1]. También destaca RAG, que recupera fragmentos relevantes mediante búsqueda semántica evitando así el reentrenamiento [1][2].

El *fine-tuning* plantea dos retos importantes: la privacidad y el despliegue en entornos locales. Por un lado, el modelo puede memorizar ejemplos del conjunto de entrenamiento, por lo que se emplean técnicas como *Differential Privacy* (que añade ruido a los gradientes) y *Data Scrubbing* (que elimina información personal identificable) [2]. Por otro, para facilitar su ejecución en dispositivos con recursos

limitados, se recurre a técnicas de compresión como la cuantización, que consiste en representar los pesos del modelo con menor precisión numérica.[3].

La evaluación tras el ajuste debe considerar tanto métricas lingüísticas (como perplejidad, factualidad e integridad) como computacionales. El estudio de Yao et al. [4] propone un *benchmark* con tres tipos de métricas: precisión de test, métricas de entrenamiento (memoria y tiempo), y métricas de inferencia (tiempo y consumo).

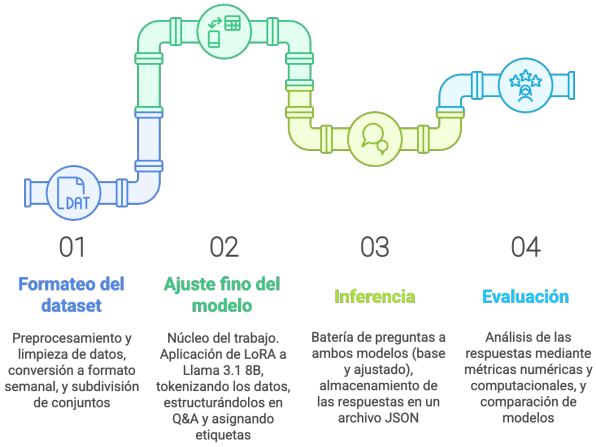
Este tipo de enfoques no solo han demostrado su eficacia en entornos de investigación, sino que también se han trasladado al ámbito empresarial con gran éxito. Actualmente, empresas como OpenAI o Databricks ofrecen plataformas integradas para entrenamiento y despliegue. También existen modelos especializados creados mediante *fine-tuning*, como FinGPT en el sector financiero o LAWGPT en el legal [5] [6].

Partiendo de este contexto, surge la motivación del proyecto: consolidar estos enfoques en una solución integral *end-to-end* que incluya preprocesamiento, entrenamiento, evaluación e inferencia. Aunque se ha trabajado con un conjunto de datos público, todo el proceso se ha tratado como si se usaran datos confidenciales, garantizando su reproducibilidad en contextos reales. Para lograr este fin, se han establecido los siguientes cuatro objetivos:

1. Aplicar técnicas de ajuste fino, como *Low-Rank Adaptation*(LoRA), para optimizar un modelo de lenguaje en entornos con restricciones de hardware.
2. Ajustar un LLM para una tarea específica de preguntas y respuestas (Q&A) sobre un corpus determinado.
3. Evaluar la eficiencia y el rendimiento del modelo antes y después del fine-tuning.
4. Identificar posibles mejoras para desarrollos de proyectos similares.

2. Metodología

La figura 3.1 resume el *workflow* del proyecto, estructurado en distintos *notebooks* de Google Colab para mantener modularidad, trazabilidad y claridad en el desarrollo. Cada bloque representa una etapa independiente, descrita en detalle en los capítulos 3 y 4.



Resumen del flujo de trabajo del proyecto y cuadernos empleados

Se ha utilizado el conjunto de datos público **GHCN (*Global Historical Climatology Network*)**, con registros diarios de estaciones meteorológicas. Tras aplicar filtrado por país, promediación semanal y submuestreo aleatorio, se obtuvieron 25 000 muestras de entrenamiento, 2 000 de validación y más de 100 000 de test, respetando la división temporal 2014–2022 (entrenamiento) y 2023–2024 (test).

El modelo seleccionado fue Llama 3.1 8B, versión *open-source* de Meta, elegido por su buen equilibrio entre tamaño y rendimiento. Se cargó cuantizado en 4 bits mediante *Unsloth* y se ajustó con la técnica LoRA, actualizando solo un 0.52 % de sus parámetros para poder completar el proceso en Google Colab con eficiencia.

La evaluación del proyecto combina métricas lingüísticas y numéricas. Se ha empleado *FactScore* y detección de alucinaciones para analizar coherencia factual, y MAE, MSE y R^2 para evaluar precisión en las respuestas numéricas generadas.

Asimismo, se ha medido el uso de recursos desde tres perspectivas:

Métricas del uso de recursos empleadas en el proyecto.

Esta triple evaluación permitió valorar no solo la calidad del modelo ajustado, sino también su eficiencia y sostenibilidad en entornos con recursos limitados.

3. Resultados

Como resumen de los hallazgos, se presentan las siguientes tablas, que recogen tanto las métricas generales del proceso de entrenamiento como los resultados en las métricas de evaluación tras el ajuste fino:

Métricas generales del entrenamiento	**Valor**
Curva de pérdidas (entrenamiento)	Valor más bajo: 0.0135
Curva de pérdidas (validación)	Valor más bajo: 0.024919
Consumo energético medio del entrenamiento	260.61 W
Máximo uso de memoria GPU	8753 MB

Resumen de resultados del entrenamiento

A pesar de que el entrenamiento no pudo completarse con todas las épocas

inicialmente previstas, las curvas de pérdida para el conjunto de entrenamiento y validación muestran una evolución paralela y sin indicios claros de sobreajuste o subajuste. Esto sugiere una convergencia adecuada del modelo.

Métricas de evaluación del modelo	**Modelo base**	**Modelo ajustado**
Tiempo promedio por respuesta	0.5909 s	1.8692 s
Consumo de memoria promedio por respuesta	0.04 MB	0.11 MB
Fact Score	43.7 %	56.2 %
Detección de alucinaciones	21.5 %	9.6 %
MAE	3.1	2.4
MSE	10.1	6.8
R^2	0.67	0.79

Resumen de comparación entre modelo base y ajustado

Los resultados numéricos confirman la mejora del modelo ajustado frente al base, con menor MAE y MSE, mayor R^2 y reducción significativa de alucinaciones. Aunque la exactitud global sigue siendo baja, debe considerarse el criterio de evaluación empleado, como se explica en el capítulo 4.

El tiempo de inferencia se triplica, pero sigue siendo aceptable en herramientas centradas en precisión y no en inmediatez. El consumo energético medio (260.61 W) y el uso de memoria GPU (8753 MB) se mantuvieron en márgenes razonables, y el bajo consumo en inferencia refuerza su viabilidad en entornos con recursos limitados.

4. Conclusiones

El trabajo ha demostrado la viabilidad del ajuste fino de LLMs en local, incluso con recursos limitados como Google Colab. Aunque el *dataset* era público, se trató

como confidencial, haciendo reproducible el proceso para otros contextos.

Pese a los retos de adaptar datos numéricos estructurados a modelos de lenguaje, el modelo entrenado ofrece mejoras estables sin sobreajuste. El flujo de trabajo modular facilita su extensión a otros dominios, y sería escalable en empresas si se complementa con una interfaz accesible. Entre las líneas futuras destacan: uso de datos ausentes, incorporación de estructura secuencial y formatos conversacionales más complejos.

Referencias del resumen

[1] Mathav Raj J et al. *Fine Tuning LLM for Enterprise: Practical Guidelines and Recommendations*. 23 de mar. de 2024. arXiv: 2404.10779. URL: http://arxiv.org/abs/2404.10779 (visitado 07-11-2024).

[2] Venkatesh Balavadhani Parthasarathy et al. *The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities*. version: 1. 23 de ago. de 2024. arXiv: 2408.13296. URL: http://arxiv.org/abs/2408.13296 (visitado 07-11-2024).

[3] Peijie Dong et al. *Can Compressed LLMs Truly Act? An Empirical Evaluation of Agentic Capabilities in LLM Compression*. 1 de jun. de 2025. DOI: 10.48550/arXiv.2505.19433. arXiv: 2505.19433[cs]. URL: http://arxiv.org/abs/2505.19433 (visitado 08-06-2025).

[4] Zhiwei Yao et al. *Efficient Deployment of Large Language Models on Resource-constrained Devices*. 5 de ene. de 2025. DOI: 10.48550/arXiv.2501.02438. arXiv: 2501.02438[cs]. URL: http://arxiv.org/abs/2501.02438 (visitado 08-06-2025).

[5] *Mosaic AI Model Training*. Databricks. URL: https://www.databricks.com/product/machine-learning/mosaic-ai-training (visitado 07-06-2025).

[6] *Fine-tuning*. OpenAI API. URL: https://platform.openai.com/docs/guides/fine-tuning (visitado 07-06-2025).

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

FINE-TUNING A LLM FOR THE EFFICIENT MANAGEMENT OF PRIVATE INFORMATION

ABSTRACT

1. Introduction

In recent years, Large Language Models (LLMs) have transformed natural language processing (NLP), although their deployment remains complex in environments with hardware, privacy, or cost constraints. This project addresses this challenge through the fine-tuning of a local LLM using the *Low-Rank Adaptation* (LoRA) technique, applied to a Question Answering (Q&A) task on structured meteorological data from Spanish stations.

When there is not enough data to train a model from scratch, transfer learning allows pre-trained models to be adapted to new tasks via fine-tuning [1]. *Parameter-Efficient Fine-Tuning* (PEFT) techniques, such as LoRA and QLoRA (used in this project), reduce costs by updating only a small fraction of the parameters [1]. Another relevant approach is RAG, which retrieves relevant fragments through semantic search and adds them to the prompt, thus avoiding retraining [1][2].

Fine-tuning poses two major challenges: privacy and deployment in local environments. On the one hand, the model may memorize training examples, so techniques such as *Differential Privacy* (which adds noise to the gradients) and *Data Scrubbing* (which removes personally identifiable information) are employed [2]. On the other hand, to facilitate execution on resource-limited devices, compression techniques such as quantization are used, which consist of representing model weights with lower numerical precision [3].

Post-tuning evaluation must consider both linguistic metrics (such as perplexity, factuality, and completeness) and computational metrics. The study by Yao

et al. [4] proposes a benchmark based on three metric categories: test accuracy, training metrics (memory and time), and inference metrics (latency and energy consumption).

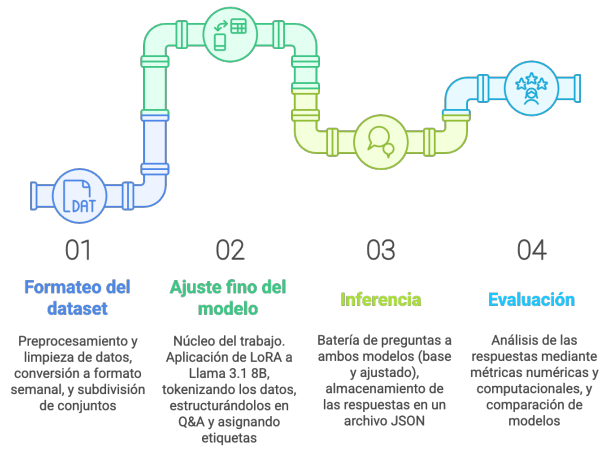
These approaches have proven effective not only in research contexts but also in the business world. Companies such as OpenAI and Databricks currently offer integrated platforms for training and deployment. There are also domain-specific models developed through fine-tuning, such as FinGPT for the financial sector or LAWGPT for legal applications [5] [6].

Based on this context, the motivation of the project is to consolidate these approaches into an integrated end-to-end solution that includes preprocessing, training, evaluation, and inference. Although a public dataset was used, the entire process was handled as if it involved confidential data, ensuring its reproducibility in real-world scenarios. To this end, the following four objectives were defined:

1. Apply fine-tuning techniques, such as *Low-Rank Adaptation* (LoRA), to optimize a language model in hardware-constrained environments.
2. Fine-tune an LLM for a specific question-answering (Q&A) task over a defined corpus.
3. Evaluate the model's efficiency and performance before and after fine-tuning.
4. Identify potential improvements for similar future projects.

2. Method

Figure 3.1 summarizes the project's workflow, structured into separate Google Colab notebooks to maintain modularity, traceability, and clarity throughout development. Each block represents an independent stage, described in detail in Chapters 3 and 4.



Summary of the project's workflow and notebooks used

The dataset used was the public **GHCN (*Global Historical Climatology Network*)**, which includes daily records from weather stations. After applying country filtering, weekly averaging, and random subsampling, the final dataset included 25,000 training samples, 2,000 for validation, and over 100,000 for testing, respecting the temporal split: 2014–2022 for training and 2023–2024 for testing.

The model selected was Llama 3.1 8B, an *open-source* version from Meta, chosen for its balance between size and performance. It was loaded quantized to 4 bits using *Unsloth* and fine-tuned with the LoRA technique, updating only 0.52 % of its parameters to enable efficient training within Google Colab.

The evaluation combined linguistic and numerical metrics. *FactScore* and hallucination detection were used to assess factual consistency, while MAE, MSE, and R^2 were used to evaluate the accuracy of the generated numerical responses. Additionally, resource usage was measured from three perspectives:

Resource usage metrics applied in the project

This threefold evaluation allowed for assessing not only the quality of the fine-tuned model but also its efficiency and sustainability in resource-constrained environments.

3. Results

As a summary of the findings, the following tables present both the general metrics of the training process and the evaluation results after fine-tuning:

General Training Metrics	Value
Training loss curve	Minimum value: 0.0135
Validation loss curve	Minimum value: 0.024919
Average training power consumption	260.61 W
Maximum GPU memory usage	8753 MB

Summary of training results

Although the training process could not be completed with all the initially planned epochs, the loss curves for both training and validation sets evolved in

parallel, with no clear signs of overfitting or underfitting. This suggests an adequate convergence of the model.

Model Evaluation Metrics	**Base Model**	**Fine-Tuned Model**
Average response time	0.5909 s	1.8692 s
Average memory usage per response	0.04 MB	0.11 MB
Fact Score	43.7 %	56.2 %
Hallucination detection	21.5 %	9.6 %
MAE	3.1	2.4
MSE	10.1	6.8
R^2	0.67	0.79

Summary comparison between base and fine-tuned model

The numerical results confirm the improvement of the fine-tuned model over the base model, with lower MAE and MSE, higher R^2, and a significant reduction in hallucinations. Although the overall accuracy remains low, this must be interpreted considering the evaluation criteria used, as explained in Chapter 4.

Inference time is approximately three times longer, but remains acceptable for tools focused on accuracy rather than immediacy. The average energy consumption (260.61 W) and GPU memory usage (8753 MB) remained within reasonable limits, and the low memory usage during inference reinforces the model's viability in resource-constrained environments.

4. Conclusions

This work has demonstrated the feasibility of fine-tuning LLMs locally, even with limited resources such as Google Colab. Although the dataset used was public, it was treated as confidential, ensuring the reproducibility of the process in other

contexts.

Despite the challenges of adapting structured numerical data to language models, the fine-tuned model showed stable improvements without overfitting. The modular workflow facilitates its extension to other domains and could be scaled for enterprise use if complemented by an accessible interface. Future lines of work include handling missing data, incorporating sequential structure, and exploring more complex conversational formats.

Abstract references

[1] Mathav Raj J et al. *Fine Tuning LLM for Enterprise: Practical Guidelines and Recommendations*. 23 de mar. de 2024. arXiv: 2404.10779. URL: http://arxiv.org/abs/2404.10779 (visitado 07-11-2024).

[2] Venkatesh Balavadhani Parthasarathy et al. *The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities*. version: 1. 23 de ago. de 2024. arXiv: 2408.13296. URL: http://arxiv.org/abs/2408.13296 (visitado 07-11-2024).

[3] Peijie Dong et al. *Can Compressed LLMs Truly Act? An Empirical Evaluation of Agentic Capabilities in LLM Compression*. 1 de jun. de 2025. DOI: 10.48550/arXiv.2505.19433. arXiv: 2505.19433[cs]. URL: http://arxiv.org/abs/2505.19433 (visitado 08-06-2025).

[4] Zhiwei Yao et al. *Efficient Deployment of Large Language Models on Resource-constrained Devices*. 5 de ene. de 2025. DOI: 10.48550/arXiv.2501.02438. arXiv: 2501.02438[cs]. URL: http://arxiv.org/abs/2501.02438 (visitado 08-06-2025).

[5] *Mosaic AI Model Training*. Databricks. URL: https://www.databricks.com/product/machine-learning/mosaic-ai-training (visitado 07-06-2025).

[6] *Fine-tuning*. OpenAI API. URL: https://platform.openai.com/docs/guides/fine-tuning (visitado 07-06-2025).

Índice general

1. Introducción **1**

1.1. Motivación del proyecto 2

1.2. Objetivos del proyecto 3

2. Estado de la cuestión **5**

2.1. Técnicas de ajuste fino para modelos 6

2.1.1. *Adapters* 7

2.1.2. LoRA (*Low-Rank Adaptation*) y QLoRA 8

2.1.3. Técnicas ligeras 9

2.2. Alternativa al ajuste fino: RAG (Retrieval-Augmented Generation) 11

2.2.1. Funcionamiento y ventajas 11

2.2.2. Limitaciones y comparativa 12

2.3. Privacidad en Fine-Tuning 14

2.3.1. *Differential Privacy* (DP) 15

2.3.2. *Data scrubbing* 16

2.3.3. Ataques de privacidad y mitigaciones 17

2.4. Métricas y evaluación del rendimiento de los LLMs ajustados 19

2.4.1. Métricas clásicas 19

2.4.2. Métricas avanzadas de uso de recursos 21

2.5. Optimización y escalabilidad para recursos limitados 24

2.5.1. Técnicas de compresión de modelos 24

2.5.2. Infraestructura y despliegue *on-premise* 26

2.6. Aplicación del ajuste fino de LLMs en el entorno empresarial 27

2.6.1. Empresas proveedoras de servicios de *fine-tuning* 28

2.6.2. Casos de éxito reales en distintos sectores 31

2.6.3. Oportunidades en dominios especializados: el caso de la climatología 32

3. Metodología **35**

3.1. Esquema general del flujo de trabajo 35

3.2. Planificación del proyecto 36

3.3. Recursos empleados 37

3.3.1. Hardware utilizado 37

3.3.2. Herramientas y librerías de software 41

3.3.3. Descripción del *dataset* y preprocesamiento de datos 41

3.4. Entrenamiento del modelo 47

3.4.1. Configuración del entorno 48

3.4.2. Configuración del modelo con LoRA 51

3.4.3. Hiperparámetros del entrenamiento 52

4. Resultados y discusión **57**

4.1. Resultados del entrenamiento 58

4.1.1. Curva de pérdidas (*Training Loss*) 58

4.1.2. Evaluación de la eficiencia computacional en el entrenamiento 61

4.2. Evaluación del modelo después del ajuste fino 63

4.2.1. Proceso de inferencia 64

4.2.2. Evaluación de la eficiencia computacional en la inferencia . . 67

4.2.3. Evaluación de la precisión numérica 68

5. Conclusiones **73**

5.1. Resumen de objetivos y hallazgos 73

5.2. Limitaciones del trabajo 75

5.3. Conclusiones finales e impacto 77

6. Trabajos Futuros **79**

6.1. Posibles mejoras en el ajuste fino 79

6.2. Ampliaciones del enfoque actual 80

6.3. Escalabilidad y aplicación en entornos empresariales 82

Bibliografía **85**

Anexo A: Contribución a los Objetivos de Desarrollo Sostenible (ODS) **91**

Índice de figuras

1. Resumen del flujo de trabajo del proyecto y cuadernos empleados III

2. Métricas del uso de recursos empleadas en el proyecto. IV

3. Summary of the project's workflow and notebooks used III

4. Resource usage metrics applied in the project IV

2.1. En el ajuste fino regular, se actualizan todos los pesos del modelo, mientras que en LoRA, se utilizan dos matrices de bajo rango (A y B) para aproximar la actualización de pesos, reduciendo el número de parámetros entrenables. Esto hace que LoRA sea más eficiente en memoria y computación, ideal para modelos grandes [2]- 9

2.2. "Operacionalización de LLMs en PYMES en entornos con recursos limitados" [8]. 12

2.3. Comparativa entre *fine-tuning* y RAG 13

2.4. Categorías de vulnerabilidades en los LLMs [12] 17

2.5. Tipos de ataques de seguridad [12] 17

2.6. Tipos de ataques de privacidad [12] 18

2.7. Métricas del uso de recursos empleadas en el proyecto. 23

2.8. Flujo de trabajo para el ajsute fino mediante la plataforma de OpenaAI [19]. 29

3.1. Resumen del flujo de trabajo del proyecto y cuadernos empleados . 36

3.2. Cronograma de Gantt del proyecto 37

3.3. Entrenamiento de IA hasta 3 veces superior en los modelos más grandes [32] 39

3.4. Tendencia en eficiencia computacional y capacidad de memoria de las GPUs NVIDIA. Cada color representa una arquitectura distinta, y el tamaño del punto refleja la memoria integrada. La línea roja marca la tendencia general de mejora en eficiencia computacional [15]. 40

3.5. Esquema de preprocesamiento del *dataset* desde el GHCN original hasta la versión final para ajuste fino. 44

3.6. Ciclo habitual de entrenamiento, validación y evaluación [35]. 45

3.7. Distribuciones de las etiquetas de los datos 51

4.1. Resumen de métricas utilizadas en las fases de entrenamiento y evaluación. 58

4.2. Evaluación de la pérdida en el conjunto de entrenamiento. 60

4.3. Evaluación de la pérdida en el conjunto de entrenamiento y validación. 60

4.4. Uso de la memoria GPU durante el entrenamiento. 63

Anexo A. Alineación con los ODS 92

Índice de cuadros

3.1. Resumen de las distintas versiones del dataset utilizadas en el proyecto. 44

3.2. Parámetros utilizados en la configuración de LoRA 55

4.1. Consumos energéticos del entrenamiento. 62

4.2. Tiempos de inferencia. 67

4.3. Consumos de memoria en inferencia. 68

4.4. Comparación de métricas entre el modelo base y el modelo ajustado 70

5.1. Resumen de resultados del entrenamiento y comparación entre modelo base y ajustado 74

Capítulo 1

Introducción

En los últimos años, los modelos de lenguaje de gran tamaño (LLM, por sus siglas en inglés, *Large Language Model*) han revolucionado el procesamiento de lenguaje natural (NLP, por sus siglas en inglés *Natural Language Processing*), mejorando tareas como la generación de texto y la clasificación de datos. Su capacidad para comprender y generar lenguaje humano los ha convertido en una herramienta esencial en múltiples aplicaciones tecnológicas.

Sin embargo, el potencial de estos modelos para aplicaciones especializadas en entornos con restricciones de hardware aún presenta desafíos significativos. Estos modelos requieren habitualmente una gran cantidad de recursos computacionales y datos para ofrecer buenos resultados, lo que limita su uso en contextos donde la privacidad, el coste o la eficiencia energética son factores clave.

Este trabajo aborda esta problemática mediante la implementación de técnicas de ajuste fino, como *Low-Rank Adaptation* (LoRA), con el objetivo de optimizar un LLM para la tarea específica de Preguntas y Respuestas (Q&A) sobre un dominio numérico concreto: datos meteorológicos de estaciones españolas. A diferencia de otros enfoques, el proyecto se ha desarrollado de forma integral, cubriendo todas las etapas del proceso, desde la preparación del *dataset* hasta la evaluación final

del modelo.

Una de las principales particularidades del proyecto es la adaptación de un conjunto de datos numéricos estructurados (provenientes de un repositorio de Amazon Web Services) al formato requerido por modelos de lenguaje autoregresivos, que procesan el texto de manera secuencial. Este reto ha requerido un trabajo exhaustivo de limpieza, conversión de unidades, formateo semántico y escalado, con el fin de que el modelo pueda aprender eficazmente a generar respuestas numéricas coherentes.

Además, se ha diseñado un sistema de evaluación doble: por un lado, midiendo el rendimiento del modelo antes y después del *fine-tuning*; y por otro, analizando el uso de recursos computacionales como la memoria, el tiempo de inferencia y el consumo energético. De esta manera, se busca no solo mejorar la capacidad del modelo, sino también garantizar su eficiencia en un entorno accesible.

La modularidad del enfoque adoptado, junto con el uso de plataformas estándares como HuggingFace, permite garantizar la reproducibilidad y escalabilidad del proyecto. Este diseño facilitará su integración en futuras publicaciones o trabajos relacionados.

Finalmente, esta memoria se estructura en seis capítulos que presentan los objetivos del proyecto, contextualizan su desarrollo, detallan la metodología empleada, analizan los resultados obtenidos y extraen conclusiones relevantes para la comunidad investigadora y para posibles aplicaciones futuras.

1.1. Motivación del proyecto

La motivación principal de este trabajo surge de la creciente necesidad de disponer de soluciones basadas en modelos de lenguaje que sean personalizables, eficientes y que preserven la privacidad de datos. Aunque los LLMs han demostrado

tener capacidades destacables en tareas generales, su aplicación en contextos específicos requiere de procesos de adaptación que resulten accesibles para usuarios sin disponibilidad a grandes infraestructuras.

El uso de técnicas como LoRA permite ajustar modelos existentes reduciendo considerablemente los parámetros entrenables para optimizar el consumo de memoria, lo que abre la puerta a su uso en entornos *on-premise*. Esto resulta especialmente útil para pequeñas empresas o usuarios independientes que deseen desarrollar soluciones especializadas sin comprometer la privacidad de los datos empleados.

Además, al tratar con un *dataset* numérico que no ha sido ideado originalmente para modelos de lenguaje, el trabajo plantea retos únicos en cuanto al preprocesamiento y la configuración del entrenamiento. Esta adaptación ha permitido explorar los límites de los LLMs y comprobar su potencial más allá del texto libre, en aplicaciones que requieren respuestas exactas.

La integración de todos estos elementos en un único proyecto *end-to-end*, acompañado de una evaluación tanto de calidad de las respuestas como de eficiencia computacional, convierte este trabajo en una base sólida para desarrollos posteriores, como una publicación científica en conjunto con otro Trabajo de Fin de Master.

1.2. Objetivos del proyecto

En este apartado se presentan los objetivos que se van a llevar a cabo en este proyecto, que están alineados con el propósito final de adaptar LLMs para aplicaciones locales garantizando eficiencia en el uso de recursos.

1. **Aplicar técnicas de ajuste fino, como *Low-Rank Adaptation*(LoRA),**

para optimizar un modelo de lenguaje en entornos con restricciones de hardware. Investigar y emplear métodos de ajuste fino que reduzcan los parámetros entrenables y se adapten a las capacidades de dispositivos con limitaciones de procesamiento y memoria.

2. **Ajustar un LLM para una tarea específica de preguntas y respuestas (Q&A) sobre un corpus determinado.** Aunque el entrenamiento se realizará en Google Cloud Platform (GCP) para facilitar el desarrollo del proyecto, el uso de técnicas PEFT permitirá que el modelo resultante pueda implementarse en entornos con restricciones de hardware, manteniendo un control sobre el uso de datos de manera local.

3. **Evaluar la eficiencia y el rendimiento del modelo antes y después del fine-tuning.** Establecer métricas y métodos de evaluación para comparar el rendimiento del LLM antes y después de aplicar las técnicas de ajuste fino, enfocándose en el uso de memoria, la velocidad de inferencia y la precisión en las respuestas.

4. **Identificar posibles mejoras para desarrollos de proyectos similares.** Se desarrollarán sugerencias de mejores prácticas basadas en los resultados obtenidos, con el objetivo de servir como referencia para futuros trabajos y aplicaciones en contextos semejantes.

Este proyecto también contribuye de manera significativa a los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030, especialmente en relación con los que se nombran en el anexo Anexo A.

Capítulo 2

Estado de la cuestión

En este capítulo se presenta una revisión bibliográfica que define el estado del arte en relación con los grandes modelos de lenguaje y su adaptación a contextos específicos, estableciendo así el marco conceptual del proyecto.

En primer lugar, se analizan las principales técnicas de ajuste fino, especialmente relevantes cuando no se dispone de un corpus de entrenamiento lo suficientemente amplio como para entrenar un modelo desde cero. Se detalla una de las estrategias más utilizadas actualmente, basada en la adaptación de parámetros mediante matrices de rango reducido, que es la empleada en este trabajo.

A continuación, se aborda una técnica alternativa al *fine-tuning* cada vez más extendida en entornos empresariales, ya que permite especializar los modelos sin necesidad de reentrenarlos, siendo un enfoque muy presente en soluciones actuales de inteligencia artificial generativa.

El siguiente apartado se centra en el problema de la privacidad, desglosando el problema en tres vertientes: una técnica para proteger los datos durante el entrenamiento, un enfoque preventivo para eliminar información sensible del corpus, y una revisión de los principales tipos de ataques de privacidad que pueden afectar

a estos modelos.

Posteriormente, se exponen mecanismos de optimización para reducir el tamaño de los modelos y facilitar su uso en entornos con recursos limitados. Se incluye además una descripción de los elementos clave que debe contemplar una infraestructura *on-premise* para garantizar un despliegue eficiente de LLMs.

Finalmente, se revisa la aplicación del ajuste fino en entornos reales, prestando especial atención a su uso en el ámbito empresarial: desde proveedores del servicio hasta casos de éxito concretos y tendencias actuales. Se cierra el capítulo con una revisión específica del sector climatológico, que justifica la elección del dominio de datos en este proyecto, basado en información meteorológica estructurada.

De esta manera, se ofrece una perspectiva integral y contextualizada sobre el estado actual del ajuste fino de LLMs.

2.1. Técnicas de ajuste fino para modelos

Los modelos de lenguaje suelen entrenarse en dos grandes fases. La primera consiste en el preentrenamiento con objetivos definidos, como el modelado causal del lenguaje, el modelado enmascarado o la reconstrucción de fragmentos. A continuación, tiene lugar la fase de *transfer learning* (aprendizaje por transferencia), que permite reutilizar estos modelos en tareas concretas. Dentro de esta fase se distinguen dos enfoques: el aprendizaje basado en extracción de características y el ajuste fino de parámetros [1].

En aquellos casos en los que no se dispone de un conjunto de datos suficientemente amplio como para entrenar un modelo desde cero, el aprendizaje por transferencia permite partir de pesos preentrenados. Mediante el ajuste fino, estos pesos pueden modificarse para adaptar el modelo a nuevas tareas, mejorando así su rendimiento [1].

Tradicionalmente, el *fine-tuning* implicaba modificar todos o casi todos los parámetros del modelo. Sin embargo, este proceso es costoso en términos de memoria y computación, y puede causar un fenómeno conocido como olvido catastrófico, que consiste en la pérdida de capacidades previas del modelo, especialmente aquellas de carácter más general y básico [2].

Para abordar estas limitaciones, ha surgido un enfoque denominado *Parameter-Efficient Fine-Tuning* (PEFT), que consiste en mantener congelada la mayoría del modelo base y actualizar únicamente una pequeña porción de sus parámetros. Estas técnicas permiten reducir significativamente los requisitos computacionales, manteniendo un buen rendimiento y facilitando el despliegue en entornos con recursos limitados [1].

En este proyecto, donde el ajuste fino se realiza bajo restricciones de hardware, se ha optado por estudiar algunas de las técnicas más representativas dentro de la estrategia PEFT: *adapters*, LoRA y técnicas ligeras, que se describen a continuación.

2.1.1. *Adapters*

Este tipo de ajuste fino consiste en insertar módulos adicionales, denominados adaptadores, entre las capas *feed-forward* (las que procesan individualmente cada token) de la red neuronal, entrenando exclusivamente estos módulos para tareas específicas mientras se mantienen congelados los parámetros originales del modelo. Una de las principales ventajas de este enfoque es que permite mantener la estructura del modelo preentrenado sin modificar, lo que facilita la reutilización de su conocimiento base en múltiples tareas sin necesidad de volver a entrenarlo por completo [3].

Existen diferentes variantes de módulos adaptadores, cada una con ventajas particulares según la naturaleza de la tarea y los recursos computacionales dis-

ponibles. Por ejemplo, los adaptadores en serie se insertan dentro de las capas existentes del modelo, lo que los hace adecuados para tareas en las que basta con ajustar ciertas representaciones intermedias. En cambio, los adaptadores en paralelo operan de forma paralela al flujo principal de datos, permitiendo preservar más el conocimiento original del modelo mientras se integran características especializadas propias de la tarea objetivo [4].

2.1.2. LoRA (*Low-Rank Adaptation*) y QLoRA

Low-Rank Adaptation (LoRA) es una técnica de ajuste fino eficiente que consiste en añadir matrices de bajo rango a ciertas capas del modelo, manteniendo congelados los pesos originales. LoRA transforma los parámetros entrenables en productos de matrices de rango reducido, lo que permite reducir drásticamente la cantidad de parámetros modificados durante el ajuste fino. De este modo, se disminuye el uso de memoria en GPU y se acelera el entrenamiento, sin incrementar el coste de inferencia [2] [5].

Esta técnica es especialmente útil para adaptar un mismo modelo preentrenado a múltiples tareas o dominios, ya que permite conservar una base común y entrenar únicamente adaptaciones específicas. Los beneficios de LoRA quedan por tanto bien identificados: reduce significativamente los requisitos de almacenamiento y cálculo, facilita el entrenamiento en entornos con recursos limitados y evita el sobreajuste, al introducir cambios mínimos sobre el modelo base. Sin embargo, su rendimiento depende de una correcta selección del hiperparámetro de rango, y puede no ser suficiente cuando se requieren modificaciones estructurales profundas en el modelo [2].

A partir de esta técnica se ha desarrollado QLoRA (*Quantized LoRA*), que combina la eficiencia de LoRA con técnicas de cuantización del modelo base. En lugar de usar el modelo original en precisión completa, QLoRA lo carga en formato cuantizado (normalmente en 4 bits), lo que reduce todavía más el uso de memoria.

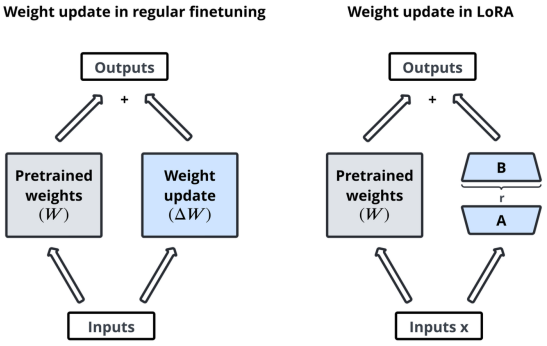


Figura 2.1: En el ajuste fino regular, se actualizan todos los pesos del modelo, mientras que en LoRA, se utilizan dos matrices de bajo rango (A y B) para aproximar la actualización de pesos, reduciendo el número de parámetros entrenables. Esto hace que LoRA sea más eficiente en memoria y computación, ideal para modelos grandes [2]-

Aun así, las matrices de LoRA se entrenan en mayor precisión (por ejemplo, en 16 o 32 bits), permitiendo conservar la calidad del ajuste fino. Esta combinación permite ejecutar el entrenamiento en GPUs con mucha menos VRAM, manteniendo una buena precisión y rendimiento del modelo [6].

En el contexto de este trabajo, se ha utilizado la técnica de LoRA sobre el modelo base preentrenado cargado en formato cuantizado (4 bits). Aunque a lo largo del documento se hace referencia general a LoRA, el enfoque empleado corresponde técnicamente a QLoRA, al combinar la cuantización del modelo con la actualización de los parámetros en matrices de bajo rango.

2.1.3. Técnicas ligeras

Dentro de los métodos de ajuste fino eficiente de parámetros (PEFT), existen algunas técnicas especialmente ligeras que destacan por requerir del entrenamiento de un número muy reducido de parámetros. Entre estas, se encuentran los enfoques basados en *prompt tuning* y *prefix tuning*, que no modifican los pesos del modelo, sino que se centran en optimizar las entradas que recibe.

El *prompt tuning* consiste en optimizar las secuencias de entrada (*prompts*) para adaptar un modelo preentrenado a tareas específicas. A diferencia de los *prompts* tradicionales, que están formados por palabras o tokens del vocabulario del modelo, aquí se utilizan vectores numéricos que no corresponden a ninguna palabra concreta. Estos vectores se sitúan "fuera del alfabeto de tokens" porque no forman parte del conjunto de palabras conocidas por el modelo, pero son capaces de activar ciertas representaciones internas del modelo de manera más precisa. De esta forma, permiten manipular directamente el comportamiento del modelo, logrando ajustes que no serían posibles con texto normal [7].

Una variante de esta técnica es el *prefix tuning*, en la que se añade una secuencia de estos vectores aprendidos al inicio de la entrada. Estos vectores también se optimizan específicamente para la tarea deseada, y su función es condicionar el estado interno del modelo desde el principio de la secuencia para facilitar así una adaptación rápida [7].

No obstante, aunque estos métodos parezcan más sencillos de implementar y requieran menos recursos computacionales, están sujetos a algunas restricciones. En primer lugar, si la tarea objetivo no se encuentra dentro de la distribución cubierta durante el preentrenamiento, ni siquiera un *prompt* óptimo será suficiente para obtener un rendimiento adecuado. En estos casos, se requiere una modificación explícita de los pesos del modelo, como sucede con métodos como LoRA o los adaptadores. Además, tanto *prompt tuning* como *prefix tuning* solo permiten reutilizar el conocimiento ya existente en el modelo, sin capacidad para adquirir habilidades nuevas ni extender sus conocimientos más allá de lo aprendido previamente [7].

Por tanto, aunque las técnicas ligeras de ajuste fino son atractivas por su bajo coste computacional y facilidad de implementación, en tareas complejas o fuera de distribución resulta imprescindible recurrir a métodos que involucren ajuste de pesos para lograr una verdadera adaptación del modelo.

2.2. Alternativa al ajuste fino: RAG (Retrieval-Augmented Generation)

Una alternativa muy popular frente al fine-tuning es el *Retrieval-Augmented Generation* (RAG), un enfoque que se basa en dividir los documentos, convertirlos en *embeddings* (representaciones numéricas de las palabras) y almacenarlos en una base de datos vectorial. Ante una consulta, el sistema recupera los fragmentos más relevantes mediante búsqueda semántica por similitud, y estos se incorporan como contexto al *prompt* que se pasa al modelo. De este modo, el LLM puede generar una respuesta más precisa sin necesidad de haber sido entrenado previamente con esa información. Este planteamiento permite que sea una alternativa idónea para una implantación rápida, ya que evita los costes y el tiempo asociados al ajuste fino o al preentrenamiento del modelo [1][2].

2.2.1. Funcionamiento y ventajas

RAG, junto al *fine-tuning*, se ha consolidado como una estrategia eficaz en PYMES que cuentan con bases de datos propias, permitiendo mejorar la relevancia de las respuestas generadas por los modelos en función del contexto corporativo. En la figura 2.7 se muestra un diagrama que recoge los componentes esenciales involucrados en un despliegue típico de LLMs en PYMES, destacando el papel de RAG y del *fine-tuning*. Ambos enfoques parten del almacenamiento estructurado de los datos internos de la empresa. Mediante un proceso de *embedding*, estos datos se transforman en vectores y se almacenan en una base vectorial que el sistema RAG consulta en cada inferencia. El LLM accede a este contexto recuperado a través de la interfaz de inferencia *Front End* [8].

Una de las principales ventajas que tiene el *Retrieval-Augmented Generation* es su bajo coste inicial y su capacidad de actualización dinámica del conocimiento. A diferencia del *fine-tuning*, no requiere reentrenar el modelo base: basta con crear

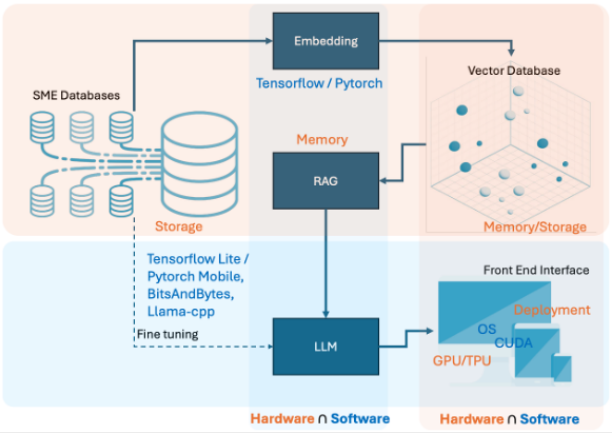


Figura 2.2: "Operacionalización de LLMs en PYMES en entornos con recursos limitados" [8].

los *embeddings* de los nuevos documentos y añadirlos a la base vectorial. Esto lo convierte en una opción especialmente útil en dominios con información en constante cambio, como el sector legal, médico o tecnológico, donde el conocimiento se actualiza con frecuencia [9].

Asimismo, RAG contribuye a reducir el riesgo de alucinación en los modelos, ya que incorpora fragmentos relevantes del corpus directamente en el *prompt*. Este *prompt* incluye tanto la pregunta del usuario como el contexto recuperado automáticamente, lo que mejora la precisión y fundamentación de las respuestas del modelo. De hecho, el estudio de Balaguer et al. [9] muestra que, con RAG, las respuestas generadas tienden a ser más concisas y mejor enfocadas en comparación con los modelos sin recuperación previa.

2.2.2. Limitaciones y comparativa

Según Balaguer et al. [9], este sistema también presenta ciertas limitaciones importantes. En primer lugar, la calidad de las respuestas depende fuertemente del sistema de recuperación y del corpus vectorizado. Si el sistema de búsqueda no es capaz de recuperar los fragmentos clave, o si el conjunto de documentos no tiene

suficiente cobertura temática ni nivel de detalle en la información, el rendimiento del modelo se ve comprometido. Además, como el contexto recuperado se incorpora directamente al *prompt*, aumenta considerablemente el tamaño de los tokens de entrada (*input token size*), lo que repercute en un mayor coste computacional por inferencia.

La imagen 2.3 muestra una comparación entre *fine-tuning* y RAG, de la que se puede concluir que RAG es más adecuado para aplicaciones que requieren acceso a datos externos y dinámicos, como sistemas de atención al cliente o consultas en tiempo real, mientras que el ajuste fino es preferible cuando se necesita un modelo altamente personalizado para tareas específicas, como en entornos especializados con datos etiquetados [2].

	Beneficios	**Inconvenientes**
RAG	· Respuestas actualizadas y precisas usando datos externos en tiempo real. · Reduce respuestas inexactas al basarse en información recuperada. · Respuestas contextuales específicas al dominio sin reentrenar el modelo.	· La precisión depende de la búsqueda por similitud, no de las capacidades del LLM. · Costes crecientes si se incrementa el número de consultas. · Desafíos en seguridad y privacidad al integrar datos externos.
Fine-Tuning	· Respuestas más precisas y adaptadas al dominio específico del ajuste fino. · Mayor control y personalización completa del modelo para tareas específicas. · Independencia de un sistema de recuperación externo.	· Coste elevado en recursos computacionales y tiempo para entrenar el modelo. · El modelo ajustado puede quedar desactualizado si los datos cambian. · Escalabilidad limitada, ya que entrenar múltiples versiones para distintas tareas es costoso.

Figura 2.3: Comparativa entre *fine-tuning* y RAG

Otro aspecto relevante es que ambos enfoques no son excluyentes, sino complementarios. En el estudio de Balaguer et al. [9] se demuestra que, incluso tras el ajuste fino, el uso de RAG mejora aún más los resultados, ya que ayuda al modelo a comprender mejor el contexto y generar respuestas más adecuadas. Asimismo,

según J et al. [1], el ajuste fino también puede emplearse para mitigar los efectos de alucinación en sistemas que utilizan RAG, afinando el modelo para reconocer cuándo debe o no confiar en el contenido recuperado.

2.3. Privacidad en Fine-Tuning

El ajuste fino de modelos de lenguaje puede implicar riesgos importantes en cuestiones de privacidad, especialmente cuando se trabaja con datos sensibles. Este es un escenario habitual en entornos empresariales, donde los datos utilizados pueden incluir información confidencial del negocio o datos personales de los empleados. La protección de esta información es prioritaria, ya que los modelos pueden acceder, aprender e incluso difundirla durante su funcionamiento posterior. Este reto surge del propio proceso de entrenamiento, en el que se utilizan grandes volúmenes de datos que pueden incluir información sensible, porque en muchos casos no han sido filtrados ni anonimizados adecuadamente [10]. Esto cobra especial relevancia en el contexto de *fine-tuning*, donde los datos empleados para ajustar los parámetros suelen ser más específicos y sensibles, como puede ocurrir en aplicaciones corporativas, sanitarias o legales.

En este proyecto no se ha trabajado con datos privados, dado que se trata de un Trabajo Fin de Máster y se ha optado por trabajar con datos públicos de carácter climatológico. Por tanto, no ha sido necesario aplicar técnicas específicas de protección de la privacidad. Sin embargo, se considera relevante incluir este apartado, ya que la privacidad durante el ajuste fino constituye uno de los principales desafíos en el despliegue real de modelos de lenguaje. A continuación, se describen los principales enfoques existentes para mitigar estos riesgos, así como un resumen de los ataques más comunes que pueden comprometer la confidencialidad de los datos.

2.3.1. *Differential Privacy* (DP)

Uno de los principales objetivos de los enfoques de privacidad en el *fine-tuning* es evitar que el modelo memorice y reproduzca información sensible presente en los datos de entrenamiento. Una de las técnicas más estudiadas para lograr este propósito es la Privacidad Diferencial (DP, por sus siglas en inglés), un marco formal que asegura que la información personal no pueda ser identificada a partir de la salida del modelo.

Tradicionalmente, DP se ha aplicado para proteger unidades de datos, como un único registro o mensaje. Sin embargo, cuando los datos provienen de usuarios que generan múltiples muestras similares (por ejemplo, varios correos electrónicos del mismo autor), proteger únicamente las instancias individuales puede resultar insuficiente. Por ello, se ha propuesto extender la privacidad diferencial al nivel de usuario (*user-level DP*), de modo que se garantice la protección de todos los datos asociados a un mismo individuo [11].

Para implementar esta privacidad, una de las estrategias más comunes consiste en añadir ruido a los gradientes durante el proceso de entrenamiento o ajuste fino mediante algoritmos específicos. Esto dificulta que el modelo memorice patrones exactos de los datos, reduciendo el riesgo de filtrado posterior sin comprometer gravemente la utilidad del modelo.

El estudio de Charles et al. [11] aborda uno de los retos principales en el contexto de LLMs: la escalabilidad de la privacidad diferencial a nivel de usuario, ya que existe un cierto compromiso entre los algoritmos que implementan la DP y la eficiencia computacional. Concluyen que es posible obtener modelos con un rendimiento competitivo y privacidad garantizada, siempre que se diseñe adecuadamente la estrategia de entrenamiento.

En definitiva, la privacidad diferencial emerge como una herramienta fundamental en el desarrollo de modelos de lenguaje seguros, especialmente en contextos

donde los datos sensibles de los usuarios (como en salud o educación) están presentes en los conjuntos de entrenamiento. No obstante, su implementación práctica aún plantea desafíos computacionales y de rendimiento que requieren soluciones específicas para los modelos a gran escala.

2.3.2. *Data scrubbing*

En el trabajo de Li et al. [10] exploran también el potencial del *scrubbing*, otro enfoque que consiste en eliminar o reemplazar la información personal identificable (PII, por sus siglas en inglés) que ha sido etiquetada con modelos preentrenados de Reconocimiento de Entidades Nombradas (NER, por sus siglas en inglés). Esta PII puede ser sustituida por tokens neutros (etiquetas como "[NOMBRE]") o por valores generados aleatoriamente [10].

Este enfoque permite reducir significativamente el riesgo de que la información privada sea memorizada por el modelo, ya que evita el acceso directo a datos sensibles en vez de protegerlos una vez han sido aprendidos. Se trata, por tanto, de una estrategia preventiva [12]. Sin embargo, también presenta ciertas limitaciones. En primer lugar, la eliminación directa de las entidades puede comprometer el sentido semántico de los ejemplos, lo que puede afectar negativamente al rendimiento del modelo en tareas que requieren coherencia narrativa o comprensión contextual [12]. Además, la efectividad del *scrubbing* depende en gran medida de la precisión del NER empleado, porque una detección incompleta o incorrecta de la PII puede dejar datos sensibles expuestos [10].

A pesar de estos inconvenientes, los autores de Li et al. [10] sugieren que el *scrubbing* puede ser una solución viable en escenarios donde se prioriza la privacidad y no se dispone de los recursos computacionales necesarios para aplicar técnicas como la privacidad diferencial. Asimismo, evalúan sistemáticamente el impacto de esta técnica en la aplicación del modelo, concluyendo que, aunque el rendimiento puede verse reducido en algunos casos, los beneficios en términos de

privacidad justifican su aplicación, especialmente en dominios sensibles.

2.3.3. Ataques de privacidad y mitigaciones

Existen múltiples tipos de amenazas a la privacidad y la seguridad, como demuestran claramente Das, Amini y Wu [12], y abarcan desde ataques adversariales (los que manipulan o introducen datos maliciosos para alterar el comportamiento del modelo) hasta ataques en el *prompt* (los que engañan al modelo mediante instrucciones diseñadas para modificar sus respuestas), vulnerabilidades específicas de privacidad, como filtración de información personal (PII); y ataques de inferencia de membresía (para conocer si un elemento o individuo específico pertenece a un determinado conjunto de datos). Esta clasificación, junto con la visión del conjunto de vulnerabilidades, se reflejan en las figuras 2.4, 2.5 y 2.6. [12].

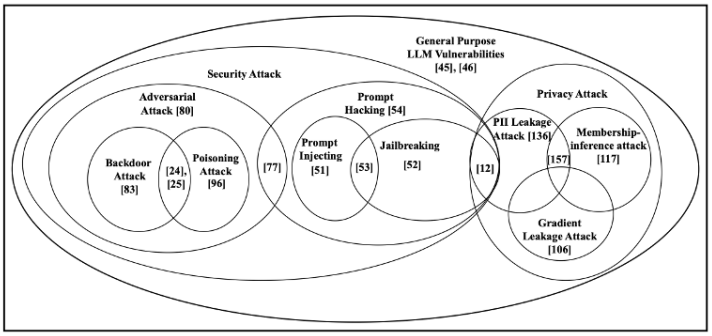


Figura 2.4: Categorías de vulnerabilidades en los LLMs [12]

.

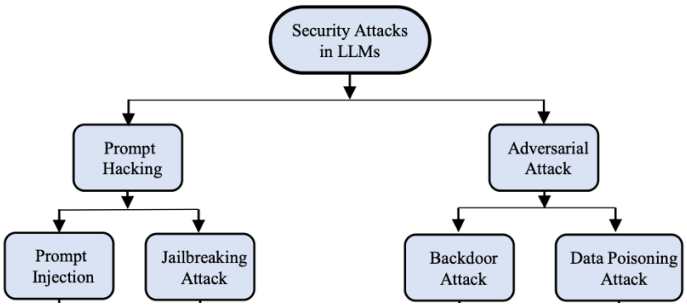


Figura 2.5: Tipos de ataques de seguridad [12]

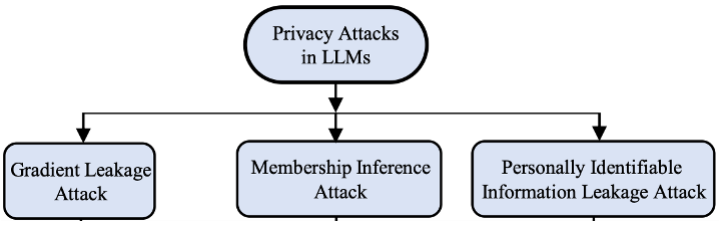


Figura 2.6: Tipos de ataques de privacidad [12]

Por otra parte, Li et al. [10] distingue dos vectores principales de filtración accidental o maliciosa de datos (*data leakage*) durante el *fine-tuning*: extracción de memoria, cuando el modelo reproduce ejemplos textuales que ha aprendido (como nombres o datos sensibles); y filtración de *prompts*, en la que se difunde las instrucciones utilizadas durante el entrenamiento o ajuste del modelo. Ambos casos exponen información sensible en sistemas desplegados y subrayan la necesidad de controles previos como *data scrubbing* y métodos como la privacidad diferencial [10].

Sin embargo, los métodos de mitigaciones siguen presentando desafíos importantes. En el caso de privacidad diferencial, agregar ruido para proteger los datos puede reducir la precisión del modelo en tareas específicas [11]. Por otro lado, el *scrubbing* puede alterar el contexto de los datos originales, afectando a la calidad semántica de las entradas y limitando la efectividad del modelo [10]. Esto demuestra que lograr un equilibrio entre privacidad y rendimiento sigue siendo uno de los principales retos en la adopción de LLMs a gran escala.

A nivel práctico, recientemente se han documentado brechas reales que evidencian estos riesgos. En mayo de 2025 se detectó un incidente denominado *EchoLeak* que afectó a Microsoft 365 Copilot, y que consistió en la manipulación del modelo mediante entradas especialmente diseñadas para forzar la revelación de información sensible (ataque al *prompt*). Esta técnica permitió extraer fragmentos de instrucciones internas y datos corporativos sin necesidad de interacción explícita por parte del usuario, lo que evidencia la peligrosidad de los ataques *zero-click* en entornos donde los LLM están integrados en herramientas de productividad. Este caso refuerza la necesidad de aplicar controles robustos, como la validación

de entradas y la supervisión continua del comportamiento del modelo en sistemas desplegados [13].

2.4. Métricas y evaluación del rendimiento de los LLMs ajustados

El análisis del rendimiento tras el ajuste fino de modelos de lenguaje requiere emplear tanto métricas cuantitativas clásicas como enfoques más específicos y avanzados. En este apartado se presenta un contexto técnico de referencia para la evaluación que combina medidas estándar ampliamente utilizadas en el ámbito del aprendizaje automático con métricas punteras que permiten valorar aspectos como la eficiencia en tareas complejas, el comportamiento tras la compresión, o el impacto en entornos con recursos limitados.

2.4.1. Métricas clásicas

El rendimiento general de los modelos de lenguaje ajustados puede analizarse utilizando un conjunto de métricas clásicas ampliamente aceptadas en la comunidad. El estudio de Parthasarathy et al. [2] propone un listado de varias que permiten analizar la generación de texto desde una perspectiva integral. A continuación, se describen algunas de las más representativas.

En primer lugar, la perplejidad mide cómo de bien el modelo predice una secuencia de tokens. Técnicamente, evalúa la incertidumbre del modelo ante la siguiente palabra. Cuanto más baja sea la perplejidad, mayor será la confianza y el rendimiento del modelo al generar texto.

La factualidad evalúa la precisión de la información proporcionada. Es decir, mide si las afirmaciones generadas por el modelo se corresponden con hechos veri-

ficables.

La incertidumbre del modelo puede estimarse a través del registro de probabilidades (*probabilities log*) asociado a cada token generado. Esta información permite identificar cuándo el modelo está menos seguro de lo que dice, lo cual es útil para interpretar errores.

La integridad se refiere a si la respuesta del modelo aborda de forma efectiva y conveniente la consulta planteada, teniendo en cuenta el contexto proporcionado. Una respuesta incompleta puede ser técnicamente correcta pero insuficiente para resolver la necesidad del usuario.

Por último, se encuentran las métricas de seguridad, que buscan garantizar que las respuestas generadas sean apropiadas y no dañinas. Estas métricas se centran en detectar contenidos tóxicos, ofensivos, sesgados o que inciten a comportamientos peligrosos. No existe una métrica única, sino que se suelen utilizar clasificadores entrenados para identificar estos riesgos.

En el estudio de Balaguer et al. [9], estas métricas también son consideradas a la hora de comparar los enfoques de RAG y *fine-tuning*. El *paper* destaca la importancia de evaluar la factualidad y la integridad para analizar la calidad de las respuestas generadas en contextos prácticos. Además, se observa que la combinación de estas métricas permite una evaluación más robusta del comportamiento del modelo, especialmente cuando se emplean técnicas como RAG para incorporar conocimiento externo.

En el contexto de este trabajo, las métricas utilizadas en la fase de evaluación han sido el *FactScore* (una métrica similar a la factualidad, pero adaptada a la verificación de hechos mediante comparación con respuestas de referencia), y la detección de alucinaciones, centrada en identificar respuestas generadas que no se correspondan con los datos del entrenamiento. Además, dado que el proyecto consiste en la generación de respuestas numéricas, se han empleado métricas específicas como el error absoluto medio (MAE), el error cuadrático medio (MSE)

y el coeficiente de determinación (R^2), con el objetivo de cuantificar la precisión del modelo en esta tarea. Estas métricas permiten una evaluación integral, tanto desde el punto de vista lingüístico como cuantitativo.

2.4.2. Métricas avanzadas de uso de recursos

Una vez abordadas las métricas centradas en la calidad de las respuestas generadas, se procede a analizar indicadores más orientados al rendimiento computacional y consumo de recursos, aspectos especialmente relevantes en contextos donde se trabaja con recursos hardware limitados, como ocurre a menudo en organizaciones pequeñas o medianas.

El despliegue de modelos en este tipo de entornos requiere no solo garantizar la privacidad de los datos, sino también optimizar el uso de energía, memoria y tiempo de inferencia. Para ello, Yee et al. [8] propone una visión integral de la eficiencia del sistema, que incluye tanto el software como el hardware. Destaca la necesidad de marcos de trabajo ligeros, bibliotecas especializadas y compatibilidad entre plataformas, así como soluciones hardware eficientes (como GPUs) que permitan minimizar el consumo energético y el uso de memoria sin comprometer el rendimiento funcional del modelo.

Por otra parte, el estudio de Yao et al. [6] plantea un *benchmark* centrado en tres tipos de métricas para evaluar distintos enfoques de entrenamiento y despliegue:

- *Test accuracy (precisión de test)*: miden la proporción de datos predichos correctamente de los modelos entrenados con diferentes enfoques (LoRA, *pruning*, etc).
- *Métricas de entrenamiento*: incluyen el pico de uso de memoria y el tiempo total necesario para el proceso de ajuste fino.
- *Métricas de inferencia*: evalúan el tiempo de inferencia medio y el pico de

memoria utilizado en distintos dispositivos *edge* (es decir, dispositivos locales o periféricos con capacidad de cálculo, como móviles).

Los resultados indican que el uso de LoRA reduce significativamente el uso de recursos sin comprometer la precisión, lo que lo convierte en una estrategia viable para entornos de hardware limitado, como los característicos de las PYMEs [6].

Además, en la literatura reciente se han propuesto marcos más amplios para estructurar la evaluación de la eficiencia en modelos de lenguaje, en respuesta a los crecientes retos relacionados con el consumo de recursos y el impacto ambiental. El trabajo de Bai et al. [14] plantea que estas cuestiones deben abordarse a lo largo de todo el ciclo de vida del modelo (desde la arquitectura del diseño hasta su despliegue). En él, se subraya:

> La enorme necesidad de recursos para entrenar o implementar modelos tan extensos puede resultar prohibitiva, especialmente en entornos con recursos limitados, como laboratorios académicos o el sector médico. Además, el impacto ambiental es una preocupación creciente, ya que el uso extensivo de GPU se traduce en un consumo significativo de electricidad y un aumento de las emisiones de dióxido de carbono.
>
> [14, p. 2]

Este *paper* define una serie de métricas para proporcionar una visión holística de los recursos que clasifican como esenciales y dividen en cinco categorías clave: cálculo, memoria, energía, costes y costes de comunicación.

Más recientemente, el *benchmark* de Yuan et al. [15] aplica seis métricas detalladas para capturar la saturación de hardware, el equilibrio entre latencia y rendimiento y el impacto en la huella de carbono. En esta comparativa se evalúan más de 100 configuraciones de modelo-técnica, obteniendo dos conclusiones clave:

(I) La eficiencia implica compensaciones cuantificables: no existe una técnica universalmente óptima; cada mejora en una métrica suele implicar pérdidas en otra. Por ejemplo, la cuantización a 4 bits permite reducir el uso de memoria y energía hasta en un factor de 3.9, pero a costa de una pérdida de precisión del 3 % al 5 %.

(II) Los óptimos dependen de la tarea y la escala: por ejemplo, RSLoRA solo supera a LoRA en eficiencia por encima de los 14 mil millones de parámetros, lo que refleja la complejidad de las interacciones entre técnica, arquitectura y hardware.

En conjunto, estos trabajos subrayan la necesidad de mantener un marco de evaluación completo, actualizado y contextualizado, que permita valorar no solo la calidad de las respuestas, sino también la viabilidad y sostenibilidad del uso de modelos de lenguaje en entornos reales.

En el contexto de este trabajo, los recursos han sido medidos desde tres vectores principales:

Figura 2.7: Métricas del uso de recursos empleadas en el proyecto.

Estas tres perspectivas permiten una evaluación más completa del comportamiento del modelo ajustado, no solo desde el punto de vista cualitativo de las respuestas, sino también en relación con su eficiencia, escalabilidad y sostenibilidad en entornos reales con limitaciones técnicas.

2.5. Optimización y escalabilidad para recursos limitados

Ya se ha comentado anteriormente la importancia de utilizar modelos en entornos *on-premise* para garantizar la seguridad y el control sobre la información sensible. A continuación, se detallan algunas de las técnicas actualmente disponibles para la compresión de modelos de lenguaje, así como los tipos de infraestructura que se emplean en su despliegue local, con el objetivo de facilitar su ejecución en sistemas con recursos computacionales limitados.

2.5.1. Técnicas de compresión de modelos

En primer lugar, destaca la técnica de cuantización, que consiste en el proceso de restringir un conjunto continuo de valores a un conjunto discreto. Aplicada a los LLMs implica representar los pesos del modelo con menor precisión numérica, lo que reduce significativamente el tamaño del modelo sin comprometer sustancialmente su rendimiento. Por defecto, la mayoría de los pesos de los modelos de lenguaje de código abierto se publican con precisión de punto flotante de 32 bits, lo que significa que cada valor numérico se representa utilizando 32 bits, ofreciendo aproximadamente 7 cifras decimales significativas de precisión. Esto implica que, incluso para modelos relativamente pequeños como el de este proyecto, se requieren cerca de 28 GB de espacio solo para almacenar los pesos [1].

Esto se debe a que una mayor precisión implica más operaciones de movimiento de memoria durante el ajuste fino, lo que aumenta tanto el consumo de energía como el coste computacional. La cuantización, al reducir la precisión a formatos más ligeros como enteros de 8 o incluso 4 bits, permite disminuir el uso de memoria, reducir el tiempo de inferencia y hacer viable el despliegue del modelo en dispositivos menos potentes [1].

Otra técnica ampliamente utilizada es la destilación de conocimiento, en la que un modelo más pequeño y eficiente es entrenado para replicar el comportamiento de uno más grande y complejo. A través de este proceso, el modelo pequeño retiene el conocimiento y las habilidades del original pero siendo mucho más ligero [2].

Por otra parte, la técnica conocida como *pruning* (poda en español) consiste en identificar y eliminar los pesos menos representativos o redundantes del modelo, para reducir su tamaño y por tanto complejidad. Existen variantes como el *structured pruning* (elimina bloques completos de la red) y el *unstructured pruning* (elimina conexiones individuales), cada uno con diferentes implicaciones de hardware [16].

En el paper de Dong et al. [17], se establece la primera comparativa que evalúa cómo afectan las técnicas de compresión a las capacidades agénticas de los LLMs, es decir, su capacidad para ejecutar tareas complejas como planificación, razonamiento, uso de herramientas o comprensión de contexto extenso. Este estudio analiza el impacto de la compresión desde tres dimensiones:

1. Cómo se ven afectadas esas capacidades
2. Cómo se altera la comprensión general del modelo
3. Qué diferencias existen entre técnicas como cuantización, destilación y *pruning*

Como principales conclusiones, se observa que la elección de la técnica de compresión (como cuantización o *pruning*) es clave, y debe adaptarse a la tarea. Por ejemplo, la cuantización a 4 bits apenas degrada las capacidades agenticas (1–3 %) en muchas tareas, pero en casos concretos puede provocar una reducción de precisión de hasta un 10–15 % [17].

Este último artículo pone de manifiesto la necesidad de incorporar la compresión como factor relevante en la evaluación de modelos, ya que puede afectar no solo

al rendimiento clásico, sino también a su capacidad de razonamiento y toma de decisiones [17]. En definitiva, aunque la compresión permite ejecutar LLMs en entornos con recursos limitados, es esencial valorar el equilibrio entre eficiencia y calidad de respuesta, especialmente en tareas complejas.

2.5.2. Infraestructura y despliegue *on-premise*

A continuación, se aborda la creciente tendencia a desplegar modelos en infraestructuras con recursos limitados, una estrategia motivada por la necesidad de aplicaciones más autónomas que prioricen la privacidad de los datos y reduzcan la dependencia de la conectividad con la nube. En Yee et al. [8], se analiza este contexto aplicado a PYMES, planteando estrategias desde las perspectivas tanto de software como de hardware.

Desde el punto de vista del software, se destaca la relevancia de utilizar *frameworks* (marcos de trabajo) optimizados que permiten reducir el uso de memoria, mejorar los tiempos de inferencia y garantizar que los modelos puedan ejecutarse correctamente en dispositivos con menor capacidad. Además, se subraya la importancia de trabajar con librerías que estén disponibles y mantenidas en múltiples sistemas operativos (Windows, macOS, Linux), asegurando la compatibilidad y consistencia en los entornos heterogéneos de las empresas. Otro aspecto clave es el uso de plataformas como CUDA (*Compute Unified Device Architecture*), que permiten aprovechar la aceleración por GPU para operaciones en paralelo, aumentando notablemente el rendimiento en tareas de inferencia intensiva [8].

Aunque más adelante, en el apartado 3.3, se justifica el uso de Google Colab para el desarrollo de este proyecto, los dos puntos anteriores refuerzan dicha decisión. En concreto, el ordenador disponible durante el desarrollo utiliza macOS, un sistema cuyo hardware (basado en chips de Apple) no es compatible con CUDA al carecer de tarjetas gráficas NVIDIA. Por tanto, el uso de Colab ha permitido trabajar en un entorno compatible con múltiples sistemas operativos y que, además, dispone

de soporte completo para CUDA, asegurando la replicabilidad y escalabilidad del proyecto.

En cuanto al hardware, la integración de LLMs en dispositivos móviles o entornos de *edge computing* (procesamiento de datos en la periferia de la red, lo más cerca posible del origen de los datos) no solo mejora la eficiencia, sino que también garantiza la privacidad del usuario y una mayor capacidad de respuesta del sistema. En el estudio reciente de Yao et al. [6], se propone un marco innovador de *fine-tuning* que optimiza tanto la precisión de inferencia como la eficiencia energética de los LLMs desplegados localmente. El método combina *pruning* (poda) adaptativo en los módulos congelados con la asignación dinámica de rangos LoRA en los bloques LoRA entrenables, lo que permite adaptarse a la heterogeneidad de hardware, especialmente común en PYMES. Este enfoque, denominado FedSpine, se apoya en un algoritmo que aprende la relación cuantitativa entre el consumo de recursos y el rendimiento obtenido, lo que permite ajustar el nivel de compresión a las capacidades reales de cada dispositivo.

En definitiva, tanto el diseño de la infraestructura software como las soluciones adaptativas de hardware resultan esenciales para que el despliegue *on-premise* de modelos de lenguaje sea viable, eficiente y escalable, especialmente en entornos profesionales con recursos limitados y necesidades de privacidad elevadas.

2.6. Aplicación del ajuste fino de LLMs en el entorno empresarial

Tras revisar la parte más técnica del proyecto, se aborda ahora su aplicación práctica en el ámbito empresarial, con especial atención a proveedores de servicios y casos de uso reales.

2.6.1. Empresas proveedoras de servicios de *fine-tuning*

Este subapartado tiene como objetivo mostrar cómo se está utilizando el *fine-tuning* en el mundo real, a través de empresas que lo ofrecen como producto o servicio comercial. En particular, se identifican compañías que permiten a otras organizaciones adaptar modelos de lenguaje a sus propios dominios o tareas específicas, ya sea proporcionando infraestructura, modelos base o ejecutando directamente el proceso de ajuste.

Entre los ejemplos más representativos se encuentra Databricks, una empresa estadounidense de software empresarial con orígenes en el entorno académico y de la comunidad de código abierto. A través de su propia plataforma, ofrecen una suite de soluciones de inteligencia artificial llamada Mosaic AI, que incluye la funcionalidad de Mosaic AI Training. Este módulo permite realizar el *fine-tuning* de modelos de lenguaje de forma segura, escalable y reproducible, tanto para modelos propios como para modelos *open source* alojados en Hugging Face. La herramienta facilita el preprocesamiento, entrenamiento, evaluación y despliegue de modelos ajustados en un único entorno integrado, lo que simplifica significativamente el trabajo para empresas que buscan adaptar modelos a sus datos internos o a dominios altamente especializados [18].

Por otro lado, OpenAI ofrece una línea de negocio enfocada a empresas a través de su plataforma API. En este caso, el enfoque es algo diferente, ya que OpenAI no realiza directamente el ajuste fino para terceros, sino que proporciona la infraestructura necesaria en su plataforma para que los propios clientes puedan realizar el *fine-tuning* de modelos como GPT-3.5-turbo. El proceso se basa en subir ejemplos de entrada y salida en un formato estándar, que permiten ajustar el comportamiento del modelo para tareas como clasificación, asistencia personalizada, generación de textos con estilo propio, etc. Además, OpenAI garantiza que los datos utilizados en estos procesos no se reutilizan para mejorar sus modelos base, lo cual es especialmente relevante para empresas que manejan información confidencial o sensible [19].

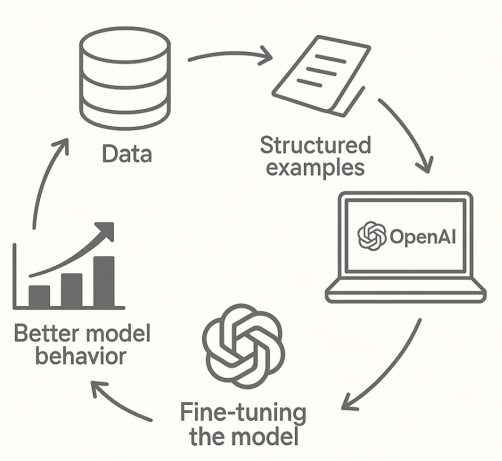


Figura 2.8: Flujo de trabajo para el ajsute fino mediante la plataforma de OpenaAI [19].

Una tercera empresa destacada es Cohere, una tecnológica canadiense especializada en IA aplicada al entorno empresarial. Cohere ofrece una línea de productos orientada a la customización de modelos de lenguaje, donde destaca el servicio de ajuste fino basado en la familia de modelos Command R. Este servicio permite adaptar modelos a tareas específicas del cliente utilizando técnicas como LoRA, facilitando la incorporación de contexto empresarial, terminología especializada o comportamiento deseado. La plataforma de Cohere proporciona, además, herramientas para evaluar la calidad del modelo ajustado y realizar un despliegue optimizado [20].

Aterrizando este panorama al contexto español, encontramos varias iniciativas interesantes. Una de ellas es Sherpa.ai, una empresa vasca especializada en asistentes digitales predictivos y privacidad diferencial. En su línea de soluciones de inteligencia artificial generativa, Sherpa ofrece una plataforma de entrenamiento privado de LLMs, permitiendo a las empresas enviar sus propios datos y delegar en Sherpa el proceso completo de *fine-tuning*. Esta solución cumple con la normativa de protección de datos (GDPR) y permite personalizar modelos de lenguaje sin comprometer la confidencialidad de la información. Disponen de líneas de negocio adaptadas a sectores concretos como salud, legal, ventas y publicidad, donde el lenguaje técnico y contextualización son especialmente relevantes [21].

Otra empresa nacional relevante es Open Sistemas, una consultora tecnológica con sede en Madrid que destaca por su apuesta por soluciones IA para el sector público. Han desarrollado SofIA, una plataforma diseñada para conectar aplicaciones institucionales con modelos LLM, facilitando su integración, personalización y optimización. Uno de sus casos de éxito más notables ha sido en ayuntamientos, donde SofIA ha permitido automatizar tareas administrativas, mejorar la atención ciudadana y adaptar el lenguaje de los modelos a contextos burocráticos concretos [22].

Más allá del sector privado, también empiezan a aparecer referencias al *fine-tuning* en el ámbito público. Por ejemplo, un pliego de contratación del Servicio Murciano de Salud de este año solicita que las soluciones basadas en inteligencia artificial cuenten con capacidades de adaptación mediante ajuste fino, especialmente en sistemas de procesamiento del lenguaje natural orientados a lenguaje clínico especializado. Esta exigencia busca mejorar la precisión de los modelos y facilitar su aplicación en tareas sensibles como la interpretación de textos médicos o la redacción automática de informes [23].

En otro caso, un pliego del Ministerio de Asuntos Económicos y Transformación Digital incluye como criterio de valoración la experiencia en la "adaptación (fine-tuning) y entrenamiento de modelos de lenguaje para dominios de especialidad". Esta cláusula aparece en el contexto del proyecto INESData, una incubadora nacional de espacios de datos e inteligencia artificial, en colaboración con la Escuela Técnica Superior de Ingenieros Informáticos de la UPM, y demuestra cómo incluso organismos públicos comienzan a valorar la especialización de modelos LLM como elemento diferencial [24].

Por tanto, puede afirmarse que el ajuste fino de modelos de lenguaje es una práctica de extrema relevancia y cada vez más valorada tanto en el ámbito público como empresarial, consolidándose como un eje clave en el desarrollo de soluciones avanzadas de inteligencia artificial.

2.6.2. Casos de éxito reales en distintos sectores

Algunos casos de uso representativos con éxito demostrado a nivel mundial se encuentran en sectores como las finanzas, el ámbito legal, el desarrollo de software o la salud. En estos contextos, el *fine-tuning* permite adaptar modelos base a necesidades muy específicas, mejorar el rendimiento en tareas especializadas y aprovechar datos internos de cada dominio para obtener soluciones más precisas y eficaces.

Por ejemplo, FinGPT es un modelo de código abierto diseñado para al sector financiero, que mejora la investigación y cooperación en este ámbito al promover la accesibilidad a los datos y abordar problemáticas comunes como la adquisición, calidad y estandarización de la información financiera. Ha sido desarrollado por AI4Finance Foundation, una organización sin ánimo de lucro centrada en la aplicación de IA al sector financiero. FinGPT está basado principalmente en la familia de modelos de Llama de Meta y se ha ajustado utilizando técnicas como LoRA sobre datos financieros reales como noticias, informes financieros y publicaciones en redes sociales, entre otros [2].

En el dominio legar, destaca LAWGPT, el primer modelo *open source* diseñado específicamente para aplicaciones jurídicas en chino, que ha demostrado un rendimiento superior en tareas legales en ese idioma frente a modelos generalistas como Llama. Parte del modelo base Alpaca Plus de 7B, un modelo chino, y se ha ajustado finalmente utilizando LoRA. El entrenamiento se llevó a cabo con un conjunto de datos legal y amplio, que incluía más de 300,000 ejemplos, abaracando tareas como predicción de tipo de delito, consultas penales y respuestas a preguntas legales [2].

Otros ejemplos destacados en la industria incluyen las familias de modelos Salesforce CodeGen y BioMedGPT. Ambos fueron inicialmente preentrenados desde cero para tareas específicas: CodeGen para generación de código y BioMedGPT para tareas biomédicas multimodales. Sin embargo, estos modelos han seguido evo-

lucionando mediante procesos de ajuste fino, lo que ha permitido mejorar su rendimiento en tareas especializadas como la comprensión de instrucciones en lenguaje natural para programación o la clasificación de imágenes clínicas y la generación de informes médicos [25] [26].

2.6.3. Oportunidades en dominios especializados: el caso de la climatología

Aunque los motivos de la elección de un conjunto de datos de climatología se detallan en el apartado 3.3.3, en este subapartado se justifica por qué la climatología representa también un sector especialmente interesante para la aplicación de técnicas de ajuste fino.

Como se ha observado en los apartados anteriores, la mayoría de los ejemplos industriales actuales se concentran en sectores como finanzas, la salud o el ámbito legal. Esta propuesta demuestra que también es posible aplicar *fine-tuning* en dominios como la climatología, donde los datos numéricos son abundantes y pueden utilizarse para entrenar modelos adaptados a tareas específicas como la predicción, la inferencia o la automatización de informes técnicos.

El paper de Anonymous [27] expone con claridad la relevancia del ajuste fino en este ámbito. Partiendo de que los modelos climáticos actuales se basan en ecuaciones físicas y simulaciones costosas computacionalmente, y que las parametrizaciones tradicionales de estos modelos presentan problemas de generalización y optimización, los autores proponen un enfoque innovador: emplear modelos de lenguaje para aprender dichas parametrizaciones directamente a partir de datos históricos. Mediante el ajuste fino sobre un modelo meteorológico preentrenado, demuestran que es posible especializarlo para tareas concretas del dominio físico, abriendo la puerta a una nueva generación de modelos híbridos, más eficientes y adaptables. Este trabajo explora el *fine-tuning* aplicado a datos no textuales, en una línea paralela a la que se propone en este TFM, y subraya además el potencial

de esta técnica para reducir los costes computacionales del entrenamiento [27].

Un caso complementario es el del *paper* de Subich [28], en el que se trabaja con el modelo GraphCast, desarrollado por Google DeepMind y originalmente entrenado con datos del ECMWF (European Centre for Medium-Range Weather Forecasts). En este estudio, el modelo se ajusta utilizando datos del sistema canadiense, lo que permite evaluar su capacidad de adaptación a un nuevo contexto geográfico y de datos. Como resultado, el modelo ajustado mejora la precisión en múltiples variables meteorológicas, incluso con un volumen de datos reducido, demostrando que el ajuste fino permite una especialización eficaz sin necesidad de entrenar desde cero ni disponer de grandes recursos computacionales [28].

En un artículo más reciente, de este mismo año, se subraya que los modelos puramente físicos están siendo cada vez más complementados por enfoques *data-driven* (impulsados por datos), basados en aprendizaje profundo. Se justifica la necesidad de herramientas más flexibles y adaptables, como los LLMs y redes neuronales, que puedan aprender patrones directamente de los datos sin recurrir a ecuaciones físicas complejas. El artículo señala que muchos modelos generalistas presentan limitaciones para adaptarse a regiones específicas o fenómenos locales, y que esta precisión puede mejorarse a través del ajuste con datos concretos. Además, se destaca la alta relevancia estratégica y social del sector climático, fundamental para ámbitos como la planificación agrícola, la gestión de emergencias, la energía renovable o la logística [29].

Este trabajo se centra específicamente en el ajuste fino con datos meteorológicos numéricos procedentes de estaciones climáticas, pero el dominio de la climatología abarca múltiples direcciones donde estas técnicas también resultan útiles. Así lo demuestra el caso de ClimateBERT, un modelo especializado en el análisis de lenguaje relacionado con el cambio climático. En este trabajo, se aplicó *fine-tuning* sobre un modelo generalista para adaptarlo a tareas de procesamiento de lenguaje natural en textos ESG (del inglés, *Environmental, Social and Governance*), informes financieros o declaraciones de impacto climático. El estudio evidencia que el lenguaje climático posee una complejidad semántica que requiere modelos espe-

cializados, y que el ajuste fino mejora significativamente su rendimiento en tareas como la clasificación de riesgos financieros relacionados con el clima [30]. Estos resultados refuerzan la idea de que la climatología, tanto en su vertiente textual como numérica, es un dominio especialmente adecuado para el ajuste fino de LLMs.

En conclusión, aplicar técnicas de ajuste fino al dominio de la climatología no solo es viable desde el punto de vista técnico, sino que también resulta estratégico y socialmente valioso. La posibilidad de adaptar modelos generalistas a tareas específicas mediante el uso eficiente de datos reales contribuye a mejorar la precisión, reducir costes y facilitar la aplicabilidad práctica de los LLMs en un sector crítico y en transformación.

Capítulo 3

Metodología

En este capítulo se detalla la planificación y ejecución del proyecto. En primer lugar, se expone el flujo de trabajo y la organización temporal del trabajo. A continuación, se describen los recursos empleados, incluyendo el hardware, las herramientas y librerías utilizadas, así como la descripción y el preprocesamiento del *dataset*. Por último, se detalla el entorno de entrenamiento para el ajuste fino del modelo seleccionado.

3.1. Esquema general del flujo de trabajo

Para facilitar la comprensión del *workflow* seguido en este proyecto, se presenta en la figura 3.1 un esquema que resume las etapas principales y los *notebooks* asociados a cada una. Cada bloque representa un cuaderno de Google Colab independiente, lo que permite mantener una organización clara y modular, facilitando tanto la depuración del código como la trazabilidad de los resultados. Esta representación gráfica tiene un carácter esquemático y sirve como guía general para entender el desarrollo del proyecto, que se detalla en profundidad en los capítulos

3 y 4.

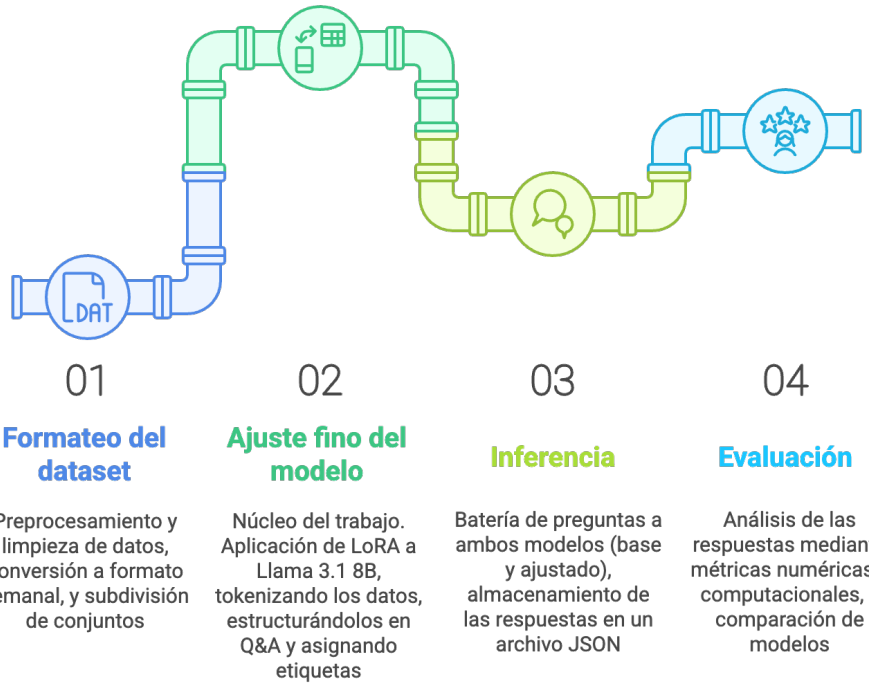


Figura 3.1: Resumen del flujo de trabajo del proyecto y cuadernos empleados

3.2. Planificación del proyecto

En la figura 3.2 se presenta un diagrama de Gantt que recoge la planificación temporal que se ha seguido, junto con el tiempo aproximado de duración. Cabe destacar la fase de "refinamiento de código", añadida debido a la naturaleza experimental del proyecto. Al tratarse de un trabajo basado en programación y tiempos de entrenamiento prolongados, es necesario disponer de un margen adicional para realizar ajustes, resolver posibles errores y optimizar los resultados obtenidos.

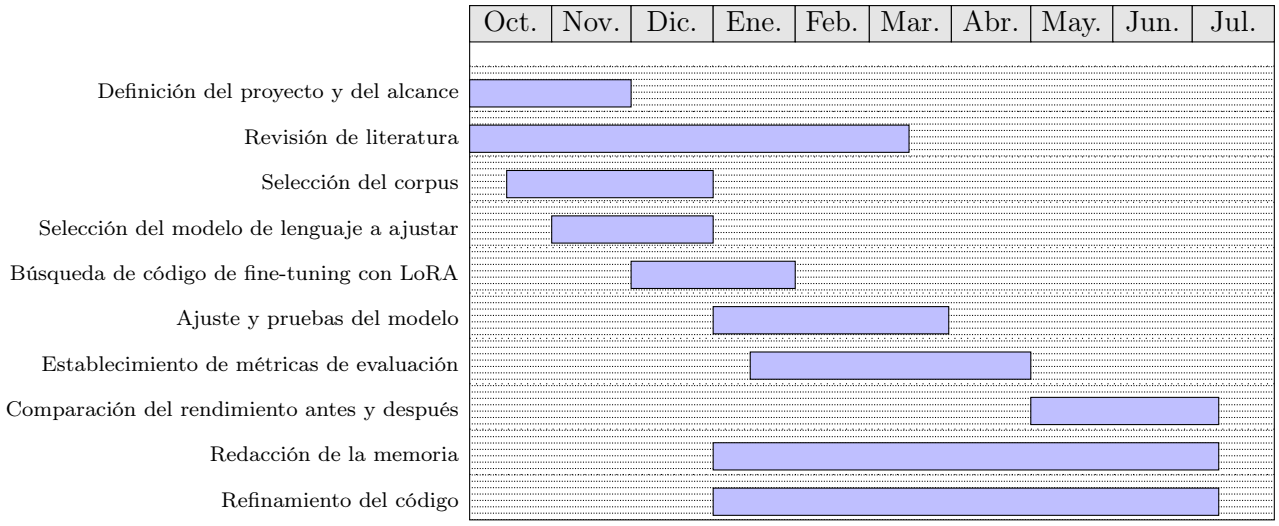


Figura 3.2: Cronograma de Gantt del proyecto

3.3. Recursos empleados

Para el *fine-tuning* del modelo se han empleado diversos recursos, tanto de hardware como de software. Se detallan seguidamente los equipos utilizados y las herramientas de software, incluyendo el *dataset* seleccionado y su procesamiento.

3.3.1. Hardware utilizado

Se ha empleado Google Cloud Platform (GCP) como principal infraestructura para escribir y compilar código. Esta plataforma ofrece un entorno flexible y fácilmente escalable, permitiendo simular capacidades de procesamiento más altas durante las etapas de entrenamiento y ajuste del modelo. Aunque no es estrictamente necesario recurrir a la nube, esta elección permite trabajar de manera más eficiente y ágil, especialmente en la fase experimental.

La suscripción de Google Colab Pro incluye acceso a dos tipos de GPU NVIDIA, la Tesla T4 de 16GB y la A100 de 40GB. La primera, basada en la arquitectura

Turing, está optimizada para tareas de inferencia en inteligencia artificial y aprendizaje automático. Esta arquitectura incorpora núcleos CUDA eficientes junto con núcleos Tensor especializados que permiten ejecutar operaciones de aprendizaje profundo con una buena relación entre consumo energético y rendimiento. Su diseño compacto y su bajo consumo (alrededor de 70W) la hacen muy adecuada para entornos en los que se prioriza la eficiencia energética y la versatilidad. Sin embargo, la T4 está pensada sobre todo para desplegar modelos ya entrenados y no para entrenarlos desde cero, ya que su capacidad de computación es limitada en comparación con modelos más avanzados [31]. En este proyecto, cuando la carga de trabajo ha superado los recursos disponibles de la A100, la plataforma ha redirigido temporalmente a una Tesla T4, lo que ha supuesto una reducción significativa en el rendimiento, llegando a impedir el entrenamiento completo del modelo.

En cambio, la A100, basada en la arquitectura Ampere, ha sido la GPU que se ha priorizado por su elevado rendimiento y su orientación a cargas de trabajo intensivas. Ampere representa una evolución significativa respecto a arquitecturas anteriores (como Turing), ya que incorpora núcleos Tensor de tercera generación, núcleos CUDA optimizados y soporte para nuevos tipos de precisión, que equilibran velocidad y exactitud en operaciones de *deep learning*. Además, su alto ancho de banda de memoria y su escalabilidad la convierten en una herramienta especialmente potente para el entrenamiento de modelos grandes y complejos. Gracias a estas características, la A100 permite reducir de forma considerable los tiempos de entrenamiento y mejorar el rendimiento en comparación con generaciones anteriores. El valor de potencia de diseño térmico (TDP, por sus siglas en inglés *Thermal Design Power*) de esta GPU es de 400 W, lo que indica la máxima potencia que puede llegar a consumir bajo cargas intensivas sostenidas [32]. En la figura 3.3 se muestra una gráfica oficial de NVIDIA que compara el rendimiento de la A100 frente a generaciones previas en diferentes tareas de entrenamiento.

Como puede observarse, el salto de rendimiento con respecto a generaciones previas es especialmente notable en tareas propias del aprendizaje profundo, por lo que ha sido esencial poder acceder a esta GPU en este proyecto para ejecutar

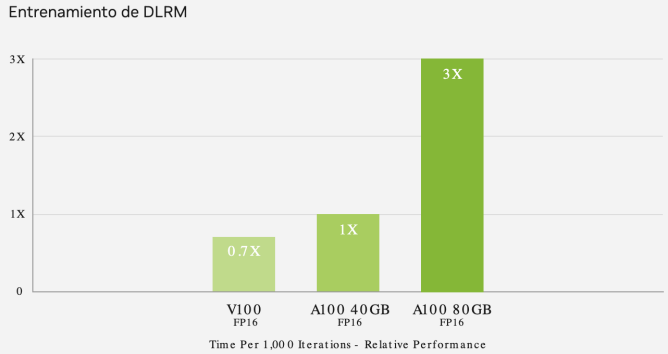


Figura 3.3: Entrenamiento de IA hasta 3 veces superior en los modelos más grandes [32]

el entrenamiento de forma fluida y reducir el tiempo.

Asimismo, resulta útil situar el rendimiento de la A100 dentro de una perspectiva evolutiva más amplia. En la figura 3.4 se representa la evolución de la eficiencia computacional y la capacidad de memoria de distintas generaciones de GPUs NVIDIA desde 2016 hasta la actualidad. Aunque el modelo Tesla T4 no aparece específicamente en el gráfico, sí lo hace su arquitectura Turing. Esto permite contextualizar su posición relativa respecto a otras arquitecturas como Ampere o las más recientes Hopper y Ada Lovelace.

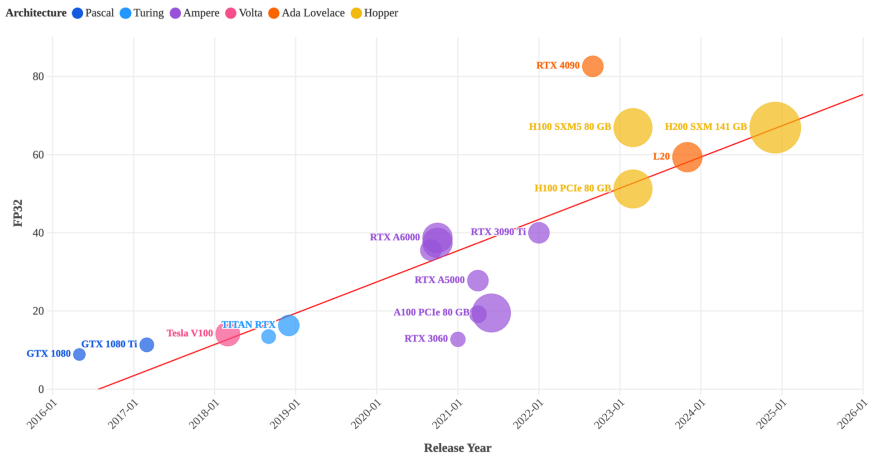


Figura 3.4: Tendencia en eficiencia computacional y capacidad de memoria de las GPUs NVIDIA. Cada color representa una arquitectura distinta, y el tamaño del punto refleja la memoria integrada. La línea roja marca la tendencia general de mejora en eficiencia computacional [15].

Como se observa, la arquitectura Ampere, sobre la que se basa la A100, supone un claro avance respecto a generaciones anteriores como Volta o Turing, tanto en capacidad de cálculo como en memoria disponible. Esto refuerza la necesidad del uso de esta GPU en tareas de entrenamiento y ajuste fino de modelos grandes, como el empleado en este proyecto.

A pesar de estas capacidades tan avanzadas, la suscripción a Google Colab Pro impone ciertas limitaciones, como sesiones con una duración máxima de 24 horas y tiempos de espera por inactividad para disuadir a los usuarios de usar Colab para tareas de larga duración. Además, este tiempo de espera es inversamente proporcional al uso de recursos, suponiendo todo esto un inconveniente que se discutirá más adelante.

3.3.2. Herramientas y librerías de software

Para acceder al modelo preentrenado y mantener una buena organización en el desarrollo del trabajo, se ha empleado la plataforma de Hugging Face, ya que proporciona repositorios públicos con código oficial y documentación extensa de modelos LLM ampliamente utilizados en la comunidad. Además, se ha utilizado para gestionar los scripts del proyecto y poder diferenciar entre las categorías de *modelos* y *datasets*, ya que a lo largo del proceso se han subido múltiples versiones del modelo tras distintas pruebas de entrenamiento, así como varios conjuntos de datos de distintos periodos temporales.

Por otro lado, el punto de partida para el fine-tuning proviene de un repositorio público de GitHub CondingMindset [33], desarrollado por el usuario *CodingMindset*, el cual contiene un cuaderno de Jupyter con un ejemplo práctico de *fine-tuning* utilizando el modelo Llama 3.1. Este cuaderno ha sido modificado en gran medida y complementados con diferentes *notebooks* alojados en Google Colab, que han sido adaptados específicamente para las necesidades del proyecto. Aunque el objetivo principal de este trabajo no es profundizar en el código, dichos *notebooks* serán brevemente analizados a lo largo de este capítulo y del siguiente, donde se describe el desarrollo completo del entrenamiento y evaluación del modelo.

3.3.3. Descripción del *dataset* y preprocesamiento de datos

La elección del *dataset* para el proceso de *fine-tuning* resultó ser considerablemente más laboriosa de lo que inicialmente se había estimado, ya que debía cumplir con tres requisitos fundamentales para servir adecuadamente al propósito del proyecto:

- **Tamaño equilibrado:** el conjunto de datos debía ser lo suficientemente grande como para permitir que el modelo aprendiese patrones relevantes,

pero sin superar las limitaciones de memoria y tiempo de ejecución impuestas por la plataforma de Google Colab.

- **Formato Q&A:** los datos debían poder estructurarse en forma de pares pregunta–respuesta.
- **Contenido específico:** el contenido no podía ser excesivamente simple ni demasiado generalista, ya que en ese caso el modelo preentrenado probablemente ya habría sido expuesto a información similar durante su entrenamiento inicial.

Tras evaluar distintas opciones, se optó finalmente por utilizar el conjunto de datos **GHCN (Global Historical Climatology Network)** de Menne et al. [34], una base de datos con más de dos siglos de observaciones meteorológicas diarias registradas por estaciones terrestres distribuidas por todo el mundo. Entre las variables disponibles, se seleccionaron y conservaron las siguientes, consideradas más relevantes y presentes en un mayor número de estaciones. Además, en todas ellas se realizó una conversión a unidades más comunes para facilitar su comprensión y procesamiento posterior:

- Temperatura máxima diaria [°C]
- Temperatura mínima diaria [°C]
- Temperatura media diaria [°C]
- Precipitación (lluvia, nieve derretida) [mm]
- Nevadas [mm]
- Racha de viento máxima [m/s]

Al comenzar a trabajar con estos datos, rápidamente se hicieron evidentes ciertas dificultades prácticas. La descarga del conjunto de datos desde el espacio de almacenamiento S3 de AWS resultaba excesivamente lenta, por lo que se optó por llevar a cabo toda la descarga y el preprocesamiento en un *notebook* independiente, separado del dedicado al ajuste fino. Además, se observó que entrenar el modelo durante una sola época con el *dataset* completo podía llevar más de 15 horas, lo

que hizo necesario reducir progresivamente el volumen de datos para adaptarse a las limitaciones de tiempo de ejecución.

En primer lugar, se aplicó un filtrado para conservar únicamente las estaciones situadas en España y desde el año 2000 en adelante. Posteriormente, se llevó a cabo un submuestreo para convertir los datos de frecuencia diaria a frecuencia semanal, disminuyendo así el número de registros pero manteniendo la representatividad climática general. Este enfoque, sin embargo, conllevaba el riesgo de perder patrones específicos a nivel diario, pero ha sido asumido para obtener una mayor eficiencia en el entrenamiento.

Aun así, los tiempos seguían siendo elevados, por lo que se realizaron pruebas adicionales acortando el rango temporal del *dataset* a tramos como 2014–2024 o incluso 2020–2024, con el objetivo de acelerar el proceso manteniendo únicamente las tendencias más recientes. Asimismo, otra alternativa fue aplicar un submuestreo adicional sobre el *dataset* ya procesado de 2014–2024, seleccionando aleatoriamente 25.000 ejemplos de entrenamiento y 2.000 de validación. De este modo, se conservaba una cobertura temporal amplia, pero con un número de ejemplos más reducido y manejable. El conjunto de test se mantuvo intacto para evitar introducir sesgos en la evaluación y conservar su función como referencia objetiva. En los tres subconjuntos se mantuvo la división temporal que se explica con detalle más adelante. Aunque este cambio redujo significativamente el consumo de memoria -de unos 22 GB a poco más 8 GB- y permitió ejecutar el entrenamiento en GPUs menos potentes, como la T4, el tiempo total seguía siendo considerable. Por ejemplo, el proceso final duraba unas 13 horas en la A100, pero se prolongaba hasta 25 horas en la T4. Esto puede deberse a la diferencia en la arquitectura y la precisión de cálculo comentadas anteriormente.

El proceso completo de filtrado, reducción y submuestreo aplicado al conjunto de datos se resume en la figura 3.5, donde se muestra de forma visual la secuencia de transformaciones realizadas desde el *dataset* GHCN original hasta la versión final utilizada para el ajuste fino.

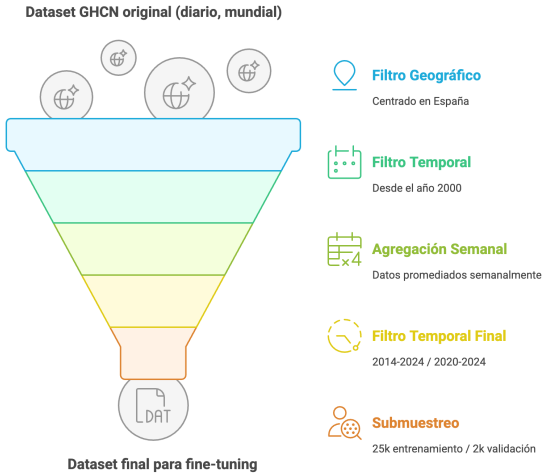


Figura 3.5: Esquema de preprocesamiento del *dataset* desde el GHCN original hasta la versión final para ajuste fino.

A continuación, la tabla 3.1 muestra un resumen cuantitativo de cada una de las versiones generadas del *dataset* durante el preprocesamiento.

Versión del Dataset	**Rango temporal**	**Nº de filas**	**Frecuencia de los datos**
Dataset original	Hasta 3 siglos en algunas estaciones	10 486 827	Diaria
Dataset para España	2000–2024	4 536 626	Diaria
Dataset reducido *versión 1*	2000–2024	1 356 000	Semanal
Dataset reducido *versión 2*	2014–2024	694 700	Semanal
Dataset reducido *versión 3*	2020–2024	313 000	Semanal
Dataset final	2014–2024	143 000	Semanal

Tabla 3.1: Resumen de las distintas versiones del dataset utilizadas en el proyecto.

La versión final, correspondiente al rango temporal de 2014-2024, con submuestreo semanal y selección aleatoria de ejemplos, contiene 172 000 filas, siendo 25 000 de entrenamiento, 2 000 de validación y 116 000 de test.

El preprocesamiento se realizó inicialmente sobre el *dataset* en bruto, aplicando

posteriormente la división en subconjuntos de entrenamiento, validación y test. Este orden garantiza que los tres conjuntos partan del mismo rango temporal reducido, asegurando homogeneidad en los datos de entrada.

El conjunto de entrenamiento, que constituye el 80 % del total disponible para esa fase, abarca datos entre 2014 y 2022. Esta elección permite simular un caso de uso habitual en entornos empresariales, donde es frecuente adaptar un modelo con datos históricos antes de ser desplegado sobre información más reciente.

Por su parte, el 20 % restante se reservó para el conjunto de validación, también limitado al periodo 2014-2022 y obtenido mediante muestreo aleatorio. Esta selección busca mantener la independencia de los datos utilizados para ajustar los hiperparámetros sin introducir sesgos temporales.

El conjunto de test está formado exclusivamente por datos de los años 2023 y 2024, posteriores al *cutoff* oficial de conocimiento de Llama 3.1 (diciembre de 2023). Esta decisión permite evaluar con mayor precisión la capacidad del modelo de generalizar a datos futuros no vistos durante el entrenamiento. El proceso de división se resume visualmente en la figura 3.6, tomada de *Conjuntos de datos: división del conjunto de datos original* [35], donde se observa cómo la validación se utiliza para ajustar el modelo y el conjunto de test se reserva exclusivamente para confirmar su rendimiento final.

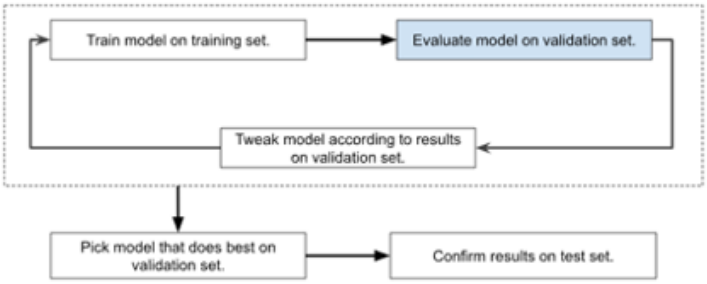


Figura 3.6: Ciclo habitual de entrenamiento, validación y evaluación [35].

Esta estrategia reproduce un entorno realista en el que se entrena con datos históricos y se despliega el modelo para operar con datos más actuales, permitiendo

así comprobar su capacidad de extrapolación temporal.

Una vez filtrado y preprocesado el conjunto de datos, se procedió a darle formato de pregunta-respuesta (*Q&A*), con el objetivo de adaptar esos datos a una tarea generativa. Esta estrategia se debe a que, durante el preentrenamiento de los modelos LLM, se utilizan grandes cantidades de texto no estructurado. Al convertir los datos a un formato Q&A, donde el *input* es una pregunta y el *output* la respuesta, se facilita que el modelo procese el texto de forma continua y aproveche mejor la información, similar a su modo de aprendizaje previo. En este caso, se diseñó una plantilla de preguntas ligeramente adaptada al contexto climatológico, para estructurar los *inputs* del modelo siguiendo el siguiente formato: *¿Cuál fue el valor de [variable meteorológica] se registró en la estación [nombre estación] durante la semana del [rango temporal]?*.

```
def apply_template(batch):
    elements = ["TMAX", "TMIN", "PRCP", "TAVG", "SNOW", "WSFG"]
    texts = []
    labels = []

    for i in range(len(batch["ID"])):
        for element in elements:
            value = batch[element][i]
            if value is not None and not pd.isna(value):
                pregunta = (f"Cul fue el valor de {element} en la estacin
↪  {batch['NAME'][i]} durante la semana del {batch['WEEK_RANGE'][i]}?
↪ ")
                texts.append(pregunta)
                labels.append(float(value))

    return {"text": texts, "label": labels}
```

Código 3.1: Función `apply_template` de estructuración de los datos en formato Q&A.

Este formateo se aplica tanto al conjunto de entrenamiento como al de validación, para garantizar que el modelo procese los datos en un formato uniforme durante todo el entrenamiento. La estructura resultante separa la entrada textual (*texts*) del valor objetivo (*labels*), lo que facilita que el modelo aprenda a generar una respuesta numérica concreta a partir de una pregunta formulada en lenguaje natural, algo coherente con su naturaleza autoregresiva.

Por último, es importante destacar que el objetivo final de este trabajo es realizar el ajuste fino sobre una base de datos desarrollada en el marco de otro Trabajo de Fin de Máster (TFM), actualmente pendiente de presentación. Dado que dicho conjunto de datos aún no está disponible de forma oficial, se ha optado por emplear temporalmente el dataset AWS GHCN como referencia, permitiendo así construir y validar todo el flujo de trabajo del preprocesamiento y del entrenamiento.

3.4. Entrenamiento del modelo

En este apartado se describe el proceso completo de entrenamiento y ajuste fino del modelo, incluyendo la configuración del entorno y de la técnica LoRA, y los parámetros empleados. El cuaderno de Jupyter principal empleado para el ajuste fino ha sido adaptado a partir de un código base de referencia, con el objetivo de ajustarse a las necesidades específicas del proyecto. Los detalles sobre la configuración del entorno, los hiperparámetros utilizados y el funcionamiento del proceso de entrenamiento se presentan a lo largo de los siguientes subapartados.

El modelo seleccionado para realizar el proceso de ajuste fino ha sido Llama 3.1 8B, un modelo de lenguaje de código abierto desarrollado por Meta y lanzado en julio de 2024. Este modelo corresponde a la versión más pequeña de la familia Llama 3.1, con 8.000 millones de parámetros entrenables, frente a los 405.000 millones de parámetros de su versión más grande. A pesar de ser menor, Llama 3.1 8B ofrece un rendimiento adecuado dentro del ecosistema de modelos *open source*, siendo suficiente para las necesidades de este proyecto. Además, según

Meta, este modelo ha sido optimizado específicamente para tareas de generación de lenguaje natural, priorizando la robustez, la adaptabilidad a distintos contextos y la facilidad de integración en diferentes entornos de desarrollo [36].

Aunque Llama 3.1 8B no alcanza los mejores resultados absolutos frente a modelos cerrados de última generación, su rendimiento en el benchmark independiente *LiveBench A Challenging, Contamination-Free LLM Benchmark* [37] se encuentra dentro de lo esperable para modelos de su tamaño, superando a alternativas de código abierto de menor capacidad como Vicuna 7B. Además, al ser *open source*, se ha podido disponer de un ecosistema amplio de recursos, como el *notebook* base, para facilitar la implementación práctica del proyecto. A ello se suma el respaldo de la comunidad de desarrolladores y usuarios de Meta, que garantiza un soporte activo para la resolución de problemas y la mejora continua del modelo.

Por todo ello, Llama 3.1 8B ha sido considerado el modelo adecuado para emplear en este trabajo.

3.4.1. Configuración del entorno

El cuaderno de entrenamiento comienza con la instalación de los paquetes necesarios para el proceso de ajuste fino, entre ellos la librería *Unsloth*, una herramienta optimizada que permite cargar modelos de HuggingFace ya cuantizados en 4 bits. Esto implica que el modelo se carga utilizando menos bits por parámetro, lo que permite reducir el peso del modelo en memoria sin afectar de forma significativa a su rendimiento. Esta cuantización reduce considerablemente el uso de memoria y acelera el entrenamiento, lo que resulta especialmente relevante en entornos con recursos computacionales limitados, como Google Colab o GPUs de capacidad baja.

Antes de proceder con la carga del modelo, se verifica que haya una GPU disponible, ya que los procesos de cuantización y ajuste fino con LoRA requieren

capacidades de cálculo que no son viables en CPU. De hecho, la propia carga del modelo cuantizado con *Unsloth* lanza errores si no se detecta GPU, por lo que esta comprobación se realiza al principio. Después, el modelo Llama 3.1-8B se descarga cuantizado en 4 bits, y reduciendo la longitud máxima de las secuencias a 512 tokens (`max_seq_length`), ya que en el caso de este trabajo las entradas y salidas tokenizadas son muy cortas. Esta decisión permite reducir el consumo de memoria y acelerar el entrenamiento sin afectar al rendimiento. Este mismo valor se ha mantenido posteriormente durante la inferencia para mantener la coherencia.

A continuación, se procede a descargar los conjuntos de entrenamiento y validación del *dataset* GHCN, previamente formateado y publicado en el repositorio interno del proyecto. Después, se tokenizan los ejemplos utilizando una función que adapta los datos al formato requerido por el modelo. Para cada muestra, se construye una cadena de texto combinando la entrada (*text*) y la salida (*label*), y se procesa con el tokenizador oficial del modelo para convertirla en una secuencia de identificadores. Este paso es necesario para transformar las preguntas y respuestas del formato Q&A en vectores numéricos que el modelo pueda interpretar durante el entrenamiento. Esta implementación se muestra en la función 3.2, que se encarga de construir los *prompts*, tokenizar los datos y generar las etiquetas para el entrenamiento.

```
def tokenize_sample(batch):
    prompts = [text + " La respuesta es: " + str(label) for text, label
    ↪ in zip(batch["text"], batch["label"])]
    model_inputs = tokenizer(prompts, truncation=True, padding=True)

    answers = [" " + str(label) for label in batch["label"]]
    answer_ids = tokenizer(answers, add_special_tokens=False)["input_ids"
    ↪ ]

    labels = []
    for input_ids, ans_ids in zip(model_inputs["input_ids"], answer_ids):
        l = [-100] * len(input_ids)
        l[-len(ans_ids):] = ans_ids
```

```
        labels.append(l)

    model_inputs["labels"] = labels
    return model_inputs
```

Código 3.2: Función `tokenize_sample` de conversión de los datos a cadenas de texto.

Una vez establecido este formato, se normalizan las etiquetas para facilitar el aprendizaje por parte del modelo, ya que en iteraciones anteriores se había observado que los valores originales, al tener rangos muy distintos, dificultaban la convergencia durante el entrenamiento. Además, como el modelo se entrena con formato conversacional, tanto la entrada como la salida de los datos deben estar en formato textual, por lo que las etiquetas numéricas se convierten en cadenas de texto antes de ser utilizadas.

Tras realizar la normalización, se ha verificado la distribución de las etiquetas tanto en el conjunto de entrenamiento como en el de validación, ya que una distribución desbalanceada podría afectar negativamente al aprendizaje. En la figura 3.7 se observa una concentración mayor de valores en el tramo inferior del rango normalizado para el conjunto de entrenamiento, aunque con suficiente dispersión hasta valores cercanos a 1. En el conjunto de validación se detecta un pico marcado alrededor de 0.25, posiblemente asociado a la sobrerrepresentación de una condición meteorológica concreta, pero no se ha considerado un problema crítico dado que el modelo sigue expuesto a un rango amplio de ejemplos.

El *scaler* que realiza la normalización se guarda para poder aplicarlo posteriormente durante la inferencia, pudiendo recuperar así los valores reales a partir de las respuestas generadas por el modelo.

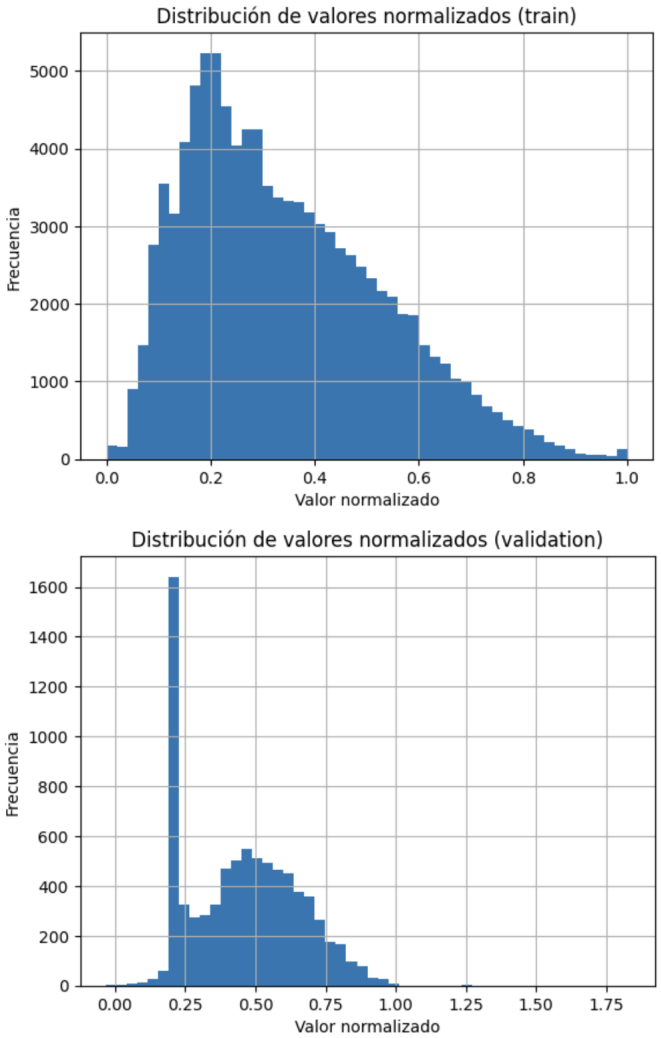


Figura 3.7: Distribuciones de las etiquetas de los datos

3.4.2. Configuración del modelo con LoRA

Como ya se ha comentado anteriormente, la técnica principal utilizada para el ajuste fino del modelo ha sido LoRA, que permite adaptar modelos grandes sin necesidad de modificar todos sus parámetros, mediante la introducción de matrices de bajo rango. Esta técnica se configura a través de tres parámetros principales: rango, alfa y módulos objetivo. Estos valores se establecen al momento de cargar el modelo cuantizado, y determinan respectivamente la capacidad de adaptación

del modelo, el equilibrio con la red original y los componentes a los que se aplica la modificación.

Además de estos parámetros esenciales, se emplean otras configuraciones complementarias como el estabilizador del rango, la desactivación del *dropout* o el almacenamiento de gradientes en disco (*gradient checkpointing*), que influyen en la estabilidad y eficiencia del entrenamiento.

La tabla 3.2 resume los valores utilizados y una breve explicación de cada uno. Cabe señalar que todos estos parámetros provienen del *notebook* base del repositorio original del proyecto, y no han sido modificados porque proporcionaban buenos resultados.

3.4.3. Hiperparámetros del entrenamiento

Por otra parte, en la función de entrenamiento se definen varios hiperparámetros que rigen el comportamiento del proceso de ajuste fino. Todos se encuentran definidos en el *notebook* de ajuste, por lo que para simplificar se van a explicar únicamente aquellos que han sido cambiados o añadidos con respecto al código base:

- **Conjunto de validación**: se añade explícitamente un conjunto de validación separado para evaluar el rendimiento del modelo durante el entrenamiento.
- **Número de épocas** (`num_train_epochs`): se incrementa a 10 con el objetivo de disponer de una visión más completa del entrenamiento, y observar en qué punto se estabiliza o mejora el rendimiento del modelo.
- **Pasos de registro** (`logging_steps`): se fija en 100, lo que permite registrar la pérdida cada 100 pasos. Esto suaviza las gráficas resultantes y evita saturar los registros.

- **Estrategia de evaluación y guardado** (`eval_strategy` y `save_strategy`): se configuran para evaluar y guardar el modelo al final de cada época. Esto asegura una evaluación periódica y evita pérdidas de progreso.
- **Pasos de calentamiento** (`warmup_steps`): Se han establecido como un 10 % del total de pasos de entrenamiento, con el objetivo de reducir las oscilaciones iniciales de la función de pérdida. Este porcentaje ha sido elegido siguiendo recomendaciones empíricas adaptadas en la comunidad, como en el caso de Devlin et al. [38], que utilizaron un 10 % de pasos de *warmup* para el preentrenamiento del modelo BERT.
- ***Early Stopping***: se incorpora esta estrategia para detener el entrenamiento si no se observa mejora en la métrica de validación durante dos épocas consecutivas. Esta técnica es especialmente útil por tres razones según Parthasarathy et al. [2]:
  - *Prevención del sobreajuste*: permite detener el entrenamiento cuando el modelo comienza a memorizar el conjunto de entrenamiento.
  - *Eficiencia computacional*: reduce el tiempo de entrenamiento si el modelo deja de mejorar.
  - *Optimización de recursos*: evita el uso innecesario de GPU en un entorno con recursos limitados.

A continuación, en el código 3.3, se muestra un fragmento de la configuración utilizada en el entrenamiento, destacando únicamente los hiperparámetros que han sido modificados respecto al código base.

```
args=TrainingArguments(
        learning_rate=2e-4,
        lr_scheduler_type="linear",
        per_device_train_batch_size=8,
        gradient_accumulation_steps=2,
        num_train_epochs=10,
        fp16=not is_bfloat16_supported(),
        bf16=is_bfloat16_supported(),
```

```
        logging_steps=100, # Guarda la prdida cada 100 pasos para
    ↪ suavizar la grfica sin saturar los logs
        eval_strategy="epoch", # Se evala al final de cada epoch
        save_strategy="epoch", # Se guarda al final de cada epoch
        save_total_limit=2, # Se guardan los dos ltimos modelos (de las
    ↪ ltimas epochs)
        optim="adamw_8bit",
        weight_decay=0.01,
        warmup_steps=warmup_steps,
        seed=123,
        load_best_model_at_end=True,  # Early stopping. Carga el mejor
    ↪ modelo basado en la mtrica de validacin
        metric_for_best_model="eval_loss",
        greater_is_better=False,  # Early stopping. Detiene si no hay
    ↪ mejora despus de 2 epochs
    )

trainer.add_callback(EarlyStoppingCallback(early_stopping_patience=2))
```

Código 3.3: Fragmento de hiperparámetros ajustados en el entrenamiento

Parámetro	Descripción	Valor aplicado/Aplicación	Justificación
Parámetros esenciales			
Rango	Determina el tamaño de las matrices de LoRA. Puede variar desde 8 a 256.	16	Permite mantener el coste computacional bajo, aunque no se almacene tanta información.
Alfa	Factor de escala que establece la contribución de los adaptadores LoRA a la red principal.	16	Suele valer lo mismo que el rango o el doble, ya que valores altos aumentan la capacidad de adaptación pero también el riesgo de sobreajuste.
Módulos objetivo	Componentes a los que se aplican los adaptadores de LoRA (bloques *feed-forward*, capas de salida, etc.).	Aplicado a módulos lineales	Limitarlo a módulos lineales mejora la calidad sin aumentar excesivamente el coste computacional.
Otros parámetros			
Dropout	Técnica de regularización que desactiva conexiones aleatorias durante el entrenamiento.	No	No se emplea porque ralentiza el entrenamiento.
Estabilizador del rango	Introduce una modificación en el factor de escala (alfa) para estabilizar el aprendizaje.	Sí	Busca un equilibrio entre la capacidad de adaptación y la estabilidad del entrenamiento.
***Checkpointing* del gradiente**	Almacena temporalmente capas en disco para liberar memoria en la GPU.	Sí	Permite reducir el consumo de VRAM y mejorar la eficiencia.

Tabla 3.2: Parámetros utilizados en la configuración de LoRA

Capítulo 4

Resultados y discusión

En este capítulo se presenta el análisis y la discusión de los resultados obtenidos, organizados en dos subapartados. El primero se centra en la evaluación del propio proceso de ajuste fino, mientras que el segundo aborda la evaluación del modelo ajustado como producto final, comparando su rendimiento con el del modelo base preentrenado, en línea con uno de los objetivos principales del proyecto.

Lo que se consigue con este enfoque es obtener una visión completa tanto del proceso de entrenamiento como del rendimiento final del modelo, permitiendo valorar no solo la mejora en precisión, sino también la eficiencia y la robustez del sistema ajustado frente al modelo base.

Para ello, se han definido distintas métricas distribuidas en *notebooks* independientes, según el momento del flujo de trabajo en el que se recogen. En el archivo de *fine-tuning* se evalúa la evolución del modelo durante el entrenamiento mediante la curva de pérdidas, complementado con métricas de eficiencia como el consumo energético estimado y el uso de memoria (VRAM). Por su parte, en el *notebook* de evaluación (a partir de los resultados almacenados previamente en el de inferencia) se analizan la velocidad de generación y la calidad de las respuestas. Esta separación estructural permite no solo una mejor organización del análisis, sino también

aislar el impacto del ajuste fino en cada etapa del ciclo de vida del modelo: tanto durante su entrenamiento como en su rendimiento posterior.

Esta visión completa de la evaluación se resume visualmente en la tabla 4.1 con una comparativa de métricas por etapa.

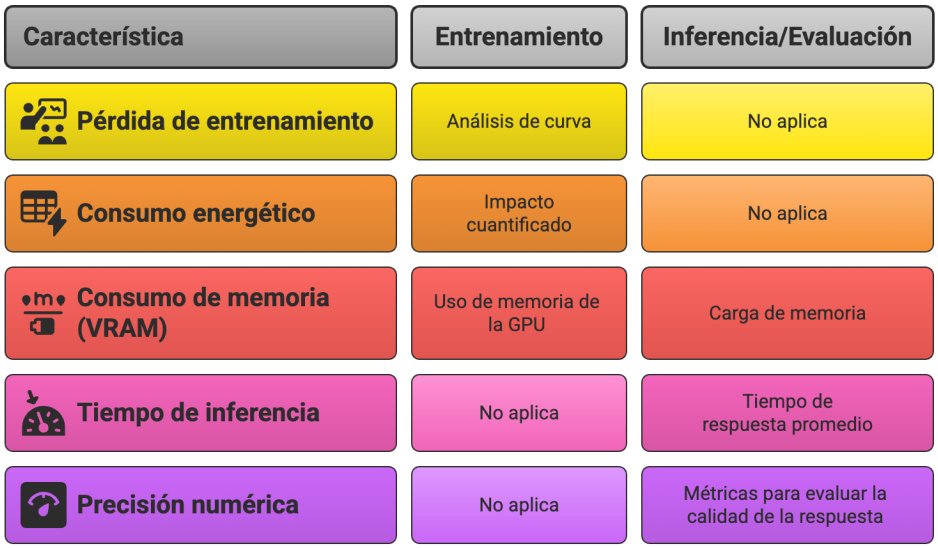


Figura 4.1: Resumen de métricas utilizadas en las fases de entrenamiento y evaluación.

4.1. Resultados del entrenamiento

4.1.1. Curva de pérdidas (*Training Loss*)

Durante el proceso de entrenamiento, se ha monitorizado la evolución de la función de pérdida con el objetivo de evaluar la estabilidad del aprendizaje y detectar posibles problemas de ajuste. Para ello, se han generado las siguientes gráficas:

1. **Función de pérdidas para los subconjuntos de entrenamiento y va-**

lidación frente al número de *epochs*: Esta gráfica permite comparar el comportamiento del modelo sobre datos vistos (entrenamiento) y no vistos (validación), lo que resulta fundamental para identificar signos de sobreajuste (cuando la pérdida de validación aumenta mientras la de entrenamiento sigue disminuyendo) o de subajuste (cuando ambas pérdidas se mantienen elevadas).

2. **Función de pérdida frente al número de *epochs* únicamente para el subconjunto de entrenamiento:** Se incluye esta gráfica adicional para facilitar un análisis más detallado de la evolución del error sobre los datos de entrenamiento sin la posible interferencia visual de la curva de validación.

A partir de estas representaciones gráficas se busca extraer conclusiones sobre el comportamiento general del modelo durante el ajuste fino del proyecto. En primer lugar, el análisis de la forma de la curva de pérdida a lo largo del entrenamiento permite verificar que el modelo está aprendiendo progresivamente y de forma estable. Además, la comparación entre las curvas de entrenamiento y validación facilita la identificación de posibles síntomas de sobreajuste, cuando el modelo se adapta en exceso a los datos de entrenamiento y pierde capacidad de generalización, o de subajuste, cuando no consigue capturar patrones relevantes en los datos, manteniendo errores elevados.

En las gráficas 4.2 y 4.3 se observan, en general, resultados muy positivos. Los puntos representados corresponden al valor medio de la pérdida por *epoch*, lo que implica que el entrenamiento no partió de valores tan bajos como los inicialmente representados, sino que descendió rápidamente durante la primera pasada por los datos. La pérdida de entrenamiento muestra un descenso gradual y consistente, que se estabiliza en torno a valores muy bajos (por debajo de 0.02) a partir de la sexta *epoch*, lo que sugiere una buena capacidad del modelo para ajustarse a los datos vistos.

En cuanto a la pérdida en el conjunto de validación, aunque presenta una mayor variabilidad, su tendencia general también es decreciente. Destaca un incremento

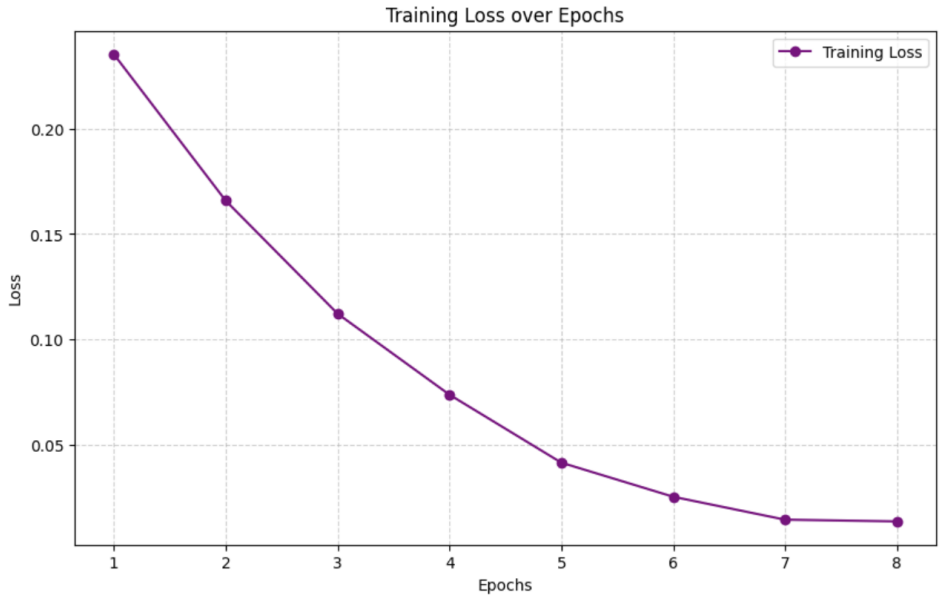


Figura 4.2: Evaluación de la pérdida en el conjunto de entrenamiento.

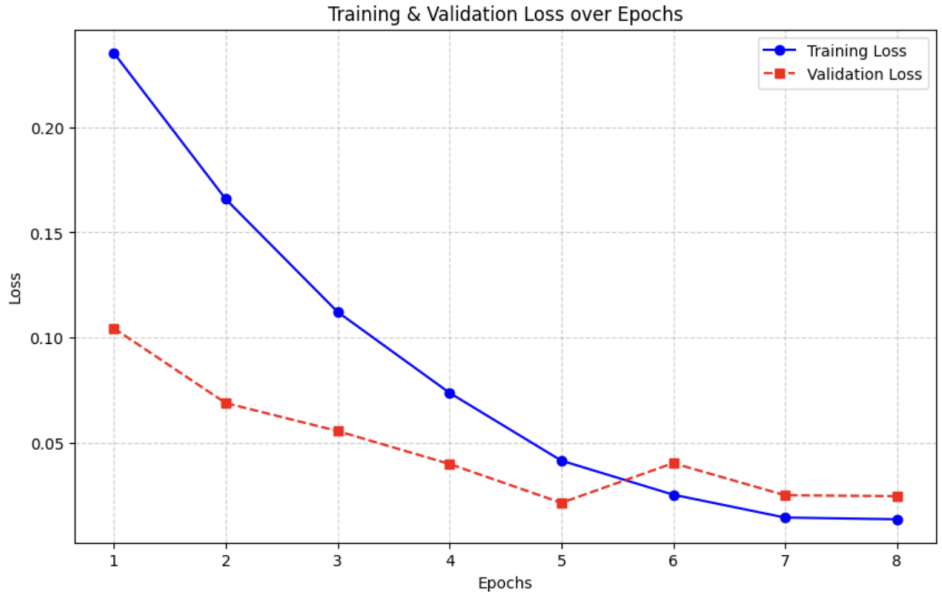


Figura 4.3: Evaluación de la pérdida en el conjunto de entrenamiento y validación.

puntual en la *epoch* seis, donde la pérdida prácticamente se duplica respecto a la anterior. Este fenómeno podría deberse a un *minibatch* de validación atípico, es decir, un conjunto de ejemplos que el modelo no ha logrado generalizar correcta-

mente. Sin embargo, esta subida no se mantiene y en las dos últimas *epochs* vuelve a descender, aproximándose otra vez a la curva de entrenamiento.

Comparando ambas curvas, no se aprecian signos evidentes de sobreajuste ni subajuste. Las pérdidas no divergen de manera muy significativa y, de hecho, tienden a aproximarse a partir de la cuarta *epoch*. Esto sugiere que el modelo ha adquirido una buena capacidad de generalización. Si el entrenamiento se hubiese prolongado durante más *epochs*, es posible que ambas curvas hubiesen convergido aún más, aunque esto no se puede afirmar con certeza.

Cabe destacar que, aunque se configuró un máximo de 10 *epochs*, el entrenamiento finalizó tras la octava debido al *early stopping*. Se intentó completar la ejecución completa eliminando esta técnica, pero aun así la sesión de Google Colab se interrumpió al superarse el límite de actividad continua de 24 horas. Disponer de estas dos *epochs* adicionales habría resultado útil para confirmar si la pérdida de validación sí que tendía a estabilizarse cercana a la curva de entrenamiento.

4.1.2. Evaluación de la eficiencia computacional en el entrenamiento

Esta evaluación se ha llevado a cabo en el propio *notebook* de *fine-tuning*, ya que todas las métricas consideradas están directamente relacionadas con el proceso de entrenamiento del modelo. El objetivo es analizar el impacto computacional del ajuste fino desde dos perspectivas complementarias: el consumo energético y el uso de memoria de la GPU.

En primer lugar, se ha realizado una estimación del consumo energético total durante el entrenamiento. Para ello, se combinan dos métodos:

- Una **estimación teórica**, basada en el TDP (*Thermal Design Power*) de la GPU utilizada (400 W en este caso), multiplicado por el tiempo total de

entrenamiento.

- Una **medición empírica**, obtenida mediante un muestreo periódico cada 5 minutos de la potencia consumida, lo que permite aproximar el consumo medio real a lo largo del proceso.

Consumo energético teórico [kWh]	Consumo energético total registrado [kWh]	Consumo energético medio [W]
5.040	3.2837	260.61

Tabla 4.1: Consumos energéticos del entrenamiento.

En la tabla 4.1 se recogen los resultados obtenidos mediante ambos enfoques. Si se hubiera mantenido un uso constante de la tarjeta gráfica A100 en su consumo máximo (400W), el entrenamiento hubiese supuesto un total de 5 kWh. Sin embargo, la medición real indica un consumo de 3.28 kWh, lo que refleja que la GPU no estuvo en uso pleno durante toda la sesión, como se observa también en la gráfica 4.4, donde la memoria utilizada se mantiene por debajo de los 40 GB. Esto se corresponde con una potencia media consumida de 260.61 W, ponderada cada cinco minutos a lo largo de toda la sesión.

En segundo lugar, se ha analizado el uso de memoria de la GPU (VRAM) mediante un sistema de monitoreo automatizado. Para ello, se ha implementado una función que registra periódicamente el uso de memoria, guardando los valores en un archivo de log y mostrándolos en pantalla. Este registro se activa justo antes de iniciar el entrenamiento y realiza capturas cada cinco minutos, un intervalo elegido para equilibrar la capacidad de detección de picos de uso con la generación de archivos de tamaño razonable. Esta métrica permite observar el comportamiento de la carga de GPU durante el ajuste fino y detectar posibles cuellos de botella o limitaciones del entorno de ejecución.

Como se observa en la curva de la gráfica 4.4, el ascenso en el uso de la memoria VRAM es rápido al inicio, probablemente debido a la carga inicial de pesos del modelo y la reserva de espacio para los tensores. Cuando llega a 8753 MB, que ha sido el valor pico, se estabiliza para el resto de la duración del entrenamiento. Este

Figura 4.4: Uso de la memoria GPU durante el entrenamiento.

patrón muestra una asignación de recursos eficiente, sin caídas ni cambios bruscos en la memoria. La duración total del entrenamiento fue de aproximadamente 12 horas y media.

4.2. Evaluación del modelo después del ajuste fino

Para garantizar una valoración precisa e imparcial del modelo ajustado, la evaluación se ha realizado mediante preguntas generadas exclusivamente a partir del conjunto de test del dataset climatológico. Este conjunto no ha sido empleado en ninguna fase del entrenamiento ni ajuste de hiperparámetros, evitando así problemas habituales en aprendizaje automático como la fuga de información (*data leakage*) o la contaminación del conjunto de evaluación. De este modo, los resultados obtenidos reflejan de manera fiel la capacidad de generalización del modelo ante datos completamente nuevos.

La evaluación consiste en calcular una serie de métricas, tanto numéricas como computacionales, para comparar el rendimiento del modelo preentrenado y el modelo ajustado a partir de sus respuestas a una batería de preguntas. Estas

1000 preguntas, simples y estructuradas, se han generado de manera aleatoria a partir del conjunto de test completo, correspondiente a los años 2023 y 2024, con muestreo semanal aplicado. Este enfoque permite una evaluación rápida, masiva y reproducible. Dado que todas las preguntas se refieren a un periodo posterior al utilizado durante el ajuste, las métricas obtenidas permiten valorar con mayor precisión la capacidad del modelo para generalizar a datos futuros no vistos durante el entrenamiento.

4.2.1. Proceso de inferencia

El proceso de inferencia se lleva a cabo en un *notebook* independiente con el fin mantener una organización clara dentro del flujo de trabajo. El objetivo en esta etapa es obtener las respuestas, tanto del modelo base como del ajustado, a las preguntas explicadas previamente. Estos datos se almacenan en sendos archivos JSON (uno por cada modelo), que posteriormente se utilizarán en el cuaderno de evaluación descrito en el siguiente subapartado.

El primer problema identificado en el proceso de inferencia fue el elevado tiempo de respuesta del modelo preentrenado Llama 3.1 8B, debido a que cada paso de generación requiere una gran cantidad de operaciones computacionales sobre sus 8 mil millones de parámetros. Esta carga no se observa en el modelo ajustado, ya que, al aplicar LoRA, solo se han entrenado y activado el 0.52 % de los parámetros durante el *fine-tuning*, lo que permite una inferencia más eficiente. Para reducir este tiempo, se tomaron tres medidas. En primer lugar, se empleó la herramienta `torch.no_grad()` de la librería *PyTorch*, que desactiva el cálculo de gradientes durante la inferencia. En la función de `model.generate()` de *PyTorch* se activa por defecto el seguimiento de operaciones para el cálculo de gradientes aunque no se esté entrenando al modelo, como es el caso en este *notebook*. Esto supone una carga innecesaria en memoria y tiempo, que se evita con el uso de esta herramienta *no_ grad context manager* [39]. Además, se limitó el número máximo de tokens generados (`max_new_tokens`) a solamente 30, ya que las respuestas esperadas son

breves y de carácter numérico. Esto permite al modelo responder de forma más rápida y concisa, reduciendo además la probabilidad de alucinar. Finalmente, se implementó un guardado automático cada 10 respuestas generadas, lo que permite conservar resultados parciales en caso de que el entorno de Google Colab se desconecte o supere su capacidad, además de facilitar el seguimiento del progreso. El fragmento 4.1 muestra una parte del código correspondiente al proceso de generación y guardado de respuestas durante la inferencia:

```
with torch.no_grad():
          outputs = model.generate(
              input_ids=input_ids,
              attention_mask=attention_mask,
              max_new_tokens=30,
              do_sample=False
          )
output_tokens = outputs[0][input_ids.shape[-1]:]

respuesta = tokenizer.decode(
            output_tokens,
            skip_special_tokens=True
        )

# Guardar cada 10 respuestas
        if len(respuestas) % 10 == 0:
            with open(nombre_salida_json, "w", encoding="utf-8") as f:
                json.dump(respuestas, f, indent=4, ensure_ascii=False)
```

Código 4.1: Parte de generación de respuestas y guardado durante la inferencia.

El segundo problema identificado fue el diseño del *prompt* utilizado durante la inferencia. Este aspecto resulta especialmente crítico, y ya en las primeras pruebas se pudo observar la alta sensibilidad del modelo a la instrucción empleada. Aunque los LLMs poseen una notable capacidad de generalización, la formulación concreta del *prompt* influye significativamente en su rendimiento, especialmente cuando han sido entrenados para una tarea siguiendo un formato fijo. Esto se debe a que este

tipo de modelos no comprenden el lenguaje de forma semántica como lo haría un ser humano, sino que generan texto basándose en patrones estadísticos aprendidos durante el entrenamiento. Por tanto, pequeñas variaciones en la estructura de la instrucción durante la inferencia pueden dar lugar a respuestas distintas, al activar patrones diferentes. En el contexto de este proyecto, este aspecto es fundamental, ya que se requiere una alta precisión en las respuestas numéricas generadas.

Esta sensibilidad al *prompt* está respaldada por evidencia científica. Por ejemplo, en el estudio de He et al. [40] se demuestra que el formato del *prompt* puede afectar al rendimiento de modelos como GPT-3.5-turbo hasta en un 40 % en tareas específicas, como la traducción de código. Asimismo, se concluye que no existe un único formato de *prompt* óptimo que funcione para todos los casos, y que el rendimiento puede variar considerablemente según cómo se estructure la entrada. Por otro lado, el estudio de Sclar et al. [41], revela que modelos como Llama 2-13B son especialmente sensibles a aspectos superficiales del *prompt*, como la puntuación o el orden de las palabras, lo que puede afectar de forma significativa a la precisión de sus respuestas.

Por todo lo anterior, en este trabajo se ha mantenido el mismo estilo de preguntas durante la inferencia que se utilizó en el proceso de *fine-tuning*: *¿Cuál fue el valor de [variable meteorológica] en la estación [nombre estación] durante la semana del [rango temporal]?* Dada la naturaleza numérica y directa de la tarea, esta elección favorece una mejor generalización y precisión en las respuestas. Además, no supone una ventaja injusta para el modelo ajustado ya que, aunque el modelo base no ha sido expuesto a este *prompt* específico, sí que ha sido entrenado con millones de ejemplos de estructuras similares (preguntas en lenguaje natural, referencias a unidades físicas, etc.). Por tanto, una instrucción clara, simple y bien estructurada también mejora el rendimiento del modelo base, por similitud estadística con lo aprendido durante su preentrenamiento.

4.2.2. Evaluación de la eficiencia computacional en la inferencia

Como se ha explicado al comienzo del capítulo 4, los resultados generados durante la inferencia se almacenan en un archivo, mientras que su análisis se lleva a cabo en otro. Dentro de esta fase, se han considerado dos métricas clave para valorar la eficiencia computacional del modelo durante la generación de respuestas: el tiempo de inferencia y el consumo de memoria en GPU. Ambas se han calculado para los dos modelos (el preentrenado y el base) con el objetivo de comparar su rendimiento práctico y cuantificar el posible impacto del ajuste fino en términos de coste computacional. Este análisis está alineado con el enfoque principal del proyecto, que busca no solo mejorar la precisión del modelo, sino hacerlo compatible con entornos de recursos limitados mediante técnicas de ajuste fino eficientes.

En primer lugar, se ha medido el tiempo de inferencia, es decir, el tiempo total que tarda cada modelo en generar todas las respuestas de la batería de preguntas. Esta métrica permite valorar la rapidez de ejecución y detectar posibles penalizaciones o mejoras introducidas por el proceso de ajuste fino. Comparar el tiempo medio por respuesta entre ambos modelos ayuda a determinar si el modelo ajustado mantiene un rendimiento eficiente o si, por el contrario, introduce una sobrecarga significativa.

Tiempo promedio por respuesta - Modelo Base [s]	**Tiempo promedio por respuesta - Modelo Ajustado [s]**
0.5909	1.8692

Tabla 4.2: Tiempos de inferencia.

Los resultados muestran un incremento claro en el tiempo promedio empleado en la respuesta tras el *fine-tuning*, pasando a valer casi el triple, de 0.59 a 1.87 segundos. Este aumento era esperable, ya que con la técnica LoRA se introducen matrices adicionales entrenables dentro de las capas del modelo base, que incrementan ligeramente el tiempo de procesamiento. No obstante, pese a este incremento, los tiempos siguen siendo razonables para tareas que no requieren una

respuesta inmediata, lo que sugiere que el modelo ajustado puede utilizarse en una gran mayoría de contextos prácticos.

En segundo lugar, se ha registrado el consumo de memoria en la inferencia, monitorizando cuánta memoria de GPU utiliza cada modelo durante la generación. Para ello, se mide la VRAM disponible justo antes y justo después del proceso de inferencia. Esta comparación permite observar posibles diferencias en el uso de recursos entre el modelo base y el ajustado. Con esta métrica se pretende evaluar si el nuevo modelo es más exigente computacionalmente o si, por el contrario, mantiene un uso de memoria similar al modelo original, lo cual sería deseable para su aplicación práctica en entornos con recursos limitados.

Memoria promedio por respuesta - Modelo Base [MB]	**Memoria promedio por respuesta - Modelo Ajustado [MB]**
0.04	0.11

Tabla 4.3: Consumos de memoria en inferencia.

En cuanto al consumo de memoria durante la inferencia, se observa también un ligero aumento en el modelo ajustado, que pasa de 0.04 MB a 0.11 MB por respuesta. Aunque este incremento representa casi el triple en términos relativos, el valor absoluto sigue siendo muy bajo. Este resultado indica que el modelo ajustado mantiene un perfil eficiente en términos de uso de memoria, y que su despliegue sigue siendo viable en entornos con recursos computacionales limitados.

4.2.3. Evaluación de la precisión numérica

En las preguntas realizadas durante la inferencia, se ha garantizado que no se formulen preguntas sobre datos faltantes NaN (del inglés, *Not a number*), ya que no tiene sentido evaluar la predicción de un valor inexistente: se busca comparar la respuesta del modelo con un valor real que haya ocurrido. La presencia de valores nulos en este tipo de *dataset* puede deberse a que no todas las estaciones meteorológicas registran todas las variables en todas las semanas, ya sea por limitaciones

técnicas o fallos puntuales en los sensores.

Los archivos JSON generados en la etapa previa de inferencia, que contienen las respuestas de ambos modelos, se cargan en el *notebook* independiente de evaluación, donde se validan mediante diferentes métricas que permiten analizar la precisión numérica:

- ***FactScore***: mide la exactitud factual de respuestas generadas a partir de datos reales. Dado que las respuestas son valores numéricos asociados a magnitudes físicas, se establece un umbral de tolerancia de ±0,5 unidades. Si la diferencia entre la respuesta generada y el valor real no supera dicho umbral, se considera que la predicción es correcta.
- **Detección de alucinaciones**: consiste en identificar respuestas que se alejan excesivamente del valor real, indicando una posible invención por parte del modelo. En este trabajo, se considera que una respuesta constituye una alucinación si difiere en más de 5 unidades respecto al valor esperado.
- **Métricas estándar de regresión**: se calculan tres métricas comunes en tareas de predicción numérica: el error absoluto medio (MAE, del inglés *Mean Absolute Error*), el error cuadrático medio (MSE, del inglés *Mean Squared Error*) y el coeficiente de determinación (R^2). Estas métricas permiten cuantificar la magnitud media del error de forma continua, complementando así la evaluación binaria anterior basada en umbrales. Mientras el MAE refleja el error promedio sin penalizar desvíos grandes, el MSE penaliza errores más amplios al elevar las diferencias al cuadrado, lo que permite identificar más fácilmente outliers o desviaciones importantes. Asimismo, el R^2 mide la proporción de varianza explicada por el modelo respecto al valor real, pudiendo evaluar así la capacidad predictiva global del sistema.

Todas estas métricas se computan para el modelo base preentrenado y para el modelo ajustado.

Modelo	Fact Score	Detección de alucinaciones	MAE	MSE	R^2
Modelo Base	43.7 %	21.5 %	3.1	10.1	0.67
Modelo Ajustado	56.2 %	9.6 %	2.4	6.8	0.79

Tabla 4.4: Comparación de métricas entre el modelo base y el modelo ajustado

En la tabla 4.4 se recogen todas las métricas utilizadas para la evaluación de la calidad de las respuestas generadas en términos de precisión numérica. Este formato permite una comparación directa entre el modelo base y el modelo ajustado, facilitando la interpretación final de los resultados. En líneas generales, se aprecia una mejora consistente del modelo ajustado con respecto al base, aunque esta mejora no es excesiva, y para ninguno de los modelos se puede afirmar que se alcance un rendimiento excepcional en esta tarea.

En concreto, destaca la baja puntuación obtenida en el *Fact Score*, con valores cercanos o incluso inferiores al 50 % en ambos modelos. Este resultado no parece muy coherente con el del resto de métricas, más razonables, pero se explica por el criterio de evaluación utilizado de solo permitir una desviación respecto del valor real de 0.5 unidades. Aunque se trata de un umbral muy exigente, fue escogido así de manera intencional para identificar cuántas predicciones alcanzaban una precisión numérica muy alta. Esta configuración tan restrictiva, que penaliza desviaciones leves, explica que incluso el modelo ajustado, a pesar de presentar una mejoría, no obtenga puntuaciones elevadas en esta métrica.

En cuanto al porcentaje de alucinaciones promedio, los valores son más razonables, ya que el umbral empleado fue más amplio (5 unidades físicas), permitiendo detectar solo desviaciones grandes. Como era esperable, aunque el modelo ajustado siga alucinando, lo hace en menor medida que el base. Por ejemplo, Llama 3.1 en ocasiones generaba respuestas vacías o sin contenido numérico reconocible, mientras que el modelo ajustado tendía a generar valores numéricos mal formateados o incompletos (por ejemplo, una respuesta parcial como “0.”). Es decir, aunque el

modelo ajustado también alucinada, la respuesta suele iniciar siguiendo el formato numérico esperado, lo que sugiere cierta alineación con la tarea aunque no logre completarla correctamente.

Por último, las tres métricas estándar de regresión (MAE, MSE y R^2) respaldan la conclusión principal de que el modelo ajustado ha mejorado respecto al modelo base, aunque todavía no alcanza niveles de precisión altos. El error absoluto medio (MAE) presenta valores razonables para el tipo de magnitudes tratadas (temperaturas, precipitaciones o velocidades de viento), lo que indica que ambos modelos se encuentran relativamente cerca del valor correcto. En cambio, el error cuadrático medio (MSE) es más elevado, especialmente para el modelo base, lo que sugiere que los errores grandes tienen un impacto importante, aunque sean menos frecuentes, ya que el MSE penaliza con mayor severidad las desviaciones amplias. Finalmente, el coeficiente de determinación R^2, que evalúa la proporción de variabilidad explicada por el modelo, muestra una mejora clara tras el *fine-tuning*. Aunque ninguno de los modelos supera el umbral de 0.8, el valor obtenido por el modelo ajustado indica que entiende mejor los patrones y tendencias de los datos, y demuestra que el proceso de ajuste ha tenido un impacto positivo.

Capítulo 5

Conclusiones

En este capítulo se resumen los resultados globales del trabajo, así como las principales dificultades encontradas durante el desarrollo y las conclusiones finales en cuanto a su escalabilidad e impacto.

5.1. Resumen de objetivos y hallazgos

Como punto de partida, se puede afirmar que se han cumplido los cuatro objetvios establecidos en el primer capítulo de la memoria:

1. Aplicar técnicas de ajuste fino, como *Low-Rank Adaptation* (LoRA), para optimizar un modelo de lenguaje en entornos con restricciones de hardware.
2. Ajustar un LLM para una tarea específica de preguntas y respuestas (Q&A) sobre un corpus determinado.
3. Evaluar la eficiencia y el rendimiento del modelo antes y después del fine-tuning.

4. Identificar posibles mejoras para desarrollos de proyectos similares.

El cumplimiento de estos objetivos confirma que la idea planteada inicialmente ha podido llevarse a cabo de manera coherente, respetando tanto el enfoque previsto como las fases de desarrollo establecidas.

Como resumen de los hallazgos, se presenta la tabla 5.1, que recoge tanto las métricas generales del proceso de entrenamiento como los resultados en las métricas de evaluación tras el ajsute fino:

Métricas generales del entrenamiento	**Valor**
Curva de pérdidas (entrenamiento)	Valor más bajo: 0.0135
Curva de pérdidas (validación)	Valor más bajo: 0.024919
Consumo energético medio del entrenamiento	260.61 W
Máximo uso de memoria GPU	8753 MB

Métricas de evaluación del modelo	**Modelo base**	**Modelo ajustado**
Tiempo promedio por respuesta	0.5909 s	1.8692 s
Consumo de memoria promedio por respuesta	0.04 MB	0.11 MB
Fact Score	43.7 %	56.2 %
Detección de alucinaciones	21.5 %	9.6 %
MAE	3.1	2.4
MSE	10.1	6.8
R^2	0.67	0.79

Tabla 5.1: Resumen de resultados del entrenamiento y comparación entre modelo base y ajustado

A pesar de que el entrenamiento no pudo completarse con todas las épocas inicialmente previstas, las curvas de pérdida para el conjunto de entrenamiento y validación muestran una evolución paralela y sin indicios claros de sobreajuste o subajuste. Esto sugiere una convergencia adecuada del modelo.

Los resultados numéricos respaldan esta observación, ya que el modelo ajustado muestra mejoras consistenes en la calidad de las respuestas frente al modelo base, tanto en términos de precision (menores valores de MAE y MSE) como de coherencia (mayor R^2 y menor detección de alucinaciones). Aunque la exactitud global sigue siendo baja, debe tenerse en cuenta el criterio de evaluación utilizado, como se explica en el capítulo 4.

En cuanto al tiempo de inferencia, se observa que el modelo ajustado tarda aproximadamente el triple que el preentrenado, lo cual representa un incremento notable considerando que las respuestas generadas son exclusivamente numéricas. Aun así, un tiempo cercano a los dos segundos por respuesta sigue siendo aceptable en el contexto de una herramienta privada, especialmente cuando se prioriza la precisión sobre la velocidad.

Respecto al consumo de recursos, el entrenamiento no alcanzó el límite térmico de la tarjeta A100 (de 400 W), y tanto el uso máximo de memoria como el consumo energético medio se mantuvieron en valores razonables. Además, el bajo consumo de memoria observado durante la inferencia para ambos modelos refuerza la viabilidad de su despliegue en entornos con recursos computacionales limitados.

5.2. Limitaciones del trabajo

A lo largo del proyecto se han identificado diversas limitaciones que han condicionado el desarrollo y los resultados obtenidos. Una de las más importantes ha sido la infraestructura del código. A pesar de contar con la suscripción de Google Colab Pro, las sesiones seguían estando sujetas a restricciones de tiempo (máximo 24 horas) y desconexiones por inactividad. Además, la asignación de recursos en esta plataforma es dinámica y depende de la disponibilidad en el momento de iniciar sesión. Aunque la suscripción Pro otorga prioridad, en varias ocasiones se asignó una GPU T4, cuya capacidad resultó insuficiente para las necesidades del proyecto. Esta variabilidad dificultó la planificación del entrenamiento y obligó a

interrumpir o repetir procesos en varias ocasiones.

Otra dificultad asociada al uso de la nube de Google es que no es un entorno estático, sino que las dependencias instaladas o configuraciones funcionales en una sesión podían dejar de funcionar en la siguiente, debido a que cada sesión de Colab se ejecuta sobre una máquina virtual nueva. Esto implica que librerías, versiones o rutas del sistema pueden cambiar sin previo aviso, lo que requiere tiempo adicional para reconfigurar o depurar errores inesperados.

En cuanto al *dataset*, aunque inicialmente se seleccionó por su especificidad y volumen de datos, resultó ser mucho más complejo de adaptar de lo previsto. Fue necesario aplicar múltiples etapas de filtrado y limpieza para reducir su tamaño y hacerlo manejable en el entrenamiento. Sin embargo, la mayor dificultad fue la integración con el modelo de lenguaje, que se complicó significativamente por el hecho de tratarse de un conjunto de datos estrictamente numérico. Esto no se identificó como una incompatibilidad crítica hasta que los resultados comenzaron a mostrar incoherencias. Esto provocó que una parte considerable del tiempo total del proyecto se destinara a adaptar el *dataset* y diseñar un *prompt* adecuado que permitiera al modelo aprender de forma estructurada.

Por otro lado, se eligió el modelo base Llama 3.1 8B inicialmente por dar continuidad a un trabajo anterior desarrollado con la universidad. Aunque esta elección está justificada por su potencia y disponibilidad abierta en el apartado 3.4, en retrospectiva, podría haber sido más eficiente partir de un modelo previamente entrenado en tareas meteorológicas, como *GraphCast* de Google. Como se menciona en el capítulo 2, estos modelos a menudo son adaptados para personalizarlos a ciertas regiones o fenómenos climáticos en concreto.

En lo que respecta a la evaluación, algunas métricas como el *FactScore* pueden haber penalizado al modelo más de lo necesario. En tareas numéricas, pequeñas desviaciones en los resultados pueden implicar resultados insuficientes, a pesar de que el modelo haya captado correctamente patrones y tendencias en los datos.

Finalmente, otra limitación significativa fue la imposibilidad de realizar un entrenamiento con más *epochs* que las permitidas por el mecanismo de `early_stopping`. Dado el tamaño del *dataset* y las restricciones computacionales, no fue viable prolongar el entrenamiento hasta alcanzar la estabilización completa de las curvas de pérdida. No obstante, en el proyecto de TFM asociado, donde se va a trabajar con un conjunto de datos considerablemente más reducido, se espera que este aspecto pueda mejorarse y ofrecer resultados más estables.

5.3. Conclusiones finales e impacto

A la vista de los resultados obtenidos y comentados a lo largo de este capítulo, puede afirmarse que se ha cumplido el objetivo princial de este trabajo: demostrar la viabilidad del ajuste fino de un modelo de lenguaje orientado al manejo eficiente de información privada. Aunque en este caso se haya empleado un conjunto de datos públicos, este se ha gestionado en todo momento desde un repositorio privado interno, lo que permite validar que el flujo de trabajo propuesto es realmente reproducible con datos confidenciales, como se plantea en el proyecto asociado con el otro TFM.

Sin embargo, cabe destacar que, en el caso de un entorno corporativo real, las consideraciones en cuanto a privacidad y seguridad serían considerablemente más estrictas, tanto durante la recopilación y preprocesamiento de los datos como en el posterior despliegue del modelo o herramienta. Pese a ello, este trabajo ha demostrado que es posible llevar a cabo un proceso completo de *fine-tuning* en condiciones de infraestructura limitada, lo que abre la puerta a futuras aplicaciones en organizaciones pequeñas o con escasos recursos computacionales.

Capítulo 6

Trabajos Futuros

6.1. Posibles mejoras en el ajuste fino

Durante la realización del proyetco, han surgido diversas ideas que, de repetirse el proceso, permitirían optimizar tanto el desarrollo como los resultados obtenidos. Estas mejoras no solo responden a problemas concretos encontrados, sino también a aprendizajes adquiridos durante el trabajo.

Una de las primeras decisiones que habría sido útil replantear es la elección de un corpus de entrenamiento tan amplio desde el principio. En este proyecto no se generó una versión reducida del conjunto de datos para pruebas iniciales, lo que dificultó realizar validaciones rápidas y provocó que cualquier cambio en el flujo completo implicara tiempos de espera prolongados. En retrospectiva, habría sido conveniente definir un entorno de pruebas más ligero, en paralelo al flujo principal, para facilitar los ajustes iterativos.

Otro aspecto mejorable, aprendido a raíz de la experiencia en la etapa de inferencia, es la implementación de guardados intermedios durante el entrenamiento.

Disponer de *checkpoints* periódicos de los pesos del modelo habría permitido retomar el proceso en caso de desconexión de Google Colab, evitando así tener que reiniciar el entrenamiento. Esta estrategia ya se aplicó en la inferencia, mediante el guardado automático cada 10 resupuestas generadas, y podría adaptarse al proceso de *fine-tuning*.

Además, con más tiempo y recursos sería deseable continuar perfeccionando el entrenamiento mediante el ajuste de hiperparámetros, mayor número de épocas, y posibles mejoras en la arquitectura o configuración del optimizador.

Finalmente, como se comentó en el capítulo 2, existen modelos de lenguaje diseñados específicamente para fenómenos meteorológios o regiones concretas. En el futuro, podría explorarse una personalización del modelo para el contexto climático español, incorporando fenómenos extremos de estos últimos años como Filomena o la Dana, que no solo enriquecerían el conjunto de entrenamiento sino que también permitirían evaluar la capacidad del modelo para adaptarse a eventos locales complejos.

6.2. Ampliaciones del enfoque actual

Además de las mejoras técnicas planteadas anteriormente, existen distintas líneas que permitirían ampliar el enfoque actual del trabajo, tanto en términos de evaluación como de diseño del entrenamiento.

En la evaluación principal realizada en este trabajo, se generaron exclusivamente preguntas para las cuales existía un valor real en el *dataset*, lo que garantizaba la posibilidad de comparar las respuestas del modelo con datos verídicos. No obstante, en un entorno real, el modelo podría enfrentarse a situaciones en las que no se disponga de ciertos registros, como ocurriría en el caso de una empresa con datos incompletos o usuarios que consulten información que aún no ha sido recolectada. Por este motivo, una posible línea de trabajo futuro sería evaluar el comporta-

miento del modelo ante preguntas formuladas sobre datos incompletos o ausentes, analizando cómo responde en condiciones de incertidumbre o cómo gestiona las lagunas de información.

Por otra parte, aunque en este trabajo se ha respetado la dimensión temporal al entrenar con datos históricos (2014–2022) y evaluar exclusivamente sobre años posteriores (2023–2024), el modelo no ha sido expuesto a los ejemplos en orden cronológico secuencial. Este enfoque parte del supuesto de que las muestras semanales son independientes entre sí, algo habitual en muchos procedimientos de *fine-tuning* sobre LLMs. Como propuesta de ampliación, podría explorarse el impacto de incorporar dicha secuencia temporal de forma explícita en el entrenamiento. Esto podría abordarse mediante técnicas como la segmentación por periodos o el entrenamiento progresivo por años. Integrar esta dimensión permitiría estudiar si el modelo mejora su capacidad para capturar patrones estacionales, tendencias meteorológicas o evoluciones temporales complejas.

Por último, también podría ampliarse el enfoque hacia un formato conversacional de pregunta-respuesta, incluyendo interacciones multivuelta. Este tipo de estructura no se ha aplicado en el presente trabajo por la simplicidad del *dataset* y la naturaleza sencilla de las preguntas, pero sería una evolución lógica para tareas en las que el usuario interactúe de forma continuada con el modelo y requiera respuestas encadenadas o más contextuales.

6.3. Escalabilidad y aplicación en entornos empresariales

Como se ha comentado en el capítulo 5, los resultados obtenidos demuestran la viabilidad de aplicar técnicas de ajuste fino en modelos de lenguaje para entornos empresariales que gestionen información privada. El flujo de trabajo diseñado en este proyecto presenta una estructura modular que favorece su escalabilidad: las etapas de preprocesamiento, limpieza y formateo del dataset están claramente separadas del proceso de entrenamiento, lo que permite adaptar el sistema a nuevos dominios introduciendo únicamente los datos específicos de cada organización. Esta lógica similar a un sistema *plug-and-play* evita la necesidad de conectar el modelo a bases de datos externas durante la inferencia, lo que resulta esencual en escenarios con altos requerimientos de privacidad.

Aun así, debe señalarse que una de las principales complejidades del proceso ha sido precisamente el encaje entre el *dataset* y la estructura del entrenamiento. Dado que se trata de un modelo de lenguaje ajustado con datos numéricos, ha sido necesario definir cuidadosamente el *prompt* de entrada y separar explícitamente las variables objetivo, lo que introduce una dependencia del tipo y formato de datos. Por tanto, este componente requerirá una personalización adhoc en cada caso concreto. Sin embargo, para sectores con datos estructurados similares entre organizaciones (por ejemplo, compañías del sector energético, meteorológico o financiero), podría desarrollarse una plantilla reutilizable adaptada al dominio específico.

Otro aspecto a tener en cuenta para la escalabilidad es la accesibilidad del sistema para organizaciones pequeñas o usuarios sin experiencia en inteligencia artificial. Actualmente, el desarrollo se ha llevado a cabo íntegramente mediante *notebooks* en Python, lo que requiere conocimientos técnicos para su ejecución. Un paso clave y fundamental hacia la aplicación práctica sería la paquetización de todo el flujo en forma de interfaz o asistente. Esta herramienta podría integrarse

con un directorio inicial donde el usuario cargaría sus datos, y posteriormente se ejecutarían de forma automática las fases de preprocesamiento, entrenamiento e inferencia, sin necesidad de conocimientos técnicos avanzados por parte del usuario final.

Finalmente, en sectores dinámicos o con alta rotación de la información (como el científico-tecnológico), podría plantearse una solución híbrida que combine el modelo ajustado para el dominio específico con un sistema RAG (*Retrieval-Augmented Generation*) que incorpore los documentos con información más reciente. Como se comentó en la tabla comparativa entre RAG y *Fine-Tuning* 2.3, esta combinación permitiría mantener la precisión alcanzada mediante el ajuste fino sin renunciar a la actualización continua de información, lo que ofrece un equilibrio ideal entre fiabilidad, adaptabilidad y escalabilidad.

Bibliografía

[1] Mathav Raj J et al. *Fine Tuning LLM for Enterprise: Practical Guidelines and Recommendations*. 23 de mar. de 2024. arXiv: 2404.10779. URL: http://arxiv.org/abs/2404.10779 (visitado 07-11-2024).

[2] Venkatesh Balavadhani Parthasarathy et al. *The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities*. version: 1. 23 de ago. de 2024. arXiv: 2408.13296. URL: http://arxiv.org/abs/2408.13296 (visitado 07-11-2024).

[3] Rajvardhan Patil y Venkat Gudivada. «A Review of Current Trends, Techniques, and Challenges in Large Language Models (LLMs)». En: *Applied Sciences* 14.5 (1 de mar. de 2024), pág. 2074. ISSN: 2076-3417. DOI: 10.3390/app14052074. URL: https://www.mdpi.com/2076-3417/14/5/2074 (visitado 07-11-2024).

[4] Zhiqiang Hu et al. *LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models*. 9 de oct. de 2023. arXiv: 2304.01933. URL: http://arxiv.org/abs/2304.01933 (visitado 09-11-2024).

[5] Jiawei Zheng et al. *Fine-tuning Large Language Models for Domain-specific Machine Translation*. 23 de feb. de 2024. arXiv: 2402.15061[cs]. URL: http://arxiv.org/abs/2402.15061 (visitado 07-11-2024).

[6] Zhiwei Yao et al. *Efficient Deployment of Large Language Models on Resource-constrained Devices*. 5 de ene. de 2025. DOI: 10.48550/arXiv.2501.02438.

arXiv: 2501.02438[cs]. URL: http://arxiv.org/abs/2501.02438 (visitado 08-06-2025).

[7] Tim Genewein et al. *Understanding Prompt Tuning and In-Context Learning via Meta-Learning*. 22 de mayo de 2025. DOI: 10.48550/arXiv.2505.17010. arXiv: 2505.17010[cs]. URL: http://arxiv.org/abs/2505.17010 (visitado 29-06-2025).

[8] Jeremy Stephen Gabriel Yee et al. *On-Device LLMs for SMEs: Challenges and Opportunities*. 22 de oct. de 2024. DOI: 10.48550/arXiv.2410.16070. arXiv: 2410.16070[cs]. URL: http://arxiv.org/abs/2410.16070 (visitado 08-06-2025).

[9] Angels Balaguer et al. *RAG vs Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture*. 30 de ene. de 2024. DOI: 10.48550/arXiv.2401.08406. arXiv: 2401.08406[cs]. URL: http://arxiv.org/abs/2401.08406 (visitado 08-06-2025).

[10] Qinbin Li et al. *LLM-PBE: Assessing Data Privacy in Large Language Models*. 6 de sep. de 2024. arXiv: 2408.12787[cs]. URL: http://arxiv.org/abs/2408.12787 (visitado 07-11-2024).

[11] Zachary Charles et al. *Fine-Tuning Large Language Models with User-Level Differential Privacy*. 10 de jul. de 2024. arXiv: 2407.07737[cs]. URL: http://arxiv.org/abs/2407.07737 (visitado 07-11-2024).

[12] Badhan Chandra Das, M. Hadi Amini y Yanzhao Wu. *Security and Privacy Challenges of Large Language Models: A Survey*. 18 de nov. de 2024. DOI: 10.48550/arXiv.2402.00888. arXiv: 2402.00888[cs]. URL: http://arxiv.org/abs/2402.00888 (visitado 18-06-2025).

[13] *Boletín Semanal de Ciberseguridad, 7-13 junio*. URL: https://telefonicatech.com/blog/boletin-ciberseguridad-7-13-junio-2025 (visitado 18-06-2025).

[14] Guangji Bai et al. *Beyond Efficiency: A Systematic Survey of Resource-Efficient Large Language Models*. 29 de dic. de 2024. DOI: 10.48550/arXiv.2401.00625. arXiv: 2401.00625[cs]. URL: http://arxiv.org/abs/2401.00625 (visitado 14-06-2025).

[15] Zhengqing Yuan et al. *EfficientLLM: Efficiency in Large Language Models*. 20 de mayo de 2025. DOI: 10.48550/arXiv.2505.13840. arXiv: 2505.13840[cs]. URL: http://arxiv.org/abs/2505.13840 (visitado 14-06-2025).

[16] Xinyin Ma, Gongfan Fang y Xinchao Wang. «LLM-Pruner: On the Structural Pruning of Large Language Models». En: ().

[17] Peijie Dong et al. *Can Compressed LLMs Truly Act? An Empirical Evaluation of Agentic Capabilities in LLM Compression*. 1 de jun. de 2025. DOI: 10.48550/arXiv.2505.19433. arXiv: 2505.19433[cs]. URL: http://arxiv.org/abs/2505.19433 (visitado 08-06-2025).

[18] *Mosaic AI Model Training*. Databricks. URL: https://www.databricks.com/product/machine-learning/mosaic-ai-training (visitado 07-06-2025).

[19] *Fine-tuning*. OpenAI API. URL: https://platform.openai.com/docs/guides/fine-tuning (visitado 07-06-2025).

[20] *Customized AI Solutions | Secure and Scalable*. Cohere. URL: https://cohere.com/customization (visitado 07-06-2025).

[21] *IA Generativa Privada*. Sherpa.ai. URL: https://sherpa.ai/es/ia-generativa/ (visitado 07-06-2025).

[22] *IA para Ayuntamientos*. OpenSistemas. URL: https://opensistemas.com/ia-para-ayuntamientos-e-instituciones/ (visitado 07-06-2025).

[23] *SOLUCIÓN DE CODIFICACIÓN CLÍNICA MEDIANTE PROCESAMIENTO DEL LENGUAJE NATURAL (PLN). PLAN DE RECUPERACIÓN, TRANSFORMACIÓN Y RESILIENCIA – FINANCIADO POR LA UNIÓN EUROPEA – NEXTGENERATIONEU*. Pliego de prescripciones técnicas. Subdirección General de Tecnologías de la Información, 2024.

[24] *SERVICIOS PARA EL ENRIQUECIMIENTO DE MODELOS DE LENGUAJE CON GRAFOS DE CONOCIMIENTO PARA LA ETSI INFORMÁTICOS DE LA UPM*. Valoración de las ofertas. Ministerio de Asuntos Económicos y Transformación Digital, 2023.

[25] *Inside CodeGen, Our In-House Open-Source LLM*. Salesforce Developers Blog. URL: https://developer.salesforce.com/blogs/2023/11/inside-codegen-our-in-house-open-source-llm (visitado 07-06-2025).

[26] Kai Zhang et al. «BiomedGPT: A generalist vision–language foundation model for diverse biomedical tasks». En: ().

[27] Anonymous. «Finetuning Weather Foundation Models to Develop Climate Model Parameterizations». En: ICLR. Singapore, 2025.

[28] Christopher Subich. *Efficient fine-tuning of 37-level GraphCast with the Canadian global deterministic analysis*. 25 de abr. de 2025. DOI: 10.48550/arXiv.2408.14587. arXiv: 2408.14587[cs]. URL: http://arxiv.org/abs/2408.14587 (visitado 07-06-2025).

[29] Anna Allen et al. «End-to-end data-driven weather prediction». En: *Nature* 641.8065 (29 de mayo de 2025), págs. 1172-1179. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-025-08897-0. URL: https://www.nature.com/articles/s41586-025-08897-0 (visitado 07-06-2025).

[30] Eduardo C. Garrido-Merchán, Cristina González-Barthe y María Coronado Vaca. *Fine-tuning ClimateBert transformer with ClimaText for the disclosure analysis of climate-related financial risks*. 21 de mar. de 2023. DOI: 10.48550/arXiv.2303.13373. arXiv: 2303.13373[cs]. URL: http://arxiv.org/abs/2303.13373 (visitado 07-06-2025).

[31] *GPU NVIDIA Tesla T4 con núcleos Tensor para inferencias de IA*. NVIDIA Data Center. URL: https://www.nvidia.com/es-es/data-center/tesla-t4/ (visitado 07-06-2025).

[32] *NVIDIA A100*. NVIDIA Data Center. URL: https://www.nvidia.com/es-es/data-center/a100/ (visitado 07-06-2025).

[33] CondingMindset. *Fine-tuning-llama3.1 at main*. GitHub. URL: https://github.com/codingmindset/CodingMindset-YouTube/tree/main/fine-tuning-llama3.1 (visitado 07-06-2025).

[34] Matthew J. Menne et al. «An Overview of the Global Historical Climatology Network-Daily Database». En: *Journal of Atmospheric and Oceanic Technology* 29.7 (1 de jul. de 2012), págs. 897-910. ISSN: 0739-0572, 1520-0426. DOI: 10.1175/JTECH-D-11-00103.1. URL: http://journals.ametsoc.org/doi/10.1175/JTECH-D-11-00103.1 (visitado 07-06-2025).

[35] *Conjuntos de datos: división del conjunto de datos original*. Google Crash Course: Machine Learning. URL: https://developers.google.com/machine-learning/crash-course/overfitting/dividing-datasets?hl=es-419 (visitado 07-06-2025).

[36] *Introducing Llama 3.1: Our most capable models to date*. Meta AI. URL: https://ai.meta.com/blog/meta-llama-3-1/ (visitado 07-06-2025).

[37] *A Challenging, Contamination-Free LLM Benchmark*. LiveBench. URL: https://livebench.ai/#/ (visitado 07-06-2025).

[38] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 24 de mayo de 2019. DOI: 10.48550/arXiv.1810.04805. arXiv: 1810.04805[cs]. URL: http://arxiv.org/abs/1810.04805 (visitado 07-06-2025).

[39] *no_grad context manager*. PyTorch 2.7 documentation. URL: https://docs.pytorch.org/docs/stable/generated/torch.no_grad.html (visitado 07-06-2025).

[40] Jia He et al. *Does Prompt Formatting Have Any Impact on LLM Performance?* 15 de nov. de 2024. DOI: 10.48550/arXiv.2411.10541. arXiv: 2411.10541[cs]. URL: http://arxiv.org/abs/2411.10541 (visitado 07-06-2025).

[41] Melanie Sclar et al. «QUANTIFYING LANGUAGE MODELS' SENSITIVITY TO SPURIOUS FEATURES IN PROMPT DESIGN or:» en: (2024).

Apéndice Anexo A

Contribución a los Objetivos de Desarrollo Sostenible (ODS)

Este proyecto contribuye de manera significativa a los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030, especialmente en relación con los siguientes:

- **ODS 8: Trabajo decente y crecimiento económico.** Este trabajo reduce la dependencia de soluciones costosas en la nube, haciendo accesible los LLMs para empresas pequeñas y medianas, fomentando así la adopción de tecnologías avanzadas y el crecimiento económico.
- **ODS 9: Industria, Innovación e Infraestructura.** Se promueve el uso de inteligencia artificial en entornos con limitaciones de hardware a través de técnicas PEFT. Apoya a organizaciones con infraestructura limitada en la incorporación de tecnología innovadora.
- **ODS 10: Reducción de las Desigualdades** El proyecto facilita la implementación de modelos de lenguaje on-premise, reduciendo barreras económicas y promoviendo una inclusión tecnológica amplia.

- **ODS 16: Paz, Justicia e Instituciones Sólidas.** En un contexto donde la protección de datos es crucial, se contribuye a la confidencialidad de la información mediante el uso de LLMs en sistemas locales.

Figura Anexo A.1: Alineación con los ODS