



Escuela
Politécnica
Superior

Sistema de IA generativa para lectura facilitada



Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Nizar Nortes Pastor

Tutores:

José Ignacio Abreu Salas

Rafael Muñoz Guillena

Julio 2025



Universitat d'Alacant
Universidad de Alicante

Sistema de IA generativa para lectura facilitada

Autor

Nizar Nortes Pastor

Tutores

José Ignacio Abreu Salas

Lenguajes y sistemas informáticos

Rafael Muñoz Guillena

Lenguajes y sistemas informáticos



Grado en Ingeniería Informática



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2025

Preámbulo

Este proyecto consiste en el estudio de las capacidades de los grandes modelos de lenguaje en las tareas de simplificación de texto, además del desarrollo de una solución de Inteligencia Artificial (IA) generativa para la adaptación de texto a lectura facilitada. Todo esto utilizando un conjunto de pautas establecidas para facilitar la comprensión lectora a personas con dificultades en la lectura, las normas de Lectura Fácil.

El objetivo final de esta investigación es medir la eficacia de los modelos generativos de lenguaje en la producción de textos accesibles, y explorar distintas estrategias relacionadas con el Procesamiento del Lenguaje Natural y el Aprendizaje Automático, además de la evaluación automática de los resultados.

Agradecimientos

En primer lugar, este trabajo no habría sido siquiera posible sin la ayuda y dirección de mi tutor, José Abreu Salas, bajo cuya supervisión escogí y desarrollé el tema a estudiar. Él ha sido quien me ha guiado durante la realización de este trabajo.

También quiero agradecer a mi tutor Rafael Muñoz Guillena, y en general a todo el Grupo de Procesamiento del Lenguaje y Sistemas de Información (GPLSI), por haberme permitido realizar las prácticas externas en sus instalaciones, por su apoyo y por la ayuda que me han brindado en la elaboración de este trabajo.

Finalmente, quiero mostrar mi gratitud a mi familia, a mis amigos de toda la vida y a los amigos que he conocido durante la carrera, quienes han sido mi mayor apoyo a lo largo de estos años. Sin ellos no habría sido capaz de llegar hasta aquí.

“No todos los que vagan están perdidos”

J.R.R. Tolkien.

Índice general

1	Introducción	1
2	Marco Teórico	3
2.1	Lectura Fácil	3
2.1.1	Pautas de lectura facilitada	4
2.1.1.1	Pautas de ortotipografía	4
2.1.1.2	Pautas de vocabulario	4
2.1.1.3	Pautas de oraciones	5
2.2	Machine Learning	6
2.3	Procesamiento del Lenguaje Natural	7
2.4	Modelos de Lenguaje	7
2.4.1	Transformers	7
2.4.2	Ingeniería de Prompts y Ajuste Fino	8
2.5	Métricas de Evaluación	8
2.5.1	BLEU	8
2.5.2	ROUGE	9
2.5.3	METEOR	9
2.5.4	BERTScore	10
2.5.5	Flesch Reading Ease	11
2.6	GAN	11
2.7	LoRA	12
2.8	API REST	12
3	Objetivos	15
4	Metodología	17
4.1	Plataformas	17
4.1.1	Git y GitHub	17
4.1.2	Hugging Face	17
4.2	Entornos de Desarrollo	18
4.2.1	Google Colab	18
4.2.2	Visual Studio Code	18
4.3	Lenguajes y Librerías	19
4.3.1	HTML, CSS y JavaScript	19
4.3.2	Python	19
4.3.3	Jupyter Notebook	20
4.3.4	JSON	20
4.3.5	NumPy	21
4.3.6	Natural Language Toolkit	21

4.3.7	PyTorch	21
4.3.8	FastAPI y Uvicorn	22
4.4	Corpus ClearSim	22
4.5	LLMs	22
4.5.1	LLaMA 3.1 8B Instruct	23
4.5.2	Mistral 8B Instruct	23
4.5.3	Phi 3 Small 128K Instruct	23
4.5.4	Qwen 2.5 7B Instruct	23
4.5.5	DeepSeek R1 Distill Qwen 3 8B	24
5	Desarrollo	25
5.1	Inferencia de modelos	25
5.2	Evaluación de modelos	26
5.3	GAN	27
5.4	Ajuste fino	29
5.5	Capa de servicios y Prototipo	30
6	Resultados	33
6.1	Estudio de LLMs	33
6.2	Estudio de sistemas propios	35
6.2.1	GAN	35
6.2.2	Ajuste fino	37
6.3	Prototipo	39
7	Conclusiones	41
7.1	Objetivos cumplidos	41
7.2	Dificultades encontradas	41
7.3	Mejoras a futuro	42
7.4	Reflexiones finales	42
	Bibliografía	43
	Lista de Acrónimos y Abreviaturas	45

Índice de figuras

5.1	Diagrama de la arquitectura del sistema de GAN.	27
6.1	Comparativa de resultados de LLMs.	34
6.2	Comparativa de resultados del modelo de GAN con LLMs.	36
6.3	Comparativa de resultados del modelo de ajuste fino con LLMs.	37
6.4	Interfaz del prototipo de la aplicación.	39

Índice de tablas

4.1	Especificaciones del entorno Colab con GPU Tesla T4.	18
4.2	Especificaciones del entorno Colab Pro con GPU A100.	19
4.3	Estructura del archivo JSON utilizado.	22
5.1	Estructura del archivo JSON generado.	26
5.2	Parámetros de configuración QLoRA.	30
6.1	Resultados de evaluación BLEU.	33
6.2	Resultados de evaluación ROUGE.	33
6.3	Resultados de evaluación METEOR.	34
6.4	Resultados de evaluación BERTScore (F1).	34
6.5	Ejemplos de entrada y salida del modelo basado en GAN.	36
6.6	Resultados de evaluación del modelo de ajuste fino.	37
6.7	Ejemplos de entrada y salida del modelo de ajuste fino.	38

1 Introducción

La comprensión lectora es una habilidad fundamental para el acceso a la información, a la educación y para la participación social. A pesar de esto, hay un gran número de personas que encuentran problemas a la hora de entender textos escritos, ya sea por razones cognitivas, educativas o sociales. Entre estas, se incluyen personas con discapacidad intelectual, con dificultades de aprendizaje, personas mayores, personas inmigrantes con poco dominio del idioma y otros colectivos que se pueden ver afectados de forma similar.

Para solucionar esto surge la iniciativa de la Lectura Fácil, un método que promueve la creación de documentos accesibles mediante una serie de pautas concretas de redacción, maquetación, diseño y validación. Estas pautas están redactadas en normativas, algunas incluso reconocidas a nivel europeo, como las promovidas por “Inclusion Europe”. En este contexto las tecnologías innovadoras de IA, como los modelos generativos de lenguaje, pueden ser de gran ayuda para mejorar la accesibilidad de los textos de forma automatizada. Estos modelos entrenados con grandes cantidades de datos son capaces de generar texto coherente a partir de un texto de entrada, lo que puede facilitar la creación de resúmenes y explicaciones adaptadas a las necesidades de los usuarios siguiendo las pautas de Lectura Fácil.

En este trabajo se propone diseñar un conjunto de pruebas para evaluar los sistemas de IA generativa existentes en la adaptación de texto en español, orientado a la producción de textos en lectura facilitada siguiendo las pautas de Lectura Fácil aportadas por la Asociación Española de Lectura Fácil (ALF). Esto con el objetivo de investigar la eficacia de los modelos Large Language Model (LLM) en la generación de textos accesibles. Además de esto exploraremos alguna estrategias además de implementar alguna pequeña solución propia de IA generativa.

Los sistemas resultantes se adaptarán para poder ser utilizados a través de una sencilla aplicación que mediante una capa de servicios permitirá a los usuarios introducir un texto y recibir una versión adaptada a lectura facilitada.

2 Marco Teórico

En este capítulo se presenta el marco teórico que sirve de base para el desarrollo de este trabajo. A continuación se describen los conceptos y tecnologías relevantes para la comprensión del estudio y desarrollo realizado.

En primer lugar se incluye una breve descripción de la lectura fácil y su importancia en la accesibilidad, luego se presentan de forma más detallada las normas que seguiremos para la generación de textos en lectura facilitada. Finalmente se explican los modelos de lenguaje y otros conceptos relacionados con la IA que son también de importancia.

2.1 Lectura Fácil

La Lectura Fácil (Asociación Lectura Fácil, s.f.) es un conjunto de directrices, normas y recomendaciones sobre la redacción, el diseño y la maquetación de textos y documentos. También se establecen pautas para validar su comprensión. Su propósito es hacer que la información sea accesible a personas con dificultades en la comprensión lectora. Estas pautas no se limitan a textos escritos, también se aplican en páginas web, guiones de obras audiovisuales, formularios y todo tipo de material informativo. Este movimiento de universalización de la información aparece por primera vez en los países nórdicos durante la década de 1980, cuando se empezaron a sistematizar métodos para simplificar el lenguaje. Desde entonces, la Lectura Fácil, o “*Easy Read*” en inglés, se ha expandido progresivamente por Europa.

Podemos encontrar numerosas normas y pautas a nivel mundial y europeo con este mismo objetivo. Concretamente en España tenemos vigente a nivel nacional la norma UNE 153101:2018 EX (AENOR, 2018), titulada “Lectura Fácil. Pautas y recomendaciones para la elaboración de documentos”, que es la que regula el proceso de adaptación de textos con el objetivo de asegurar su calidad.

A la hora de crear un documento o adaptar uno ya existente para lectura fácil se suelen llevar a cabo tres fases: redacción, diseño/maquetación y validación. En la primera fase se adapta el contenido del texto en sí siguiendo las normas de ortotipografía, vocabulario, composición de oraciones y estilo. En la fase de adaptación del diseño y la maquetación se prioriza la legibilidad usando fuentes de texto, tamaños y espacios adecuados, además de utilizar información visual mediante viñetas. Por último, en la fase de validación participa un grupo de personas con dificultades de comprensión, que actuarán como validadores revisando los textos y respondiendo a preguntas de validación que permiten comprobar como de efectivas son las adaptaciones.

De estas fases, solamente nos centraremos en la primera, ya que es la que se ocupa de

la redacción y adaptación del contenido del texto, y es la que se puede abordar mediante técnicas de Procesamiento del Lenguaje Natural (PLN) y Machine Learning (ML). A este tipo de adaptaciones de textos se les conoce como “lectura facilitada”, que es un término que se utiliza para referirse a textos que no cumplen todas las pautas de lectura fácil, pero que son más accesibles que el texto original.

2.1.1 Pautas de lectura facilitada

Ahora repasaremos un listado de las distintas pautas de lectura fácil que vamos a seguir en el desarrollo de este trabajo. Algunas de las que encontramos en las normas oficiales no son aplicables del todo a la generación automática de textos, otras están relacionadas con el diseño, maquetación y validación, y otras simplemente están orientadas a la creación de documentos y no a la adaptación de textos, por lo que solamente están recogidas las que se adaptan a los objetivos planteados. Estas pautas se dividen en tres categorías: ortotipografía, vocabulario y oraciones.

2.1.1.1 Pautas de ortotipografía

- Evitar el uso de mayúsculas en palabras y frases completas, excepto en siglas.
- Utilizar mayúscula inicial al inicio de párrafos, títulos, después de punto o en nombres propios.
- Utilizar punto y aparte para separar ideas diferentes en lugar de coma.
- Utilizar punto y aparte o conjunciones en lugar del punto y seguido o la coma para separar ideas relacionadas.
- Utilizar dos puntos (:) para introducir listas con más de tres elementos.
- No utilizar punto y coma (;).
- Evitar paréntesis, corchetes y signos ortográficos poco habituales (% , & , / , etc.).
- No utilizar etcétera ni puntos suspensivos (...). Reemplazarlos por “entre otros” o “y muchos más”.
- Evitar las comillas si es posible. Si se utilizan, por ejemplo en una cita, deben ir acompañadas de una explicación.

2.1.1.2 Pautas de vocabulario

- Usar lenguaje sencillo y de uso frecuente.
 - Adaptar el vocabulario al público objetivo.
 - Evitar utilizar términos abstractos, técnicos o complejos.
 - Evitar el uso de palabras homófonas u homógrafas, sustituyéndolas por sinónimos.
 - Evitar utilizar palabras largas o con sílabas complejas.
-

- Evitar utilizar adverbios terminados en -mente.
- Evitar utilizar superlativos, es preferible utilizar “muy” + adjetivo.
- Evitar palabras redundantes o que no aportan información y alargan el texto.
- Evitar palabras en otros idiomas salvo las de uso común (ej. wifi).
- Evitar utilizar abreviaturas.
- Evitar utilizar siglas. Cuando se trata de acrónimos de uso común, se deberá explicar el significado del acrónimo en su primer uso.
- Evitar utilizar frases nominales o formas nominales de adjetivos.
- Evitar utilizar lenguaje figurado, o en caso de utilizarlo incluir su explicación.
- Evitar el uso nominal de los verbos cuando resulta en una frase metafórica o abstracta.
- Dentro de lo posible se usará la misma palabra para el mismo referente u objeto.
- Evitar utilizar palabras de indeterminadas (ej. cosa, algo).
- Escribir los números con cifras. En caso de ser números muy grandes será mejor utilizar comparaciones cualitativas si es posible.
- Separar los dígitos de números de teléfono por bloques.
- Evitar utilizar números ordinales, mejor expresarlo de forma cardinal.
- Evitar fracciones y porcentajes, se usarán descripciones escritas equivalentes.
- Escribir las fechas con el día, mes y año escritos de forma completa. (ej. “el 1 de enero de 2023” en lugar de “01/01/2023”).
- Evitar utilizar el formato de 24 horas, es preferible usar el de 12 horas especificando con texto la etapa del día (ej. “las 3 de la tarde” en lugar de “15:00”).
- Evitar números romanos, es preferible escribirlos como se leen.

2.1.1.3 Pautas de oraciones

- Utilizar frases sencillas, evitar oraciones complejas.
 - Utilizar el presente de indicativo siempre que sea posible.
 - Evitar utilizar tiempos compuestos, condicionales y subjuntivos.
 - Evitar utilizar la voz pasiva, siempre que sea posible se utilizará la activa.
 - Evitar utilizar la forma pasiva refleja.
 - Utilizar el imperativo solo en contextos claros, aclarando a quién se dirige.
-

- Evitar utilizar oraciones impersonales siempre que sea posible.
- Evitar utilizar oraciones con gerundio.
- Evitar utilizar verbos consecutivos, salvo las perífrasis con deber, querer, saber o poder.
- Utilizar preferiblemente oraciones afirmativas, excepto en prohibiciones sencillas.
- Evitar utilizar la doble negación, formular las oraciones de forma clara.
- Evitar utilizar elipsis, expresar claramente todas las ideas.
- Evitar utilizar explicaciones entre comas que interrumpan la lectura.
- Evitar utilizar aposiciones que corten el ritmo del texto.
- Limitar las oraciones a dos ideas por frase como máximo.
- Utilizar conectores simples, evitar conectores complejos como “por lo tanto” o “sin embargo”.

2.2 Machine Learning

El Machine Learning¹, o Aprendizaje Automático en español, es una rama de la IA que se centra en el desarrollo de algoritmos capaces de “aprender” a partir de datos y mejorar su desempeño en una tarea sin ser programados concretamente para cada propósito específico. Estas tareas suelen consistir en identificar patrones en los datos y utilizarlos para realizar predicciones o tomar decisiones. El aprendizaje automático se basa en métodos estadísticos y de optimización matemática para construir los modelos que realizan estas tareas.

Existen distintas categorías de aprendizaje automático, entre las que destacan:

- **Aprendizaje supervisado:** El algoritmo se entrena con un conjunto de datos etiquetados, aprendiendo a predecir la salida que corresponde a nuevas entradas.
- **Aprendizaje no supervisado:** El algoritmo analiza datos sin etiquetas, encontrando estructuras o patrones ocultos en los datos.
- **Aprendizaje por refuerzo:** Un agente aprende a tomar decisiones mediante la interacción con un entorno, recibiendo recompensas o penalizaciones en función de sus acciones.

El aprendizaje automático tiene aplicaciones en muchos campos distintos, incluyendo el PLN, que es el área que vamos a tratar a continuación.

¹https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

2.3 Procesamiento del Lenguaje Natural

El Procesamiento del Lenguaje Natural² es el área de la IA que estudia cómo las máquinas pueden entender, interpretar y generar lenguaje. Para ello combina técnicas de lingüística computacional, aprendizaje automático y estadística.

En sus inicios los sistemas de PLN se basaban en conjuntos de reglas diseñadas manualmente. Sin embargo, a partir de finales de la década en 1980 hubo un cambio notorio con la introducción de los algoritmos de ML, que resultaron ser por norma general más adaptables y eficientes en el procesamiento del lenguaje.

Los mayores problemas que encuentra el PLN principalmente tienen que ver con la complejidad del lenguaje humano, como la ambigüedad léxica y estructural, la variabilidad en la pronunciación y escritura de las palabras, y la necesidad de comprender el contexto para interpretar el significado correcto del texto. A lo largo de su historia, el PLN ha avanzado considerablemente y lo podemos encontrar en muchas áreas de aplicación distintas, como en la traducción automática, los asistentes virtuales o en sistemas de respuesta a preguntas, por mencionar algunos ejemplos.

2.4 Modelos de Lenguaje

Los modelos de lenguaje son sistemas de IA que aprenden patrones estadísticos del lenguaje a partir de grandes cantidades de texto, de forma que pueden predecir y generar secuencias de palabras en lenguaje natural. En el contexto del PLN son una herramienta fundamental para tareas como la traducción automática, generación de texto y la simplificación.

Los primeros modelos tenían enfoques matemáticos y estadísticos más simples como los n-gramas, que son grupos de palabras o tokens, que están algo limitados en la captura del contexto a largo plazo. Con el avance de las técnicas de ML surgieron modelos más sofisticados como las redes neuronales recurrentes (RNN), que mejoraron el procesamiento de secuencias.

El punto clave de los avances en este área fue la aparición de la arquitectura *Transformer*, que revolucionó el PLN y dio lugar a modelos como GPT y Bidirectional Encoder Representations from Transformers (BERT). Estos modelos muestran capacidades muy superiores en tareas lingüísticas respecto a lo que existía anteriormente.

2.4.1 Transformers

Los transformers, introducidos por Vaswani y cols. (2023) en el famoso paper *Attention is All You Need*, fueron un cambio en el paradigma de los modelos de lenguaje. Anteriormente las arquitecturas procesaban el texto secuencialmente, la novedad de los transformers fue que estos procesan todas las posiciones de una secuencia simultáneamente, siendo esto mucho más eficiente y además captura mejor las referencias y contextos a largo plazo.

²https://es.wikipedia.org/wiki/Procesamiento_de_lenguajes_naturales

El componente principal es el mecanismo de atención multi-cabeza, que permite al modelo enfocarse en diferentes partes de la secuencia al mismo tiempo, capturando relaciones complejas entre palabras muy separadas. La atención calcula un peso para cada palabra en relación con todas las demás, así determina qué información es más importante en el contexto.

La arquitectura consiste en un codificador que procesa la entrada y genera representaciones que tienen en cuenta el contexto, y un decodificador que usa estas representaciones internas para generar la salida en lenguaje natural.

En cuanto a la simplificación de texto, los transformers tienen ciertas ventajas con respecto a soluciones anteriores basadas en sistemas de redes neuronales (como RNN y CNN) al comprender el contexto completo. Por ello en este proyecto se realizará un estudio de varios modelos de lenguaje, específicamente LLM, basados en esta arquitectura.

2.4.2 Ingeniería de Prompts y Ajuste Fino

Tanto la ingeniería de prompts como el ajuste fino son estrategias o técnicas para adaptar modelos de lenguaje ya entrenados a tareas específicas, como podría ser la generación de textos en lectura facilitada.

La ingeniería de prompts consiste en diseñar cuidadosamente las instrucciones que se proporcionan al modelo para guiar su comportamiento. En esta área se trabaja con prompts ajustados y contextualizados, con instrucciones claras que guían al modelo a la respuesta deseada, evaluación de los resultados y el uso de distintas técnicas como la inserción de ejemplos o de roles.

Por otro lado, el ajuste fino (o Fine Tuning) consiste en continuar el entrenamiento de un modelo preentrenado con datos específicos de la tarea que pretendemos mejorar. Hay algunas variantes como el “ajuste fino completo”, que actualiza todos los parámetros, y técnicas eficientes como LoRA, que adaptan el modelo con menor coste computacional. Más adelante profundizaremos más en este último método de ajuste fino.

2.5 Métricas de Evaluación

La evaluación de la calidad de los textos generados en lectura facilitada es clave para validar cómo de efectivas son las soluciones desarrolladas. Para ello empleamos distintas métricas y técnicas de evaluación. Principalmente utilizamos métricas de similitud léxica junto a un corpus de textos facilitados manualmente, vamos a repasar las métricas utilizadas.

2.5.1 BLEU

Bilingual Evaluation Understudy (BLEU) es una métrica que se utiliza en la evaluación de sistemas de traducción automática, propuesta por Papineni y cols. (2002). Fundamentalmente se basa en la idea de que cuanto más cerca esté una traducción automática de una traducción humana escrita por un profesional, mejor será. En nuestro caso, la traducción automática sería el texto facilitado generado por el modelo y la traducción humana serían los textos

adaptados manualmente.

A nivel técnico, para medir esta proximidad lo que hace BLEU es comparar n-gramas (secuencias de un número n de palabras) entre la traducción candidata y la de referencia de alta calidad, y cuenta las coincidencias que encuentre entre ambas sin importar su posición en los textos. La métrica aplica un factor de penalización para evitar que se generen textos muy cortos, ya que pueden obtener una puntuación bastante alta sin ser realmente de alta calidad. Estos factores se combinan mediante una fórmula matemática que utiliza un promedio geométrico de las coincidencias de n-gramas, que da como resultado un valor decimal entre 0 y 1, donde 1 es una coincidencia total con la referencia.

2.5.2 ROUGE

Recall-Oriented Understudy for Gisting Evaluation (ROUGE), desarrollado por Lin (2004) es un conjunto de métricas diseñadas para la evaluación de resúmenes en comparación con resúmenes de referencia escritos a mano. De forma similar a BLEU, ROUGE compara n-gramas entre el resumen generado y el de referencia, pero también tiene en cuenta coincidencias de unidades, pares y secuencias completas de palabras. Para esto hay varias variantes de ROUGE, pero las que utilizaremos en nuestras evaluaciones son ROUGE-1 (coincidencia de unigramas, es decir, una sola palabra o token), ROUGE-2 (coincidencia de bigramas) y ROUGE-L, que es el número de coincidencias de la subsecuencia más larga común teniendo en cuenta el orden y permitiendo huecos entre las palabras. Los valores de ROUGE también se expresan como un valor entre 0 y 1, donde 1 indica una coincidencia perfecta con el resumen de referencia, ya que se calcula como una proporción entre el número de coincidencias y el número total de n-gramas tanto en el texto de referencia como en el texto generado.

2.5.3 METEOR

Metric for Evaluation of Translation with Explicit ORdering (METEOR), por Banerjee y Lavie (2005), es otra métrica de evaluación de traducción que surge como una alternativa a BLEU debido a ciertas limitaciones de esta. En primer lugar se identificó que en BLEU el sistema de penalización por brevedad no era del todo efectivo, ya que no se compensa de forma adecuada la calidad de la traducción con su longitud. Además, BLEU no tiene en cuenta la semántica de las palabras, por lo que dos palabras con el mismo significado pero que no coinciden léxicamente no se consideran como una coincidencia. En nuestro caso, una facilitación puede no ser exactamente igual que otra, pero pueden tener el mismo significado y ser igualmente válidas.

Lo que plantea METEOR es comparar las traducciones emparejando palabras de forma más inteligente. En primer lugar busca palabras idénticas, después palabras que comparten la misma raíz (como podrían ser “apóstol” y “apóstoles”) y también palabras sinónimas. Esto permite que una traducción pueda ser correcta aunque use palabras diferentes a las de la referencia. El modo en el que se lleva a cabo esto es mediante WordNet³, una base de datos léxica que almacena palabras agrupadas en conjuntos de sinónimos, así que con la referencia

³<https://wordnet.princeton.edu/>

de la palabra se buscan sus sinónimos y se comprueba si coincide con la traducción automática.

Una vez que identifica estas correspondencias se calcula la puntuación utilizando la precisión (porcentaje de palabras de la traducción automática que coinciden con la referencia), el *recall* (porcentaje del texto de referencia que coincide con el automático) y una penalización por fragmentación, que mide si las palabras que coinciden aparecen en el mismo orden. La fórmula final da más peso al *recall* que a la precisión, ya que se considera más importante que la traducción cubra todo el contenido.

2.5.4 BERTScore

BERTScore (Zhang y cols., 2020) es la más moderna de las métricas de evaluación que vamos a utilizar. Surge para superar algunas de las limitaciones de las métricas “tradicionales” como BLEU o METEOR. El problema en estas métricas es que buscan coincidencias superficiales entre palabras o n-gramas, de modo que no reconocen paráfrasis, reordenamientos ni equivalencias semánticas de forma correcta. En vez de buscar coincidencias exactas, lo que plantea BERTScore es evaluar la similitud en la semántica entre el texto generado y el texto de referencia. Para esto utiliza modelos de lenguaje preentrenados como lo es BERT (Devlin y cols., 2018), que generan representaciones vectoriales (*embeddings*) para cada palabra teniendo en cuenta el contexto en el que aparece. Estas representaciones capturan el significado de cada palabra en función de su entorno (su contexto).

El funcionamiento de BERTScore se basa en calcular la similitud coseno entre los *embeddings* de las palabras del texto generado y los de la referencia, es decir, se mide la diferencia entre los ángulo que forman los vectores que representan las palabras en el espacio vectorial. Luego se empareja cada palabra con la más similar del otro texto (de forma *greedy* o “voraz”, es decir, seleccionando la más cercana) y a partir de estas correspondencias se calculan tres métricas: la precisión (porcentaje del texto generado que coincide con la referencia), el *recall* (porcentaje de la referencia que coincide con el texto generado) y una medida llamada “F1” que combina ambas. También puede aplicarse una ponderación por IDF (frecuencia inversa de documento) para dar más peso a las palabras menos frecuentes, que suelen aportar más información.

Una de las mayores ventajas de BERTScore es que es independiente del idioma y de la tarea, ya que se utiliza en el conocimiento del modelo BERT. Esto le permite encontrar correlaciones más altas con evaluaciones manuales que otras métricas. En experimentos sobre traducción automática y subtitulado de imágenes BERTScore supera con un gran margen a métricas como BLEU o METEOR. Según distintas pruebas también ha resultado ser más robusta frente a ejemplos difíciles en los que otras métricas suelen fallar. Se suele utilizar la medida F1 como puntuación principal, aunque en nuestro caso estudiaremos las 3 métricas.

2.5.5 Flesch Reading Ease

La métrica de Flesch Reading Ease⁴ es una fórmula de legibilidad desarrollada por Rudolf Flesch en 1948 que evalúa la facilidad de lectura de un texto en función de la longitud promedio de las oraciones y el número de sílabas por palabra. Esta métrica puede ser útil para la evaluación de textos en lectura facilitada, ya que proporciona una medida objetiva de la complejidad lingüística del texto.

La fórmula de Flesch Reading Ease se calcula mediante la siguiente ecuación:

$$\text{Flesch Reading Ease} = 206.835 - 1.015 \times \left(\frac{\text{total de palabras}}{\text{total de oraciones}} \right) - 84.6 \times \left(\frac{\text{total de sílabas}}{\text{total de palabras}} \right)$$

El resultado es un valor entre 0 y 100, donde los valores más altos significan una mayor facilidad de lectura. Una de las principales ventajas de esta métrica es su simplicidad a la hora de ser calculada, ya que solo requiere contar palabras, oraciones y sílabas. Sin embargo, es bastante limitada en el contexto de la facilitación de textos, ya que no considera aspectos semánticos, de vocabulario, de estructura sintáctica o de coherencia. Además, la fórmula original fue diseñada para el inglés, por lo que su aplicación directa al español puede no ser del todo funcional y podría requerir ajustes.

En el contexto de este trabajo, la métrica de Flesch Reading Ease se utiliza como una medida complementaria durante el proceso de entrenamiento de uno de los modelos de lenguaje, para comparar la complejidad de los textos generados con respecto a los textos de referencia. Es más utilizado como una guía para ver el cambio de funcionamiento del modelo durante el entrenamiento que como una métrica de evaluación final.

2.6 GAN

Las Generative Adversarial Networks (GAN)⁵, o Redes Generativas Adversativas en español, son una arquitectura de ML que se basa en un enfoque de aprendizaje adversario donde dos redes neuronales “compiten” entre sí. Propuesto por Goodfellow y cols. (2014), consiste en entrenar al mismo tiempo dos modelos: un “generador” que aprende a crear datos que imiten los reales y un “discriminador” que aprende a distinguir entre datos reales y generados.

Durante el entrenamiento el generador intenta engañar al discriminador produciendo datos cada vez más realistas, y el discriminador aprende a detectar mejor las diferencias. Este proceso adversario continúa hasta llegar a un punto en el que el generador produce muestras de alta calidad que el discriminador prácticamente no puede distinguir de los datos reales.

Las GAN presentan desafíos cuando se aplican al PLN por las complicaciones que encuentra en el entrenamiento para adaptar la generación de texto. También la evaluación de la calidad del texto generado es más compleja que en otro tipo de datos, ya que se deben considerar

⁴https://simple.wikipedia.org/wiki/Flesch_Reading_Ease

⁵https://es.wikipedia.org/wiki/Red_generativa_adversativa

variables como la coherencia y el significado. Más adelante, en el capítulo de resultados, veremos cómo estas complicaciones han afectado al desarrollo de este tipo de modelos.

2.7 LoRA

Low-Rank Adaptation (LoRA) (Hu y cols., 2021) una técnica de adaptación de modelos que permite realizar *fine-tuning* de forma eficiente sobre redes neuronales de gran tamaño, reduciendo de forma significativa los recursos computacionales necesarios. La idea principal de LoRA es descomponer las matrices de parámetros del modelo original y “congelarlas”, añadiendo nuevas matrices de baja dimensión entre ellas que representan la “adaptación” específica a la tarea objetivo. De esta manera, solamente es necesario optimizar un conjunto pequeño de parámetros adicionales, ya que estaremos solamente modificando estas matrices intermedias con menos cantidad de parámetros, mientras que los pesos originales del modelo se mantienen intactos.

En LoRA, dada una capa lineal con matriz de pesos original W_0 , se introduce una corrección de “bajo rango” W expresada como

$$W = W_0 + \Delta W, \quad \Delta W = AB$$

Los parámetros de A y B son los únicos optimizados durante el proceso de *fine-tuning*, y tienen una cantidad mucho menor de parámetros que W_0 , ya que solamente necesitan coincidir en una de sus dimensiones y se puede permitir tener un tamaño en su otra dimensión mucho menor que en las matrices originales. Esta descomposición disminuye bastante el consumo de memoria y la demanda de cómputo. A este tipo de técnica se le conoce como Parameter-Efficient Fine-Tuning (PEFT).

En este proyecto se ha aplicado LoRA para intentar adaptar un modelo preentrenado a la tarea de facilitación. Para esto se ha implementado un módulo **QLoRA** personalizado para ser usado junto al modelo base. QLoRA, introducido por Dettmers y cols. (2023), es una variante de LoRA que utiliza cuantización para reducir aún más el tamaño del modelo y los recursos necesarios para el ajuste fino. La cuantización es un proceso por el cual se reduce la precisión de los pesos del modelo (por ejemplo de 32 bits a 8 bits), lo que permite que el modelo ocupe menos espacio y sea más eficiente a la hora de trabajar con él. Mediante estos dos métodos combinados se consigue un procesamiento mucho más ligero.

2.8 API REST

Una API REST (*Representational State Transfer*) es una interfaz que permite la comunicación entre sistemas a través del protocolo HTTP, utilizando una arquitectura basada en recursos. Cada recurso se accede mediante una URL, especificando las distintas posibles rutas, y puede ser manipulado utilizando los métodos HTTP: **GET** (solicita datos sobre un recurso al servicio), **POST** (envía datos para crear un recurso), **PUT** (envía datos para modificar un recurso) y **DELETE** (solicita la eliminación de un recurso).

Las APIs RESTful siguen principios como la ausencia de estado (*stateless*) y la separación entre cliente y servidor. Estas características hacen que sea mucho más sencilla la escalabilidad de los sistemas, la reutilización de código y la integración en distintos entornos.

En este proyecto se ha utilizado una API REST para funcionar como capa de servicio de un prototipo implementado.

3 Objetivos

Como ya se ha mencionado en la introducción, el objetivo principal de este trabajo es evaluar una serie de modelos de lenguaje y diseñar un sistema de IA generativa para la producción de textos en lectura facilitada. Para ello, los objetivos específicos que se pretenden conseguir son los siguientes:

- **Desarrollar un sistema para utilizar los modelos de lenguaje:** Desarrollar código que permita procesar un texto de entrada y generar una versión facilitada a partir de un prompt enviado a modelos. Se utilizará un corpus de textos para ello, y lo procesarán cada uno de los modelos de lenguaje seleccionados para su estudio.
- **Diseñar soluciones para la generación de texto en lectura facilitada:** Proponer diferentes enfoques para la generación de texto en lectura facilitada. Concretamente implementaremos pequeños sistemas de redes adversarias (GAN) y un modelo ajustado para que procesen los textos de entrada y generen una versión facilitada.
- **Evaluar las soluciones y LLMs existentes en la generación de lectura facilitada:** Implementar un conjunto de tests con las métricas de evaluación que se han seleccionado, ya explicadas en el capítulo del marco teórico, para evaluar la calidad de los textos generados por los modelos de lenguaje. Una vez obtenidos los resultados, se estudiarán y compararán los resultados de cada uno de los sistemas.
- **Diseñar un servicio y prototipo que permita el uso de la solución desarrollada:** Integrar los sistemas de IA generativa en una aplicación a modo de *demo* que permita a los usuarios introducir un texto y recibir una versión facilitada a través de él.

4 Metodología

En este capítulo se describen algunas de las herramientas y tecnologías empleadas durante el desarrollo de este trabajo que han resultado más relevantes. Repasaremos las tecnologías, lenguaje, plataformas y librerías que han sido fundamentales para la implementación y evaluación de los sistemas utilizados.

4.1 Plataformas

4.1.1 Git y GitHub

Git¹ es un sistema de control de versiones distribuido de código abierto, creado por Linus Torvalds en 2005. Esta herramienta es utilizada para hacer un seguimiento de los cambios realizados en el código fuente y archivos del proyecto a lo largo del tiempo, facilitando la colaboración entre varios desarrolladores y registrando el historial completo de cambios.

En el desarrollo de este trabajo Git ha sido utilizado para mantener el control sobre las diferentes versiones del código, pudiendo experimentar y realizar grandes cambios sin riesgo de perder trabajo anterior. Además me ha permitido poder trabajar entre mis diferentes dispositivos de forma fluida, sincronizando los cambios fácilmente.

En concreto, he utilizado la página web de GitHub² para mantener el repositorio del proyecto, que es una de las plataformas más populares para alojar proyectos de código abierto y colaborar *online*.

4.1.2 Hugging Face

Hugging Face³ es una plataforma especializada en IA y ML que pone a disposición de cualquiera distintas herramientas, modelos preentrenados y corpus o conjuntos de datos. La plataforma, especialmente recientemente, se ha convertido en uno de los elementos centrales para el desarrollo de aplicaciones de PLN.

Durante el desarrollo del proyecto he utilizado Hugging Face tanto para acceder a LLMs como para utilizar su librería “transformers”, que es muy cómoda para configurar y utilizar modelos preentrenados públicos en la página. También algo que ha resultado clave ha sido el uso de su API de proveedores de inferencia a modelos, que permite realizar inferencias de forma sencilla y rápida, lo cual ha resultado muy útil debido a la escasez de recursos en mi equipo personal. Por último, he utilizado su plataforma para almacenar y descargar mis

¹<https://git-scm.com/>

²<https://github.com/>

³<https://huggingface.co/>

propios modelos generados para el proyecto, de forma similar a como uno podría hacer con un repositorio de GitHub.

4.2 Entornos de Desarrollo

4.2.1 Google Colab

Google Colaboratory⁴, o Google Colab, es un servicio gratuito basado en la nube que permite ejecutar código Python en navegadores web sin necesidad de realizar ejecución local. Permite acceso a recursos computacionales como GPUs y TPUs de bastante potencia por un tiempo limitado, esto lo hace una herramienta clave para proyectos de ML e IA como podría ser este.

La ventaja de estos sistemas es que permiten ejecutar los procesos utilizando Compute Unified Device Architecture (CUDA) (NVIDIA y cols., 2020), que es una tecnología de NVIDIA que permite la ejecución de código de forma paralela en sus GPUs, repartiendo la carga de trabajo entre los diferentes núcleos de la unidad gráfica y haciendo que el entrenamiento de modelos sea mucho más rápido que mediante la ejecución en CPU.

Google Colab ha sido fundamental para realizar experimentos que requerían mucha potencia de cálculo, como en el entrenamiento de mis propios modelos y la evaluación de los LLMs. Su integración con Google Drive ha facilitado el almacenamiento y la gestión de los datos y archivos generados. Como ya he dicho antes, mi equipo personal no tiene la potencia necesaria para realizar estos experimentos, por lo que ha sido de las herramientas a las que más he recurrido para el desarrollo. Finalmente, en cuanto a las especificaciones hardware de los entornos de Google Colab que se han utilizado durante el desarrollo del proyecto, en las tablas 4.1 y 4.2 tenemos los datos más relevantes, siendo la primera la especificación del entorno gratuito y la segunda la del entorno Pro.

Característica	Detalle
GPU	NVIDIA Tesla T4
Memoria VRAM (GPU)	16 GB (15 GB utilizable)
CPU	Intel Xeon @ 2.20GHz (2 vCPU)
Memoria RAM	13 GB

Tabla 4.1: Especificaciones del entorno Colab con GPU Tesla T4.

4.2.2 Visual Studio Code

Visual Studio Code⁵ (o VSCode) es un editor de código fuente desarrollado por Microsoft que se ha establecido como una de las herramientas más populares para el desarrollo de software. Es hoy en día el Entorno de Desarrollo Integrado (IDE) más utilizado en el mundo. Tiene varias características que lo hacen ser una herramienta muy valiosa, como el coloreado

⁴<https://colab.research.google.com/>

⁵<https://code.visualstudio.com/>

Característica	Detalle
GPU	NVIDIA Tesla A100
Memoria VRAM (GPU)	40GB
CPU	Intel Xeon @ 2.20GHz (2 vCPU)
Memoria RAM	43GB

Tabla 4.2: Especificaciones del entorno Colab Pro con GPU A100.

de sintaxis, un sistema de depuración, integración con el control de versiones de Git y el enorme repertorio de extensiones que aportan muchas funcionalidades para diferentes lenguajes y frameworks.

En mi opinión, esto lo vuelve mucho más versátil que otros editores de texto como Vim u otros IDEs como Eclipse. En este proyecto, Visual Studio Code ha servido como el entorno de desarrollo principal para la programación y organización del código de Python escrito, pero además ha servido para editar los archivos de texto y JavaScript Object Notation (JSON) generados durante el desarrollo, además de poder ejecutar las celdas de Jupyter Notebook directamente desde el editor, haciendo el proceso bastante cómodo.

4.3 Lenguajes y Librerías

4.3.1 HTML, CSS y JavaScript

Se agrupan estos tres lenguajes debido a que han sido utilizados en conjunto, concretamente para el prototipo usable que se ha desarrollado. Estos lenguajes suelen ir de la mano ya que son el núcleo principal del desarrollo web del lado del cliente. Vamos a repasar rápidamente cada uno.

- **HyperText Markup Language (HTML)** se utiliza para definir la estructura del contenido. En el prototipo se ha usado para organizar los elementos de la interfaz: formularios, textos, botones y contenedores.
- **Cascading Style Sheets (CSS)** se ha empleado para aplicar estilos visuales a los elementos HTML, mejorando la estética. Se han definido colores, tipografías, bordes, sombras, animaciones y diseño adaptable a diferentes dispositivos (*responsive design*).
- **JavaScript**⁶ se ha usado para programar la parte interactiva de la herramienta. Su función ha sido gestionar eventos (como el envío del formulario), realizar peticiones a la API REST y mostrar los resultados devueltos sin necesidad de recargar la página.

4.3.2 Python

Python⁷ es un lenguaje de programación de alto nivel, interpretado y de propósito general. Es conocido por su sintaxis clara y legible, su sistema de tipado dinámico y su gran comunidad

⁶<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

⁷<https://www.python.org/>

que desarrolla y mantiene una enorme cantidad de librerías. La versión 3.10.13 utilizada en este trabajo no es de las más recientes que podemos encontrar (existiendo ya en versión estable hasta la 3.13.5), pero es una versión que encontré adecuada para el proyecto por compatibilidad con las librerías utilizadas.

Python ha sido el lenguaje principal de este trabajo, ya que podemos hacer uso de muchísimas librerías y herramientas especializadas en ciencia de datos, ML y PLN. En este tipo de áreas de desarrollo es el lenguaje más utilizado y recomendado. Además, su simplicidad ha facilitado la experimentación rápida. Por último, otra de las motivaciones principales para escoger este lenguaje ha sido su integración con Google Colab, que como ya he mencionado, ha sido una herramienta clave para el desarrollo del proyecto.

4.3.3 Jupyter Notebook

Jupyter Notebook⁸ es una aplicación y formato de código abierto que permite crear documentos interactivos que contienen código ejecutable, visualizaciones y texto. Su nombre viene de los tres lenguajes de programación que soporta: Julia, Python y R. Aunque hoy en día es compatible con más de 100 lenguajes de programación.

Los notebooks de Jupyter organizan el contenido en celdas que contienen código, texto en formato Markdown, o salidas de ejecución como texto, gráficos y tablas. Esta estructura lo vuelve una herramienta que permite ordenar el trabajo de forma muy clara y facilita la documentación y ejecución del código.

En este proyecto se ha trabajado bastante con Jupyter Notebook para organizar el código y las pruebas realizadas. Además, como ya he mencionado, he utilizado Google Colab, que utiliza el formato de Jupyter Notebook para ejecutar el código en la nube.

4.3.4 JSON

JSON⁹ es un formato ligero de almacenamiento de datos que es fácil de leer y escribir para humanos, y simple de *parsear* (convertir datos de un formato a otro) y generar para máquinas. Está basado en el lenguaje JavaScript, pero JSON es independiente del lenguaje de programación y se ha convertido en un estándar para el intercambio de datos en aplicaciones web y APIs.

JSON ha sido utilizado como formato principal para el almacenamiento e intercambio de datos en el proyecto, tanto en conjuntos de datos como en resultados de las evaluaciones. Su estructura hace que organizar de manera eficiente los datos sea muy sencillo. La compatibilidad nativa de JSON en Python (mediante la librería “json”) ha facilitado bastante el manejo del corpus y los textos generados durante el procesamiento.

⁸<https://jupyter.org/>

⁹<https://en.wikipedia.org/wiki/JSON>

4.3.5 NumPy

NumPy¹⁰ es la librería fundamental para computación científica en Python. Proporciona soporte para arrays multidimensionales eficientes y una gran cantidad de funciones matemáticas implementadas de forma óptima para operar sobre estos arrays. NumPy es la base sobre la que se construyen muchas otras librerías científicas de Python.

La librería ha sido de gran ayuda para el manejo y operaciones sobre datos numéricos, específicamente en las operaciones con vectores y matrices para los cálculos relacionados con las métricas de evaluación. Gracias a sus funciones estadísticas el análisis de los resultados ha sido más sencillo de implementar.

4.3.6 Natural Language Toolkit

Natural Language Toolkit (NLTK)¹¹ es una librería muy completa de PLN en Python. Ofrece interfaces fáciles de usar para más de 50 corpus y recursos, además de un conjunto de librerías de procesamiento de texto para clasificar, tokenizar, etiquetar y realizar análisis sintáctico.

En este proyecto NLTK ha sido utilizado para el preprocesamiento de textos a la hora de realizar el análisis y cálculo de las métricas de evaluación de los modelos. En concreto se ha hecho uso de sus funciones de tokenización, además de utilizar recursos como el corpus de WordNet y PUNKT para encontrar los límites de las oraciones para el contexto de las palabras. En general, las métricas BLEU y METEOR han sido calculadas utilizando funciones de la librería NLTK.

4.3.7 PyTorch

PyTorch¹² es un framework de ML de código abierto desarrollado por Meta, que proporciona una plataforma para el uso y desarrollo de modelos. Permite modificar fácilmente la arquitectura de la red neuronal que se está utilizando o desarrollando y está muy integrado con Python, lo que facilita la depuración y experimentación.

PyTorch ha sido fundamental para la implementación y manipulación de modelos. He utilizado la librería “transformers” de Hugging Face, que está construida sobre PyTorch, para cargar y utilizar los modelos de lenguaje, pero también he implementado mis propios modelos utilizando su propia librería. Es una herramienta muy potente y muy utilizada a los más altos niveles de investigación en áreas de IA y ML.

¹⁰<https://numpy.org/>

¹¹<https://www.nltk.org/>

¹²<https://pytorch.org/>

4.3.8 FastAPI y Uvicorn

Estas dos librerías, FastAPI¹³ y Uvicorn¹⁴, han sido las utilizadas para el desarrollo de la capa de servicio del prototipo, implementada mediante una API REST. Para esto ha sido utilizado el framework y librería de Python `fastapi`, con el que se pueden definir fácilmente rutas y métodos HTTP (como **GET** o **POST**), validando de forma automática los datos de entrada y ejecutando código local en Python para cada distinta ruta accedida.

Por otra parte, `uvicorn` es un servidor ASGI (Asynchronous Server Gateway Interface) que se encarga de ejecutar la aplicación desarrollada con FastAPI. Soporta llamadas asíncronas y permite manejar numerosas peticiones al mismo tiempo. Al ejecutar la API con Uvicorn, esta se mantiene a la espera en el dominio especificado para recibir las peticiones que procesará FastAPI.

4.4 Corpus ClearSim

El corpus ClearSim¹⁵ (Botella-Gil y cols., 2024) es un conjunto de datos textuales creado por el GPLSI. Este corpus está compuesto por textos en español, específicamente de organismos públicos, con textos originales y sus versiones simplificadas siguiendo las pautas de Lectura Fácil.

ClearSim me ha sido de gran ayuda para el desarrollo del proyecto. No he llegado a hacer uso del corpus en su totalidad, ya que solamente disponía de una parte de sus textos, pero ha sido fundamental para el entrenamiento de soluciones propias y para la evaluación de los modelos de lenguaje y las distintas pruebas realizadas. Disponía de un total de 500 textos, originales y facilitados, en formato JSON para utilizarlos de forma sencilla. La estructura del archivo JSON es la que podemos ver en la tabla 4.3.

Campo	Descripción
URL	Dirección web del texto original
TXT	Texto original sin procesar
FAC	Versión del texto adaptada manualmente a lectura facilitada

Tabla 4.3: Estructura del archivo JSON utilizado.

4.5 LLMs

En este apartado veremos los distintos modelos abiertos que vamos a evaluar con las métricas que hemos explicado anteriormente. En cuanto al motivo de la elección de estos modelos, los he seleccionado teniendo en cuenta varios factores. En primer lugar, he tenido

¹³<https://fastapi.tiangolo.com/>

¹⁴<https://www.uvicorn.org/>

¹⁵<https://cleartext.gplsi.es/resultados/>

en cuenta que fueran modelos abiertos. En segundo lugar, le he dado prioridad a aquellos modelos que son más conocidos y utilizados por la comunidad, intentando no repetir modelos desarrollados por la misma empresa. Y por último, que tuvieran soporte para el idioma español. Además de esos factores principales, he intentado seleccionar modelos relativamente ligeros, ya que no dispongo de los recursos para manejar modelos de gran tamaño.

4.5.1 LLaMA 3.1 8B Instruct

El modelo LLaMA 3.1 (AI@Meta, 2024) forma parte de la familia de modelos desarrollados por Meta. Es un modelo abierto y disponible públicamente, de los más recientes y populares en la comunidad de ML. Este modelo en concreto es una versión con 8 mil millones de parámetros, lo que lo hace relativamente ligero en comparación a muchos otros. Además, está diseñado para tareas de instrucción, lo que significa que ha sido entrenado para responder preguntas y seguir instrucciones de forma más efectiva.

Podemos encontrar este modelo en el repositorio de Hugging Face “meta-llama/Llama-3.1-8B-Instruct”.

4.5.2 Mistral 8B Instruct

Mistral 8B¹⁶ es un modelo desarrollado por Mistral AI, también de código abierto mediante licencia “Mistral Research License”, donde se permite su uso para investigación y desarrollo no comercial. Al igual que el de LLaMA, es un modelo de 8 mil millones de parámetros. También diseñado para tareas de instrucción. Mistral ha tenido bastante impacto recientemente en la comunidad, por lo que me pareció interesante evaluar uno de sus modelos.

Para hacer uso de este modelo tenemos su repositorio “mistralai/Ministral-8B-Instruct-2410”.

4.5.3 Phi 3 Small 128K Instruct

Phi es una serie de modelos desarrollados por Microsoft, y Phi 3 Small 128K¹⁷ es una de sus versiones más recientes. Tiene 7 mil millones de parámetros y está también diseñado para tareas de instrucción. El número 128K se refiere a su capacidad de contexto, lo que significa que puede manejar entradas de texto de hasta 128 mil tokens, que es una capacidad bastante grande en comparación con otros modelos.

Se encuentra disponible en el repositorio “microsoft/Phi-3-small-128k-instruct”.

4.5.4 Qwen 2.5 7B Instruct

Los modelos Qwen (Team, 2024) son desarrollados por Alibaba Cloud, una empresa china que ha estado trabajando en diversos modelos abiertos. Recientemente han dado un gran salto en la calidad de sus modelos. Este en concreto es la versión 2.5 de 7 mil millones de

¹⁶<https://huggingface.co/mistralai/Ministral-8B-Instruct-2410>

¹⁷<https://huggingface.co/microsoft/Phi-3-small-128k-instruct>

parámetros, de nuevo diseñada para tareas de instrucción.

Lo podemos encontrar en “Qwen/Qwen2.5-7B-Instruct”.

4.5.5 DeepSeek R1 Distill Qwen 3 8B

DeepSeek R1 Distill Qwen 3 8B (DeepSeek-AI, 2025) es un modelo desarrollado por DeepSeek AI, una empresa china de IA que en los últimos años ha estado trabajando en modelos de código abierto. A inicios de 2025 lanzaron su modelo R1, que es un modelo razonador, que emplea la técnica de cadena de pensamiento (*Chain of Thought*, CoT) para mejorar la calidad de las respuestas. Cuando se lanzó este modelo se hizo un gran revuelo en la comunidad, ya que era capaz de superar a muchos otros modelos que eran considerados de vanguardia, y todo bajo un coste de computación mucho menor.

Este modelo en concreto es una versión destilada de Qwen 3, lo que significa que se ha utilizado un proceso en el que el modelo original, Qwen 3, ha sido utilizado para entrenar y transferir su conocimiento a un modelo más pequeño y eficiente. Según la documentación, este modelo utiliza la misma arquitectura que Qwen 3 en su versión de 8B parámetros, pero utiliza un tokenizador (que es el componente que convierte el texto en tokens que el modelo puede entender) propio de DeepSeek R1.

Su repositorio de Hugging Face es “deepseek-ai/DeepSeek-R1-0528-Qwen3-8B”.

5 Desarrollo

En este capítulo se detallan los procesos de desarrollo que se han llevado a cabo para cumplir cada uno de los objetivos planteados. Estará dividido en secciones que cubrirán cada uno de los objetivos desarrollados, explicando los pasos seguidos y las herramientas utilizadas.

5.1 Inferencia de modelos

En primer lugar se decidió llevar a cabo las pruebas sobre modelos ya existentes. Antes de nada se escogieron los modelos a utilizar, que son los que encontramos explicados en la sección 4.5, y se implementó una forma de realizar las peticiones de facilitación a lectura fácil con los modelos mediante código Python. Estos modelos concretos fueron elegidos por diversos motivos, principalmente debido a la fácil disponibilidad (modelos abiertos), al hecho de que son multilingües (para el procesamiento de textos en español) y a su tamaño relativamente fácil de manejar. Los modelos escogidos fueron: **LLaMA 3.1 8B Instruct**, **Mistral 8B Instruct**, **Phi 3 Small 128K Instruct**, **Qwen 2.5 7B Instruct** y **DeepSeek R1 Distill Qwen 3 8B**.

En un principio, se intentó utilizar los modelos de Hugging Face utilizándolos de forma local, esto se realizó mediante el uso de la librería `transformers`, que permite importar modelos a partir del nombre del repositorio de Hugging Face. Tras bastantes intentos resultó obvio que los recursos de hardware de los que disponía no eran suficientes para poder realizar las pruebas, por ello decidí buscar una alternativa que fuera más eficiente. La primera opción fue hacer uso de Google Colab, que mediante cuadernos de Jupyter permite ejecutar el código que se le proporcione en la nube, sin embargo descubrí que la plataforma de Hugging Face ofrece una opción más sencilla y directa para este tipo de tareas y que me venía mucho mejor debido a su rapidez, que es el uso de su API de inferencia. Esta API permite realizar peticiones a algunos de los modelos disponibles, sin necesidad de tener que descargar nada.

Para poder hacer uso de la API de inferencia, es necesario tener una cuenta en Hugging Face y generar un token de acceso. Teniendo esto ya se pudo hacer uso de la misma mediante la clase `InferenceClient` de la librería `huggingface_hub`, con la que se hacen peticiones HTTP de forma sencilla que contienen los prompts para los modelos. Una vez implementado esto, se buscó alternativas para poder hacer uso de otros modelos que no estaban directamente disponibles en Hugging Face, por lo que finalmente se llegaron a utilizar también los servicios de API de GitHub, GitHub Models (s.f.), los servicios de API de Mistral y los de NVIDIA.

El funcionamiento del código es sencillo: se obtiene un texto sin procesar del corpus de lectura fácil (ClearSim, en la sección 4.4), el texto se envía a las diferentes funciones implementadas para cada modelo, y por último se almacenan los resultados de todos los

modelos junto con el texto original y el facilitado original en un archivo de formato JSON para facilitar su posterior análisis. Este proceso se repite para los 500 textos del corpus. La estructura final del archivo JSON la tenemos representada en la tabla 5.1.

Campo	Descripción
original	Texto original sin procesar
facilitado	Versión del texto adaptada manualmente a lectura facilitada
mistral	Texto generado por Mistral 8B Instruct
llama3	Texto generado por LLaMA 3.1 8B Instruct
phi	Texto generado por Phi 3 Small 128K Instruct
qwen25	Texto generado por Qwen 2.5 7B Instruct
deepseek	Texto generado por DeepSeek R1 Distill Qwen 3 8B

Tabla 5.1: Estructura del archivo JSON generado.

En cuanto al prompt utilizado para la generación de los textos, se optó por utilizar un gran prompt que contiene toda la información necesaria para que el modelo realice la facilitación. Se han introducido ciertas instrucciones para que el modelo pueda generar el texto de la forma que se espera, aquí tenemos una lista de las “partes” que componen el prompt:

1. **Contexto:** Se introduce un rol en el prompt para el modelo, indicando que se trata de un experto en facilitación de lectura fácil.
2. **Tarea:** Se indica la tarea que debe realizar, que es simplificar el texto original, con detalles sobre cómo debe hacerlo, como por ejemplo responder solamente con el texto facilitado y no con explicaciones adicionales.
3. **Reglas de facilitación:** Se especifican una a una las distintas reglas de Lectura Fácil que se deben aplicar. Estas serían las reglas que se explicaron en la sección 2.1.1.
4. **Texto original:** Finalmente, se introduciría el texto original que se debe facilitar, dando paso a la generación del texto facilitado.

5.2 Evaluación de modelos

Para evaluar los modelos, se implementaron una serie de *scripts* en Python que permiten calcular las métricas de evaluación que planteamos: BLEU, ROUGE, METEOR y BERTScore. Para cada una se programó un pequeño script que recibe el archivo JSON con los resultados de la inferencia completa de los 500 textos y calcula las métricas correspondientes para cada uno de los modelos. El desarrollo del cálculo fue sencillo ya que existen librerías que implementan las métricas y permiten usarlas de forma muy simple, como es el caso de `nltk` para BLEU y METEOR, `bert-score` para BERTScore, y `rouge_score` para ROUGE.

De esta forma, para cada uno de los textos del corpus se calcula el valor de la métrica (o métricas) comparando el texto facilitado manualmente con el texto facilitado de cada modelo. Finalmente, una vez teniendo las métricas de cada modelo, se utilizan las funciones

de la librería `numpy` para calcular la media, desviación típica, y los valores mínimo y máximo.

Cabe destacar que para el uso de la métrica ROUGE, que usualmente utiliza *WordNet* para su cálculo, se agregó el uso del Open Multilingual Wordnet (Bond y Foster, 2013), una base de datos similar a WordNet que utiliza datos de muchos otros idiomas aparte del inglés, incluyendo el español. De esta manera nos aseguramos su correcto funcionamiento para capturar los sinónimos en nuestro idioma.

5.3 GAN

Una vez realizadas las pruebas con los modelos de lenguaje existentes, el objetivo era intentar desarrollar una solución propia que pudiera servir para la facilitación de lectura. Para ello, se propuso distintos enfoques basados en ML, y el primero que se implementó fue un sistema de GAN.

El desarrollo de esta solución planteó varios desafíos en cuanto a los resultados y el rendimiento del modelo, que se abordarán en el capítulo sobre los resultados obtenidos. La implementación se basó en la combinación de un modelo **generador** (G) construido sobre el modelo T5¹ y un modelo **discriminador** (D) basado en BERT, formando la arquitectura adversativa adaptada para la tarea de facilitación que se muestra en la figura 5.1. Por supuesto, el corpus de datos utilizado fue el corpus ClearSim, que se preparó para el entrenamiento del sistema.

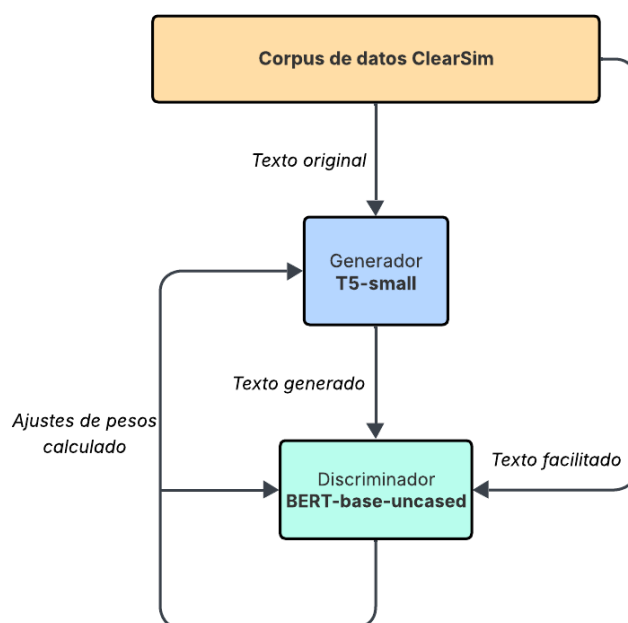


Figura 5.1: Diagrama de la arquitectura del sistema de GAN.

¹[https://en.wikipedia.org/wiki/T5_\(language_model\)](https://en.wikipedia.org/wiki/T5_(language_model))

El funcionamiento general del sistema es el siguiente: el generador recibe un texto original e intenta crear un texto facilitado, tras esto el discriminador recibe o bien un texto facilitado manualmente o uno generado por el generador, intentando detectar si dichos textos son artificiales. Si el discriminador consigue discernir correctamente los textos generados, el generador ajusta sus parámetros para poder mejorar y “engañarlo” la próxima vez. Además del aprendizaje adversarial, el generador también se entrena en cierta medida de forma supervisada comparando sus salidas con textos simplificados reales, para mejorar su capacidad de generar textos de forma más rápida. A lo largo de varias épocas este proceso hace que el generador vaya aprendiendo a producir textos más similares a los simplificados por humanos.

Para crear el generador se utilizó **T5-small**, conocido como *Text-to-Text Transfer Transformer* e introducido por primera vez por Raffel y cols. (2020). Se trata de un modelo que utiliza la arquitectura *transformer* preentrenado para abordar tareas de PLN como problemas de texto a texto. La elección de T5 se debe a su versatilidad y capacidad para manejar tareas de generación de texto, además de ser un modelo ligero. El generador se configuró con un prompt específico para la tarea y utiliza parámetros de generación adaptados para la producción del texto.

El discriminador se implementó utilizando **BERT-base-uncased** como base, añadiendo una capa de clasificación que determina si un par de textos (original y simplificado) se trata de una facilitación auténtica o generada automáticamente. La arquitectura del discriminador incluye una capa de dropout con probabilidad “0.3” para prevenir el sobreajuste y una capa de clasificación lineal que produce una salida binaria mediante una función sigmoide, de forma que el modelo pueda asignar un valor de autenticidad a cada par de textos. El *dropout* es una técnica que consiste en desactivar aleatoriamente un porcentaje de neuronas durante el entrenamiento para evitar el sobreajuste, lo que mejora el desempeño del modelo sobre datos fuera del conjunto de entrenamiento. Por otra parte, la función sigmoide se utiliza para convertir la salida del modelo en una probabilidad entre 0 y 1, donde valores cercanos a 1 indican que el par de textos es considerado auténtico y valores cercanos a 0 indican que es considerado generado. La fórmula de la función sigmoide sería

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (5.1)$$

donde x es la salida numérica de las capas del modelo antes de aplicar la función sigmoide. Es una de las funciones de activación más comúnmente aplicadas en redes neuronales, especialmente en tareas de clasificación binaria como esta.

El proceso de entrenamiento se estructuró de forma alternada, optimizando primero el discriminador con ejemplos reales y generados, y después el generador con una función de pérdida. La implementación incluye técnicas para manejar textos de longitud variable (*padding*) y de segmentación para procesar textos excesivamente largos, dividiéndolos en segmentos manejables con solapamiento para mantener la coherencia de su contexto. Esta segmentación divide los textos en fragmentos de hasta 400 palabras con un solapamiento de 50 palabras entre segmentos consecutivos, esto es especialmente relevante dado que muchos textos del corpus ClearSim exceden la longitud máxima de tokens de los modelos base.

La validación del modelo se lleva a cabo utilizando métricas específicas para simplificación de texto, incluyendo puntuaciones ROUGE para medir la similitud con las referencias de facilitación, y métricas de legibilidad para evaluar la mejora en la comprensión del texto, utilizando el índice de Flesch Reading Ease para comparar la diferencia entre el texto original y el generado por el modelo.

El desarrollo completo se llevó a cabo en Google Colab, siendo el tiempo del proceso de entrenamiento final algo superior a una hora aprovechando las GPU A100 disponibles para acelerar el entrenamiento. Respecto a esto, en pruebas anteriores se encontraron problemas de memoria, ya que en un principio la implementación estaba bastante mal optimizada, lo que provocaba que la memoria VRAM de la GPU se saturara rápidamente. El resultado final del entrenamiento fue un modelo que posteriormente se guardó en un repositorio de Hugging Face para su uso posterior. El repositorio es “Nizaress/e2r-gan”, disponible en el siguiente enlace: <https://huggingface.co/Nizaress/e2r-gan>.

5.4 Ajuste fino

Otra de las soluciones propias que propuse fue realizar un ajuste fino, o *fine-tuning*, sobre un modelo de lenguaje relativamente grande. El objetivo es tener una base sólida de entrenamiento mediante el modelo original, y a la vez intentar mejorar el rendimiento en la tarea de facilitación de lectura. El modelo elegido para este proceso fue **LLaMA 2 7B**. La elección de este modelo en concreto se debe al hecho de que, como los modelos que hemos evaluado, se trata de una opción popular y compatible con el lenguaje español, pero además es un modelo algo más ligero y más antiguo de los demás. Esto supone que haya más documentación, guías, herramientas y librerías disponibles que son compatibles con este modelo, además de que por su tamaño supone una carga menor de computación al trabajar con él, que dada la intensidad de trabajo en VRAM que utilizan este tipo de tareas sobre el hardware es un punto muy positivo a favor.

El proceso de ajuste fino se implementó utilizando técnicas de entrenamiento eficiente, específicamente **QLoRA** (Quantized Low-Rank Adaptation), que como ya se explicó en la sección 2.7, combina la técnica de LoRA con cuantización para reducir el uso de memoria durante el entrenamiento. La librería de Python **peft** fue la que proporcionó la funcionalidad para implementar LoRA, y el entrenamiento se llevó a cabo mediante el uso de **SFTTrainer** de la librería **trl**, que está optimizada para el ajuste fino de modelos de lenguaje.

Las configuraciones de parámetros de entrenamiento y de QLoRA ha sido la parte del desarrollo que más problemas ha supuesto, ya que se tuvo que realizar numerosas pruebas fallidas debido a que, al estar configurado incorrectamente, la memoria de la GPU no era suficiente para llevar a cabo las tareas, incluso aunque se estuvieran utilizando GPUs bastante potentes de entornos de Google Colab. Finalmente, la configuración de valores de parámetros de QLoRA que resultó exitosa fue la que podemos observar en la tabla 5.2.

Y en cuanto a los parámetros de entrenamiento, también se ajustaron para poder llevar a cabo el proceso sin consumir toda la memoria, estableciendo el *batch size* a solamente 1 y

Parámetro	Valor
Dimensión de atención LoRA	64
Parámetro alfa LoRA	16
Probabilidad de dropout	0.1
Bits de cuantización	4
Tipo de cuantización	nf4
Compute dtype	float16

Tabla 5.2: Parámetros de configuración QLoRA.

además activando el *gradient checkpointing* para reducir el uso de memoria.

El conjunto de datos utilizado para el entrenamiento se preparó esta vez en formato JSONL² (JSON Lines), conteniendo pares de texto original y texto facilitado del corpus ClearSim. Cada entrada del dataset se estructuró con un prefijo a modo de prompt (se trata del mismo prompt que se ha utilizado anteriormente para otras tareas) que indica el contexto, las tarea a realizar y las reglas de facilitación a seguir.

Para la configuración del tokenizador se añadió el uso de *padding*. Esto es necesario para que el modelo pueda manejar entradas de diferentes longitudes de forma correcta, añadiendo espacio a la derecha de las entradas más cortas hasta que completen la longitud.

Teniendo todo configurado ya se pudo ejecutar el entrenamiento sin problemas. Una vez completado el entrenamiento, el modelo PEFT resultante se envió a un repositorio de Hugging Face para poder utilizarlo posteriormente. El repositorio es “Nizaress/e2r-finetuned”, disponible en el siguiente enlace: <https://huggingface.co/Nizaress/e2r-finetuned>.

5.5 Capa de servicios y Prototipo

El objetivo durante esta etapa del desarrollo era implementar un prototipo que permitiera utilizar las soluciones trabajadas en las secciones anteriores de forma cómoda y sencilla a través de una capa de servicios.

Esta parte del proyecto resultó bastante simple en comparación a todo lo desarrollado anteriormente. A la hora de implementar la capa de servicios para acceder a las soluciones desarrolladas se decidió utilizar la librería FastAPI, que permite crear APIs de forma sencilla, y además haciendo uso de *uvicorn* podemos ejecutarlo como un servidor local asíncrono, quedando disponible para recibir peticiones HTTP en el puerto 8000.

Se establecieron dos rutas principales en la API implementada, una para realizar la simplificación de texto utilizando el modelo LLaMA 3.1 mediante la API de inferencia de Hugging Face y otra para hacer uso del modelo de ajuste fino que se desarrolló anteriormente de forma local,

²<https://jsonlines.org/>

siendo respectivamente:

- `/llama`: Ruta para facilitar con la API de Hugging Face.
- `/finetune`: Ruta para facilitar localmente con el modelo de ajuste fino.
- `/gan`: Ruta para facilitar localmente con el modelo de GAN.

Ambas rutas reciben un texto sin procesar y devuelven el texto facilitado, además de recibir el token de acceso de Hugging Face en el caso del uso de inferencia.

En cuanto al prototipo, se implementó una sencilla interfaz web utilizando HTML, CSS y JavaScript, que permite introducir un texto y un token, y recibir el texto facilitado, enviando las peticiones a la API implementada.

6 Resultados

En este capítulo se presentan los resultados obtenidos de las distintas pruebas realizadas a los modelos de lenguaje y sistemas de IA generativa desarrollados. Se incluyen métricas de evaluación, análisis de rendimiento y comparaciones entre los diferentes enfoques implementados.

6.1 Estudio de LLMs

En primer lugar, vamos a presentar los resultados obtenidos del estudio de los modelos de lenguaje ya existentes. Como ya vimos en la sección 5.2, se utilizaron una serie de métricas de evaluación para medir la calidad de los textos generados por los modelos. Estas distintas métricas se calcularon comparando los textos generados con los facilitados manualmente del corpus ClearSim. De esta forma podemos medir cómo de cercano es el texto generado a la facilitación que consideramos correcta, de forma que podemos utilizarlo como medida de la calidad de la facilitación por el modelo. Aunque no es una métrica perfecta, ya que una facilitación puede ser correcta y no coincidir con la facilitación del corpus, sí que nos permite tener una idea de cómo de bien lo realiza el modelo.

En las siguientes tablas se presentan los resultados obtenidos del estudio de las métricas BLEU (6.1), ROUGE (6.2), METEOR (6.3) y BERTScore (6.4) para cada uno de los modelos.

Tabla 6.1: Resultados de evaluación BLEU.

Modelo	Media	Desv. Est.	Min	Max
Mistral	0.1845	0.0832	0.0000	0.4830
LLaMA 3.1	0.1673	0.0935	0.0000	0.5971
Phi 3	0.1518	0.0833	0.0000	0.5194
Qwen 2.5	0.1875	0.0804	0.0000	0.4913
DeepSeek R1	0.0893	0.0638	0.0000	0.3024

Tabla 6.2: Resultados de evaluación ROUGE.

Modelo	ROUGE-1	ROUGE-2	ROUGE-L
Mistral	0.5394	0.3112	0.3687
LLaMA 3.1	0.5167	0.2879	0.3391
Phi 3	0.5064	0.2643	0.3202
Qwen 2.5	0.5515	0.3034	0.3694
DeepSeek R1	0.3929	0.1848	0.2309

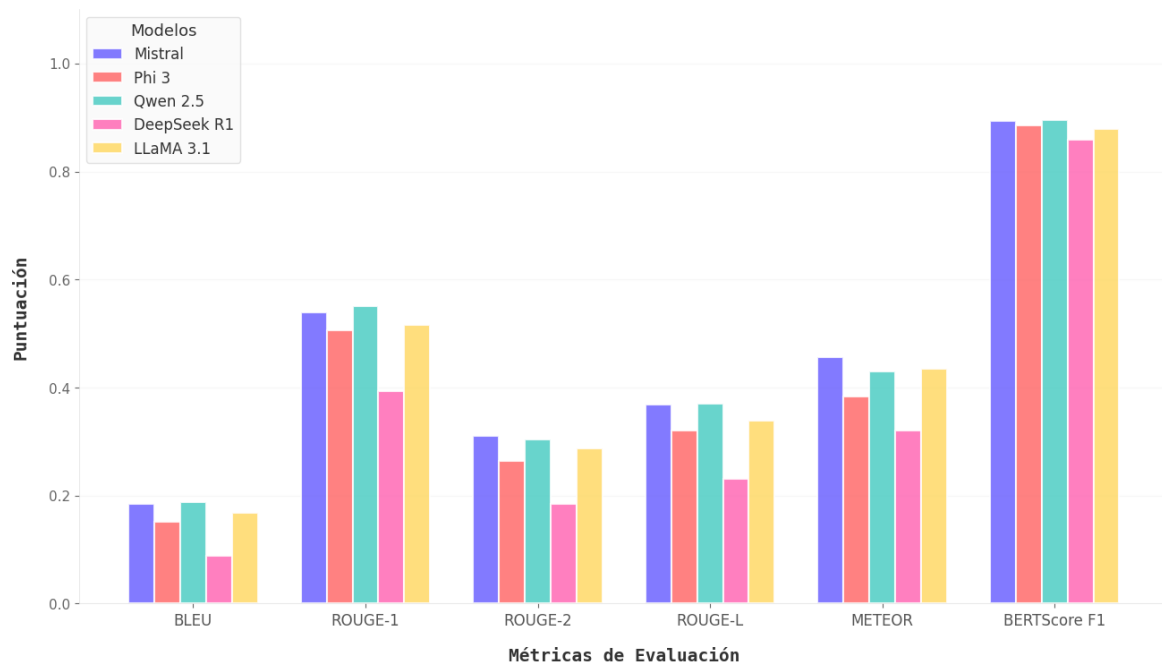
Tabla 6.3: Resultados de evaluación METEOR.

Modelo	Media	Desv. Est.	Min	Max
Mistral	0.4566	0.0912	0.1579	0.7120
LLaMA 3.1	0.4341	0.0893	0.2315	0.7292
Phi 3	0.3835	0.1061	0.1144	0.6474
Qwen 2.5	0.4296	0.1038	0.0324	0.7241
DeepSeek R1	0.3208	0.1077	0.0413	0.5689

Tabla 6.4: Resultados de evaluación BERTScore (F1).

Modelo	Media	Desv. Est.	Min	Max
Mistral	0.8937	0.0177	0.8304	0.9426
LLaMA 3.1	0.8793	0.0183	0.8384	0.9517
Phi 3	0.8857	0.0197	0.8224	0.9562
Qwen 2.5	0.8958	0.0188	0.8269	0.9637
DeepSeek R1	0.8583	0.0294	0.7915	0.9139

Además, para poder visualizar mejor los resultados obtenidos se ha realizado una gráfica comparativa de los resultados de los modelos agrupados según cada métrica. En la figura 6.1 podemos observar cómo se comportan los distintos modelos en cada una de las métricas.

**Figura 6.1:** Comparativa de resultados de LLMs.

A partir de los resultados obtenidos, podemos observar que el modelo Qwen 2.5 ha obtenido los mejores resultados en la mayoría de las métricas, seguido por Mistral y LLaMA 3.1. Específicamente, Qwen 2.5 despunta en las métricas BLEU, ROUGE-1, ROUGE-L y BERTScore, mientras que Mistral despunta en ROUGE-2 y METEOR.

Por otro lado, el modelo DeepSeek R1 ha obtenido los peores resultados en todas las métricas, lo que probablemente signifique que no es tan efectivo como los otros modelos. El motivo al que se puede deber esto es el hecho de que este sea un modelo destilado de otro original (Qwen 3 8B). Usualmente los modelos destilados suelen tener un rendimiento inferior al modelo original, ya que se busca reducir el tamaño del modelo a costa de una pérdida de calidad. Al parecer, en este caso, la pérdida de calidad ha sido notable.

Comentando los resultados de las métricas, observamos ciertas tendencias comunes y diferencias relevantes entre ellas. Por ejemplo, en las métricas más estrictas basadas en coincidencias de n-gramas (como BLEU y ROUGE) los valores tienden a ser más bajos. Esto es esperable dado a que una facilitación no siempre usará las mismas palabras que otra, ya que el lenguaje es muy variable. Por otra parte, métricas como METEOR y especialmente BERTScore, que tienen en cuenta la semántica y el contexto de los textos, han tenido puntuaciones bastante más altas, lo puede significar que los modelos son capaces de generar textos con significado similar al de las facilitaciones de referencia, aunque no haya coincidencia exacta en las palabras utilizadas. Aunque esto no significa que la facilitación sea correcta, ya que al fin y al cabo el contexto de la facilitación será similar al del texto original, por lo que es normal que los modelos generen textos con significado muy similar.

6.2 Estudio de sistemas propios

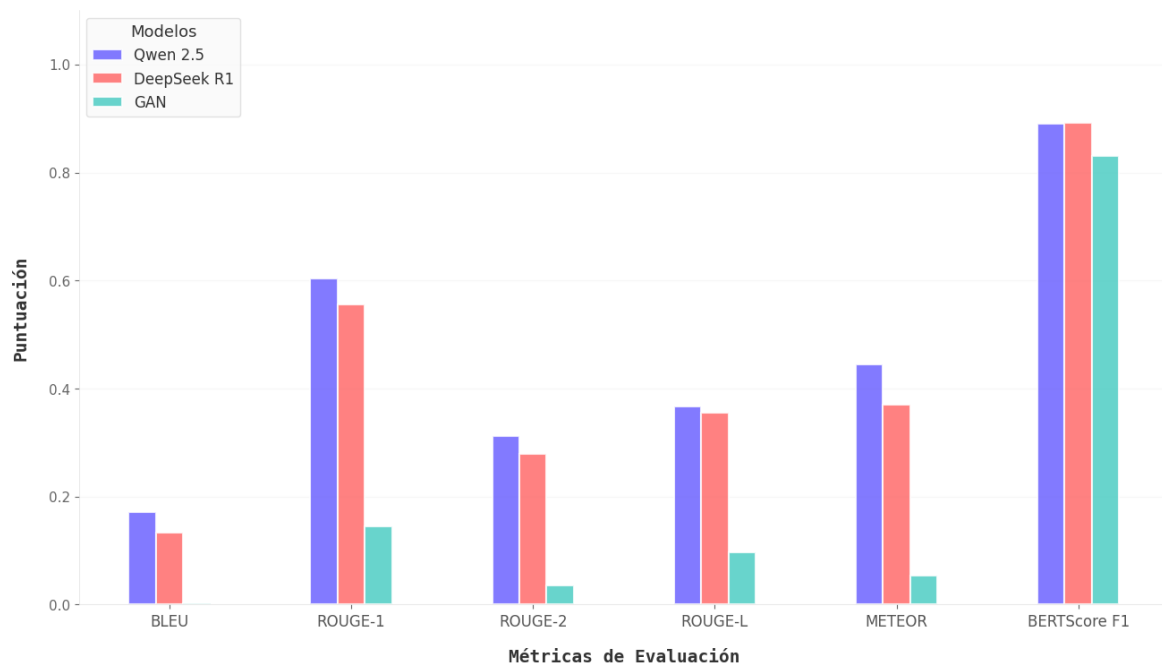
A continuación, se presentaran los resultados obtenidos de los sistemas propios desarrollados para la generación de textos en lectura facilitada. Estos sistemas incluyen un modelo de GAN y un modelo al que se le ha realizado un ajuste fino (*fine-tuning*) utilizando QLoRA.

6.2.1 GAN

El sistema de GAN desarrollado no ha obtenido resultados satisfactorios. Los resultados obtenidos por el modelo basado en GAN son comparables a los de los modelos de lenguaje ya existentes. El modelo no ha sido capaz de generar textos de calidad. Tenemos un ejemplo de texto generado por el modelo en la tabla 6.5, y una gráfica comparativa de los resultados en la figura 6.2, vemos el modelo de GAN comparado con los LLMs que peores y mejores resultados han obtenido, donde curiosamente vemos que DeepSeek R1 ha obtenido un resultado mucho mejor en este conjunto de datos que en el corpus completo, llegando incluso a superar a Qwen 2.5 en la métrica de BERTScore.

Tabla 6.5: Ejemplos de entrada y salida del modelo basado en GAN.

Entrada al modelo	Salida generada
“Alicante, 9 de febrero de 2023. La obra ganadora de varios Oscars ‘Ghost, el Musical’ protagonizada por David Bustamante...”	``Alicante, 9 de febrero de 2023. La obra ganadora de varios Oscars ‘Ghost, el Musical’ protagonizada por David Bustamante...”
“Prueba”	``simplifie simplifie: Prueba"
“Texto a ser simplificado, facilitando su lectura mediante las reglas de Lectura Facil para hacer que sea más accesible por los lectores.”	``Texto a ser simplificado, facilitando su lectura medianta las reglas de Lectura Facil para hacer que sea más accesible por los lectores."

**Figura 6.2:** Comparativa de resultados del modelo de GAN con LLMs.

Como vemos en la tabla 6.5, el modelo no es capaz de generar salidas bien facilitadas. En el primer caso el modelo apenas altera el texto original, y en los otros casos o bien genera texto innecesario que no aporta información relevante (“simplifie simplifie:”) o bien genera un texto casi idéntico al original pero añadiendo errores ortográficos (“medianta” en lugar de “mediante”).

6.2.2 Ajuste fino

Los resultados obtenidos del modelo ajustado con QLoRA se presentan en la tabla 6.6. En la tabla se muestran las métricas de evaluación obtenidas al comparar los textos generados por el modelo con los textos facilitados manualmente del corpus ClearSim, de la misma forma que se ha hecho con los modelos de lenguaje ya existentes. La diferencia es que en este caso se ha utilizado un conjunto de datos de prueba reducido en comparación al utilizado para los modelos de lenguaje, ya que el ajuste fino se ha realizado con gran parte del corpus. Además, se presenta una gráfica comparativa entre los resultados del modelo de ajuste fino con los LLMs que peores y mejores resultados han obtenido en la figura 6.3.

Métrica	Media	Desv. Est.	Min	Max
BLEU	0.0717	0.0123	0.0601	0.0887
ROUGE-1 (F1)	0.4141	—	—	—
ROUGE-2 (F1)	0.1703	—	—	—
ROUGE-L (F1)	0.2303	—	—	—
METEOR	0.2002	0.0102	0.1862	0.2100
BERTScore (F1)	0.8589	0.0112	0.8433	0.8689

Tabla 6.6: Resultados de evaluación del modelo de ajuste fino.

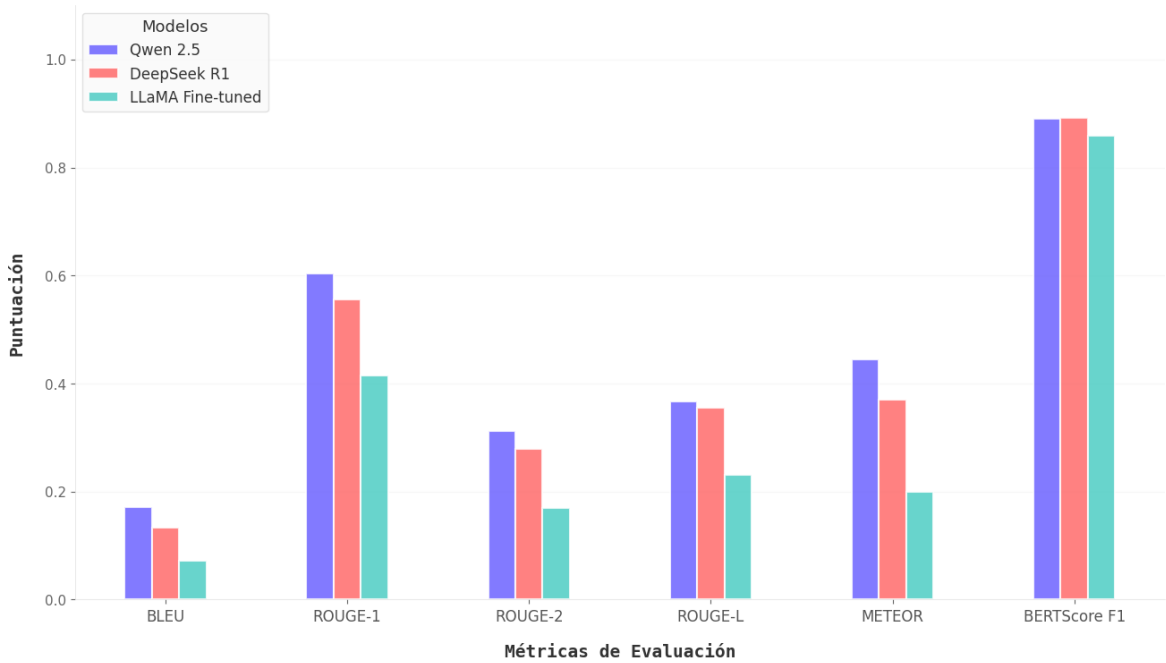


Figura 6.3: Comparativa de resultados del modelo de ajuste fino con LLMs.

Como podemos ver en la tabla y la figura, los resultados obtenidos del modelo de ajuste

fino son bastante bajos en comparación con los modelos de lenguaje ya existentes, aunque son claramente mejores que los del modelo de redes GAN. Esto es esperable, ya que el modelo de ajuste fino se ha entrenado con un conjunto de datos reducido (menos de 500 textos). Como vemos, los valores de las métricas ROUGE y METEOR son bastante bajos, lo que indica que el modelo no es capaz de generar textos con una calidad similar a la de los modelos de lenguaje ya existentes. Sin embargo, el modelo ha obtenido un valor de BLEU cercano al que ha obtenido el modelo DeepSeek. Además ha obtenido un valor de BERTScore relativamente alto, lo que indica que el modelo ha generado textos con un significado similar al de las facilitaciones de referencia, pero de nuevo, esto no tiene relevancia ya que el contexto de la facilitación sería muy similar al del texto original, por lo que es normal que los modelos generen textos con significado muy similar.

A pesar de tener resultados no tan negativos en algunas métricas, el desempeño del modelo no es para nada comparable al de los LLMs estudiados anteriormente. De hecho, el modelo ajustado encuentra problemas a la hora de generar textos que sean si quiera coherentes. Si observamos los textos generados por el modelo, podemos ver que en muchos casos no son ni siquiera coherentes, y en otros casos son demasiado simples y no aportan información relevante. Tenemos un ejemplo de un texto generado por el modelo.

Tabla 6.7: Ejemplos de entrada y salida del modelo de ajuste fino.

Entrada al modelo	Salida generada
prompt + “Alicante, 9 de febrero de 2023. La obra ganadora de varios Oscars ‘Ghost, el Musical’ protagonizada por David Bustamante, Ricky Merino y Ana Dachs...”	``\n \n La obra ganadora de varios Oscars ‘Ghost, el Musical’ protagonizada por David Bustamante, Ricky Merino y Ana Dachs...”
prompt + “Esta es una de las pruebas de simplificación que estamos haciendo.”	``Aquí encontrarás la versión original y la versión simplificada. \n La versión simplificada: \n Esta es una de las pruebas de simplificación que estamos haciendo.”

Como vemos en la tabla 6.7, el modelo no es capaz de generar salidas bien facilitadas. En el primer caso el modelo apenas altera el texto original, y en el segundo caso genera texto innecesario que no aporta información relevante. El desempeño del modelo es pobre, como ya hemos comentado el motivo de esto es que el modelo ha sido ajustado con un conjunto de datos reducido, lo que no le permite aprender a generar textos de calidad. Sin embargo, dados los pocos recursos de los que disponía, no se podía esperar un resultado mucho mejor.

6.3 Prototipo

El prototipo desarrollado para el uso de los distintos sistemas es sencillo pero funcional. Permite enviar texto de forma cómoda y obtener una versión facilitada del mismo en el cuadro inferior.

La mayor desventaja de esta interfaz es el uso del modelo en local como es el caso del desarrollado con QLoRA, que al ser un modelo relativamente pesado, tarda un tiempo en generar la respuesta y además requiere de recursos computacionales abundantes para poder funcionar. En mi caso, al no disponer de una GPU he tenido que utilizar un entorno de Google Colab para poder realizar las pruebas de esta funcionalidad local.

Tenemos una muestra de la interfaz en la figura 6.4. En esta imagen podemos ver cómo se puede introducir un texto y el token, y obtener una versión facilitada del texto original.

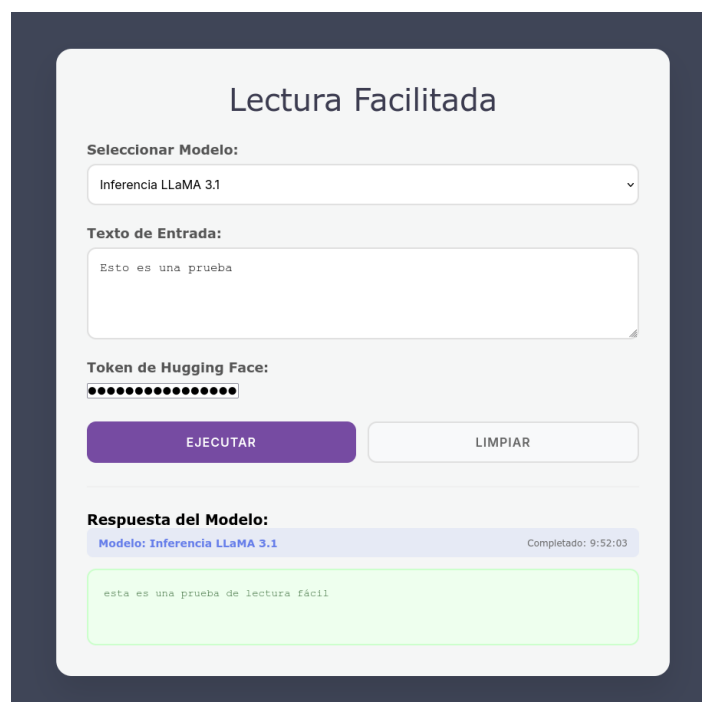


Figura 6.4: Interfaz del prototipo de la aplicación.

No se recomienda utilizar el modelo de ajuste fino ni el de GAN en el prototipo, ya que a menos que se disponga de una GPU potente y bastante memoria RAM, la ejecución puede ser muy lenta o incluso fallar al consumir demasiados recursos. En caso de querer utilizar esta interfaz para probar la inferencia, o en caso de querer revisar el código, se puede encontrar todo el código fuente implementado para todo el proyecto (incluyendo las secciones anteriores) en el repositorio de GitHub del proyecto, a través del siguiente enlace: <https://github.com/nizaress/e2r-tfg>.

7 Conclusiones

Para concluir con este trabajo, repasaremos los objetivos planteados al inicio y analizaremos si se han cumplido, así como las dificultades encontradas y las posibles mejoras a futuro.

7.1 Objetivos cumplidos

El objetivo principal de este trabajo era evaluar una serie de modelos de lenguaje y diseñar sistemas para la producción de textos en lectura facilitada. Para ello, se plantearon diferentes tareas concretas que se han trabajado en este proyecto. A lo largo del desarrollo se han cumplido los siguientes objetivos específicos:

- Se ha implementado un script de inferencia que permite procesar un texto a través del uso de modelos de lenguaje.
- Se ha diseñado un sistema de IA generativa basado en la arquitectura de GAN.
- Se ha diseñado un sistema de PEFT para la adaptación de un modelo mediante ajuste fino.
- Se han evaluado los modelos de lenguaje seleccionados utilizando métricas de evaluación como BLEU, ROUGE y METEOR, obteniendo resultados que permiten comparar la calidad de los textos generados.
- Se ha diseñado un prototipo que permite a los usuarios introducir un texto y recibir una versión facilitada a través de una interfaz web sencilla.

7.2 Dificultades encontradas

Durante el desarrollo del trabajo se han encontrado numerosos problemas, entre los cuales podemos destacar los siguientes:

- La limitación de recursos computacionales para manejar modelos de gran tamaño, lo que ha llevado a seleccionar modelos más ligeros y eficientes.
- La complejidad de la adaptación de los modelos a tareas específicas de lectura facilitada.
- El funcionamiento incorrecto de los modelos propios desarrollados, que ha requerido numerosos ajustes y entrenamientos, y no se ha podido completar satisfactoriamente.

7.3 Mejoras a futuro

En cuanto a las posibles mejoras que se podrían llevar a cabo en un futuro, se pueden considerar las siguientes:

- Ampliar el conjunto de modelos evaluados (incluyendo modelos de mayor tamaño) para realizar un estudio más exhaustivo de las capacidades de los modelos más punteros en este tipo de tareas.
- Mejorar el entrenamiento de los modelos propios, utilizando técnicas más avanzadas de ajuste fino y haciendo uso de conjuntos de datos más amplios.
- Mejorar la interfaz del prototipo para que sea más accesible, y expandir sus funcionalidades incluyendo opciones de configuración para personalizar la generación de los textos.

7.4 Reflexiones finales

En resumen, este trabajo ha permitido explorar el potencial de los modelos de lenguaje en la facilitación de textos. A pesar de que los resultados obtenidos de los sistemas implementados hayan dejado mucho que desear, esto me ha permitido aprender mucho sobre el funcionamiento de los modelos de lenguaje y muchos otros conceptos relacionados con el PLN y la IA generativa que se han aplicado en este trabajo.

Como vimos en los resultados de los LLMs, los modelos de lenguaje como Mistral y Qwen 2.5 han demostrado ser bastante efectivos en la generación de textos facilitados considerando el hecho de que son versiones ligeras dentro de sus respectivas “familias” de modelos, de entre 7 y 8 mil millones de parámetros. Esto indica que los modelos de lenguaje pueden ser herramientas valiosas para este tipo de tareas, y que su uso puede contribuir bastante a la mejora de la accesibilidad de la información a las personas si se utilizan adecuadamente.

Bibliografía

- AENOR. (2018). *Norma UNE 153101:2018 EX*. Descargado 2025-05-08, de <https://www.une.org/encuentra-tu-norma/busca-tu-norma/norma/?c=N0060036>
- AI@Meta. (2024). Llama 3 Model Card. Descargado de https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md
- Asociación Lectura Fácil. (s.f.). *¿Qué es la Lectura Fácil (LF)?* Descargado 2025-05-08, de <https://www.lecturafacil.net/es/info/1-que-es-la-lectura-facil-lf/>
- Banerjee, S., y Lavie, A. (2005, junio). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. En J. Goldstein, A. Lavie, C.-Y. Lin, y C. Voss (Eds.), *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization* (pp. 65–72). Ann Arbor, Michigan: Association for Computational Linguistics. Descargado de <https://aclanthology.org/W05-0909/>
- Bond, F., y Foster, R. (2013, agosto). Linking and extending an open multilingual Wordnet. En H. Schuetze, P. Fung, y M. Poesio (Eds.), *Proceedings of the 51st annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 1352–1362). Sofia, Bulgaria: Association for Computational Linguistics. Descargado de <https://aclanthology.org/P13-1133/>
- Botella-Gil, B., Espinosa-Zaragoza, I., Moreda, P., y Palomar, M. (2024). *GPLSI: Corpus ClearSim*.
- DeepSeek-AI. (2025). *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. Descargado de <https://arxiv.org/abs/2501.12948>
- Dettmers, T., Pagnoni, A., Holtzman, A., y Zettlemoyer, L. (2023). *QLoRA: Efficient Finetuning of Quantized LLMs*. Descargado de <https://arxiv.org/abs/2305.14314>
- Devlin, J., Chang, M., Lee, K., y Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, *abs/1810.04805*. Descargado de <http://arxiv.org/abs/1810.04805>
- GitHub Models. (s.f.). Descargado 2025-06-28, de <https://github.com/marketplace/models>
- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). *Generative adversarial networks*. Descargado de <https://arxiv.org/abs/1406.2661>

- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., y Chen, W. (2021). LoRA: Low-Rank Adaptation of Large Language Models. *CoRR*, *abs/2106.09685*. Descargado de <https://arxiv.org/abs/2106.09685>
- Lin, C.-Y. (2004, julio). ROUGE: A package for automatic evaluation of summaries. En *Text summarization branches out* (pp. 74–81). Barcelona, Spain: Association for Computational Linguistics. Descargado de <https://aclanthology.org/W04-1013/>
- NVIDIA, Vingelmann, P., y Fitzek, F. H. (2020). *Cuda, release: 10.2.89*. Descargado de <https://developer.nvidia.com/cuda-toolkit>
- Papineni, K., Roukos, S., Ward, T., y Zhu, W.-J. (2002, julio). Bleu: a method for automatic evaluation of machine translation. En P. Isabelle, E. Charniak, y D. Lin (Eds.), *Proceedings of the 40th annual meeting of the association for computational linguistics* (pp. 311–318). Philadelphia, Pennsylvania, USA: Association for Computational Linguistics. Descargado de <https://aclanthology.org/P02-1040/> doi: 10.3115/1073083.1073135
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, *21*(140), 1-67. Descargado de <http://jmlr.org/papers/v21/20-074.html>
- Team, Q. (2024, September). *Qwen2.5: A Party of Foundation Models*. Descargado de <https://qwenlm.github.io/blog/qwen2.5/>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2023). *Attention Is All You Need*. Descargado de <https://arxiv.org/abs/1706.03762>
- Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., y Artzi, Y. (2020). BERTScore: Evaluating Text Generation with BERT. En *International conference on learning representations*. Descargado de <https://openreview.net/forum?id=SkeHuCVFDr>
-

Lista de Acrónimos y Abreviaturas

ALF	Asociación Española de Lectura Fácil.
API	Interfaz de Programación de Aplicaciones.
BERT	Bidirectional Encoder Representations from Transformers.
BLEU	Bilingual Evaluation Understudy.
CSS	Cascading Style Sheets.
CUDA	Compute Unified Device Architecture.
GAN	Generative Adversarial Networks.
GPLSI	Grupo de Procesamiento del Lenguaje y Sistemas de Información.
HTML	HyperText Markup Language.
IA	Inteligencia Artificial.
IDE	Entorno de Desarrollo Integrado.
JSON	JavaScript Object Notation.
LLaMA	Large Language Model Meta AI.
LLM	Large Language Model.
LoRA	Low-Rank Adaptation.
METEOR	Metric for Evaluation of Translation with Explicit ORdering.
ML	Machine Learning.
NLTK	Natural Language Toolkit.
PEFT	Parameter-Efficient Fine-Tuning.
PLN	Procesamiento del Lenguaje Natural.
ROUGE	Recall-Oriented Understudy for Gisting Evaluation.
TFG	Trabajo de Fin de Grado.