Universidad Zaragoza
1474

Trabajo Fin de Grado

Aplicación de técnicas de Inteligencia Artificial para optimizar la generación de requisitos en el desarrollo de software

Application of Artificial Intelligence techniques to optimize requirements generation in software development

Autora

Elizabeth Lilai Naranjo Ventura

Directores

José Javier Merseguer Hernaiz
Jorge Raul Bernad Lusilla

Grado en Ingeniería Informática

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2025

AGRADECIMIENTOS

Quiero expresar mi más sincero agradecimiento, en primer lugar, a mis tutores José Merseguer y Jordi Bernad, por su guía constante, sus valiosos consejos y su confianza en mí durante este proyecto.

Asimismo, agradezco a Gregorio de Miguel por su participación en el formulario de validación y por permitirnos emplear los enunciados de sus exámenes y prácticas para el desarrollo de este trabajo. Y agradezco a Javier Nogueras también por completar el formulario.

Finalmente, mi gratitud más profunda a mi familia, pareja y amigos por su paciencia, comprensión y ánimo incondicional, que me han acompañado en cada paso del camino.

Aplicación de técnicas de Inteligencia Artificial para optimizar la generación de requisitos en el desarrollo de software

RESUMEN

La calidad de los requisitos en el desarrollo de software es un factor crítico que influye directamente en el éxito o fracaso de los proyectos, pero su redacción manual conlleva numerosos riesgos como ambigüedades, omisiones y errores humanos. Este Trabajo Fin de Grado explora el uso de técnicas de Inteligencia Artificial, y en particular de modelos de lenguaje de gran escala (LLMs), para asistir y automatizar la generación de requisitos a partir de descripciones en lenguaje natural de sistemas software.

Para ello, se ha llevado a cabo una comparación entre distintos modelos de lenguaje, incluyendo modelos de propósito general como ChatGPT y DeepSeek, y modelos de código abierto como LLaMA y Mistral. Sobre estos últimos, se ha aplicado fine-tuning mediante adaptadores LoRA y cuantización en 4 bits utilizando el dataset DaReC, especializado en requisitos de software. Asimismo, se ha diseñado una metodología de evaluación subjetiva basada en un formulario que analiza dimensiones como claridad, corrección gramatical, consistencia, cobertura, nivel de detalle y diferenciación entre requisitos funcionales y no funcionales.

Los resultados muestran que, si bien modelos generalistas como ChatGPT ofrecen una calidad elevada desde el inicio, los modelos entrenados específicamente para esta tarea, como Mistral tras el fine-tuning, logran mejorar la estructura y el nivel de detalle de sus salidas. Sin embargo, aún persisten limitaciones como la cobertura y la redacción técnica formal. Por otro lado, el análisis de acuerdo entre anotadores mediante el alfa de Krippendorff y el coeficiente de Spearman reveló un bajo nivel de consistencia, lo que pone de manifiesto la complejidad inherente a la evaluación subjetiva en este tipo de tareas.

En conclusión, este trabajo demuestra el potencial de los LLMs como herramientas de apoyo en el desarrollo del software, especialmente cuando se adaptan al dominio específico. A pesar de las limitaciones detectadas, los resultados obtenidos evidencian el potencial de una línea de trabajo que podría contribuir a reducir errores humanos, mejorar la calidad de los requisitos y aumentar la eficiencia del ciclo de vida del software desde sus primeras etapas.

Application of Artificial Intelligence techniques to optimize requirements generation in software development

ABSTRACT

The quality of requirements in software development is a critical factor that directly influences project success or failure. However, their manual specification entails numerous risks, such as ambiguities, omissions, and human errors. This Bachelor's Thesis explores the use of Artificial Intelligence techniques, particularly large language models (LLMs), to support and automate the generation of requirements from natural language descriptions of software systems.

To this end, a comparative analysis was carried out between different LLMs, including general-purpose models such as ChatGPT and DeepSeek, as well as open-source models such as LLaMA and Mistral. Fine-tuning was applied to the latter using LoRA adapters and 4-bit quantization on the DaReC dataset, which is specialized in software requirements. Additionally, a subjective evaluation methodology was designed through a questionnaire assessing dimensions such as clarity, grammatical correctness, consistency, coverage, level of detail, and the distinction between functional and non-functional requirements.

The results show that while general-purpose models like ChatGPT offer high-quality outputs from the outset, models specifically fine-tuned for this task, such as Mistral, demonstrated improvements in structure and level of detail. Nevertheless, limitations remain in terms of coverage and formal technical phrasing. Furthermore, the analysis of inter-annotator agreement using Krippendorff's alpha and Spearman's rank correlation coefficient revealed a low level of consistency, highlighting the inherent complexity of subjective evaluation in this context.

In conclusion, this work demonstrates the potential of LLMs as support tools in software engineering, especially when adapted to domain-specific tasks. Despite current limitations, the findings indicate a promising research direction that could help reduce human errors, enhance requirement quality, and increase development lifecycle efficiency from the earliest stages.

Tabla de contenidos

Capítulo 1

Introducción

Este capítulo contextualiza el trabajo realizado, destacando la relevancia de la Inteligencia Artificial (IA), y en particular de las técnicas de Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés) [1], en el ámbito de la ingeniería de requisitos [2]. Se plantea el problema abordado, motivado por las limitaciones de los métodos tradicionales en la generación de requisitos de software y, a continuación, se presentan los objetivos principales del proyecto, las fases en las que se ha estructurado el trabajo y la organización general del documento.

1.1. Contexto y motivación

La fase de definición y análisis de requisitos es una de las más críticas en el desarrollo de software, ya que establece las bases sobre las que se construirá el sistema final. Una especificación incorrecta, deficiente o ambigua de los requisitos de un sistema puede derivar en errores costosos, malentendidos entre los implicados, o incluso en la insatisfacción del cliente con el producto entregado. La calidad de los requisitos impacta directamente en el éxito de un proyecto de software [2].

Un ejemplo paradigmático de las consecuencias que puede acarrear una mala especificación es el fallo del cohete Ariane 5 en 1996 [3]. La misión fue abortada apenas 40 segundos después del despegue debido a un error en el software de navegación, provocado por una conversión de datos incorrecta no contemplada adecuadamente en los requisitos del sistema. El resultado fue la destrucción del vehículo y la pérdida de aproximadamente 370 millones de dólares. Este incidente ilustra la importancia crítica de unos requisitos precisos, completos y verificables.

Tradicionalmente, los requisitos se han obtenido mediante entrevistas, análisis de documentación, talleres y otras técnicas manuales que dependen en gran medida de la experiencia y habilidad del analista. Estas técnicas, aunque ampliamente utilizadas, están sujetas a errores humanos, omisiones y ambigüedades inherentes al lenguaje natural [4, 5].

Con la aparición de técnicas avanzadas de NLP y el auge de los modelos de lenguaje de gran tamaño (Large Language Models, LLMs) [6], se abren nuevas oportunidades para automatizar tareas que antes requerían intervención humana intensiva, como la generación de requisitos. En este marco, el presente trabajo se plantea como una oportunidad para explorar el uso de modelos de lenguaje en la automatización del proceso de generación de requisitos, con el objetivo de aumentar la eficiencia, reducir los errores inherentes a los enfoques tradicionales y mejorar la calidad de los requisitos generados, lo que debería optimizar el desarrollo del software desde sus etapas iniciales.

1.2. Planteamiento del problema

A pesar del avance en metodologías y herramientas para el desarrollo de software, la elicitación y especificación de requisitos continúa dependiendo en gran medida de la intervención humana, por lo que sigue siendo propenso a errores humanos, ambigüedades y malentendidos que, si no se detectan a tiempo, pueden propagarse al resto del ciclo de vida del software, provocando fallos funcionales, sobrecostes o retrasos.

Además, aunque existen modelos de lenguaje entrenados sobre grandes corpus de texto, su rendimiento en tareas especializadas como la generación de requisitos formales o semiformales aún presenta limitaciones. Estos modelos pueden generar resultados gramaticalmente correctos pero semánticamente inadecuados, o bien no adaptarse con suficiente precisión a los formatos y estilos requeridos en documentos de ingeniería de requisitos debido a que no están específicamente optimizados para este dominio.

Por tanto, el problema que se aborda en este trabajo es la mejora del proceso de generación de requisitos mediante técnicas automáticas que reduzcan la intervención humana y la necesidad de evaluar y ajustar los modelos de NLP existentes para que se adapten eficazmente a este dominio tan específico.

1.3. Objetivos

El principal objetivo de este Trabajo Fin de Grado (TFG) es investigar y aplicar técnicas de IA, concretamente del ámbito del NLP, con el fin de optimizar la generación de requisitos en el desarrollo de software. La idea central consiste en explorar el potencial de los LLMs para asistir e incluso automatizar el proceso de redacción de requisitos, tradicionalmente realizado de forma manual por analistas humanos.

Con este marco general, se plantean los siguientes objetivos específicos:

- Explorar distintos modelos de lenguaje preentrenados (como GPT [7], LLaMA [8], DeepSeek [9], entre otros) para evaluar su capacidad de generar requisitos de software a partir de descripciones en lenguaje natural o escenarios funcionales.
- Implementar una metodología de evaluación comparativa, en la que se midan diversos aspectos de la calidad de los requisitos generados, tales como su coherencia, completitud, corrección lingüística y utilidad práctica para desarrolladores y analistas.
- Aplicar técnicas de ajuste fino (fine-tuning) y adaptación de modelos, como LoRA (Low-Rank Adaptation) [10], con el objetivo de especializar los modelos en el dominio de la ingeniería de requisitos, mejorando así su rendimiento específico en esta tarea.
- Realizar experimentos con diferentes configuraciones, incluyendo ajustes de hiperparámetros y distintas variantes de entrada, para determinar qué combinaciones ofrecen los mejores resultados según los criterios de evaluación definidos.

- Reflexionar sobre la utilidad real de los modelos de lenguaje en este contexto y proponer recomendaciones para su uso práctico, así como posibles mejoras o líneas futuras de trabajo.

1.4. Fases del proyecto

El desarrollo del trabajo se ha dividido en las siguientes fases, cada una con una duración y objetivos definidos:

- Fase 1 – Formación en modelos de lenguaje y NLP (1 mes)
- Fase 2 – Preparación del dataset y experimentación inicial (2 semanas)
- Fase 3 – Selección del modelo pre-entrenado y configuración inicial (1 semana)
- Fase 4 – Entrenamiento y ajuste del modelo (1,5 meses)
- Fase 5 – Evaluación y análisis de resultados (1 mes)
- Fase 6 – Redacción del informe final y preparación de su defensa (2 semanas)

1.5. Estructura de la memoria

La presente memoria se estructura en cinco capítulos, que reflejan de forma coherente el desarrollo del trabajo. Cada uno de ellos aborda un aspecto fundamental del TFG, desde la motivación inicial hasta la exposición de los resultados obtenidos y las conclusiones finales.

A continuación, se describe brevemente el contenido de cada uno:

- **Capítulo 1 – Introducción:** Presenta el contexto general del trabajo, la motivación que lo impulsa, el problema abordado, los objetivos perseguidos, la planificación seguida y la estructura general del documento.
- **Capítulo 2 – Estado del arte y fundamentos teóricos:** Se analizan los conceptos clave relacionados con la ingeniería de requisitos y el NLP, incluyendo los modelos de lenguaje actuales, técnicas de fine-tuning y enfoques previos aplicados a la generación de requisitos. Este capítulo proporciona el marco teórico necesario para comprender las decisiones metodológicas adoptadas.
- **Capítulo 3 – Metodología e implementación:** Describe el enfoque seguido para llevar a cabo el trabajo. Se detallan las características del *dataset* empleado, las fuentes de datos y los modelos utilizados, el proceso de entrenamiento, la generación de resultados (inferencia) y los criterios de validación aplicados.
- **Capítulo 4 – Resultados:** Expone los resultados obtenidos en cada uno de los diferentes escenarios de prueba utilizados y una comparativa entre distintas configuraciones de modelos y técnicas aplicadas. Este capítulo constituye el núcleo empírico del trabajo.

- **Capítulo 5 – Discusión y conclusiones:** Se interpretan los resultados obtenidos y su impacto en la mejora de la calidad de los requisitos, se analizan las principales limitaciones del estudio, se recogen las principales conclusiones alcanzadas, el grado de cumplimiento de los objetivos establecidos inicialmente y se proponen posibles líneas de trabajo futuro que podrían ampliar o profundizar los resultados obtenidos.

- **Bibliografía:** Incluye todas las referencias bibliográficas consultadas y citadas a lo largo del trabajo, que han servido de base para el desarrollo del mismo.

Con esta estructura se busca ofrecer una visión completa y ordenada del trabajo realizado, facilitando su comprensión y evaluación.

Capítulo 2

Estado del arte y fundamentos teóricos

2.1. La ingeniería de requisitos en el desarrollo de software

La ingeniería de requisitos (IR) [2] es una fase fundamental dentro del ciclo de vida del software. Su objetivo principal es identificar, documentar y gestionar las necesidades y restricciones que debe satisfacer un sistema software, es decir, sus requisitos. Dichos requisitos se pueden clasificar en diferentes tipos según su naturaleza y propósito:

- **Requisitos funcionales:** Describen comportamientos específicos del sistema.
- **Requisitos no funcionales:** Especifican atributos de calidad del sistema, como rendimiento, usabilidad, fiabilidad o seguridad.
- **Requisitos de usuario:** Engloban los requisitos funcionales y no funcionales pero expresados desde el punto de vista del usuario final y redactados en lenguaje natural.
- **Requisitos del sistema:** Requisitos de usuario redactados de manera técnica y detallada, que emplean los desarrolladores como base para el diseño e implementación del sistema [11].

No obstante, en este trabajo, se distingue únicamente entre requisitos funcionales y no funcionales, ya que esta clasificación resulta suficiente y práctica para abordar el problema de generación automática de requisitos mediante modelos de lenguaje.

2.2. Procesamiento del lenguaje natural y su aplicación a la ingeniería de requisitos

El procesamiento del lenguaje natural (NLP) es una rama de la inteligencia artificial que se ocupa de la interacción entre los ordenadores y el lenguaje humano [1]. Su objetivo es que las máquinas comprendan, interpreten, generen y respondan al lenguaje natural de forma similar a como lo haría un ser humano y, para ello, combina técnicas de lingüística computacional, aprendizaje automático y modelos estadísticos. El NLP ha permitido mejorar en el rendimiento de tareas como la comprensión lectora, el análisis de sentimientos o la generación de texto, marcando un punto de inflexión en el desarrollo de aplicaciones basadas en lenguaje [12].

El funcionamiento del NLP suele incluir las siguientes fases [12, 13]:

- **Preprocesamiento del texto:** Limpiar y normalizar el texto antes de procesarlo.

- **Tokenización:** Dividir el texto en unidades más pequeñas como palabras, frases o caracteres.
- **Eliminación de *stopwords*:** Quitar palabras vacías como "el", "de", "y", etc.
- **Lematización/*stemming*:** Reducir las palabras a su forma base o raíz.
- **Normalización:** Pasar a minúsculas, eliminar puntuación, corrección ortográfica, etc.
- **Análisis sintáctico:** Analizar la estructura gramatical de las oraciones según las reglas del idioma.
- **Análisis semántico:** Representar el significado del texto a través de vectores o *embeddings*.
- **Análisis pragmático:** Interpretar el significado del texto en función del contexto.
- **Tareas específicas:** Clasificación de texto, extracción de entidades, resumen automático, traducción o generación de lenguaje natural.

En el contexto del desarrollo de software, el NLP tiene múltiples aplicaciones en la IR, entre las que se identifican cuatro categorías principales [14]:

- Clasificación de requisitos (por ejemplo, funcionales vs. no funcionales).
- Extracción automática de requisitos desde, por ejemplo, documentos de texto o especificaciones.
- Transformación y formalización de los requisitos a modelos estructurados o semiformales.
- Análisis de calidad, detectando ambigüedades, redundancias o incoherencias.

En este ámbito, se han desarrollado herramientas y prototipos de investigación, como por ejemplo, un sistema que emplea técnicas de NLP para identificar automáticamente requisitos ambiguos en especificaciones escritas en lenguaje natural. El enfoque se basa en el análisis de patrones lingüísticos y el uso de modelos preentrenados para detectar expresiones vagas, modales indefinidos o construcciones condicionales poco claras, lo que permite a los analistas mejorar la calidad de los requisitos desde etapas tempranas [5].

A pesar de los avances, aún existen desafíos importantes como la escasez de *datasets* públicos y etiquetados, la falta de estandarización en los criterios de evaluación, la dependencia del dominio y la necesidad de intervención humana para validar los resultados. Por ello, este trabajo se enmarca en una línea de investigación activa que busca aprovechar modelos LLMs ajustados para generar requisitos más útiles y fiables [14].

2.3. Modelos de lenguaje de gran escala (LLMs)

Los modelos de lenguaje de gran escala (Large Language Models, LLMs) son modelos de aprendizaje automático típicamente basados en arquitecturas de tipo *Transformer* [18] (véase Sección 2.4) y entrenados con grandes cantidades de texto para aprender las estructuras y patrones del lenguaje natural. Su principal objetivo es modelar la probabilidad de ocurrencia de secuencias lingüísticas, lo que les permite predecir, completar o generar texto coherente. Además, estos modelos se caracterizan por contar con miles de millones de parámetros, lo que les otorga una gran capacidad de representación semántica. Ejemplos populares de LLMs incluyen GPT-4 [7], PaLM [15], Claude [16], LLaMA [8] o Mistral [17], entre otros.

A nivel funcional, los LLMs se entrenan inicialmente mediante un proceso de preentrenamiento no supervisado, donde el objetivo es predecir tokens faltantes o siguientes en un texto. Posteriormente, pueden ajustarse mediante técnicas como *fine-tuning*, *instruction tuning* o aprendizaje por refuerzo con retroalimentación humana (RLHF) para especializarlos en tareas concretas. Todo ello permite resolver problemas como la traducción automática, generación de resúmenes, clasificación de texto, extracción de información o asistencia conversacional, sin necesidad de diseñar sistemas específicos para cada tarea [18, 19].

Los LLMs utilizan mecanismos de atención, que permiten capturar relaciones contextuales a largo plazo entre palabras o frases dentro de un texto y mejorar la comprensión semántica y la generación coherente de respuestas.

En el contexto de la ingeniería de requisitos, los LLMs abren nuevas posibilidades para automatizar tareas como la generación de requisitos a partir de descripciones textuales, la clasificación semántica, la detección de ambigüedades o la transformación de texto natural a lenguajes formales [19, 20].

2.4. Arquitectura *Transformer*

Mientras que modelos como las redes recurrentes (RNN) o las redes LSTM (Long Short-Term Memory) procesaban texto de forma secuencial, la arquitectura Transformer emplea mecanismos de atención que permiten procesar toda la secuencia de texto en paralelo (cada palabra del *input* puede acceder directamente a cualquier otra del contexto, independientemente de su posición) [21]. El componente clave de esta arquitectura es el mecanismo de *self-attention*, que permite al modelo asignar diferentes pesos a cada palabra del contexto a la hora de procesar un *token* determinado calculando cómo de relevante es cada palabra con respecto a las demás.

Además, esta arquitectura se compone de dos bloques: el codificador (*encoder*), que recibe como entrada una secuencia de *tokens* y genera su representación contextualizada, y el decodificador (*decoder*), que genera la secuencia de salida de manera autoregresiva, es decir, token a token, condicionada a la entrada codificada por el encoder. Cada bloque está formado por una pila de capas idénticas que contienen [21]:

- **Mecanismo de atención multi-cabeza (*Multi-Head Attention*)**: Permite que cada *token* preste atención a otros *tokens* de la misma secuencia, capturando relaciones internas sin importar la distancia entre palabras.

- **Capa *feed-forward***: Una red neuronal conectada que se aplica de forma independiente a cada *token* para transformar su representación.

- **Normalización y conexiones residuales**: Ayudan a estabilizar y mejorar el entrenamiento.

- **Codificación posicional**: Dado que la arquitectura no tiene una noción inherente del orden de las palabras, se introducen vectores de posición (*positional encodings*) para incorporar esta información.

Actualmente, muchos modelos de lenguaje utilizan variantes de esta arquitectura. En función del tipo de tarea, se emplean diferentes configuraciones [21]:

- **Modelos encoder-decoder**: Como T5 [22] o BART [23], se usan principalmente en tareas de traducción, resumen o respuesta automática, donde se transforma una secuencia de entrada en otra completamente distinta.

- **Modelos sólo con bloque decoder**: Como GPT-4 [7], LLaMA [8] o Mistral [17], están diseñados para tareas de generación de texto, donde solo se requiere producir una secuencia de salida a partir de un *prompt*, sin una entrada que deba ser codificada por separado.

- **Modelos solo encoder**: Como BERT [24], se utilizan en tareas de clasificación, extracción de entidades o análisis semántico, donde es necesario comprender el significado del texto pero no generar texto nuevo.

Los modelos de lenguaje de gran escala utilizados en este trabajo se basan exclusivamente en el bloque *decoder*, ya que su propósito principal es la generación de texto.

2.5. Técnicas de ajuste fino

El ajuste fino (*fine-tuning*) es una técnica para adaptar modelos de lenguaje preentrenados (modelos base entrenados con grandes cantidades de datos generales) a tareas específicas mediante una segunda fase de entrenamiento sobre un conjunto de datos más pequeño y representativo del problema [25]. Este proceso aprovecha el conocimiento general que el modelo ya ha aprendido, evitando la necesidad de entrenar desde cero y reduciendo significativamente el coste computacional y de datos. A continuación se describen las principales técnicas:

- **Fine-tuning supervisado clásico:** Consiste en reentrenar todos los parámetros del modelo base con los nuevos datos, lo que implica tener una gran capacidad de GPU/TPU [26].

- **Instruction tuning:** Se basa en entrenar al modelo con pares de entrada-salida en forma de *prompt* + respuesta para enseñar al modelo a seguir instrucciones expresadas en lenguaje natural [27].

- **Aprendizaje por refuerzo con retroalimentación humana (RLHF):** Tras un primer ajuste supervisado, se recopila *feedback* humano sobre salidas generadas y se entrena un *reward* model. A continuación, se aplica aprendizaje por refuerzo para maximizar dicha recompensa, alineando las respuestas con criterios de calidad definidos por evaluadores humanos [28].

- **Adaptadores ligeros (LoRA):** Inyecta matrices de bajo rango entrenables en determinadas capas del modelo, manteniendo el resto de pesos congelados y permitiendo así adaptar el comportamiento del modelo con un número muy reducido de parámetros entrenables [10].

En este proyecto, se emplean adaptadores ligeros (LoRA) a través del framework PEFT de Hugging Face [29] y se entrenan los modelos mediante *instruction tuning*.

2.6. Métodos de evaluación de salidas generadas

Para evaluar la calidad de los textos generados por modelos de lenguaje se suelen combinar métodos automáticos y evaluaciones humanas, de manera que se cubren tanto aspectos cuantitativos como cualitativos. En este trabajo se ha optado únicamente por una evaluación humana de las salidas generadas mediante un formulario, dado que los criterios de calidad considerados como claridad, gramática, detalle o ambigüedad no pueden ser capturados adecuadamente mediante métricas automáticas convencionales. Para valorar la fiabilidad del juicio de los anotadores, se han empleado dos métricas estadísticas:

- **Alfa de Krippendorff (α):** mide el grado de acuerdo entre varios evaluadores, adaptándose a diferentes tipos de escalas (nominal, ordinal, de intervalo) y tolerando datos faltantes. Su valor varía entre 1 (acuerdo perfecto), 0 (acuerdo por azar) y valores negativos (desacuerdo sistemático).

- **Coeficiente de correlación de Spearman (ϱ):** analiza la similitud en el ordenamiento de modelos por parte de pares de anotadores, sin requerir coincidencia exacta en los valores absolutos. Su valor oscila entre –1 (orden completamente inverso) y 1 (orden idéntico).

Para una descripción más detallada de estas métricas y sus fórmulas, así como otros métodos de evaluación existentes, véase el Anexo B.

Capítulo 3

Metodología e implementación

Este capítulo describe en detalle la metodología seguida para llevar a cabo el proyecto, así como los detalles de implementación técnica. Se abordan los aspectos clave relacionados con las fuentes de datos utilizadas, el diseño de los *prompts* empleados, los modelos de lenguaje seleccionados, el proceso de ajuste fino de los modelos y las estrategias de inferencia y validación aplicadas. Esta sección constituye el núcleo experimental del trabajo y sirve de base para el análisis de resultados que se presenta en el siguiente capítulo.

3.1. Diseño de *prompts*

La forma en la que se formula la entrada (*prompt*) de un modelo de lenguaje tiene un impacto significativo en la calidad, relevancia y precisión del resultado obtenido ya que los modelos generan texto a partir de instrucciones en lenguaje natural [37]. Por esta razón, se ha llevado a cabo un análisis y diseño sistemático de diferentes tipos de *prompts*, con el objetivo de conseguir la mejor calidad en los requisitos generados y reducir al mínimo posible las ambigüedades o formulaciones poco claras.

Para abordar este diseño, se exploraron distintas estrategias de *prompting* incluyendo, entre otros, enfoques directos ("Extrae los requisitos funcionales y no funcionales del siguiente sistema: ...") o enfoques en los que se presentaban previamente varios requisitos de ejemplo (*few-shot prompting*). También se experimentó con variaciones en la forma de la redacción, el nivel de contexto proporcionado y el uso de instrucciones explícitas para fomentar un lenguaje técnico, específico y orientado a requisitos de software reales.

Además, se evaluó la influencia del idioma (español frente a inglés) en la calidad de los requisitos generados, comparando *prompts* equivalentes redactados en español y en inglés. Esto se debe a que algunos modelos han sido entrenados mayoritariamente con datos en inglés, lo que puede influir en su rendimiento en otros idiomas. El conjunto completo de *prompts* evaluados durante el estudio se encuentran recogidos en el Anexo A.

3.2. Fuentes de datos

Durante el desarrollo del TFG se han empleado tres fuentes de datos distintas, con el objetivo de disponer de descripciones variadas de sistemas software en lenguaje natural que permitieran evaluar la capacidad de los modelos para generar requisitos funcionales y no funcionales de calidad. Estas fuentes cubren distintos niveles de formalidad, idioma y estructura, lo que ha permitido una evaluación más robusta de los resultados generados.

1. **Enunciados de problemas de examen (Universidad de Zaragoza)**

   Se utilizaron enunciados redactados en castellano, procedentes de exámenes y prácticas de la asignatura Ingeniería de Requisitos del Grado en Ingeniería Informática de la Universidad de Zaragoza. Estos enunciados presentan escenarios detallados y estructurados, a partir de los cuales los estudiantes deben identificar y redactar los requisitos funcionales y no funcionales.

   Durante la fase inicial del trabajo, estos enunciados se emplearon para realizar pruebas preliminares de inferencia. Además, se generaron versiones resumidas y más informales de algunos de ellos, con el objetivo de evaluar cómo los modelos se comportaban frente a entradas más cercanas al lenguaje natural no técnico.

2. ***DaReC* (Dataset for Requirements Classification)**

   El dataset DaReC, disponible públicamente a través de GitHub [38], contiene descripciones de sistemas reales recopiladas de diversas fuentes, redactadas en inglés y cada una acompañada de una lista de requisitos funcionales y no funcionales.

   En este trabajo, DaReC fue utilizado para entrenar los modelos mediante técnicas de ajuste fino (*fine-tuning*), así como para evaluarlos posteriormente. Esto permitió evaluar la capacidad del modelo para generalizar tras haber sido expuesto a este dominio concreto.

3. **Descripciones de aplicaciones (Google Play Store)**

   Se recopilaron descripciones públicas de aplicaciones en la Google Play Store, escritas en castellano, como fuente adicional de descripciones informales y reales de sistemas software.

   Estas descripciones se utilizaron para realizar pruebas preliminares de inferencia y observar el comportamiento de los modelos ante entradas del mundo real y orientadas al usuario final, con una redacción más abierta y con menor nivel de formalización.

3.3. Conjunto de datos (*dataset*)

Para la construcción del conjunto de datos utilizado en la fase de *fine-tuning* de los modelos de lenguaje, se valoraron las distintas fuentes de datos descritas en la sección anterior. El objetivo era contar con descripciones de sistemas en lenguaje natural acompañadas de requisitos correctamente formulados para que los modelos supieran el formato y estilo requerido. Finalmente, se optó por utilizar únicamente el *dataset* DaReC para la construcción del conjunto de datos ya que era la única fuente que ofrecía descripciones acompañadas de requisitos asociados.

No obstante, el *dataset* original de DaReC fue adaptado y transformado para ajustarse a las necesidades del proyecto. De los 50 sistemas que posee el dataset original de DaReC, se emplearon un total de 49 para la creación del nuevo conjunto de datos, reservando uno

exclusivamente para la validación posterior a la fase de *fine-tuning*. Esta elección se realizó manualmente, seleccionando un sistema representativo pero no demasiado específico, de modo que sirviera para evaluar la capacidad de generalización del modelo sin estar influido por ejemplos similares vistos durante el entrenamiento. El sistema no incluido en el dataset es el *2001 - space fractions*, cuya descripción es la siguiente:

> *The Space Fractions project is a learning tool created to help improve fraction-solving skills for sixth-grade students. The product will be a web-based, interactive game. At the end of the game, students will be given feedback based on their game scores. We are also providing an umbrella for the past games created. The umbrella will be a web-based menu system allowing the user to choose between the games.*

El *dataset* construido se almacenó en formato CSV, con tres columnas:

- **system:** Nombre del fichero o sistema original.
- **prompt:** Plantilla de entrada en inglés utilizada para guiar al modelo en la generación de requisitos.
- **requirements:** Requisitos esperados, estructurados de forma estandarizada.

Tras probar y analizar los resultados de las distintas variantes de *prompts* (véase Sección 3.1), se seleccionó aquel que ofrecía mejores resultados tanto en completitud como en claridad de las salidas generadas. Este *prompt* incluye instrucciones específicas para minimizar ambigüedades, inducir ejemplos concretos y forzar una organización clara en requisitos funcionales y no funcionales. A continuación se muestra el formato del *prompt* utilizado:

> *Given the following system description, tell me in full detail and without ambiguities or vagueness everything that someone implementing the system should know. If necessary, make assumptions as you see fit, providing concrete examples for any potential doubts that may arise—for instance, when there are references to time without specific durations, when conditions are mentioned but not explicitly defined, or when dealing with device connections, etc. Present the information in the form of functional and non-functional requirements.*
>
> *<descripción del sistema>*

En cuanto al formato de los requisitos esperados, se optó por una estructura basada en las recomendaciones del estándar IEEE Std 830-1998 [39]. En particular, se siguió la fórmula habitual “The system shall…” como base para expresar los requisitos de manera clara y formal, y se empleó una codificación estructurada (FR1, FR2, NFR1…) para asegurar que cada requisito fuese identificable de forma única.

> *Functional Requirements (FR)*
> *FR1: The system shall …*
>
> *Non-Functional Requirements (NFR)*
> *NFR1: …*

Para garantizar la coherencia, se adaptaron algunos requisitos originales de DaReC al nuevo formato, aplicando ajustes mínimos cuando fue necesario. El *dataset* final se encuentra disponible en el repositorio del TFG ReqGen-AI-Opt[1].

3.4. Modelos de lenguaje utilizados

En este trabajo se utilizaron diversos LLMs, todos ellos basados en la arquitectura Transformer y afinados mediante técnicas de *instruction tuning* [27]. Esto les permite interpretar instrucciones expresadas en lenguaje natural y generar texto de forma coherente, aspecto fundamental para la tarea de generación automática de requisitos a partir de descripciones informales de sistemas. Concretamente se emplearon:

- **Modelos de propósito general:** Son accesibles a través de plataformas web y se emplearon únicamente para inferencia.
    - GPT-4o Mini [41]
    - DeepSeek-R1 [44]
- **Modelos *open-source* (de código abierto):** Están disponibles desde la plataforma Hugging Face, la cual permite emplear sus modelos gratuitamente para investigación y, por lo tanto, permitió emplearlos tanto para inferencia como para su ajuste fino.
    - meta-llama/Meta-Llama-3.2-1B-Instruct [45]
    - meta-llama/Meta-Llama-3.2-3B-Instruct [46]
    - meta-llama/Meta-Llama-3.1-8B-Instruct [47]
    - mistralai/Mistral-7B-Instruct-v0.3 [48]

Para más información y detalles técnicos de los modelos véase el Anexo C.

3.5. Entrenamiento (*fine-tuning*)

El proceso de *fine-tuning* se llevó a cabo exclusivamente sobre los modelos de código abierto disponibles en Hugging Face: los modelos de la familia Meta-Llama 3 Instruct (1B, 3B y 8B) y Mistral 7B Instruct. Debido a sus tamaños (que oscilan entre 1.000 y 8.000 millones de parámetros) y las elevadas necesidades de memoria que implican, no fue viable realizar el entrenamiento en plataformas como Google Colab. En su lugar, se recurrió al servidor Berlin [49], un servidor de prácticas de la Universidad de Zaragoza que cuenta con un procesador AMD EPYC 7313P, sistema operativo CentOS GNU/Linux, y una GPU NVIDIA A10 con 24 Gb de VRAM, proporcionando así una infraestructura adecuada para este tipo de tareas intensivas en cómputo.

[1] https://github.com/lilainaranjo/ReqGen-AI-Opt

El *script* de entrenamiento fue desarrollado en Python utilizando la biblioteca Transformers de Hugging Face y su clase Trainer, que proporcionan una interfaz flexible para cargar, configurar y entrenar modelos de lenguaje preentrenados. Para el manejo de datos se empleó el módulo datasets, que facilita el procesamiento y manipulación del conjunto de datos, y para la aplicación de técnicas de ajuste eficiente se utilizó la librería peft (Parameter-Efficient Fine-Tuning) [29]. El entrenamiento se realizó con un *batch size* de 1 y una *learning rate* de 1e-4, parámetros que se mantuvieron constantes debido a las limitaciones de memoria. Se exploraron otras combinaciones de estos hiperparámetros, pero la memoria limitada y la capacidad computacional restringida impidieron experimentar con *batch sizes* mayores. El número de épocas osciló entre 1 y 32 para observar su efecto en la calidad de los resultados.

Durante el proceso de entrenamiento, el modelo recibe cada bloque de entrada y calcula mediante *forward pass* (procesar la secuencia de entrada *tokenizada* a través de sus capas para generar una predicción de los tokens de salida) la probabilidad de los tokens de salida esperados. A continuación, se evalúa la pérdida entre la salida real del modelo y el requisito "ground-truth" (los requisitos esperados en el *dataset*). Esta pérdida se retropropaga para obtener los gradientes de cada parámetro y, finalmente, un optimizador actualiza los pesos del modelo y los parámetros de los adaptadores LoRA, si los hay, en la dirección que minimiza la función de pérdida. Repetir este ciclo a lo largo de varias épocas hace que el modelo aprenda patrones específicos del dominio de requisitos, ajustando sus representaciones internas para generar texto más preciso y coherente con los ejemplos de entrenamiento.

Dado el reducido tamaño del *dataset* (49 descripciones de sistemas), no se realizó una división convencional en subconjuntos de entrenamiento, validación y test. Reservar una parte para validación habría reducido aún más la muestra disponible, y el objetivo del trabajo no era obtener métricas cuantitativas estándar para evaluar la calidad de los requisitos generados (véase Sección 3.7). En su lugar, se optó por reservar un sistema no visto durante el entrenamiento para comprobar la capacidad del modelo afinado de generalizar a nuevos escenarios. Este sistema es el *2001 - space fractions*, mencionado anteriormente en la Sección 3.3.

Una parte fundamental del entrenamiento fue el preprocesamiento del *dataset* para adaptarlo al formato conversacional esperado por los modelos Instruct por lo que se utilizó el método apply_chat_template del *tokenizer* de Hugging Face para generar secuencias de entrada con los roles "system", "user" y "assistant". En un primer intento, se intentó pasar toda la conversación como una única secuencia, pero debido a errores por falta de memoria, se modificó la función tokenize_with_chat_template para dividir las secuencias generadas en bloques más pequeños de longitud fija (context_length = 2048). Esta estrategia de división evitó errores de memoria a costa de cierta pérdida de contexto y coherencia en los ejemplos. El código completo utilizado para el entrenamiento se encuentra disponible en el repositorio del TFG ReqGen-AI-Opt[1].

Se probaron distintas longitudes de corte para evaluar si era posible evitar el uso de técnicas de optimización de memoria, pero el entrenamiento completo de los modelos más

grandes requería más memoria de la disponible. Por este motivo, en los modelos de mayor tamaño (3B, 7B y 8B) fue necesario recurrir a dos técnicas complementarias:

- **Cuantización en 4 bits:** mediante la clase BitsAndBytesConfig, que permite reducir el tamaño de los tensores del modelo utilizando representaciones más compactas, con un impacto limitado en la precisión.
- **Adaptadores LoRA (Low-Rank Adaptation):** implementados mediante la librería peft, los cuales permiten insertar capas ligeras entrenables dentro del modelo base, evitando la necesidad de ajustar todos sus parámetros.

Estas técnicas están descritas con más detalle en la Sección 2.5. Su uso resultó imprescindible para lograr completar el proceso de fine-tuning en los modelos más complejos, preservando la viabilidad del experimento sin comprometer gravemente la calidad de las salidas a pesar de las posibles pérdidas de fidelidad asociadas al uso de cuantización y fragmentación del contexto. La Tabla 3.5.1 resume qué técnicas fueron necesarias según el modelo empleado:

Modelo	Cuantización 4-bit	Adaptadores LoRA
Meta-Llama 3 1B	No	Sí
Meta-Llama 3 3B	Sí	Sí
Meta-Llama 3 8B	Sí	Sí
Mistral 7B Instruct	Sí	Sí

Tabla 3.5.1 Técnicas de optimización empleadas por cada modelo

Como se observa, sólo el modelo Meta-Llama 1B pudo ser ajustado directamente sin necesidad de cuantización. Para todos los demás modelos, el ajuste fino habría sido inviable sin aplicar ambas técnicas de optimización de memoria.

3.6. Inferencia

La fase de inferencia tuvo como objetivo evaluar la capacidad de los modelos, tanto preentrenados como ajustados mediante *fine-tuning*, para generar requisitos de software a partir de descripciones de sistemas en lenguaje natural. Esta evaluación se llevó a cabo en distintos momentos del proyecto y sobre diversos tipos de entrada, con el fin de analizar el comportamiento de los modelos en escenarios variados y realistas.

Antes de entrenar los modelos *open-source*, se probaron ChatGPT y DeepSeek Chat para obtener una primera valoración de las formulaciones de *prompt* diseñadas (ver Sección 3.1) y analizar si los modelos generaban mejores salidas respecto a unas categorías/temas u otras. Esto permitió contrastar la claridad y completitud de los requisitos según cada variante de *prompt*, identificar qué estilo de redacción inducía salidas más organizadas y tecnológicas, ajustar el *prompt* final definitivo que, posteriormente, se utilizaría también

para los modelos entrenados y elegir las fuentes de datos que se emplearían en el *dataset* final.

Las fuentes de entrada empleadas en estas primeras pruebas fueron las descritas en la Sección 3.2. En concreto:

- Descripciones del dataset DaReC, en inglés. Se emplearon un total de 13 descripciones de las 50 totales, seleccionadas para cubrir distintos dominios y niveles de complejidad.
- Enunciados técnicos de prácticas y exámenes de la asignatura Ingeniería de Requisitos de la Universidad de Zaragoza. Se emplearon un total de 14 descripciones (7 originales y 7 adaptadas con tono más informal).
- Descripciones informales de aplicaciones reales de la Google Play Store. Se emplearon un total de 7 descripciones con diversidad temática.

Cuando se utilizó como entrada una descripción del *dataset* DaReC, fue posible comparar las salidas generadas por los modelos con los requisitos originales de referencia. En estos casos, se analizaron las salidas en base a los siguientes criterios:

- **Cobertura de requisitos:** Capacidad de extraer todos los requisitos funcionales y no funcionales presentes en la especificación original.
- **Claridad estructural:** Diferenciación clara entre requisitos funcionales (RF) y no funcionales (RNF), uso de identificadores únicos, organización formal.
- **Nivel de detalle:** Precisión en la redacción, especificidad o ejemplos adicionales que no aparecían en el enunciado.

La Tabla 3.6.1 refleja cómo se ha valorado cada uno de los criterios.

	Cobertura de requisitos	Claridad estructural	Nivel de detalle
Muy bajo	< 15% de los requisitos originales cubiertos	Lista sin separación RF/RNF ni numeración	Enunciados genéricos
Bajo	15–30% de los requisitos originales cubiertos	Separación en RF/RNF, pero sin identificadores únicos	Redacción sin ejemplos
Medio	30–60% de los requisitos originales cubiertos	Numeración única parcial con estructura inconsistente	Algún ejemplo o suposición mínima
Alto	60–80% de los requisitos originales cubiertos	Numeración única consistente y agrupación clara en RF/RNF	Varios ejemplos y matices de implementación
Muy alto	> 80% de los requisitos originales cubiertos	Inclusión de subtítulos o jerarquías avanzadas	Ejemplos de pseudocódigo, casos de uso

Tabla 3.6.1 Criterios de evaluación y umbrales cualitativos

Con el resto de descripciones de sistemas, donde no se disponía de un conjunto de requisitos de referencia con los que comparar, la evaluación fue puramente cualitativa, basada en inspección manual.

Con el *prompt* final seleccionado y el *dataset* construido, se generaron requisitos usando los modelos ajustados de Hugging Face (Meta-Llama Instruct 1B/3B/8B y Mistral 7B) mediante un script desarrollado en Python, utilizando las bibliotecas Transformers, PEFT y BitsAndBytes de Hugging Face, las cuales permitieron cargar los modelos con adaptadores LoRA y aplicar cuantización para optimizar el uso de memoria durante la generación. Para controlar el número de tokens generados, se estableció un límite de salida de 3000 tokens. No obstante, algunos modelos generaban salidas notablemente más cortas. A medida que se aumentaban las épocas de entrenamiento (especialmente a partir de 8), se detectaron problemas de repetición en las salidas, tanto a nivel de frase como en bloques enteros de requisitos. Para mitigar este problema, se utilizó el parámetro *repetition_penalty* durante la inferencia, experimentando con distintos valores que oscilaron entre 1.1 y 1.5. El código completo empleado para esta fase está disponible en el repositorio del TFG ReqGen-AI-Opt[1].

3.7. Validación

La validación del sistema se llevó a cabo mediante una evaluación manual (véase Sección 2.6). El objetivo de esta fase era valorar la calidad global de los requisitos generados por distintos modelos, así como detectar fortalezas y debilidades en aspectos clave como precisión, claridad, coherencia y utilidad práctica. Para ello, se diseñó un formulario de evaluación en Google Forms, donde se recopilaron valoraciones de anotadores humanos sobre un conjunto de salidas generadas por distintos modelos para una misma descripción de sistema. Concretamente se evaluó una de las salidas generada por:

- ChatGPT
- DeepSeek
- meta-llama/Meta-Llama-3.1-8B-Instruct sin ajustar
- meta-llama/Meta-Llama-3.2-1B-Instruct sin ajustar
- DaReC
- mistralai/Mistral-7B-Instruct-v0.3 sin ajustar
- mistralai/Mistral-7B-Instruct-v0.3 ajustado (4 épocas, repetition_penalty=1.3)

La elección de estos modelos respondió al objetivo de cubrir una muestra representativa de todos los modelos estudiados, así como una instancia del *dataset* DaReC para contar con una referencia humana original. Pese a que se ajustaron varios modelos de Hugging Face con diversas configuraciones, solo se seleccionó uno de esos modelos ajustados para su evaluación, eligiendo aquel cuya salida ofrecía mayor completitud o coherencia. Esta

decisión respondió a limitaciones observadas durante la inferencia, como fallos de generación, repeticiones excesivas o salidas incompletas en muchas ejecuciones.

Se seleccionaron ejemplos con el mismo sistema como punto de partida (*2001 - space fractions*), de forma que se pudiera realizar una comparación directa entre sus salidas y las valoraciones se realizaron a través de siete preguntas, cada una puntuada en una escala del 1 (muy deficiente) al 10 (excelente):

1. ¿Qué tan correctamente han sido etiquetados los requisitos como funcionales o no funcionales?
2. ¿Qué tan precisos y libres de ambigüedad son los términos empleados en los requisitos generados?
3. ¿Qué tan correctos están los requisitos en cuanto a gramática, estilo y ortografía?
4. ¿Qué tan fáciles de entender e interpretar resultan los requisitos sin necesidad de aclaraciones adicionales?
5. ¿Qué tan consistente es el conjunto de requisitos entre sí (sin contradicciones, sin duplicados o copias)?
6. ¿Qué tan bien equilibrado está el nivel de detalle (suficiente para entender, sin exceso de “micro-requisitos”)?
7. Valoración general de la salida

El proceso contó con la participación de cuatro anotadores con diferentes perfiles y niveles de experiencia: una estudiante de grado en Ingeniería Informática, dos profesores de la rama de ingeniería del software y un profesor especialista en la asignatura de Ingeniería de Requisitos. Esta diversidad de perfiles permitió recoger opiniones desde distintos niveles de experiencia, combinando la perspectiva de usuarios finales con la de expertos académicos en el dominio.

Con el fin de cuantificar el nivel de acuerdo entre las evaluaciones emitidas por los distintos anotadores, se calculó el alfa de Krippendorff en su variante ordinal, adecuada para escalas como la utilizada en este estudio (1 a 10). Adicionalmente, para explorar la relación entre las valoraciones otorgadas por distintos anotadores y evaluar si siguen patrones similares en la ordenación de las salidas, se calculó también el coeficiente de correlación de Spearman por pares. La combinación de ambas métricas proporciona una visión más completa sobre la consistencia interna de la evaluación: mientras que Krippendorff refleja el nivel de acuerdo exacto, Spearman revela si los anotadores tienden a priorizar o penalizar los mismos modelos de manera coherente. Para más información sobre estas métricas véase la Sección 2.6 y el Anexo B.

Capítulo 4

Resultados

Este capítulo recoge los resultados obtenidos tras aplicar las distintas configuraciones y técnicas descritas en el capítulo anterior. Se presentan los resultados de los modelos sin ajustar, los obtenidos tras el entrenamiento, y los derivados del formulario de validación, así como el grado de acuerdo entre anotadores.

4.1. Resultados de los modelos sin ajustar

En esta sección se presentan los resultados obtenidos al aplicar diferentes tipos de *prompts* sobre modelos de lenguaje de propósito general (ChatGPT y DeepSeek), así como sobre modelos de HuggingFace sin ajustar, utilizando todas las descripciones explicadas en la Sección 3.6 (13 de DaReC, 14 de exámenes y 7 de Google Play Store). Esto proporcionó una referencia inicial para medir la mejora que aportan los modelos ajustados posteriormente.

En el caso de ChatGPT y DeepSeek, se utilizaron múltiples variantes de *prompting*, descritas en el Anexo A, incluyendo versiones simples, detalladas, con ejemplos (*few-shot prompting*) y con definiciones de lo que son los requisitos. En el caso de los modelos de HuggingFace sin ajustar (Meta-Llama 1B, 3B, 8B y Mistral), se evaluaron únicamente los *prompts* simple y detallado. A continuación, la Tabla 4.1.1 resume los resultados más representativos respecto a cobertura, claridad y detalle, que se han categorizado según los umbrales explicados en la Sección 3.6.

Modelo	Prompt	Cobertura	Claridad	Detalle
ChatGPT	Simple	Muy baja	Baja	Bajo
ChatGPT	Detallado	Baja	Alta	Muy alto
ChatGPT	Con ejemplo	Muy baja	Media	Medio
ChatGPT	Con definición de requisitos	Muy baja	Baja	Bajo
DeepSeek	Simple	Muy baja	Media	Medio
DeepSeek	Detallado	Baja	Alta	Muy alto
DeepSeek	Con ejemplo	Muy baja	Alta	Medio
DeepSeek	Con definición de requisitos	Muy baja	Baja	Bajo

Meta-Llama 1B	Detallado	Muy baja	Baja	Medio
Meta-Llama 3B	Detallado	Muy baja	Baja	Alto
Meta-Llama 8B	Detallado	Muy baja	Baja	Medio
Mistral	Detallado	Muy baja	Baja	Medio

Tabla 4.1.1 Rendimiento de los modelos sin ajustar

Uno de los principales factores que ha condicionado los resultados ha sido el contenido y longitud de las descripciones originales del sistema. En aquellos casos donde el sistema estaba definido con escaso contexto, la cobertura fue especialmente baja, incluso con *prompts* detallados. Por el contrario, sistemas más extensamente descritos permitieron extraer un mayor número de requisitos incluso con *prompts* simples, aunque no siempre coincidentes con los definidos originalmente en DaReC.

Respecto a los modelos de propósito general, ChatGPT y DeepSeek han generado salidas similares. DeepSeek ha ofrecido salidas más estructuradas en algunos escenarios, especialmente al utilizar *prompts* detallados, donde su claridad fue superior, pero en términos de cobertura y nivel de detalle ambos modelos tuvieron un rendimiento similar: generaron algunos requisitos no alineados con los de referencia, mostrando una tendencia a "imaginar" funcionalidades no descritas explícitamente, sobre todo con prompts detallados y respecto a los requisitos no funcionales debido a la falta de contexto explícito en muchas descripciones. Además, fueron capaces de incluir subrequisitos y matices adicionales cuando se emplearon prompts detallados pero se observó que, en algunos casos, agruparon múltiples requisitos bajo un mismo encabezado, lo que dificultaba su identificación individual. En cuanto a la numeración y organización, se detectaron diferencias según el prompt. Los prompts simples y con definición de requisitos numeraban listas por separado para RF y RNF, repitiendo identificadores mientras que los prompts detallados y con ejemplos mejoraron la consistencia, generando identificadores únicos para cada requisito (por ejemplo, RF1, NF1, etc.), especialmente en los modelos generalistas.

Los modelos *open-source* evaluados (LLaMA y Mistral) mostraron resultados limitados en todas las métricas, incluso con *prompts* detallados. Si bien el nivel de detalle fue aceptable en algunos casos, la estructura y cobertura fueron deficientes. En general, no asignaron identificadores únicos a cada requisito y, en algunos casos, generaron listas planas con afirmaciones demasiado genéricas. Además, la escala del modelo (1B, 3B, 8B) no implicó mejoras sustanciales en esta fase, lo que sugiere que el modelo necesita un ajuste fino específico para adaptarse a tareas especializadas como la generación de requisitos.

Es importante destacar que la métrica de cobertura se evaluó en relación a los requisitos existentes en el *dataset* DaReC como *ground-truth*. Bajo este criterio estricto, muchos modelos no alcanzaron un buen rendimiento. Sin embargo, si no se tuviera en cuenta dicha comparación directa, es decir, si se valoraran los requisitos generados únicamente por su adecuación al dominio descrito, las salidas de los modelos podrían considerarse razonablemente completas. Esto indica que los modelos, aunque no reproduzcan los

requisitos esperados, son capaces de generar descripciones funcionales coherentes, aunque a menudo divergentes respecto al conjunto de referencia.

Por último, otro aspecto observado en los requisitos generados, tanto por modelos ajustados como sin ajustar, fue la falta de una clasificación explícita según el grado de obligatoriedad. Ninguno de los modelos aplicó una redacción diferenciada utilizando verbos modales como “debe” (requisito obligatorio), “debería” (requisito deseable) o “podría” (requisito opcional), lo cual es habitual en metodologías formales de especificación de requisitos. Esta carencia puede afectar a la utilidad práctica de los requisitos generados en entornos reales de desarrollo.

En resumen, en esta fase preliminar, los modelos generalistas (ChatGPT y DeepSeek) ofrecieron un rendimiento superior al de los modelos *open-source* sin ajustar, especialmente en claridad estructural y nivel de detalle, aunque todos mostraron limitaciones importantes en cobertura y alineación con requisitos de referencia.

4.2. Resultados tras *fine-tuning*

En esta sección se presentan los resultados obtenidos tras aplicar técnicas de *fine-tuning* sobre los modelos *open-source* Meta-Llama y Mistral, con el objetivo de mejorar su rendimiento en la tarea de generación automática de requisitos. Esta evaluación permite valorar en qué medida el entrenamiento específico con el *dataset* DaReC aporta mejoras respecto a los modelos sin ajustar y cómo se comparan con modelos de propósito general como ChatGPT o DeepSeek.

Para esta evaluación se utilizó el *prompt* simple y el detallado (el que ofreció mejores resultados en la fase anterior) y se emplearon únicamente dos sistemas de DaReC: *2001 - space fractions*, no empleado para el entrenamiento, y un segundo sistema, empleado en el entrenamiento, para contrastar el efecto de la adaptación. Aplicando los umbrales definidos en la Sección 3.6 para cobertura, claridad y detalle, la Tabla 4.2.1 recoge los resultados más representativos:

Modelo	*Prompt*	Cobertura	Claridad	Detalle
Meta-Llama 1B	Simple	Muy baja	Baja	Bajo
Meta-Llama 1B	Detallado	Muy baja	Baja	Medio
Meta-Llama 3B	Simple	Muy baja	Baja	Bajo
Meta-Llama 3B	Detallado	Muy baja	Baja	Medio
Mistral	Detallado	Muy baja	Alta	Alto

Tabla 4.2.1 Rendimiento de los modelos open-source tras fine-tuning

Los resultados muestran que el ajuste fino ha permitido ciertas mejoras, especialmente en el modelo Mistral, que generó requisitos más estructurados, mejoró en la asignación de

identificadores únicos a cada requisito, y tuvo un mayor nivel de detalle en la redacción. Aun así, la cobertura siguió siendo limitada, en gran medida debido a la brevedad o ambigüedad de algunas descripciones presentes en el dataset DaReC. En este sentido, es importante volver a destacar que la cobertura se evaluó comparando con los requisitos definidos en dicho *dataset* como *ground-truth*, lo que implica un criterio muy estricto. Si se evaluaran únicamente los requisitos generados por su coherencia con el dominio del sistema, independientemente de su coincidencia exacta con los de DaReC, los resultados podrían considerarse más satisfactorios.

En los modelos Meta-Llama (1B y 3B), en cambio, no se observaron mejoras notables, lo que sugiere que el tamaño del modelo y su arquitectura podrían estar limitando su capacidad para aprender patrones y generar salidas estructuradas incluso tras el entrenamiento adicional. Además, durante la evaluación se detectó que estos modelos tendían a repetir frases o entrar en bucles, lo cual afectó negativamente tanto a la claridad como a la utilidad de las salidas.

Tanto en los modelos ajustados como en los no ajustados, se mantuvo la ausencia de una clasificación por grado de obligatoriedad. Ninguno de ellos incorporó una redacción diferenciada mediante verbos modales como “debe” (obligatorio), “debería” (deseable) o “podría” (opcional), una práctica habitual en especificaciones formales de requisitos, como se comentó anteriormente.

Dado que Mistral-7B fue el único modelo que demostró una mejora significativa en las métricas evaluadas, fue el seleccionado como representante de los modelos *fine-tuned* para la evaluación humana (véase Sección 3.7).

Para ofrecer una visión comparativa más clara, en la Figura 4.2.1 se muestra una gráfica resumen que compara el rendimiento de los *open-source* antes y después del *fine-tuning* de los modelos para cada una de las métricas evaluadas:

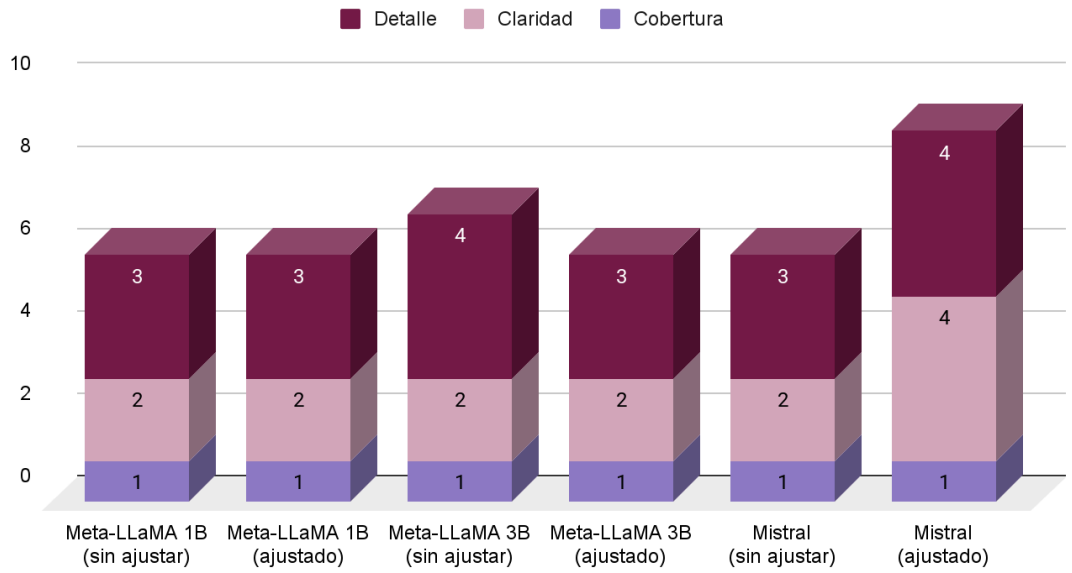

Figura 4.2.1 Comparativa del rendimiento de modelos open-source antes y después del fine-tuning. Escala: 1 (muy bajo), 2 (bajo), 3 (medio), 4 (alto), 5 (muy alto).

Finalmente se incluye también en la Figura 4.2.2 una comparativa del rendimiento observado en los modelos comerciales ChatGPT y DeepSeek y el modelo Mistral ajustado, según la misma escala:

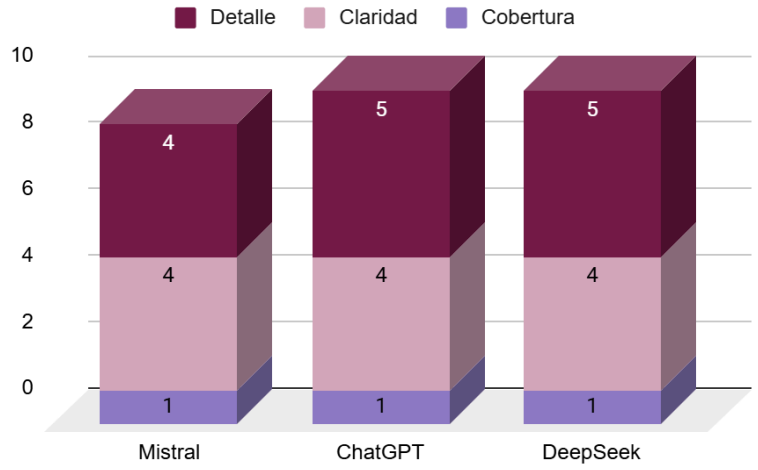

Figura 4.2.2 Comparativa del rendimiento de modelos open-source ajustados y modelos de propósito general. Escala: 1 (muy bajo), 2 (bajo), 3 (medio), 4 (alto), 5 (muy alto).

Esta visualización permite observar cómo el ajuste fino ha tenido un impacto positivo en la estructura y el detalle de los requisitos generados del modelo Mistral. No obstante, los modelos ajustados aún no alcanzan el nivel de rendimiento global de los modelos de propósito general como ChatGPT o DeepSeek.

4.3. Resultados del formulario de validación

Tras el análisis cuantitativo de las salidas generadas, tanto por modelos sin ajustar como por modelos tras *fine-tuning*, se consideró fundamental complementar los resultados con una evaluación subjetiva. Para ello, se diseñó un formulario de validación dirigido a usuarios, cuyo objetivo era identificar qué modelo (independientemente de su arquitectura o ajuste) generaba requisitos percibidos como más claros, completos y útiles desde una perspectiva humana (véase Sección 3.7).

A partir de las puntuaciones obtenidas en todas las preguntas, se calcularon las medias globales de cada salida, considerando todas las dimensiones evaluadas. Esto permite establecer un ranking general de las salidas y, en consecuencia, de los modelos que las generaron en función de la percepción por parte de los usuarios. En la Tabla 4.3.1 se resumen estos resultados:

Salida	Media global
Llama 8B (sin ajustar)	6,64
Mistral (ajustado)	6,43
ChatGPT	6,29

Mistral (sin ajustar)	5,82
Llama 1B (sin ajustar)	5,46
DaReC	5,43
DeepSeek	5,25

Tabla 4.3.1 Puntuación media global por modelo (escala de 1 a 10).

Como puede observarse, el modelo Llama 8B sin ajustar obtuvo la puntuación media más alta, superando incluso a modelos ampliamente utilizados como ChatGPT. También destaca el buen rendimiento del modelo Mistral tras el *fine-tuning*, que se situó como el segundo modelo mejor valorado, por delante del resto de modelos sin ajustar e incluso del modelo base DaReC. Este resultado refuerza la hipótesis de que el entrenamiento específico con datos de requisitos puede mejorar significativamente la percepción subjetiva de calidad de las salidas generadas.

Además del promedio global, se calcularon las medias por cada pregunta del formulario, lo que permitió analizar el rendimiento de los modelos en aspectos concretos como claridad, corrección gramatical, cobertura o ambigüedad, entre otros. La Figura 4.3.1 muestra las puntuaciones medias obtenidas por cada modelo en cada dimensión evaluada:

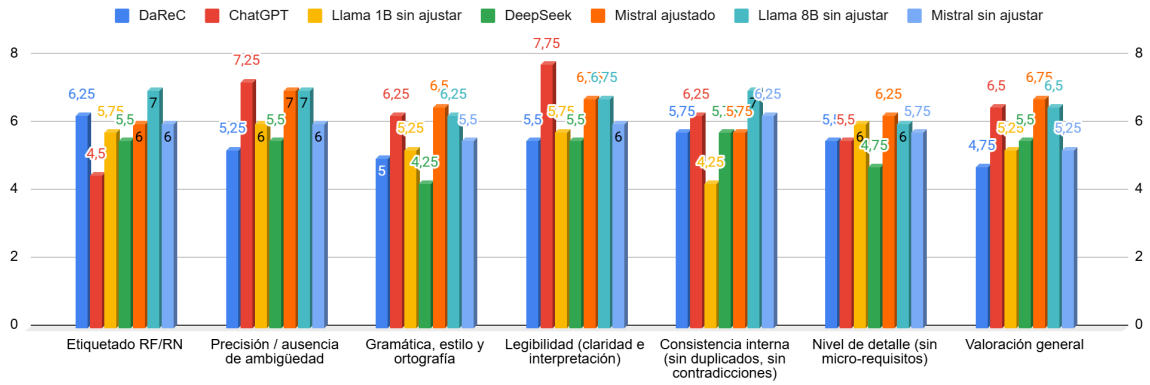

Figura 4.3.1 Puntuación media por pregunta y modelo

Los resultados muestran que el modelo Llama 8B sin ajustar, el mejor valorado en términos generales, obtuvo puntuaciones especialmente altas en el etiquetado correcto de RF/RNF, la precisión/ausencia de ambigüedad, la consistencia interna y la legibilidad. El modelo Mistral ajustado fue también consistentemente bien valorado en la mayoría de dimensiones, especialmente en gramática, estilo y ortografía, y en el nivel de detalle.

Por su parte, ChatGPT fue el modelo mejor puntuado en precisión y legibilidad, lo que confirma su robustez como modelo de propósito general, si bien fue notablemente peor en la diferenciación entre requisitos funcionales y no funcionales, una dimensión especialmente relevante en este contexto. El modelo DeepSeek obtuvo puntuaciones discretas y, en general, inferiores a las de los modelos más avanzados, lo que podría indicar una menor adecuación de este modelo a la tarea específica de generación de requisitos.

En cuanto al modelo DaReC, que representa los requisitos originales del *dataset*, se observó una buena puntuación en el etiquetado RF/RNF (6,25), lo que refleja su consistencia estructural, pero una valoración general más baja (4,75), posiblemente debido a una redacción menos clara o pulida que la de los modelos generativos actuales, más centrados en la fluidez y comprensión del lenguaje natural.

Con el objetivo de analizar la fiabilidad de la validación subjetiva, se calcularon métricas de consistencia interanotador. En particular, se utilizó el alfa de Krippendorff como medida del grado de acuerdo entre los evaluadores, tanto de forma global como por cada una de las preguntas del formulario.

En la Tabla 4.3.2 se presentan los valores del alfa obtenidos para cada una de las dimensiones evaluadas:

Dimensión evaluada	Alfa de Krippendorff
Etiquetado RF/RNF	-0,165
Precisión / ausencia de ambigüedad	-0,141
Gramática, estilo y ortografía	-0,146
Legibilidad (claridad e interpretación)	-0,171
Consistencia interna	-0,156
Nivel de detalle	-0,213
Valoración general	-0,150
Media global	**-0,163**

Tabla 4.3.2 Alfa de Krippendorff por dimensión

Como puede observarse, todos los valores obtenidos fueron negativos, lo cual indica que el nivel de acuerdo entre los anotadores fue incluso inferior al que se esperaría por azar. Este resultado sugiere una notable disparidad en la manera en que los distintos participantes interpretaron y evaluaron las salidas generadas por los modelos. Este desacuerdo puede deberse a factores como la subjetividad inherente a cada dimensión evaluada, diferencias de criterio personales, la falta de una guía de evaluación común y detallada o la diferencia en la formación o experiencia entre los participantes.

Por otro lado, de los siete criterios evaluados, el de gramática, estilo y ortografía y el de precisión / ausencia de ambigüedad mostraron el menor desacuerdo entre anotadores, mientras que el de nivel de detalle y legibilidad registraron el mayor, lo que indica que los juicios puramente lingüísticos son más uniformes que los basados en criterios interpretativos.

Con el fin de complementar el análisis de la fiabilidad interanotador, se calcularon también los coeficientes de correlación de Spearman entre los rankings otorgados por cada

pareja de evaluadores (Tabla 4.3.3). Esta medida permite cuantificar el grado de relación monotónica entre los juicios de los distintos participantes, evaluando si tienden a coincidir en su ordenación relativa de las salidas generadas, aunque no necesariamente en los valores absolutos asignados.

Par de anotadores	Correlación de Spearman
Anotador 1 - Anotador 2	-0,185
Anotador 1 - Anotador 3	0,436
Anotador 1 - Anotador 4	-0,714
Anotador 2 - Anotador 3	0,585
Anotador 2 - Anotador 4	0,482
Anotador 3 - Anotador 4	0,200
Media global	**0,134**

Tabla 4.3.3 Correlación de Spearman entre anotadores

Como puede observarse, los coeficientes obtenidos son muy variables. Algunos pares de evaluadores muestran una correlación moderadamente positiva (por ejemplo, Anotador 2 - Anotador 3 con $\rho = 0{,}585$), lo que sugiere cierta coherencia en la forma de valorar las respuestas. Sin embargo, en otros casos, como Anotador 1 - Anotador 4 ($\rho = -0{,}714$), se observa una fuerte correlación negativa, lo que indica que las respuestas mejor valoradas por un anotador fueron sistemáticamente peor valoradas por el otro.

En conjunto, los resultados obtenidos mediante la correlación de Spearman refuerzan lo ya apuntado por el alfa de Krippendorff, es decir, existe una alta variabilidad en las valoraciones individuales, lo que refleja la dificultad de alcanzar un consenso claro en la evaluación subjetiva de requisitos generados automáticamente.

Capítulo 5

Discusión y conclusiones

Este capítulo integra el análisis crítico de los resultados obtenidos con las conclusiones finales del trabajo. En primer lugar, se interpretan los datos generados durante la experimentación, destacando las tendencias observadas y contrastándolas con los objetivos iniciales del proyecto. A continuación, se discuten las principales limitaciones técnicas y metodológicas del estudio, así como el impacto potencial de las técnicas aplicadas en la reducción de errores y ambigüedades durante la generación de requisitos. Finalmente, se evalúa el grado de cumplimiento de los objetivos planteados, se sintetizan las conclusiones más relevantes y se proponen posibles líneas de mejora y futuras investigaciones que podrían ampliar el alcance y la aplicabilidad de este enfoque.

5.1. Interpretación de resultados

Los resultados obtenidos en el presente trabajo permiten extraer patrones relevantes sobre el comportamiento y las limitaciones de los distintos modelos de lenguaje aplicados a la generación automática de requisitos.

Los modelos de propósito general (ChatGPT y DeepSeek) ofrecieron una cobertura reducida respecto a los requisitos originales del *dataset* DaReC, extrayendo solo los RF más evidentes, independientemente del estilo de *prompt* empleado. Sin embargo, sus salidas fueron más estructuradas, claras y detalladas que las de los modelos *open-source* sin ajustar, especialmente cuando se emplearon *prompts* detallados. Este comportamiento puede atribuirse en parte al tamaño y la escala de los modelos de propósito general (disponen de decenas o cientos de miles de millones de parámetros frente a las 1–8 B de los modelos open-source), así como a técnicas avanzadas como el *instruction tuning* y el RLHF, que aportan una mayor capacidad para organizar y enriquecer las respuestas aun sin entrenamiento específico en el dominio.

Por su parte, los modelos *open-source* de Hugging Face (Meta-Llama 1B/3B/8B y Mistral) sin ajustar mostraron un rendimiento inferior: claridad estructural pobre, listas planas sin numeración única y detalle escaso. Tras aplicar *fine-tuning* con LoRA y cuantización 4-bit, únicamente el modelo Mistral mejoró en la claridad y detalle de los requisitos, aunque sin incrementar la cobertura. En cambio, las variantes LLaMA 1B y 3B apenas experimentaron cambios, lo que sugiere que debido a su menor tamaño requieren un volumen de datos mayor o ajustes más intensivos para lograr una adaptación eficaz al dominio de DaReC.

El buen desempeño del modelo Mistral ajustado, especialmente en cuanto a claridad, estructura y nivel de detalle, motivó su selección para el formulario de validación. Su capacidad para generar secciones RF/RNF numeradas de forma consistente y enriquecerlas

con ejemplos y casos de uso coincidió con la percepción de mayor calidad por parte de los anotadores.

La baja cobertura general observada se atribuye en gran parte a la ambigüedad y brevedad de muchas descripciones de DaReC. Enunciados muy cortos no proporcionan contexto suficiente para extraer todos los requisitos y, en consecuencia, todos los sistemas fallaron al intentar cubrir el conjunto de referencia completo. Esto pone de manifiesto que, más allá de la potencia del modelo, el diseño del *prompt* y la riqueza del *input* son determinantes para maximizar la cobertura de requisitos.

La evaluación humana confirmó que los modelos *open-source* de gran tamaño (Llama-8B y Mistral-7B) lideran la percepción general de calidad, junto con ChatGPT. Además, estos modelos mantienen perfiles elevados y relativamente planos a lo largo de los siete criterios, mientras que modelos como DeepSeek y DaReC presentan descensos pronunciados en aspectos como detalle y consistencia. Por otra parte, Mistral ajustado alcanza picos destacados en gramática, legibilidad y detalle, confirmando la eficacia del *fine-tuning* aplicado.

Para cuantificar la consistencia de estas valoraciones, se calculó el alfa de Krippendorff por cada criterio (Tabla 4.3.2). Los valores negativos (entre –0,213 y –0,141) obtenidos en todos los casos indican que el acuerdo entre anotadores quedó por debajo del nivel aleatorio esperado, reflejando la alta subjetividad y la falta de una guía de evaluación homogénea. Esta baja fiabilidad sugiere que cada evaluador interpretó de forma distinta los criterios, especialmente en el nivel de detalle ($\alpha = –0{,}213$) y legibilidad ($\alpha = –0{,}171$). Este análisis por criterio sugiere que, aunque el acuerdo general es bajo, existe mayor fiabilidad en los criterios técnicos, mientras que criterios más interpretativos requieren pautas de evaluación más precisas para reducir la variabilidad entre anotadores.

Complementariamente, el coeficiente de Spearman entre pares de anotadores (Tabla 4.3.3) ofrece una visión del grado de concordancia en los *rankings* de modelos. Con una media global de $\rho = 0{,}134$, el acuerdo es muy bajo. Solo el par de anotadores 2–3 mostró correlación moderada ($\rho = 0{,}585$). La gran variabilidad refuerza la conclusión de que las valoraciones son altamente personales y difíciles de alinear.

En resumen, estos análisis indican que, aunque ciertos modelos destacan de forma consistente (Llama-8B y Mistral fine-tuned), la fiabilidad de la evaluación subjetiva es baja. Para futuros estudios, sería recomendable ampliar la muestra de evaluadores, estandarizar las pautas de anotación y explorar métricas de acuerdo alternativas (por ejemplo, ICC) o agrupar criterios para mejorar la consistencia interanotador.

5.2. Limitaciones del estudio

El presente estudio presenta una serie de limitaciones que deben ser tenidas en cuenta al interpretar sus resultados. En primer lugar, el *dataset* empleado en el *fine-tuning* se construyó únicamente a partir de 49 descripciones, una muestra que aunque suficiente para prototipos exploratorios, resulta reducida para entrenar y evaluar modelos de gran escala de forma robusta. Por esta misma limitación del número de sistemas con requisitos ya formulados, se empleó una única descripción como conjunto de validación para evaluar el

entrenamiento, lo que no permite valorar completamente la adaptabilidad del modelo a dominios variados o a sistemas con características muy distintas.

Por otro lado, el número de evaluadores participantes fue reducido (sólo 4), con perfiles diversos (distintos niveles de formación, experiencia y familiaridad con la ingeniería de requisitos) y no contaban con una guía de evaluación formal, lo que puede haber contribuido a la variabilidad observada en las puntuaciones asignadas y limita la generalización de las conclusiones extraídas. Asimismo, el proceso de evaluación se centró exclusivamente en la percepción subjetiva de las salidas generadas, sin considerar el impacto que estas podrían tener en fases posteriores del ciclo de vida del software, como el diseño, la implementación o las pruebas.

Por último, las técnicas de optimización de memoria (cuantización en 4 bits y LoRA) fueron imprescindibles, pero también introdujeron degradaciones en el entrenamiento y la coherencia de las salidas. Asimismo, la imposibilidad de entrenar variantes de mayor tamaño o explorar más configuraciones de hiperparámetros restringe la exhaustividad del análisis.

En conjunto, estas limitaciones sugieren que, para avanzar hacia una solución industrialmente viable, sería necesario ampliar y diversificar el *dataset*, mejorar la fiabilidad de la evaluación humana, explorar métricas automáticas, probar arquitecturas de mayor escala y perfeccionar las estrategias de *prompting* y *fine-tuning*.

5.3. Impacto en la reducción de errores y ambigüedades

Uno de los objetivos principales del presente trabajo era explorar el potencial de los modelos de lenguaje para contribuir a la reducción de errores y ambigüedades en la fase de generación de requisitos.

Gracias al *prompt* detallado y al *fine-tuning*, Mistral-7B y Llama-8B producen listas de requisitos correctamente agrupados en funcionales y no funcionales, con numeración única y consistente (por ejemplo, RF1, RNF1...). Este formato estandarizado disminuye la probabilidad de omisiones o duplicaciones y facilita la revisión por parte de analistas humanos.

Aunque los modelos aún no realizan una clasificación por grado de obligatoriedad en su redacción, los modelos *fine-tuned* generan descripciones gramaticalmente correctas y usan terminología precisa, reduciendo ambigüedades asociadas a verbos modales imprecisos. Estos modelos consiguieron la mayor puntuación (6,5) de entre todos los modelos evaluados en la categoría de gramática y estilo.

Por otro lado, el *prompt* detallado induce la generación de supuestos explícitos, así como ejemplos de uso o flujos de interacción. Al anticipar posibles casos límite, estos detalles minimizan malentendidos o errores en etapas posteriores de diseño e implementación.

En conjunto, estos avances automatizan tareas de redacción que suelen estar sujetas a errores humanos (olvidos, inconsistencias de estilo, ambigüedades de lenguaje natural) y proporcionan un punto de partida estructurado y completo. Aunque aún es necesaria la supervisión de analistas de requisitos para validar y ajustar el resultado final, los modelos

fine-tuned ofrecen un impacto tangible en la eficiencia y calidad de la fase de especificación, reduciendo tanto la tasa de errores como la ambigüedad inherente a la documentación de requisitos.

5.4. Cumplimiento de objetivos

El propósito principal de este TFG era poner a prueba el potencial de los LLMs para asistir y automatizar la redacción de requisitos de software. Los resultados obtenidos demuestran que, con un diseño de *prompt* apropiado y técnicas de *fine-tuning* específicas, los modelos de lenguaje pueden ofrecer un apoyo real y significativo al proceso tradicionalmente manual de elaboración de requisitos.

Respecto a los objetivos más específicos, se han explorado distintos LLMs como GPT, LLaMA, DeepSeek y Mistral para realizar pruebas de generación de requisitos con diferentes *prompts* y descripciones, así como para aplicar técnicas *fine-tuning* (LoRA junto con cuantización en 4 bits). Además, se experimentó con diferentes configuraciones e hiperparámetros (épocas, repetition_penalty), lo que permitió seleccionar las combinaciones óptimas para cada modelo, destacando la importancia de adaptar tanto los datos de entrada como los parámetros de generación.

ChatGPT y DeepSeek proporcionaron un *baseline* sólido en claridad y detalle, mientras que las variantes de LLaMA y Mistral requirieron ajuste fino para acercarse a ese nivel de calidad. Esta comparación permitió identificar que los servicios web destacan de salida, pero que un modelo open-source como Mistral-7B, tras LoRA y cuantización, podría alcanzar su desempeño en criterios clave.

Los capítulos 3 y 4 reflejan la metodología de evaluación comparativa llevada a cabo, abarcando desde la generación de requisitos hasta su valoración por cuatro anotadores en siete criterios, proporcionando así un diagnóstico exhaustivo de cada sistema.

Finalmente, en este capítulo 5 se recoge la reflexión sobre la utilidad práctica de los LLMs y las posibles líneas futuras. Si bien la revisión humana experta sigue siendo necesaria, los modelos afinados pueden reducir drásticamente las ambigüedades y errores en la fase de redacción de requisitos, liberando a los analistas para tareas de mayor valor añadido.

5.5. Líneas de investigación y mejoras futuras

Los resultados obtenidos en este trabajo abren diversas vías para continuar investigando y mejorar la aplicación de técnicas de inteligencia artificial en la generación automática de requisitos en el desarrollo de software. Algunas líneas de investigación y mejoras futuras relevantes son las siguientes:

- **Ampliación y diversificación del *dataset*:** La calidad y generalización de los modelos ajustados se ven limitadas por el tamaño y la diversidad del conjunto de datos disponible. En futuros trabajos, sería recomendable recopilar y anotar un mayor volumen de descripciones de sistemas junto con sus requisitos, incluyendo

distintos dominios y niveles de complejidad, para mejorar la robustez y capacidad de generalización del modelo.

- **Exploración de técnicas avanzadas de ajuste fino:** Aunque en este proyecto se ha utilizado LoRA y cuantización en 4 bits para optimizar el entrenamiento, existen otras técnicas como el RLFH que podrían mejorar la alineación de los modelos con las expectativas reales de los usuarios y la calidad de los requisitos generados.

- **Mejora en la generación y evaluación de *prompts*:** La calidad de las respuestas generadas está altamente condicionada por el diseño de los *prompts* por lo que en investigaciones futuras, se podrían implementar técnicas automáticas o semi-automáticas para la optimización de *prompts*, así como explorar métodos de evaluación más objetivos y automatizados que complementen la validación subjetiva.

- **Estudio de métricas de evaluación y acuerdos entre anotadores:** Dada la baja consistencia observada en la validación subjetiva, es necesario investigar métodos más robustos para evaluar la calidad de los requisitos generados, así como estrategias para mejorar la concordancia entre evaluadores, incluyendo formación o guías más detalladas.

- **Pruebas en infraestructuras de mayor capacidad computacional:** Experimentar con modelos de mayor tamaño en clústeres equipados con GPUs de gran memoria eliminaría las restricciones de batch size, longitud de contexto y necesidad de cuantización extrema. Esto viabilizaría explorar ajustes de hiperparámetros más agresivos, contextos más largos y múltiples épocas sin recurrir a compresión.

- ***Fine-tuning* de modelos comerciales de gran escala:** Obtener acceso a APIs o licencias para afinar modelos como GPT-4 con técnicas de LoRA o PEFT adaptadas a su infraestructura permitiría combinar su amplia preformación con un ajuste preciso al dominio de la ingeniería de requisitos. Esta línea evaluaría si los LLMs líderes, ya entrenados con RLHF, pueden mejorar aún más su rendimiento en tareas especializadas.

- **Clasificación de grado de obligatoriedad:** Incorporar un módulo de post-proceso o un *prompt* dinámico que distinga requisitos “debe” (obligatorios), “debería” (recomendables) y “podría” (opcionales) enriquecería la semántica de las especificaciones, alineándose con metodologías formales y reduciendo ambigüedades sobre prioridades.

- **Generación automática de casos de uso e historias de usuario:** Extender la capacidad de los modelos para que, a partir de cada requisito generado, produzcan automáticamente casos de uso detallados o historias de usuario con roles, acciones y criterios de aceptación facilitaría la transición desde la especificación hasta el diseño y desarrollo, promoviendo una documentación más completa y útil para los equipos de desarrollo y *stakeholders*.

Estas líneas futuras permitirán profundizar en la integración de técnicas de inteligencia artificial en el desarrollo del software, aumentando su automatización, precisión y aplicabilidad en entornos reales.

Bibliografía

[1] Chowdhary, K., and Chowdhary, K. R. (2020). Natural language processing. *Fundamentals of artificial intelligence*, 603-649.

[2] Nuseibeh, B., and Easterbrook, S. (2000, May). Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 35-46).

[3] 'Alvy, Á. I. (2014, June 4). El error de software que convirtió un lanzamiento espacial en carísimos fuegos artificiales. RTVE.es. Disponible en: https://www.rtve.es/noticias/20140604/error-software-convirtio-lanzamiento-espacial-carisimos-fuegos-artificiales/948262.shtml

[4] Martínez, O. G. E., Reyes, S. V., and González, A. M. (2021). Use of natural language processing techniques for software requirements detection. *South Florida Journal of Development*, *2*(5), 7323-7335. Available from: https://ojs.southfloridapublishing.com/ojs/index.php/jdev/article/download/911/788/2595

[5] Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M., Chioasca, E.-V., and Batista-Navarro, R. T. (2021). Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. ACM Computing Surveys , 54(3), 1–41. Article 55. Available from: https://doi.org/10.1145/3444689

[6] Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., ... and Xie, X. (2024). A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, *15*(3), 1-45.

[7] Brown, T. B., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.

[8] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... and Jegou, H. (2023). *LLaMA: Open and Efficient Foundation Language Models*. arXiv preprint arXiv:2302.13971.

[9] DeepSeek-AI, Xiao Bi,et al.. (2024). *DeepSeek LLM: Scaling Open-Source Language Models with Longtermism*. arXiv. Available from: https://arxiv.org/abs/2401.02954

[10] Bergmann D. What is fine-tuning? IBM Think [Internet]. Available from: https://www.ibm.com/think/topics/fine-tuning

[11] University Data Resource Centre, University of Lucknow. Available from: https://udrc.lkouniv.ac.in/Content/DepartmentContent/SM_6b8ee085-2d45-4a29-9c72-5e6c1d6d8551_34.pdf

[12] J. Cambria y B. White, “Jumping NLP Curves: A Review of Natural Language Processing Research,” IEEE Computational Intelligence Magazine, vol. 9, no. 2, pp. 48–57, May 2014.

[13] GeeksforGeeks. (2025, May 1). Phases of Natural Language Processing (NLP). GeeksforGeeks. Available from: https://www.geeksforgeeks.org/machine-learning/phases-of-natural-language-processing-nlp/

[14] Khan, M.A., Aydemir, F.B., Oriol, M. (2025) 8th International Workshop on Natural Language Processing for Requirements Engineering (NLP4RE'25). Barcelona, Spain, April 7-11

[15] Chowdhery, A., et al.. (2022). *PaLM: Scaling language modeling with Pathways*. *Journal of Machine Learning Research*, 24(240), 1–113.

[16] Priyanshu, A., Maurya, Y., and Hong, Z. (2024). *AI governance and accountability: An analysis of Anthropic's Claude*. arXiv. Available from: https://arxiv.org/abs/2407.01557

[17] Moulton, B. (2024, 14 de septiembre). *Mistral Large 2: French open source model newcomer leading AI innovation, surpassing Llama3.1*. Medium. Available from: https://beckmoulton.medium.com/mistral-large-2-french-open-source-model-newcomer-leading-ai-innovation-surpassing-llama3-1-78d99f45414a

[18] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... and Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30. Available from: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[19] IBM. (n.d.). What are Large Language Models (LLMs)? IBM Think. Available from: https://www.ibm.com/think/topics/large-language-models

[20] Cloudflare. (n.d.). What is a large language model? Available from: https://www.cloudflare.com/es-es/learning/ai/what-is-large-language-model/

[21] J. P. Hutchison, "Language Model Perplexity and Overfit," J. Comput. Linguist., 2012.

[22] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... and 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.

[23] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., ... and Zettlemoyer, L. (2020). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. ACL 2020.

[24] Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2019, April 21). BERTScore: Evaluating Text Generation with BERT. arXiv.org. Available from: https://arxiv.org/abs/1904.09675

[25] Howard, J., and Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. ACL 2018.

[26] Devlin, J., Chang, M. W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL-HLT 2019.

[27] Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., ... and Le, Q. (2022). Finetuned Language Models Are Zero-Shot Learners. ICML 2022.

[28] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... and Christiano, P. (2022). Training language models to follow instructions with human feedback. NeurIPS 2022.

[29] Hugging Face (2023). PEFT: Parameter-Efficient Fine-Tuning. Available from: https://huggingface.co/blog/peft

[30] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics. Available from: https://aclanthology.org/P02-1040.pdf

[31] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In Text Summarization Branches Out, pages 74–81, Barcelona, Spain. Association for Computational Linguistics. Available from: https://aclanthology.org/W04-1013.pdf

[32] K. Krippendorff, Content Analysis: An Introduction to Its Methodology, 3rd ed. Sage, 2013.

[33] Cohen, J. (1960). A coefficient of agreement for nominal scales. Educational and Psychological Measurement, 20(1), 37–46. Available from: https://doi.org/10.1177/001316446002000104

[34] Fleiss, J. L. (1971). Measuring nominal scale agreement among many raters. Available from: Psychological Bulletin, 76(5), 378–382. Available from: https://doi.org/10.1037/h0031619

[35] Shrout PE, Fleiss JL. Intraclass correlations: uses in assessing rater reliability. Psychological Bulletin. Available from: https://www.researchgate.net/file.PostFileLoader.html?id=573b2e9af7b67e2efd6743c6&assetKey=AS:362744118300682@1463496346017

[36] Spearman, C. (1904). The Proof and Measurement of Association Between Two Things. The American Journal of Psychology, 15(1), 72–101.

[37] EAE Barcelona. La importancia de los prompts en el uso de la IA. n.d. Available from: https://www.eaebarcelona.com/es/blog/que-son-los-prompts-y-su-importancia

[38] diegomurciamart/DaReC. (n.d.). GitHub. Available from: https://github.com/diegomurciamart/DaReC

[39] Méndez G. IEEE 830: Recommended Practice for Software Requirements Specifications. Universidad Complutense de Madrid; n.d. Available from: https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf

[40] OpenAI. ChatGPT: Overview. n.d. Available from: https://openai.com/es-ES/chatgpt/overview/

[41] OpenAI. GPT-4o Mini: Advancing cost-efficient intelligence. n.d. Available from: https://openai.com/es-ES/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

[42] Brown T, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, et al. Language Models are Few-Shot Learners. arXiv preprint arXiv:2005.14165; 2020. Available from: https://arxiv.org/pdf/2005.14165

[43] Deepseek AI. deepseek-ai. n.d. Available from: https://huggingface.co/deepseek-ai

[44] Deepseek AI. DeepSeek-R1-Distill-Llama-8B. n.d. Available from: https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-8B

[45] Meta. Llama-3.2-1B-Instruct. n.d. Available from: https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct

[46] Meta. Llama-3.2-3B-Instruct. n.d. Available from: https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

[47] Meta. Llama-3.1-8B-Instruct. n.d. Available from: https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

[48] Mistral AI. Mistral-7B-Instruct-v0.3. n.d. Available from: https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3

[49] José Antonio Gutierrez (Guti). Servidor de prácticas *hendrix*. Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza; n.d. Available from: https://diis.unizar.es/WebEst/hendrix/

Acrónimos

API	Application Programming Interface
GPT	Generative Pre-trained Transformer
IA	Inteligencia Artificial
IR	Ingeniería de Requisitos
LLaMA	Large Language Model Meta AI
LLM	Large Language Models (Modelos de Lenguaje a Gran Escala)
LoRA	Low-Rank Adaptation
NLP	Natural Language Processing (Procesamiento del Lenguaje Natural)
PEFT	Parameter-Efficient Fine-Tuning
RF	Requisito Funcional
RLFH	Reinforcement Learning from Human Feedback (Aprendizaje por refuerzo con retroalimentación humana)
RNF	Requisito No Funcional

Índice de figuras

Índice de tablas

Anexos

Anexo A

Prompts utilizados para la generación automática de requisitos

A.1 Prompt directo (castellano)

Prompt simple en español, sin ejemplos previos.

> *Extrae los requisitos funcionales y no funcionales del siguiente sistema.*
>
> *<descripción del sistema>*

A.2 Prompt directo (inglés)

Prompt equivalente al A.1, pero en inglés.

> *Extract the functional and non-functional requirements from the following system.*
>
> *<descripción del sistema>*

A.3 Prompt directo (inglés) con definición de términos

Prompt en inglés que solicita requisitos funcionales y no funcionales, con una breve definición de ambos tipos antes de formular la instrucción principal.

> *Functional requirements are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. Non-functional requirements are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another.*
> *Now, based on this description, extract the functional and non-functional requirements from the following system.*
>
> *<descripción del sistema>*

A.4 Prompt contextualizado con ejemplo (few-shot, inglés)

Prompt que incluye un ejemplo previo de cómo estructurar los requisitos (técnica few-shot).

> *Here is an example of extracted requirements from another system and its description:*
>
> *Description:*
> *<descripción del sistema>*
>
> *Requirements:*
> *<requisitos funcionales y no funcionales del sistema>*

A.5 Prompt con system message (instrucciones detalladas, inglés)

Instrucción que podría usarse como system prompt (en entornos con roles diferenciados). Define expectativas de formato, estilo y precisión, con énfasis en la claridad y testabilidad.

> *I'll give you a system message to use through all the conversation:*
>
> *You are an assistant tasked with taking the client proposals and producing a structured set of functional and non-functional requirements. Specifically:*
> *Include all explicitly stated requirements in the provided references.*
> *Infer any implicit requirements that are suggested by context but not stated outright.*
> *Write no vague or overly generic statements. Each requirement must be specific and testable—avoid phrases like "the system should be user friendly."*
> *Group each requirement as functional or non-functional under clearly named sections, and label them with a unique identifier (for example, "F1, F2, ..." for functional requirements and "NF1, NF2, ..." for non-functional requirements).*
> *Make sure the final set of requirements has enough detail to guide a development team in understanding and potentially implementing them.*
> *Your output must be well-organized, clear, and sufficiently detailed to convey exactly what the system needs to do (functional) and what constraints or qualities the system must meet (non-functional).*
>
> *<descripción del sistema>*

A.6 Prompt detallado (castellano)

Prompt en español que solicita una interpretación precisa, asumiendo y concretando ambigüedades potenciales, con orientación a implementación.

> *Dada la siguiente descripcion de un sistema, dime con todo detalle y sin ambiguedades ni vaguedades lo que deberia de saber alguien que fuera a implementar dicho sistema. Si es necesario, asume como quieras dando ejemplos concretos, las dudas que podrian surgir como por ejemplo, cuando haya algo sobre tiempos pero no se especifique cuanto concretamente, cuando hay condiciones pero no esten expresamente cuales, temas de conexiones entre dispositivos, etc. Presenta la información en formato de requisitos funcionales y no funcionales.*
>
> *<descripción del sistema>*

A.7 Prompt detallado (inglés)

Prompt equivalente al A.6, pero en inglés.

> *Given the following system description, tell me in full detail and without ambiguities or vagueness everything that someone implementing the system should know. If necessary, make assumptions as you see fit, providing concrete examples for any potential doubts that may arise—for instance, when there are references to time without specific durations, when conditions are mentioned but not explicitly defined, or when dealing with device connections, etc. Present the information in the form of functional and non-functional requirements.*
>
> *<descripción del sistema>*

A.8 Resumen de variantes evaluadas

Código	Idioma	Estilo
A.1	ES	Directo
A.2	EN	Directo
A.3	EN	Directo con definiciones
A.4	EN	Few-short (con ejemplos)
A.5	EN	System prompt
A.6	ES	Detallado
A.7	EN	Detallado

Tabla A.8.1 Resumen de las variantes de prompts evaluadas

Anexo B

Métodos de evaluación

B.1 Evaluación automática

Las métricas automáticas permiten medir rápidamente similitudes cuantitativas entre las salidas del modelo y unos textos de referencia (*ground-truth*), pero no capturan aspectos de claridad, consistencia interna o adecuación al dominio. Algunas de las métricas más comunes son:

- **Perplejidad (*Perplexity*)**: Mide la capacidad del modelo para predecir la siguiente palabra en una secuencia. Cuanto menor sea la perplejidad, mejor se ajusta el modelo al texto de referencia [21].
- **BLEU (*Bilingual Evaluation Understudy*)**: Mide la coincidencia de n-gramas entre la salida y una o varias referencias, penalizando la longitud excesiva [30].
- **ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*)**: Conjunto de métricas centradas en recuperación de n-gramas y subsecuencias. ROUGE-N evalúa n-gramas y ROUGE-L emplea la longitud de la subsecuencia más larga común [31].
- **BERTScore**: Mide la similitud semántica token-a-token usando *embeddings* de BERT, calculando precisión, *recall* y F1 sobre vectores de alto nivel [24].

B.2 Evaluación humana

La evaluación por anotadores expertos es esencial cuando la generación de texto debe responder a criterios complejos de dominio, usabilidad o interpretación. Suele implicar:

1. **Definición de criterios**: Se establecen dimensiones tales como corrección semántica (adecuación al contenido de referencia), coherencia y consistencia (sin contradicciones ni repeticiones), claridad y legibilidad (gramática, estilo) o completitud y nivel de detalle.
2. **Escalas de valoración**: Sistemas ordinales (por ejemplo, 1–5 o 1–10) o categóricos (malo/regular/bueno/excelente).
3. **Protocolo de anotación**: Guías claras para los evaluadores, ejemplos calibrados y sesiones de entrenamiento para mitigar sesgos.
4. **Muestreo de salidas**: Evaluación de un subconjunto representativo para comparar mejoras.

B.3 Medidas de acuerdo interanotador

La fiabilidad de la evaluación humana depende de la consistencia entre evaluadores. Existen varias métricas que varían en la escala de valoración (nominal vs. ordinal vs. continuo) y el número de anotadores:

- **Alfa de Krippendorff**: Mide el acuerdo en datos nominales, ordinales o de intervalos, y tolera valores faltantes. Se basa en discrepancias observadas vs. esperadas por azar [32].
- **Kappa de Cohen**: Para dos anotadores, ajusta el porcentaje de acuerdo por el azar [33].
- **Kappa de Fleiss**: Generaliza el kappa de Cohen a más de dos anotadores, evaluando la proporción de acuerdo mayor que el azar [34].
- **Coeficiente de correlación intraclase (ICC)**: Útil para datos numéricos continuos o intervalos, mide la consistencia de valoraciones individuales respecto a la varianza total [35].
- **Coeficiente de correlación de Spearman:** Mide la fuerza y dirección de la asociación entre dos variables ordinales, basándose en los rangos en lugar de los valores absolutos [36].

En este trabajo se emplean únicamente el alfa de Krippendorff y coeficiente de correlación de Spearman.

B.3.1 Alfa de Krippendorff

El alfa de Krippendorff (α) es una medida robusta de acuerdo interanotador que se adapta a distintos niveles de medición (nominal, ordinal, de intervalo o de razón) y permite valores faltantes y múltiples evaluadores. Se define como:

$$\alpha = 1 - \frac{D_o}{D_e}$$

donde D_o es la discrepancia observada, es decir, el grado medio de desacuerdo real entre los anotadores y D_e es la discrepancia esperada por azar, es decir, la discrepancia que se esperaría si los anotadores asignaran las puntuaciones aleatoriamente siguiendo la misma distribución.

La discrepancia observada D_o se define como:

$$D_o = \frac{1}{N} \sum_{i=1}^{n} \sum_{j=1}^{n} o_{ij} \cdot \delta(v_i, v_j)^2$$

donde o_{ij} es la proporción de veces que dos anotadores asignaron respectivamente las categorías v_i y v_j al mismo ítem y N es el número total de comparaciones realizadas.

La discrepancia esperada D_e se calcula como:

$$D_e = \sum_{i=1}^{n} \sum_{j=1}^{n} p_i \cdot p_j \cdot \delta(v_i, v_j)^2$$

donde p_i y p_j son las proporciones marginales de aparición de las categorías v_i y v_j.

Tanto D_o como D_e se calculan a partir de una función de distancia δ entre las categorías. En el caso de escalas ordinales (como la empleada en este estudio, del 1 al 10), se usa típicamente la distancia cuadrática:

$$\delta(a, b) = (a - b)^2$$

Finalmente, el valor de α se interpreta de la siguiente forma:

- α = 1: acuerdo perfecto
- α = 0: acuerdo igual al azar
- α < 0: desacuerdo sistemático

B.3.2 Coeficiente de correlación de Spearman

Como complemento, se utilizó el coeficiente de correlación de Spearman (ϱ) para estudiar el grado de similitud entre los rankings de puntuaciones asignados por pares de anotadores. Este coeficiente mide si dos evaluadores tienden a ordenar los modelos de forma similar, aunque usen escalas distintas, y se define como:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

donde d_i es la diferencia entre los rangos de un ítem i para dos anotadores y n es el número de ítems evaluados. Un valor de ϱ próximo a 1 indica una alta concordancia en el orden relativo asignado a los modelos mientras que valores cercanos a 0 indican ordenación no relacionada y negativos indican ordenaciones inversas.

Anexo C

Detalle de los modelos de lenguaje utilizados

C.1 GPT-4o Mini

El modelo GPT-4o Mini se empleó a través de ChatGPT, el cual está desarrollado por OpenAI y es un servicio de modelo de lenguaje alojado en la nube que proporciona acceso a variantes de la familia GPT (Generative Pre-Trained Transformer) mediante interfaz web o API de pago [40]. No se pudo hacer fine-tuning de ninguno de los modelos de OpenAI ya que para ello se necesita acceso a la API, que es de pago y excede los recursos presupuestarios disponibles para este trabajo. Por esta razón, se utilizó la versión gratuita de ChatGPT, accesible en https://chatgpt.com/, únicamente para inferencia.

No obstante, las pruebas de inferencia con ChatGPT se realizaron en marzo de 2025, fecha en que la versión gratuita aún no había sido actualizada a GPT-4o y continuaba ejecutando un modelo intermedio denominado GPT-4o Mini [41]. Las consultas siguieron los diseños de *prompt* del Anexo A para explorar la capacidad del modelo de producir requisitos funcionales y no funcionales a partir de descripciones de sistemas.

Respecto a su arquitectura, los modelos GPT están basado en una arquitectura Transformer de decodificador, entrenada en dos fases:

- Preentrenamiento autoregresivo con aprendizaje auto-supervisado (predicción del siguiente token).
- *Fine-tuning* instruccional, donde el modelo se entrena mediante instruction tuning y posteriormente se aplica aprendizaje por refuerzo con retroalimentación humana (RLHF) para alinear las salidas con preferencias humanas.

Internamente, mantiene miles de millones de parámetros distribuidos en múltiples capas de atención multi-cabeza, alimentadas por embeddings posicionales y normalizaciones residuales [42].

La elección de este modelo se debe a que ChatGPT es uno de los LLMs más conocidos y avanzados, además de que ha sido entrenado para generar salidas adaptadas al usuario.

C.2 DeepSeek-R1

DeepSeek Chat, desarrollado por DeepSeek AI, es un servicio de modelo de lenguaje que proporciona acceso gratuito a través de su web oficial https://chat.deepseek.com/ al modelo DeepSeek-R1 [44]. Aunque hay disponibles modelos de DeepSeek AI en Hugging Face [43], estos no fueron empleados para *fine-tuning* ya que al generar texto, el modelo genera razonamientos y explicaciones intermedias (“*chain-of-thought*”) que hace que se

consuman muchos más tokens por petición. Se utilizó DeepSeek Chat a través de su web únicamente para inferencia y las consultas siguieron los diseños de *prompt* del Anexo A.

El modelo DeepSeek-R1 [44] se basa en una versión destilada de LLaMA-8B (un Transformer de decodificador) y aplica técnicas de *knowledge distillatio*n para conservar la calidad de generación de un modelo de gran tamaño, reduciendo al mismo tiempo sus requisitos computacionales. Conserva capas de atención multi-cabeza, *feed-forward* y *positional encodings* propios de LLaMA, pero reduce el número total de tokens procesados gracias a la destilación.

C.3 Meta-Llama 3 Instruct

La serie de modelos Meta-Llama 3 Instruct son modelos open-source desarrollados por Meta AI, basados en la tercera generación de la familia LLaMA (Large Language Model Meta AI) y disponibles a través de la plataforma Hugging Face. Los modelos empleados en este trabajo fueron las versiones de 1.24 mil millones, 3.21 mil millones y 8.03 mil millones de parámetros, disponibles respectivamente en los repositorios:

- meta-llama/Meta-Llama-3.2-1B-Instruct [45]
- meta-llama/Meta-Llama-3.2-3B-Instruct [46]
- meta-llama/Meta-Llama-3.1-8B-Instruct [47]

Todos ellos están basados en arquitectura Transformer de decodificador con múltiples capas de atención multi-cabeza, normalización y capas *feed-forward* y añaden un paso de *instruction tuning* tras el preentrenamiento inicial para seguir instrucciones en lenguaje natural. Su inclusión en el estudio responde a su disponibilidad libre, su compatibilidad con entornos de entrenamiento locales y su adecuación a tareas de NLP centradas en extracción o generación de información estructurada, por lo que se emplearon tanto en la fase de *fine-tuning* como en inferencia.

C.4 Mistral 7B Instruct

El modelo mistralai/Mistral-7B-Instruct-v0.3 [48] es un modelo de lenguaje open-source de 7.25 mil millones de parámetros desarrollado por Mistral AI y disponible a través de la plataforma Hugging Face. Está basado en la arquitectura Transformer de decodificador y también se le aplica instruction tuning para optimizarlo en la comprensión de instrucciones complejas. Al igual que los modelos Meta-Llama 3 Instruct, este modelo se empleó tanto para inferencia como para fine-tuning.