Universidad Autónoma de Madrid

Escuela Politécnica Superior

24|25

Trabajo fin de grado

Aplicación de PLN y LLMs a textos no estructurados para generar imágenes

Antonio Van-Oers Luis

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Aplicación de PLN y LLMs a textos no estructurados para generar imágenes

Autor: Antonio Van-Oers Luis
Tutor: Alejandro Bellogin Kouki

mayo 2025

Antonio Van-Oers Luis
Aplicación de PLN y LLMs a textos no estructurados para generar imágenes

Antonio Van-Oers Luis
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi madre, quien lo ha dado todo para llevarme hasta donde estoy.

Per aspera ad astra

Séneca

Agradecimientos

En primer lugar, quiero expresar mi agradecimiento a mis compañeros por esas experiencias en prácticas, charlas en la cafetería y viajes inolvidables, que me han ayudado a crecer tanto en lo académico como en lo personal.

Quiero reconocer a todos los docentes que me han acompañado a lo largo de mi formación. Su dedicación y pasión por las materias me han ayudado a seguir ampliando mis conocimientos cada día.

Expreso también mi especial agradecimiento a Alejandro Bellogín. Por su dedicación como tutor durante el desarrollo de este trabajo, por su ayuda, su guía y sus mensajes, atentos siempre a la vanguardia de la Inteligencia Artificial.

Quiero dar las gracias a mis amigos por su apoyo constante en el proyecto y por ayudarme a desconectar cuando tocaba.

A mi madre, por todo el esfuerzo que hace para ayudarme a llegar lejos.

A mi padre, por estar ahí siempre que lo he necesitado.

A todos los que me han acompañado; habéis convertido el viaje en una experiencia increíble, sin importar el destino.

Resumen

Hoy en día, el campo de la Inteligencia Artificial avanza a pasos agigantados y cada vez más rápido. El Procesamiento de lenguaje natural es una de las disciplinas que ha pasado de utilizar técnicas clásicas de Aprendizaje Automático, a aplicar las capacidades analíticas de las redes neuronales y el Aprendizaje Profundo, hasta hacer uso de las nuevas arquitecturas de los grandes modelos de lenguaje (LLMs) actuales. Con todo ello, investigadores y desarrolladores disponemos de una gran cantidad de herramientas para superar los desafíos que se plantean en estas áreas.

Este proyecto tiene como objetivo principal explorar, hacer uso e implementar dichas herramientas, haciéndolas encajar de forma que se consiga analizar un texto no estructurado independientemente de su extensión. La aplicación elegida para este análisis es la generación de imágenes coherentes con el texto y con alto nivel de detalle.

Trata de superar, de forma transparente al usuario y mediante una aproximación al problema poco habitual, limitaciones de modelos similares como el máximo de tokens que pueden procesar o una alta demanda de memoria para ejecutarse. Por ello, se incluye como prioridad hacer un sistema ligero, con código abierto y ejecutable localmente con un hardware razonable a nivel de usuario.

Palabras clave

Procesamiento de Lenguaje Natural; PLN; Modelos de Lenguaje de Gran Tamaño; LLM; Textos no estructurados; Generación de imágenes; Arquitectura modular; Ingeniería de prompts

Abstract

Nowadays, the field of Artificial Intelligence is advancing by big leaps and increasingly faster. Natural Language Processing is one of the disciplines that has gone from using classical Machine Learning techniques, to applying the analytical capabilities of neural networks and Deep Learning, up to making use of the new architectures of current large language models (LLMs). With all this, researchers and developers now have a large number of tools available to overcome the challenges posed in these areas.

This project has as its main objective to explore, make use of, and implement these tools, making them fit together in such a way that it is possible to analyze an unstructured text regardless of its length. The chosen application for this analysis is the generation of images that are coherent with the text and have a high level of detail.

It seeks to overcome, in a way that is transparent to the user and through an uncommon approach to the problem, limitations of similar models such as the maximum number of tokens they can process or a high memory demand to run. Therefore, it is included as a priority to create a lightweight system, with open-source code and executable locally with reasonable user-level hardware.

Keywords

Natural Language Processing; NLP; Large Language Models; LLM; Unstructured text; Image generation; Modular architecture; Prompt engineering

Índice

Listas

Lista de códigos

Lista de figuras

Lista de tablas

1

Introducción

1.1. Motivación

Uno de los grandes retos actuales en el ámbito del Procesamiento del Lenguaje Natural (PLN) es el análisis e interpretación de textos de gran extensión [1]. Muchos de los modelos actuales, aunque muy potentes, están limitados por el número máximo de tokens que pueden procesar o por los requisitos de hardware para funcionar con soltura [2, 3]. Este trabajo surge de la necesidad de superar estas barreras sin depender de modelos cada vez más grandes, costosos o inaccesibles.

En lugar de seguir la tendencia predominante de ampliar la capacidad de un único modelo, se ha optado por una aproximación modular y descentralizada. Se plantea un sistema capaz de analizar textos no estructurados, independientemente de su extensión y de forma transparente para el usuario final. La tarea se divide entre módulos especializados que colaboran para obtener un resultado final coherente: la generación de imágenes que representen fielmente el contenido narrativo del texto.

Aplicaciones

La generación de imágenes a partir de texto tiene aplicaciones que van mucho más allá del ámbito artístico. En el entorno educativo, por ejemplo, podría utilizarse para ilustrar libros de texto en contextos donde no se dispone de recursos para imágenes originales, ayudando a estudiantes con estilos de aprendizaje más visuales. También resulta útil en el campo editorial, facilitando la creación de portadas o ilustraciones para escritores independientes que no pueden recurrir a artistas profesionales.

Además, el sistema diseñado está enfocado a la interpretación de historias, identificando elementos como escenarios, personajes o emociones. Esto abre la puerta a usos alternativos, como la generación visual asistida para personas con dificultades del habla, el diseño de campañas visuales publicitarias a partir de descripciones detalladas, o incluso como herramienta guía para productores de cine, dando una idea inicial acerca de cómo se ven los personajes o escenarios de forma accesible. El diseño modular y escalable de la solución permite su adaptación a nuevas áreas de aplicación conforme evolucione el campo.

1.2. Objetivos propuestos

El principal objetivo de este proyecto es diseñar e implementar un sistema capaz de analizar textos no estructurados de gran longitud mediante una arquitectura modular basada en modelos especializados de PLN e Inteligencia Artificial. El análisis realizado debe permitir la generación de imágenes coherentes con el contenido narrativo, sin que el usuario tenga que preocuparse por las limitaciones de los modelos subyacentes.

Se prioriza, además, que la solución sea ligera, de código abierto y ejecutable en entornos locales sin necesidad de hardware avanzado, promoviendo así su accesibilidad. El sistema debe ser escalable y permitir la incorporación futura de nuevos modelos sin comprometer la funcionalidad existente. De este modo, se pretende no solo ofrecer una solución a un problema técnico actual, sino también sentar las bases para herramientas más accesibles, adaptables y resilientes en el ámbito del PLN.

1.3. Estructura de la memoria

Este proyecto se organiza en cinco capítulos, cada uno centrado en un aspecto fundamental del desarrollo e investigación:

- **Capítulo 1. Introducción:** Presenta la motivación del proyecto y el problema que se pretende abordar: la dificultad de procesar textos de gran extensión en el ámbito del Procesamiento del Lenguaje Natural. También se exponen las aplicaciones potenciales y los objetivos perseguidos.
- **Capítulo 2. Estado del arte:** Recoge los principales avances en técnicas de PLN, generación de imágenes y uso de modelos especializados, analizando sus limitaciones y oportunidades. Se revisan modelos existentes y herramientas utilizadas como base para el diseño del sistema.
- **Capítulo 3. Diseño e implementación:** Describe en detalle la arquitectura modular desarrollada, los modelos empleados en cada subtarea, la integración entre componentes y las decisiones técnicas que han guiado el desarrollo de la herramienta.
- **Capítulo 4. Pruebas:** Expone la metodología empleada para validar el sistema, así como los resultados obtenidos. Se analizan tanto el rendimiento como las limitaciones detectadas, evaluando la coherencia y calidad de las imágenes generadas.
- **Capítulo 5. Conclusiones:** Resume las aportaciones principales del proyecto, discute su impacto y plantea líneas de mejora y posibles extensiones, especialmente en relación con la eficiencia, escalabilidad y evolución futura del campo.

2

Estado del Arte

2.1. Procesamiento de lenguaje natural

El procesamiento de lenguaje natural (PLN) constituye una de las áreas más relevantes dentro de la inteligencia artificial y la informática actual, ya que estudia los marcos teóricos y metodológicos que permiten establecer una comunicación efectiva entre los seres humanos y los sistemas computacionales. Su objetivo principal es dotar a las máquinas de la capacidad de interpretar, procesar y generar lenguaje natural de forma que puedan abordar tareas como la traducción automática, la clasificación de texto o el análisis de sentimiento, entre otras [4]. Este campo abarca múltiples etapas complejas, algunas siendo la segmentación de palabras, el etiquetado morfosintáctico, el análisis sintáctico, la lematización o el reconocimiento de entidades, todas ellas fundamentales para simular una comprensión lingüística similar a la humana en los sistemas artificiales.

Algunas aplicaciones de esta tecnología se dan en la vida cotidiana, como asistentes virtuales (Siri, Alexa), herramientas de traducción automática, motores de recomendación, y sistemas de análisis de sentimientos, transformando radicalmente la manera en la que los usuarios interactúan con la tecnología digital [5].

2.1.1. Procedimientos PLN

En este proyecto se aplican algunas de las diversas técnicas de preprocesamiento para optimizar la calidad del texto antes de su análisis por los modelos de lenguaje.

Preprocesamiento

Tal como demuestran en [6], esta etapa sigue siendo relevante en cuanto al rendimiento de tareas PLN, incluso si se aplican modelos como redes neuronales profundas o LLMs. Un ejemplo de estas técnicas de preprocesamiento es la limpieza del ruido, que consiste en eliminar etiquetas HTML, caracteres especiales y cadenas no deseadas (URLs, menciones, hashtags, Unicode residual) mediante expresiones regulares. Otra técnica muy utilizada, es la eliminación de *stop words*, que implica descar-

tar términos de alta frecuencia —preposiciones, artículos, conjunciones y pronombres— considerados poco informativos para la minería de texto, lo que reduce la dimensión del vocabulario y mejora la coherencia de los embeddings [6].

Estas operaciones de preprocesado facilitan la ingesta de datos y aumentan la precisión en procesos posteriores de análisis del texto, en especial al tratarse de uno no estructurado.

Extracción de información

En este trabajo se emplea un enfoque de pregunta-respuesta (del inglés, *question-answering* [7]), mediante un LLM, aprovechando las fortalezas de estos modelos para resolver preguntas de opción única y múltiple, extraer datos numéricos, localizar respuestas en documentos y adaptarse a distintos formatos de consulta. Su rendimiento depende en gran medida de la relevancia de los fragmentos recuperados o analizados [7].

Detección de emociones

Otra técnica de PLN aplicada en este trabajo es la detección de emociones. Esta se realiza con modelos especializados basados en la arquitectura BERT, los cuales, tras un preentrenamiento mediante *Masked Language Modeling y Next Sentence Prediction*, y una fase de *fine-tuning* para clasificación, aprovechan su capacidad bidireccional para capturar relaciones contextuales y mantener dependencias a largo plazo en el texto [8].

Prompting

La ingeniería de prompts ha emergido como una disciplina fundamental para orientar a los modelos generativos hacia la producción de resultados coherentes y de alta calidad. Técnicas pioneras como el *few-shot prompting* y el *chain-of-thought prompting* demostraron que, ejemplos cuidadosamente seleccionados y cadenas de razonamiento, pueden mejorar drásticamente el rendimiento. De manera paralela, se han establecido guías de diseño que subrayan la importancia de los modificadores de estilo en tareas de texto a imagen, y estudios de usuario han validado su impacto en la calidad visual obtenida [9].

El *zero-shot prompting*, introducido en [10], consiste en emplear instrucciones en lenguaje natural para que un modelo de lenguaje entrenado con enfoque de instrucciones (*instruction-tuned*) ejecute una tarea específica sin requerir ningún ejemplo de entrenamiento.

Tal como se destaca en [11], la eficacia de un prompt reside en su claridad, la inclusión de contexto relevante y la definición precisa de la tarea a realizar, facilitando al modelo la generación de respuestas más coherentes y acordes con el caso de uso.

2.1.2. Modelos de lenguaje

Grandes modelos de lenguaje (LLMs)

Los *Large Language Models* (LLMs) son modelos de aprendizaje profundo entrenados sobre enormes cantidades de texto con el objetivo de predecir la probabilidad de aparición de una palabra (o token) dado un contexto previo. A través de esta tarea de modelado del lenguaje, los LLMs aprenden representaciones estadísticas complejas que les permiten generar texto coherente, responder preguntas, traducir entre idiomas, resumir documentos o realizar razonamiento básico sobre el contenido textual.

Transformer LLMs

Los modelos de lenguaje actuales se fundamentan en la arquitectura Transformer. Concebida originalmente para tareas de traducción automática, fue rápidamente adoptada por las grandes LLMs por su capacidad de procesar secuencias completas de entrada mediante mecanismos de atención que capturan dependencias globales entre tokens. En esencia, un Transformer combina bloques de atención escalada por producto escalar con redes *feed-forward posición-wise*, intercaladas con conexiones residuales y normalización de capas, lo que configura un sistema de procesamiento secuencial paralelo altamente efectivo. Gracias a su diseño, los LLMs pueden escalar una enorme cantidad de parámetros, dando lugar a habilidades de razonamiento y generación muy avanzadas que han revolucionado tanto el entendimiento como la síntesis del lenguaje humano. Esto tiene aplicaciones en dominios tan diversos como el análisis de sentimientos, la generación de resúmenes y la programación asistida [12].

A pesar de sus éxitos, la complejidad cuadrática de la atención impone límites a la longitud práctica del contexto y exige abundantes recursos de memoria y cómputo. Por ello, la investigación reciente se ha volcado en extender la ventana de contexto —mediante técnicas de atención eficiente, memorias externas y codificaciones posicionales extrapolativas— y en optimizar el modelo para manejar secuencias de hasta 128 000 tokens sin fragmentación, manteniendo la calidad de la generación en textos extensos [12].

El éxito de los Transformers se fundamenta en la sinergia de varios componentes innovadores que, en conjunto, han redefinido el procesamiento de secuencias [13]. En primer lugar, el mecanismo de atención asigna dinámicamente pesos de relevancia a cada par de consulta (Q) y clave-valor (K,V), lo que permite al modelo dirigir su “mirada” hacia los fragmentos de texto más informativos. Sobre esta base, la atención lleva a cabo de forma simultánea múltiples proyecciones lineales de Q, K y V, de modo que el modelo puede capturar patrones semánticos desde diversos subespacios de representación. Para conservar el orden de las palabras en una arquitectura sin recurrencia, las codificaciones posicionales introducen señales numéricas—ya sean senoidales o relativas—que enriquecen los embeddings con información de posición. Acto seguido, las redes *feed-forward* aplicadas posición por posición apor-

tan la profundidad y la no linealidad necesarias, procesando cada token de manera independiente tras la fase de atención. Finalmente, las conexiones residuales, acompañadas de la correspondiente normalización de capas, aseguran la estabilidad del entrenamiento y facilitan la propagación del gradiente en redes de gran profundidad. Gracias a la conjunción de estos elementos, los Transformers logran paralelizar completamente el cómputo sobre secuencias, capturar relaciones a largo plazo y mantener su robustez frente a variaciones en la longitud de entrada o desplazamientos posicionales [13].

La familia de modelos Llama3.1

Desde su aparición, los modelos Llama han seguido de cerca la evolución de la arquitectura Transformer. En su versión inicial, Llama 1 (febrero 2023) adoptó el esquema de atención escalada por producto escalar y las capas de *feed-forward posición-wise*, integrando mejoras como la pre-normalización y RMSNorm para estabilizar el entrenamiento. Con el lanzamiento de Llama 2 (julio 2023), se amplió la ventana de contexto y se introdujo el mecanismo de *Grouped-Query Attention*, que optimiza la eficiencia de la atención en parámetros moderados. La tercera generación elevó aún más esta capacidad al incorporar el tokenizador TikToken y extender el límite de contexto a 8 192 tokens, permitiendo procesar textos más extensos sin fragmentación. Estas innovaciones, junto con las conexiones residuales que garantizan la paralelización y robustez de los Transformers [14], sentaron las bases para la creación de Llama 3.1.

En julio de 2024, Llama 3.1 emergió como la versión más ambiciosa de la familia [15], capaz de manejar contextos de hasta 128 000 tokens y de operar en ocho idiomas adicionales gracias a un corpus de preentrenamiento de más de 15 billones de tokens extraídos de la web, repositorios de código, libros y conversaciones estructuradas [16, 17]. Esta diversidad de datos multimodales no solo optimiza su rendimiento en tareas de generación de texto y código, sino que también potencia su adaptabilidad a dominios especializados sin necesidad de un *fine-tuning* extensivo. Además, la familia de modelos Llama 3.1 demuestra una notable competencia en generación de texto, traducción automática, resumen de documentos y sistemas de pregunta-respuesta, convirtiéndose en una herramienta versátil para aplicaciones como chatbots, creación de contenido, comunicación multilingüe, recuperación de información y otras soluciones basadas en procesamiento de lenguaje natural.

Llama3.1-8B

En este proyecto se priorizó una implementación completamente open source, ligera y capaz de ejecutarse en un único dispositivo, y Llama 3.1-8B cumple estos requisitos [18]. Esta variante de 8 000 millones de parámetros es la más pequeña y eficiente de la serie, diseñada para entornos con recursos limitados sin renunciar a funcionalidades críticas como el entrenamiento orientado al uso de herramientas. Con una ventana de contexto de 128 000 tokens, Llama 3.1-8B procesa pasajes extensos en una sola pasada, lo que la hace idónea para tareas que requieren manejar grandes volúmenes

de texto en una sola sesión, como generación de contenido, resumen de documentos o comprensión de argumentos complejos. Además, su bajo coste computacional y su capacidad de adaptación mediante *fine-tuning* la convierten en una solución versátil para startups y pymes que deben integrar IA en sus flujos de trabajo con presupuestos ajustados. Llama 3.1-8B sobresale en escenarios de automatización de atención al cliente, análisis de sentimiento y otras aplicaciones específicas de procesamiento de lenguaje natural, ofreciendo un equilibrio óptimo entre rendimiento fiable y eficiencia operativa, sin necesidad de infraestructuras de gran escala [19].

Deepseek R1

El 10 de enero de 2025 salió a la luz un nuevo LLM desarrollado por una empresa china: Deepseek R1. Su aparición marcó un hito en la industria al combinar eficiencia, bajo costo y capacidades avanzadas de razonamiento, compitiendo directamente con modelos del estado del arte como GPT-4o. Esto lo consiguió gracias a la introducción de varias soluciones que no habían llegado a tener éxito en anteriores modelos y sus entrenamientos.

Para empezar, su arquitectura de expertos permite que solo se activen un 18 % de sus parámetros totales por consulta, optimizando recursos. Además, en su entrenamiento, se optó por emplear el aprendizaje por refuerzo pero con optimización de políticas relativas a grupo (GRPO), permitiendo generar cadenas de pensamiento explícitas muy precisas en sus respuestas destacando en tareas de razonamiento complejo [20].

Por otra parte, R1 incluyó versiones destiladas de sus propios modelos, un proceso de destilación de conocimiento por el cual se transfiere la capacidad de razonamiento de un “profesor” de gran tamaño (Deepseek R1) a un “alumno” más pequeño y eficiente. Mediante *fine-tuning* supervisado (SFT) sobre muestras curadas por Deepseek R1, se obtuvieron variantes como Deepseek-R1-Distill-Qwen y Deepseek-R1-Distill-Llama, que conservan gran parte de las habilidades de razonamiento del modelo original. Logran así reducir drásticamente los requisitos de cómputo y la latencia de inferencia, todo ello sin necesidad de recurrir a etapas adicionales de refuerzo [20, 21].

2.1.3. Evolución de PLN gracias a los grandes modelos de lenguaje

Además de su expansión práctica, la investigación en PLN ha crecido en profundidad. Los últimos desarrollos se han centrado en algoritmos de aprendizaje profundo, técnicas de atención y modelos generativos, capaces de realizar tareas complejas como el reconocimiento de entidades, clasificación textual, traducción automática o generación de lenguaje natural. Estos avances han alcanzado niveles de rendimiento comparables al humano en tareas específicas, y han abierto nuevas posibilidades en la computación conversacional y en el procesamiento contextual del lenguaje [5].

La irrupción de las herramientas de IA generativa como LLMs en el campo del PLN ha transforma-

do radicalmente las capacidades de los sistemas lingüísticos al proporcionar respuestas inmediatas, coherentes y con alta semejanza al lenguaje humano. Tal como se analiza en [22], plataformas como ChatGPT, Perplexity AI, YouChat y Bard, entre otras, se distinguen por su facilidad de uso y por la velocidad con que generan contenido en múltiples modalidades —desde texto hasta audio y visuales—, lo que ha democratizado el acceso a modelos avanzados sin requerir conocimientos de programación profunda. No obstante, esta conveniencia conlleva desafíos significativos: la dependencia de grandes volúmenes de datos preentrenados puede inducir sesgos, y los elevados costes computacionales asociados a los procesos de inferencia y afinado de modelos limitan su despliegue en entornos con recursos restringidos [22].

Sin embargo, en este proyecto se ha implementado un sistema el cual sacrifica precisión para hacer posible el uso de estas herramientas en un entorno local, con un solo dispositivo y haciendo uso de modelos totalmente de código abierto. A pesar de la pérdida de precisión y del mayor consumo computacional, emplear estos LLM para tareas de PLN ofrece ventajas sustanciales gracias a su arquitectura Transformer, explicada en la Sección 2.1.2. Estos modelos se preentrenan con grandes cantidades de texto extraído de Internet, lo que les permite generar respuestas coherentes a partir de indicaciones mínimas. La capacidad de capturar dependencias a largo plazo—fundamental para tareas como traducción automática, resumen de texto o análisis de sentimiento—se ve potenciada por la auto-atención, que supera las limitaciones de memoria de los modelos recurrentes y mejora la calidad de la generación. Además, la naturaleza paralela del Transformer acelera drásticamente tanto el entrenamiento como la inferencia, permitiendo procesar secuencias más extensas sin incurrir en los cuellos de botella que caracterizaban a redes neuronales tipo RNN y LSTM [23]. Frente a los enfoques previos, en los que la dependencia secuencial de las redes recurrentes dificultaba la captura de contexto global y alargaba los tiempos de cómputo, los Transformers optimizan la gestión de datos y reducen la complejidad temporal, lo que se traduce en mejoras significativas en precisión y velocidad en tareas como análisis de sentimiento o traducción [23].

2.1.4. Modelos especializados

SamLowe/roberta-base-go_emotions

El modelo SamLowe/roberta-base-go_emotions es una adaptación de RoBERTa preentrenado en tareas de clasificación multi-etiqueta de emociones, específicamente ajustado sobre el conjunto de datos GoEmotions, que contiene comentarios de Reddit codificados con 28 categorías emocionales [24]. Gracias a su robustez y capacidad para discriminar múltiples estados emocionales simultáneamente, este modelo se ha adoptado en diversos estudios de procesamiento de lenguaje natural, como para categorizar con alta concordancia constructos psicológicos complejos en reflexiones humanas [25] o para medir niveles de confusión en transcripciones de audio de sesiones de programación [26].

distilbert/distilbert-base-uncased-finetuned-sst-2-english

El modelo distilbert/distilbert-base-uncased-finetuned-sst-2-english es una versión destilada de BERT que ha sido ajustada específicamente sobre el corpus Stanford Sentiment Treebank (SST-2) para clasificación binaria de sentimiento, alcanzando un 91,3 % de exactitud en el conjunto de validación [27]. Gracias a su arquitectura más ligera, mantiene la mayoría de las capacidades de entendimiento lingüístico de BERT con menos de la mitad de parámetros y hasta el doble de velocidad, resultando idóneo para entornos con recursos limitados o despliegues en la nube [28].

Su uso principal recae en tareas de clasificación de texto, como análisis de sentimiento en redes sociales, donde su habilidad para discriminar entre polaridades positiva y negativa aporta resultados fiables y rápidos [27]. Investigaciones posteriores han explorado estrategias de fine-tuning sobre este modelo, demostrando que la elección de hiperparámetros —tasa de aprendizaje, tamaño de lotes y número de épocas— influye de forma no lineal en métricas como F1-score y pérdida, lo que subraya la necesidad de un ajuste cuidadoso para maximizar el rendimiento en cada caso de uso [29].

paraphrase-mpnet-base-v2

El modelo paraphrase-mpnet-base-v2 es una variante de Sentence-BERT diseñada para generar *embeddings* semánticas de 768 dimensiones a partir de oraciones y párrafos, facilitando su uso en tareas de búsqueda semántica, agrupamiento y detección de paráfrasis [30]. Basado en la arquitectura MPNet, que combina las ventajas de las redes de atención y el preentrenamiento de tipo móvil, este modelo fue afinado usando una pérdida de similitud de coseno sobre un gran corpus de pares de oraciones equivalentes, lo que le permite capturar relaciones conceptuales profundas y reducir drásticamente el coste de inferencia frente a un encoder cruzado tradicional [31].

Su eficacia se refleja en evaluaciones comparativas en las que supera a otros modelos SBERT populares —como bert-base-nli-mean-tokens o all-roberta-large-v1— al alcanzar una precisión de hasta 0,96 en conjuntos de prueba de paráfrasis (tamaño de lote 32) y en aplicaciones prácticas de extensión de taxonomías ESG, donde demostró ofrecer las mejores métricas para identificar términos conceptualmente similares [32]. Gracias a este equilibrio entre velocidad, eficiencia y calidad de incrustaciones, paraphrase-mpnet-base-v2 se ha convertido en una opción de referencia para una amplia variedad de flujos de trabajo de PLN que requieren comprensión y comparación rápida de fragmentos textuales.

2.1.5. Métodos de similitud entre textos

Con la necesidad de detectar entidades similares en el análisis del texto, se realiza una investigación de los métodos estado del arte que miden la similitud léxico-semántica de un texto.

Redes neuronales siamesas

Las redes neuronales siamesas constan de dos subredes gemelas f_θ que comparten parámetros y procesan por separado dos textos x_1, x_2, generando vectores $\mathbf{h}_1 = f_\theta(x_1)$ y $\mathbf{h}_2 = f_\theta(x_2)$.

Para estimar la similitud se calcula luego una distancia $d = \|\mathbf{h}_1 - \mathbf{h}_2\|_2$, y se entrena el sistema con una pérdida contrastiva del tipo:

$$\mathcal{L} = y\, d^2 + (1 - y)\ \mathrm{máx}(0, m - d)^2,$$

donde $y \in \{0, 1\}$ indica si los textos son similares y m es un margen positivo. De este modo, la red aprende una métrica de similitud adaptada al dominio del texto, maximizando la proximidad de vectores para pares semánticos y separando naturalmente aquellos con significado distinto [33–35].

Embeddings

Los embeddings de texto son representaciones densas en un espacio vectorial continuo donde la proximidad geométrica refleja la semejanza semántica entre unidades lingüísticas (palabras, oraciones o párrafos). Modelos preentrenados como BERT, MPNet o SBERT codifican textos en vectores de alta dimensión que capturan relaciones contextuales profundas.

Más en detalle, los embeddings de texto proyectan tokens discretos en vectores densos $\mathbf{e}_i \in \mathbb{R}^d$ a través de una matriz de proyección $E \in \mathbb{R}^{|V| \times d}$. Un texto $t = (w_1, \ldots, w_n)$ puede representarse como $\mathbf{h} = \frac{1}{n} \sum_{i=1}^{n} E_{w_i}$, o bien tomando el token especial `[CLS]` en modelos Transformer. Este mapeo continuo captura relaciones semánticas —por ejemplo, $\mathbf{e}(\text{rey}) - \mathbf{e}(\text{hombre}) + \mathbf{e}(\text{mujer}) \approx \mathbf{e}(\text{reina})$— y facilita la alimentación de un LLM, que opera sobre vectores reales en lugar de secuencias de caracteres o índices discretos. Matemáticamente, esta transformación hace diferenciable la similitud semántica y permite optimizaciones basadas en gradiente en el espacio vectorial.

La evaluación más reciente en benchmarks de embeddings demuestra que su versatilidad y capacidad para adaptarse a tareas variadas de PLN los convierten en una herramienta esencial para el cálculo de similitud entre textos [36, 37].

Similitud del coseno

La similitud del coseno mide el coseno del ángulo entre dos vectores normalizados, centrándose únicamente en su orientación y siendo invariante a la magnitud.

La similitud del coseno entre dos vectores $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ se define como $\cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\,\|\mathbf{v}\|}$. A diferencia de la distancia Euclídea ($\|\mathbf{u} - \mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2 - 2\,\mathbf{u} \cdot \mathbf{v}$,) que depende de la magnitud de los vectores, la similitud de coseno es invariante al escalado ($\cos(\alpha\mathbf{u}, \beta\mathbf{v}) = \cos(\mathbf{u}, \mathbf{v})$). Esto la hace especialmente adecuada para comparar embeddings de texto, donde el ángulo entre vectores captura

mejor la correlación semántica y reduce el efecto de diferencias de norma o longitud de texto. De hecho, es la medida más empleada para comparar embeddings de texto debido a su bajo coste computacional y su eficacia en recuperación de información y búsqueda semántica, donde explica más del 90 % de la variación observada en tareas de emparejamiento de oraciones.

Recientes mejoras basadas en transformaciones de la fórmula original han aumentado su robustez ante ruido y diferencias de escala [38, 39].

2.2. Modelos de generación de imágenes

2.2.1. Diffusion

Los modelos de difusión basados en scores se han consolidado como una de las técnicas más efectivas para la generación de datos en dominios tan diversos como imagen, audio y vídeo. Su principio fundamental consiste en definir un proceso de difusión que corrompe progresivamente la información original mediante la adición de ruido gaussiano, y a continuación invertir dicho proceso a través de una red neuronal entrenada para estimar la versión limpia a partir de una muestra ruidosa. De este modo, basta iniciar la generación desde ruido puro y aplicar de manera determinista sucesivos pasos de denoising hasta reconstruir una muestra coherente con la distribución de entrenamiento [40].

A pesar de estos éxitos, escalar los modelos de difusión a resoluciones elevadas plantea retos de memoria y cómputo. Para paliar estas limitaciones, la literatura ha explorado variantes como la difusión en espacios latentes (Latent Diffusion Models) [41], la cascada de super-resolución [42] o las arquitecturas de expertos en denoising [43], cada una introduciendo capas adicionales de complejidad para mantener la calidad en imágenes de alta resolución.

Funcionamiento

En esencia, un modelo de difusión aprende a revertir un proceso destructivo en el que los datos originales x se transforman paso a paso en ruido siguiendo:

$$z_t = \alpha_t x + \sigma_t \epsilon, \quad \epsilon \sim \mathcal{N}(0, I),$$

donde (α_t, σ_t) conforman el *noise-schedule* (la forma en la que los coeficientes varían a lo largo del tiempo). La red neuronal f_θ se entrena para predecir la versión limpia: $\hat{x}_t = f_\theta(z_t; t)$.

Durante la generación, se parte de ruido puro z_1 y, de forma iterativa, se alterna la estimación del estado limpio con la propagación del proceso directo a un nivel de ruido inferior $r < t$, es decir:

$$z_r = \alpha_r \hat{x}_t + \sigma_r \hat{\epsilon}, \quad \hat{\epsilon} = \frac{z_t - \alpha_t \hat{x}_t}{\sigma_t},$$

hasta recuperar la muestra sin corrupción [44].

Para acelerar este muestreo inverso sin recorrer toda la secuencia de ruido se emplea DDIM, *Denoising Diffusion Implicit Models* una técnica que reduce significativamente el número de pasos manteniendo la calidad de las imágenes generadas [19]. Además, formulando el proceso en términos de ecuaciones diferenciales estocásticas, es posible ajustar el grado de aleatoriedad en la inferencia y optimizar el equilibrio entre velocidad y diversidad de resultados [45].

Combinación con Flow matching

El flow matching redefine el proceso de síntesis como un transporte óptimo entre la distribución de datos y la normal estándar a través de ecuaciones diferenciales ordinarias, generando trayectorias casi rectas en el espacio latente [46]. Frente a los modelos de difusión puros, donde la curva del muestreo impone pasos pequeños y costosos, el flow matching permite dar saltos más amplios sin degradar la calidad, acelerando tanto el entrenamiento como la inferencia. Al combinarse con la difusión —que aporta fidelidad visual y estabilidad mediante un denoising multietapa—, se obtiene un enfoque híbrido capaz de producir imágenes de mayor resolución y detalle, manteniendo velocidades de muestreo significativamente superiores a las de los modelos basados únicamente en difusión [46, 47].

2.2.2. FLUX.1

FLUX.1 es una familia de modelos de síntesis de imágenes a partir de texto desarrollada por Black Forest Labs, diseñada para ofrecer la mejor calidad visual, fidelidad al prompt y diversidad de estilos en distintos escenarios de uso [48].

Construidos sobre una arquitectura híbrida que combina bloques de difusión multimodal con transformadores de atención paralela y refinados mediante técnicas de flow matching —explicadas en la Sección 2.2.1—, estos modelos escalados a 12 000 millones de parámetros incorporan embeddings posicionales rotatorios y atención local-global para maximizar tanto el detalle como la eficiencia de hardware. La suite se presenta en tres variantes: FLUX.1 [pro], el buque insignia accesible vía API y plataformas como Replicate y fal.ai, orientado a aplicaciones de alto rendimiento con soluciones empresariales dedicadas; FLUX.1 [dev], un modelo de código abierto y pesos disponibles en Hugging Face, destilado de la versión pro para uso no comercial con una eficiencia superior en el mismo tamaño; y FLUX.1 [schnell], el modelo más rápido de la familia, liberado bajo licencia Apache 2.0, optimizado para desarrollo local y uso personal sin sacrificar diversidad de salidas ni calidad de imagen [49].

La arquitectura de FLUX.1, combina de forma estrecha los flujos de información de imagen y texto antes de refinar la representación visual en bloques. En primer lugar, tanto los embeddings de la imagen como los del texto (provenientes de CLIP y de un encoder T5) se proyectan mediante capas lineales y MLP, y se mezclan junto a las señales de timestep y guidance. A continuación, ese vector

combinado atraviesa una serie de DoubleStream Blocks (N pasos), en los que módulos de atención paralela, normalización y capas feed-forward aplican operaciones conjuntas sobre ambas corrientes, además de inyectar embeddings posicionales rotatorios y módulos de modulación que ajustan escala y desplazamiento según el estado interno del modelo. Tras estos pasos de procesamiento multimodal, las características resultantes se concatenan y pasan por los SingleStream Blocks (M pasos), que actúan únicamente sobre el flujo de imagen, perfeccionando el detalle visual mediante subcapas de atención, QK-Norm y redes feed-forward posición-wise, todo ello con conexiones residuales y normalización de capas para asegurar estabilidad. Finalmente, un bloque de salida ("LastLayer") proyecta la representación refinada de vuelta al espacio de píxeles, generando la imagen sintetizada con alta fidelidad al texto de entrada. Este diseño híbrido de bloques dobles y simples permite acelerar la inferencia, capturar dependencias globales y mantener un flujo de contexto largo sin fragmentar la información [49].

Como se menciona en [50], este modelo es, además de código abierto, el más efectivo en coste-rendimiento, incluso comparándolo con otros modelos privados del estado del arte como DALL-E 3 o Stable Diffusion v6.

2.2.3. Cuantización

La cuantización de grandes modelos se ha usado para permitir desplegar grandes modelos en entornos con recursos limitados, habilitando a usuarios y desarrolladores con un hardware menos potente la posibilidad de usar dichos modelos sin sacrificar en sobremanera su rendimiento y la calidad de los resultados.

Este proceso de cuantización se basa en reducir la precisión de los pesos de las redes neuronales de los modelos, habitualmente representados en números float de 32 bits, y que gracias al proceso de cuantización llegan a reducirse a enteros de 16, 8 bits o incluso menores aún.

Si bien aplicable a LLMs, en este proyecto nos centramos en la cuantización de modelos de difusión especializados en pasar de texto a imagen, donde surgen retos específicos ligados al muestreo secuencial a lo largo de múltiples pasos de inferencia. Las variantes de difusión posterior al entrenamiento suelen adoptar cuantización uniforme con calibración de parámetros mediante redondeo adaptativo o inicialización min–max, ajustando los factores de escala tras recopilar estadísticas de activación en datos de calibración representativos [51].

En el caso de FLUX.1, la cuantización ha demostrado ser igualmente efectiva, tal como se detalla en [52]. Aplicando una cuantización post-training a 1,58 bits sobre el 99,5 por ciento de sus parámetros —sin necesidad de datos de imagen adicionales ni afinamiento específico— se consigue reducir drásticamente el tamaño de los pesos de las capas lineales de los bloques FluxTransformer y FluxSingleTransformer. Esta estrategia, complementada con técnicas de inferencia optimizada, apenas degrada

la calidad visual, mientras que el almacenamiento del modelo y el consumo de memoria de GPU se reducen de manera sustancial. Además, en hardware de despliegue más modesto, la latencia de inferencia mejora notablemente, lo que confirma que la cuantización en FLUX.1 permite ejecutar modelos de difusión de última generación en entornos con recursos limitados sin comprometer prácticamente su rendimiento [52, 53].

2.3. Hugging Face

Hugging Face se ha consolidado como el registro de modelos preentrenados (PTM registry) de referencia para la comunidad de aprendizaje profundo, situándose al nivel de ecosistemas establecidos como NPM o PyPI [54]. Al centralizar pesos, configuraciones y documentación en un único catálogo colaborativo, acelera el desarrollo y fomenta la reutilización de modelos, reduciendo drásticamente los costes y tiempos que implica entrenar redes desde cero. Un estudio reciente ha cuantificado que Hugging Face presenta una tasa de renovación de modelos significativamente superior a la de los registros de software tradicionales, y que la calidad de la documentación correlaciona fuertemente con la adopción de los PTM, lo que subraya la importancia de mantener "model cards" completas y actualizadas para impulsar la confianza y la difusión de las soluciones de IA [54].

El Hugging Face Hub funciona como plataforma social basada en Git, diseñada específicamente para el ciclo de vida de proyectos de ML [55]. En ella conviven repositorios de tres tipos —modelos, conjuntos de datos y "spaces" (aplicaciones demo)—, cada uno enriquecido con "cards" que describen parámetros de entrenamiento, métricas de evaluación, licencias y posibles sesgos. A pesar de su robustez para la colaboración, la separación frecuente entre los pesos alojados en Hugging Face y el código fuente en GitHub complica la trazabilidad y ha motivado estudios sobre la seguridad de estos modelos, pues sin un vínculo claro resulta difícil evaluar vulnerabilidades a gran escala [56].

En conjunto, Hugging Face ofrece una comunidad [57] y entornos únicos para compartir y versionar artefactos de IA, al mismo tiempo que plantea retos de transparencia y seguridad que resultarían críticos de abordar en nuestro proyecto.

3

Diseño e Implementación

3.1. Estructura

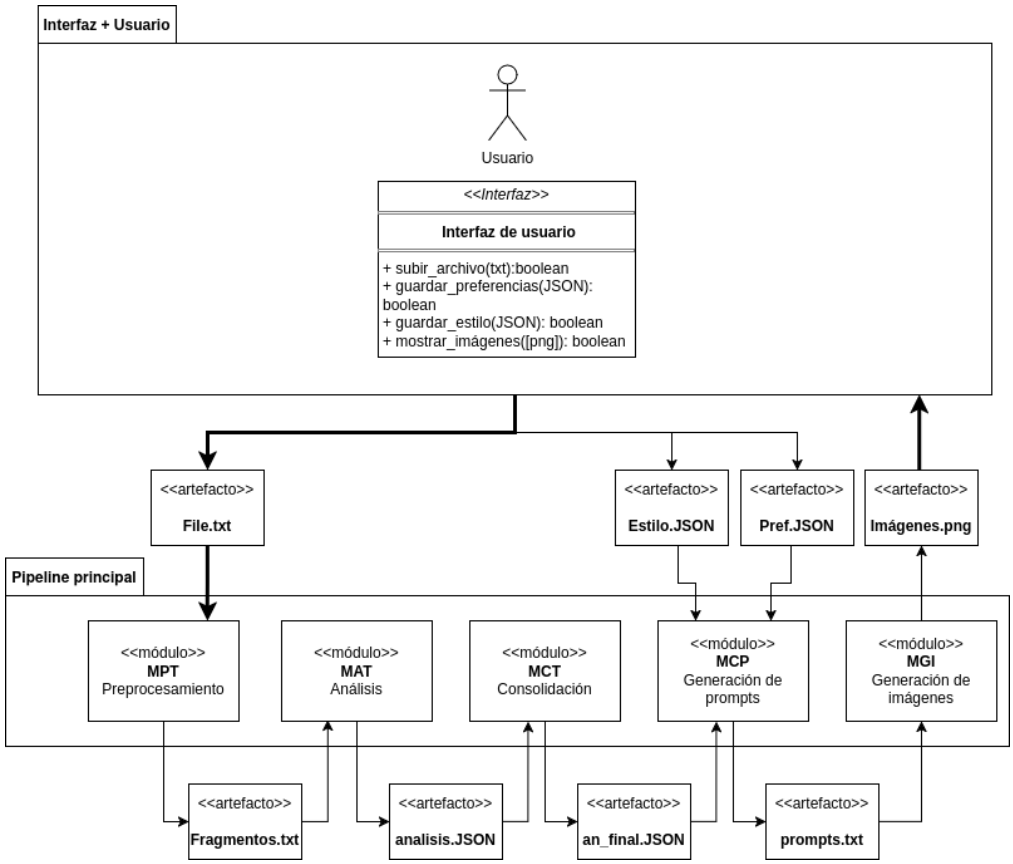


Figura 3.1: Arquitectura del proyecto

En la Figura 3.1 se presenta de manera sintética la arquitectura global de la aplicación, mostrando la interacción entre el usuario, los artefactos de entrada/salida y los cinco módulos principales del pipeline. Este diagrama sirve como guía para comprender cómo fluye la información desde la carga del texto hasta la generación de imágenes.

3.1.1. Módulos que conforman la aplicación

Con el propósito de simplificar y facilitar el desarrollo, la aplicación se divide en cinco módulos independientes pero intercomunicados. A nivel de diseño, cada uno realiza una función clave y expone unas entradas y salidas bien definidas:

Módulo de preprocesamiento de texto (MPT) Aplica técnicas clásicas de limpieza y segmentación para preparar el texto para su análisis.

- **Entrada:** Texto raw (formato .txt).
- **Salida:** Texto limpio, dividido en fragmentos de longitud manejable.

Módulo de análisis de texto (MAT) Emplea modelos PLN/LLM para extraer emociones, escenarios, personajes y contexto de cada fragmento.

- **Entrada:** Fragmentos raw de longitud determinada.
- **Salida:** JSON con el análisis de cada fragmento individual. Incluye información de escenarios, personajes, emociones y contexto.

Módulo de consolidación de texto (MCT) Emplea técnicas PLN y de comparación de texto para limpiar, juntar y ajustar los análisis de los fragmentos individuales en un solo análisis final.

- **Entrada:** JSON con el análisis de cada fragmento individual. Incluye información de escenarios, personajes, emociones y contexto.
- **Salida:** JSON final con escenarios, personajes, emociones y contexto de la obra.

Módulo de construcción de prompts (MCP) Aplica técnicas de *prompt engineering* para generar el texto óptimo de entrada al motor de imágenes.

- **Entrada:** Tokens, contexto y preferencias del usuario.
- **Salida:** Prompt final listo para la generación de imagen.

Módulo de generación de imágenes (MGI) Usa el modelo *text-to-image*, para generar imágenes a partir de una lista de prompts.

- **Entrada:** Prompts de texto.
- **Salida:** Imágenes generadas.

3.2. Diseño

El diseño del proyecto se basa en una ejecución de módulos secuencial, que combinados consiguen analizar un texto no estructurado para generar imágenes. Se ha pensado para que el usuario pueda usar la interfaz para subir el archivo a analizar, así como seleccionar sus preferencias artísticas y acordes con su caso de uso. Existirá un botón que permite al usuario iniciar el análisis y, tras completarse, se podrán visualizar las imágenes resultado en la propia interfaz.

3.2.1. Requisitos

Requisitos funcionales

MPT – Preprocesamiento de Texto

- **MPT-01:** El sistema deberá poder leer un archivo de texto de entrada.
- **MPT-02:** El sistema deberá ser capaz de normalizar el texto para unificar comillas y apóstrofes.
- **MPT-03:** El sistema deberá poder sustituir abreviaciones por su forma completa utilizando.
- **MPT-04:** El sistema deberá dividir el texto en fragmentos de hasta 500 palabras sin cortar frases.
- **MPT-05:** El sistema deberá aclarar localismos si se usan en el texto.
- **MPT-06:** El sistema deberá eliminar caracteres especiales y ruido del texto.
- **MPT-07:** El sistema deberá sustituir los números por su representación en palabras.
- **MPT-08:** El sistema deberá guardar los fragmentos preprocesados en un archivo de salida.

MAT – Análisis de Texto

- **MAT-01:** El sistema deberá leer los fragmentos preprocesados desde los archivos de entrada.
- **MAT-02:** El sistema deberá cargar y preparar el modelo LLM de generación de texto (Llama 3.1-8B).
- **MAT-03:** El sistema deberá extraer al LLM con preguntas el escenario principal y sus detalles de cada fragmento.
- **MAT-04:** El sistema deberá extraer al LLM con preguntas detalles de los personajes principales en fragmento.
- **MAT-05:** El sistema deberá cargar los clasificadores de emociones y sentimiento y procesar cada fragmento para obtener su valoración afectiva.
- **MAT-06:** El sistema deberá guardar en formato JSON los resultados de escenarios, personajes, emociones y contexto.

MCT – Consolidación de Texto

- **MCT-01:** El sistema deberá leer el análisis intermedio desde el fichero JSON correspondiente.
- **MCT-02:** El sistema deberá filtrar instancias de escenarios, personajes y emociones incompletas en caso de no ser relevantes.
- **MCT-03:** El sistema deberá fusionar entradas equivalentes para escenarios y personajes.
- **MCT-04:** El sistema deberá recalcular los contadores de escenarios y personajes tras la consolidación.
- **MCT-05:** El sistema deberá eliminar repeticiones restantes comparando y unificando información redundante.
- **MCT-06:** El sistema deberá guardar el análisis final depurado en un archivo JSON de salida.

MCP – Construcción de Prompts

- **MCP-01:** El sistema deberá leer el análisis final, las preferencias artísticas y de imagen, y el LLM generador.
- **MCP-02:** El sistema deberá determinar si el análisis resulta demasiado extenso y, en tal caso, limitar el uso de memoria.
- **MCP-03:** El sistema deberá construir dinámicamente un prompt de generación de imagen para cada elemento (personaje, escenario o emoción) según el enfoque seleccionado.
- **MCP-04:** El sistema deberá garantizar la cantidad de prompts solicitada, así como su ajuste a preferencias.
- **MCP-05:** El sistema deberá guardar todos los prompts generados en un fichero de salida.

MGI – Generación de Imágenes

- **MGI-01:** El sistema deberá cargar y cuantizar el modelo de generación de imagenes.
- **MGI-02:** El sistema deberá generar una imagen por cada prompt que reciba.

Requisitos no funcionales

- **NF-01:** Uso exclusivo de software gratuito y OSS.
- **NF-02:** Ejecutable en un solo dispositivo, localmente y de nivel usuario (requerida como mínimo una GPU NVIDIA RTX 4090 con 24GiB de VRAM)
- **NF-03:** Rendimiento: alta correlación imagen-texto y que el uso de memoria no aumente notablemente, independientemente del texto de entrada.
- **NF-04:** Implementación escalable, modularizada y capaz de funcionar independientemente de la longitud del texto de entrada.
- **NF-05:** Sistema y pipeline transparente al usuario.
- **NF-06:** Proporcionar una interfaz simple, clara y accesible para cualquier usuario.

3.3. Implementación

3.3.1. Módulo de Preprocesamiento de Texto (MPT)

Este apartado describe la implementación y justificación del módulo encargado del preprocesamiento (Sección 2.1.1) de texto previo al análisis por parte del modelo de lenguaje. Se presentan tanto las técnicas exploradas como las decisiones adoptadas en la versión final, guiadas por los resultados empíricos obtenidos en las pruebas.

Estrategia adoptada

En una fase inicial, se investigaron y aplicaron varias técnicas comunes en sistemas estado del arte [58, 59]. Sin embargo, durante el desarrollo y evaluación del módulo de análisis, centrado en el reconocimiento de personajes, escenarios y emociones, se observó empíricamente que cuanto más natural y detallado era el texto, mejores eran las respuestas del LLM. Comparaciones sistemáticas entre versiones resumidas y completas de los fragmentos mostraron consistentemente que las versiones completas producían respuestas más ricas, precisas y coherentes. Para más detalle, se encuentran en la Sección 4.1 del capítulo de Pruebas.

A raíz de estos hallazgos, se optó por una estrategia de preprocesado simple y directa. El objetivo pasó a ser limpiar los textos sin alterar su estructura o nivel de detalle, con el fin de preservar al máximo el contexto, mejorando la comprensión semántica y extracción de información (Sección 2.1.1) del LLM.

El preprocesamiento final se basa en tres operaciones principales:

- **Normalización de texto**: sustitución de comillas irregulares, caracteres especiales, emojis, puntuación redundante o confusa y sustitución de números por sus numerales.
- **Expansión de contracciones y abreviaciones comunes**: se utiliza una tabla de sustituciones para homogeneizar el lenguaje.
- **Eliminación de ruido**: se aplican expresiones regulares para detectar y eliminar patrones no deseados (p. ej., secuencias de signos, caracteres de control o saltos de línea innecesarios).

A diferencia de otros enfoques, no se eliminan stop words ni se reduce el texto a una versión lematizada o cualquier técnica que modifique el lenguaje natural del texto. Aunque esto es para el análisis inicial realizado por el LLM, ya que en la Sección 3.3.2 de análisis, vemos cómo sí que se acaban eliminando stop-words para el análisis de emociones ya que es más óptimo.

Orden y lógica del preprocesamiento

En algunas publicaciones consultadas [60], se indica que el orden de aplicación de las técnicas de preprocesamiento puede afectar significativamente al rendimiento posterior, especialmente cuando se aplican transformaciones agresivas como STWR o lematización. En este módulo, al utilizar un enfoque más conservador, se estableció el siguiente orden lógico:

1.– Normalización de comillas, símbolos especiales y números
2.– Sustitución de abreviaciones
3.– Eliminación de ruido residual mediante expresiones regulares

Este orden garantiza que los fragmentos estén limpios y listos antes de su división en tokens y subfragmentos, sin introducir ambigüedades ni errores de segmentación.

Tamaño de fragmentos

Se adoptó una longitud de 500 tokens por fragmento como valor estándar, en línea con las ventanas de contexto más comunes en los LLM actuales, generalmente 512 tokens. Este tamaño deja margen suficiente para incluir instrucciones o contexto adicional en el prompt sin alcanzar el límite del modelo, asegurando una correcta interpretación de los textos durante el análisis. La división se realiza de forma que los fragmentos mantengan coherencia narrativa, priorizando siempre el corte en puntos naturales del texto (párrafos o frases completas).

Consideraciones sobre herramientas externas y decisiones finales

Durante la fase de validación, se exploró el uso de herramientas adicionales que permitieran enriquecer los fragmentos con metadatos (entidades reconocidas, palabras clave, clasificaciones temáticas, etc.). Aunque algunas de estas soluciones ofrecían resultados prometedores, su naturaleza de caja negra (al depender de modelos externos sin posibilidad de modificación) introducía errores difí-

ciles de controlar. Si bien estas soluciones son muy precisas, no obtienen un 100 % de aciertos. Una entidad no reconocida no sería un fallo insalvable; el problema viene con las entidades etiquetadas erróneamente. Por ejemplo, al usar el modelo en_core_web_sm de la librería Spacy [61, 62], hay textos en los que las personas aparecen etiquetadas como lugares y viceversa. Esta información, si se la proporcionamos al módulo de análisis de texto (MAT) como contexto, se tomaría como verdad y las preguntas posteriores supondrían respuestas erróneas, haciendo que se acaben teniendo errores con gran impacto para el resultado final.

Además, estas herramientas suelen requerir recursos computacionales adicionales y tiempos de ejecución más prolongados. Por estas razones, y en línea con el requisito de eficiencia del proyecto, se descartaron en favor de una solución ligera, explicable y casi instantánea, basada exclusivamente en expresiones regulares y tablas de sustitución locales.

3.3.2. Módulo de Análisis de Texto (MAT)

Este apartado describe la elección del modelo de lenguaje, su parametrización y el flujo de trabajo diseñado para extraer de fragmentos de texto la información necesaria sobre personajes, escenarios y contexto. Se detallan asimismo las estrategias de prompting, el tratamiento de emociones y sentimientos y la gestión de memoria.

Selección e integración del LLM

Para garantizar que el proyecto permaneciera fiel a sus requisitos, código abierto y ejecución local, se examinó durante noviembre de 2024 una serie de LLMs (Sección 2.1.2) de disponibilidad pública. Tras evaluar en escenarios de pregunta-respuesta con los fragmentos preprocesados, se eligió Llama 3.1-8B-Instruct (Sección 2.1.2) por ofrecer un equilibrio aceptable entre calidad de respuesta y demanda de recursos. La implementación se apoya en la librería Hugging Face (Sección 2.3), lo que permite reemplazar o escalar el modelo fácilmente conforme aparezcan versiones más avanzadas.

Ajuste de hiperparámetros y precisión mixta

La configuración óptima de los hiperparámetros se alcanzó tras sucesivas iteraciones, siempre monitorizando el uso de VRAM. Se constató que emplear precisión mixta (bfloat16) en lugar de float32 reducía significativamente la memoria requerida y aceleraba el cómputo, sin penalizar de forma apreciable la calidad de las respuestas. En concreto, el límite de generación de tokens (max_new_tokens) se fijó en 100 para obtener respuestas concisas, y el recuento de beam searches (num_beams) ascendió a 5, fortaleciendo la coherencia a costa de un ligero incremento en el tiempo de procesamiento.

Estos ajustes permitieron procesar contextos extensos—hasta varios miles de tokens—con un coste de memoria estable y tiempos de respuesta adecuados para una estación de trabajo con una única

NVIDIA RTX 4090.

Flujo de procesamiento secuencial

El pipeline de análisis inicia la carga del generador de texto y ajusta las rutas de acceso al modelo. Para cada fragmento, se formulan dos consultas generales destinadas a identificar el escenario principal y los personajes implicados. Las preguntas sobre el contexto del fragmento siempre se realizan. Si la respuesta del LLM resulta poco concluyente, el LLM responde con la etiqueta UNKNOWN, sobre la que la lógica del módulo actúa, omitiendo las preguntas detalladas correspondientes a esa categoría.

Entrando en detalle, el módulo funciona mediante un pipeline estrictamente secuencial, lo que facilita la trazabilidad y hace sencilla la escalabilidad. Primero, se formulan las preguntas base mencionadas, para determinar escenarios y personajes principales. A continuación, para cada uno de ellos, se lanzan las consultas detalladas, siguiendo el diseño preestablecido: en el caso de los escenarios (hora del día, interior/exterior, climatología, ...) y para los personajes (apariencia, vestimenta, característica significativa, ...). Tras esto se realizan las preguntas de contexto.

Tras completar la fase de pregunta-respuesta con LLM, se libera la memoria RAM de Python antes de invocar los modelos especializados en análisis de emociones y sentimiento sobre cada fragmento y guardar los resultados. Juntando todo, se obtiene un JSON que recoge, para cada fragmento individual, la información de personajes, escenarios, emociones y contexto, y que servirá de entrada al siguiente módulo de consolidación de la Sección 3.3.3.

Estrategias de prompting

La efectividad del análisis depende en gran medida de la claridad de las instrucciones proporcionadas al LLM. Siguiendo las recomendaciones explicadas en la Sección 2.1.1, cada prompt comienza por asignar al modelo el rol de “experto en extracción de información de texto”. A continuación, se incorpora el fragmento completo y se emite la pregunta en formato directo, sin solicitar justificaciones, con el fin de traducir la respuesta de forma directa a un campo JSON.

Para estandarizar las respuestas erráticas —longitudes excesivas, omisiones o ambigüedades— se indica al LLM que responda con la etiqueta UNKNOWN. Este mecanismo garantiza que las salidas posean un formato uniforme y facilita su postprocesado sin requerir intervenciones adicionales.

Análisis de emociones y sentimiento

Una vez completado el barrido de preguntas sobre escenarios, personajes y contexto con el LLM principal, se ejecuta una capa adicional de clasificación de emociones y sentimiento. Previo a ello, realizamos un paso que omitimos en el preprocesamiento para facilitar la comprensión al LLM, limpiar los fragmentos de stop-words para centrar la atención en los términos portadores de carga emocional [60].

Se utilizan modelos especializados de Hugging Face—SamLowe, roberta-base-go-emotions (Sección 2.1.4) y distilbert-base-uncased-finetuned-sst-2-english (Sección 2.1.4) —que cumplen con los requisitos de código abierto y presentan un perfil de consumo de recursos muy inferior al LLM general. Los resultados de esta etapa se integran en el JSON bajo subcampos de “emoción” y “polaridad”.

Gestión de memoria

Para garantizar la continuidad del proceso en una sola GPU, se aplican dos métodos de limpieza. Entre bloques de preguntas del LLM y antes de cargar los clasificadores de sentimiento, se libera la caché de PyTorch (`torch.cuda.empty_cache()`), manteniendo las referencias a tokenizador y generador, ya que se reutilizan. Al concluir la fase de sentimiento, el modelo se traslada por completo a CPU, se elimina la referencia en Python y se invoca `gc.collect()` seguido de un nuevo `torch.cuda.empty_cache()`. De este modo, nos aseguramos con redundancias de que la GPU se encuentra completamente libre antes de la siguiente carga de modelo en el MCT (Sección 3.3.3), evitando fugas de memoria y maximizando la estabilidad.

3.3.3. Módulo de Consolidación de Texto (MCT)

Este apartado describe en detalle la consolidación de las instancias individuales de cada fragmento en el JSON de análisis, en un único JSON final, coherente y cohesionado. Se explica la evolución de los problemas y soluciones que surgen en esta tarea, evaluando distintas técnicas de comparación de texto, la lógica de filtrado, fusión y ajuste aplicado en el pipeline.

Objetivo y desafíos

Al inicio de este módulo se cuenta con un conjunto de respuestas generadas por el LLM, estructuradas en JSON y clasificadas por categorías —personajes, escenarios y emociones—. Dado el carácter probabilístico del modelo de lenguaje, resulta frecuente encontrar reconocimientos erróneos (por ejemplo, detección de un personaje inexistente) o la duplicación de detalles en fragmentos distintos. Asimismo, preguntas sencillas pueden quedar sin respuesta por falta de información o por fallos de coherencia interna del LLM. Por último, al haber analizado cada fragmento por separado, existen instancias equivalentes de personajes o escenarios que en el JSON de entrada están separadas, probablemente conteniendo detalles complementarios en cada una de las menciones. El reto consiste en limpiar estas inconsistencias, agrupar las instancias equivalentes y rellenar huecos de forma precisa, todo ello manteniendo un uso moderado de recursos computacionales en una sola GPU de 24 GB de VRAM.

Técnicas exploradas y resultados

En una primera aproximación se generaron embeddings (Sección 2.1.5) individuales para cada entrada, inspirándose en la lógica de Word2Vec [63]. Cada frase se transformó en un vector $\mathbf{u} \in \mathbb{R}^n$ y, para medir similitud, se calculó la norma del vector diferencia: $\|\mathbf{d}\| = \|\mathbf{u}_1 - \mathbf{u}_2\|$.

Sin embargo, esta técnica demostró ser demasiado sensible a la magnitud de los vectores: cuando dos embeddings apuntaban en direcciones similares pero con longitudes distintas, la norma $\|\mathbf{d}\|$ resultaba elevada, generando falsos negativos.

Como alternativa, se entrenó una red siamesa basada en BERT (Sección 2.1.5), utilizando datasets como SNLI, Quora Question Pairs y MRPC, de implementación propia pero inspirada en [35]. A pesar de que la precisión mejoró, las exigencias de cómputo —limitando el entrenamiento a 72 horas de GPU y reduciendo la precisión en la inferencia— no permitieron alcanzar la robustez deseada para su uso en un entorno de usuario medio. Estos resultados hicieron que se volviera al punto de partida, obligando a investigar más posibles soluciones.

Al final se optó por volver a utilizar embeddings, pero esta vez comparándolos por similitud del coseno, en la Sección 2.1.5 del Estado del Arte, que, recordando lo explicado, a diferencia de la primera aproximación, se centra en la alineación semántica, resultando más estable y fiable para detectar duplicidades y agrupaciones. Con esto resuelto, se procedió a implementar el pipeline de consolidación.

Implementación del pipeline de consolidación

La lógica de procesamiento se articula en cuatro etapas secuenciales:

1.– **Filtrado de entradas insuficientes.** Se eliminan aquellas respuestas que no alcanzan un umbral mínimo de completitud, en concreto, menos del 50 % de campos válidos en escenarios, menos de 15 campos con información en personajes o puntuación de confianza <0.2 en emociones.

2.– **Fusión de entidades equivalentes.** Para cada par de entradas dentro de una categoría, se computa $\mathrm{sim}_{\cos}$ entre sus embeddings generados por `paraphrase-mpnet-base-v2`. Cuando el valor supera 0.87 —determinado arbitrariamente mediante prueba y error— se consolidan los atributos de ambas en un único objeto JSON.

3.– **Depuración de repeticiones internas.** Dentro de cada objeto fusionado, se vuelve a aplicar la similitud del coseno con umbral 0.5 para identificar y eliminar frases redundantes en cada campo. Se garantiza la coherencia sin pérdida de matices ya que no se eliminan frases sin asegurarse de que se guarde la que más información aporta, en este caso el criterio es quedarse siempre con la respuesta más detallada.

4.– **Ajuste y estandarización del esquema.** Finalmente, se actualizan los contadores de personajes, escenarios y emociones, y se homogeneizan los nombres de las claves JSON para producir un documento final que refleje fielmente el conjunto de información disponible.

Al centralizar toda la lógica en la métrica de similitud del coseno, se consiguió una consolidación fiable y eficiente, evitando la complejidad y el coste de entrenamiento de la red siamesa y superando los problemas de la imprecisión de los embeddings.

3.3.4. Módulo de Construcción de Prompts (MCP)

Este apartado expone la implementación del módulo encargado de generar los prompts dirigidos al módulo de generación de imágenes (Sección 3.3.5). Se describe el proceso completo, desde la inicialización del estado de ejecución hasta las estrategias de prompting empleadas para maximizar la calidad y diversidad de las salidas.

Preprocesamiento y limpieza de estado

Para garantizar que el módulo puede operar de forma independiente o reiniciarse tras posibles errores en etapas anteriores, se incluye un procedimiento inicial de depuración de memoria. Mediante la invocación a `torch.cuda.empty_cache()` se asegura que la GPU quede libre de cargas residuales, estableciéndose un punto de partida con consumo nulo de VRAM. Además, cuando el archivo de análisis sobrepasa cierta extensión, se activa una bandera haciendo que se limite la extensión de los prompts enviados al LLM para evitar un uso excesivo de memoria en esos casos.

Estructura de entradas y flujo de generación

El módulo recibe tres estructuras JSON como entrada: primero, el análisis final proveniente del MCT (Sección 3.3.3); segundo, las preferencias artísticas del usuario —incluyendo estilo, calidad y matices cromáticos—; y tercero, la especificación del número de imágenes solicitadas junto con la categoría en la que se quiere enfocar (personajes, emociones o escenarios). Estas preferencias se definen por el usuario en la interfaz implementada antes de cada llamada. A partir de estos datos, se construye un pipeline en el que, para cada iteración, se invoca al LLM con un único objetivo: formular un prompt detallado y coherente para la generación de la correspondiente imagen.

Gestión de múltiples prompts y cobertura temática

Cuando el usuario solicita más de una imagen, resulta imprescindible evitar la generación de prompts redundantes y asegurar la representación de todas las entradas relevantes dentro de la categoría elegida. Para ello, se ha diseñado un mecanismo en el que el LLM recibe en cada iteración el análisis completo, junto con una instrucción adicional que indica explícitamente el elemento específico de la categoría que debe priorizar. De este modo, si existen cinco escenarios definidos y se requieren tres imágenes, el módulo generará tres prompts distintos, cada uno centrado en un escenario diferente, garantizando diversidad en la cobertura temática sin depender de una elección arbitraria del LLM.

Postprocesado de la salida del LLM y superación de limitaciones

El LLM empleado, Llama 3.1-8B-Instruct (Sección 2.1.2) en modalidad de precisión mixta, ofrece un rendimiento adecuado en hardware de gama media pero presenta restricciones inherentes a su

capacidad de razonamiento y formato de salida. Con el fin de homogeneizar los prompts y facilitar su posterior consumo por el pipeline de imágenes, se implementaron dos soluciones clave. Por un lado, se configuró el hiperparámetro `max_new_tokens` en 500, ajustando la longitud máxima de la respuesta debido a que el encoder de muchos modelos de generación de imágenes tiene como límite 512 tokens.

Por otro lado, el simple uso de comillas para marcar la respuesta no funcionaba, ya que Llama 3.1-8B mostraba inconsistencias al cerrar automáticamente cadenas entre comillas y otros métodos de delimitación, haciendo imposible un postprocesado limpio de la respuesta obtenida. Se introdujo como solución un mecanismo de delimitación basado en una palabra de inicio y fin, de modo que el texto relevante quedara siempre contenido entre marcadores predefinidos.

Estrategia de prompting

La formulación de un prompt óptimo exige una estructuración cuidadosa que combina rol, contexto y tareas específicas. En cada petición, se establece primero el LLM como “experto en creación de prompts para generación de imágenes”, a continuación se incorpora el JSON de análisis íntegro y se define la tarea concreta orientada a la categoría seleccionada. Finalmente, se añaden las preferencias artísticas del usuario y la instrucción de formato para garantizar la delimitación correcta del prompt. Este enfoque —que parte de un rol claro, pasa por la exposición completa del contexto y culmina en directrices precisas— ha demostrado, tras múltiples iteraciones de prueba y error, maximizar tanto el detalle descriptivo como la coherencia semántica de los prompts producidos.

3.3.5. Módulo de Generación de Imágenes (MGI)

Este apartado describe detalladamente el diseño e implementación del componente encargado de traducir los prompts generados en la sección anterior en imágenes de alta calidad. Se explican las motivaciones para descartar las primeras versiones del módulo, la integración de FLUX v1.0 [dev], los problemas resueltos y las optimizaciones que terminaron de perfilar la solución implementada.

Modelos iniciales y limitaciones detectadas

En la fase preliminar de prototipado se exploró la incorporación de dos de los sistemas más reputados en generación de imágenes disponibles a finales de 2024: MidJourney v1.5 y OpenJourney v4. El primero fue escogido por su eficacia en la generación de paisajes realistas; el segundo, por la capacidad de producir personajes con gran riqueza de detalle. No obstante, ambos modelos presentaron dos problemas críticos al desplegarse localmente en una estación de trabajo equipada con NVIDIA RTX 4090 (24 GB de VRAM). Para empezar, el consumo de memoria oscilaba constantemente por encima de los 20 GB, lo cual impedía la coexistencia de otros módulos y ponía en riesgo la estabilidad del sistema. Si se optaba por ejecutar los modelos en remoto, la dependencia de llamadas a la API

de Hugging Face (Sección 2.3) introducía latencias variables –desde treinta segundos hasta más de dos minutos– y restricciones de uso que resultaban incompatibles con la exigencia de autonomía y rapidez establecida en los objetivos del proyecto. Además, cuando los prompts requerían la conjunción de múltiples elementos o detalles complejos, la calidad final de las imágenes se alejaba de lo deseado, con alucinaciones o la omisión de detalles esenciales.

Adopción y evaluación de FLUX v1.0 [dev]

La llegada de FLUX v1.0 [dev] de Black Forest Labs supuso un giro decisivo. Este modelo, descrito en la Sección 2.2.2 del Estado del Arte, permite equilibrar velocidad de inferencia y fidelidad visual. La calidad de las imágenes obtenidas a partir de las primeras pruebas suponía una clara mejora en comparación con los otros modelos considerados. La adopción de este modelo permitió reflejar los detalles descritos en los prompts de entrada, demostrando alta correlación entre el texto y la imagen generada. Sin embargo, para tener estos resultados FLUX v1.0 [dev] es un modelo bastante exigente a nivel de recursos computacionales, por lo que en un principio se mantuvo la implementación con APIs, hasta conseguir comprobar que todos los módulos funcionaban y se coordinaban adecuadamente. Tras esto, se optó por investigar maneras de reducir los requisitos computacionales del modelo, de forma que pudiese ejecutarse en local, independiente de servicios de terceros.

Cuantización de pesos e implementación técnica

Para viabilizar la ejecución íntegra en la RTX 4090 sin saturar la VRAM, se adoptó una estrategia de cuantización (Sección 2.2.3) de pesos basada en la función `quantize_` del paquete `torchao.quantization` [53], utilizando únicamente la modalidad `int8_weight_only`. La secuencia de pasos fue la siguiente:

- En primer lugar, se cargaron los parámetros del transformador en tipo `bfloat16`.
- A continuación, las capas de atención y *feed-forward* se redujeron a `int8`, mientras que los componentes más sensibles permanecieron en `bfloat16`.

Esta decisión permitió rebajar el uso de memoria en aproximadamente un 60 %, habilitando la posibilidad de la ejecución completa del modelo en la máquina local. Simultáneamente, se mejora la velocidad de procesamiento, alcanzando un promedio de diez segundos por imagen en formato 896×1280 px con veinte pasos de muestreo y un factor de *guidance* de 4.0. Para optimizar aún más el uso de recursos, el *pipeline* se configuró con *offload* a CPU de los bloques menos solicitados, liberando espacio en GPU durante cada ciclo de inferencia.

Gestión de la longitud de los prompts y override de encoder

Durante las pruebas iniciales se detectó que prompts de más de 256 tokens perdían la información tras este número en las imágenes generadas. Esta limitación respondía al encoder CLIP, por defecto en

la biblioteca `Diffusers`, cuya ventana de contexto máxima es, efectivamente, de 256 tokens. Para superar este obstáculo, se implementó un *override*, reemplazando CLIP por el encoder del modelo FLUX: T5, con capacidad para procesar hasta 512 tokens de contexto. La integración de T5 no solo extendió la longitud máxima de prompt, sino que mejoró la coherencia semántica de los embeddings, factor determinante para la correcta correspondencia entre texto e imagen cuando se manejan descripciones extensas (típicamente entre 400 y 500 tokens en los prompts generados por el módulo previo).

Arquitectura del pipeline de inferencia

El flujo de ejecución del módulo se estructuró en cuatro fases claramente diferenciadas:

1.– En la primera, se libera la memoria de la GPU mediante `torch.cuda.empty_cache()`, garantizando un estado limpio de VRAM.

2.– En la segunda, se instancia el transformador con `FluxTransformer2DModel.from_pretrained`, aplicando la cuantización `int8` y habilitando el *offload* de componentes a CPU.

3.– La tercera etapa consiste en crear el *pipeline* de difusión a través de `DiffusionPipeline.from_pretrained`, reutilizando el transformador ya optimizado y especificando `dtype = bfloat16`.

4.– Finalmente, para cada prompt en la lista de entrada, se calculan los *embeddings* ponderados usando `get_weighted_text_embeddings_flux1` y se genera la imagen con los parámetros fijados.

Cada resultado se guarda en disco con una nomenclatura estandarizada, facilitando su posterior análisis y comparación.

3.3.6. Interfaz de usuario

La interfaz de usuario ha sido diseñada con un enfoque minimalista y funcional, priorizando la usabilidad por encima de la complejidad visual. Construida con Streamlit, esta interfaz permite al usuario cargar fácilmente los documentos de texto que desea analizar, así como definir una serie de preferencias artísticas y de salida antes de ejecutar el análisis. La disposición por pestañas facilita la navegación entre las distintas secciones, manteniendo el flujo de trabajo claro y segmentado.

En la barra lateral, el usuario puede subir los archivos a procesar y lanzar la ejecución mediante un simple botón, mientras que las pestañas principales permiten ajustar las preferencias estilísticas —como el tipo de iluminación, ángulo de cámara o textura visual— y establecer criterios para las ilustraciones generadas. Una vez finalizado el proceso, los resultados se muestran directamente en la pestaña correspondiente. Esta organización sencilla y guiada reduce la fricción en el uso, permitiendo que tanto usuarios técnicos como no técnicos puedan utilizar la herramienta sin necesidad de conocimientos previos.

3.3.7. Conclusiones de la sección de Implementación

Para concluir la sección de implementación, cabe destacar que el módulo de preprocesado ha alcanzado un equilibrio óptimo entre simplicidad y eficacia, preservando el detalle y la estructura del texto original y manteniendo un bajo coste de tiempo y recursos. En el futuro, podría intentar enriquecerse explorando el uso de mejores herramientas de extracción de metadatos específicas de dominio.

Por su parte, la arquitectura secuencial del módulo de análisis ha demostrado su modularidad y fiabilidad, aunque su latencia podría reducirse mediante batching de peticiones, así como la precisión, con encadenamiento de fragmentos para aportar contexto adicional. Un punto a favor es que su diseño flexible permite sustituir el LLM base por modelos de mayor capacidad sin alterar la lógica del módulo.

El módulo MCT (Sección 3.3.3) ha confirmado que la similitud del coseno sobre embeddings ligeros proporciona una consolidación rápida y coherente de la información, asumiendo sin riesgo algunas redundancias que serán corregidas en etapas posteriores de prompting. Cualquier mejora que ayude a completar mejor la información de instancias de personajes, escenarios y contexto se verá reflejada en los resultados finales. Un ejemplo sería una de las ideas del diseño inicial, en la que se proponía añadir una capa de coherencia y redundancia, comparando el informe final con los fragmentos una vez más en busca de elementos que añadir.

En cuanto al generador de prompts, la solución local y escalable ha superado las limitaciones del LLM open-source, produciendo salidas diversas y fieles al análisis semántico y a las preferencias artísticas. Está preparada para integrar modelos futuros más potentes con mínimos cambios.

Finalmente, el componente de generación de imágenes, basado en FLUX v1.0 [dev] y optimizaciones de cuantización y override de encoder, ha satisfecho los requisitos de ligereza, velocidad y calidad, estableciendo una base sólida para explorar técnicas de distilación de modelos y estrategias de muestreo avanzadas que incrementen la diversidad y el realismo sin comprometer el rendimiento.

En conjunto, estos módulos configuran un flujo robusto, ligero, autónomo y plenamente extensible, que podrá evolucionar con futuras líneas de mejora e investigación.

4

Pruebas

Esta sección se centra en la evaluación empírica del sistema y sus módulos. Se realizan pruebas para comprobar si se ha conseguido alcanzar las metas propuestas en la Sección 1.2.

En base a los resultados obtenidos, se trata de validar la hipótesis de que, mediante la unión cuidadosa y coordinada de modelos y herramientas, se pueden sobrepasar desafíos como el límite de tokens o la potencia computacional requerida por modelos con aplicaciones similares, sin sacrificar calidad en el producto final.

Entorno de pruebas

Todas las pruebas realizadas durante la implementación y en esta sección han sido realizadas en un único dispositivo, de forma local. Más concretamente, el hardware utilizado es una GPU NVIDIA RTX 4090, que cuenta con 24 GiB de VRAM.

4.1. Pruebas de funcionalidad

En estas pruebas, se ejecuta el pipeline con una muestra de 1.731 palabras, o un cuarto de capítulo del primer libro de Harry Potter: La Piedra Filosofal, escrito por J.K. Rowling, generando 2 imágenes centradas en los escenarios y con un estilo realista.

Para evaluar la funcionalidad de los módulos, se analizan los archivos generados entre ellos, ya que estos suponen también la entrada del siguiente en el pipeline.

El objetivo es comprobar que cada módulo funciona como se espera, así como el guardado e ingesta de los archivos intermedios y el tiempo que tardan en ejecutarse. De esta forma, no sólo se refleja la funcionalidad, sino que se ofrece una comparativa entre ellos, útil para saber qué parte del proyecto puede ser más optimizada en un futuro.

Módulo de preprocesamiento (MPT)

Tiempo en ejecutar: 0.001 minutos = 0.6 segundos.

Gracias al uso de herramientas ligeras en su implementación (Sección 3.3.1), MPT, es capaz de ejecutar en menos de un segundo. En el Cuadro 4.1, se puede ver una parte del fragmento preprocesado, completo en el Anexo A - A.1, cada uno con menos de 500 tokens. En su versión original, este tenía contracciones como *didn't*, abreviaciones como *Mr. y Mrs.*, sustituidas ahora por su versión completa, tal y como se esperaba. Con esto, se facilita la comprensión del contexto al LLM, a la vez que las transformaciones a embeddings que hace internamente, optimizando la precisión de sus respuestas.

mister and misses Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal , thank you very much. They were the last people you would expect to be involved in anything strange or mysterious, because they just did not hold with such nonsense. mister Dursley was the director of a firm called Grunnings , which made drills. He was a big, beefy man with hardly any neck, although he did have a very large mustache. misses Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences , spying on the neighbors. The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere . The Dursleys had everything they wanted , but they also had a secret , and their greatest fear was that somebody would discover it . They did not think they could bear it if anyone found out about the Potters

Cuadro 4.1: Fragmento representativo del preprocesado

Pruebas iniciales con los fragmentos preprocesados

Una vez se implementaron ambos módulos, como se comenta en la Implementación de MPT, se realizaron pruebas para comprobar qué tipo de preprocesamiento funcionaba mejor.

Uno de los hallazgos más interesantes es que, cuánto más contexto tenía y más natural era el fragmento, mejores eran las respuestas de LLama3.1-8B ante las preguntas. Esto se podía apreciar a simple vista en el JSON de análisis tras comparar los resultados obtenidos al tener como entrada los fragmentos preprocesados sin stop-words, resumidos, o, como se optó en la implementación final para personajes y escenarios, un preprocesado que conservaba el lenguaje más natural posible. Una métrica que podemos utilizar es el número de campos vacíos en el JSON en cada caso, observable en la Tabla 4.1, que demuestra claramente que la aproximación implementada funciona mejor que las otras alternativas.

Tipo	Campos vacíos	Total campos	Ratio
Implementación	53	374	0,14
Resúmenes	187	680	0,28
Sin stop-words	75	374	0,20

Tabla 4.1: Número de campos vacíos para cada tipo de preprocesado

Módulo de análisis (MAT)

Tiempo en ejecutar: 7.06 minutos = 7 minutos, 3.6 segundos.

En el tiempo que tarda este Módulo de análisis (Sección 3.3.2), se refleja su implementación secuencial. Durante la ejecución, al LLM se le han realizado una serie de preguntas detalladas sobre cada escenario y personaje detectado, para cada fragmento. El Código 4.1, muestra la instancia del escenario principal del primer fragmento. En este JSON, se aprecian los resultados de dichas preguntas, habiéndose extraído la información esperada de las preguntas. Esto nos indica que el escenario se describía con detalle en el fragmento, exactamente el resultado esperado para la instancia en concreto.

Código 4.1: Escenario del JSON del análisis inicial


```
"Scenarios": {
    "Scenarios_list": [
        {
            "Real": {
                "Value": "FICTIONAL",
                "Coordinates": ""
            },
            "Name": "Number 4 PRIVET DRIVE",
            "Place": {
                "Type": "House",
                "Outdoors": "NO"
            },
            "Time_of_day": "HALF PAST EIGHT",
            "Weather": "GRAY AND CLOUDY",
            "Other_details": "DURSLEYS LIVED THERE"
        },
        {
```


Módulo de consolidación (MCT)

Tiempo en ejecutar: 3.68 minutos = 3 minutos, 40.8 segundos.

A pesar de hacer varias pasadas por el JSON de análisis, analizando sus entradas y calculando embeddings y similitud entre estas — detallado en la Implementación de MCT (Sección 3.3.3) — el tiempo de ejecución no es muy elevado en comparación con el Módulo de análisis (Sección 4.1). Esto se debe a que, desde la limpieza hasta el reajuste, pasando por el *merge*, evitan el uso de modelos,

lo que reduce la latencia del módulo al coste base de los cálculos y operaciones que se realizan en su lógica.

Código 4.2: Escenario del JSON del análisis final


```
            "Coordinates": ""
        },
        "Name": "Number 4 PRIVET DRIVE",
        "Place": {
            "Type": [
                "House",
                "Residential"
            ],
            "Outdoors": "YES"
        },
        "Time_of_day": [
            "HALF PAST EIGHT",
            "DAYTIME"
        ],
        "Weather": "GRAY AND CLOUDY",
        "Other_details": [
            "DURSLEYS LIVED THERE",
            "Driveway"
        ]
    },
```


En el Código 4.2, se observa el resultado de este módulo en la instancia de *"Number 4 Privet Drive"*. Se puede apreciar la diferencia con el Código 4.1. El mayor detalle en campos como tipo o momento del día se debe a que, en otro fragmento posterior, se detectó el mismo escenario y, tras compararlos como se explica en la Implementación del MCT (Sección 3.3.3), se han clasificado como equivalentes. Consiguiendo el resultado que se esperaba, se han unido las instancias, otorgando en el JSON final (completo en el Anexo A - A.2) un mayor nivel de detalle al escenario.

Módulo de creación de prompts (MCP)

Tiempo en ejecutar: 4.14 minutos = 4 minutos, 8.4 segundos.

Similarmente a lo que pasaba en MAT (Sección 4.1), las llamadas secuenciales al LLM provocan un aumento considerable en el tiempo de ejecución de este módulo.

En el Cuadro 4.2, se muestra un ejemplo reducido del prompt generado para el escenario 1, mismo que se lleva analizando en los Códigos 4.1 y 4.2. En la muestra del prompt se aprecia el cumplimiento de las expectativas para este, reflejando las instrucciones proporcionadas. No solo se realiza una descripción detallada del escenario en cuestión en base a los datos recibidos del análisis final, sino que se crea un prompt diseñado para usarse en un modelo de texto a imagen. Cabe apreciar que también se reflejan con éxito las especificaciones de estilo pedidas para esta prueba: estilo hiperrealista, a nivel

> *A hyper-realistic, full-body shot of Number 4 Privet Drive at half past eight on a gray and cloudy day, with the sun peeking through the clouds, casting a warm, golden light on the scene. The camera is positioned at eye-level, capturing the entire facade of the house, with a slight emphasis on the front door, which is slightly ajar. The scene is set in a residential area, the focus on the Dursleys' home. The overall mood of the scene is one of quiet, suburban normalcy, punctuated by the hint of something mysterious lurking just beneath the surface. In the foreground, a small patch of overgrown grass and a few scattered leaves add a touch of whimsy to the scene, hinting at the magic that lies just beyond the edges of this seemingly ordinary world. The resolution is full HD, and the aspect ratio is 16 by 9.*

Cuadro 4.2: Fragmento del prompt escenario 1 generado

del ojo, resolución full-HD y una ratio de 16:9.

Módulo de generación de imágenes (MGI)

Tiempo en ejecutar: 1.38 minutos = 1 minuto, 22.8 segundos.

Gracias a la cuantización del modelo de generación de imágenes, se logra con éxito ejecutar localmente en un tiempo muy reducido para la tarea, sobre todo, como se menciona en la Implementación del MGI (Sección 3.3.5), en comparación con llamar a la API.

Se observa en las imágenes generadas de la Figura 4.1, el cumplimiento de las especificaciones del prompt, así como la presencia de los detalles esperados en cada uno de los escenarios. Algunos ejemplos notables son el hecho de que la primera imagen (izquierda) se sitúe en una zona residencial y el cielo esté nublado o que, en la segunda imagen, se observe una oficina en Inglaterra, denotado por el Big Ben de fondo.

Figura 4.1: Imágenes generadas, escenarios 1 y 2

Conclusiones extraidas

Cada módulo obtiene los resultados esperados en cuanto a funcionalidad se refiere. Si sumamos los tiempos parciales de cada módulo, obtenemos que el proceso completo dura 16.26 minutos (16 minutos y 15.6 segundos), generando las imágenes de forma satisfactoria.

Sin embargo, existen algunas limitaciones a la hora de realizar el análisis, en su mayoría por parte del LLM, que no es capaz de identificar algunos de los detalles más concretos mencionados en los fragmentos, dejando campos del análisis vacíos o incompletos a pesar de contar con los detalles en el fragmento. A partir de las mediciones de tiempo también podemos observar cómo, por su implementación secuencial y la cantidad de preguntas que realiza, explicado en detalle en la Sección 3.3.2, MAT es el módulo que más podría optimizarse junto con MCP.

4.2. Evaluación del rendimiento

El presente apartado tiene como meta poner a prueba las capacidades del proyecto, mostrando dos métricas en relación a la extensión del texto procesado que se consideran importantes para evaluar el rendimiento general: tiempo y uso de memoria. Para ambas métricas y cada una de las distintas extensiones, las pruebas se realizan sobre 7 imágenes a generar.

En la Tabla 4.2 se puede observar un resumen de los datos obtenidos en las pruebas.

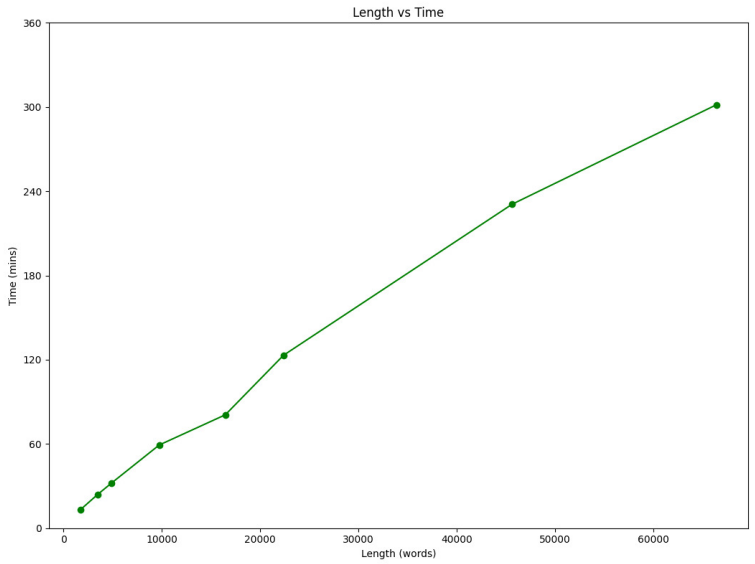
Longitud (palabras)	Tiempo (min)	Pico de memoria (GiB)	Etiqueta
1.731	13,275	17,58	1/4_capítulo
3.477	24,086	18,69	1/2_capítulo
4.900	32,279	19,84	1_capítulo
9.731	59,246	16,81	2_capítulos
16.461	80,782	16,81	3_capítulos
22.373	123,142	16,81	5_capítulos
45.631	230,719	16,81	10_capítulos
66.437	301,638	16,81	15_capítulos

Tabla 4.2: Datos de longitud, tiempo y memoria según el número de capítulos procesados.

4.2.1. Extensión frente a tiempo

Por un lado, se han realizado mediciones sobre cuánto tiempo tarda el programa en ejecutarse según se aumenta el número de palabras.

Calculando la velocidad con la que se procesan las palabras a partir de la Figura 4.2, se ven



Length (words)	Time (mins)
1731	13.275
3477	24.086
4900	32.279
9731	59.246
16461	80.782
22373	123.142
45631	230.719
66437	301.638

Figura 4.2: Datos de tiempo según el número de capítulos procesados

irregularidades, obteniendo una media de 174.29 palabras/minuto (wpm), donde la mediana es 172.97 wpm y la desviación típica llega a las 29.62 wpm.

- A medida que aumenta la longitud, la velocidad de procesamiento suele aumentar (por ejemplo, de aproximadamente 130 wpm en 1 731 palabras a 220 wpm en 66 437 palabras).
- Esto se explica porque existe un coste fijo inicial (carga del modelo, preprocesamiento…) que, en textos muy cortos, penaliza muchísimo el ratio palabras/minuto. Conforme el texto crece, ese coste se amortiza sobre más palabras y la velocidad media crece.

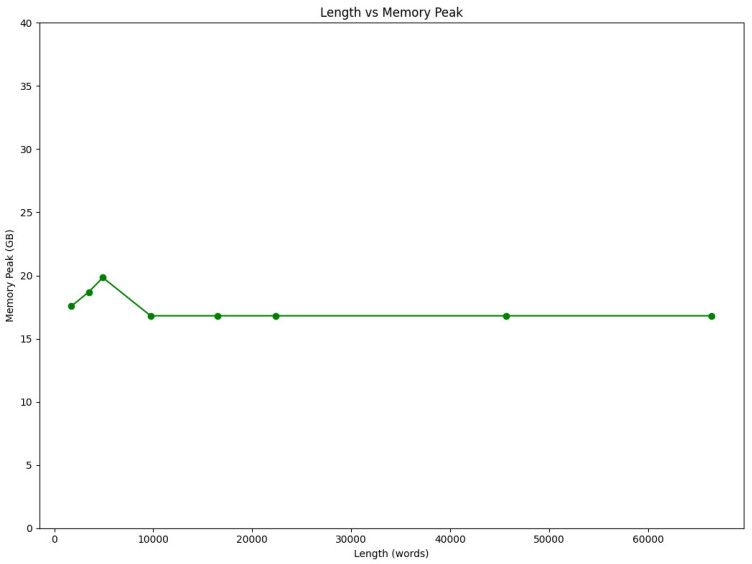
Si ajustamos una regresión lineal del tipo $\mathrm{wpm} \approx a \times (\text{palabras}) + b$, concretamente, realizando un ajuste $a \approx 1{,}3 \times 10^{-3}\ \frac{\mathrm{wpm}}{\text{palabra}}$, $R^2 \approx 0{,}77$, obtenemos un coeficiente de determinación R^2 moderado (alrededor de 0,75–0,80), lo que denota cierto crecimiento aproximadamente lineal, pero no estricto. La dispersión se debe a las irregularidades innatas en la complejidad del texto o la gestión de memoria.

Utilizando el ajuste lineal $\mathrm{wpm}(L) \approx a\,L + b$ con $a = 0{,}001168$ y $b = 149{,}356$ (velocidad base), para $L = 100000$ palabras se tiene $\mathrm{wpm}(100000) \approx 0{,}001168 \times 100000 + 149{,}356 = 266{,}156$ palabras/min, por lo que el tiempo estimado es $T = \frac{100000}{266{,}156} \approx 375{,}8$ min.

Se puede estimar, por tanto, que para una novela de 100.000 palabras, el programa tardaría aproximadamente 6 h 16 min, un resultado que se considera dentro de lo esperado teniendo en cuenta la naturaleza secuencial de la Implementación (Sección 3.3).

4.2.2. Extensión frente a uso de memoria

Por otro lado, se realizaron las mediciones sobre cuanta memoria necesita el programa para ejecutarse en función de la extensión del texto.



Length (words)	Memory Peak (GB)
1731	17.58
3477	18.69
4900	19.836
9731	16.81
16461	16.81
22373	16.81
45631	16.81
66437	16.81

Figura 4.3: Datos de uso de memoria máximo según el número de capítulos procesados

Como se puede observar en la Figura 4.3 se consigue con éxito limitar el uso de memoria máximo, independientemente de la extensión del texto a procesar.

Gracias a la implementación (Sección 3.3), el mayor uso de memoria viene de cargar los modelos LLM cuando se necesitan en los módulos. El pico inicial en uso de memoria se debe a que, para textos de extensión reducida, se proporciona el análisis completo al LLM antes de crear los prompts para crear imágenes, dándole mayor contexto. Sin embargo, en cuanto pasa de cierta extensión, se omite esta inclusión del análisis y por lo tanto se estabiliza el uso de memoria a 16.81 GiB.

Esto, por tanto, supone la superación con éxito de uno de los requisitos principales (Sección 3.2), al conseguir un programa capaz de procesar un texto no estructurado de forma local con mínimo 24GiB de VRAM, independientemente de su longitud.

4.3. Evaluación de resultados

Al tratarse de un sistema capaz de generar imágenes a partir de textos no estructurados, se hace difícil realizar una evaluación de los resultados de forma totalmente objetiva y automática. No solo por la escasez de herramientas y métricas estandarizadas sino porque también, teniendo en cuenta las posibles aplicaciones mencionadas en la Introducción, es importante la evaluación humana de los resultados.

Con el fin de aportar diversidad a la muestra, para esta prueba se seleccionaron varios usuarios de distintos ámbitos o áreas de trabajo (Figura 4.4), quienes realizaron una encuesta evaluando la calidad, alineamiento con el texto, detalles y tiempo en generar de las imágenes, a partir de varios fragmentos

de un libro que ellos seleccionaron libremente. Se muestran también en el Anexo A - A.3, las preguntas y respuestas del formulario al completo.

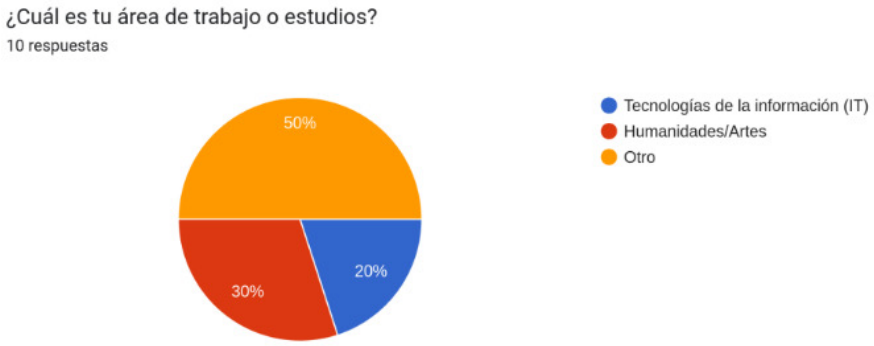


Figura 4.4: Datos del área de trabajo o estudios de los encuestados

Categoría	Puntuación media (/5)
Cumplimiento expectativas	4.7
Tiempo de respuesta	4.1
Calidad de imagen	5.0
Ajuste a preferencias	4.6
Coherencia narrativa	4.8
Nivel de detalle	4.6
Evaluación global	5.0

Tabla 4.3: Puntuación media por categoría en la evaluación de la aplicación.

La ejecución del pipeline se hizo en directo, presencialmente o por videollamada compartiendo pantalla. Para no superar los 30 minutos por sesión y obtener resultados relevantes, antes de cada reunión se seleccionaron manualmente fragmentos del libro relacionados con los personajes y escenarios que interesaban a los encuestados. Esto permitió un mayor nivel de detalle con menos tiempo de cómputo. Como se observa en la Tabla 4.3, los resultados de valoración fueron muy positivos. La calidad de imagen fue la mejor valorada y el tiempo de respuesta, la peor, con una media global de 4.69 sobre 5.

También fue interesante ver cómo cada encuestado centró su análisis según su especialidad. Los perfiles técnicos valoraron aspectos como el uso de código abierto o la ejecución local, comparando con modelos como DALL-E 3 o Stable Diffusion 3.5. El resto analizó más en detalle las imágenes, valorando la coherencia con el texto y la presencia o ausencia de ciertos elementos. En la Figura 4.5 se muestra una selección representativa; el resto puede consultarse en el Anexo B: Galería de imágenes.

Figura 4.5: Daenerys - Juego de tronos // Darrow - Amanecer Rojo // Vin - Nacidos de la Bruma // Biblioteca - La sombra del viento // Arrakis - Dune // Entrada al reino de la fantasía - Jerónimo Stilton

5

Conclusiones

5.1. Trabajo futuro

Desde su diseño inicial, durante la implementación, e incluso tras las pruebas y evaluación, este ha sido un proyecto ambicioso y dinámico. Algunos requisitos o incluso módulos enteros han evolucionado y cambiado según se avanzaba, haciendo elevado el tiempo que se le ha dedicado tanto a cada módulo, como al proyecto en su totalidad. Al final, se ha conseguido una versión funcional, además de ligera y escalable. Sin embargo, como se menciona y aprecia en la sección de Implementación (3.3), y el capítulo de Pruebas (4), hay mucho espacio para mejorar el pipeline.

Una de las mayores limitaciones es el tiempo de ejecución del proyecto (reflejado en las Pruebas de Funcionalidad, 4.1 y Evaluación de Resultados, 4.3). La implementación secuencial, si bien simple y fácilmente escalable, implica sucesivas llamadas a los modelos, lo cual se podría mejorar introduciendo elementos de paralelización o métodos de procesamiento en bloque, especialmente en la implementación del módulo de análisis del texto (Sección 3.3.2) y en el módulo de creación de prompts (Sección 3.3.4).

Por parte del uso de memoria, podría ser interesante aplicar cuantización (Sección 2.2.3) al LLM utilizado, ya que, como se explica en la implementación del módulo de generación de imágenes (Sección 3.3.5), se reduce la demanda de recursos computacionales sin sacrificar precisión en los resultados obtenidos de forma notable.

En cuanto a la mejora de los resultados, una idea sería añadir en el módulo de consolidación (Sección 3.3.3), una capa extra de coherencia, consultando una vez más al LLM sobre un resumen del fragmento para buscar fallos o llenar campos que no se completaron en el análisis inicial. Otro ejemplo sería hacer asociaciones entre personajes y escenarios en el análisis, facilitando así la creación de escenas más coherentes.

Además, proyectando hacia la creación de un software funcional, en un futuro se podría añadir a la interfaz (Sección 3.3.6) un chatbot en el que el usuario pudiese ver y modificar el análisis, los prompts y cualquiera de las salidas intermedias con el fin de ajustarlo mejor a sus expectativas.

En caso de llevar este proyecto a un SaaS en el futuro, será necesario negociar licencias comerciales con Meta [64] y Black Forest Labs [49], ya que el uso de sus modelos es libre para lo personal e investigación, pero se restringen a acuerdos los usos comerciales. Queda prohibido cualquier uso ilegal o dañino. Con respecto a los derechos de autor de las obras, los textos procesados de entrada no se guardan en ningún momento. Se avisaría al usuario en los Términos y Condiciones de uso de que es necesario contar con los derechos de la obra si se van a usar las imágenes generadas para fines comerciales o de distribución, delegando la responsabilidad legal en el usuario. Ayudaría también en este caso tener un centro de imágenes generadas con un sistema de *Notice and takedown* efectivo en casos de infracción.

5.2. Discusión final

Este trabajo aborda de forma transparente uno de los retos actuales del Procesamiento del Lenguaje Natural: la limitación en la extensión de los textos que pueden procesarse. Lejos de seguir la tendencia actual e intentar ampliar las capacidades de un único modelo o red neuronal, se propone una alternativa novedosa que supera dicha barrera mediante una arquitectura modular.

A partir de modelos, técnicas y herramientas de PLN, se han desarrollado módulos autónomos capaces de ejecutar tareas específicas que, combinados, permiten analizar textos no estructurados y generar imágenes, sin importar la longitud de la obra original.

Tal como se detalla en la implementación (Sección 3.3) y en las pruebas realizadas (Sección 4), el sistema resultante es *open-source*, escalable y competitivo en rendimiento. Todo ello se consigue manteniendo un código ligero, ejecutable localmente en hardware de usuario gracias al control sobre el uso de memoria. Además, el diseño escalable y resiliente facilita tanto la mejora continua de los resultados como la integración de nuevos modelos, evitando así la obsolescencia de la solución.

En conjunto, este trabajo combina avances y herramientas ya desarrolladas con un diseño e implementación propios para combinarlos de forma que superen ciertas limitaciones que son incapaces de sobrepasar individualmente. Sin embargo, esta arquitectura requiere un procesamiento intermedio considerable para suplir las contrapartidas de modelos menos avanzados y obtener resultados relevantes. Esto se ha traducido en un mayor tiempo de cómputo por cada llamada. Como se discute en la sección de Trabajo futuro (5.1), este aspecto es una de las principales áreas de mejora.

Paralelamente, el PLN y la IA avanzan hacia protocolos estándar (MCP), agentes autónomos y sistemas multiagente capaces de orquestar flujos sin intervención humana. Aunque la combinación de modelos y herramientas de este proyecto aporta gran flexibilidad, también introduce desafíos compartidos con las investigaciones más punteras: estandarizar la entrada/salida de LLMs, ajustar la intervención del desarrollador y reducir memoria y tiempo sin sacrificar rendimiento. En este trabajo se ha tratado de ofrecer un punto de vista alternativo para abordarlos.

Bibliografía

[1] IBM, "What is nlp (natural language processing)?." https://www.ibm.com/think/topics/natural-language-processing. Accessed: 05 May 2025.

[2] Zilliz, "What is the token limit in openai models?." https://zilliz.com/ai-faq/what-is-the-token-limit-in-openai-models. Accessed: 05 May 2025.

[3] M. Sai, "Open-source llms and their maximum input token capacities." https://medium.com/@saimoguloju2/open-source-large-language-models-and-their-maximum-input-token-capacit Accessed: 05 May 2025.

[4] Y. Hu and Y. Lu, "RAG and RAU: A survey on retrieval-augmented language model in natural language processing," *CoRR*, vol. abs/2404.19543, 2024.

[5] J. G. C. Ramírez, "Natural language processing advancements: Breaking barriers in human-computer interaction," *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, vol. 3, no. 1, pp. 31–39, 2024.

[6] M. Siino, I. Tinnirello, and M. L. Cascia, "Is text preprocessing still worth the time? a comparative survey on the influence of popular preprocessing methods on transformers and traditional classifiers," *Information Systems*, vol. 121, 3 2024.

[7] Z. Rasool, S. Kurniawan, S. Balugo, S. Barnett, R. Vasa, C. Chesser, B. M. Hampstead, S. Belleville, K. Mouzakis, and A. Bahar-Fuchs, "Evaluating llms on document-based qa: Exact answer selection and numerical extraction using cogtale dataset," *Natural Language Processing Journal*, vol. 8, p. 100083, 9 2024.

[8] F. A. Acheampong, H. Nunoo-Mensah, and W. Chen, "Transformer models for text-based emotion detection: a review of bert-based approaches," *Artif. Intell. Rev.*, vol. 54, no. 8, pp. 5789–5829, 2021.

[9] Z. Wang, Y. Huang, D. Song, L. Ma, and T. Zhang, "Promptcharm: Text-to-image generation through multi-modal prompting and refinement," in *Conference on Human Factors in Computing Systems - Proceedings*, Association for Computing Machinery, 5 2024.

[10] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net, 2022.

[11] G. Z. Higginbotham and N. S. Matthews, "Prompting and in-context learning: Optimizing prompts for mistral large," 5 2024.

[12] Y. Huang, J. Xu, Z. Jiang, J. Lai, Z. Li, Y. Yao, T. Chen, L. Yang, Z. Xin, and X. Ma, "Advancing transformer architecture in long-context large language models: A comprehensive survey," *CoRR*, vol. abs/2311.12351, 2023.

[13] A. Rahali and M. A. Akhloufi, "End-to-end transformer-based models in textual-based nlp," 3 2023.

[14] L. Roque, “The evolution of llama: From llama 1 to llama 3.1.” https://towardsdatascience.com/the-evolution-of-llama-from-llama-1-to-llama-3-1. Accessed: 2025-05-04.

[15] M. AI, “Huggingface meta-llama/llama-3.1-8b-instruct.” https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct. Accessed: 2025-05-04.

[16] K. Sam and R. Vavekanand, “Llama 3.1: An in-depth analysis of the next generation large language model llama 3.1: An in-depth analysis of the next-generation large language model.”

[17] M. AI, “Llama: Llama 3.1.” https://en.wikipedia.org/wiki/Llama_(language_model)#LLaMA_3.1. Publicado el 23 de julio de 2024. Accedido: 04 de mayo de 2025.

[18] M. AI, “Huggingface llama 3.1-405b, 70b and 8b with multilinguality and long context.” https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct. Accessed: 2025-05-04.

[19] T. D. Science, “Llama 3.1 models: 405b vs 70b vs 8b – which one to choose?.” https://towardsdatascience.com/llama-3-1-models-405b-vs-70b-vs-8b-which-one-to-choose. Accessed: 2025-05-04.

[20] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang, “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” *arXiv preprint arXiv:2501.12948*, 1 2025.

[21] D. D. Baek and M. Tegmark, “Towards understanding distilled reasoning models: A representational approach,” *arXiv preprint arXiv:2503.03730*, 3 2025.

[22] A. Iorliam and J. A. Ingio, “A comparative analysis of generative artificial intelligence tools for natural language processing,” *Journal of Computing Theories and Applications*, vol. 1, pp. 311–325, 2 2024.

[23] A. B. Yadav, “Generative ai in the era of transformers: Revolutionizing natural language processing

with llms," *Journal of Image Processing and Intelligent Remote Sensing*, pp. 54–61, 3 2024.

[24] SamLowe, "Samlowe/roberta-base-go_emotions hugging face." `https://huggingface.co/SamLowe/roberta-base-go_emotions`. Accessed: 2025-05-04.

[25] A. Stavropoulos, D. L. Crone, and I. Grossmann, "Shadows of wisdom: Classifying meta-cognitive and morally grounded narrative content via large language models," *Behavior Research Methods*, vol. 56, pp. 7632–7646, 10 2024.

[26] S. Jiang and M. Coblenz, "An analysis of the costs and benefits of autocomplete in ides," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 1284–1306, 2024.

[27] H. Face, "distilbert/distilbert-base-uncased-finetuned-sst-2-english · hugging face." `https://huggingface.co/distilbert/distilbert-base-uncased-finetuned-sst-2-english`. Accessed: 2025-05-04.

[28] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 10 2019.

[29] G. Lorenzoni, I. Portugal, P. Alencar, and D. Cowan, "Exploring variability in fine-tuned models for text classification with distilbert," *arXiv preprint arXiv:2501.00241*, 12 2024.

[30] H. Face, "sentence-transformers/paraphrase-mpnet-base-v2 · hugging face." `https://huggingface.co/sentence-transformers/paraphrase-mpnet-base-v2`. Accessed: 2025-05-04.

[31] D. S. C. Venkatachalapathi and C. P. Chavan, "Data-efficient training for effective paraphrase retrieval techniques using language models to identify research gaps," in *Lecture Notes in Networks and Systems*, vol. 1108 LNNS, pp. 97–106, Springer Science and Business Media Deutschland GmbH, 2024.

[32] E. L. Pontes, M. Benjannet, J. G. Moreno, and A. Doucet, "Using contextual sentence analysis models to recognize ESG concepts," *CoRR*, vol. abs/2207.01402, 2022.

[33] K. P. Juhong Chen, "Siamese network-based text similarity algorithm research," *Journal of Artificial Intelligence Practice*, vol. 7, 2024.

[34] P. Neculoiu, M. Versteegh, and M. Rotaru, "Learning text similarity with siamese recurrent networks," in *Proceedings of the 1st Workshop on Representation Learning for NLP, Rep4NLP@ACL 2016, Berlin, Germany, August 11, 2016* (P. Blunsom, K. Cho, S. B. Cohen, E. Grefenstette, K. M. Hermann, L. Rimell, J. Weston, and S. W. Yih, eds.), pp. 148–157, Association for Computational Linguistics, 2016.

[35] T. Ranasinghe, C. Orasan, and R. Mitkov, "Semantic textual similarity with siamese neural networks," in *International Conference Recent Advances in Natural Language Processing, RANLP*, vol. 2019-September, pp. 1004–1011, Incoma Ltd, 2019.

[36] N. Muennighoff, N. Tazi, L. Magne, N. Reimers, and C. Ai, "Mteb: Massive text embedding benchmark hugging face," *arXiv preprint arXiv:2210.07316*, 2014.

[37] J. Opitz, L. Möller, A. Michail, and S. Clematide, "Interpretable text embeddings and text similarity explanation: A primer," *arXiv preprint arXiv:2502.14862*, 2 2025.

[38] H. Steck, C. Ekanadham, and N. Kallus, "Is cosine-similarity of embeddings really about similarity?," in *Companion Proceedings of the ACM Web Conference 2024*, 3 2024.

[39] N. Pradhan, B. M. Gyanchandani, A. Professor, and B. R. Wadhvani, "A review on text similarity technique used in ir and its application," *International Journal of Computer Applications*, vol. 120, pp. 975–8887, 2015.

[40] E. Hoogeboom, J. Heek, and T. Salimans, "simple diffusion: End-to-end diffusion for high resolution images," in *International Conference on Machine Learning*, 2023.

[41] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

[42] K. Yoon, G. Gankhuyag, J. Park, H. Son, and K. Min, "Casr: Efficient cascade network structure with channel aligned method for 4k real-time single image super-resolution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7911–7920, 2024.

[43] Y. Balaji, S. Nah, X. Huang, A. Vahdat, J. Song, Q. Zhang, K. Kreis, M. Aittala, T. Aila, S. Laine, *et al.*, "ediff-i: Text-to-image diffusion models with an ensemble of expert denoisers," *arXiv preprint arXiv:2211.01324*, 2022.

[44] Z. Patel, J. DeLoye, and L. Mathias, "Exploring diffusion and flow matching under generator matching," *arXiv preprint arXiv:2412.11024*, 12 2024.

[45] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, 2020.

[46] J. S. Fischer, M. Gui, P. Ma, N. Stracke, S. A. Baumann, V. T. Hu, and B. Ommer, "Fmboost: Boosting latent diffusion with flow matching," in *European Conference on Computer Vision*, 2024.

[47] S. Ren, Q. Yu, J. He, X. Shen, A. Yuille, and L.-C. Chen, "Flowar: Scale-wise autoregressive image generation meets flow matching," *arXiv preprint arXiv:2412.15205*, 12 2024.

[48] B. F. Labs, "black-forest-labs/flux: Official inference repo for flux.1 models." `https://github.com/black-forest-labs/flux`. Accessed: 2025-05-04.

[49] B. F. Labs, "Black forest labs – frontier ai lab." `https://bfl.ai/`. Accessed: 05 May 2025.

[50] S. Cakic, T. Popovic, S. Krco, I. Jovovic, and D. Babic, "Evaluating the flux.1 synthetic data on yolov9 for ai-powered poultry farming," *Applied Sciences (Switzerland)*, vol. 15, 4 2025.

[51] V. Egiazarian, D. Kuznedelev, A. Voronov, R. Svirschevski, M. Goin, D. Pavlov, D. Alistarh, and D. Baranchuk, "Accurate compression of text-to-image diffusion models via vector quantization," *arXiv preprint arXiv:2409.00492*, 8 2024.

[52] C. Yang, C. Liu, X. Deng, D. Kim, X. Mei, X. Shen, and L. Chen, "1.58-bit FLUX," *CoRR*, vol. abs/2412.18653, 2024.

[53] S. Z. Zhu), "Long prompt weighted stable diffusion embedding." `https://github.com/xhinker/sd_embed`, 2024.

[54] J. Jones, W. Jiang, N. Synovic, G. Thiruvathukal, and J. Davis, "What do we know about hugging face? a systematic literature review and quantitative validation of qualitative claims," in *Proceedings*

of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 13–24, 6 2024.

[55] A. Ait, J. L. C. Izquierdo, and J. Cabot, "Hfcommunity: A tool to analyze the hugging face hub community," in *2023 IEEE international conference on software analysis, evolution and reengineering (SANER)*, pp. 728–732, IEEE, 2023.

[56] J. Zhao, S. Wang, Y. Zhao, X. Hou, K. Wang, P. Gao, Y. Zhang, C. Wei, and H. Wang, "Models are codes: Towards measuring malicious code poisoning attacks on pre-trained model hubs," *arXiv preprint arXiv:2409.09368*, 2024. Accedido: 04 de mayo de 2025.

[57] C. Osborne, J. Ding, and H. R. Kirk, "The ai community building the future? a quantitative analysis of development activity on hugging face hub," *arXiv preprint arXiv:2405.13058*, 2024. Accedido: 04 de mayo de 2025.

[58] S. Vijayarani and M. P. R. Scholar, "Preprocessing techniques for text mining-an overview," tech. rep., Bharathiar University, Coimbatore, Tamilnadu, India, 2015.

[59] A. Tabassum and R. R. Patil, "A survey on text pre-processing & feature extraction techniques in natural language processing," *International Research Journal of Engineering and Technology*, 2020.

[60] M. A. Palomino and F. Aider, "Evaluating the effectiveness of text pre-processing in sentiment analysis," *Applied Sciences (Switzerland)*, vol. 12, 9 2022.

[61] M. Honnibal and I. Montani, "spaCy: Industrial-strength natural language processing in python." Último acceso: 05 mayo 2025.

[62] Explosion, "en_core_web_sm — small english pipeline for spacy." `https://github.com/explosion/spacy-models/releases/tag/en_core_web_sm-3.8.0`. Último acceso: 05 mayo 2025.

[63] X. Rong, "word2vec parameter learning explained," *CoRR*, vol. abs/1411.2738, 2014.

[64] Meta, "Company information, culture, and principles." `https://www.meta.com/es-es/about/company-info/`. Accessed: 05 May 2025.

Apéndices

A

Detalles de las pruebas

A.1. Fragmento preprocesado completo

Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you would expect to be involved in anything strange or mysterious, because they just did not hold with such nonsense. Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large mustache. Mrs. Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbors. The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere. The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They did not think they could bear it if anyone found out about the Potters. Mrs. Potter was Mrs. Dursley's sister, but they had not met for several years; in fact, Mrs. Dursley pretended she did not have a sister, because her sister and her good-for-nothing husband were as unDursleyish as it was possible to be. The Dursleys shuddered to think what the neighbors would say if the Potters arrived in the street. The Dursleys knew that the Potters had a small son, too, but they had never even seen him. This boy was another good reason for keeping the Potters away—they did not want Dudley mixing with a child like that. When Mr. and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there was nothing about the cloudy sky outside to suggest that strange and mysterious things would soon be happening all over the country. Mr. Dursley hummed as he picked out his most boring tie for work, and Mrs. Dursley gossiped away happily as she wrestled a screaming Dudley into his high chair. None of them noticed a large, tawny owl flutter past the window. At half past eight, Mr. Dursley picked up his briefcase, pecked Mrs. Dursley on the cheek, and tried to kiss Dudley goodbye but missed, because Dudley was now having a tantrum and throwing his cereal at the walls.

Cuadro A.1: Fragmento preprocesado completo del capítulo 1

A.2. JSONs completos

Código A.1: Escenarios del JSON del análisis inicial


```
{
    "Historical": {
        "Year": [],
        "Period_Age": [
            "MODERN/POST-WWII",
            "MODERN/20TH CENTURY",
            "MODERN/EARLY 20TH CENTURY UNKNOWN",
            "MODERN/UNKNOWN"
        ],
        "Tense": ""
    },
    "Scenes": {
        "Time_passed": [
            "9:00 AM -10:30 AM (approx"
        ],
        "Scenarios": {
            "Scenarios_list": [
                {
                    "Real": {
                        "Value": "FICTIONAL",
                        "Coordinates": ""
                    },
                    "Name": "Number 4 PRIVET DRIVE",
                    "Place": {
                        "Type": "House",
                        "Outdoors": "NO"
                    },
                    "Time_of_day": "HALF PAST EIGHT",
                    "Weather": "GRAY AND CLOUDY",
                    "Other_details": "DURSLEYS LIVED THERE"
                },
                {
                    "Real": {
                        "Value": "FICTIONAL",
                        "Coordinates": ""
                    },
                    "Name": "Daily life in England",
                    "Place": {
```

Código A.2: Escenarios del JSON del análisis inicial


```
                "Type": "",
                "Outdoors": "NO"
            },
            "Time_of_day": "Morning",
            "Weather": "",
            "Other_details": ""
        },
        {
            "Real": {
                "Value": "FICTIONAL",
                "Coordinates": ""
            },
            "Name": "A Muggle's daily life",
            "Place": {
                "Type": "WORKPLACE/OFFICE",
                "Outdoors": "NO"
            },
            "Time_of_day": "LUNCHTIME, FIVE OF THE CLOCK",
            "Weather": "",
            "Other_details": ""
        },
        {
            "Real": {
                "Value": "Fictional",
                "Coordinates": ""
            },
            "Name": "Number 4 driveway",
            "Place": {
                "Type": "Residential",
                "Outdoors": "YES"
            },
            "Time_of_day": "DAYTIME",
            "Weather": "",
            "Other_details": ""
        }
    ],
    "Number_of_scenarios": 4

},
"Actions": [],
"Movement_between_scenarios": [
    "YES",
    "YES",
    "YES",
    "YES"
]
```

Código A.3: Escenario del JSON del análisis inicial


```
"Historical": {
    "Year": [],
    "Period_Age": "MODERN/EARLY 20TH CENTURY UNKNOWN",
    "Tense": ""
},
"Scenes": {
    "Time_passed": [
        "9:00 AM -10:30 AM (approx"
    ],
    "Scenarios": {
        "Scenarios_list": [
            {
                "Real": {
                    "Value": "FICTIONAL",
                    "Coordinates": ""
                },
                "Name": "Number 4 PRIVET DRIVE",
                "Place": {
                    "Type": [
                        "House",
                        "Residential"
                    ],
                    "Outdoors": "YES"
                },
                "Time_of_day": [
                    "HALF PAST EIGHT",
                    "DAYTIME"
                ],
                "Weather": "GRAY AND CLOUDY",
                "Other_details": [
                    "DURSLEYS LIVED THERE",
                    "Driveway"
                ]
            },
            {
                "Real": {
                    "Value": "FICTIONAL",
                    "Coordinates": ""
                },
                "Name": "A Muggle's daily life",
                "Place": {
                    "Type": "WORKPLACE/OFFICE",
                    "Outdoors": "NO"
                },
                "Time_of_day": "LUNCHTIME, FIVE OF THE CLOCK",
                "Weather": "",
                "Other_details": "ENGLAND"
            }
        ],
        "Number_of_scenarios": 2
    },
```

A.3. Formulario y respuestas completas

ID	Pregunta	Tipo
Q1	¿Qué historia o libro has elegido para generar las imágenes?	Abierta
Q4	¿Cuál es tu área de trabajo o estudios?	Opción múltiple
Q5	¿En qué medida la aplicación ha cumplido con tus expectativas generales (tiempo, calidad, preferencias, etc.)?	Valoración (1–5)
Q6	¿Qué tan satisfecho/a estás con el tiempo que ha tardado la aplicación en generar las imágenes?	Valoración (1–5)
Q7	¿Cómo calificarías la calidad general de las imágenes generadas?	Valoración (1–5)
Q8	¿En qué medida las imágenes se ajustan a las preferencias artísticas seleccionadas?	Valoración (1–5)
Q9	¿Qué tan coherentes son las imágenes con el contenido, la narrativa y el tono de la historia?	Valoración (1–5)
Q10	¿En qué medida las imágenes reflejan detalles significativos que esperabas ver?	Valoración (1–5)
Q11	Basándote en tu experiencia y resultado final, ¿cómo calificarías la aplicación en general?	Valoración (1–5)
Q2	¿Hay algo que te gustaría resaltar porque te haya sorprendido?	Abierta
Q3	¿Hay algo que hayas echado en falta o tienes alguna propuesta de mejora?	Abierta

Tabla A.1: Listado de preguntas con identificadores y tipo de respuesta

Q1	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Mistborn	Otro	5	5	5	5	5	5	5
Memorias de África	Otro	5	5	5	5	5	5	5
Amanecer rojo	Otro	5	5	5	5	5	5	5
La sombra del viento	Otro	5	4	5	5	5	4	5
Geronimo Stilton	Tecnologías de la información	4	4	5	5	5	4	5
Alas de sangre	Humanidades/Artes	5	4	5	5	5	5	5
Dune	Tecnologías de la información	4	3	5	4	4	4	5
El problema de los tres cuerpos	Otro	5	5	5	4	4	4	5
Harry Potter	Humanidades/Artes	4	3	5	4	5	5	5
Juego de tronos	Humanidades/Artes	5	3	5	4	5	5	5

Tabla A.2: Respuestas de evaluadores a Q1 y Q4–Q11

Evaluador	Q2: ¿Algo sorprendente?	Q3: Propuestas de mejora
Evaluador 1	La calidad de las imágenes realistas.	El personaje principal es rubio y no dejaba ponerlo rubio.
Evaluador 2	Las imágenes generadas representan muy bien la historia y son muy realistas.	—
Evaluador 3	La calidad de las imágenes ha sido increíble.	—
Evaluador 4	—	—
Evaluador 5	—	—
Evaluador 6	La precisión de los detalles de la imagen con respecto al texto.	—
Evaluador 7	Me ha sorprendido lo bien representado que las escenas del libro y la calidad que tienen las imágenes.	—
Evaluador 8	Me han sorprendido mucho ciertos detalles, como la foto del personaje femenino Ye Wenjie, donde se puede ver que el personaje se encuentra en un entorno parecido a un laboratorio, que es donde se encontraba ese personaje a esa edad (ya que el personaje a lo largo de la historia aparece en distintos contextos y distintas edades, y puede dar pie a mucha confusión para una IA), lo buenos que son los diseños de los personajes en general, y como se ajustan sus expresiones faciales a sus personalidades (Wang parece tranquilo y desenfadado en las dos fotos, tal y como se le describe en el libro).	Quizás la elección de los escenarios del libro no ha sido la mejor, aunque también es cierto que es un libro con bastantes pocas descripciones y los escenarios no suelen ser muy relevantes en cuanto a su diseño, el autor deja bastante a la imaginación del lector.
Evaluador 9	Detalles muy específicos estaban presentes.	Quizás estaría bien tener a los personajes en sus escenarios específicos.
Evaluador 10	—	—

Tabla A.3: Respuestas de evaluadores a Q2 y Q3

B

Galería de resultados

B.1. Dune

B.1.1. Escenarios

Figura B.1: Arrakeen // Arrakeen (vista lateral) // Arrakis

B.1.2. Personajes

Figura B.2: Leto Atreides // Paul Atreides (anime) // Paul Atreides

B.2. Alas de Sangre

B.2.1. Escenarios

Figura B.3: Parapeto // Basgiath // Tairn

B.2.2. Personajes

Figura B.4: Violet (versión 1) // Violet (versión 2) // Xaden (versión 1) // Xaden (versión 2)

B.3. Geronimo Stilton: Viaje al reino de la fantasía

B.3.1. Escenarios

Figura B.5: Cueva de cristales // Bosque encantado // Entrada al reino de la fantasía

B.3.2. Personajes

Figura B.6: Geronimo Stilton // Reina de las hadas // Búho mensajero

B.4. Juego de tronos

B.4.1. Escenarios

Figura B.7: Muro (vista 1) // Muro (vista 2)

B.4.2. Personajes

Figura B.8: Daenerys Targaryen // Gared // Exploradores de la Guardia de la noche

B.5. La sombra del viento

B.5.1. Escenarios

Figura B.9: Biblioteca // Escenario inicial // Escritorio

B.6. Memorias de África

Figura B.10: Sabana // Habitante de Kenia // Denys

B.7. Nacidos de la Bruma

B.7.1. Escenarios

Figura B.11: Luthadel // Luthadel (vista 2)

B.7.2. Personajes

Figura B.12: Kelsier // Vin // Vin (anime)

B.8. Amanecer Rojo

B.8.1. Escenarios

Figura B.13: Jardines de Marte // Casa Marte

B.8.2. Personajes

Figura B.14: Darrow // Eo // Darrow (dorado) // Cassius // Jackal // Mustang // Sevro

B.9. El problema de los tres cuerpos

B.9.1. Escenarios

Figura B.15: Desglose // Infierno // Trisolariano

B.9.2. Personajes

Figura B.16: Wang Miao // Ye Wenjie

B.10. Harry Potter

B.10.1. Escenarios

Figura B.17: Hogwarts // Private Drive 4

B.10.2. Personajes

Figura B.18: Harry Potter // Hermione Granger // Albus Dumbledore // Weasley


UÁM
Universidad Autónoma
de Madrid