.UBAfiuba
FACULTAD DE INGENIERÍA

Aplicación de modelos extensos de lenguaje en la ambientación narrativa

Ing. Mario Gómez Alonso

Carrera de Especialización en Inteligencia Artificial

Director: Bach. Josselyn Sofía Ordóñez Olazábal

Jurados:

Jurado 1 (pertenencia)
Jurado 2 (pertenencia)
Jurado 3 (pertenencia)

Ciudad de Madrid, España, junio de 2025

Resumen

En la presente memoria se describe el diseño e implementación de un prototipo que emplea modelos extensos de lenguaje para la asistencia de los creadores en la generación de contenido narrativo. El trabajo se realizó para Critical Match como parte de la expansión de sus servicios a los usuarios.

Para la implementación fueron imprescindibles conocimientos relacionados al procesamiento de lenguaje natural y modelos extensos de lenguaje.

Agradecimientos

Esta sección es para agradecimientos personales y es totalmente **OPCIONAL**.

Índice general

Índice de figuras

Índice de tablas

Dedicado a... [OPCIONAL]

Capítulo 1

Introducción general

En este capítulo se presenta el contexto general y la motivación del trabajo realizado. A su vez, se realiza un análisis del estado del arte relacionado con las tecnologías que se utilizaron y se establecen los objetivos y el alcance definidos durante su realización.

1.1. El desafío de la creación narrativa

La narrativa [1] es un género literario que relata un conjunto de sucesos protagonizados por uno o más personajes y que son presentados a través de un narrador. Se trata de una forma de expresión cultural fundamental, presente en todas las culturas y épocas, utilizado para entretener y transmitir conocimiento. Una de sus características esenciales es la presencia de elementos ficticios, ya sea de manera parcial o total, con la única excepción del subgénero de crónicas que se limita a relatar hechos reales. La narrativa sigue siendo un elemento central de la cultura y está presente en múltiples formatos como la literatura, el cine, la televisión y otros medios tanto físicos como digitales.

La inclusión de elementos ficticios en la narrativa es un proceso creativo en el que el autor recurre a su imaginación para concebir componentes que, aunque inventados, resulten verosímiles y significativos. En este ejercicio mental intervienen múltiples factores, como la coherencia interna y la relación entre los distintos elementos que conforman la historia en el espacio y el tiempo. Estas variables se vuelven especialmente complejas en narrativas con una alta carga ficticia donde se inventan, además de los personajes y sucesos, contextos completos. Aquí es donde cobra especial relevancia la construcción de mundos [2], un recurso narrativo fundamental que permite al escritor diseñar entornos imaginarios detallados y capaces de sostener la lógica del relato.

El proceso de creación de mundos es tan complejo como la narrativa que busca sustentar y la intención del autor de sumergir al lector dentro de la historia. En la figura 1.1 se presenta una lista amplia, aunque no completa, de componentes fundamentales en la construcción de mundos y la manera en que estos se relacionan entre sí para dar cohesión al conjunto narrativo.

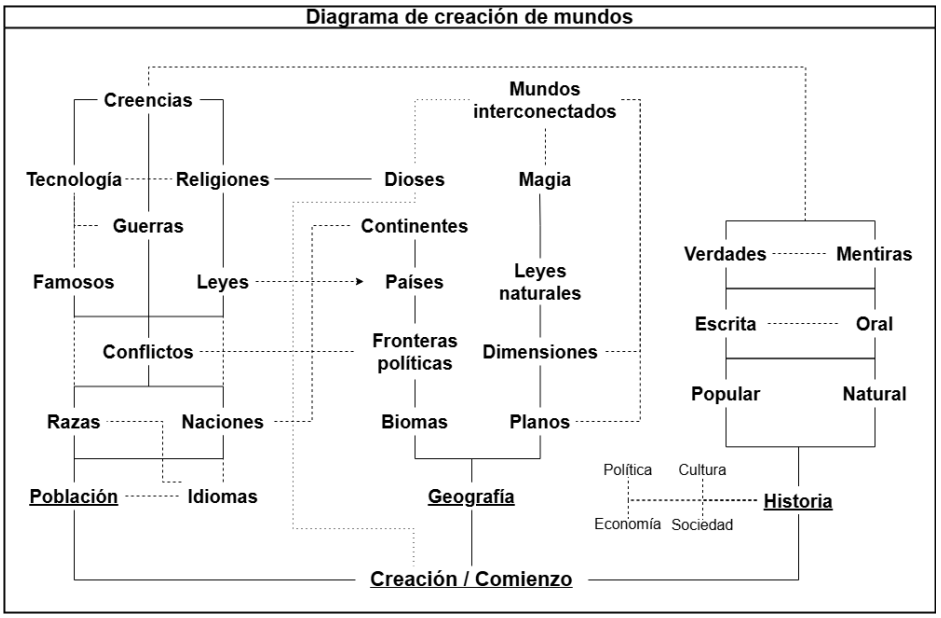

FIGURA 1.1. Esquema de elementos habituales en la construcción de mundos.

Históricamente los autores han recurrido a diversas herramientas para llevar a cabo la construcción de mundos, tales como esquemas, mapas, cronologías, fichas de personajes y la creación de lenguas artificiales [3]. Todos estos elementos han sido producto del trabajo creativo del autor, apoyado tanto en su imaginación como en la consulta de múltiples referencias. Este proceso ha requerido una considerable inversión de tiempo y, en muchas ocasiones, afectado por bloqueos creativos a la hora de articular y cohesionar todos los componentes ficticios de la narrativa.

En los últimos años la aparición de la inteligencia artificial generativa ha abierto nuevas posibilidades dentro de los entornos creativos. Aunque originalmente no fue concebida con fines narrativos, su capacidad para producir texto coherente la convierte en una herramienta con un alto potencial para incorporarse en la construcción de mundos ficticios. Sin pretender sustituir la creatividad del autor, estas tecnologías pueden ejercer un papel de apoyo al facilitar el desarrollo de contenidos y aportar ideas que estimulen la creación literaria.

1.2. Motivación

El trabajo se enfocó en la creación de entornos narrativos para juegos de rol y surgió de la colaboración con la empresa Critical Match [4] y su objetivo de ampliar las herramientas y servicios para los usuarios de su aplicación de móvil. La *app* [5] permite la creación y búsqueda de salas de juego que se ajusten a las necesidades y preferencias de cada usuario. En ese contexto, se busca mejorar la ambientación de las partidas proporcionando al autor de la narrativa a una serie de servicios para enriquecer el universo narrativo ya creado.

El trabajo aborda la necesidad de los usuarios de conseguir partidas más inmersivas y reducir el trabajo de ambientación del dueño de la sala de juego. Reconociendo que no todos los jugadores tienen experiencia con la creación narrativa, esta herramienta facilita la generación de una ambientación atractiva. De este modo se optimiza el tiempo de preparación del director de juego, permitiéndole centrarse en otros aspectos de la partida. Esto también beneficia a los más experimentados al expandir sus ideas con mayor rapidez y eficacia.

Los servicios implementados en el trabajo pretenden ofrecer una experiencia claramente diferenciada de las alternativas más populares en internet como ChatGPT [6], Gemini [7] o Meta AI [8]. Para ello, se proporciona a los usuarios el acceso a consultas especializadas que solo requieren el envío de un fichero con la información narrativa, pudiendo combinarse adicionalmente con indicaciones que pueden escribirse en un cuadro de texto. Este enfoque no solo evita la incomodidad de interactuar con el modelo a través del navegador de internet del móvil, sino que ofrece respuestas precisas a sus necesidades, que requerirían interacciones complejas con las alternativas existentes.

Además, este desarrollo se complementa a herramientas existentes de creación de mundos como WorldAnvil [9], Obsidian Portal [10] o Legend Keeper [11]. Mientras que estas plataformas brindan la estructura y el marco para la creación de mundos, el trabajo ofrece la capacidad de generar dinámicamente contenido narrativo específico con la información ya existente. De este modo los autores a través de ambas herramientas pueden centrarse en la visión general y cohesión del mundo mientras que la inteligencia artificial los ayuda con ideas con las que expandir la ambientación, lo que fomenta un proceso cíclico creativo.

1.3. Estado del arte

El avance de la inteligencia artificial generativa ha seguido un ritmo extraordinario desde la publicación del influyente artículo *Attention is All you need* [12] en 2017. La arquitectura de los transformadores revolucionó el procesamiento de lenguaje en las máquinas y permitió paralelizar los procesos a través del mecanismo de atención. Esto aumenta drásticamente la velocidad y eficacia en el proceso de aprendizaje de los modelos.

En el transcurso del año siguiente surgieron los primeros modelos extensos de lenguaje, en inglés *Large Language Models* (LLM), que adoptaron la arquitectura de los transformadores. Con el lanzamiento de GPT [13] por OpenAI y de BERT [14] por Google, ambos modelos se convirtieron en los pioneros y pilares fundamentales de los modelos extensos de lenguaje. Mientras BERT se especializó en la comprensión del contexto del texto de manera bidireccional, GPT fue desarrollado para la generación de texto.

Desde entonces la evolución de los modelos no se ha detenido. El número de parámetros ha crecido exponencialmente, lo que les ha otorgado una capacidad sin precedentes para comprender y producir texto coherente y fluido similar al humano. Además, este desarrollo ha sido transversal y está dotando a los modelos más recientes con la habilidad para generar no solo texto, sino también imágenes, audio e incluso video.

Estos avances están actualmente al alcance del público general a través de internet, con ejemplos destacados como ChatGPT [6], Gemini [7] o Meta AI [8].

ChatGPT fue el pionero y el que abarcó mayor popularidad, con una interfaz gráfica similar a la de un chat con la que el usuario podía interactuar de forma intuitiva y sencilla con el modelo. La utilización de estos modelos se masificó rápidamente y se implementaron variantes en multitud de sectores, principalmente en forma de asistentes virtuales.

En el ámbito creativo, el potencial de la inteligencia artificial generativa es tan amplio como discutido. Dependiendo de su aplicación, los modelos pueden ser catalizadores del proceso creativo al ayudar a superar bloqueos y explorar nuevas ideas con rapidez, o bien reemplazar por completo la labor del artista. Este dilema plantea importantes debates sobre la autoría, los derechos de propiedad intelectual y el futuro de diversas profesiones creativas. Actualmente, la relación entre la inteligencia artificial y los artistas se encuentra en un proceso de redefinición de su paradigma.

A pesar de sus impresionantes capacidades, los modelos generativos se enfrentan a retos muy significativos que se enumeran a continuación:

- Sensibilidad a los *prompts*: los modelos generativos son muy dependientes de los datos de entrada. Pequeñas variaciones en la información inicial pueden llevar a cambios drásticos en el resultado generado, lo que afecta la precisión de las respuestas y hace que sean menos fiables para usuarios sin experiencia previa.
- Alucinaciones e inconsistencias a largo plazo: persisten las dificultades para mantener la coherencia y consistencia en textos extensos debido a la capacidad finita de su memoria (ventana de contexto). También existe la posibilidad de que la salida se vuelva incongruente o que se genere información ficticia, debido a factores internos o a la complejidad de la solicitud.
- Sesgos en los datos de entrenamiento: los modelos pueden perpetuar o amplificar prejuicios sociales presentes en los datos con los que fueron entrenados.

Para mitigar estos problemas se utilizan las siguientes tecnicas:

- Generación aumentada por recuperación: esta técnica permite a los modelos consultar fuentes de datos externas y ampliar la información de la entrada con elementos que están fuera de su entrenamiento inicial o de su ventana de contexto inmediata. Esto reduce las alucinaciones y mejora la verosimilitud de la respuesta.
- Filtrado de datos de entrenamiento: se implementan procesos más rigurosos para limpiar y despolarizar los conjuntos de datos de entrenamiento, para minimizar la presencia de sesgos.
- *Fine-tuning* [14]: consiste en la adaptación de los modelos a tareas específicas a través de un proceso de reentrenamiento parcial.
- *Prompt engineering* [15]: se basa en la formulación precisa de instrucciones en la entrada para guiar la generación de la respuesta y obtener mejores resultados.

Este trabajo se fundamenta en este marco de conocimiento y se emplearon varias técnicas expuestas para mitigar el impacto de las limitaciones actuales de los modelos generativos.

1.4. Objetivos y alcance

El propósito del trabajo es desplegar un componente que aloje los modelos extensos de lenguaje con los que generar contenido narrativo a partir de una entrada. Este servidor también es capaz de administrar los modelos, añadir nuevos y elegir cuáles estarán activos.

A su vez, se dispone de un simple servidor web que actúa como la interfaz gráfica entre el usuario y la inteligencia artificial. Su función es recibir y mostrar la información al usuario, procesar sus instrucciones y el archivo adjunto y transformarlo en una instrucción personalizada según el servicio de generación narrativa solicitado. Esta instrucción es enviada al modelo de inteligencia artificial generativa, y su respuesta se reenvía al usuario.

Ambos servidores conforman un prototipo de pruebas que se entregará al cliente, junto con este documento, con el propósito de funcionar como estudio inicial y como base para futuros desarrollos dentro de su aplicación.

El alcance del trabajo incluye los siguientes elementos:

- Desarrollo de un servidor web.
- Descripción de los modelos extensos de lenguaje utilizados.
- Definición, implementación y descripción del *prompt engineering* para cada servicio.
- La integración de un conjunto esencial de servicios REST.

El alcance del trabajo no incluye:

- Diseño de la página que cumpla con los estándares más habituales.
- Securización del servidor web ni de ninguno de sus servicios REST.
- Rendimiento de los servicios que garanticen tiempos de respuesta cercanos a tiempo real.

Capítulo 2

Introducción específica

En este capítulo se profundiza en aquellos aspectos clave para el desarrollo de este trabajo. En primer lugar, se presentan los requerimientos de sistema y se amplía la información acerca de los modelos de inteligencia artificial que se han utilizado. Finalmente, se expone el conjunto de técnicas y herramientas que se han aplicado a dichos modelos.

2.1. Requerimientos del sistema

Para llevar a cabo este trabajo se identificaron una serie de requerimientos fundamentales. A continuación, se listan aquellos que están directamente relacionados con la implementación:

1. Requerimientos del servidor:
    a) El servidor debe alojar y administrar la información relativa al *dataset* y la configuración del modelo LLM.
    b) El servidor debe contar con los *prompts* necesarios para especializar la respuesta de la inteligencia artificial.
    c) Al ser desplegado, el servidor deberá acceder al módulo LLM y utilizarlo en el procesamiento de peticiones entrantes.
    d) El servidor debe dar acceso a clientes externos a través del protocolo API REST.
    e) Los servicios REST deben aceptar entrada de texto en varios formatos: pdf, txt, docx o texto plano en el cuerpo de la petición.
    f) Los servicios REST devolverán la respuesta en formato simple HTML para su cómoda visualización en un navegador.
    g) El prototipo del servidor debe de tener una disponibilidad del 100 % durante la demostración.
2. Requerimientos del módulo LLM:
    a) El módulo LLM debe aceptar entrada de texto y generar texto como salida.
    b) En caso de que el texto de entrada sea legible, el módulo LLM debe aportar una respuesta con un detalle y profundidad razonables, además de ser coherente con las instrucciones recibidas.

c) El módulo se ajustará a los *prompts* recibidos para que, con el mismo contexto, devuelva información enfocada en un aspecto específico de la narrativa.

d) El tiempo de respuesta del módulo LLM debe estar en un rango de tiempo razonable para un servicio REST (no más de 5 minutos).

2.2. Modelos extensos de lenguaje

Para abordar los requisitos de generación de texto resultó fundamental la utilización de modelos extensos de lenguaje, comúnmente denominados *Large Language Models* (LLM). Estos modelos son entrenados sobre enormes volúmenes de datos textuales para especializarlos en predecir la secuencia de texto más probable dada una entrada previa. Esta habilidad les permite generar texto de forma coherente con el contexto al adaptarse al tono, estilo y contenido esperado.

Los LLM se basan en redes neuronales profundas que manejan miles de millones de parámetros (lo que justifica su clasificación como modelos "extensos") para identificar patrones complejos del lenguaje natural [12]. Estos modelos utilizan *tokens* como unidad básica de texto en su procesamiento. Los *tokens* pueden representar palabras completas, fragmentos de palabras o signos de puntuación, dependiendo del sistema de tokenización utilizado. Estos *tokens* se convierten posteriormente en vectores numéricos mediante una capa de *embedding* [16], que permite al modelo operar sobre ellos.

Los LLM modernos utilizan la arquitectura de *transformers* [12], un tipo de red neuronal que permite procesar secuencias de texto en paralelo y asignar diferentes niveles de relevancia (atención) a distintas partes del texto mediante un mecanismo llamado *self-attention* [12]. Esta arquitectura supera notablemente a estructuras anteriores como *Long-Short Term Memory* [17] o *Gated Recurrent Unit* [18] en cuestiones de eficiencia y rendimiento.

Además de su capacidad de paralelización y memoria a largo plazo, los LLM basados en *transformers* operan mediante la predicción de la siguiente cadena de *tokens* a partir del contexto previo. En este proceso hay elementos que controlan la generación, entre los que se destaca la temperatura [19]. La temperatura regula la aleatoriedad de los *tokens*: valores bajos tienden a generar respuestas deterministas, mientras que valores altos proporcionan respuestas más diversas y creativas. Esto también aumenta las probabilidades de que esta respuesta sea incoherente, también llamada alucinación [20]. Existen otros parámetros que controlan la variedad del texto, por ejemplo *top-k* y *top-p* [21].

A medida que los LLM aumentan en extensión de parámetros y profundidad de entrenamiento, empiezan a manifestar cualidades que no fueron explícitamente programadas. Entre estas capacidades destacan el razonamiento en múltiples pasos, traducción automática entre idiomas, capacidad de responder a preguntas complejas y generación creativa y coherente de texto que va más allá de la simple predicción de cadenas de texto.

Por ejemplo, modelos como GPT-3 [15] de OpenAI, PaLM [22] de Google y LLaMA [23] de Meta han demostrado un rendimiento notable en multitud de tareas

complejas como realizar inferencias lógicas, resolver problemas matemáticos, resumir textos extensos y generar código a partir de descripciones en lenguaje natural. Esto no solo amplía el espectro de aplicaciones prácticas de los LLM, sino que también plantea nuevas líneas de investigación para comprender cómo el tamaño y la calidad del entrenamiento impactan en la adquisición de habilidades cognitivas sofisticadas.

2.3. Ingeniería de *prompts*

La ingeniería de *prompts* (*prompt engineering*) es una técnica fundamental en la optimización de la salida de los modelos extensos de lenguaje que se basa en guiar el proceso de razonamiento del modelo hacia un objetivo específico a través de un conjunto de entradas textuales. A diferencia de la programación tradicional, que se expresa de forma explícita la lógica mediante el uso de código en un lenguaje de programación, el comportamiento de los LLM se guía mediante lenguaje natural.

Esta técnica ha emergido como una nueva disciplina en la que intervienen conceptos linguísticos y computacionales. La forma en la que se redacta una instrucción puede influir de forma notable en la coherencia, relevancia, creatividad y precisión de las respuestas.

Entre las estrategias comunes de ingeniería de prompts se encuentran *zero/few-shot prompting* [24][25], que emplea ejemplos en el *prompt* para guiar la generación; *chain of thought prompting* [26] invita al modelo a razonar explícitamente los pasos intermedios antes de llegar a la conclusión; o *prompt chaining* [27] que invita al modelo a realizar un análisis previo sobre el contexto o instrucciones para enfocar la salida. En la figura 2.1 se ilustra cómo estas estrategias de instrucción afectan directamente la salida generada por el modelo. Cabe destacar que las distintas técnicas de *prompting* descritas no son excluyentes entre sí, sino que pueden combinarse de manera complementaria, lo que potencia la capacidad de razonamiento y la precisión de los LLM en tareas complejas.

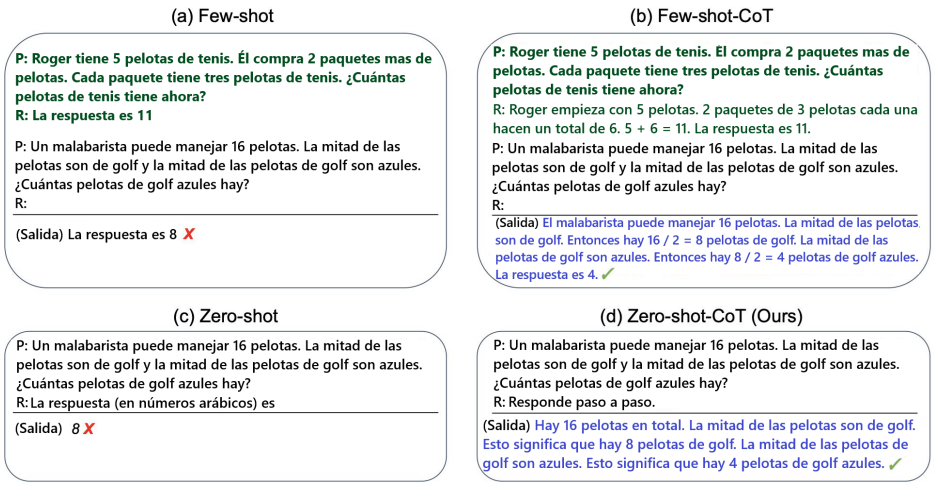

FIGURA 2.1. Ejemplo de uso de técnicas de *prompting*.

A su vez, la ingeniería de *prompts* ha demostrado ser fundamental en aquellos contextos en los que el *fine-tuning* no es viable, ya que hay muchos modelos cerrados que solo son accesibles a través de una API y la capacidad de modificar su comportamiento sin reentrenamiento se vuelve crucial. Por lo tanto, esta técnica se convierte en una herramienta práctica, eficiente y cada vez más sofisticada para adaptar modelos generalistas a tareas específicas.

2.4. Ajuste fino

El ajuste fino (*fine-tuning*) [28] es una técnica de entrenamiento de modelos extensos de lenguaje que permite adaptar un modelo previamente entrenado a una tarea en específico. A diferencia del entrenamiento desde cero, esta técnica parte de una base y lo especializa utilizando un conjunto mucho más pequeño y específico de datos. De este modo se reduce significativamente el coste computacional a la vez que se aprovecha la capacidad predictiva del modelo del que se parte.

Para llevar a cabo el ajuste fino, se conservan la estructura y los parámetros previamente entrenados del modelo base, evitando así la necesidad de un entrenamiento completo desde cero. En la práctica, el proceso de *fine-tuning* suele centrarse en modificar únicamente una parte reducida del modelo, como las capas superiores, que son las más cercanas a la salida y, por tanto, más fácilmente adaptables a tareas específicas. Alternativamente, pueden integrarse nuevas capas que se entrenan sin alterar el resto del modelo. Esta aproximación permite preservar los conocimientos generales adquiridos durante el preentrenamiento mientras se optimiza la capacidad del modelo para tareas concretas.

Además, en escenarios con recursos computacionales limitados se utilizan técnicas de ajuste fino eficiente (parameter-efficient fine-tuning, PEFT), como LoRA (*Low-Rank Adaptation*) [29], *prefix tuning* [30] o *prompt tuning*. Estas estrategias reducen la cantidad de parámetros que deben entrenarse, lo que no solo disminuye el tiempo de ajuste y el consumo de memoria, sino que también facilita la reutilización del modelo base en múltiples contextos sin interferencia entre tareas.

Al adaptar un modelo generalista a contextos específicos, se logra una mejora sustancial en la precisión, relevancia y sensibilidad del modelo frente a matices del lenguaje propios del área de aplicación. No obstante, es fundamental contar con datos de alta calidad y bien etiquetados para evitar la sobreespecialización o la introducción de sesgos.

2.5. LM Studio y otras tecnologías

LM Studio [31] es una plataforma de código abierto diseñada para facilitar la interacción y despliegue de modelos extensos de lenguaje (LLM) de manera local, lo que permite a los usuarios ejecutar y experimentar con modelos sin depender de servicios en la nube. Esta herramienta destaca por su integración sencilla, soporte para múltiples formatos de modelos y una interfaz amigable.

Una de las ventajas clave de LM Studio es su capacidad para manejar modelos grandes y complejos con eficiencia. Ofrece funcionalidades como la gestión de memoria optimizada, soporte para cuantización y carga progresiva de modelos lo que permite que usuarios con recursos limitados puedan aprovechar la potencia de los LLM sin necesidad de infraestructura costosa. Además, LM Studio facilita la personalización de modelos mediante interfaces accesibles, lo que la convierte en una opción popular tanto para investigadores como para desarrolladores que desean incorporar inteligencia artificial avanzada en sus proyectos.

Para la implementación del *backend* de la aplicación que interactúa con LM Studio, se utilizó Python, un lenguaje de programación ampliamente adoptado en el ámbito de la inteligencia artificial por su simplicidad y la gran cantidad de bibliotecas disponibles. Se empleó la biblioteca de FastAPI [32] en al construcción del servidor web gracias a su sintaxis intuitiva, lo que facilitó la creación de *endpoints* eficientes para la comunicación entre el usuario y el modelo.

Por otro lado, PyTorch [33] es la biblioteca de referencia para el desarrollo y entrenamiento de modelos de aprendizaje profundo, incluyendo los LLM. Proporciona una interfaz dinámica y flexible que permite tanto el entrenamiento como la inferencia eficiente en hardware acelerado, y es compatible con múltiples plataformas.

Capítulo 3

Diseño e implementación

En este capítulo se detalla cómo se ha ejecutado la implementación del trabajo, incluyendo la arquitectura del sistema, procesamiento de peticiones, modelos extensos de lenguaje utilizados y las técnicas empleadas sobre la inteligencia artificial. Adicionalmente, se incluyen los hallazgos intermedios significativos que fueron fundamentales para la implementación final.

3.1. Arquitectura del sistema

Los primeros esfuerzos en el desarrollo del trabajo se enfocaron en definir el conjunto de módulos que compondrían la solución. Estos componentes fueron identificados correctamente desde el principio: un servidor *web* y un módulo de modelos extensos de lenguaje. Sin embargo, su relación en el sistema sí cambió durante la fase de implementación.

Inicialmente, durante la fase de diseño, se concibió que el servidor *web* también alojaría el modelo extenso de lenguaje y sería responsable de gestionar su conjunto de datos, entrenamiento y despliegue. Esta decisión se tomó al principio de la implementación, influenciada por la forma en la que se trabaja con redes neuronales sencillas. Hacer que esa arquitectura funcionase en este trabajo era mucho más complejo de lo estimado por la propia naturaleza de los modelos extensos de lenguaje. La incertidumbre generada por la ausencia de un conjunto de datos de calidad y tamaño suficientes, y la dificultad implícita de reentrenar un modelo tan grande, fueron un factor de riesgo permanente durante la fase de diseño.

Fue necesario profundizar en las materias de procesamiento de lenguaje natural y *Large Language Models* [34] para definir una solución efectiva a este problema. Finalmente se decidió por no solo "extraer" el módulo de modelos extensos de lenguaje del servidor *web* sino por simplificar su complejidad de programación haciendo uso de la herramienta LM Studio. Este programa permite administrar los modelos locales de la computadora, añadir nuevos a través de una interfaz de descarga y hacer uso del hardware disponible para arrancarlos de forma transparente para el usuario. Además, cuenta con varias ventanas de configuración de parámetros de lanzamiento del modelo, *prompts* personalizados, salida estructurada a través de un esquema JSON, opciones relacionadas con la aleatoriedad y temperatura de la inferencia y otras características experimentales.

De este modo, el servidor *web* también se simplificó en el desarrollo, ahorró mucho trabajo de programación sin comprometer la calidad del proceso de la inteligencia artificial. Esto también es coherente con la arquitectura de servidor ligero

esperable en un prototipo y tiene una mejor sinergia con bibliotecas orientadas al despliegue de servidores sencillos.

Tal y como se muestra en la figura 3.1, el prototipo es accesible para los clientes a través del protocolo HTTP-REST que expone el servidor. Además, durante el desarrollo se implementó una sencilla página web para realizar peticiones de prueba y mostrar la salida de texto obtenida.

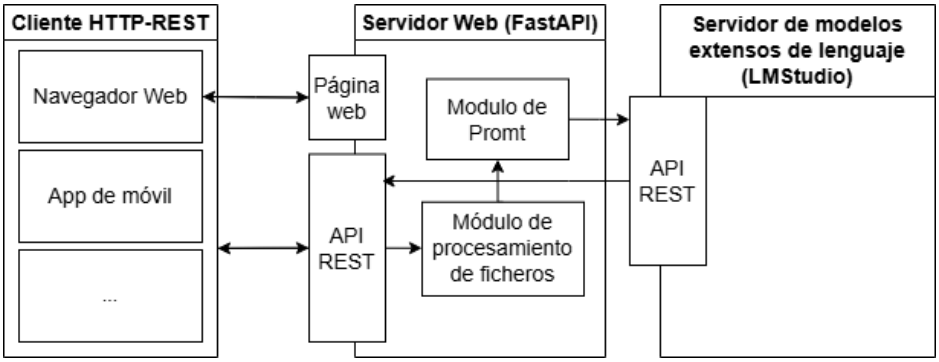

FIGURA 3.1. Arquitectura del sistema.

Otra ventaja de haber diseñado ambos módulos de forma independiente es que facilita al cliente profundizar de forma paralela en los componentes sin comprometer el funcionamiento del resto del sistema. Puede decidir si sustituir la implementación de uno u ambos módulos, escalarlos y distribuirlos en distintos entornos para ajustarlo a su aplicación para smartphones y plan de expansión de servicios.

3.2. Procesamiento de peticiones *web* y ficheros

Para la implementación del servidor web se decidió utilizar la biblioteca de FastAPI [32]. Es una biblioteca moderna y de alto rendimiento para la construcción de APIs web en Python, diseñada sobre estándares como OpenAPI [35] y JSON schema. Ofrece una forma rápida y eficiente de desarrollar interfaces REST con una sintaxis sencilla y basada en anotaciones. Esto permite una validación automática de datos de entrada y generación de documentación. Todo esto hace que FastAPI sea una opción ideal para construir microservicios y prototipos rápidos.

El servidor expone dos servicios principales. El primero es un *endpoint* de tipo GET que proporciona la página *web* que visualiza la interfaz gráfica del prototipo. El otro servicio implementa un método POST para recibir las peticiones de consulta por parte del usuario para su procesamiento y reenvío al módulo de inteligencia artificial.

3.2.1. Procesamiento de peticiones

En la figura 3.2 se representa el diagrama de flujo correspondiente al manejo de las peticiones entrantes.

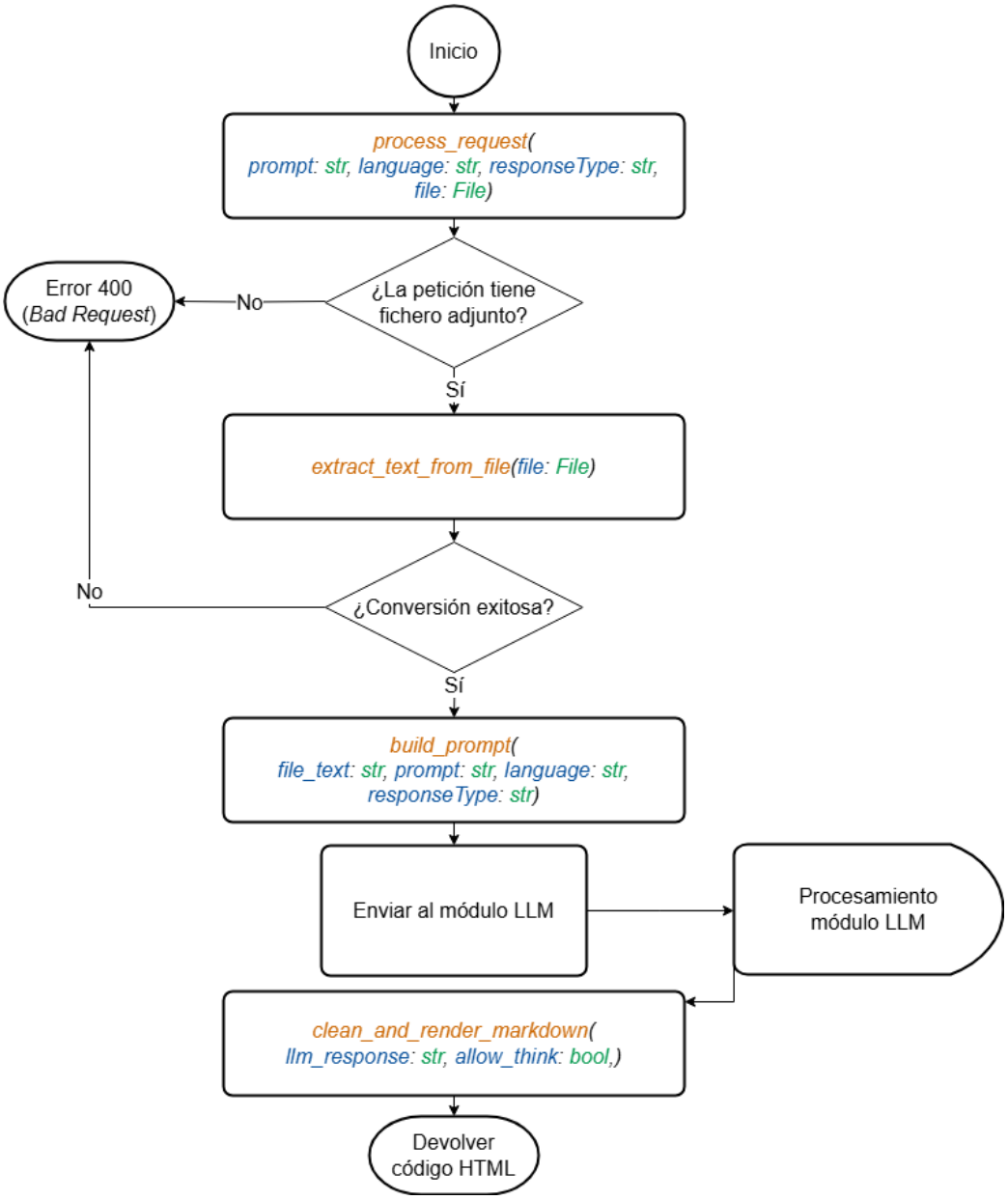

FIGURA 3.2. Diagrama de flujo del procesamiento de peticiones.

El proceso comienza cuando el cliente envía una solicitud con los datos de consulta a través de un cliente REST. Esto puede ser a través del formulario proporcionado por el servidor o a través de cualquier otra herramienta como, por ejemplo, PostMan. Tras realizar las comprobaciones iniciales, el servidor continúa procesando la información mediante una serie de funciones específicas:

- `extract_text_from_file`: es el proceso encargado de extraer la información textual de los ficheros adjunto y formatearlo para que el módulo de modelos extensos de lenguaje sea capaz de interpretarlo. En esta función

se hace uso de las bibliotecas de cgardet, docx y fitz para la detección del formato y su conversión a texto plano.

- `build_prompt`: esta función es la encargada de construir el *prompt* a partir del texto extraído del fichero y las entradas adicionales proporcionadas por el usuario, entre las que se incluyen el idioma de la respuesta, el tipo de listado que se solicita y algunas instrucciones adicionales. En esta función se emplea la ingeniería de *prompting* que se envía posteriormente al módulo LLM. Se detalla más adelante en la sección 3.4.1.
- `clean_and_render_markdown`: esta función se encarga de limpiar la salida y formatearla en código HTML a través de la biblioteca markdown para una mejor legibilidad en navegadores web.

Cabe destacar que para la comunicación con el módulo LLM, aunque LM Studio ofrece un SDK en Python para interactuar mediante programación, se optó por utilizar su interfaz REST. La razón es que así el servidor *web* no depende exclusivamente de la biblioteca de LM Studio. Al emplear un protocolo de comunicación estándar como REST, el servidor web puede interactuar fácilmente con cualquier otro servicio que aloje modelos extensos de lenguaje, lo que le da mayor flexibilidad.

3.2.2. Interfaz gráfica

El servidor también facilita el acceso a una página *web* con la interfaz gráfica del prototipo, tal y como se aprecia en la figura 3.3.

FIGURA 3.3. Interfaz gráfica del prototipo.

El formulario consta de los siguientes elementos:

- Cuadro de texto: este elemento se utiliza para darle las instrucciones específicas a la inteligencia artificial. Está pensado para que sea un mensaje breve

y conciso que pueda ayudar al usuario a guiar un poco mejor la respuesta. Es un parámetro opcional y puede dejarse en blanco.

- Idioma de respuesta: se trata de un *combobox* que contiene dos opciones: inglés y español. El módulo LLM responderá en el idioma seleccionado, independientemente del idioma del fichero de entrada. Este elemento se añadió porque algunos modelos presentan comportamientos anómalos si responden en un idioma distinto al inglés.
- Tipo de respuesta: permite a la inteligencia artificial centrar su respuesta en un aspecto concreto de la narrativa. Al estar instruida en devolver listas de elementos, la idea es que el usuario pueda elegir la opción específica de la creación de mundos que le interese.
- Fichero adjunto: es el elemento más importante del formulario. El usuario puede agregar un fichero con el contexto narrativo existente y enviarlo a la inteligencia artificial para que genere nuevo contenido.

3.3. Gestión de modelos extensos de lenguaje

El siguiente paso en la fase de implementación fue la instalación y configuración del módulo de modelos extensos de lenguaje. Inicialmente, se analizó si la gestión y comunicación con la LLM se haría a través de una API en la nube o alojarlo localmente en el servidor *web*.

La opción de la API implicaba depender de servicios externos en internet, lo que resultaba en peores tiempos de respuesta y un coste adicional significativo debido a los modelos de pago por uso que restringen la cantidad de consultas. A su vez, la interacción con servicios externos exponía la información transmitida en las consultas lo que podría incumplir los requerimientos de protección de la privacidad y propiedad intelectual de los datos del cliente. También se observó un menor número de modelos disponibles y opciones de personalización, lo que dificultaba la experimentación y adaptación de los LLM a las necesidades específicas del sistema.

Por otra parte, la opción de gestionar los modelos extensos de lenguaje de forma nativa presentaba desafíos considerables. Desarrollar la infraestructura *software* necesaria para la gestión de modelos era una tarea con alta complejidad técnica que iba más allá de la mera integración de *frameworks*. Esto podía suponer limitaciones significativas en el proceso de configuración de los modelos y, en consecuencia, provocar un coste elevado en términos de tiempo y recursos de desarrollo.

Por estos motivos se decidió buscar alternativas para este módulo y se encontraron varias herramientas que cumplían con las características deseadas, y las más llamativas fueron Ollama [36], GPT4All [37] y LM Studio [31]. Esta última fue la opción elegida eventualmente por su interfaz intuitiva y buscador de modelos, además de simplificar la gestión y configuración de los LLM. Alojar localmente los modelos permite el envío ilimitado de consultas y cumple con los requisitos de privacidad de los datos del cliente.

En los siguientes apartados se detallan los distintos procesos de gestión que se llevaron a cabo con LM Studio.

3.3.1. Descarga de modelos

El menú *discover* de LM Studio da acceso al explorador de modelos extensos de lenguaje, permitiendo a los usuarios navegar por una vasta biblioteca alojada principalmente en la plataforma Hugging Face [38]. Como se ilustra en la figura 3.4, esta herramienta de búsqueda facilita la localización de modelos específicos mediante filtros por nombre o palabras clave, presentando los resultados en un listado que se ajusta a los criterios definidos. Al seleccionar un modelo, se muestra una sección de detalles con sus características y las distintas opciones de descarga disponibles.

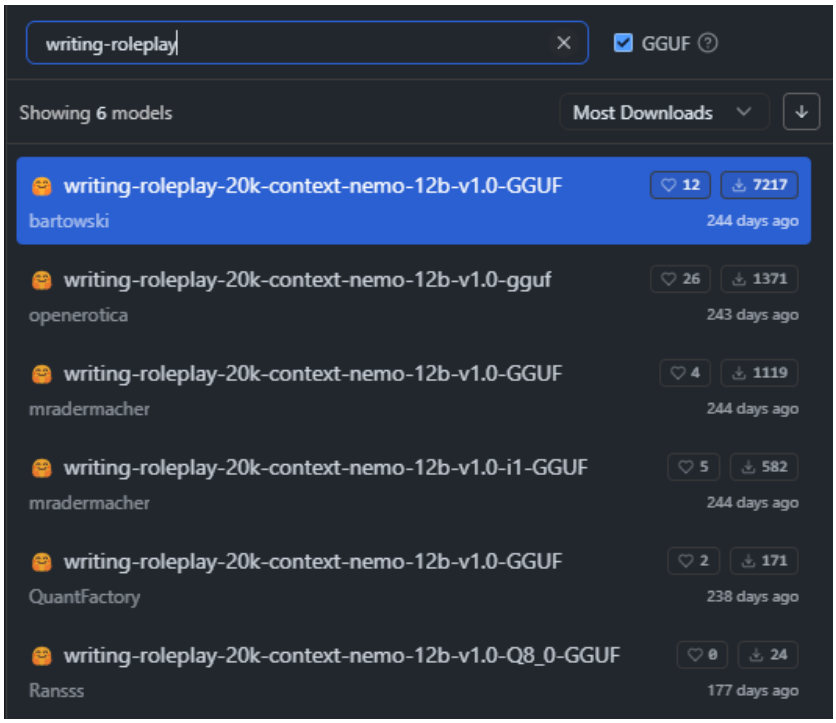

FIGURA 3.4. Buscador de modelos de LM Studio.

Esta herramienta se empleó para la búsqueda y descarga de múltiples modelos, que se enumeran y detallan en la subsección 3.3.4. Se eligieron tanto modelos generalistas como aquellos especializados en *roleplay*, con el fin de analizar y comparar su rendimiento en tareas de construcción de mundos.

3.3.2. Administración y configuración de modelos

LM Studio permite la gestión y configuración de los modelos locales en el menú *My Models*. Además de listar los LLM descargados en el equipo, esta ventana permite personalizar sus parámetros de ejecución para optimizar el rendimiento y la precisión de las respuestas.

Como se muestra en la figura 3.5, al seleccionar un modelo se tiene acceso a un conjunto de ajustes agrupados en varios conjuntos. Como se trabajó en el *prompting* en el servidor *web*, los esfuerzos de configuración se centraron en los parámetros de carga e inferencia.

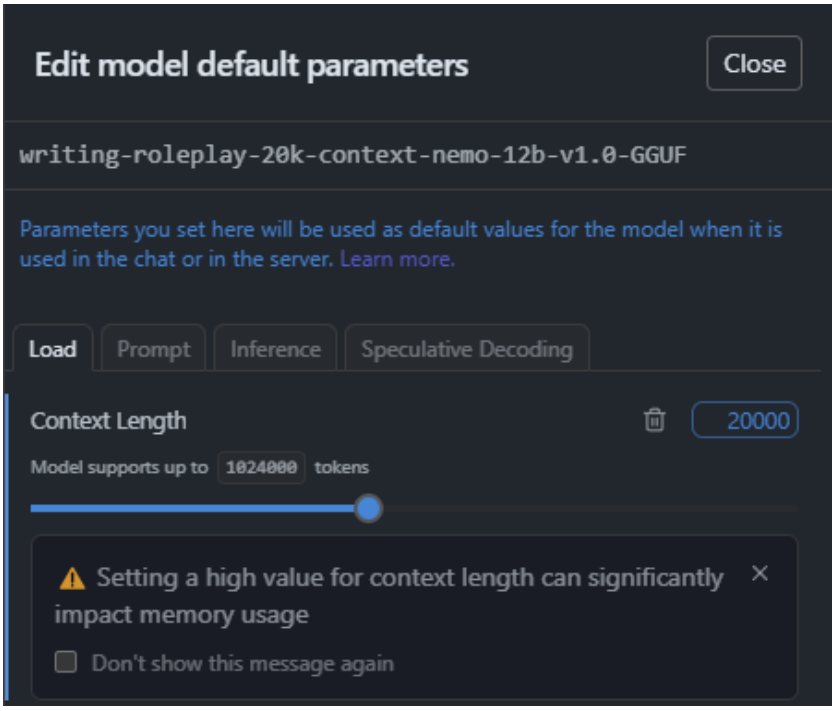

FIGURA 3.5. Configurador de modelos de LM Studio.

La longitud del contexto resultó ser un parámetro fundamental en el trabajo. Dado que las consultas de los usuarios se espera que vengan acompañadas de relatos extensos de *worldbuilding*, era crucial que el LLM pudiera procesar y comprender grandes volúmenes de texto para generar respuestas coherentes y relevantes. Los modelos tienen configurado por defecto un tamaño de contexto de 2048 o 4096 *tokens*, un valor muy por debajo del tamaño de petición que se estima.

Para superar esta limitación, se seleccionó un tamaño de contexto compatible con las capacidades relativas de cada modelo, a menudo indicado en los detalles. Esta configuración fue esencial para asegurar que los LLM pudieran asimilar la totalidad de los relatos proporcionados. Adicionalmente, se exploraron diversas tácticas para gestionar el exceso de contexto, resultando más eficaces el acortamiento desde el principio de la secuencia o desde su punto medio.

Los parámetros de configuración de inferencia más importantes fueron la temperatura y las técnicas de muestreo de probabilidad, como Top P o Top K. Durante el desarrollo del proyecto, se observó que los valores predeterminados para estas variables ya eran bastante adecuados. Por eso, solo fue necesario hacer ajustes mínimos en algunas de ellas.

Por defecto todos los modelos incluyen una sección de razonamiento. Este parámetro fue deshabilitado en la configuración del modelo y se filtró en el postprocesamiento para que no se mostrara en la respuesta al usuario.

3.3.3. Ejecución de modelos e interfaz REST

La herramienta muestra y administra qué LLMs están activos y cargados en la memoria del sistema. Además, también puede desplegar con un solo *click* un servidor local compatible con la interfaz REST de OpenAI. Cuando las peticiones llegan a este servidor, se registran en un log detallado que no solo proporciona información sobre su estado sino que también muestra datos cruciales para la depuración de salidas incorrectas o errores.

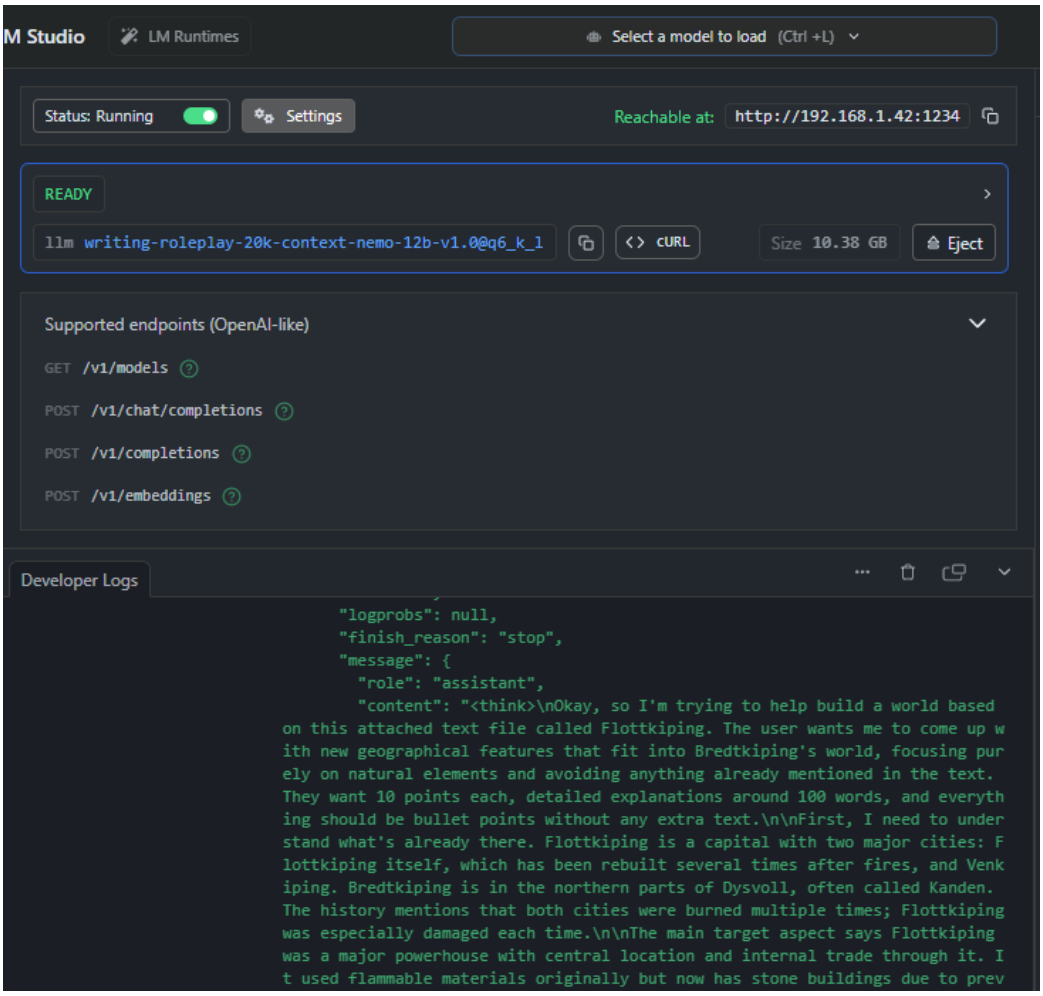

FIGURA 3.6. Menú de desarrollador de LM Studio.

Una vez el modelo y el servidor están activos, el flujo del procesamiento de peticiones de la solución está completo. El LLM, ya configurado con el tamaño de contexto ampliado y los parámetros de aleatoriedad ajustados, gestiona la entrada, generando las respuestas esperadas. Este diseño permite la misma forma de gestionar las peticiones, sin importar el modelo que esté activo.

3.3.4. Modelos extensos de lenguaje utilizados

El conjunto de modelos empleados en este trabajo se obtuvo a través de la herramienta de descarga de LM Studio. Fueron seleccionados con un número de parámetros y un nivel de cuantización adecuado, sin superar el límite de 12 GB de tamaño en memoria RAM dedicada en la tarjeta gráfica. Esta parametrización permite aprovechar al máximo las capacidades del *hardware* disponible, con el objetivo de evaluar el rendimiento de distintos LLMs en tareas de generación narrativa. En la tabla 3.1 se detallan las características de cada modelo:

TABLA 3.1. Modelos extensos de lenguaje utilizados en el trabajo.

Modelo	Editor	Arquitectura	Parámetros	Cuantización	Tamaño (Contexto / VRAM)
`arliai-rpmax-v1.4`	bartowski	Mistral	24 B	Q3_K_S	20 k tokens / 10,40 GB
`writing-roleplay-v1.0`	bartowski	LLaMA	12 B	Q6_K_L	20 k tokens / 10,38 GB
`worldbuilder`	mrademacher	LLaMA	12 B	Q6_K	4k tokens / 10,06 GB
`deepseek-r1-distill`	lmstudio	Qwen2	7 B	Q4_K_M	32 k tokens / 4,68 GB
`mistral-instruct-v0.1`	TheBloke	Mistral	8 B	Q8_0	20 k tokens / 7,70 GB

Los tres primeros modelos referenciados fueron *fine-tuneados* específicamente para tareas de *roleplay* y *worldbuilding*. Esto permite una mejor comprensión del fichero proporcionado por el usuario y una generación de contenido narrativo más coherente y contextualizado.

Con el fin de evaluar el rendimiento del sistema con las alternativas ampliamente accesibles en línea, se incorporaron dos modelos generalistas que actúan como referencia. Dado que estos modelos no han sido entrenados explícitamente para la creación narrativa, permite evaluar tanto la eficacia de la ingeniería de *prompting* implementada, como el impacto del entrenamiento específico en la capacidad de análisis del contexto y la calidad de las respuestas generadas.

3.4. Personalización del prompt en función del servicio

Una de las funcionalidades clave del sistema implementado consiste en la generación de la instrucción a partir de la información proporcionada por el usuario en la petición. Para guiar de forma efectiva la salida del modelo y obtener respuestas que cumplieran con los requerimientos del cliente, se recurrió a diversas técnicas de ingeniería de *prompting*.

3.4.1. Función generadora de la instrucción

Para aterrizar la lógica requerida se desarrolló en el módulo de *prompt* la función *build_prompt* y cuyo código se muestra debajo de este párrafo. Este método se encarga de construir la instrucción alrededor del contexto narrativo, extraído previamente del texto del fichero adjunto, y combinarlo de forma estructurada con el resto de elementos de entrada.

```
def build_prompt(file_text, additional_instructions, language,
    response_type):
    if response_type not in prompts:
        raise ValueError(
            f"Invalid output_type: '{response_type}'. Must be one of {
    list(prompts.keys())}"
        )

    general_prompt = prompts["general"]
    specific_prompt = prompts[response_type]

    content = (
        f"{general_prompt}\n"
        f"=== ATTACHED TEXT FILE ===\n"
        f"{file_text}\n\n"
        f"=== ADDITIONAL INSTRUCTIONS ===\n"
        f"{additional_instructions}"
        f"======\n"
        f"{specific_prompt}\n\n"
        f"Answer only in {language}.\n\n"
    )

    return {"role": "user", "content": content}
```

CÓDIGO 3.1. Estructura de la función que construye la instrucción para el modelo.

A continuación, se definen los elementos principales en orden secuencial:

- *response_type*: tipo de salida esperada por parte del modelo. Este parámetro selecciona el *prompt* específico dentro del conjunto de *prompts* y si el valor proporcionado no se encuentra definido en dicho conjunto, se lanza una excepción.
- *general_prompt*: fragmento introductorio de la instrucción, común para todos los tipos de respuesta, que establece el rol y las directrices principales que tiene que seguir el modelo.
- *specific_prompt*: instrucción particular asociada al tipo de respuesta especificado en `response_type`. Complementa a la instrucción principal añadiendo detalles o restricciones más concretas, adaptadas a la tarea específica que debe realizar el modelo.
- *file_text*: contenido del archivo de entrada con el contexto narrativo sobre el cual se construye la instrucción que es enviada al modelo.
- *additional_instructions*: conjunto de instrucciones adicionales proporcionadas por el usuario. Estas indicaciones permiten personalizar o enriquecer el comportamiento del modelo y ajustar la respuesta a necesidades específicas.
- *language*: idioma en el que debe responder el modelo. Esta indicación se incluye al final del mensaje para garantizar que la salida esté redactada exclusivamente en el idioma solicitado.

Esta implementación permite personalizar la instrucción de manera precisa según la categoría de respuesta y otras variables contextuales. Gracias a este enfoque estructurado, basado en la técnica de *prompt chaining*, se reduce significativamente la complejidad de la interacción entre el usuario y el modelo de lenguaje.

Además garantiza una respuesta precisa sin necesidad de que el usuario, que previsiblemente utilizará el servicio desde un dispositivo con pantalla reducida, posea conocimientos en ingeniería de *prompting*.

3.4.2. Refinado iterativo de las instrucciones

Una vez se definió la estructura general del mensaje, el siguiente paso consistió en optimizar tanto la instrucción general como las indicaciones específicas asociadas a cada elemento de la ambientación narrativa. Este proceso fue esencial para garantizar que las respuestas generadas por los modelos se alinearan con las expectativas propias de un asistente creativo que proporcionara listas de nuevas ideas.

El refinamiento se llevó a cabo de forma iterativa, a través de prueba y error con múltiples variantes para las instrucciones. En cada ciclo se evaluaron las respuestas del modelo en función tanto de la estructura del contenido como de su coherencia con el contexto narrativo aportado. Esto permitió identificar errores en la formulación del *prompt* responsables de alucinaciones, repeticiones de información y otras inconsistencias en la generación de texto de los modelos. En función del reto encontrado, se analizó la forma de solucionarlo mediante el empleo de técnicas entre las que se encuentran *chain of thought*, *instruction-based prompting*, *constrained-output prompting* y *negative prompting*.

El refinado de la instrucción general se centró en resolver los problemas comunes detectados en todas las categorías narrativas. En versiones iniciales, el modelo no generaba listas de forma adecuada ni mantenía un formato consistente en la respuesta. También mostró dificultades para distinguir entre el contexto narrativo y las instrucciones que debía seguir. Esto derivaba en repeticiones de información ya presentes en el texto de referencia, lo que limitaba considerablemente la originalidad y riqueza de sus propuestas. Por estas razones, fue necesario reformular la instrucción general con mayor precisión, incorporando directrices explícitas sobre el objetivo del modelo, el formato de salida y la distinción entre contexto e instrucciones. Se establecieron requisitos claros sobre la longitud mínima de las listas, el nivel de detalle esperado en cada punto y la obligatoriedad de que todos los elementos fueran completamente nuevos.

El refinamiento de la instrucción específica para eventos se centró en guiar al modelo hacia la generación de sucesos nuevos que enriquecieran el desarrollo del mundo narrativo sin duplicar hechos ya mencionados en el texto base. En versiones preliminares, el modelo tendía a reformular eventos existentes en lugar de inventar situaciones originales. Esto es debido al fuerte entrenamiento de los modelos a no modificar hechos que se consideran históricos. Para resolver esto, se eliminó la palabra "histórico" de la instrucción y se reforzó la indicación de que los eventos debían ser completamente inéditos y conectados de forma coherente con el contexto.

En cuanto a los personajes, las versiones iniciales del *prompt* no ofrecían suficientes indicaciones estructurales, lo que resultaba en respuestas incompletas o poco claras. Algunos personajes carecían de nombre, contexto o motivación dentro del mundo ficticio. Para solventar esto, la instrucción fue reescrita para exigir explícitamente un nombre, apellido, alias (si procede) y una explicación clara de la relación del personaje con la historia o el entorno.

La instrucción específica para geografía fue afinada para evitar confusiones entre elementos naturales y construcciones humanas. En iteraciones anteriores, el modelo a menudo mezclaba ríos o montañas con ciudades o estructuras artificiales, y no siempre justificaba la relevancia geográfica de los elementos descritos. Para corregir esto, se incorporó una aclaración sobre el tipo de elementos geográficos aceptados y se exigió una descripción detallada de cada punto, así como su importancia ecológica, simbólica o estratégica dentro del mundo narrativo.

Finalmente, el ajuste de la instrucción para localizaciones buscó mejorar la claridad y especificidad de los núcleos poblacionales propuestos. En las versiones previas, el modelo a menudo confundía localizaciones con elementos geográficos o entregaba descripciones genéricas sin conexión con el resto del mundo. La nueva versión de la instrucción estableció que debían generarse únicamente poblaciones (ciudades, pueblos, países) y que cada una debía incluir una descripción detallada de sus características culturales, políticas o históricas, así como su función o importancia dentro del contexto narrativo.

Se puede consultar la versión final de las instrucciones desarrolladas en este trabajo en el Anexo (FALTA REFERENCIA). Dado que el inglés es el idioma preferente de los modelos, las instrucciones se redactaron en ese idioma. Por esto mismo, se incluye una instrucción final para que la respuesta se genere en el idioma seleccionado por el usuario. Sin embargo, es importante señalar que no todos los modelos soportan el español, lo que ocasionalmente lleva a que esta última instrucción sea ignorada. Este fenómeno se detallará en la siguiente sección.

Capítulo 4

Ensayos y resultados

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

4.1. Pruebas funcionales del hardware

La idea de esta sección es explicar cómo se hicieron los ensayos, qué resultados se obtuvieron y analizarlos.

Capítulo 5

Conclusiones

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

En esta sección no se deben incluir ni tablas ni gráficos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se puedo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.

Bibliografía

[1] Tomás Muriel. *Narrativa*. 2 de oct. de 2023. URL: https://www.significados.com/narrativa/ (visitado 24-05-2025).

[2] Jeff VanderMeer. *Wonderbook: The Illustrated Guide to Creating Imaginative Fiction*. Abrams Image, 2013. ISBN: 978-1-4197-0681-7.

[3] J.R.R. Tolkien. *The Letters of J.R.R. Tolkien*. Houghton Mifflin, 1981.

[4] *Critical Match: Plataforma para juegos de mesa*. https://www.criticalmatch.app/. Plataforma enfocada en juegos de mesa como rol, cartas coleccionables y wargames. (Visitado 24-05-2025).

[5] Critical Match UG. *Critical Match: Encuentra mesas de rol*. Google Play Store. Versión de la aplicación móvil. 2025. URL: https://play.google.com/store/apps/details?id=com.criticalmatch (visitado 24-05-2025).

[6] OpenAI. *ChatGPT: Optimizing Language Models for Dialogue*. Blog post. Nov. de 2022. URL: https://openai.com/blog/chatgpt/ (visitado 24-05-2025).

[7] Gemini Team. «Gemini: A Family of Highly Capable Multimodal Models». En: *arXiv preprint arXiv:2312.11805* (2023). URL: https://arxiv.org/abs/2312.11805.

[8] Meta AI. *About Meta AI*. Website. Página oficial de Meta AI. URL: https://ai.meta.com/ (visitado 24-05-2025).

[9] World Anvil. *World Anvil – The Ultimate Worldbuilding Toolset & RPG Campaign Manager*. https://www.worldanvil.com. Último acceso: junio de 2025. 2025.

[10] LegendKeeper. *LegendKeeper – Worldbuilding Software for Worldbuilders and GMs*. https://www.legendkeeper.com. Último acceso: junio de 2025. 2025.

[11] Obsidian Portal. *Obsidian Portal – Campaign Websites for Dungeons and Dragons and Other Tabletop RPGs*. https://www.obsidianportal.com. Último acceso: junio de 2025. 2025.

[12] Ashish Vaswani et al. «Attention is all you need». En: (). arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[13] Alec Radford et al. «Improving Language Understanding by Generative Pre-Training». En: *OpenAI* (jun. de 2018). Technical Report. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (visitado 24-05-2025).

[14] Jacob Devlin et al. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». En: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (2019), págs. 4171-4186. DOI: 10.18653/v1/N19-1152. URL: https://arxiv.org/abs/1810.04805.

[15] Tom B. Brown et al. «Language Models are Few-Shot Learners». En: *Advances in Neural Information Processing Systems* 33 (2020), págs. 1877-1901. URL: https://proceedings.neurips.cc/paper/2020/file/145749ea3f59e0b57574be084196c703-Paper.pdf.

[16] Tomas Mikolov et al. «Efficient Estimation of Word Representations in Vector Space». En: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2013.
[17] Sepp Hochreiter y Jürgen Schmidhuber. «Long short-term memory». En: *Neural computation* 9.8 (1997), págs. 1735-1780.
[18] Kyunghyun Cho et al. «Learning phrase representations using RNN encoder-decoder for statistical machine translation». En: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, págs. 1724-1734.
[19] Alec Radford et al. «Language models are unsupervised multitask learners». En: *OpenAI blog* 1.8 (2019), pág. 9.
[20] Zhengbao Ji, Noah A. Smith y Luke Zettlemoyer. «Survey of Hallucination in Natural Language Generation». En: *arXiv preprint arXiv:2302.03626* (2023).
[21] Ari Holtzman et al. «The curious case of neural text degeneration». En: *arXiv preprint arXiv:1904.09751* (2019).
[22] Aakanksha Chowdhery et al. «PaLM: Scaling Language Modeling with Pathways». En: *arXiv preprint arXiv:2204.02311* (2022).
[23] Hugo Touvron et al. «LLaMA: Open and Efficient Foundation Language Models». En: *arXiv preprint arXiv:2302.13971* (2023).
[24] Tom B Brown et al. «Language models are few-shot learners». En: *Advances in neural information processing systems* 33 (2020), págs. 1877-1901.
[25] Takeshi Kojima et al. «Large language models are zero-shot reasoners». En: *arXiv preprint arXiv:2205.11916* (2022).
[26] Jason Wei et al. «Chain of thought prompting elicits reasoning in large language models». En: *arXiv preprint arXiv:2201.11903* (2022).
[27] Prompting Guide. *Prompt Chaining*. https://www.promptingguide.ai/techniques/prompt_chaining. 2023.
[28] Jeremy Howard y Sebastian Ruder. «Universal language model fine-tuning for text classification». En: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2018), págs. 328-339.
[29] Edward J Hu et al. «LoRA: Low-Rank Adaptation of Large Language Models». En: *arXiv preprint arXiv:2106.09685* (2021).
[30] Xiang Lisa Li y Percy Liang. «Prefix-Tuning: Optimizing Continuous Prompts for Generation». En: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*. 2021, págs. 4582-4597.
[31] Nomic AI. *LM Studio*. https://lmstudio.ai/. Accessed: 2025-05-26.
[32] Sebastián Ramírez. *FastAPI*. https://fastapi.tiangolo.com/. Accessed: 2025-05-26.
[33] PyTorch Contributors. *PyTorch*. https://pytorch.org/. Accessed: 2025-05-26.
[34] Esp. Ing Abraham Rodriguez FIUBA y Esp. Ing Ezequiel Guinsburg FIUBA. *Materia de Large Language Models*. https://github.com/FIUBA-Posgrado-Inteligencia-Artificial/CEIA-LLMIAG. Material de clase, disponible en GitHub. 2024.
[35] OpenAPI Initiative. *OpenAPI Specification*. https://spec.openapis.org/oas/latest.html. Último acceso: junio de 2025. 2024.
[36] Ollama. *Ollama*. https://ollama.com. Accessed: 2025-06-15.
[37] Nomic AI. *GPT4All*. https://gpt4all.io. Accessed: 2025-06-15.
[38] Hugging Face. *Hugging Face*. https://huggingface.co. Accessed: 2025-06-15.