Escuela
Politécnica
Superior

LLMSearch: Buscador multimedia basado en lenguaje natural

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Izan Gandía Ruiz
Tutor:
Iván Gadea Saéz

Universitat d'Alacant
Universidad de Alicante

Mayo 2025

LLMSearch: Buscador multimedia basado en lenguaje natural

Encuentra tu contenido multimedia al instante con solo describirlo.

Autor
Izan Gandía Ruiz

Tutor
Iván Gadea Saéz
Departamento de Lenguajes y Sistemas Informáticos

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Mayo 2025

Preámbulo

Este proyecto surge de una doble motivación. Por un lado, el profundo interés en explorar y adquirir un conocimiento avanzado sobre el uso y configuración de las inteligencias artificiales multimodales. Por otro lado, la identificación de una problemática recurrente y extendida en la gestión de la información digital personal como es la dificultad persistente para localizar archivos específicos (como imágenes o documentos) en volúmenes grandes de datos. Este desafío se presenta también en contextos más actuales, como la búsqueda de elementos específicos (ej. "stickers") dentro de aplicaciones de mensajería como WhatsApp o Telegram.

Agradecimientos

Este rinconcito es para vosotros, para toda esa gente increíble que ha hecho posible que hoy esté aquí, escribiendo estas líneas.

Primero, a ese montón de compañeros y amigos que me he cruzado en la carrera. He aprendido un millón de cosas con vosotros, no solo de manera académica, sino lecciones de vida que se quedan para siempre. Sin vuestras risas, vuestros ánimos en los momentos de bajón y esa forma de tirar para adelante juntos, no sé si habría encontrado las fuerzas para seguir tantas veces. Sois, en gran parte, la razón de que esté celebrando este logro.

A mi familia, mi pilar fundamental. Gracias por creer en mí incluso cuando yo dudaba, por ponerme las cosas fáciles y por todo el apoyo para que pudiera dedicarme a esto. Sois increíbles. Y un gracias enorme y especial para mi hermano, Abel Gandía Ruiz. Tú fuiste quien me abrió los ojos a este mundo tan interesante de la programación cuando yo no tenía ni idea, quien me animó y me echó una mano para empezar.

También quiero acordarme de mis profes. Algunos habéis sido una inspiración, de esos que te contagian la ilusión y te hacen descubrir la magia en sitios donde nunca te lo hubieras imaginado. Gracias por inspirarme y ayudarme a ser un mejor ingeniero.

Y, cómo no, a mi tutor, Iván Gadea Sáez. Gracias por guiarme con este proyecto, por tu paciencia infinita y por ayudarme a calmar todos los nervios y dudas que me han ido surgiendo.

De verdad, a todos y cada uno, ¡muchísimas gracias!

A quienes me inspiraron a soñar y a programar, recordándome que, si puedo imaginarlo, puedo crearlo. [1]

[1] Alejandro Taboada, creador del canal "Programación ATS"

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1. Introducción

La gestión y recuperación eficiente de la información digital se ha convertido en un desafío cotidiano en la era de la sobrecarga informativa. Los volúmenes de datos personales y profesionales que almacenamos en nuestros dispositivos crecen exponencialmente, mientras que las herramientas tradicionales de búsqueda a menudo resultan insuficientes para localizar archivos específicos de manera rápida y precisa. Este proyecto se adentra en esta problemática, proponiendo una solución innovadora basada en los avances recientes en Inteligencia Artificial (IA) y sistemas de Generación Aumentada por Recuperación (RAG).

1.1. Panorama Actual: Desafíos en la Recuperación de Información

Los métodos convencionales para la organización y búsqueda de archivos digitales se basan en gran medida en metadatos explícitos, como nombres de archivo, fechas o etiquetas manuales. Sin embargo, estas aproximaciones tienen limitaciones significativas:

- **Insuficiencia de los metadatos tradicionales:** A menudo, los metadatos son inexistentes, incompletos o no capturan la semántica real del contenido del archivo (especialmente en el caso de imágenes, vídeos o audios).
- **Falta de precisión en las búsquedas:** Las búsquedas basadas en palabras clave pueden ser ambiguas y no siempre interpretan correctamente la intención del usuario, llevando a resultados irrelevantes o a la omisión de la información que el usuario desea encontrar.
- **Desafíos técnicos y éticos en IA:** Si bien la IA ofrece nuevas vías, también enfrenta retos. Los modelos pueden carecer de la precisión necesaria para ciertas tareas o, en el caso de modelos generativos, incurrir en "alucinaciones" (generar información incorrecta pero plausible). Además, el propio entrenamiento del modelo puede afectar a la interpretación sobre el archivo que se quiere describir, generando resultados no equitativos o discriminatorios, lo que plantea también importantes consideraciones éticas.
- **Riesgos de seguridad y privacidad de los datos:** La externalización del almacenamiento a la nube y el uso de IA para la catalogación y el análisis de ficheros introducen nuevas formas de ataque y preocupaciones sobre la seguridad y privacidad. Estos sistemas pueden ser vulnerables a accesos no autorizados, fugas de datos o ciberataques, tanto en la infraestructura cloud como en los propios modelos de IA. La información sensible contenida en los archivos, o incluso los metadatos enriquecidos generados por la IA, podrían quedar expuestos, ser alterados o perderse. Esto es especialmente crítico cuando se manejan datos confidenciales o personales, donde una brecha de seguridad no solo implica la pérdida de información valiosa, sino también posibles consecuencias legales, financieras y reputacionales significativas.

Este contexto muestra la necesidad de sistemas más inteligentes y contextuales capaces de comprender el contenido de los archivos de forma más profunda, más allá de sus metadatos superficiales, y que al mismo tiempo garanticen la integridad y confidencialidad de la información.

1.2. Avances Tecnológicos Fundamentales

Para abordar los desafíos mencionados, este proyecto se apoya en los desarrollos más recientes en el campo de la Inteligencia Artificial, particularmente en las siguientes áreas:

1.2.1. Inteligencia Artificial Multimodal: Convergencia de Lenguaje y Visión

La Inteligencia Artificial ha experimentado avances exponenciales, especialmente con el auge del Procesamiento del Lenguaje Natural (PLN) y la Visión Artificial. La multimodalidad representa la capacidad de los sistemas de IA para procesar, comprender y generar información a partir de múltiples tipos de datos (o "modalidades") simultáneamente, como texto, imágenes, audio y vídeo.

- Procesamiento del Lenguaje Natural (PLN): Permite a las máquinas comprender, interpretar y generar lenguaje humano. Los Grandes Modelos de Lenguaje (Large Language Model (LLM)), como Generative Pre-trained Transformer (GPT) (Generative Pre-trained Transformer) y sus variantes, han revolucionado este campo, demostrando una capacidad asombrosa para entender el contexto, generar texto coherente e incluso razonar sobre la información proporcionada.
- Visión Artificial: Es la disciplina que permite a las máquinas "ver" e interpretar el contenido de imágenes y vídeos. Implica tareas como la detección de objetos, el reconocimiento facial, la segmentación de imágenes y la generación de descripciones visuales.
- Modelos Multimodales (ej. Contrastive Language-Image Pre-training (CLIP))[1]: Modelos como CLIP (Contrastive Language-Image Pre-training) de OpenAI son un ejemplo paradigmático de esta convergencia. CLIP aprende representaciones visuales a partir de descripciones en lenguaje natural, permitiendo realizar búsquedas de imágenes mediante consultas textuales con una alta comprensión semántica, o viceversa. Funciona entrenando un codificador de imágenes y un codificador de texto para predecir qué imágenes se emparejan con qué textos en un gran conjunto de datos.

1.2.2. Optimización de Modelos: Cuantización y Modelos Ligeros

Para la aplicación práctica de estos modelos, especialmente en entornos con recursos limitados (como dispositivos personales), es crucial considerar su eficiencia.

- Modelos Cuantizados: La cuantización es un proceso que reduce la precisión numérica de los pesos y activaciones de un modelo de red neuronal (por ejemplo, de punto flotante de 32 bits a enteros de 8 bits). Esto disminuye significativamente el tamaño del modelo y acelera la inferencia, con una pérdida de precisión a menudo mínima.

[1] Más información sobre CLIP de OpenAI disponible en: `https://openai.com/es-ES/index/clip/`

- Modelos Ligeros (Lightweight Models): Son arquitecturas de redes neuronales diseñadas específicamente para ser computacionalmente eficientes y tener un tamaño reducido, facilitando su despliegue en dispositivos móviles o embebidos sin sacrificar excesivamente el rendimiento.

1.2.3. Sistemas de Generación Aumentada por Recuperación (RAG)

La Generación Aumentada por Recuperación (RAG) es una técnica que mejora el rendimiento de los LLM al conectarlos con fuentes de conocimiento externas. En lugar de depender únicamente de la información (potencialmente desactualizada o incompleta) aprendida durante su entrenamiento, un sistema RAG funciona en dos fases:

1. Recuperación (Retrieval): Dada una consulta del usuario, el sistema primero busca y recupera fragmentos de información relevante de una base de datos, un conjunto de documentos o un corpus de conocimiento. Esta base de datos puede estar compuesta por embeddings (representaciones vectoriales densas) del contenido de los archivos.

2. Generación (Generation): La información recuperada se proporciona como contexto adicional al LLM junto con la consulta original. El LLM utiliza este contexto enriquecido para generar una respuesta más precisa, relevante y fundamentada.

Los sistemas RAG ofrecen ventajas significativas, como la reducción de alucinaciones, la capacidad de citar fuentes del contexto dado y la facilidad para actualizar la base de conocimiento sin necesidad de reentrenar el LLM completo. Si bien existen diversas arquitecturas RAG (ej. generando consultas SQL, incorporando texto directamente al prompt, o utilizando embeddings), el enfoque basado en embeddings suele ofrecer un buen equilibrio entre eficiencia y calidad de los resultados, aunque presenta desafíos como la gestión de la ventana de contexto del LLM, punto crucial a tener en cuenta si se utilizan sobre dispositivos personales.

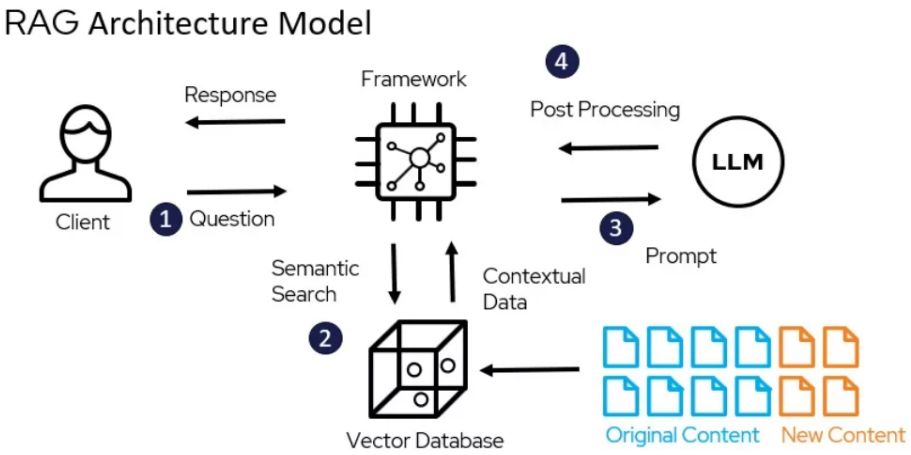

Figura 1.1: Esquema visual del funcionamiento de un sistema RAG, mostrando el flujo desde la consulta del usuario, pasando por la recuperación de información relevante, hasta la generación de la respuesta final por el LLM.

1.2.4. Justificación del Proyecto

Este Trabajo Final de Grado (TFG) busca abordar la necesidad de obtener información de manera rápida, sencilla y muy precisa desarrollando un sistema inteligente de búsqueda de archivos que permita a los usuarios encontrar información utilizando consultas en lenguaje natural, trascendiendo las limitaciones de las búsquedas basadas en metadatos tradicionales. La aplicación de modelos multimodales permitirá indexar el contenido semántico de diversos tipos de ficheros, y la arquitectura RAG proporcionará un marco robusto para recuperar la información más relevante y presentarla de forma útil al usuario.

2. Estado del Arte

Antes de adentrarse en los detalles técnicos, es esencial establecer el contexto actual en el dominio de los buscadores multimedia basados en lenguaje natural. Esta revisión permitirá asentar una fundamentación teórica y metodológica sólida, comprender los desafíos y las limitaciones identificadas en investigaciones previas e identificar las brechas en el conocimiento existente, así como las oportunidades para realizar contribuciones significativas en LLMSearch.

2.1. Modelos de Lenguaje Natural (LLMs) para Búsqueda

Los **LLMs** han revolucionado el procesamiento del lenguaje natural en años recientes. Modelos como *GPT-3* y *GPT-4* (base de **ChatGPT**) demuestran que, con miles de millones de parámetros entrenados en enormes corpus de texto, es posible comprender y generar lenguaje con notable fluidez y contexto. Estos modelos capturan representaciones semánticas ricas, lo que habilita nuevas formas de **búsqueda semántica** y recuperación de información.

Características clave:

- **Búsqueda por significado**: En lugar de limitarse a coincidencias de palabras clave, un LLM puede interpretar la intención de una consulta en lenguaje natural y relacionarla con documentos relevantes aunque no compartan palabras literalmente.

- **Embeddings semánticos**: Técnicas como *embeddings* de oraciones (usando modelos tipo Bidirectional Encoder Representations from Transformers (BERT) o Sentence Transformers) convierten documentos y consultas a vectores en un espacio vectorial común, donde la similitud de coseno permite recuperar los contenidos más cercanos en significado.

- **RAG**: Los LLMs pueden integrarse en pipelines donde primero se recuperan documentos candidatos y luego el modelo genera una respuesta o resumen usando esos textos.

- **Interfaz conversacional**: Modelos tipo ChatGPT permiten refinar iterativamente las consultas de búsqueda mediante diálogo, mejorando la precisión de resultados en consultas ambiguas.

Los avances más recientes se centran en mejorar la **eficiencia y apertura** de estos modelos. Mientras GPT-4 (de OpenAI) es de uso cerrado y con un tamaño muy grande no divulgado (>100B parámetros), han emergido modelos de código abierto como *LLaMA* (Meta) y sus variantes, que con 7–70B parámetros logran desempeños competitivos.

2.2. Modelos Visión-Lenguaje para Imágenes

En un buscador multimedia, es esencial manejar consultas sobre contenido visual (imágenes) usando lenguaje natural. Aquí destacan los **modelos visiolingüísticos** o **Modelo de Visión-Lenguajes (VLMs)**, que conectan representaciones de imágenes con representaciones textuales en un espacio común.

2.2.1. CLIP y Embeddings Multimodales

Un hito fue el modelo **CLIP** de OpenAI, que entrena conjuntamente un codificador de texto (transformer) y un codificador visual (Red Neuronal Convolucional o Vision Transformer (ViT)) para proyectar ambos tipos de entrada en **vectores de embedding** de la misma dimensión. Mediante aprendizaje contrastivo en 400 millones de pares imagen–texto, CLIP logró que textos e imágenes con contenido semántico equivalente quedaran cercanos en el espacio vectorial.

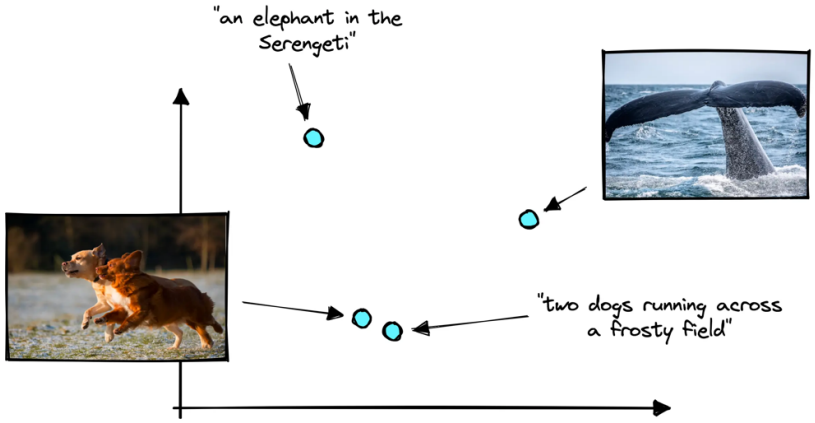

Figura 2.1: Ejemplo conceptual de un espacio vectorial multimodal entrenado por CLIP, donde imágenes y descripciones semánticas correspondientes se representan mediante vectores cercanos.

2.2.2. Modelos Generativos de Descripción de Imágenes

Otra línea de desarrollo significativa se centra en los **modelos generativos de descripción de imágenes**. Estos sistemas realizan la tarea conocida como *image captioning*, que consiste en generar una descripción en lenguaje natural para una imagen dada. Modelos recientes como **Bootstrapping Language-Image Pre-training (BLIP)-2** ejemplifican esta aproximación, combinando un encoder visual pre-entrenado (por ejemplo, CLIP ViT), un modelo de lenguaje grande congelado y un transformador ligero intermedio denominado Q-Former. Esta arquitectura logra puentear eficientemente la brecha entre visión y lenguaje: el encoder de imagen extrae las características visuales relevantes, mientras que el LLM se encarga de generar la descripción textual coherente.

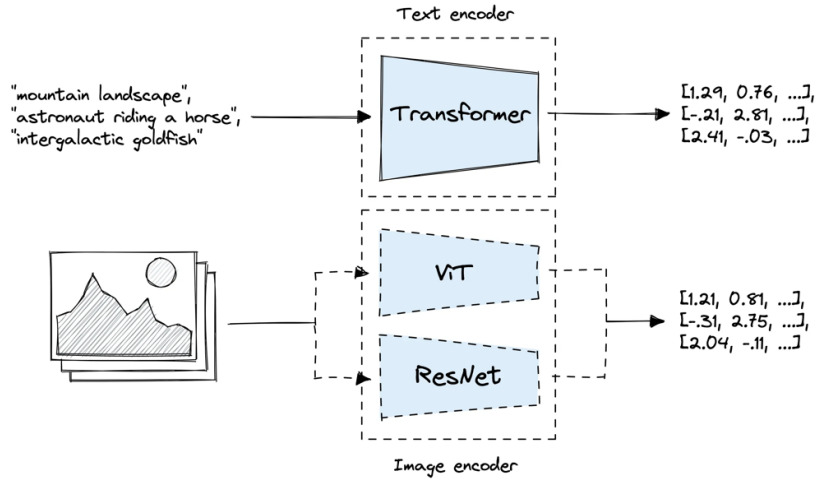

Figura 2.2: Arquitectura del modelo CLIP: encoder de texto y encoder de imagen que proyectan al mismo espacio de embedding.

2.2.3. VQA y Diálogo Multimodal

Junto al desarrollo de modelos como BLIP-2, han aparecido numerosos modelos abiertos que permiten la **Pregunta-Respuesta Visual (VQA)** y el diálogo multimodal. Entre ellos destaca **Large Language and Vision Assistant (LLaVA)**, que utiliza GPT-4 para generar datos sintéticos de entrenamiento y posteriormente afina un modelo basado en *Vicuna* (un derivado de LLaMA) acoplado a un encoder visual. Otro modelo relevante es **Moondream**, un VLM open-source de tan solo 2 mil millones de parámetros (2B), capaz de operar en tiempo real incluso en CPUs o dispositivos móviles. Moondream ha demostrado capacidades notables en la generación de descripciones detalladas, la respuesta a preguntas visuales, la detección de objetos en modalidad cero-shot y el Optical Character Recognition (OCR) básico para leer texto en imágenes. En esta misma línea, **JoyCaption** se presenta como un modelo de captioning de imágenes libre y sin censura, concebido originalmente para generar descripciones ricas que ayuden a entrenar modelos de difusión. Finalmente, aunque de naturaleza propietaria, **GPT-4 con visión** (GPT-4V) ha demostrado capacidades impresionantes al responder con acierto a entradas que combinan imagen y texto, si bien su acceso limitado restringe su uso en entornos académicos.

En síntesis, el estado del arte en la convergencia de imagen y lenguaje ofrece dos enfoques complementarios para la búsqueda multimedia. Por un lado, los *embeddings* multimodales tipo CLIP permiten realizar una **búsqueda directa por similitud** entre consultas textuales y contenido visual. Por otro lado, los *modelos generativos visiolingüísticos* facilitan la **describir o entender imágenes en texto**, permitiendo indexar y razonar sobre imágenes mediante lenguaje natural.

2.3. Modelos Multimodales para Vídeo

Extender la búsqueda basada en lenguaje natural al dominio del **vídeo** conlleva retos adicionales, pues los vídeos combinan secuencias de imágenes con audio y, en ocasiones, texto

incrustado.

2.3.1. Técnicas de Procesamiento de Video

Para abordar la complejidad del vídeo, se emplean diversas técnicas. Una fundamental es el **análisis por frames**, que implica extraer fotogramas representativos del vídeo. A estos se les aplican VLMs, convirtiendo el problema de vídeo en el manejo de un conjunto de imágenes con marcas de tiempo. Paralelamente, el **procesamiento de audio** es crucial. Mediante modelos de **Automatic Speech Recognition (ASR)** como *Whisper*, es posible transcribir con alta calidad el diálogo o narración presente en los vídeos, permitiendo indexar cada vídeo por su transcripción textual completa. Además, se están desarrollando **modelos vídeo-texto end-to-end**, como *VideoCLIP*, que extienden la idea de CLIP al dominio temporal, o transformadores específicos para vídeo que realizan *video captioning*.

2.3.2. Arquitecturas para Búsqueda en Video

Una arquitectura emergente para la búsqueda en vídeo combina los enfoques anteriores en un pipeline RAG multimodal. Este sistema indexa, por un lado, los *frames* visuales mediante embeddings y, por otro, las transcripciones de voz como texto. Ante una consulta, recupera fragmentos candidatos por similitud visual o textual, y posteriormente utiliza un modelo de lenguaje para sintetizar ambas fuentes de información y determinar la respuesta más adecuada.

2.3.3. Modelos Unificados Multimodales

Recientemente, han surgido modelos unificados que procesan múltiples modalidades de forma integrada. **MiniGPT-4**, por ejemplo, puede aceptar secuencias de imágenes como entrada, simulando un vídeo corto. **MiniCPM-V** soporta entradas de vídeo directamente, generando una descripción general del contenido. Google con **Gemini** ha avanzado en la integración de visión, vídeo y sonido en un mismo LLM, y Meta con **ImageBind** ha propuesto aprender una representación común para imágenes, texto, audio y otros sensores, abriendo nuevas vías para la comprensión multimodal holística.

2.4. Análisis de Audio y Búsqueda mediante Sonido

Para completar un buscador verdaderamente multimedia, es imprescindible considerar el contenido de **audio** independiente de los vídeos, como archivos de sonido o música.

2.4.1. Procesamiento de Habla

En el caso de que el audio contenga habla, como en podcasts, grabaciones o conferencias, se aplican técnicas de ASR con modelos robustos como Whisper. Esto permite obtener una transcripción textual que se convierte en contenido indexable, facilitando búsquedas por palabras clave o semántica mediante el uso de LLMs o embeddings textuales.

2.4.2. Audio No Verbal

Para el audio que no es voz, como sonidos ambientales, música o efectos sonoros, existen modelos como **Contrastive Language-Audio Pretraining (CLAP)**. Este entrena conjuntamente un codificador de audio y uno de texto, lo que permite buscar efectos de sonido a partir de descripciones textuales ("sonido de lluvia", "pasos en la grava") y facilita la clasificación cero-shot de audio.

2.4.3. Modelos Generadores de Descripciones Auditivas

Complementariamente, modelos como **AudioCaption** de Microsoft pueden generar frases descriptivas de clips de audio. Esta capacidad permite describir cada archivo de sonido en formato textual, indexar dichas descripciones y, en consecuencia, facilitar un acceso más semántico al contenido auditivo, más allá de simples metadatos.

2.5. Comparativa de Modelos Representativos

La tabla 2.1 resume algunos modelos representativos, destacando la distinción entre modelos propietarios como ChatGPT y una creciente diversidad de iniciativas abiertas. Para el desarrollo de un sistema como **LLMSearch**, los módulos open-source son particularmente relevantes. Es factible combinar herramientas como MiniCPM-V, Moondream, Whisper y CLAP para construir un sistema completo: Whisper se encargaría de la transcripción de audio; CLAP, del indexado de sonidos no verbales; Moondream o BLIP-2, de la descripción de imágenes; y un LLM generalista como Vicuna o LLaMA podría orquestar la interacción conversacional y la fusión de información.

Modelo	Modalidades	Tamaño	Características principales
ChatGPT (GPT-4)	Texto (y visión en GPT-4V)	>100 B?	LLM propietario de OpenAI, rendimiento puntero en comprensión y generación de lenguaje.
MiniCPM-V 2.5	Texto, Imágenes, Vídeo, Audio	~8 B	Open-source, eficiente para despliegue en dispositivos; consultas multimodales.
Moondream 2	Imágenes–Texto	2 B	VLM ultraligero con VQA, captioning, detección y OCR en CPU en tiempo real.
Whisper	Audio–Texto	~1.6 B	ASR multilingüe de código abierto, muy robusto ante acentos y ruido.

Tabla 2.1: Comparativa de modelos representativos en lenguaje y multimodalidad.

2.6. Conclusión

El estado del arte actual ofrece los bloques fundamentales necesarios para construir un buscador multimedia avanzado que opere mediante lenguaje natural. Este TFG se propone integrar y adaptar estas tecnologías de vanguardia en una única plataforma unificada, denominada *LLMSearch*. El proyecto evaluará el rendimiento de esta integración y buscará proponer mejoras con el objetivo de lograr búsquedas multimodales más precisas, naturales e intuitivas para el usuario.

3. Objetivos

3.1. Objetivo general

El objetivo principal de este TFG es diseñar y desarrollar un buscador multimedia inteligente que permita a los usuarios realizar búsquedas avanzadas utilizando lenguaje natural. De esta manera, el usuario podrá localizar documentos de texto, imágenes, vídeos o archivos de audio explorando el contenido semántico intrínseco de los ficheros, trascendiendo las limitaciones de las búsquedas basadas únicamente en metadatos explícitos.

La idea es crear una herramienta que facilite a los usuarios encontrar contenido multimedia de manera eficiente y precisa mediante descripciones detalladas en lenguaje natural. Por ejemplo, se podría buscar una fotografía específica entre miles con una consulta como: "busca una foto en la que salía un elefante levantando la trompa y que la hice en Tailandia hace unos 5 o 6 años"; o encontrar un archivo PDF relevante mediante una búsqueda del tipo: "encuentra los datos para la declaración de la renta de 2016". De esta forma, se pretende obtener un sistema de búsqueda que no solo identifique el archivo específico que se busca, sino que también tenga la capacidad de extraer datos relevantes del contenido del archivo para responder a preguntas específicas formuladas en la consulta, aprovechando las capacidades de los modelos de lenguaje aumentados por recuperación (RAG).

3.2. Objetivos específicos

Adicionalmente, se plantean los siguientes objetivos secundarios que complementan y dan soporte al objetivo principal:

3.2.1. Estudiar modelos multimodales

Estudiar diferentes modelos multimodales con el fin de seleccionar aquellos que ofrezcan los mejores resultados en términos de precisión y eficiencia (tiempo de respuesta razonable).

3.2.2. Seleccionar una solución de base de datos

Investigar y seleccionar una solución de base de datos adecuada para el almacenamiento y consulta eficiente de metadatos enriquecidos y embeddings vectoriales generados por los modelos de IA.

3.2.3. Diseñar una arquitectura modular

Diseñar una arquitectura de sistema que sea modular, escalable y eficiente, permitiendo la integración de los diferentes componentes y facilitando futuras expansiones o mejoras.

3.2.4. Desarrollar una interfaz gráfica

Desarrollar una interfaz gráfica de usuario (GUI) intuitiva y amigable que permita a los usuarios interactuar fácilmente con el sistema, realizar búsquedas, visualizar los resultados obtenidos y gestionar sus archivos.

4. Metodología

En este capítulo se detalla la metodología empleada para la planificación, desarrollo y gestión del presente TFG. Se describirá tanto la organización del proyecto, basada en una adaptación de la metodología ágil Scrum, como el entorno técnico configurado, abarcando el hardware y software utilizados. El objetivo es proporcionar una visión clara de los procesos y herramientas que han sustentado la realización de LLMSearch, desde su concepción hasta la implementación de sus funcionalidades.

4.1. Organización del Proyecto y Metodología Scrum Adaptada

La gestión y desarrollo del presente TFG se ha articulado mediante una adaptación simplificada de la metodología ágil **Scrum**. Scrum es un marco de trabajo diseñado para abordar proyectos complejos, promoviendo la autoorganización de los equipos, el desarrollo iterativo e incremental a través de ciclos cortos denominados *sprints*, y la entrega continua de valor.

4.1.1. Adaptación de Roles y Dinámicas de Scrum

Dada la naturaleza individual del proyecto, donde un único estudiante es el responsable de su ejecución, los roles tradicionales de Scrum se han concentrado en esta figura. Así, el estudiante ha asumido las responsabilidades de:

- **Product Owner**: Definiendo la visión del producto (LLMSearch), gestionando el *Product Backlog* (lista priorizada de funcionalidades y requisitos) y asegurando que el desarrollo se alinea con los objetivos del proyecto.
- **Development Team**: Encargándose del diseño, implementación, pruebas y entrega de los incrementos funcionales del software en cada sprint.
- **Scrum Master**: Facilitando el proceso, eliminando impedimentos, asegurando que se sigan las prácticas ágiles adaptadas y promoviendo la mejora continua.

En este contexto adaptado, el tutor del TFG ha desempeñado un rol fundamental como **cliente principal (Stakeholder)**, proporcionando los requisitos iniciales, ofreciendo retroalimentación continua sobre los avances y validando los entregables. Su participación ha sido clave para guiar la dirección del proyecto y definir posibles ajustes a lo largo de su desarrollo.

4.1.2. Estructura y Ejecución de los Sprints

El proyecto se ha dividido en una serie de *sprints*, cada uno con una duración aproximada de dos semanas. Al inicio de cada cuatrimestre, y de manera continua, se establecieron reuniones periódicas (equivalentes a las *Sprint Planning* y *Sprint Review* de Scrum) entre el estudiante y el tutor. En estas reuniones se:

- Revisaba el progreso del sprint anterior.
- Se presentaban y discutían los avances realizados (incremento del producto).
- Se resolvían dudas y se abordaban los impedimentos identificados.
- Se definían y priorizaban los objetivos y tareas para el siguiente sprint, conformando el *Sprint Backlog*.

La planificación de los sprints ha sido un proceso dinámico, ajustándose a la evolución del proyecto y los descubrimientos realizados. A continuación, se describe de forma general la progresión del trabajo a lo largo de los sprints:

- **Sprint Inicial (Fase de Conceptualización e Investigación)**: Este sprint se centró en la definición detallada del alcance del proyecto, la elaboración del estado del arte, la investigación exhaustiva de las tecnologías y herramientas de IA pertinentes (especialmente LLMs y modelos multimodales), y la organización inicial de las tareas. Se sentaron las bases para la arquitectura del sistema.
- **Sprints de Desarrollo del Backend y Núcleo de IA (Fase de Construcción I)**: Durante estos ciclos, el foco principal fue el diseño y la implementación de la arquitectura del sistema backend. Esto incluyó el desarrollo de los módulos encargados de la lógica de negocio, la gestión de datos y, crucialmente, la integración inicial de los modelos de IA seleccionados para el procesamiento de texto, imágenes y otros formatos multimedia.
- **Sprints de Desarrollo de la Interfaz y Orquestación (Fase de Construcción II)**: Paralelamente o a continuación, se abordó el desarrollo de la interfaz de usuario (frontend), buscando una experiencia intuitiva para la interacción mediante lenguaje natural. Se implementó un orquestador de tareas para gestionar las diferentes operaciones del buscador (indexación, consulta, recuperación multimodal). Asimismo, se estableció la comunicación entre el frontend y el backend, típicamente a través de una Application Programming Interface (API) REST, para asegurar un flujo de datos coherente.
- **Sprints de Integración Avanzada y Pruebas (Fase de Refinamiento)**: Estos sprints se dedicaron a la integración completa de todos los componentes del sistema, con especial atención a la interacción fluida entre los modelos de IA y el resto de la aplicación. Se llevaron a cabo pruebas de rendimiento para evaluar la eficiencia del buscador bajo grandes cargas de datos y se realizaron pruebas de usabilidad para garantizar que la interfaz cumplía con los requisitos de accesibilidad y facilidad de uso.
- **Sprints Finales (Fase de Consolidación y Documentación)**: Los últimos ciclos de desarrollo se enfocaron en la corrección de errores (bug fixing), la optimización de funcionalidades existentes, la incorporación de mejoras basadas en las pruebas y la retroalimentación recibida. Una parte significativa de este periodo se dedicó también a la elaboración de la documentación técnica del proyecto y la memoria del TFG.

4.1.3. Gestión de Tareas y Adaptabilidad

Para cada sprint, el estudiante elaboró una lista de tareas en sucio (equivalente al *Sprint Backlog*) a partir de los objetivos definidos. El progreso de estas tareas se monitorizó de forma continua, marcando aquellas completadas para mantener un control efectivo del avance y anotando las posibles dudas e inquietudes para comentarlas con el tutor en el siguiente sprint. El proceso de desarrollo seguía un ciclo de ideación (definición de la funcionalidad o mejora) seguido de su implementación y prueba.

Es importante destacar que, en consonancia con los principios ágiles, el plan del proyecto no fue rígido. A medida que se avanzaba, se identificaron nuevos desafíos técnicos, se descubrieron herramientas más adecuadas o surgieron limitaciones imprevistas. Esta realidad condujo a la redefinición de algunas tareas y al ajuste de los objetivos de ciertos sprints, siempre en comunicación con el tutor, para asegurar la viabilidad y la calidad del resultado final. Esta flexibilidad fue fundamental para navegar la complejidad inherente a un proyecto de investigación y desarrollo como LLMSearch.

4.2. Apartado técnico

Para la ejecución y desarrollo del presente TFG, se ha dispuesto del siguiente entorno técnico, tanto a nivel de hardware como de software. Esta configuración ha sido la base sobre la cual se han realizado todas las pruebas, desarrollos y validaciones del sistema propuesto.

4.2.1. Equipamiento Hardware

El equipo informático utilizado para el desarrollo del proyecto cuenta con las siguientes especificaciones:

- **Procesador (Central Processing Unit (CPU)):** AMD Ryzen 9 7900X3D 4.4GHz/5.6GHz
- **Memoria (Random Access Memory (RAM)):** Corsair Vengeance RGB DDR5 6000MHz 64GB 2x32GB CL30
- **Tarjeta Gráfica (Graphics Processing Unit (GPU)):** RTX 4070 Ti SUPER Trinity 16GB GDDR6X
- **Almacenamiento (Solid State Drive (SSD)):** NVMe Samsung 970 EVO Plus de 1TB
- **Sistema Operativo (Operating System (OS)):** Windows 11 Pro / Ubuntu 22.04 LTS

4.2.2. Software y Herramientas de Desarrollo

La selección del software y las herramientas de desarrollo ha sido crucial para garantizar un flujo de trabajo eficiente y productivo. Para el **Integrated Development Environment (IDE)**, se ha optado por **Visual Studio Code (VS Code)**. Esta elección se fundamenta en su ligereza, su amplia gama de extensiones que facilitan el desarrollo en múltiples lenguajes

(especialmente Python, previsiblemente central en un proyecto con LLMs), su depurador integrado, y su excelente integración con sistemas de control de versiones como Git.

Precisamente, para el **control de versiones**, se ha utilizado **Git**, el estándar de facto en la industria, gestionando los repositorios a través de **GitHub**. Esta plataforma no solo permite un seguimiento exhaustivo de los cambios y la experimentación segura mediante ramas, sino que también facilita la colaboración (aunque en este proyecto sea individual, es una buena práctica) y ofrece un respaldo del código en la nube.

Considerando la naturaleza del proyecto, que involucra el uso intensivo de modelos de lenguaje y otras bibliotecas de IA, se ha empleado Python como uno de los lenguajes de programación principales, decisión que se justificará más adelante en el desarollo. Para la **gestión de entornos y paquetes** de Python, se ha utilizado **pip**, el instalador de paquetes estándar de Python. Su simplicidad y eficacia permiten manejar las dependencias del proyecto de manera ordenada, asegurando la reproducibilidad del entorno de desarrollo en diferentes sistemas si fuera necesario.

En cuanto a la validación de la interfaz de usuario, si el proyecto la incluye, las pruebas se realizarán en una selección de navegadores web modernos. Principalmente, se utilizará **Google Chrome**, en su versión más reciente, debido a su amplia cuota de mercado y sus robustas herramientas integradas para desarrolladores, lo que facilita la depuración y asegura una alta compatibilidad con la mayoría de los usuarios. Adicionalmente, se realizarán pruebas en **OperaGX**, también en su última versión. La elección de OperaGX responde, en parte, a que es el navegador principal utilizado por el desarrollador, lo que agiliza las pruebas iterativas y la verificación rápida de cambios durante el ciclo de desarrollo. Aunque ambos navegadores comparten el motor Chromium, permitiendo una base de compatibilidad similar, esta doble comprobación ayuda a identificar posibles particularidades menores y asegura una experiencia de usuario consistente en un entorno familiar para el desarrollador.

Finalmente, para la **documentación** del proyecto, se ha recurrido a **LaTeX**, utilizando la distribución **MiKTeX**. LaTeX es la herramienta por excelencia para la redacción de documentos técnicos y científicos, gracias a su insuperable calidad tipográfica, su manejo eficiente de referencias bibliográficas, y su capacidad para estructurar documentos complejos. Complementariamente, para la creación de diagramas y esquemas visuales, se ha empleado **Excalidraw**, una herramienta online que permite generar diagramas de forma rápida y con un estilo claro y moderno, facilitando la comunicación de ideas y arquitecturas complejas.

5. Análisis, Especificación y Diseño

5.1. Arquitectura del Sistema

La arquitectura de LLMSearch se ha concebido como un sistema modular y distribuido, con el objetivo de facilitar la escalabilidad, el mantenimiento y la posible reutilización de componentes. Un esquema visual inicial de esta arquitectura se presenta en la Figura 5.1.

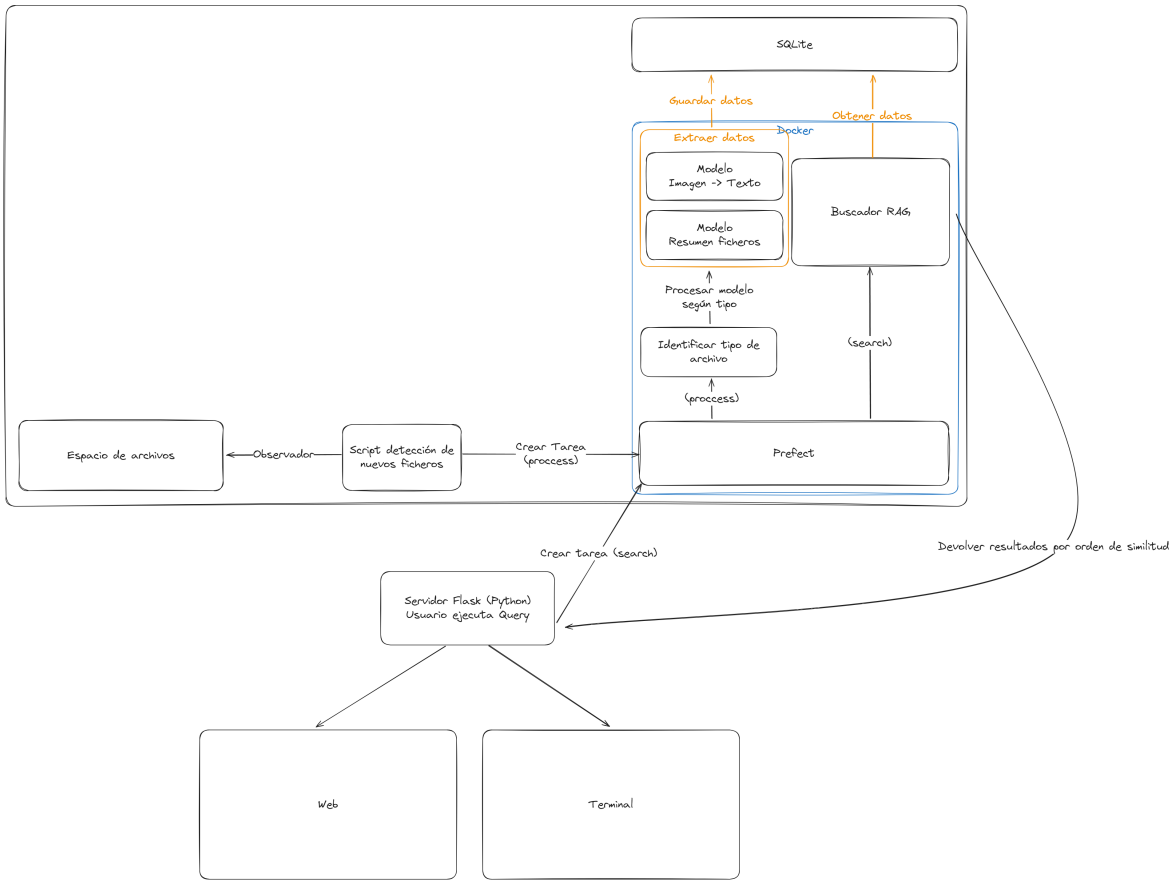

Figura 5.1: Diagrama de la arquitectura general de LLMSearch.

Es importante señalar que el diagrama de la Figura 5.1 representa una instantánea conceptual de las primeras etapas del diseño del proyecto. Si bien captura la esencia de la modularidad y los flujos principales, ciertos componentes y sus interacciones han sido refinados o modificados durante el proceso de desarrollo para optimizar el rendimiento, la simplicidad o la adecuación a las herramientas finalmente seleccionadas. Por ejemplo, la especificación y el

tipo de la base de datos han evolucionado desde la concepción inicial. Los siguientes apartados describen en detalle cada componente en su estado final de implementación, destacando las decisiones de diseño clave y cualquier desviación significativa respecto al esquema preliminar.

5.1.1. Componentes Principales de la Arquitectura

- **Script Observador (Watcher)**: Este componente es el encargado de monitorizar de forma continua el directorio o directorios especificados por el usuario (según RC-01) en busca de nuevos ficheros o modificaciones en los existentes (RF-01). Para su implementación, se ha seleccionado la librería `watchdog` de Python, conocida por su eficiencia y compatibilidad multiplataforma para la observación de eventos del sistema de ficheros. Cuando detecta un cambio relevante, el observador notifica al servidor para iniciar el proceso de análisis del fichero.
- **Servidor Central (Backend API)**: El núcleo del sistema reside en una aplicación servidor desarrollada en Python utilizando el microframework **Flask**. Este servidor actúa como punto central de comunicación y control. Sus responsabilidades principales son:
  1. Inicializar y gestionar el ciclo de vida del `Script Observador`.
  2. Recibir notificaciones del observador sobre nuevos ficheros o ficheros modificados.
  3. Exponer una API RESTful para atender las peticiones provenientes tanto de la interfaz web (RF-05) como de la interfaz de línea de comandos (Command Line Interface (CLI)) (RF-07). Estas peticiones incluyen las consultas de búsqueda de los usuarios y, potencialmente, comandos de gestión y configuración del sistema (RNF-07).
  4. Interactuar con el Orquestador de Tareas para delegar el procesamiento de ficheros y la ejecución de búsquedas.
- **Orquestador de Tareas (Task Orchestrator)**: Dada la naturaleza asíncrona y potencialmente intensiva en recursos del procesamiento de ficheros y las consultas a LLMs, se ha decidido incorporar un sistema de orquestación de tareas. Se ha optado por **Prefect**, una moderna herramienta de orquestación de flujos de trabajo en Python. El servidor Flask enviará solicitudes de creación de tareas al servicio de Prefect, que se encargará de encolarlas, ejecutarlas (posiblemente en procesos separados o workers), monitorizar su estado y gestionar reintentos o fallos. Las tareas principales gestionadas por Prefect serán:
  1. **Tarea de Procesamiento (Process Task)**: Al recibir la ruta de un nuevo fichero, esta tarea coordinará varias subtareas:
     - Invocará al `Identificador del Tipo de Fichero` para determinar la naturaleza del archivo (RF-02).
     - En función del tipo, seleccionará y ejecutará el `Modelo de Extracción de Datos` correspondiente (RF-03).
     - Paralelamente, se extraerán metadatos generales del fichero (nombre, tamaño, fechas, etc.).

   - Finalmente, todos los datos extraídos (contenido semántico, metadatos) se persistirán en la `Base de Datos` (RF-04).

2. **Tarea de Búsqueda (Search Task)**: Cuando el usuario realiza una consulta, esta tarea:
   - Recibirá la consulta en lenguaje natural del usuario como parámetro.
   - Realizará una primera fase de recuperación de información relevante de la `Base de Datos`.
   - Construirá un prompt optimizado, incorporando la información recuperada y la consulta original, para ser procesado por el `Buscador RAG` (utilizando un LLM).
   - Devolverá un conjunto de resultados ordenados por relevancia o similitud con la consulta (RF-06).

- **Identificador del Tipo de Fichero (File Type Identifier)**: Para asegurar una correcta clasificación de los ficheros (RF-02) y evitar depender únicamente de la extensión (que puede ser engañosa o incorrecta), este módulo analizará las cabeceras o "números mágicos" de los ficheros para determinar su formato real. Se utilizarán librerías de Python especializadas para esta tarea, como `python-magic` o similares, que permiten una identificación robusta de una amplia variedad de tipos de archivo.

- **Modelos de Extracción de Datos (Data Extraction Models)**: Este es un conjunto de módulos especializados, cada uno diseñado para procesar un tipo específico de fichero (texto, imagen, vídeo, audio, etc., según RF-10). La arquitectura de estos modelos será eminentemente modular, permitiendo la fácil incorporación de nuevos extractores para formatos de archivo futuros o la actualización de los existentes. Cada modelo será responsable de invocar las herramientas de IA o librerías pertinentes (ej. OCR para imágenes, ASR para audio, análisis semántico para texto) para extraer la información significativa y estructurarla.

- **Base de Datos (Database)**: Para el almacenamiento persistente de la información extraída de los ficheros y los metadatos asociados (RF-04), se utilizará una base de datos **SQLite**. Esta elección se basa en su simplicidad, su naturaleza embebida (no requiere un servidor de base de datos separado, facilitando la configuración local) y su buena integración con Python a través de la librería estándar `sqlite3`. Se desarrollará una capa de acceso a datos (funciones CRUD y otras operaciones específicas) en Python para interactuar con la base de datos de manera estructurada y segura.

  Nota sobre la estructura de la Base de Datos: La definición detallada del esquema de la base de datos (tablas, campos, relaciones) se abordará en una fase posterior del diseño, una vez se tenga una comprensión completa de los datos específicos que cada modelo de extracción generará y los requisitos de consulta del buscador.

- **Buscador RAG (Retrieval-Augmented Generation Engine)**: El componente central para la búsqueda semántica (RF-06) se basará en la técnica de RAG. Este enfoque combina la recuperación eficiente de información de la `Base de Datos` con las

capacidades de comprensión y generación de lenguaje natural de un LLM. Durante el análisis, se han considerado varias estrategias para implementar el RAG:

- *Generación de Consultas SQL*: El LLM podría generar consultas Structured Query Language (SQL) para interrogar directamente la base de datos. Si bien es una opción, presenta el riesgo de generar consultas subóptimas o incorrectas, y podría limitar la expresividad semántica si el LLM no comprende bien el esquema.
- *Incorporación de Texto Completo al Prompt*: Convertir fragmentos relevantes de la base de datos a texto e incluirlos directamente en el prompt del LLM. El principal desafío aquí es la limitación de la ventana de contexto de muchos LLMs. Aunque modelos recientes (como algunos modelos chinos con ventanas de contexto muy amplias) podrían mitigar esto, requiere una filtración previa muy efectiva de la información de la base de datos para no exceder los límites o incurrir en altos costos computacionales.
- *Uso de Embeddings para Contexto*: Transformar los datos recuperados y la consulta del usuario en embeddings (representaciones vectoriales) y utilizar estos embeddings para enriquecer el contexto del LLM. Esta suele ser la aproximación más eficiente y robusta, aunque puede implicar una mayor complejidad en la implementación de la infraestructura de embeddings y la gestión de la similitud vectorial.

La elección final de la estrategia RAG o una combinación de ellas dependerá de la experimentación y la evaluación del rendimiento y la complejidad.

5.1.2. Consideraciones sobre Contenerización (Docker)

Durante la fase de análisis, se evaluó la posibilidad de utilizar **Docker** para la contenerización de los diferentes servicios del sistema. Docker ofrece ventajas significativas en términos de reproducibilidad del entorno, aislamiento de dependencias y simplificación del despliegue. Sin embargo, para la etapa actual del proyecto, y dado que el objetivo principal es desarrollar y validar la funcionalidad central en un entorno local, se ha optado por no implementar Docker inicialmente. La gestión de dependencias a través de entornos virtuales de Python (`venv`) y la configuración directa de los servicios (Flask, Prefect, SQLite) en el sistema operativo local se considera suficiente y más ágil para el desarrollo iterativo. No obstante, la arquitectura modular propuesta facilitaría una futura migración a una infraestructura basada en Docker si el proyecto escalara o se requiriera un despliegue en entornos más complejos.

5.2. Requisitos del sistema

En esta sección se detallan los requisitos del sistema, divididos en requisitos funcionales, no funcionales y de configuración. Los requisitos se han estructurado en formato tabular para facilitar su comprensión y seguimiento durante el desarrollo del proyecto.

5.2.1. Requisitos funcionales

Los requisitos funcionales describen el comportamiento que debe tener el sistema, las funcionalidades que debe ofrecer y las operaciones que debe realizar.

ID	Nombre	Descripción
RF-01	Detección de ficheros	El sistema debe detectar nuevos ficheros en el directorio observado.
RF-02	Diferenciación de tipos	El sistema debe diferenciar el tipo de archivo a analizar (texto, imagen, vídeo, audio, otros).
RF-03	Ejecución de modelos	El sistema debe ejecutar el modelo correspondiente que extraerá la información del fichero a la base de datos.
RF-04	Almacenamiento	El sistema debe almacenar todos los datos posibles sobre el fichero analizado en una base de datos.
RF-05	Interfaz web	El sistema debe tener una interfaz web super-simple donde el usuario podrá escribir su consulta en lenguaje natural y darle a un botón para realizar la búsqueda.
RF-06	Resultados de búsqueda	El sistema responderá con un conjunto de resultados potencialmente interesantes a partir de la consulta de búsqueda, ordenados de más a menos ”interesante”.
RF-07	Entrada por línea de comandos	El sistema debe tener una entrada por línea de comandos (ej: `LLMSearch --query "mapa del mundo en el que hay marcados los mejores parques naturales"`).
RF-08	Presentación de resultados	El resultado será la ruta del fichero junto a una pequeña descripción del mismo (enlaces clicables al fichero y a la carpeta que lo contiene).
RF-09	Inspección de archivos comprimidos	Los ficheros comprimidos deberían poder inspeccionarse por dentro.
RF-10	Tipos de ficheros a procesar	El sistema debe procesar los siguientes tipos de ficheros: - Documentos de texto - Imágenes - Vídeos - Ficheros de sonido - Otros (bases de datos, ejecutables, etc.)

Tabla 5.1: Requisitos funcionales del sistema

5.2.2. Requisitos no funcionales

Los requisitos no funcionales especifican criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos.

ID	Nombre	Descripción
RNF-01	Configuración web	La web debe tener una pequeña parte de configuración discreta pero accesible en todo momento.
RNF-02	Arquitectura modular	La arquitectura se debe dividir en un "buscador" y un "explorador" y deben ser completamente separadas para poder ser reutilizadas.
RNF-03	Ejecución sin GPU	El sistema debe poder ejecutarse en un ordenador sin GPU (opcional).
RNF-04	Parámetro de consulta	La entrada por línea de comandos aceptará un parámetro `--query` junto al término de búsqueda.
RNF-05	Resultados en CLI	La entrada por línea de comandos devolverá los resultados de la misma manera que el buscador web con la diferencia de que solo devolverá información adicional si se le añade el parámetro `--verbose`.
RNF-06	Estado del sistema	La entrada por línea de comandos tendrá un parámetro `--status` que devolverá el estado del sistema: número de archivos procesados sobre el número total de archivos en observación, cantidad de ficheros de cada tipo, errores encontrados...
RNF-07	Configuración por CLI	Se añadirán los parámetros necesarios para poder configurar el sistema desde línea de comandos.

Tabla 5.2: Requisitos no funcionales del sistema

5.2.3. Requisitos de configuración

Los requisitos de configuración especifican las opciones que el usuario debe poder ajustar en el sistema.

ID	Nombre	Descripción
RC-01	Directorio de observación	Directorio donde se están observando nuevos ficheros.
RC-02	Regulación de carga	Regular la carga (limitar la CPU al X%).
RC-03	Tipo de modelo LLM	Tipo de modelo LLM a utilizar (Local *(LLM Studio)* ó en la nube).
RC-04	Búsqueda por imagen	Posibilidad de poner una foto de una persona y que la busque en los ficheros.

Tabla 5.3: Requisitos de configuración del sistema

5.3. Casos de uso

Los casos de uso describen las interacciones típicas entre los usuarios y el sistema, mostrando cómo se utilizarían las funcionalidades principales.

6. Desarrollo

La construcción de un sistema inteligente para la búsqueda y gestión de archivos personales requiere la integración de diversas tecnologías y herramientas consolidadas en el ámbito del desarrollo de software y la inteligencia artificial. Este capítulo tiene como objetivo revisar el estado del arte de los componentes tecnológicos clave que se han considerado o que forman la base para la implementación del presente proyecto. Se analizarán diferentes opciones en áreas fundamentales como la orquestación de tareas, la detección de cambios en el sistema de archivos, las soluciones de bases de datos para el almacenamiento de metadatos y embeddings, la contenerización para el despliegue y, finalmente, los frameworks para el desarrollo de la interfaz de usuario. Esta revisión permitirá contextualizar las decisiones de diseño tomadas y justificar la selección de las herramientas específicas utilizadas en el desarrollo de la solución.

6.1. Orquestador de tareas

La gestión eficiente de flujos de trabajo complejos, especialmente aquellos que involucran procesamiento de datos y tareas de machine learning, es crucial para el sistema propuesto. Un orquestador de tareas permite automatizar, programar y monitorizar estas secuencias de operaciones. A continuación, se presentan las herramientas que se han estudiado en el presente trabajo para seleccionar la más adecuada para la coordinación de las diversas etapas del procesamiento de archivos.

6.1.1. Prefect

Prefect se presenta como una moderna plataforma de orquestación de flujos de trabajo, escrita principalmente en Python. Está diseñada específicamente para permitir a los desarrolladores diseñar, programar, ejecutar y monitorizar pipelines de datos y flujos de machine learning de manera fiable y escalable, con un enfoque en la simplicidad y la experiencia del desarrollador.

6.1.1.1. Ventajas

- **Facilidad de uso:** Prefect ofrece una sintaxis intuitiva y una configuración sencilla, lo que facilita la definición y gestión de flujos de trabajo complejos.
- **Flexibilidad:** Permite la orquestación de tareas en entornos locales, en la nube o híbridos, adaptándose a diversas necesidades.
- **Monitoreo y gestión:** Incluye herramientas integradas para el monitoreo, registro y manejo de errores en tiempo real.

6.1.1.2. Desventajas

- **Madurez:** Aunque ha ganado popularidad, Prefect es relativamente nuevo en comparación con otras herramientas más consolidadas.
- **Comunidad:** Su comunidad es más pequeña, lo que puede limitar la disponibilidad de recursos y soporte.

6.1.2. Kafka

Apache Kafka es un sistema de mensajería distribuido de código abierto, reconocido por su alto rendimiento y capacidad para manejar flujos de datos en tiempo real. Aunque su función principal es la de broker de mensajes, a menudo se utiliza en arquitecturas complejas para desacoplar sistemas y como parte de pipelines de datos más amplios, pudiendo actuar como un componente en la orquestación de eventos.

6.1.2.1. Ventajas

- **Alto rendimiento:** Kafka es conocido por su capacidad para manejar grandes volúmenes de datos con baja latencia.
- **Escalabilidad:** Diseñado para escalar horizontalmente, puede manejar cargas de trabajo crecientes de manera eficiente.
- **Ecosistema robusto:** Cuenta con una amplia gama de herramientas y conectores que facilitan su integración con otros sistemas.

6.1.2.2. Desventajas

- **Complejidad:** La configuración y gestión de Kafka pueden ser complejas, especialmente para usuarios sin experiencia previa.
- **Requisitos de recursos:** Para un rendimiento óptimo, Kafka suele requerir una infraestructura robusta, lo que puede ser excesivo para proyectos más pequeños.

6.1.3. Airflow

Apache Airflow es una plataforma de código abierto ampliamente adoptada para la creación, programación y monitorización programática de flujos de trabajo. Originalmente desarrollada por Airbnb, permite definir flujos de trabajo como Grafos Acíclicos Dirigidos (DAGs) de tareas, utilizando Python para su definición.

6.1.3.1. Ventajas

- **Popularidad y comunidad:** Amplia adopción y una comunidad activa que proporciona numerosos recursos y soporte.
- **Flexibilidad:** Permite la programación y monitoreo de flujos de trabajo complejos.

6.1.3.2. Desventajas

- **Curva de aprendizaje:** Puede ser complejo de configurar y requiere conocimientos avanzados para su implementación efectiva.

6.2. Script de detección de cambios en un path y Servidor

Un componente esencial del sistema es la capacidad de detectar automáticamente la creación, modificación o eliminación de archivos dentro de los directorios personales del usuario. Esta funcionalidad es la que desencadena el proceso de análisis y catalogación. En esta sección se revisan diferentes lenguajes y bibliotecas que ofrecen mecanismos para el monitoreo del sistema de archivos y que podrían servir de base para implementar un servidor que gestione estas detecciones.

6.2.1. Python

Python, debido a su versatilidad y extenso ecosistema de bibliotecas, ofrece múltiples opciones para la detección de cambios en el sistema de archivos. Estas soluciones varían en cuanto a su dependencia de la plataforma y su nivel de abstracción.

- **Watchdogs:** Es una biblioteca multiplataforma escrita en Python, diseñada específicamente para observar eventos del sistema de archivos (como creación, eliminación, modificación de archivos o directorios) de manera eficiente.

- **pyinotify:** Esta biblioteca es un wrapper de Python para la API inotify del kernel de Linux, lo que permite monitorear eventos del sistema de archivos de forma muy eficiente en sistemas Linux, pero no es portable a otras plataformas.

- **inotify-simple:** Se trata de un wrapper más sencillo y ligero alrededor de la API inotify de Linux, que ofrece una interfaz de programación más directa para tareas básicas de monitoreo de archivos en sistemas Linux.

- **inotifyx:** Similar a pyinotify, esta biblioteca proporciona acceso al subsistema inotify de Linux. Está diseñada para ofrecer una API estable, pero su uso también está restringido a plataformas Linux.

- **Polling Methods:** En plataformas donde mecanismos como inotify no están disponibles, o para requisitos más simples, se puede implementar un mecanismo de sondeo (polling). Este método implica verificar periódicamente el estado del sistema de archivos para detectar cambios, aunque puede ser menos eficiente.

6.2.2. Node.js

Node.js, como entorno de ejecución de JavaScript del lado del servidor, también proporciona herramientas robustas para interactuar con el sistema de archivos, incluyendo el monitoreo de cambios.

- **chokidar:** Es una biblioteca popular y eficiente para Node.js que normaliza y mejora el comportamiento de fs.watch y fs.watchFile, ofreciendo una solución multiplataforma robusta para vigilar cambios en el sistema de archivos.

6.2.3. Java

Java, siendo una plataforma madura y ampliamente utilizada, incluye en su biblioteca estándar (NIO.2) mecanismos nativos para el monitoreo de eventos del sistema de archivos.

- **WatchService:** Es una API integrada en Java (desde Java 7) que permite registrar un directorio (o directorios) con el servicio de vigilancia. Este servicio monitorea los cambios en los objetos registrados y los encola para su procesamiento.

6.2.4. C++/C/C#

Los lenguajes de la familia C (C++, C, C#) ofrecen acceso a APIs de bajo nivel del sistema operativo o bibliotecas específicas de cada plataforma para implementar la vigilancia de archivos, permitiendo un control granular y un alto rendimiento.

- **FileSystemWatcher:** En el ecosistema .NET (C#), la clase FileSystemWatcher permite monitorear cambios en el sistema de archivos (archivos y directorios) y reaccionar a ellos. Para C/C++, se suele recurrir a las APIs específicas del sistema operativo como inotify en Linux o ReadDirectoryChangesW en Windows.

6.2.5. Go

Go, conocido por su eficiencia y concurrencia, cuenta con bibliotecas desarrolladas por la comunidad para el monitoreo del sistema de archivos, aprovechando las capacidades nativas de los sistemas operativos.

- **fsnotify:** Es una biblioteca popular en Go para el monitoreo del sistema de archivos que proporciona una interfaz común sobre las APIs específicas de cada plataforma (inotify, kqueue, FSEvents, ReadDirectoryChangesW).

6.2.6. Rust

Rust, un lenguaje de programación moderno enfocado en la seguridad y el rendimiento, también dispone de bibliotecas multiplataforma para la notificación de eventos del sistema de archivos.

- **notify:** Es una biblioteca de Rust multiplataforma para vigilar cambios en el sistema de archivos, ofreciendo una API unificada sobre diferentes backends específicos del sistema operativo.

6.3. Base de datos

El almacenamiento persistente de metadatos, tanto los extraídos directamente de los archivos como los generados por la IA (incluyendo embeddings vectoriales), es fundamental. La elección de la base de datos impacta en el rendimiento, la escalabilidad y la facilidad de consulta. Se exploran opciones relacionales y no relacionales, cada una con sus propias fortalezas y debilidades.

6.3.1. Relacional

Las bases de datos relacionales se caracterizan por su estructura tabular, esquemas predefinidos y el uso de SQL (Structured Query Language) para la manipulación de datos. Son adecuadas para datos con relaciones bien definidas y donde la consistencia transaccional (ACID) es prioritaria.

6.3.1.1. SQLite

SQLite es un sistema de gestión de bases de datos relacional autocontenido, sin servidor, que no requiere configuración y es transaccional (ACID). Toda la base de datos (definiciones, tablas, índices y los propios datos) se almacena como un único archivo en el disco del host, lo que la hace extremadamente portable y fácil de integrar en aplicaciones.

6.3.1.1.1. Ventajas

- **Ligereza y simplicidad:** SQLite es una biblioteca de base de datos integrada que no requiere una configuración de servidor independiente, lo que facilita su implementación y uso.
- **Portabilidad:** Al almacenar toda la base de datos en un único archivo, es fácil de transferir y gestionar, especialmente útil para aplicaciones móviles o integradas.
- **Rendimiento en entornos de bajo recurso:** Funciona eficientemente incluso en sistemas con recursos limitados.

6.3.1.1.2. Desventajas

- **Concurrencia limitada:** SQLite permite múltiples lecturas simultáneas, pero las escrituras se gestionan de una en una, lo que puede ser un cuello de botella en aplicaciones con alta concurrencia de escritura.
- **Escalabilidad:** No está diseñada para manejar grandes volúmenes de datos o aplicaciones que requieren escalabilidad horizontal.

6.3.1.2. MariaDB

MariaDB Server es un popular sistema de gestión de bases de datos relacional de código abierto, creado por los desarrolladores originales de MySQL. Está diseñado para ser un reemplazo directo de MySQL, ofreciendo mayor velocidad, nuevas características y una comunidad más abierta y vibrante.

6.3.1.2.1. Ventajas

- **Rendimiento y escalabilidad:** MariaDB ofrece un alto rendimiento y puede manejar una gran cantidad de transacciones, siendo adecuada para aplicaciones empresariales.
- **Compatibilidad con MySQL:** Como un fork de MySQL, mantiene una alta compatibilidad, facilitando la migración desde MySQL.
- **Soporte para almacenamiento en columnas:** Incluye el motor ColumnStore, optimizado para cargas de trabajo analíticas.

6.3.1.2.2. Desventajas

- **Complejidad en la configuración:** Requiere una configuración y gestión más complejas en comparación con SQLite.
- **Requisitos de recursos:** Necesita más recursos del sistema, lo que puede ser excesivo para aplicaciones pequeñas o integradas.

6.3.2. No relacional

Las bases de datos no relacionales, o NoSQL, ofrecen modelos de datos flexibles, escalabilidad horizontal y están optimizadas para tipos específicos de datos o patrones de acceso. Son una alternativa cuando la rigidez de los esquemas relacionales es una limitación o se requiere un alto rendimiento para grandes volúmenes de datos no estructurados o semiestructurados. Dentro de esta categoría, las bases de datos vectoriales han ganado prominencia para aplicaciones de IA.

6.3.2.1. MongoDB

MongoDB es una base de datos NoSQL orientada a documentos, de código abierto y multiplataforma. Almacena los datos en documentos flexibles similares a JSON (en formato BSON), lo que permite que los campos varíen de un documento a otro y que la estructura de los datos cambie con el tiempo, facilitando la evolución de las aplicaciones.

6.3.2.1.1. Ventajas

- **Flexibilidad del esquema:** Al ser una base de datos NoSQL orientada a documentos, permite almacenar datos en un formato flexible similar a JSON, adaptándose fácilmente a cambios en la estructura de los datos.
- **Escalabilidad horizontal:** Diseñada para escalar horizontalmente mediante sharding, lo que facilita el manejo de grandes volúmenes de datos y altas tasas de tráfico.
- **Alto rendimiento en operaciones de lectura/escritura:** Optimizada para manejar operaciones simultáneas de lectura y escritura de manera eficiente.

6.3.2.1.2. Desventajas

- **Consumo de recursos:** Requiere una cantidad significativa de recursos, especialmente en implementaciones a gran escala.
- **Falta de soporte para transacciones complejas:** Aunque MongoDB ha mejorado en este aspecto con transacciones multi-documento ACID, las transacciones complejas que involucran múltiples colecciones pueden no ser tan robustas o directas como en bases de datos relacionales tradicionales.

6.3.2.2. ChromaDB

ChromaDB es una base de datos vectorial de código abierto diseñada específicamente para facilitar el desarrollo de aplicaciones con IA que requieren el almacenamiento y la búsqueda de embeddings. Permite a los desarrolladores añadir capacidades de memoria a largo plazo y búsqueda semántica a sus modelos de lenguaje y otras aplicaciones de aprendizaje automático de forma sencilla. Puede operar en memoria para prototipado rápido o utilizar almacenamiento persistente en disco.

6.3.2.2.1. Ventajas

- **Especializada en embeddings:** Su diseño está optimizado para almacenar, gestionar y realizar búsquedas de similitud eficientes sobre grandes cantidades de vectores de embeddings.
- **Facilidad de uso y API intuitiva:** Ofrece una API simple, especialmente para desarrolladores de Python, lo que reduce la curva de aprendizaje y acelera la integración.
- **Integraciones con ecosistema de IA:** Se integra de forma nativa con frameworks populares como LangChain y LlamaIndex, simplificando la construcción de flujos de trabajo de IA.
- **Ligera y embebible:** Puede ejecutarse localmente sin necesidad de un servidor complejo, siendo adecuada para desarrollo, prototipado y aplicaciones más pequeñas.
- **Código abierto:** Permite su uso sin restricciones de licencia y fomenta la contribución de la comunidad.

6.3.2.2.2. Desventajas

- **Madurez y escalabilidad para producción masiva:** Aunque evoluciona rápidamente, puede no tener la misma robustez o características avanzadas de escalabilidad horizontal y gestión de clústeres que soluciones de bases de datos vectoriales más maduras o servicios gestionados en la nube para cargas de trabajo extremadamente grandes.
- **Funcionalidades de BD tradicional limitadas:** No está diseñada para ser una base de datos de propósito general. Carece de soporte para consultas relacionales complejas, transacciones ACID en el sentido tradicional o esquemas rígidos que ofrecen las bases de datos SQL.

- **Operaciones y gestión avanzada:** Para despliegues a gran escala, las herramientas de monitorización, backup, y recuperación pueden ser menos sofisticadas en comparación con sistemas de bases de datos más establecidos.

6.4. Docker

La contenerización se ha convertido en un estándar para el desarrollo, despliegue y ejecución de aplicaciones, garantizando la consistencia entre diferentes entornos y simplificando la gestión de dependencias. Docker es la plataforma líder en este ámbito. Docker es una plataforma de software de código abierto que permite automatizar el despliegue de aplicaciones dentro de contenedores de software ligeros y portátiles. Un contenedor empaqueta una aplicación y todas sus dependencias, bibliotecas y archivos de configuración necesarios para que se ejecute de forma aislada.

6.4.1. Ventajas

- **Portabilidad:** Los contenedores Docker aseguran que el software se ejecute de manera consistente en cualquier entorno que soporte Docker, desde el portátil del desarrollador hasta servidores de producción en la nube o locales.
- **Aislamiento:** Cada componente del sistema puede ejecutarse en su propio contenedor, con sus propias dependencias, evitando conflictos entre ellas y con el sistema anfitrión.
- **Facilidad de despliegue:** Simplifica significativamente la distribución y actualización de aplicaciones, permitiendo ciclos de desarrollo y despliegue más rápidos y fiables.

6.4.2. Desventajas

- **Consumo de recursos:** Aunque los contenedores son más ligeros que las máquinas virtuales, el uso de Docker y sus contenedores introduce una capa adicional que consume recursos del sistema (CPU, memoria, disco).
- **Complejidad adicional:** La gestión de contenedores, redes, volúmenes y la orquestación de múltiples contenedores (e.g., con Docker Compose o Kubernetes) requiere conocimientos específicos sobre Docker y sus conceptos asociados.

6.5. Interfaz

La interfaz de usuario (UI) es el punto de interacción principal del usuario con el sistema, permitiéndole realizar búsquedas, visualizar resultados y gestionar sus archivos. La elección de un framework de desarrollo para la UI impacta en la experiencia del usuario, la velocidad de desarrollo y la mantenibilidad de la aplicación. Se consideran varios frameworks web modernos.

6.5.1. Angular

Angular, desarrollado por Google, es un framework de desarrollo de aplicaciones web basado en TypeScript, completo y opinado. Proporciona una plataforma integral para construir aplicaciones web complejas y escalables de una sola página (SPA), ofreciendo herramientas para routing, gestión de estado, y más, directamente desde su núcleo.

6.5.1.1. Ventajas

- **Framework completo:** Angular ofrece una solución integral con herramientas integradas para el desarrollo de aplicaciones web robustas.
- **Arquitectura estructurada:** Su naturaleza opinada y el uso de TypeScript facilitan la escalabilidad y el mantenimiento de aplicaciones complejas a largo plazo.

6.5.1.2. Desventajas

- **Curva de aprendizaje pronunciada:** Requiere tiempo para dominar conceptos como TypeScript, RxJS, y la inyección de dependencias, que son fundamentales en Angular.
- **Complejidad innecesaria para proyectos simples:** Su estructura y conjunto de herramientas pueden resultar excesivos para aplicaciones pequeñas o con funcionalidades limitadas.

6.5.2. React

React, mantenido por Meta (anteriormente Facebook) y una comunidad de desarrolladores individuales y empresas, es una biblioteca de JavaScript de código abierto para construir interfaces de usuario o componentes de UI. Se enfoca principalmente en la capa de vista (el "V" en MVC) y a menudo se utiliza junto con otras bibliotecas para construir aplicaciones completas.

6.5.2.1. Ventajas

- **Biblioteca flexible:** React se centra en la construcción de interfaces de usuario, permitiendo la integración con una amplia variedad de bibliotecas y herramientas según las necesidades específicas del proyecto (e.g., para routing, gestión de estado).
- **Amplia comunidad y recursos:** Su popularidad asegura una gran cantidad de tutoriales, bibliotecas de terceros, y soporte por parte de una comunidad activa y extensa.

6.5.2.2. Desventajas

- **Necesidad de configuraciones adicionales:** Para funcionalidades más allá de la UI básica (como enrutamiento o gestión de estado global), es necesario integrar y configurar bibliotecas adicionales, lo que puede aumentar la complejidad inicial del proyecto.

6.5.3. Vue

Vue.js (comúnmente referido como Vue) es un framework de JavaScript de código abierto, progresivo y accesible, utilizado para construir interfaces de usuario y aplicaciones de una sola página. Se distingue por su facilidad de integración con proyectos existentes y otras bibliotecas, y por su diseño que permite adoptarlo gradualmente.

6.5.3.1. Ventajas

- **Simplicidad y facilidad de uso:** Vue es conocido por su curva de aprendizaje suave y su documentación clara, lo que permite una adopción rápida por parte de los desarrolladores.

- **Flexibilidad:** Adecuado tanto para proyectos pequeños donde se integra en partes de una página existente, como para el desarrollo de aplicaciones de una sola página más complejas.

6.5.3.2. Desventajas

- **Menor adopción en grandes empresas:** Aunque está ganando popularidad rápidamente, Vue aún no tiene la misma penetración en entornos corporativos grandes en comparación con Angular o React, lo que podría traducirse en menos ofertas de empleo o recursos específicos para escenarios empresariales muy complejos.

6.5.4. Astro

Astro es un framework web moderno diseñado para construir sitios web rápidos y centrados en el contenido. Su principal característica es la arquitectura de "islas" (Astro Islands), que permite renderizar componentes de UI en el servidor y enviar menos JavaScript al cliente por defecto, mejorando significativamente el rendimiento de carga y la interactividad inicial.

6.5.4.1. Ventajas

- **Optimización para contenido estático y rendimiento:** Astro está diseñado para generar sitios web estáticos o renderizados en servidor (SSR) muy rápidos, lo que es beneficioso para aplicaciones donde el rendimiento y el SEO son críticos.

- **Integración con otros frameworks:** Permite utilizar componentes de UI escritos en React, Vue, Svelte, y otros frameworks populares dentro de los proyectos Astro, ofreciendo flexibilidad al desarrollador.

6.5.4.2. Desventajas

- **Menor madurez para aplicaciones altamente interactivas:** Aunque soporta componentes interactivos, su enfoque principal en el contenido estático y la minimización de JavaScript del lado del cliente puede hacerlo menos ideal para aplicaciones web muy complejas y altamente dinámicas en comparación con SPAs tradicionales.
- **Ecosistema en crecimiento:** Al ser relativamente nuevo, Astro puede carecer de la amplitud de recursos, herramientas y comunidad que tienen otros frameworks más establecidos, aunque está creciendo rápidamente.

7. Resultados

7.1. Evaluación

TODO: Meter por aquí lo de ChromaDB y pruebas de rendimiento.

8. Conclusiones

A. Anexo I

Aquí vendría el anexo I