

LLMSearch: buscador multimedia basado en lenguaje natural

Dades del Treball Final d'Estudis

Títol del treball * LLMSearch: buscador multimedia basado en lenguaje natural

Resum *

El objetivo de este TFG es diseñar y desarrollar un buscador multimedia que permita a los usuarios realizar búsquedas avanzadas utilizando lenguaje natural. El objetivo final es poder buscar documentos de texto, imágenes, vídeos o archivos de audio usando el lenguaje natural para explorar el contenido de los ficheros, no solo los metadatos de los archivos.

La idea es crear una herramienta que permita a los usuarios encontrar contenido multimedia de manera eficiente y precisa utilizando descripciones detalladas en lenguaje natural. Por ejemplo, se podría buscar una fotografía entre miles con una descripción 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 excel con una búsqueda del tipo "encuentra los datos para la declaración de la renta de 2016".

Objectius concrets

- Analizar los modelos de IA que podrían utilizarse para resolver los problemas que el proyecto plantea
- Especificar los requisitos funcionales y no funcionales del sistema
- Diseñar la arquitectura de la aplicación
- Desarrollo del proyecto usando un lenguaje de programación adecuado para el alcance del mismo


Metodologia que s'utilitzarà

- Durante todo el proceso se utilizará una metodología ágil con entregas y revisiones periódicas, usando herramientas de planificación y seguimiento de tareas
- Se utilizará un sistema de control de versiones distribuido (Git, Subversion...)

Relació amb assignatures cursades i/o itinerari relacionat

Asignaturas relacionadas con Bases de Datos, Programación e Ingeniería del Software (Análisis y Especificación de Sistemas Software, etc).

Bibliografia o fonts d'informació



Peticiones del cliente

Comprender qué quiere y necesita el cliente

Objetivo principal

- Implementar un sistema que permita explorar el contenido de los documentos de texto, imágenes, vídeos o archivos de audio más allá de sus metadatos describiéndolo en lenguaje natural.
- Diseñar y desarrollar un buscador multimedia que permita a los usuarios realizar búsquedas avanzadas utilizando lenguaje natural.

Idea general

- Poder buscar con lenguaje natural (“documento de texto en el que planifiqué el viaje a China”).
- Para ello, es necesario previamente tener una descripción de los ficheros:
 - Foto001.jpg: “Imagen dónde se ve un payaso sentado sobre una silla con el fondo....”
 - Viaje.doc: “Documento relativo a la planificación de un viaje a China en el que se pretende recorrer....”
 - Musica.mp3: “Canción del grupo musical XX titulada XX con una duración de... La canción trata acerca de ...”
- ¡No se va a entrenar un modelo de IA, se van a usar los preexistentes!

Qué quiere el cliente: arquitectura

- Interfaz gráfico super-simple, tipo Google (un campo *input* para introducir la búsqueda y un botón para empezar a buscar)
- Hay una pequeña parte de configuración, a la que se debe poder acceder en cualquier momento, pero mantenerla discreta.
- Aparte del interfaz gráfico, para facilitar la comunicación con otros elementos y las pruebas, debe poderse buscar desde línea de comandos, tipo:

```
LLMSearch --query "mapa del mundo en el que hay marcados los mejores parques naturales"
```
- La arquitectura se debe dividir en un “buscador” y un “explorador” y deben ser completamente separadas para poder ser reutilizadas.
- El rendimiento va a depender mucho del hardware, pero estaría bien que se pudiera ejecutar, aunque le costara, incluso desde un ordenador sin GPU.

Qué quiere el cliente: buscador

- Cuando un usuario realice una búsqueda, se le devolverá un conjunto de resultados potencialmente interesantes a partir de su consulta de búsqueda (*query*).
- Estos resultados se ordenarán siempre antes de mostrarlos según el criterio de más a menos “interesante”.
- El resultado será el PATH (ruta) al fichero encontrado, junto con una pequeña descripción del mismo y enlaces clicables al fichero y la carpeta contenedora (algo similar a los resultados de búsqueda de Google, pero aplicado a un sistema de ficheros local).

Qué quiere el cliente: línea de comandos

- Aceptará un parámetro *--query* con el término de búsqueda.
- Estos resultados se ordenarán siempre antes de mostrarlos según el criterio de más a menos “interesante”.
- El resultado será el PATH (ruta) al fichero encontrado (uno por cada línea de texto).
- Si se añade el parámetro *--verbose* se añadirá también una pequeña descripción de cada uno de los resultados.
- Habrá otro 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...
- Se añadirán los parámetros necesarios para poder configurar el sistema desde línea de comandos.

Qué quiere el cliente: explorador

- Se deberán poder explorar diversos tipos de ficheros:
 - Documentos de texto
 - Imágenes
 - Vídeos
 - Ficheros de sonido
 - Otros
- En la medida de lo posible se ha de inspeccionar el **contenido** de esos ficheros (mínimo en los documentos de texto e imágenes), pero aunque no se explore el contenido, sí que se debe crear una mínima descripción a partir de los **metadatos** (p.e. los ficheros MP3 habitualmente contienen nombre del grupo musical, album y canción).

Python:
librería “watchdog”
C/C++:
inotify

Qué quiere el cliente: configuración

- Se debe poder configurar qué directorio es objeto de la búsqueda de ficheros (¡no es todo el disco duro!). Ese directorio y recursivamente todos sus subdirectorios, deberán tenerse en cuenta única y exclusivamente para las búsquedas.
- Como el funcionamiento de este programa va a requerir un gran esfuerzo computacional, si es posible, habría que añadir algún tipo de selector para poder regular la carga (p.e. limitar el uso de la CPU al X%).

Qué quiere el cliente: generalidades

- Un requisito deseable, aunque no imprescindible, sería que en la configuración se pudieran subir algunas fotos de personas en concreto para que luego el sistema pudiera reconocerlas, por ejemplo:
 - “busca una foto en la que está Maria subida en una moto en la playa”
- Al menos, se espera que el buscador sea capaz de buscar en el contenido de documentos de texto y en imágenes y en los metadatos de los ficheros de audio, pero debería estar abierto a que en un futuro se pudieran añadir otro tipo de formatos como vídeo, bases de datos (.SQL, .db), ejecutables...
- Los ficheros comprimidos, inicialmente se pueden tratar como tal, pero idealmente se debería inspeccionar también los ficheros internos que lo componen.
- El resultado final no debería tener dependencias de otras aplicaciones funcionando (no hay problema en usar librerías externas, pero mejor si no se usan aplicaciones externas que deban estar funcionando, tipo LLMStudio).

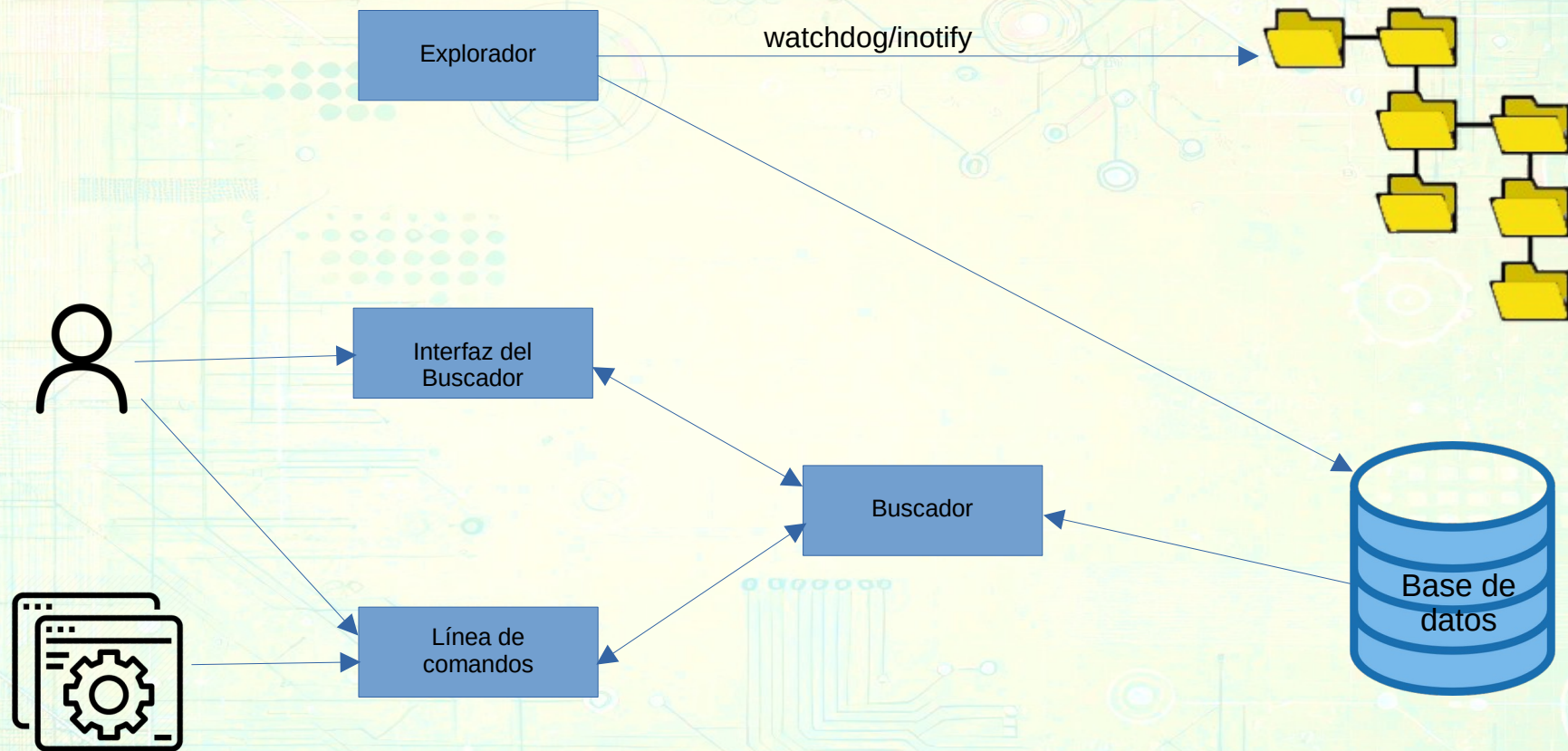


Aportaciones del tutor

Para facilitar el desarrollo

Arquitectura recomendada

- Plataforma: Linux ofrece mejor integración y hay más opciones que en Windows.
- La mejor estrategia en este tipo de aplicaciones suele ser dividir la funcionalidad en una especie de arquitectura cliente-servidor
 - El cliente sería el **interfaz** de usuario (por línea de comandos o gráfico)
 - **Buscador**: El servidor sería un Daemon o Servicio (Linux / Windows) que en segundo plano esperaría a que un cliente se conecte y solicite una búsqueda y en tal caso, la realizaría y devolvería el resultado
 - **Explorador**: Existiría un segundo daemon/servicio que se encargaría todo el rato de comprobar si hay algún fichero nuevo para el que no se haya generado su descripción y en ese caso la generaría
- Los principales motivos para mantenerlo todo como un daemon/servicio en segundo plano son:
 - No bloquear el interfaz de usuario y poder controlar la sobrecarga del sistema durante la ejecución
 - Mantener el modelo de deep learning cargado en memoria para no tener que hacerlo cada vez (es una operación costosa)



Base de datos

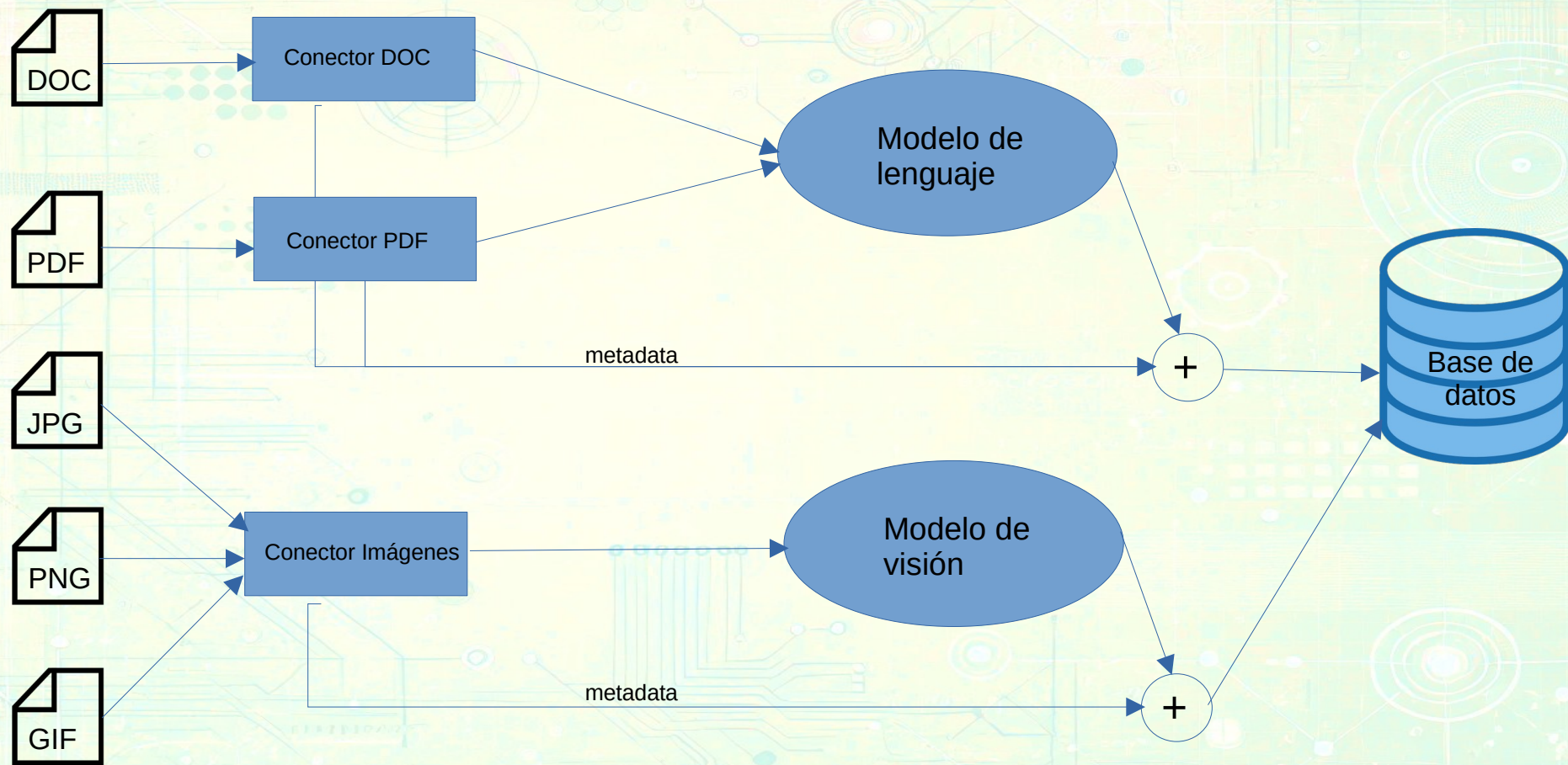
- De algún modo, esta aplicación necesita recurrir a una base de datos.
- Cuando el *Explorador* encuentra un nuevo fichero, generará para el mismo una descripción en forma de texto que hay que almacenar para que posteriormente el *Buscador* pueda usarla.
- Por eficiencia, esta BBDD debería estar en local y ser lo más ligera posible (SQLite, podría ser una buena opción o incluso podrías crear tu propia BBDD basada en un fichero de texto).
- El explorador tiene que tener en cuenta los ficheros que se puedan eliminar y cuya entrada siga en la BBDD (ficheros fantasma que no queremos tener en cuenta en las búsquedas).

Implementación de los modelos

- Los LLM, modelos de visión, etc se pueden implementar al menos de 3 formas:
 - 1) Como servicio, conectando con una API (por ejemplo, ChatGPT 4o): muy caro, pero muy potente porque se puede acceder a los modelos con más parámetros.
 - 2) En local, cargando el modelo memoria y ejecutándolo en CPU o GPU: económico, pero puede ser lento y limitado a los modelos más pequeños por la falta de memoria.
 - 3) En un cloud dedicado (pagando en función del uso de la GPU de un servidor remoto): caro, pero se pueden llegar a usar modelos medianos y grandes.
- También se puede usar durante el desarrollo Google Colab que ofrece gratis una GPU en la nube.
- Lo mejor sería implementar una capa de abstracción que permitiera al usuario elegir qué modelo necesita (p.e. si tiene un hardware potente y quiere usar un buen modelo en local, podría decantarse por la opción 2; otro usuario que tenga un hardware muy limitado, pero no le importe gastar para obtener una gran calidad podría ir a la opción 1; un usuario menos exigente, con un hardware medio, podría elegir un modelo pequeño con la opción 2...)

Explorador: arquitectura modular

- Lo más importante al diseñar la arquitectura de esta parte es que sea completamente modular, de manera que sea fácil añadir nuevos módulos (habitualmente a estos elementos se les llama “*conector*”).
- Esto se consigue en POO con herencia o interfaces, cada conector es la implementación de una clase interfaz dada.
- Así, si en una primera versión solo se puede buscar en ficheros de texto, para añadir ficheros de imagen en la segunda versión, simplemente habría que implementar una interfaz ya existente y registrar los nuevos tipos de archivo.
- Es importante no confiar en la extensión del fichero, sino mirar en las cabeceras del fichero de qué formato se trata en realidad (existen librerías para ello).
- Tener en cuenta que es posible que un conector pueda utilizar un modelo de deep learning diferente a otro conector (p.e. el conector de vídeo podría usar el modelo X y el conector de ficheros de texto el modelo Y porque al probarlo ves que es más eficiente así). Uno de los problemas que enfrentarás será la posibilidad de tener varios modelos cargados al mismo tiempo.



Buscador: RAG

- **Retrieval-Augmented Generation** es una técnica en IA que combina dos procesos: la recuperación de información (retrieval) y la generación de texto (generation). Funciona buscando información relevante en una base de datos o conjunto de documentos (retrieval) y luego utiliza un LLM para procesar y generar una respuesta o contenido basado en esa información.
- Los RAG pueden funcionar de varios modos:
 - Generan una consulta SQL para buscar en la BBDD (menor eficiencia, problema: pérdida de resultados válidos)
 - Incorporan al prompt la BBDD convertida en texto (problema: ventana de contexto)
 - Incorporan como embeddings al contexto del modelo (mayor eficiencia, problema: ventana de contexto y más complejo)

Buscador: prompt

- Para buscar entre los archivos identificados por el proceso *Explorador* y teniendo la BBDD accesible (RAG) para un modelo LLM, la búsqueda se puede hacer creando *prompts*.
- Solo necesitamos un procesador del lenguaje natural, un LLM. Se puede reusar alguno de los modelos que usábamos en el *Explorador* o usar otro diferente si nos viene mejor o mejora la calidad (es posible que usemos uno más pequeño en el *Explorador* y una mejor en el *Buscador*). Lo más importante en el *Buscador* es la adherencia al prompt y que la ventana de contexto sea suficientemente grande para que quepa toda la BBDD.
- Ejemplo de búsqueda para la *query* “vídeo de un chico lanzando dardos”:
 - Creación de un *prompt* que se le pasa al LLM Buscador: “Tenemos una base de datos con el formato ... Busca entre todas las entradas aquellas que podrían identificarse como ‘vídeo de un chico lanzando dardos’ e indícame su identificador”
 - Sería posible (probar) incluir en el *prompt* una petición para que las clasificara
 - Para limitar la búsqueda es posible hacer un filtrado previo, por ejemplo, como en la *query* está la palabra ‘vídeo’ que la podemos localizar fácilmente, podemos descartar otros tipos de formatos.
 - Se puede llevar más allá el filtrado y eliminar de la entrada aquellos ficheros que no contengan ninguna palabra de la *query* (si no contiene ‘vídeo’, ‘chico’, ‘lanzando’ o ‘dardos’, no se tiene en cuenta). Esto puede acelerar, pero también dar problemas con las formas verbales y sinónimos (‘chico lanzando dardos’ – ‘niño va a lanzar algo que lleva en la mano’).

Proyectos similares

- **Google** está probando “Ask Photos”
- **PhotoPrism**: Similar a LLMSearch, pero con un enfoque claro en el tagging de fotos. ¿Qué modelos usan?
- **Recognize**: app para nextCloud que permite reconocer quién aparece en las fotos y qué aparece en las fotos. También identifica géneros musicales en los ficheros de sonido. Su modelo funciona con CPU.

Modelos útiles

- **Moondream**: pequeño modelo de visión
 - Interesante también **este vídeo** de una aplicación práctica construida sobre Moondream (en general en este canal hay muchos proyectos interesantes usando librerías de IA).
- **MiniCPM-V**: pequeño modelo multimodal. Funciona incluso en CPU, capaz de describir imágenes y vídeos con bastante precisión.
- **JoyCaption**: permite describir imágenes; hay un vídeo explicando cómo instalarlo y usarlo **aquí**.
- **Mini-Omni**: otro modelo multimodal, incluyendo voz, lo cual, en principio no nos sirve de nada (a no ser que se incluya un conector que revise lo que se dice en ficheros de audio).
- **Bitnet.cpp**: es un framework para usar modelo cuantizados a 1-bit usables en CPUs. **Aquí** hay un vídeo tutorial para ver cómo usarlo.

Características avanzadas (trabajos futuros)

- Más conectores para aceptar otros tipos de ficheros
- Permitir que el sistema realice ciertas acciones (p.e. “quiero que suene la melodía de ‘Star wars’” y LLMSearch buscaría un fichero de audio con esa descripción y lo lanzaría con la aplicación que esté configurada para ello en el sistema).
- Integrar LLMSearch en las barras de búsqueda de Windows o Linux
- Búsqueda por voz

Por dónde empezar

- Empieza con la descomposición en tareas del proyecto (historias de usuario o requisitos)
- Decídete por una arquitectura concreta
- Prueba modelos pequeños:
 - Busca otros modelos open source (además de los ya comentados), instálalos en tu equipo y prueba con ellos cual podría funcionar para (1) describir texto e imágenes y (2) con las descripciones de un conjunto de ficheros, realizar la búsqueda de características.