****

**Aprovechamiento de macrodatos abiertos de proyectos de I+D con modelos de lenguaje de gran tamaño**

|  |  |  |
| --- | --- | --- |
| Titulación:  Máster en Big Data y Ciencia de Datos  Curso académico  2024 – 2025 | Alumno/a:  Moltó Quiles, Ernesto  D.N.I: 45928665B  Director/a de TFM: Desirée Ruiz Ponce, Vanessa Moscardo García, Yudith Cardinale | Convocatoria:  Primera o Segunda |

00 Mes 2023

Índice

[Resumen 5](#_Toc201488158)

[1. Introducción 6](#_Toc201488159)

[2. Objetivos 8](#_Toc201488160)

[3. Estado del Arte y Marco teórico 9](#_Toc201488161)

[4. Desarrollo del proyecto y resultados 11](#_Toc201488162)

[4.1. Metodología 11](#_Toc201488163)

[4.2. Planteamiento del problema 11](#_Toc201488164)

[4.3. Desarrollo del proyecto 12](#_Toc201488165)

[4.4. Resultados 12](#_Toc201488166)

[5. Conclusión y trabajos futuros 13](#_Toc201488167)

[6. Referencias 14](#_Toc201488168)

[Apéndice I 15](#_Toc201488169)

[Anexos I 16](#_Toc201488170)

*¡¡¡INFORMATIVO!!! (quitar)*

*\*Actualizar tabla de contenidos*

*\*Hacer uso de salto de página para los nuevos apartados.*

*\*Hacer uso de los estilos definidos: Normal, Título1, Título2, Título3.*

Índice de ilustraciones

[Ilustración 1. Arquitectura Jupyter Cliente - Servidor. Fuente: https://www.paradigmadigital.com/dev/jupyter-data-science-aplicada/ 4](#_Toc6399225)

*¡¡¡INFORMATIVO!!! (quitar)*

*\*Actualizar índice de tablas (Referencias / Insertar tabla de ilustraciones)*

Índice de tablas

Tabla 1. Operaciones matemáticas utilizadas en el estudio realizado. Elaboración propia. 7

Tabla 3 resume las principales diferencias entre los cuatro modelos de embeddings evaluados. 7

*¡¡¡INFORMATIVO!!! (quitar)*

*\*Actualizar índice de tablas (Referencias / Insertar tabla de ilustraciones)*

# Resumen

Consistente en una breve descripción (menos de una página, generalmente de 200 a 300 palabras) del trabajo a realizar, con sus requisitos y especificaciones, mencionando explícitamente si se basa en trabajos previos realizados por el tutor del alumno, un proyecto anterior o similar.

Las faltas de ortografía no son admisibles en un trabajo académico como un TFM asociado a Estudios Superiores.

Un TFM no es un diario personal. Trata de evitar la primera persona. Por ejemplo, en lugar de, “~~En mi opinión el modelo xxxxxx es muy bueno...~~” cambiar a “A la vista de los datos analizados o fuentes consultadas el modelo está obteniendo buenos resultados…”. Está permitido dar tu opinión siempre que lo precise y esté argumentada.

La claridad en la redacción del texto para la compresión del mismo es fundamental. Trata descomponer oraciones excesivamente compuestas en simples. Evita en la medida de lo posible oraciones subordinadas.

Evita usar estructuras generales o estereotipos del tipo “~~Como afirman numerosos expertos…~~” y sustitúyelos por “De acuerdo a los datos o información obtenidos …”

Evita usar las expresiones demasiado coloquiales, el humor o la ironía.

Usa sinónimos para evitar repetir el mismo término varias veces. Lee continuamente el texto para evitar contenido repetido o inconsistente.

Se recomienda incluir una versión en inglés(Abstract) al final del resumen.

**Palabras clave:** Entre 4 y 8 palabras clave (*keywords*).

# Introducción

En los últimos años, se han producido grandes avances en el campo de la inteligencia artificial (IA) , gracias a la convergencia de diversos factores como el aumento de la capacidad computacional, la disponibilidad masiva de datos, y la mejora en el diseño de los algoritmos. Esto ha favorecido el desarrollo de soluciones inteligentes, que se han ido integrando de forma paulatina en numerosos sectores productivos, desde el comercio electrónico y la asistencia sanitaria, hasta la automatización industrial y los servicios de atención al cliente. En este contexto, el procesamiento de lenguaje natural (NLP) se ha consolidado como una de las áreas de mayor interés y crecimiento dentro de la IA, al centrar esfuerzos en dotar a las máquinas de la capacidad para comprender, procesar y generar lenguaje humano de forma efectiva.

Uno de los avances más representativos en este ámbito ha sido la aparición de los Modelos de Lenguaje de Gran Tamaño (*Large Language Models*, LLMs), capaces de generar texto coherente y contextualizado a partir de una entrada en lenguaje natural. Estos modelos, entrenados con enormes volúmenes de datos y estructuras de redes neuronales avanzadas, han demostrado un rendimiento sobresaliente en tareas como la generación de texto, la traducción automática, la resolución de preguntas, la generación de código y la asistencia en toma de decisiones. Sin embargo, a pesar de su gran capacidad, presentan tres limitaciones principales: su conocimiento es estático (limitado al momento de su entrenamiento), carecen de información especializada en dominios concretos, y pueden generar contenidos no verídicos o imprecisos, fenómeno conocido como “alucinación”.

Con el fin de paliar estas limitaciones, han surgido estrategias como el fine-tuning, la ingeniería de prompts (prompt enginering) o Retrieved Augmented Generation (RAG)una técnica que permite enriquecer las respuestas del modelo mediante la incorporación de fragmentos relevantes recuperados desde una base de conocimiento externa. Esta última técnica ha demostrado ser eficaz en contextos donde se requiere una precisión elevada y una actualización constante de la información [Patrick Lewis, 2020].

En este marco, el presente Trabajo de Fin de Máster (TFM) surge como una continuación del estudio desarrollado por Desirée Ruiz Ponce centrado en la exploración de bases de datos abiertas de proyectos europeos de I+D, concretamente el programa Horizon 2020, mediante el uso de LLMs combinado con técnicas RAG. Su trabajo permitió avanzar hacia la creación de un asistente inteligente capaz de asistir al personal de centros tecnológicos en tareas clave como la búsqueda de líneas temáticas afines a su actividad investigadora.

En esta línea, el presente trabajo realiza una ampliación de dicha propuesta mediante una reconfiguración completa del proceso de recuperación y generación. En primer lugar, se ha implementado un modelo LLM local utilizando la plataforma Ollama y el modelo llama3.2:latest 8b, ejecutado en entorno local vía HTTP, lo que proporciona independencia del uso de APIs externas y garantiza un mayor control sobre los recursos. El modelo se ha configurado con temperatura cero para asegurar un comportamiento determinista.

En segundo lugar, se ha sustituido los embeddings usados en el trabajo anterior por dos modelos más ligeros y versátiles: sentence-transformers/paraphrase-MiniLM-L6-v2 y intfloat/e5-small-v2, ambos ejecutados en CPU con el fin de garantizar la compatibilidad con equipos de recursos limitados, se ha utilizado un portátil con las siguientes especificaciones: 477 GB de almacenamiento, tarjeta gráfica integrada Intel® Iris® Xe, 16 GB de memoria RAM y un procesador Intel® Core™ i7-1165G7 de 11ª generación con una frecuencia base de 2.80 GHz. La vectorización semántica se ha gestionado a través de la librería FAISS, sobre la cual se construyó la base vectorial que permite una recuperación eficiente de fragmentos relevantes. FAISS ha sido preferida frente a otras opciones como ChromaDB debido a su alto rendimiento en entornos con recursos limitados y a su eficiencia en operaciones de búsqueda de vecinos más cercanos (*nearest neighbors*) a gran escala. Mientras que Chroma ofrece una integración sencilla con herramientas de *retrieval-augmented generation* y una interfaz más orientada a desarrolladores, FAISS destaca por su madurez, optimización en CPU y soporte avanzado para índices comprimidos, lo que resulta más adecuado en contextos donde la velocidad y la eficiencia en memoria son prioritarias [Johnson, 2019].

Una novedad fundamental introducida en este trabajo es el uso del componente MultiQueryRetriever de LangChain. Esta herramienta permite la creación automática de múltiples reformulaciones semánticas a partir de una misma consulta inicial, lo que incrementa la diversidad de resultados recuperados y mejora la cobertura de los aspectos clave solicitados por el usuario. Esta técnica resulta especialmente útil para minimizar el sesgo de interpretación de una única formulación y para enriquecer el contexto que alimenta al modelo generador.

Las motivaciones de las que han surgido este trabajo se articulan en torno a tres ejes fundamentales:

1. Optimizar la eficiencia y precisión de los asistentes basados en LLMs aplicados a dominios específicos.
2. Facilitar la integración de estos sistemas en infraestructuras modestas.
3. Explorar técnicas avanzadas de recuperación que mejoren la calidad de las respuestas generadas ante consultas complejas o ambiguas.

Este TFM pretende contribuir al diseño de asistentes conversacionales expertos, robustos, escalables y fácilmente integrables en procesos reales de investigación y gestión de proyectos. Su objetivo final es promover la creación de herramientas que automáticamente parcialmente tareas de análisis, búsqueda y consulta sobre bases de datos masivas, y que resulten especialmente útiles para profesionales técnicos que operan en centros tecnológicos, instituciones de investigación y departamentos de innovación

# Metodología.

La metodología que se ha seguido para el desarrollo de este Trabajo de Fin de Máster ha sido en X fases fundamentales:

1. Análisis de trabajo previo: En primer lugar, se realizó una lectura del Trabajo de Fin de Más de Desirée, el cual ha construido la base. Está revisión ha permitido entender el enfoque previo, identificar posibles líneas de mejora y establecer el punto de partida para el desarrollo del nuevo TFM.
2. Configuración del entorno de trabajo: Se realizo la instalación y configuración de Docker como herramienta de virtualización, con el objetico de facilitar la descarga gestión e implementación de los distintos modelos de lenguaje que se han empleado en este trabajo.
3. Generación y almacenamiento: A partir de los modelos que se han seleccionado, se han construido los correspondientes almacenes de vectores. Estos nos han permitido representar de manera eficiente el contenido semántico de los textos, siendo fundamental para las siguientes tareas.
4. Consultas y recuperación aumentada con generación (RAG): Con los almacenes vectoriales ya operativos, se llevaron a cabo consultad mediante técnicas de Recuperación Aumentada con Generación (RAG), utilizando los LLMs previamente instalados.
5. Evaluación de las respuestas generadas: Las respuestas obtenidas fueron analizadas y evaluadas, tanto cualitativa como cuantitativamente, utilizando métricas adecuadas y la propia capacidad de evaluación de los LLMs, con el fin de determinar su coherencia, precisión y utilidad.
6. Visualización de los embeddings: Se han realizado visualizaciones de los embeddings generados, con el objetivo de analizar la representación espacial en 2D de los datos y facilitar la interpretación de los resultados que se han obtenido a lo largo del proyecto.

# Estado del Arte y Marco teórico

En este capítulo se presentan los fundamentos teóricos necesarios para contextualizar las herramientas y técnicas empleadas en el desarrollo del sistema implementado. Se explican los principios de funcionamiento de la arquitectura RAG, los modelos de embeddings utilizados para la recuperación semántica, el motor de ejecución Ollama y su integración con modelos LLM locales, así como el sistema FAISS para indexación y búsqueda vectorial.

## Arquitectura RAG (Retrieval-Augmented Generatios).

Los sistemas RAG (Generación Aumentada por Recuperación) combinan dos etapas fundamentales: la recuperación de información desde una base documental externa y la generación de respuestas por parte de un modelo LLM. Este enfoque permite ampliar el conocimiento base de un modelo de lenguaje con información específica y actualizada, mejorando así la veracidad, precisión y utilidad de las respuestas generadas.

Un sistema RAG típico se compone de:

* Un motor de recuperación en similitud semántica (generalmente usando embeddings vectoriales).
* Una base vectorial indexada, como FAISS, donde se almacenan los vectores correspondientes a los documentos.
* Un modelo generativo, que recibe como entrada tanto la pregunta del usuario como los fragmentos recuperados.

Este tipo de arquitectura es especialmente útil en contextos donde los modelos de lenguaje no pueden almacenar toda la información relevante de un dominio específico en sus parámetros a entrenar.

## Modelos de embeddings.

Los embeddings vectoriales son representaciones numéricas de fragmentos de texto (palabras, frases o documentos completos) en un espacio de alta dimensión en las que la cercanía entre vectores refleja su similitud semántica. Estos vectores permiten a los sistemas de recuperación identificar documentos relevantes no solo por coincidencias literales, sino por el significado contextual de los términos. En arquitecturas como RAG, la calidad y eficiencia de los embeddings es fundamental, ya que determinan qué fragmentos se recuperan como contexto para el modelo generador.

En el trabajo original de referencia se utilizaron dos modelos ampliamente reconocidos en tareas de recuperación semántica:

* **bge-large-en**: un modelo de gran tamaño, orientado a búsqueda semántica multilingüe y tareas multietiqueta, entrenado con contrastive learning.
* **all-mpnet-base-v2**: modelo de tamaño medio, especializado en tareas de similaridad y recuperación, optimizado para el idioma inglés.

Ambos modelos demostraron un rendimiento notable en el estudio anterior, pero presentaron limitaciones significativas en lo relativo a la eficiencia computacional. Concretamente, en pruebas preliminares de este TFM se observó que la generación de los vectores llegaba a tardar en más de 48 horas en completarse, lo que haría complicado realizar las comprobaciones de los distintos experimentos.

Po este motivo, en el presente trabajo se ha decidido sustituirlos por dos modelos más ligeros y eficientes, optimizados específicamente para ejecución en CPU:

* **intfloat/e5-small-v2**: desarrollado por Microsoft, entrenado con weak supervision para tareas de recuperación. Alcanza aproximadamente un 59.9 % en el benchmark MTEB y requiere el uso explícito de prefijos como "query:" y "passage:" para su correcta alineación semántica. Destaca por su bajo peso (~50 MB) y por estar especialmente diseñado para entornos sin aceleración por GPU.
* **sentence-transformers/paraphrase-MiniLM-L6-v2**: modelo derivado de MiniLM, con arquitectura ligera y entrenado para maximizar la similitud entre frases parafraseadas. Se caracteriza por su versatilidad, rapidez de inferencia y facilidad de integración, ya que no requiere preprocesamiento adicional.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modelo | Tamaño aprox. | MTEB Score Est. | Prefijos requeridos | Optimizado para CPU |
| bge-large-en | ~400 MB | ~60–62 % | No |  |
| |  | | --- | | all-mpnet-base-v2 |  |  | | --- | |  | | ~110 MB | |  | | --- | |  |  |  | | --- | | ~58–60 % | | No | Parcialmente |
| |  | | --- | | e5-small-v2 |  |  | | --- | |  | | |  | | --- | |  |  |  | | --- | | ~50 MB | | ~59.9 % | Sí | Sí |
| |  | | --- | | paraphrase-MiniLM-L6-v2 |  |  | | --- | |  | | ~80 MB | ~53–55 % | No | Sí |

*Tabla 2 resume las principales diferencias entre los cuatro modelos de embeddings evaluados*.

## Ejecución local de modelos LLM con Ollama.

Ollama es una plataforma ligera y de código abierto que permite ejecutar modelos de lenguaje (LLMs) locamente sin necesidad de acceder a APIs externas. Ofrece una interfaz sencilla vía HTTP y compatibilidad con arquitecturas modernas como LlaMA, Mistral o Gemma, lo que permite su uso eficiente en entornos personalizados, con control completo sobre recursos, latencias y privacidad.

En el ecosistema de Ollama se encuentran disponibles diferentes modelos de lenguaje optimizados para distintas necesidades y entornos de despliegue. Aunque todos estos modelos comparten una arquitectura basada en transformes, presentan diferencias importantes en cuanto a tamaño, consumo de recursos y casos de uso ideales. En este TFM se ha trabajado con distintos modelos con el fin de obtener los mejores resultados posibles, en un tiempo optimo a partir de los recursos que se tenían.

* **gemma:2b**: Este modelo pertenece a la familia Gemma, pensada para ofrecer un buen equilibrio entre tamaño y capacidad de razonamiento. Con aproximadamente 2.000 millones de parámetros (de ahí viene el nombre “2b”), se presenta como una opción muy versátil que permite ejecutar tareas de generación de texto, asistente conversacional y resumen de información con un consumo moderado de recursos. Su tamaño de 1.7 GB facilita su uso incluso en entornos locales con GPUs de gama media o en CPU potentes, sin sacrificar en exceso la calidad de las respuestas.
* **tinyllama:1.1b**: Uno de los modelos más ligero que está diseñado específicamente para casos donde los recursos son muy limitados, como portátiles sin GPU. Con solo 637 MB de tamaño es extremadamente ágil en la carga y ejecución, sacrificando algo de capacidad en comprensión y generación de texto, pero es ideal para aplicaciones en las que la rapidez y el bajo consumo de memoria son críticos. Suele utilizarse para tareas sencillas.
* **phi3:mini**: Este modelo de aproximadamente 2,2 GB destaca por buscar un punto intermedio entre rendimiento y eficacia. Es una versión “Mini”, ya que es una versión reducida de modelos más grandes de la familia Phi, pero aun así ofrece una calidad notable en tareas de generación, reescritura y asistencia conversacional. Gracias a su mayor tamaño en comparación con tinyllama, puede producir respuestas más coherentes y detalladas.
* **llama3.2:latest**: A partir de la arquitectura LLaMA, llama3.2:latest es una versión optimizada y comprimida que conserva muchas de las fortalezas de modelos más grandes, pero con un tamaño mucho más manejable, permitiendo así ejecutarlo de manera local sin grandes requerimientos de GPU, a la vez que mantiene un buen rendimiento para tareas generales como son la generación de texto, resumen y conversación.

Estos modelos me han permitido realizar una selección con el fin de elegir el que mejor se adapta a su contexto: desde modelos ultraligeros que pueden ejecutarse prácticamente en cualquier dispositivo, hasta opciones más grandes que ofrecen una mayor capacidad de razonamiento. Gracias a Ollama, todos los modelos pueden desplegarse localmente, lo que facilita la creación de aplicaciones que respetan la privacidad y reducen la dependencia de servicios externos.

En el estudio anterior se empleó el modelo LLaMA 3 70B, es una de las versiones más grandes y potentes de la familia LLaMA, entrenada por Meta AI. Este modelo presenta un rendimiento sobresaliente en tareas generales, pero requiere infraestructuras de alto nivel (GPUs avanzadas y servidores con mucha RAM) para su ejecución.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Características | gemma:2b | tinyllama:1.1b | phi3:mini | llama3.2:latest | LLaMA 3 70B |
| Nº de parámetros | ~2 mil millones | ~1.1 mil millones | No especificado (≈2–3B) | No especificado (reducido) | 70 mil millones |
| Tamaño del modelo | 1.7 GB | 637 MB | 2.2 GB | 2.0 GB | Muy grande (decenas de GB) |
| Requiere GPU | Opcional (mejora rendimiento) | Opcional (CPU posible) | Opcional (CPU posible) | Opcional (CPU posible) | Sí (alta gama, varias GPUs) |
| Latencia de inferencia | Baja / Media | Muy baja | Baja / Media | Baja / Media | Alta |
| Casos de uso | Aplicaciones locales con más contexto | Dispositivos edge / prototipos muy ligeros | Asistentes locales, resumen y generación de texto | Prototipado / aplicaciones locales con mejor equilibrio calidad-rendimiento | Producción a gran escala, aplicaciones empresariales |

## FAISS: búsqueda vectorial eficiente.

FAISS (Facebook AI) es una biblioteca desarrollada por Meta (anteriormente Facebook) con el fin de realizar una búsqueda rápida y eficiente de vectores en espacios de alta dimensión. Su principal función es identificar los vectores más similares (vecinos más cercanos) a un vector de consulta, lo que lo convierte en una herramienta fundamental en tareas de recuperación semántica, donde se busca localizar documentos relevantes no por coincidencias léxicas, sino por proximidad en el significado.

El presente trabajo, FAISS se ha utilizado como vectorstore principal para almacenar e indexar los vectores generados a partir de los documentos preprocesados. Su implementación permite realizar búsqueda mediante vecinos aproximados más cercanos (ANN, Approximate Nearest Neighbors), lo que reduce significativamente los tiempos de inferencia sin afectar negativamente la relevancia de los fragmentos recuperados.

La elección de FAISS responde a varios factores:

* Compatibilidad directa con LangChain, lo que facilita su integración en pipelines modulares.
* Alto rendimiento en CPU, sin requerir el uso de GPU, aspecto crítico para la ejecución en entornos con recursos limitados.
* Escalabilidad y eficiencia, incluso grandes volúmenes de datos, gracias a sus algoritmos optimizados para indexación y búsqueda.

En el trabajo de referencia se utilizó la biblioteca Chroma como vectorstore, siento está una alternativa moderna y flexible, que ofrece una API sencilla y buena integración con LangChain, especialmente útil en entornos de prototipado rápido. Pero también tiene algunas limitaciones como cuando se trabaja con grandes volúmenes de datos o cuando se requiere optimizar la inferencia en CPU.

En este TFM se ha optado por FAISS debido a los siguientes motivos diferenciales frente a Chroma.

|  |  |  |
| --- | --- | --- |
| Criterios | LLaMA 3 70B | llama3.2:latest |
| Rendimiento en CPU | Alto (diseñado para eficiencia) | Bueno (pero menos optimizado) |
| Escalabilidad | Excelente para grandes corpus | Limitado a cargas medias |
| Control de indexación | Muy configurable | Más automatizado |
| Soporte comunitario | Amplio y maduro | En crecimiento |
| Integración en LangChain | Total | Total |

*Tabla 4 resume las principales diferencias entre las dos búsquedas vectoriales*.

# Desarrollo del proyecto y resultados.

En este proyecto como bien se ha iniciado anteriormente se trata de un sistema de recuperación aumentada con generación (RAG), se ha realizado un proceso en el que primeramente se crea un vector a partir de la biblioteca FAISS, para crear y realizar una búsqueda rápida y eficiente de los vectores, seguidamente estos vectores se recuperan a través de la consulta Multiquery, a partir de los fragmentos recuperados realizamos un filtro de los documentos más relevantes, por ultimo estos documentos que han sido recuperados se utilizan como el contexto para obtener una respuesta para la consulta final. Para obtener los resultados, también se han utilizado modelos LLM con el fin de obtener puntuaciones para las respuestas obtenidas.

## Recuperación semántica.

Tras haber seleccionado los modelos de *embeddings* descritos en el apartado anterior. Este proceso incluye la carga estructurada de los datos, su segmentación en fragmentos manejables (*text splitting*), la generación de embeddings y, finalmente, la construcción de índices vectoriales para permitir búsquedas rápidas y eficientes.

El objetivo general de esta fase ha sido construir diferentes configuraciones de índices vectoriales que permitiesen comparar el impacto del modelo de embeddings, el tamaño de los fragmentos y el solapamiento en el rendimiento final del sistema, así como analizar su viabilidad práctica en un entorno limitado a CPU.

* + 1. Preparación y carga de datos.

La colección documental empleada para el experimento procede de los proyectos Horizonte 2020, exportando originalmente desde un repositorio en formato CSV. Para facilitar el tratamiento de la información, se realizó una transformación previa a ficheros **.txt** estructurados, donde cada fichero contiene los datos de un único proyecto.

Para automatizar la carga y extracción de los metadatos, se ha desarrollado una función propia. Esta función ha permitido recorrer de manera recursiva un directorio que contenga ficheros **.txt** y leer cada documento.

Un aspecto clave de esta función es la extracción automática de metadatos a partir de las cabeceras de cada documento. Se diseño un patrón basado en expresiones regulares para capturar líneas con formato **clave: valor**, permitiendo así crear un diccionario de metadatos homogéneo que incluirá.

* + - Identificador del proyecto (**grant agreement**).
    - Acrónimo (**project acronym**).
    - Nombre de la organización, país y tipo.
    - Costes, financiación y otros datos económicos.
    - Datos del programa y subcall correspondiente.

De esta forma, cada documento cargado en memoria queda representado como un objeto Document de la librería LangChain, que contiene tanto el contenido textual page como un conjunto de metadatos que enriquecerán las recuperaciones posteriores.

* + 1. Segmentación del texto.

Una de las fases más importantes en un sistema de recuperación semántica es dividir los documentos largos en fragmentos más prqueños y coherentes. Esta técnica permite.

* + - Incrementar la granularidad de la búsqueda: los sistemas de embeddings funcionan mejor cuando los textos tienen una longitud limitada.
    - Reducir la probabilidad de que fragmentos irrelevantes arrastren resultados incorrectos.
    - Controlar la cantidad de contexto que se proporciona al modelo generador.

Para esta tarea se han utilizado la herramienta **RecursiveCharacterTextSplitter** de LangChain, que ofrece una estrategia flexible de segmentación indicando caracteres y delimitadores. Se definieron distintos valores de **chunk\_size** y **chunk\_overlap** para observar cómo afectan a la cobertura de contexto y la redundancia:

|  |  |
| --- | --- |
| Configuración | Descripción breve |
| chunk\_size=1000, overlap=800 | Fragmentos más largos y muy solapados, maximizando el contexto compartido. |
| chunk\_size=500, overlap=400 | Fragmentos más cortos,mayor granularidad y menor coste computacional. |
| chunk\_size=1000, overlap=0 | Fragmentos largos, sin solapamiento, serán más eficientes en memoria y procesamiento. |

Gracias a estas configuraciones, se prepararon varias versiones, cada una adaptada a los distintos experimentos realizados.

* + 1. Generación de embeddings.

Una vez segmentados los textos, el siguiente paso ha sido convertir cada fragmento en su representación numérica **embedding vector** mediante los modelos que hemos seleccionado:

* + - **Sentence-transformers/paraphrase-MiniLM-L6-v2**: modelo ligero, que no requiere preprocesamiento espacial.
    - **Infloat/e5-small-v2:** modelo más reciente, que exige anteponer prefijo en cada fragmento para que las representaciones estén alineadas semánticamente con las consultas.

Dado que los recursos han sido limitados para la ejecución del TFM no disponía de GPU, se configuraron ambos modelos para ejecutarse exclusivamente en CPU. Utilizando la normalización de los vectores, esta práctica es habitual, ya que facilita la comparación mediante distancias coseno.

Esta fase ha permitido obtener, para cada fragmento, un vector de dimensiones fijas que codifica el contenido semántico.

* + 1. Construcción del índice vectorial.

El último paso del proceso ha consistido en indexar todos los embeddings generados para permitir consultas semánticas rápidas. Para ello se empleó la librería FAISS, ampliamente utilizada por su eficiencia y por ser compatible con CPU.

Cada índice vectorial creado se guardó localmente para utilizarlo posteriormente. Se generaron distintas versiones correspondientes a cada experimento que difieren por el modelo de embeddings empelado y la configuración del splitter.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Exp | Datos | Modelo | Chunk\_size | Chunk\_overlap |
| 1 | txt | paraphrase-MiniLM-L6-v2 | 1000 | 80% |
| 2 | txt | 500 |
| 3 | txt | intfloat/e5-small-v2 | 1000 |
| 4 | txt | 1000 | 0% |

Está arquitectura flexible ha permitido comparar el efecto del modelo y la granularidad del fragmento sobre la calidad de la recuperación, el tamaño del índice y el tiempo de consulta

## Despliegue de Ollama mediante contenedor Docker.

Para el despliegue de los modelos Ollama, se ha utilizado Docker. Esta elección se ha basado en las ventajas que ofrece Docker en términos de aislamiento, portabilidad y facilidad de gestión de dependencias.

Se creo un contenedor, con el finde de obtener un puerto para que las aplicaciones locales puedan conectarse a Ollama mediante una URL, permitiendo así separar el servidor de modelos (Ollama) de las aplicaciones cliente, con el fin de facilitar el mantenimiento y la escalabilidad del sistema.

Las ventajas de utilizar Docker para desplegar Ollama aporta beneficios clave para el proyecto:

* Modularidad: El servidor de modelos queda completamente desacoplado de la lógica de recuperación y de la capa de aplicación.
* Reproducibilidad: El entorno de ejecución puede replicarse fácilmente en otras máquinas, garantizando resultados consistentes.
* Persistencia: Gracias al volumen montado, los modelos descargados no se pierden al reiniciar o actualizar el contenedor.
* Escalabilidad futura: Esta arquitectura facilita integrar otros modelos o incluso añadir balanceo de carga si es necesario.

En definitiva, el despliegue de Ollama mediante contenedor Docker completa la infraestructura del sistema RAG local desarrollado en el TFM, aportando una solución robusta, portable y fácilmente mantenible para servir modelos de lenguaje de manera eficiente.

## Evolución y optimización del pipeline de recuperación y generación.

En el desarrollo del sistema de recuperación aumentada con generación (RAG) , se inició a partir de una versión que era más o menos funcional, ya que se obtenían respuestas con resultados muy similares a los obtenidos finalmente, pero resultaban muy ineficiente en términos de tiempo de ejecución, ya que podían durar hasta 15 minutos por cada consulta, por ello este punto se va a centrar en detallar el proceso que he realizado con el fin de reducir de manera significativa los tiempos explicando las principales decisiones.

Desde el inicio del TFM, se buscaba no solo construir un código funcional, sino buscar un equilibrio entre calidad de las respuestas, tiempos de ejecución y recursos computacionales. Este objetivo global me ha obligado a repensar y optimizar distintas etapas del proceso, como la reformulación de consultas, la recuperación de documentos y el filtrado posterior. En un contexto donde la velocidad de respuesta es crítica —como señaló un profesor de la Universidad Internacional de Valencia, "Google ha evitado muchas discusiones en bares al ofrecer respuestas inmediatas"—, la agilidad del sistema se convirtió en una prioridad.

4. 2.1. Objetivo inicial.

El primer pensamiento del sistema partía de la idea que dada una pregunta origina, podría ser útil generar varias reformulaciones o variaciones semánticas para así aumentar el mayor número posible de fragmentos que puedan responder a la pregunta y no depender exclusivamente de la redacción literal de la primera pregunta.

Posteriormente, aplicar un filtrado inteligente sobre los documentos recuperados, preguntado a un modelo de lenguaje grande (LLM) si realmente estaban relacionados con la consulta inicial, para eliminar datos irrelevantes o demasiado periféricos, ya que inicialmente se realizaron consultas sobre el contexto entero y se obtuvieron respuestas totalmente irrelevantes, ya que el LLM olvidaba la pregunta tras obtener un texto demasiado largo y por ello alucinaba. Por eso finalmente, con el conjunto de fragmentos más precisos, se genera la respuesta final al usuario.

Este enfoque perseguía varios fines, aumentar la diversidad de documentos recuperados, ampliando así el contexto que el modelo LLM podía utilizar, para mejorar la precisión de la respuesta evitando ruido, manteniendo así una arquitectura modular, donde cada etapa (reformulación, recuperación, filtrado, generación) pudiera mejorarse o sustituirse de manera independiente.

* + 1. Retos detectados .

Durante la implementación inicial se detectaron los primeros problemas. El coste computacional era elevado debido a las múltiples llamadas al LLM para reformular consultas y filtrar documentos, consumiendo tiempo y recursos. Los tiempos de respuesta eran excesivos, incluso para una única pregunta, ya que el sistema tardaba varios minutos en devolver una respuesta, lo que resultaba inviable para un entorno real o para una demo interactiva. Además, la escalabilidad se veía comprometida.

Estas limitaciones evidenciaron una tensión habitual en proyectos de RAG, donde a más reformulaciones y filtrado, mayor precisión, pero peor rendimiento. A menos pasos, mayor velocidad, pero peor control de la relevancia.

* + 1. Proceso de optimización: repensando el código.

Con la reflexión comentada anteriormente, se plantearon una serie de mejoras estratégicas, más centradas en los conceptos de diseño que en cambios de código específicos.

1. Sustituir la generación manual de múltiples reformulaciones por técnicas automáticas integradas como MultiQueryRetriever.
2. Reevaluar el filtrado, de manera que se buscaba simplificar el prompt, reducir la longitud de texto que el modelo debía evaluar y, en algunos casos, asumir que la recuperación aportaba suficiente relevancia como para prescindir de filtrados muy detallados.
3. Ajustar algunos parámetros como el número de documentos devueltos por la búsqueda para encontrar un punto medio que mantenga calidad sin sobrecargar el pipeline.
4. La elección del modelo, donde se probaron modelos más pequeños con el gemma y tiny:llama, pero finalmente decidí utilizar los modelos (llama3 o phi3), que ofrecen un buen compromiso entre velocidad y calidad.
   * 1. Arquitectura.

Este proceso ha servido de manifiesto para manifestar algunas ideas clave sobre el diseño de los sistemas RAG, destacando la importancia de la etapa de recuperación ya que un modelo LLM por más potente que sea solo genera respuestas sólidas si el contexto es verdaderamente relevante.

La diversificación de consultas mediante multiquery ha demostrado ser una estratégica efectiva para superar limitaciones en la formulación inicial y abarcar mayor riqueza semántica del corpus.

El filtrado selectivo incrementa la precisión, pero requiera una calibración cuidadosa debido a su impacto directo en el rendimiento y los tiempos de procesamiento.

El modularidad del pipeline facilita la experimentación permitiendo cambios en el modelo ajuste en los prompts o incluso la sustitución de métodos de filtrado por alternativas más ligeras como sistemas basados en similitud.

## Evaluadores.

Para realizar una evaluación de la calidad de las respuestas generados por los modelos de lenguaje, se ha definido y utilizado distintos prompts que actúan como instrucciones para evaluadores automáticos. Estos evaluadores, simulados a través de otros modelos de lenguaje se han encargado de valorar la precisión y adecuación de las respuestas producidas. La evaluación se realiza en dos idiomas (inglés y castellano), con el fin de analizar posibles variaciones de rendimiento atribuibles al idioma del prompt o del contenido evaluado.

4. 4.1. Precisión de la Respuesta.

Se diseña un prompt para que el evaluador compare la respuesta generada con la respuesta esperada, sin considerar ningún contexto.

El objetivo es medir el grado de coincidencia entre las respuestas del modelo y la respuesta de referencia, de forma precisa y controlada. Las métricas que se han utilizado han sido del 1 a 5 basadas en la calidad del acierto.

4. 4.2. Precisión basada en el Contexto Recuperado.

Este prompt ha sido diseñado para permitir una evaluación más rica, ya que incorpora el contexto utilizado por el modelo para generar la respuesta.

EL objetivo es analizar no solo si la respuesta es correcta, sino también se busca justificar con base en el contexto, la información que incluye el contexto, y si la respuesta es clara y está bien formulada. Todo esto a partir de las métricas de fidelidad (correspondencia entre respuesta y contexto), cobertura (suficiencia del contexto para responder) y fluidez (claridad, coherencia y redacción), manteniendo la puntuación de 1 a 5.

4. 4.3. Objetivo del diseño multilingüe.

La duplicación de los prompts no es meramente lingüística, sino experimental. El objetivo es ver cómo puede variar la evaluación de las respuestas cuando el evaluador está influenciado por el idioma del prompt. Esto permite explorar si existen sesgos o diferencias en la severidad del juicio, dependiendo del idioma en el que se formule la tarea de evaluación.

## Resultados Evaluadores.

Una vez se definieron los prompts de evaluación descritos anteriormente, he aplicado un procedimiento de evaluación automática para algunas de las respuestas, ya que hay otras que únicamente tiene que obtener uno o varios resultados, y por ello se han obtenido dos análisis distintos, con el fin de medir la calidad de las respuestas generadas por los modelos de lenguaje, utilizando tanto evaluadores en español como en inglés. El objetivo era explotar si el idioma del prompt influye en la puntuación final, y cómo distintos parámetros del sistema afectan el rendimiento del modelo.

* + 1. Parámetros evaluados.

Se realizaron evaluaciones con diferentes configuraciones de recuperación de documentos:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Modelos de Embeddings | Tamaño de Fragmento | Solapamiento |
| 1 | paraphrase-MiniLM-L6-v2 | 1000 | 80% |
| 2 | paraphrase-MiniLM-L6-v2 | 500 | 80% |
| 3 | intfloat/e5-small-v2 | 1000 | 80% |
| 4 | intfloat/e5-small-v2 | 1000 | 0% |

A cada configuración se le aplico una evaluación en ambos idiomas (prompts en inglés y español), y se calcularon las métricas comentadas anteriormente para cada una de las configuraciones.

# Conclusión y trabajos futuros

Especifica y resalta los resultados de la investigación o del desarrollo en relación con los objetivos planteados.

Habla sobre las las dificultades que te has encontrado y las perspectivas para el futuro de este trabajo.

# Referencias

Para realizar la memoria del Trabajo Fin de Máster, se debe hacer uso de referencias bibliográficas externas con el fin de aportar mayor rigor en todas las etapas de desarrollo de trabajo. Además de incluir en el texto referencias y apoyarse de otros estudios, es **importante** la manera y formatos para citar referencias. Por un lado, las referencias deberán ser incluidas en el texto, en un formato corto mostrando año y apellido de autor, por otro lado, se debe incluir esta misma cita en el apartado de referencias.

La normativa para seguir para hacer referencias será APA.

* Trabajo de referencia: <https://www.mdpi.com/2504-2289/9/2/26>
* Wang, S., et al. (2022). *Text Embeddings by Weakly-Supervised Contrastive Pre-training*. arXiv preprint [arXiv:2212.03533](https://arxiv.org/abs/2212.03533).
* Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. arXiv preprint [arXiv:1908.10084](https://arxiv.org/abs/1908.10084).
* Ollama: <https://ollama.com/>
* HuggingFace Model Cards:
  + intfloat/e5-small-v2
  + sentence-transformers/paraphrase-MiniLM-L6-v2
* MTEB Leaderboard: <https://huggingface.co/spaces/mteb/leaderboard>
* HuggingFace model card: <https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2>
* HuggingFace Hub: <https://huggingface.co/intfloat/e5-small-v2>
* LangChain documentation: <https://docs.langchain.com>
* LangChain Embedding Wrappers: <https://docs.langchain.com/docs/integrations/text_embedding>
* FAISS: <https://github.com/facebookresearch/faiss> , <https://python.langchain.com/docs/integrations/vectorstores/>
* PCA: <https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202>
* t-SNE: <https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf>
* Johnson, 2019: <https://ieeexplore.ieee.org/document/8733051>
* Patrick Lewis, 2020 <https://arxiv.org/abs/2005.11401>

# Apéndice I

El apéndice es un adjunto al documento académico de autoría propia. No es un documento independiente, pues no se entendería si no es en relación con el resto del trabajo. Contiene información que complementa o aclara la tesis y que se considera que es demasiado larga o detallada para incluirse en el texto principal. Dicha información podría incluir gráficos o tablas, listas de datos sin procesar, etc.

# Anexos I

Los anexos también contienen información adicional que se considera relevante para justificar las conclusiones del trabajo, pero, por lo general, el autor de contenido del anexo es distinto al autor del trabajo. Suele ser un documento independiente del trabajo. Pueden ser tablas de datos, imágenes, etc. Es necesario incluir las referencias de los documentos de donde procedan.