****

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

|  |  |  |
| --- | --- | --- |
| Titulación:  Mastér 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, la inteligencia artificial (IA) ha experimentado una evolución increíble, gracias a la convergencia de factores como el aumento de la capacidad computacional, la disponibilidad masiva de datos, y la mejora en el diseño de los algoritmos. Este auge ha fomentado el desarrollo de soluciones inteligentes que se ha integrado paulatinamente en múltiples sectores, desde el comercio electrónico hasta la asistencia sanitaria, pasando por la automatización industrial o la atención al cliente. Dentro de este contexto, el procesamiento de lenguaje natural (NLP) se ha consolidado como una de las áreas de mayor interés y crecimiento, con el objetivo de 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 ala aparición de los Modelos de Lenguaje de Gran Tamaño (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 capacidad, presentan tres limitaciones principales: el conocimiento estático (limitado al momento de su entrenamiento), la ausencia de información especializada en dominios concretos, y la generación de contenidos no verídicos o imprecisos, fenómenos conocidos como “alucinaciones”.

Con el fin de paliar estas limitaciones, han surgido estrategias como el fine-tuning, la ingeniería de prompts (prompt enginering) y, especialmente, la técnica de Generación Aumentada por Recuperación (RAG), 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.

En este marco, el presente Trabajo de Fin de Máster (TFM) surge como una continuación del estudio desarrollado por Desirée Ruiz Ponce, que además ha sido una de las directoras de este TFM, 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. El trabajo previo sentó las bases para 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 pipeline de recuperación y generación. En primer lugar, se ha implementado un modelo LLM local utilizando la plataforma Ollama y el modelo llama3.2:latest, 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 asegurarnos la compatibilidad con equipos de recursos limitados, como es un portátil. 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.

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 tes 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

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

# Objetivos

En este apartado se trata de definir los objetivos del trabajo (TFM). Hay que tener en cuenta que el documento a entregar como memoria de prácticas tiene que evidenciar la relación de competencias y contenidos específicos del título con el trabajo realizado. A continuación, se explican las directrices sobre cómo expresar los objetivos formalmente en la memoria del TFM.

Los objetivos en una memoria se definen como los logros que se quiere alcanzar durante el desarrollo del trabajo. La definición de los objetivos tiene que ser clara y concisa para no dar lugar a interpretaciones. Un ejemplo es comenzar con frases como: “Los objetivos establecidos para este proyecto…” “El objetivo principal es la implantación de un clasificador…”. Los objetivos tienen que cumplir que sean específicos, realistas y alcanzables.

Es posible que a lo largo del periodo de duración del trabajo los objetivos puedan sufrir alguna modificación. Ojo! Quizá también sea necesario cambiar el título del trabajo. Deberá consensuarlo con el tutor y comunicarlo al coordinador.

En la definición de objetivos, es conveniente separar entre generales y específicos. El/los objetivos generales se podrían considerar como la/las unidades más complejas para realizar el trabajo o proyecto. Estas unidades se deberían descomponer en otros objetivos más detallados o específicos, que son en metas más sencillas para poder alcanzar el objetivo general al que hace referencia.

# 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 regleja su similaridad 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 recuperán 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 similaridad 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 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 persenta un rendimiento sobresaliente en tareas generales, pero requiere infraestructuras de alto nivel (GPUs avanzadas y servidores con mucha RAM) para su ejecución.

El sistema desarrollado en este TFM utiliza Ollama para alojar y servir un modelo lenguaje local (llama3.2:latest), una variante optimizada y comprimida para entronos más ligeros. Aunque no dispone del mismo volumen de parámetros, está versión ha sido ajustada para ofrecer un equilibrio entre rendimiento y eficiencia, y permite su ejecución en CPU o entornos de GPU moderada con la ayuda de Ollama.

|  |  |  |
| --- | --- | --- |
| Características | LLaMA 3 70B | llama3.2:latest |
| Nº de parámetros | 70 mil millones | No especificado (reducido) |
| Requiere GPU | Sí (alta gama) | No / Opcional (CPU posible) |
| Integración local | Compleja | Sencilla con Ollama |
| Latencia de inferencia | Alta | Baja / Media |
| Casos de uso | Producción a gran escala | Prototipado / local /edge |

*Tabla 3 resume las principales diferencias entre los dos modelos LLaMA*.

Ambos modelos están basados en la misma arquitectura de transformers, pero adaptados a diferentes contextos de despliegue.

# 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 significativametne 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 utilizo la biblioteca Chroma como vectorstroe, 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

Habla sobre la estructura de este apartado. No todos los apartados serán necesarios, dependerá de cada trabajo. Tampoco es necesario que los títulos sean exactamente los que aparecen abajo.

## Metodología

Al comienzo del trabajo se realiza un proceso de evaluación de las tareas que completen los objetivos propuestos. Se establece una planificación y la metodología de gestión se las tareas. Se recomienda realizar un diagrama de tareas con su evolución en cuanto al desempeño.

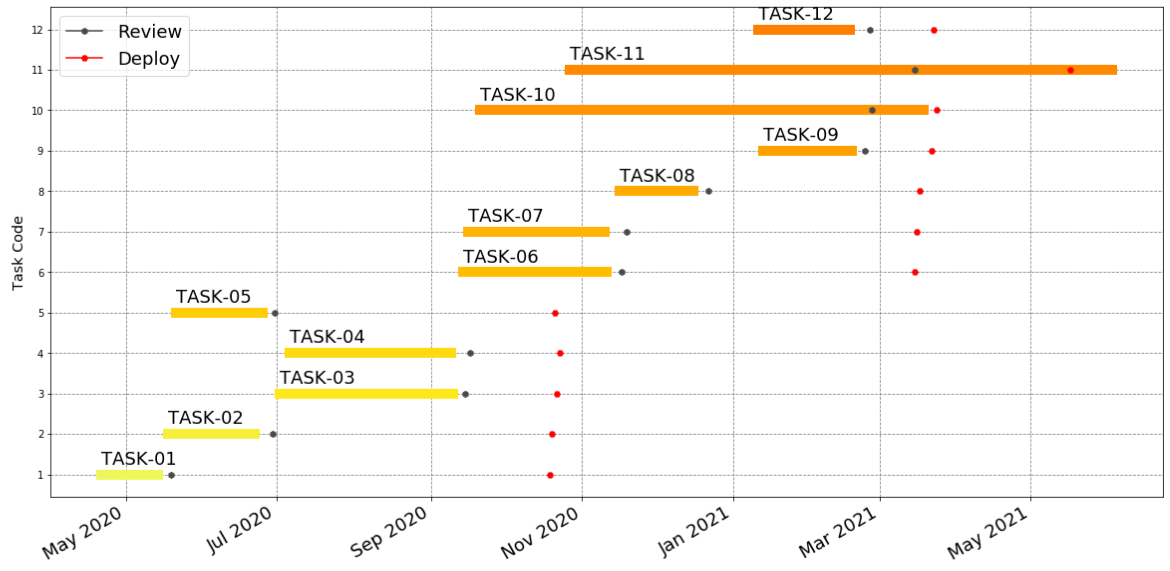


Ilustración 2. Cronograma de las tareas definidas. Elaboración propia.

## Planteamiento del problema

## Arquitectura general del sistema RAG

A continuación, se describe la construcción del sistema RAG (Retrieval-Augmented Generation) implenetado para este TFM, optimizado específicamente para su ejecución en CPU. El objetivo de esta arquitectura es permitir la recuperación semántica de documentos relacionados con los proyectos financiados por la Unión Europea bajo el programa Horizon 2020

## Resultados

# 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 a 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/>

# 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.