



**Universidad
Europea**

UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MÁSTER DE FORMACIÓN PERMANENTE EN INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE MÁSTER

**Procesamiento del BOE mediante una arquitectura
compuesta por un sistema de RAG y en un agente
basado en un modelo grande de lenguaje (LLM)**

JORGE RESINO MARTIN

Dirigido por

JORGE MORATALLA

CURSO 2023-2024

Procesamiento del BOE mediante una arquitectura compuesta por un sistema de RAG y en un agente basado en un modelo grande de lenguaje (LLM)



Jorge Resino Martin

TÍTULO: Procesamiento del BOE mediante una arquitectura compuesta por un sistema de RAG y en un agente basado en un modelo grande de lenguaje (LLM)

AUTOR: Jorge Resino Martin

TITULACIÓN: MÁSTER DE FORMACIÓN PERMANENTE EN INTELIGENCIA ARTIFICIAL

DIRECTOR/ES DEL PROYECTO: Jorge Moratalla

FECHA: septiembre de 2024

Procesamiento del BOE mediante una arquitectura compuesta por un sistema de RAG y en un agente basado en un modelo grande de lenguaje (LLM)



**Universidad
Europea**

Jorge Resino Martin

RESUMEN

El proyecto aborda la automatización del procesamiento del Boletín Oficial del Estado (BOE), utilizando una arquitectura que combina técnicas de procesamiento de lenguaje natural, un sistema de Recuperación Aumentada con Generación (RAG) y un grafo lógico formado por varios modelos de lenguaje grandes (LLMs). El problema principal es la necesidad de extraer información relevante y precisa de grandes volúmenes de datos publicados en el BOE, un desafío debido a la complejidad, lenguaje técnico y variedad de los textos. La solución desarrollada consiste en una herramienta o aplicación ETL que descarga, procesa y clasifica los documentos del BOE, complementado por un sistema de RAG que recupera fragmentos semánticos y genera respuestas a partir de los mismos.

El proyecto no ha sido realizado en colaboración con una empresa. Los principales resultados incluyen el diseño de una herramienta funcional para el procesamiento de documentos en formato PDF, la creación de un grafo de modelos LLM que facilita la generación de respuestas precisas, y la mejora en la accesibilidad de la información contenida en el BOE. Las conclusiones destacan la efectividad del sistema en la generación de respuestas relevantes y la escalabilidad de este, aunque se identificaron áreas de mejora, especialmente en la recuperación de contexto.

Palabras clave: Procesamiento de lenguaje natural o NLP, BOE, Recuperación Aumentada con Generación mejorada (CRAG y SELF-RAG), LLM, ETL, clasificación semántica de fragmentos de textos.

ABSTRACT

The project addresses the automation of processing the Official State Gazette (BOE), using an architecture that combines natural language processing techniques, an Augmented Retrieval with Generation (RAG) system, and a logical graph composed of several large language models (LLMs). The main issue is the need to extract relevant and accurate information from large volumes of data published in the BOE, a challenge due to the complexity, technical language, and variety of the texts. The developed solution consists of an ETL tool or application that downloads, processes, and classifies BOE documents, complemented by a RAG system that retrieves semantic fragments and generates responses based on them.

The project was not carried out in collaboration with a company. The main results include the design of a functional tool for processing documents in PDF format, the creation of a graph of LLM models that facilitates the generation of precise responses, and an improvement in the accessibility of the information contained in the BOE. The conclusions highlight the effectiveness of the system in generating relevant responses and its scalability, although areas for improvement were identified, especially in context retrieval.

Keywords: Natural Language Processing (NLP), BOE, Enhanced Augmented Retrieval with Generation (CRAG and SELF-RAG), LLM, ETL, semantic classification of text fragments.

AGRADECIMIENTOS

Quiero expresar mi más profundo agradecimiento a todas aquellas personas que han sido fundamentales en la culminación de este proyecto.

En primer lugar, a mi familia y amigos, por el constante apoyo emocional que me han brindado durante todo este proceso. Su comprensión y ánimo han sido esenciales para que pudiera seguir adelante en los momentos más desafiantes.

A mi tutor, Jorge, por su apoyo técnico. Sus consejos han sido de gran ayuda en el desarrollo y ejecución de este trabajo.

Finalmente, quiero agradecer a la Universidad por los conocimientos aportados a lo largo de esta formación. Las herramientas y recursos proporcionados han sido clave para completar este proyecto con éxito.

“La verdadera innovación en la inteligencia artificial no es solo crear modelos poderosos, sino diseñar agentes que colaboren entre sí, integrando herramientas y automatizando procesos para resolver problemas complejos de manera autónoma.”

Demis Hassabis, cofundador y consejero delegado de *DeepMind*.

TABLA RESUMEN

	DATOS
Nombre y apellidos:	Jorge Resino Martin
Título del proyecto:	Procesamiento del BOE mediante una arquitectura compuesta por un sistema de RAG y en un agente basado en un modelo grande de lenguaje (LLM)
Directores del proyecto:	Jorge Moratalla
El proyecto se ha realizado en colaboración de una empresa o a petición de una empresa:	NO
El proyecto ha implementado un producto: (esta entrada se puede marcar junto a la siguiente)	SI
El proyecto ha consistido en el desarrollo de una investigación o innovación: (esta entrada se puede marcar junto a la anterior)	NO
Objetivo general del proyecto:	Desarrollar un sistema eficiente y automatizado de extracción de información relevante y precisa del BOE

Índice

RESUMEN	5
ABSTRACT	6
TABLA RESUMEN	9
Capítulo 1. RESUMEN DEL PROYECTO	15
1.1 Contexto y justificación.....	15
1.2 Planteamiento del problema	15
1.3 Objetivos del proyecto.....	16
1.4 Resultados obtenidos	16
1.5 Estructura de la memoria	17
Capítulo 2. ANTECEDENTES / ESTADO DEL ARTE	19
2.1 Estado del arte	19
2.2 Contexto y justificación.....	20
2.3 Planteamiento del problema	21
Capítulo 3. OBJETIVOS	24
3.1 Objetivos generales	24
3.2 Objetivos específicos	24
3.3 Beneficios del proyecto	24
Capítulo 4. DESARROLLO DEL PROYECTO	26
4.1 Planificación del proyecto.....	26
4.2 Descripción de la solución, metodologías y herramientas empleadas.....	30
4.3 Recursos requeridos	82
4.4 Presupuesto	84
4.5 Resultados del proyecto	85
Capítulo 5. DISCUSIÓN.....	106
5.1 Resultados principales	106
5.2 Discusión sobre la metodología.....	106
5.3 Limitaciones del estudio	106
5.4 Adaptación a cambios durante el proyecto.....	107

5.5	Impacto del proyecto.....	107
Capítulo 6.	CONCLUSIONES	108
6.1	Conclusiones del trabajo.....	108
6.2	Conclusiones personales.....	108
Capítulo 7.	FUTURAS LÍNEAS DE TRABAJO	110
Capítulo 8.	REFERENCIAS.....	113
Capítulo 9.	ANEXOS	116
9.1	Anexo I. BERT: <i>Pre-training of Deep Bidirectional Transformers for Language Understanding</i>	116
9.2	Anexo II. Sentence-BERT (SBERT): <i>Embeddings de Oraciones Usando Redes Siamese BERT</i> . 117	
9.3	Anexo III. Byte Pair Encoding (BPE).....	118
9.4	Anexo IV. Algoritmo de “tokenización” WordPiece	119
9.5	Anexo V. DeBERTa: <i>Decoding-enhanced BERT with disentangled attention</i>	121
9.6	Anexo VI. RoBERTa: <i>Robustly Optimized BERT Pretraining Approach</i>	122
9.7	Anexo VII. Función GELU (<i>Gaussian Error Linear Unit</i>)	123
9.8	Anexo VIII. Configuración del modelo DeBERTa V3.....	125
9.9	Anexo IX. Archivos de configuración tipo JSON de la herramienta.	129
9.10	Anexo X. RAPTOR (<i>Recursive Abstractive Processing for Tree-Organized Retrieval</i>). 134	
9.11	Anexo XI. HNSW (<i>Hierarchical Navigable Small World</i>).	135

Índice de Figuras

Ilustración 1: Arquitectura del modelo transformer. Origen de los grandes modelos de lenguaje (LLMs).....	15
Ilustración 2: Logo del BOE.	15
Ilustración 3: Diagrama 1 de la Arquitectura y Lógica de la herramienta.....	30
Ilustración 4: Diagrama 2 de la Arquitectura y Lógica de la herramienta.....	31
Ilustración 5: Caracteres especiales localizados en un PDF del BOE.....	37
Ilustración 6: Los 50 términos más frecuentes en un corpus aplicando el método BOW-TF-IDF.	38
Ilustración 7: Los 50 términos más frecuentes en un corpus aplicando el método BOW.	39
Ilustración 8: Número de semantic similar chunks en un PDF del BOE.	43
Ilustración 9: Histograma de las medidas de similarity distance entre chunks para un PDF del BOE.	44
Ilustración 10: Diagrama de la lógica seguida por el grafo multi-agente.	60
Ilustración 11: CRAG durante la inferencia. Evaluador de recuperación para evaluar la relevancia de los documentos recuperados en relación con la entrada, y estimar un grado de confianza basado en el cual se pueden desencadenar diferentes acciones de recuperación de conocimiento, como {Correcto, Incorrecto, Ambiguo}.....	62
Ilustración 12: Funcionamiento de Self-RAG.	63
Ilustración 13: Representación del grafo lógico multi-agente creado con LangGraph, sus nodos, sus conexiones y enrutamientos condicionales.....	69
Ilustración 14: Repositorio en Hugging Face Hub donde se encuentra subido el dataset para ser recuperado y utilizado posteriormente para la evaluación del sistema RAG.	76
Ilustración 15: Métricas de evaluación de la librería RAGAS.	77
Ilustración 16: Expresión de la métrica de relevancia de la respuesta.	78
Ilustración 17: Expresiones de la métrica de precisión del contexto.....	80
Ilustración 18: Cálculo de precisión para cada chunk del contexto.	80
Ilustración 19: Cálculo de la métrica de precisión del contexto.	81
Ilustración 20: Cálculo de context recall.	82
Ilustración 21: Archivo PDF BOE-A-2024-7296.	90
Ilustración 22: Estadísticas de las métricas de la librería RAGAS.....	96

Ilustración 23: Valor de cada métrica de evaluación del sistema RAG para cada una de las 10 muestras del dataset de evaluación.	97
Ilustración 24: Diagrama de cajas para cada métrica de evaluación calculada sobre el testset.99	
Ilustración 25: Valor medio para cada métrica de evaluación calculada sobre el testset.	101
Ilustración 26: Valor de la métrica faithfulness para cada query del testset.....	102
Ilustración 27: Valor de la métrica answer relevancy para cada query del testset.	103
Ilustración 28: Valor de la métrica context precision para cada query del testset.....	104
Ilustración 29: Valor de la métrica context recall para cada query del testset.....	105
Ilustración 30: Un grafo de conocimiento generado por LLM construido utilizando GPT-4 Turbo.	110
Ilustración 31: Comparativa de las funciones de activación RELU, GELU y ELU.	125
Ilustración 32: Tree construction process.	134
Ilustración 33: Algoritmo HNSW.	135

Índice de Tablas

Tabla 1: Cronograma, descripción y esfuerzo de las actividades.

Tabla 2: Atributos de cada objeto chunk tras ser procesado

Tabla 3: Hiperparámetros para el ajuste fino (entrenamiento de un modelo pre entrenado) del modelo.

Tabla 4: Archivo de configuración del grafo: graph.json

Tabla 5: Archivo de queries o consultas: queries.json

Tabla 6: Presupuesto

Tabla 7: Archivo queries.json utilizado para la evaluación del sistema RAG.

Tabla 8: Resultados obtenidos tras la ejecución del sistema RAG: query, response, context y ground truth

Capítulo 1. RESUMEN DEL PROYECTO

1.1 Contexto y justificación

El proyecto se enmarca en el ámbito de la aplicación de técnicas avanzadas de inteligencia artificial para el análisis de documentos oficiales haciendo uso de grandes modelos de lenguaje y técnicas de procesamiento del lenguaje natural, con el objetivo de abordar la automatización del proceso de extracción de datos relevantes de, en este caso, documentación técnica de varios indoles gubernamentales. Se busca la mejora así de la accesibilidad y utilidad de la información contenida en estos documentos para diversos usuarios y aplicaciones.

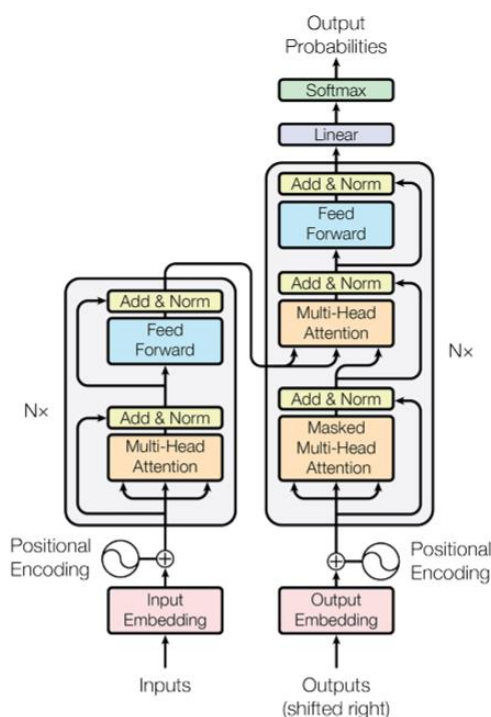


Ilustración 1: Arquitectura del modelo transformer. Origen de los grandes modelos de lenguaje (LLMs).

1.2 Planteamiento del problema

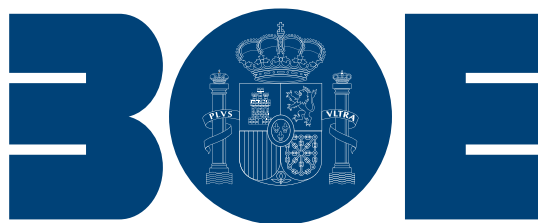


Ilustración 2: Logo del BOE.

El presente trabajo se centra en abordar un desafío significativo en el procesamiento y recuperación de información relevante y precisa de grandes volúmenes de documentos oficiales, específicamente del Boletín Oficial del Estado (BOE) de España. La creciente cantidad de datos publicados en el BOE diariamente presenta un reto considerable para los usuarios que necesitan acceder de manera rápida y efectiva a información específica dentro de este vasto corpus documental. La complejidad radica en la diversidad y volumen de los datos, lo que hace que los métodos tradicionales de búsqueda y procesamiento de información sean ineficaces y poco prácticos.

Pregunta Motriz: ¿Cómo podemos desarrollar un sistema que permita la recuperación eficiente y la generación precisa de texto a partir de grandes volúmenes de documentos oficiales, como los del BOE, utilizando modelos avanzados de lenguaje natural?

1.3 Objetivos del proyecto

- Desarrollar un sistema eficiente y automatizado para procesar y extraer información relevante y precisa de los documentos del BOE, conforme a las demandas del usuario a través de un modelo LLM.
- Crear una herramienta de ETL (Extracción, Transformación y Carga) que permita procesar y gestionar los documentos del BOE en formato PDF provenientes de la página oficial del gobierno de España.
- Diseñar un sistema de Recuperación Aumentada con Generación (RAG) que incluya un modelo *transformer-encoder* tipo BERT, entrenado y ajustado al BOE para la clasificación “multi-etiqueta” de fragmentos semánticos, mejorando así el contexto relevante para la generación de información por parte del LLM.
- Desarrollar un grafo multi nodo basado en varios modelos LLM para la generación, síntesis, obtención o evaluación de información específica según lo solicitado por el usuario, empleando el contexto proporcionado por el sistema RAG.
- Evaluar la eficiencia y precisión de la arquitectura desarrollada, tanto en la generación de texto por parte del LLM como en la recuperación de contexto por el sistema RAG, utilizando métricas adecuadas en el ámbito del procesamiento de lenguaje natural.
- Implementar mejoras para optimizar las métricas de desempeño del sistema, abordando posibles problemas como alucinaciones, respuestas irrelevantes, o contextos insuficientemente relacionados proporcionados por el sistema RAG.

1.4 Resultados obtenidos

El resultado más notable derivado del presente trabajo es la creación de una herramienta o, también se podría denominar aplicación, que utiliza la inteligencia artificial para el procesamiento de archivos de contenido no estructurado (archivos PDF) de temática técnica, legal, judicial y de más temáticas gubernamentales pertenecientes al Boletín Oficial del Estado español (BOE). Además, una herramienta totalmente funcional según muestran los resultados obtenidos, que es capaz de, mediante complejos módulos creados, hacer uso de grandes modelos de lenguaje (LLMs), ingeniería de *prompting*, grafos lógicos nodales o “multi-agente” y

técnicas de procesamiento del lenguaje natural, dar respuestas contextualizadas a las preguntas sobre el BOE que un usuario no experto pueda tener. Unas respuestas fundamentadas en el contenido procesado de fragmentos de texto de estos PDF cuyos *embeddings* son almacenados en una base de datos vectorial y, posteriormente, son recuperados, filtrados y proporcionados a los modelos LLMs como contexto para generar una respuesta precisa a la pregunta. Esto gracias a un sistema RAG mejorado creado dentro de los módulos de la herramienta y que se constituye como uno de los mayores resultados de este trabajo.

Pero, aparte de la herramienta creada que se constituye como un resultado tangible, se ha creado un *testset* para ponerla a prueba en un caso real donde un usuario quiera utilizarla para obtener información precisa de uno de estos archivos del BOE, demostrando así que su diseño y funcionalidad es adecuado. Esto se ha llevado a cabo a través de las diversas métricas de evaluación aplicadas sobre el *output* del sistema de Recuperación y Generación de Respuestas (RAG) diseñado y que se detallan en el capítulo correspondiente. Tras aplicar estas métricas sobre el *output* del sistema al *testset* creado se han analizado los valores de cada una de las métricas. Estos análisis ofrecen una visión completa del rendimiento del sistema y de toda la herramienta desarrollada para procesar textos del Boletín Oficial del Estado, destacando, donde y en qué aspectos el producto desarrollado es optimizable y donde tiene un rendimiento adecuado.

Por ejemplo y como avance al respectivo capítulo de resultados, los resultados reflejan:

- Fidelidad y Relevancia: Estas métricas son consistentemente altas, lo que indica que el sistema es capaz de generar respuestas fieles y relevantes para la mayoría de las consultas o preguntas del usuario sobre los textos técnicos y legales del BOE. Sin embargo, hay algunos casos específicos donde la fidelidad y relevancia caen, lo que sugiere que, aunque el rendimiento promedio es sólido, el sistema no es infalible.
- Problemas de Recuperación de Contexto: Las métricas de precisión del contexto y cobertura del contexto muestran más variabilidad. Esto sugiere que el sistema RAG no siempre selecciona y recupera los fragmentos más relevantes o completos del contexto, lo que puede influir en la calidad final de las respuestas.
- Oportunidades de Mejora: El enfoque principal para mejorar el rendimiento del sistema debería estar en la optimización del proceso de recuperación de documentos, tanto en precisión como en *recall*, para asegurar que la generación de respuestas esté basada en información completa y pertinente.

En resumen, la herramienta diseñada para el procesamiento del BOE demuestra un buen rendimiento general, pero existen oportunidades claras para mejorar la recuperación del contexto, es decir, mejorar los módulos de esta que realizan tareas semánticas de texto o iterar sobre otras configuraciones en las técnicas de procesamiento de lenguaje natural dado que la configuración usada actualmente es optimizable.

1.5 Estructura de la memoria

En el capítulo de antecedentes y estado del arte se presenta los fundamentos teóricos y la investigación previa relacionada con el tema. Se divide en la revisión de investigaciones y tecnologías recientes sobre el tema principal, explicación del contexto en el que se sitúa el proyecto y la necesidad de llevarlo a cabo y la descripción del problema a resolver y su importancia.

En el capítulo 3 se definen los objetivos del proyecto, tanto a nivel general como específico. Se mencionan las tareas concretas que se deben realizar para cumplir con el objetivo general y se lleva a cabo una descripción de los beneficios que se esperan obtener como resultado del proyecto.

El capítulo 4, desarrollo del proyecto, aborda el proceso de creación de producto, desarrollo de la lógica en el diseño de la herramienta, desglose de sus módulos y funcionalidades y explicaciones técnicas y de código tanto relevantes como necesarias para entender como se ha buscado resolver el problema de procesamiento del BOE. El capítulo empieza con un cronograma y una organización de las fases del proyecto para después llevar a cabo una explicación detallada de la solución técnica y las herramientas utilizadas. Por último, se incluye una descripción de los resultados obtenidos tras la evaluación en el procesamiento del BOE por parte de la herramienta desarrollada.

En el capítulo 5 se realiza una breve reflexión sobre los resultados obtenidos, comparándolos con los objetivos y discutiendo su relevancia y limitaciones.

El capítulo 6 recoge los aprendizajes del proyecto, incluyendo un resumen de las principales conclusiones técnicas y las reflexiones personales sobre el proceso y los resultados del trabajo.

El capítulo 7 está compuesto por propuestas de posibles mejoras o nuevas direcciones que se podrían explorar para continuar desarrollando o expandiendo la aplicación de inteligencia artificial desarrollada.

El capítulo 8 contiene el listado completo de las fuentes bibliográficas y recursos utilizados a lo largo del trabajo.

El capítulo 9 contiene material adicional formado por documentación técnica basada en *papers* que complementa el desarrollo del proyecto, explicaciones conceptuales necesarias para seguir y comprender la lógica de ciertos módulos diseñados, explicación de técnicas de “tokenización” usadas por los modelos de la herramienta y exposición de las arquitecturas y los métodos de entrenamiento de modelos *transformer-encoder* de tipo BERT similares o previos al usado por la herramienta.

Capítulo 2. ANTECEDENTES / ESTADO DEL ARTE

2.1 Estado del arte

El creciente volumen, complejidad y tecnicismos existentes en los documentos publicados en el Boletín Oficial del Estado (BOE) español plantea un desafío significativo para la extracción y procesamiento eficiente de información relevante, lo que exige el desarrollo de sistemas avanzados capaces de automatizar y optimizar estas tareas en contextos legales y administrativos.

En la actualidad, el procesamiento y recuperación de información a partir de grandes volúmenes de texto, especialmente en dominios específicos como el legal, ha sido un tema de gran interés en el ámbito de la inteligencia artificial y el procesamiento de lenguaje natural (NLP). El uso de técnicas avanzadas como los modelos de lenguaje de gran escala (LLM) y sistemas de Recuperación Aumentada con Generación (RAG) ha mostrado un potencial considerable para abordar estos desafíos, especialmente en tareas que requieren una alta precisión y relevancia en la información extraída y generada.

Los modelos de lenguaje de gran escala que tienen su origen en la arquitectura *transformer*, como GPT de OpenAI, BERT de Google, y más recientemente, LLAMA de Meta, han revolucionado la manera en que se aborda el procesamiento de texto en diversas aplicaciones. Estos modelos son capaces de capturar complejas relaciones contextuales y semánticas en grandes corpus de texto, lo que los hace particularmente adecuados para tareas de generación de texto, resumen y respuesta a preguntas. La evolución de estos modelos ha permitido una mejora sustancial en la capacidad de manejar grandes volúmenes de datos y generar respuestas coherentes y precisas, incluso en contextos específicos como el legal.

El sistema de Recuperación Aumentada con Generación (RAG) combina técnicas de recuperación de información con modelos de generación de texto para proporcionar respuestas más precisas y contextualmente relevantes. Este enfoque es particularmente útil en escenarios donde la información relevante está dispersa en grandes corpus de texto, como en el caso del BOE. El uso de RAG permite no solo recuperar fragmentos de texto relevantes sino también generar respuestas o resúmenes que integren esta información de manera coherente y útil para el usuario.

El procesamiento de textos legales es un área que presenta retos únicos debido a la necesidad de alta precisión y la complejidad del lenguaje utilizado. Investigaciones recientes han demostrado que los modelos de lenguaje de gran escala, combinados con técnicas de recuperación avanzada, pueden ser eficaces en la automatización de tareas legales, como la clasificación de documentos, la extracción de cláusulas específicas, y la generación de resúmenes de texto legal. Estos avances son fundamentales para aplicaciones como la que se pretende desarrollar en este TFM, donde se busca aplicar un sistema de RAG y un agente basado en LLM para el procesamiento del BOE.

Existen varias implementaciones recientes que han logrado resultados prometedores en la aplicación de estas tecnologías en contextos similares. Por ejemplo, estudios han demostrado la efectividad de modelos tipo BERT para la clasificación “multi-etiqueta” de textos legales, permitiendo una segmentación precisa y contextual de la información. Asimismo, el uso de modelos LLM en combinación con RAG ha sido explorado para mejorar la calidad de las respuestas generadas en sistemas de consulta jurídica, mostrando un alto grado de precisión en la síntesis y generación de información relevante.

A pesar de los avances, la implementación de estos modelos en contextos como el BOE presenta varios retos, incluyendo la necesidad de adaptar los modelos para manejar lenguaje técnico y legal, y la dificultad de garantizar la precisión y relevancia de la información generada. Este trabajo se propone contribuir a la literatura existente desarrollando un sistema integrado que combine estas tecnologías para mejorar significativamente la manera en que se procesa y extrae información del BOE.

2.2 Contexto y justificación

La motivación para acometer este trabajo surge de la creciente necesidad de automatizar el análisis y procesamiento de documentos oficiales complejos y voluminosos, como los que se encuentran en el Boletín Oficial del Estado (BOE) español. En el contexto actual, el BOE es una fuente fundamental de información legal y administrativa que influye en una amplia gama de decisiones y acciones, tanto en el ámbito gubernamental como en el sector privado. Sin embargo, el volumen y la diversidad de los documentos publicados diariamente dificultan el acceso rápido y preciso a la información relevante para los usuarios, quienes pueden ser desde profesionales legales hasta ciudadanos interesados en disposiciones específicas.

Este proyecto se enmarca en el ámbito de la inteligencia artificial aplicada, con un enfoque específico en el procesamiento del lenguaje natural (NLP) para la extracción y generación de información a partir de documentos técnicos y gubernamentales. La creciente demanda de sistemas que no solo recuperen información, sino que también sean capaces de entender y generar respuestas precisas en contextos específicos, resalta la relevancia de este proyecto. Se busca así mejorar la accesibilidad y la utilidad de la información contenida en los documentos del BOE, permitiendo una interacción más eficaz y automatizada con este recurso esencial.

En un entorno donde la información oficial es vital para la toma de decisiones informadas, el desarrollo de un sistema que combine la Recuperación Aumentada con Generación (RAG) y un grafo lógico basado en modelos grandes de lenguaje (LLM), representa un avance significativo. Este sistema permitirá no solo la recuperación precisa de información, sino también la capacidad de generar contenido útil en función de las necesidades del usuario, abarcando tareas como la síntesis de información, la creación de resúmenes y la generación de respuestas a consultas específicas.

Este proyecto contribuirá al campo de estudio de la inteligencia artificial aplicada y el procesamiento del lenguaje natural en varios aspectos clave:

1. Automatización del procesamiento de documentos oficiales: El desarrollo de una herramienta capaz de procesar, transformar y cargar documentos del BOE en un formato utilizable es un paso crucial hacia la automatización de tareas que tradicionalmente han sido laboriosas y propensas a errores humanos. Esta herramienta facilitará la extracción y clasificación de información relevante, mejorando la eficiencia y precisión en el manejo de grandes volúmenes de datos.
2. Innovación en la recuperación y generación de información: La implementación de una arquitectura basada en un sistema de RAG, aportará una solución innovadora para la generación de texto contextualizado y relevante. Este enfoque permitirá abordar problemas comunes en el procesamiento de textos legales, como la ambigüedad y la dispersión de la información, ofreciendo una herramienta que no solo recupera datos, sino que también proporciona un contexto adecuado para la generación de contenido preciso.
3. Mejora en la accesibilidad y utilidad de la información legal: Al facilitar la extracción y generación de información específica de manera automatizada, este proyecto contribuirá a hacer que la información contenida en el BOE sea más accesible y útil para una variedad de usuarios. Esto es particularmente relevante en contextos donde el tiempo y la precisión son críticos, como en la preparación de informes legales o la consulta sobre normativas específicas.
4. Generación de conocimiento nuevo: En el proyecto también se propone una nueva metodología dentro de una de las arquitecturas de recuperación de información para aportar contexto a los modelos LLM. Esta consiste en, tras realizar el ajuste fino de un modelo pre entrenado de tipo *transformer-encoder* para la clasificación “multi-etiqueta”, incluir este en la arquitectura RAG. Los resultados obtenidos en términos de eficiencia y precisión del sistema podrán servir de base para futuras investigaciones y aplicaciones en el procesamiento de otros tipos de documentos técnicos y gubernamentales.

Por todo ello, este proyecto no solo responde a una necesidad práctica significativa en el manejo de grandes volúmenes de información oficial, sino que también propone avanzar el estado del arte en el campo del procesamiento del lenguaje natural aplicado al derecho y la administración pública en España, ofreciendo soluciones innovadoras y eficaces para el acceso y utilización de información legal crítica.

2.3 Planteamiento del problema

El análisis del estado del arte en el campo del procesamiento de lenguaje natural (NLP) y la inteligencia artificial aplicada al análisis de documentos oficiales revela avances significativos en la capacidad de los sistemas modernos para procesar y generar información a partir de textos complejos. Sin embargo, a pesar de estos avances, se identifica una necesidad crítica en el contexto específico del Boletín Oficial del Estado (BOE) español: la falta de un sistema eficiente y automatizado que no solo sea capaz de recuperar información precisa y relevante de este

vasto corpus documental, sino que también pueda generar contenido contextualizado y útil a partir de dicha información.

El BOE, como principal fuente de información legal y administrativa en España, publica diariamente un gran volumen de documentos que abordan una amplia gama de temas técnicos, legales, y gubernamentales. Estos documentos son esenciales para la toma de decisiones informadas tanto en el ámbito público como privado. Sin embargo, la capacidad de acceder de manera rápida y precisa a la información específica dentro de estos documentos sigue siendo limitada. Los métodos tradicionales de búsqueda y recuperación de información resultan insuficientes cuando se enfrentan a la complejidad y el volumen de los textos legales, lo que genera una barrera significativa para los usuarios que necesitan acceder a información relevante de manera eficiente.

El problema se agrava cuando consideramos la necesidad de generar contenido a partir de la información recuperada, ya sea en forma de resúmenes, respuestas a consultas específicas, o síntesis de normativas. Aunque existen modelos de lenguaje de gran escala (LLM) y sistemas de Recuperación Aumentada con Generación (RAG) que han demostrado su eficacia en otros dominios, la aplicación de estas tecnologías en el contexto del BOE presenta desafíos únicos. Estos incluyen la necesidad de entrenar modelos que comprendan y manejen el lenguaje técnico y legal, así como la integración efectiva de sistemas de recuperación y generación de texto para asegurar que la información proporcionada sea precisa, relevante, y contextualizada.

En este contexto, surge la necesidad de desarrollar un sistema innovador que combine las capacidades de recuperación de información con las de generación de texto automatizado, específicamente adaptado para manejar la complejidad del corpus del BOE. Este sistema debe ser capaz de extraer información relevante y precisa de los documentos del BOE, procesarla y transformarla en contenido utilizable, todo ello dentro de un marco automatizado y eficiente. Además, es esencial que el sistema aborde los desafíos asociados con la generación de texto contextualizado, asegurando que las respuestas y resúmenes generados sean coherentes y útiles para los usuarios.

El planteamiento de este problema lleva a la definición clara de los objetivos del trabajo: desarrollar una arquitectura compuesta por un sistema de RAG que haga uso de bases de datos vectoriales para aportar un contexto adecuado y un grafo basado en modelos grandes de lenguaje o LLMs que permita no solo recuperar información de manera eficiente sino también generar contenido relevante en función de las necesidades del usuario. Este desarrollo no solo pretende llenar el vacío actual en las soluciones existentes para el procesamiento del BOE, sino que también busca avanzar el conocimiento en el campo del NLP aplicado a contextos legales y gubernamentales, proporcionando una herramienta práctica y eficaz que pueda ser utilizada tanto en la investigación como en aplicaciones concretas del mundo real.

En resumen, el planteamiento del problema en este trabajo se basa en la identificación de una necesidad insatisfecha en el campo del procesamiento de documentos oficiales, específicamente en relación con la falta de soluciones eficaces para la extracción y generación de información del BOE. Este trabajo se propone llenar ese vacío mediante el desarrollo de una

solución innovadora que combine las técnicas más avanzadas en recuperación y generación de texto, contribuyendo así tanto al avance del conocimiento en este campo como a la creación de una herramienta útil para los usuarios.

Capítulo 3. OBJETIVOS

3.1 Objetivos generales

El objetivo general del presente trabajo es desarrollar un sistema o aplicación automatizada y eficiente que permita procesar, extraer y generar información relevante y precisa a partir de los documentos del Boletín Oficial del Estado (BOE) utilizando una arquitectura modular formada por un sistema de procesamiento del lenguaje natural, un sistema de Recuperación Aumentada con Generación (RAG) y un grafo lógico basado en varios modelos grandes de lenguaje (LLMs).

3.2 Objetivos específicos

- Desarrollar una herramienta que permita la extracción, procesamiento, transformación, y carga de los documentos del BOE en formato PDF desde la página oficial del gobierno de España.
- Diseñar un sistema de Recuperación Aumentada con Generación (RAG) capaz de extraer información precisa y relevante de los textos procesados del BOE, utilizando un modelo *transformer-encoder* tipo BERT “pre-entrenado” y sobre el que se ha realizado un ajuste fino para la clasificación “multi-etiqueta” de fragmentos semánticos.
- Implementar un grafo lógico con varios modelos grandes de lenguaje que sea capaz de lidiar con un mal contexto recuperado por el sistema RAG para responder a la pregunta del usuario, lidiar con una posible alucinación en la respuesta del modelo y que sea capaz de lidiar con una respuesta errónea. Todo ello de manera iterativa hasta obtener una respuesta precisa.
- Evaluar la eficiencia y precisión de la arquitectura completa, tanto en la generación de texto por parte del LLM como en la recuperación de información y contexto por el sistema RAG, utilizando métricas adecuadas del procesamiento de lenguaje natural y librerías específicas para de este tipo de sistemas.
- Optimizar el sistema buscando la mejora de la clasificación semántica de los pedazos de texto generados mediante la medición de su semejanza vectorial utilizando varios métodos. También, mejorar las métricas de desempeño de generación de texto, abordando problemas como alucinaciones, respuestas poco relevantes, o contextos insuficientes proporcionados por el sistema RAG.

3.3 Beneficios del proyecto

El desarrollo de este proyecto ofrece varios beneficios significativos en relación con los objetivos planteados.

Mejora en la accesibilidad y precisión de la información al desarrollar una herramienta automatizada que procesa y extrae información relevante del BOE, el proyecto facilita a los usuarios acceder rápidamente a datos precisos y contextualmente relevantes, eliminando la necesidad de búsquedas manuales tediosas y mejorando la eficiencia en la obtención de información crítica.

Innovación en la recuperación y generación de información al diseñar e implementar un sistema de Recuperación Aumentada con Generación (RAG), el proyecto ofrece una solución avanzada para la generación de texto, adaptada específicamente al complejo y especializado corpus del BOE. Esto no solo mejora la capacidad de recuperación de información, sino que también permite la generación automática de respuestas, resúmenes y otros textos que se adapten a las necesidades específicas del usuario.

Optimización del proceso de toma de decisiones. La implementación de un sistema capaz de generar contenido contextualizado y relevante, basado en la información recuperada, agiliza el proceso de toma de decisiones para profesionales y ciudadanos que dependen del BOE. Esto es particularmente valioso en contextos legales, administrativos y empresariales, donde la exactitud y la rapidez en la obtención de información son cruciales.

Contribución al campo del procesamiento de lenguaje natural (NLP). Este proyecto no solo proporciona una solución práctica para el manejo de documentos oficiales, sino que también avanza el estado del arte en el uso de modelos grandes de lenguaje y sistemas de RAG en contextos legales. Los conocimientos obtenidos y las metodologías desarrolladas son altamente escalables y pueden servir de base para futuras investigaciones y aplicaciones en otros dominios que requieran procesamiento de textos complejos.

En resumen, el proyecto aporta beneficios tanto en términos prácticos, mejorando la accesibilidad y usabilidad de la información del BOE, como en términos académicos y científicos, contribuyendo al desarrollo de nuevas tecnologías en el ámbito del procesamiento de lenguaje natural.

Capítulo 4. DESARROLLO DEL PROYECTO

4.1 Planificación del proyecto

Fase/Actividad	Descripción de la Actividad	Período	Esfuerzo estimado (horas)
Fase de Investigación y Recopilación de Datos	Revisión del estado del arte y recopilación de información a partir de informes previos, documentación técnica y BOE.	Abril-Mayo	60 horas
- Estudio del Estado del Arte	Revisión bibliográfica de informes, <i>papers</i> , publicaciones y documentación relevante sobre análisis de texto, procesamiento del lenguaje, creación de aplicaciones basadas en IA y NLP y librerías necesarias para su construcción.	Abril	20 horas
- Análisis de Alternativas Actuales	Estudio de soluciones actuales de procesamiento de lenguaje natural y análisis de textos en formatos PDF.	Abril	20 horas
- Estudio de Informes Previos	Análisis de informes previos relacionados con la extracción de datos del BOE y su procesamiento.	Mayo	20 horas

Fase de Análisis Técnico	Análisis de interfaces y sistemas externos para el procesamiento del BOE, con pruebas sobre documentos seleccionados.	Mayo	40 horas
- Análisis Técnico de Interfaces	Revisión de cómo se integran los sistemas de procesamiento con otros sistemas de entrada de datos (formatos PDF, etc.).	Mayo	20 horas
- Diseño de la Validación y Pruebas	Planificación de pruebas sobre el procesamiento gramatical y la fragmentación semántica de los textos.	Mayo-Junio	20 horas
Fase de Diseño y Desarrollo	Diseño del sistema de procesamiento del lenguaje natural y su implementación.	Junio	100 horas
- Diseño del Sistema	Estructuración de las funcionalidades de cada módulo para la descarga, extracción, fragmentación semántica, aplicación de técnicas de <i>embeddings</i> , almacenamiento de los textos procesados, diseño del grafo lógico “multi-agente” que	Junio	30 horas

	genere una respuesta...		
- Desarrollo del Software	Implementación del código para realizar el procesamiento del BOE, incluyendo el desarrollo de todos los módulos de los que está formado la herramienta.	Junio-Julio	70 horas
Fase de Validación y Pruebas	Realización de pruebas del sistema con una muestra representativa de documentos del BOE y ajuste del sistema.	Julio	40 horas
- Ejecución de Pruebas	Pruebas sobre la fragmentación semántica y metadatos agregados en los documentos del BOE haciendo uso de diferentes técnicas de NLP.	Julio	20 horas
- Análisis de Resultados de las Pruebas	Ajuste del sistema basado en los resultados de las pruebas para optimizar el rendimiento.	Julio	20 horas
Fase de Revisión y Redacción	Redacción de la memoria del proyecto, incluyendo metodología, resultados y análisis.	Julio-Agosto	80 horas

Jorge Resino Martin

- Redacción de la Memoria	Documentación de los aspectos teóricos y técnicos del proyecto, así como de los resultados obtenidos.	Julio-Agosto	60 horas
- Revisión y Corrección	Revisión de la memoria, corrección de errores, ajuste del formato y preparación para entrega.	Agosto	20 horas
Fase de Presentación y Entrega Final	Preparación de la presentación final y entrega del proyecto al tribunal.	Septiembre	20 horas
- Preparación de la Presentación	Elaboración de las diapositivas, resúmenes y guion para la defensa del TFM.	Septiembre	10 horas
- Entrega Final	Presentación final y entrega de los documentos finales al tribunal evaluador.	Septiembre	10 horas

Tabla 1: Cronograma, descripción y esfuerzo de las actividades.

4.2 Descripción de la solución, metodologías y herramientas empleadas

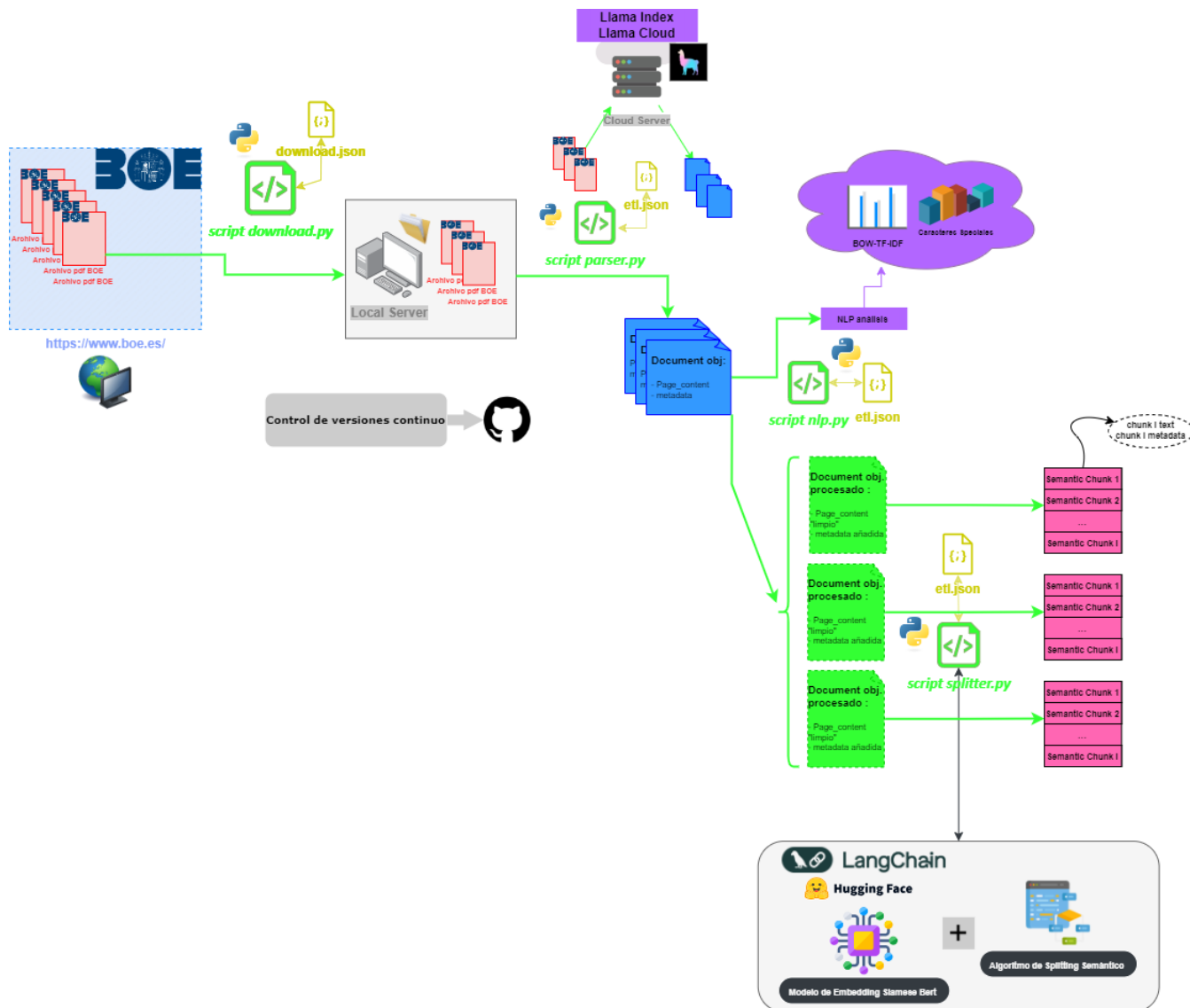


Ilustración 3: Diagrama 1 de la Arquitectura y Lógica de la herramienta.

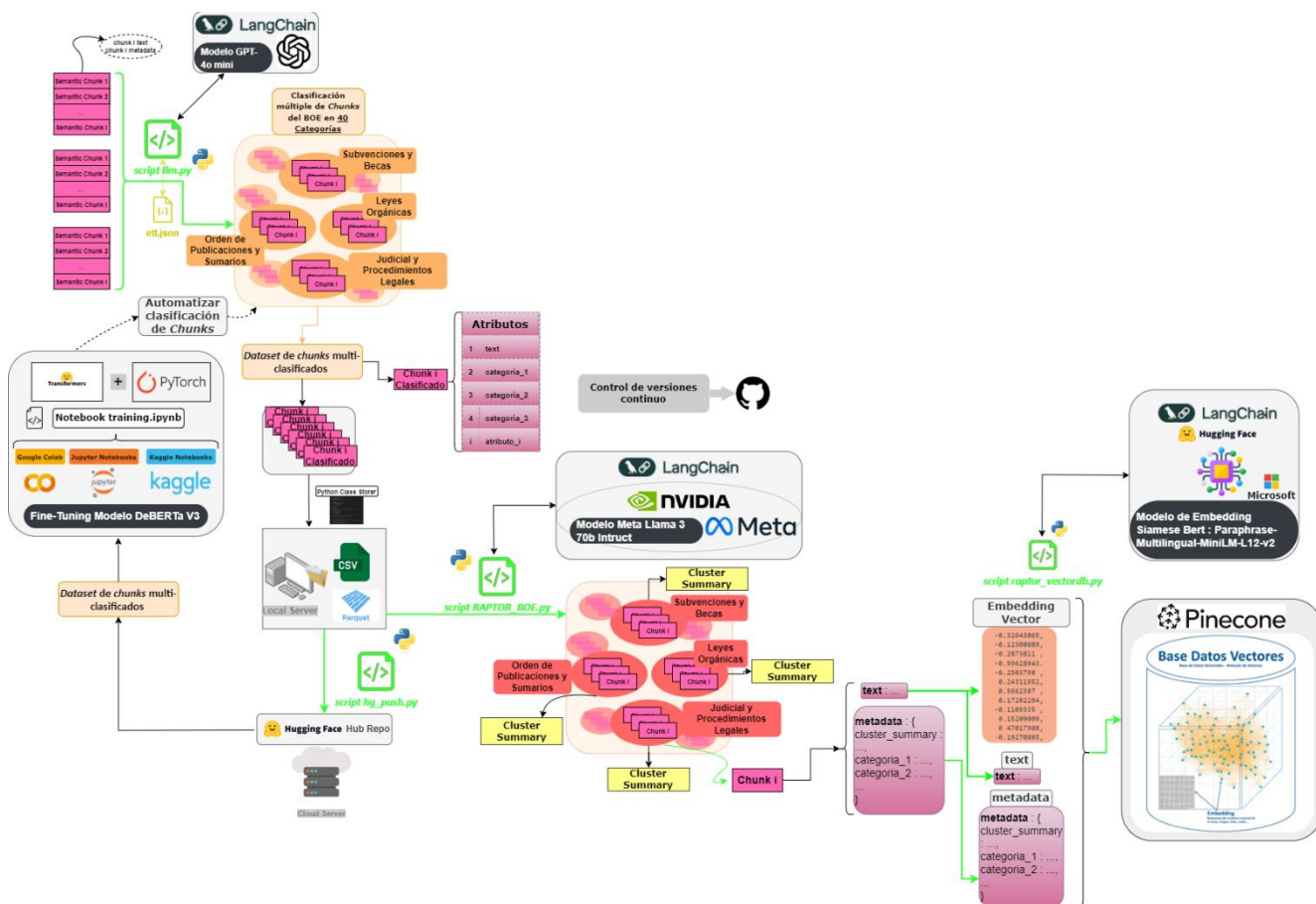


Ilustración 4: Diagrama 2 de la Arquitectura y Lógica de la herramienta.

4.2.1 Descarga, extracción, transformación y carga de los datos

En primer lugar, se ha llevado a cabo un proceso de recolección de datos que ha incluido el desarrollo del código de la herramienta encargada de la descarga o extracción, la transformación o procesamiento, y la carga o almacenamiento de los fragmentos de texto de los documentos del BOE procesados. Dentro del proyecto, esta parte se encuentra en el módulo ETL, por sus siglas en inglés *Extract, Transform, Load*.

En este módulo existen varios *scripts* de Python que se pueden dividir en dos categorías: los *scripts* dedicados a la descarga local de los archivos PDF (archivo: `download.py`) de la web oficial del BOE entre una fecha de inicio y otra fecha final (definidas externamente por el usuario) y los *scripts* dedicados al propio proceso de extracción, transformación y carga o guardado local en el formato especificado por el usuario de los fragmentos de texto de dichos documentos procesados. Estos últimos *scripts* constituyen el resto de los *scripts* dentro del módulo ETL del proyecto.

Ambas categorías de *scripts* cuentan con dos archivos de configuración de tipo JSON donde el usuario define ciertos parámetros que definen como va a llevarse a cabo el proceso de descarga,

extracción, transformación, procesamiento y guardado local de la información del BOE. Estos archivos y su contenido se hayan en los anexos del presente trabajo.

Proceso de descarga

El *script download.py* de Python está diseñado para automatizar el proceso de descarga de documentos PDF del Boletín Oficial del Estado (BOE) desde su sitio web oficial. Utilizando fechas de inicio y fin especificadas por el usuario, el *script* descarga los archivos XML que contienen resúmenes diarios y luego extrae las URLs de los PDFs correspondientes. Estos PDFs se descargan y se almacenan localmente en un sistema de archivos organizado por fecha.

El *script* se estructura en torno a una clase principal llamada *Downloader*, que contiene varios métodos para realizar las siguientes tareas:

1. Descarga de archivos XML: Se obtiene el archivo XML diario desde la API del BOE, el cual contiene los enlaces a los documentos PDF publicados en esa fecha.
2. Extracción de URLs de los PDFs: A partir del archivo XML descargado, el *script* extrae las URLs de los documentos PDF.
3. Descarga de PDFs: Los PDFs se descargan en lotes (especificados por el parámetro *batch* en el archivo de configuración que defina el usuario) y se almacenan localmente en una estructura de carpetas organizada por año, mes y día.
4. Guardado de archivos: Tanto los archivos XML como los PDFs se guardan en ubicaciones específicas dentro del sistema de archivos local.

La finalidad de este *script* dentro del proyecto es automatizar la fase de extracción y almacenamiento de los documentos del BOE, que es un componente crítico en el proceso de ETL (*Extract, Transform, Load*). Este proceso asegura que los documentos relevantes sean descargados de manera eficiente y almacenados adecuadamente para su posterior procesamiento, transformación y análisis en las siguientes etapas del proyecto. En particular, la capacidad de organizar y acceder a estos documentos de forma estructurada es esencial para las fases subsiguientes de análisis de datos y generación de información, permitiendo una integración fluida en el sistema general de Recuperación Aumentada con Generación (RAG) y el modelo LLM.

Proceso de extracción, transformación, procesamiento y carga

El resto de los *scripts* a su vez se subdividen por funcionalidad en el proceso de extracción, transformación, procesamiento y almacenamiento. En cada uno de ellos se definen una serie de clases de Python que utilizan una librería u otra de este lenguaje en sus métodos para extraer y transformar la información de los textos del BOE que han sido almacenados local y previamente por fecha.

Script parser.py

El script que primero se desarrolló fue *parser.py*, el cual, está diseñado para automatizar la tarea de *parsing* (análisis y extracción) de documentos del directorio donde se encuentran los PDFs descargados localmente. El propósito principal del *script* es procesar estos archivos, como PDFs,

dentro de un directorio especificado, utilizando una herramienta llamada *LlamaParse*, que es parte de la biblioteca *LlamaIndex*.

LlamaParse es un servicio de análisis de documentos desarrollado por *LlamaIndex*, diseñado para transformar documentos complejos, como archivos PDF, presentaciones de PowerPoint, documentos de Word y hojas de cálculo, en datos estructurados. Es una herramienta integral que se puede utilizar a través de una API REST, como un paquete de Python, un SDK de *TypeScript*, o mediante una interfaz web. Actualmente, *LlamaParse* se encuentra en fase de beta pública, lo que significa que está disponible para que los usuarios lo prueben con ciertas limitaciones de uso, como un máximo de 1,000 páginas por día y 7,000 páginas gratuitas por semana (*LlamaIndex*, s.f.).

El principal objetivo de *LlamaParse* es mejorar la forma en que los modelos de lenguaje (LLMs) interactúan con los datos, asegurando que estos estén en un formato fácilmente comprensible para los modelos. Según *LlamaIndex*, la calidad del análisis de documentos influye directamente en la efectividad de las aplicaciones de inteligencia artificial generativa. *LlamaParse* es especialmente efectivo en la conversión de documentos PDF con tablas complejas en un formato de *markdown* bien estructurado, lo que facilita su posterior uso en análisis y procesamiento por parte de LLMs. *LlamaParse* está disponible como un servicio independiente, con integración en la API de ingestión y recuperación gestionada por *LlamaIndex*. Aunque actualmente se enfoca principalmente en PDFs con tablas, *LlamaParse* también está ampliando su compatibilidad para incluir figuras y otros tipos populares de documentos, como .docx, .pptx, y .html, constituyéndose como la herramienta de extracción mejor considerada hoy en día y el estado del arte en este aspecto (*The AI Forum*, 2023).

Este *script* convierte los documentos procesados a un formato compatible con la librería *LangChain* para su posterior uso en tareas de procesamiento de lenguaje natural (NLP) o inteligencia artificial. El *script* importa varias bibliotecas necesarias para su funcionamiento, como *os*, *Document* de *langchain.schema*, *LlamaParse*, y *SimpleDirectoryReader* de *llama_index.core*. Después hace uso de la clase *SimpleDirectoryReader*, que es un lector de directorios. Este objeto se instancia y se encargará de recorrer el directorio especificado, identificar los archivos con la extensión especificada por el usuario en el archivo JSON (por ejemplo, .pdf), y procesarlos utilizando el *parser* configurado (*LlamaParse*). Esta última tarea de extracción se realiza mediante el método *load_data()* que utiliza el lector de directorios para cargar y procesar los documentos. Este método devuelve una lista de objetos de documento procesados por *LlamaParse*, después los documentos procesados son convertidos a un formato compatible con *LangChain* (*to_langchain_format()*), lo que permite que estos documentos sean utilizados en otras tareas de procesamiento dentro de aplicaciones que utilizan *LangChain*.

Por tanto, este *script* es una herramienta fundamental dentro de un flujo de trabajo que involucra el procesamiento de grandes cantidades de documentos. Su finalidad es convertir automáticamente los archivos en un directorio en objetos estructurados y procesables que puedan ser utilizados en tareas avanzadas de NLP, como análisis de contenido, generación de resúmenes, extracción de datos, entre otros. Este proceso es especialmente útil cuando se trabaja con documentos en formatos no estructurados, como es el caso de uso del BOE que son

PDFs con una gran cantidad de tablas, que necesitan ser transformadas en datos manejables y analizables por sistemas y modelos de inteligencia artificial.

Script *nlp.py*

Una vez “parseados” los PDFs del BOE y creado un objeto *Document* de la librería *LangChain* por cada uno de ellos, el pipeline lee el archivo de configuración y lleva a cabo el procesamiento de texto según los métodos que allí haya definido el usuario.

Los diferentes métodos de NLP se han programado en el script *nlp.py*. Este script está diseñado para realizar un preprocesamiento exhaustivo de textos, y está especialmente pensado para la limpieza y procesado de documentos del Boletín Oficial del Estado (BOE). Se compone de dos clases principales (*TextPreprocess* y *BoeProcessor*), varias funciones de utilidad y configuraciones que permiten la manipulación y limpieza de grandes volúmenes de texto, haciendo que los documentos estén listos y limpios para su análisis o uso posterior en modelos de procesamiento de lenguaje natural (NLP).

La clase *TextPreprocess* se encarga del preprocesamiento general del texto, eliminando elementos no deseados y transformando el contenido textual en un formato más manejable para el análisis posterior.

Las variables de esta clase son:

- `SPC_CHARACTERS`: Lista de caracteres especiales que se pueden eliminar durante el procesamiento del texto.
- `PATRON_EMOJI`: Expresión regular utilizada para identificar y eliminar emojis del texto.
- `PATRON_CH_JAP`: Expresión regular para identificar y eliminar caracteres chinos y japoneses.

Los atributos de esta clase son:

- `task`: Describe la tarea de preprocesamiento que se está realizando.
- `docs`: Lista de documentos a procesar.
- `spc_characters`: Lista personalizada de caracteres especiales que el usuario desea eliminar (opcional).
- `spc_words`: Lista de palabras específicas que se desean eliminar (opcional).
- `data`: *DataFrame* de pandas que contiene datos adicionales para el procesamiento (opcional).

Los métodos de esta clase que se han diseñado para el procesado general de cualquier texto son:

- `__post_init__`: Inicializa el corpus de texto y los metadatos de los documentos.
- `del_stopwords`: Elimina las palabras vacías (*stopwords*) del texto basado en el idioma especificado.
- `del_urls`: Elimina URLs del texto.
- `del_html`: Elimina etiquetas HTML del texto.
- `del_emojis`: Elimina emojis del texto usando `PATRON_EMOJI`.

- `del_special`: Elimina caracteres especiales del texto.
- `del_special_words`: Elimina palabras específicas del texto que coincidan con `spc_words`.
- `del_digits`: Elimina los dígitos del texto.
- `del_chinese_japanese`: Elimina caracteres chinos y japoneses usando `PATRON_CH_JAP`.
- `del_extra_spaces`: Elimina espacios adicionales en el texto.
- `get_lower`: Convierte todo el texto a minúsculas.
- `get_alphanumeric`: Elimina caracteres no alfanuméricos del texto.
- `stem`: Realiza *stemming* de las palabras. El *stemming* es un proceso en el procesamiento de lenguaje natural (NLP) que consiste en reducir las palabras a su raíz o forma base, eliminando los sufijos derivados. El objetivo es simplificar las palabras a una forma común, permitiendo que diferentes variantes de una palabra se traten como la misma en los análisis posteriores. Por ejemplo, las palabras "jugar", "jugando" y "jugador" pueden ser reducidas a la raíz "jug". A diferencia de la lematización, que busca obtener la forma canónica de una palabra, el *stemming* es una técnica más agresiva y menos precisa, que simplemente recorta los sufijos comunes (Manning, 2008).
- `lemmatize`: Lematiza las palabras, convirtiéndolas a su forma base. La lematización es un proceso en el procesamiento de lenguaje natural (NLP) que consiste en reducir las palabras a su forma canónica o lema. A diferencia del *stemming*, que simplemente corta los sufijos para obtener una raíz, la lematización considera la estructura gramatical y el contexto de la palabra para convertirla en su forma base, como aparece en un diccionario. Por ejemplo, las palabras "corriendo", "corrió" y "correrá" se reducen a su lema "correr". La lematización es más precisa que el *stemming* y es útil en aplicaciones donde la exactitud semántica es crucial (Jurafsky, 2009).
- `custom_del`: Elimina caracteres específicos del texto o del *DataFrame* proporcionado y puede generar gráficos de la frecuencia de esos caracteres.
- `bow`: Convierte el texto en un modelo de *Bag of Words* (BoW), creando una matriz de frecuencias de palabras. *Bag of Words* (BoW) es un enfoque fundamental en el procesamiento de lenguaje natural (NLP) para representar texto. Consiste en descomponer un texto en sus componentes básicos, es decir, las palabras individuales, sin tener en cuenta el orden en el que aparecen. En este modelo, un documento o texto se representa como una colección (o "bolsa") de palabras, donde la frecuencia de aparición de cada palabra es importante, pero no su posición en la oración. El objetivo es convertir el texto en un formato numérico que pueda ser utilizado por algoritmos de aprendizaje automático. A pesar de su simplicidad, BoW es ampliamente utilizado en tareas como la clasificación de texto y el análisis de sentimientos (Harris, 1954).
- `bow_tf_idf`: Convierte el texto en una representación *TF-IDF* (*Term Frequency-Inverse Document Frequency*), que es una medida de relevancia. Esta es una medida utilizada en el procesamiento de lenguaje natural (NLP) para evaluar la importancia de una palabra en un documento dentro de un conjunto de documentos o corpus. Esta medida combina dos conceptos:

1. *Term Frequency* (TF): Frecuencia de un término en un documento específico. Se calcula como el número de veces que aparece una palabra en un documento dividido por el total de palabras en ese documento.
2. *Inverse Document Frequency* (IDF): Mide la importancia de una palabra en todo el corpus. Se calcula como el logaritmo del número total de documentos dividido por el número de documentos que contienen la palabra. Esta medida disminuye la relevancia de palabras comunes que aparecen en muchos documentos y destaca las palabras que son significativas para un pequeño número de documentos.

TF-IDF es útil para identificar palabras que son importantes en un documento pero que no son comunes en el resto del corpus, lo que lo convierte en una herramienta eficaz para tareas como la recuperación de información, la búsqueda de textos y la clasificación de documentos (Salton, 1983).

Por su parte la clase *BoeProcessor* hereda los métodos y atributos de *TextPreprocess* pero está específicamente diseñada para procesar documentos del BOE. Además de las funcionalidades de limpieza, también agrega y manipula metadatos específicos para cada documento. Es más específica del BOE porque, tras estudiar los textos, incluye patrones y expresiones regulares para borrar, localizar o almacenarlos en los metadatos de cada fragmento de texto.

Sus métodos incluyen:

- `invoke`: Procesa los documentos del BOE, limpia su contenido y actualiza los metadatos. Devuelve una lista de documentos procesados.
- `reconstruct_docs`: Reconstruye los documentos después de haber sido procesados, manteniendo sus metadatos.
- `_get_id`: Genera un identificador único (UUID) para cada documento.
- `_clean_doc`: Limpia el documento eliminando patrones específicos y extrae títulos.
- `_extract_titles`: Extrae títulos del texto del documento usando patrones predefinidos.
- `_clean_titles`: Limpia los títulos eliminando patrones específicos.
- `get_del_patrones`: Limpia el documento eliminando patrones específicos y actualiza los metadatos.
- `_get_date_creation_doc`: Extrae y formatea la fecha de publicación del documento a partir de su ruta de archivo.
- `_put_metadata`: Agrega o actualiza metadatos en un documento.

Este script está diseñado para preparar grandes volúmenes de texto, específicamente documentos del BOE, para análisis posteriores. Permite limpiar el texto eliminando elementos no deseados, estandarizarlo (por ejemplo, convertirlo a minúsculas o eliminar caracteres no alfanuméricos) y estructurarlo en formatos útiles como *Bag of Words* o *TF-IDF*. Además, la clase *BoeProcessor* agrega la funcionalidad específica para gestionar y manipular metadatos asociados a los documentos del BOE, lo que es esencial para su correcta indexación y recuperación en aplicaciones posteriores de NLP.



37

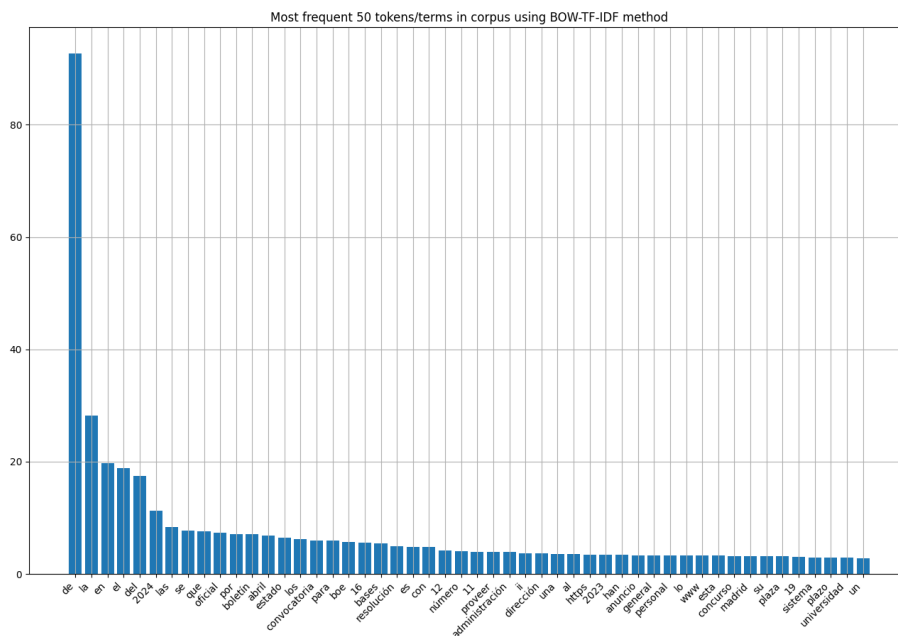


Ilustración 6: Los 50 términos más frecuentes en un corpus aplicando el método BOW-TF-IDF.

Pero el script, como se ha mencionado, también es capaz de realizar y mostrar los resultados de análisis de los textos más complejos como BOW-TF-IDF o BOW.

La anterior ilustración muestra los 50 términos más frecuentes en un corpus (en este caso, un único PDF del BOE) analizado mediante el método BOW-TF-IDF (*Bag of Words - Term Frequency-Inverse Document Frequency*). En este caso, los resultados revelan algunos patrones destacables.

En primer lugar, para este PDF, destaca la alta frecuencia de palabras comunes. Los términos "de", "la", "en", "el", y "del" son los más frecuentes en el corpus. Estos son artículos y preposiciones comunes en español, lo cual es esperado porque estas palabras son estructurales en la mayoría de los textos en español. Sin embargo, su alta frecuencia también indica que, sin una estrategia de filtrado adecuada, estas palabras pueden dominar el análisis de texto, lo que podría enmascarar términos más significativos.

En segundo lugar, existen términos relevantes del contexto. Son otros términos con frecuencias notables como "oficial", "boletín", "resolución", "convocatoria", "administración", y "dirección" sugieren que el corpus podría estar relacionado con documentos oficiales, publicaciones gubernamentales, o textos legales. Lo que es esperable tratándose de un texto del BOE. Estos términos son más específicos y relevantes para el contenido del corpus, por lo que podrían ser clave para la interpretación y el análisis de la temática del archivo, incluso para clasificar este archivo en alguna categoría.

La presencia predominante de palabras funcionales o *stop words* sugiere que una estrategia de filtrado o eliminación de estas palabras podría ser útil para obtener una visión más clara de los

términos clave en el corpus. En muchos casos, al aplicar TF-IDF, se eliminan estas palabras para centrarse en términos más representativos del contenido específico del corpus.

En lo que a frecuencia se refiere, la gráfica muestra una distribución de frecuencia que sigue una tendencia de decrecimiento rápido, donde unos pocos términos tienen una frecuencia muy alta y la mayoría tienen frecuencias mucho más bajas. Esta es una característica típica de la ley de Zipf, que describe cómo en un corpus de texto, un pequeño número de palabras aparecen muy frecuentemente mientras que la mayoría aparecen raramente. En concreto y en el ámbito del machine *learning* y, para ser más específicos, el NLP o *Natural Language Processing*, la Ley de Zipf español es una distribución de probabilidad discreta que nos indica la probabilidad de encontrar una palabra en un corpus dado (KeepCoding., 2023).

En resumen, la gráfica generada a partir del método BOW-TF-IDF muestra tanto las palabras más comunes en el corpus, que tienden a ser términos funcionales de alta frecuencia, como palabras más específicas que parecen ser más relevantes para el contenido del texto, lo que podría indicar la temática general del corpus analizado. Esta gráfica, por tanto, sugiere que para este PDF del BOE una mejor comprensión implicaría aplicar técnicas adicionales de preprocesamiento, como la eliminación de *stop words*.

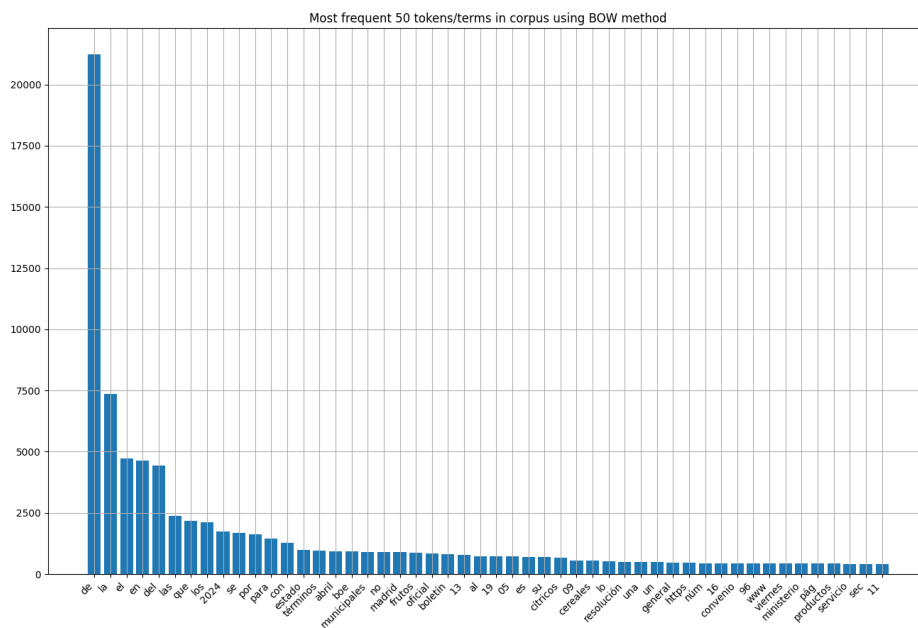


Ilustración 7: Los 50 términos más frecuentes en un corpus aplicando el método BOW.

En el caso de esta ilustración, también se muestran los 50 términos más frecuentes en un corpus, pero utilizando el método BOW (*Bag of Words*). Comparando esta imagen con la anterior, que utilizaba el método BOW-TF-IDF (*Term Frequency-Inverse Document Frequency*), podemos observar diferencias significativas en cómo se presentan las palabras.

El método BOW simplemente cuenta la aparición de cada término en el corpus, lo que resulta en que palabras muy comunes, como artículos y preposiciones ("de", "la", "en", etc.), dominen el gráfico. Estas palabras tienen frecuencias muy altas debido a su uso frecuente en el idioma, pero no necesariamente aportan mucho valor semántico en el análisis.

A diferencia del TF-IDF, BOW no pondera las palabras según su importancia relativa dentro del corpus. Esto significa que no distingue entre palabras que son comunes en todos los documentos y aquellas que son específicas y por tanto más informativas en ciertos documentos.

En la imagen anterior, usando TF-IDF, las palabras comunes como "de", "la", "en" seguían apareciendo, pero su impacto era reducido debido a la penalización que el IDF (*Inverse Document Frequency*) impone sobre las palabras que aparecen en muchos documentos. Esto permite que palabras más relevantes y específicas del corpus emerjan, proporcionando un análisis más informativo.

Mientras que BOW destaca términos frecuentes, TF-IDF resalta términos que son más significativos para los documentos específicos dentro del corpus, lo que es crucial para tareas como la clasificación de texto o la recuperación de información, donde se necesita identificar términos que realmente diferencian un documento de otro.

El método BOW proporciona una visión clara de las palabras más frecuentemente utilizadas en un corpus, pero no tiene en cuenta su relevancia o unicidad dentro del conjunto de documentos. En cambio, TF-IDF ajusta estas frecuencias para destacar términos más útiles para el análisis semántico. Por lo tanto, aunque BOW es útil para identificar términos comunes, TF-IDF es generalmente preferido cuando se necesita una comprensión más profunda del contenido del texto. Por ello en caso de usar algún método para el BOE sería apropiado BOW con TF-IDF.

Script splitter.py

Este script de Python es uno de los más importantes del presente trabajo dado que se encarga de dividir cada texto "limpiado" o procesado previamente de cada PDF del BOE en fragmentos de texto más pequeños o *chunks* que son los que luego aportarán el contexto adecuado al grafo multi nodo y a sus LLMs para generar texto y poder responder adecuadamente, sin alucinaciones, a las peticiones del usuario.

El *script* está diseñado para dividir el texto previamente procesado y almacenado en el atributo *text* de los objetos *Document* (existe uno por PDF en este punto de la ejecución de la ETL) en fragmentos más pequeños utilizando diferentes estrategias de *splitting* o división de texto. Esto es especialmente útil en tareas de procesamiento de lenguaje natural (NLP) donde se necesita dividir documentos largos en partes manejables para análisis posteriores, como la generación de *embeddings* o la clasificación de texto por parte de LLMs (como es el caso del presente trabajo). El *script* contiene dos clases principales: *CustomSemanticSplitter* y *Splitter*, cada una con métodos específicos para realizar esta tarea.

CustomSemanticSplitter se centra en dividir el texto de un mismo PDF del BOE en fragmentos basándose en la similitud semántica entre las oraciones. Utiliza un modelo de *embeddings* para generar representaciones vectoriales densas de las oraciones y luego calcula la distancia

geométrica entre pares de vectores, definida por el coseno que forman ambos, para determinar los puntos de corte adecuados para los fragmentos. Y todo en un proceso iterativo. De esta manera las oraciones más semejantes a nivel semántico se fusionan en una sola, siempre y cuando no excedan el tamaño límite establecido como “hiperparámetro” en el archivo de configuración. El modelo utilizado para generar los *embeddings* de las oraciones es crítico en esta situación. Los modelos que se encargan de generar estas representaciones vectoriales densas o *embeddings* de las oraciones son los modelos de tipo Siamese BERT que cuentan con una arquitectura similar a la de un *transformer* (Vaswani, 2017) pero solo la parte del *encoder* y con ciertas peculiaridades. Para una mayor comprensión de este tipo de modelos el lector puede acudir a los anexos I y II del presente trabajo.

Por otro lado, este algoritmo de *splitting* es una adaptación desarrollada a partir del código desarrollado por (Kamradt, 2024) en su video teórico-práctico sobre *semantic chunking*.

Los atributos de esta clase son:

- `MIN_INITIAL_LONG_CHUNKS`: Es una constante que define el número mínimo de fragmentos largos que deben existir después de la primera división del texto.
- `buffer_size`, `tokenizer`, `max_tokens`, `max_big_chunks`, `embedding_model`, `verbose`, `threshold`, `namespace_id`, `storage_path`, `min_initial_chunk_len`: Son parámetros que controlan el comportamiento de la clase, como el tamaño máximo de tokens por fragmento, el umbral de similitud, y la configuración del modelo de *embeddings*.

Los métodos de esta clase son:

- `__init__`: Inicializa la clase con los parámetros especificados por el usuario.
- `_prepare_texts`: Toma un documento y lo divide en oraciones utilizando patrones de división específicos (como puntos, saltos de línea, etc.). El objetivo es obtener fragmentos iniciales del documento que luego se combinarán si es necesario.
- `_get_id`: Genera un identificador único para un fragmento de texto utilizando UUID.
- `_combine_sentences`: Combina oraciones adyacentes en fragmentos más largos para asegurarse de que no se exceda un número máximo de tokens. Si se excede este límite, se reduce el tamaño del "buffer" y se vuelve a intentar.
- `_get_embeddings`: Genera *embeddings* para los fragmentos de texto utilizando el modelo de *embeddings* especificado.
- `_get_similarity`: Calcula la similitud entre los *embeddings* de oraciones consecutivas utilizando la similitud coseno.
- `_get_chunks`: Divide el documento en fragmentos basándose en las similitudes calculadas y en un umbral determinado.
- `_plot_similarity`: Genera gráficos de las similitudes entre oraciones para visualizar cómo se dividen los fragmentos. Los gráficos se guardan en una ubicación especificada.
- `_get_tokens`: Cuenta el número de tokens en un fragmento de texto utilizando el tokenizador especificado.
- `_create_docs`: Convierte los fragmentos generados en objetos de tipo *Document* de nuevo que incluyen el contenido del fragmento y los metadatos procedentes del

Document “padre” de este fragmento o *chunk* que son los metadatos del PDF del que proviene.

- `split_documents`: Método principal de la clase. Toma una lista de documentos y los divide en fragmentos utilizando los métodos anteriores.

La clase *Splitter* actúa como una interfaz que permite al usuario elegir entre diferentes modos de división de texto: *RECURSIVE*, *SEMANTIC* y *CUSTOM*. Cada modo utiliza una estrategia diferente para dividir los documentos. La estrategia *CUSTOM* es la definida y creada anteriormente y las demás son clases importadas de la librería *LangChain*.

Los atributos de esta clase *Splitter* son:

- `chunk_size`, `storage_path`, `embedding_model`, `tokenizer_model`, `threshold`, `max_tokens`, `buffer_size`, `verbose`, `max_big_chunks`, `splitter_mode`, `min_initial_chunk_len`: Parámetros que configuran el comportamiento del splitter seleccionado.

Sus métodos incluyen:

- `__init__`: Inicializa la clase con los parámetros específicos y selecciona el modo de splitter en función del parámetro `splitter_mode`.
- `_init_splitter`: Devuelve el *splitter* correspondiente basado en el modo seleccionado (*RECURSIVE*, *SEMANTIC*, *CUSTOM*).
- `invoke`: Método principal de la clase que toma una lista de documentos y aplica el *splitter* seleccionado para dividirlos en fragmentos.

CustomSemanticSplitter se utiliza cuando se necesita una división precisa basada en la similitud semántica de las oraciones. Es útil para dividir documentos donde la estructura semántica es importante.

Splitter es una clase más general que permite seleccionar diferentes modos de división. Facilita el uso de diferentes técnicas de *splitting* sin tener que preocuparse por los detalles de implementación de cada una.

Como conclusión, este script es una herramienta poderosa para dividir documentos de texto en fragmentos manejables, con opciones para ajustar el proceso de división según las necesidades específicas del usuario. Las clases proporcionadas permiten una gran flexibilidad en la forma en que los documentos se dividen, desde métodos más simples basados en reglas hasta métodos más complejos basados en la semántica de las oraciones.

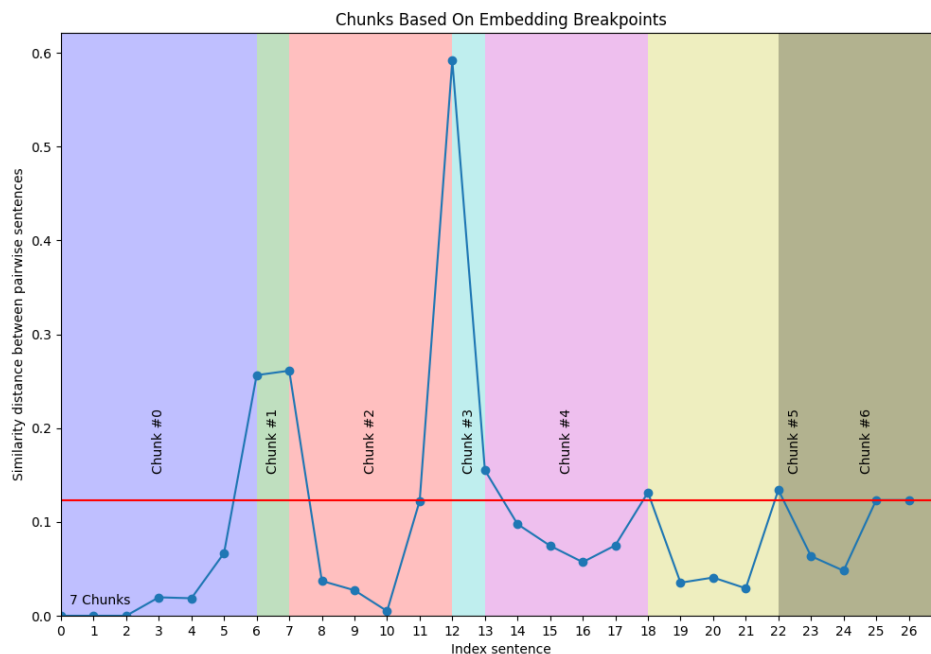


Ilustración 8: Número de semantic similar chunks en un PDF del BOE.

La anterior ilustración muestra la gráfica con el resultado del algoritmo de división para un PDF del BOE, en este caso los diferentes parámetros definidos en el archivo de configuración *etl.json* dentro de la clave *splitter* y , destacando el umbral de *similarity distance* y la dimensión de los *chunks*, han generado 7 fragmentos o *chunks*.

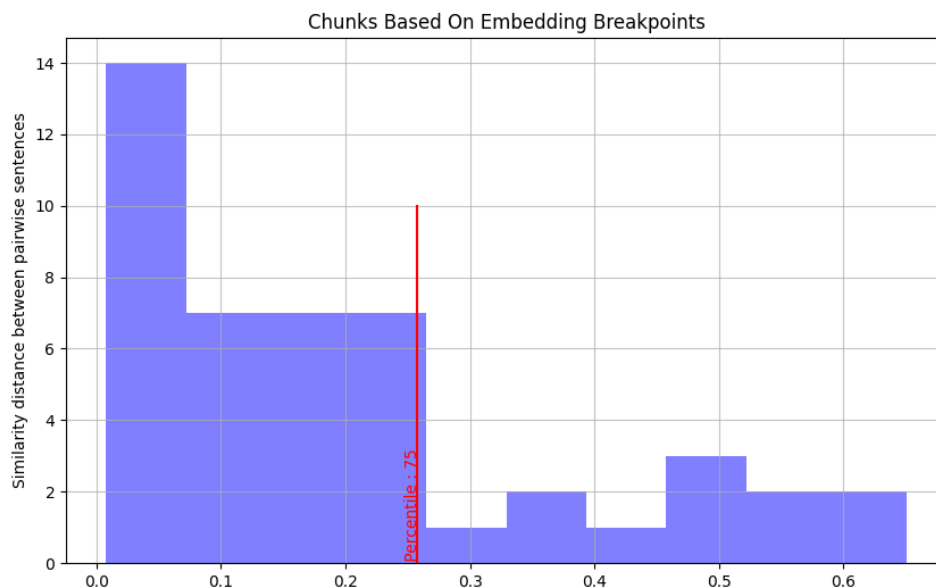


Ilustración 9: Histograma de las medidas de similarity distance entre chunks para un PDF del BOE.

Esta ilustración muestra, para el mismo PDF de la ilustración anterior, el histograma de los *similarity distance* entre *chunks*. Se dibuja el umbral establecido para generar un *chunk*. Este umbral es el percentil 75, es decir, si la *similarity distance* entre dos *chunks* está más allá de este percentil, estos *chunks* no se fusionarán en uno solo. Si es al revés y su similaridad está dentro del umbral del histograma entonces se fusionarán en un único *chunk*.

Script *llm.py*

Tras obtener los fragmentos de texto procesados o *chunks* y metadatos asociados de cada uno de los PDFs del BOE, se diseña un script de Python para clasificarlos en diferentes categorías legales, jurídicas y otras categorías tratadas en el BOE utilizando modelos de lenguaje grandes (LLMs) como GPT-3.5, LLAMA, y otros. Este proceso de clasificación utilizando LLMs tiene como finalidad, en primer lugar, mejorar el proceso de RAG posterior y, en segundo lugar, tener un *dataset* etiquetado con el que se entrenará (mediante ajuste fino o *fine-tuning*) un modelo tipo DEBERTAV3 para que sea capaz de clasificar un nuevo *chunk* de texto del BOE en la categoría adecuada sin utilizar un LLM de tipo GPT o de tipo *transformer-decoder*. Esto tiene como objetivo ahorrar costes de inferencia o de utilización de estos modelos en un futuro para la clasificación y el uso de este sistema de RAG.

El script *llm.py* es utilizado para la creación de un *dataset* de *chunks* “multi-clasificados” con tres etiquetas. La finalidad de este código de Python es automatizar la clasificación de textos oficiales (específicamente documentos del BOE) en diferentes categorías predefinidas. Para ello, utiliza modelos de lenguaje natural avanzados que son capaces de entender el contenido de los textos y asignar etiquetas correspondientes según su similitud semántica con las categorías definidas.

Las librerías utilizadas por este script son muy importantes en el mundo de la IA y la creación de aplicaciones basadas en IA. Se importan diversas librerías esenciales como *transformers* para los modelos de lenguaje, *dotenv* para cargar variables de entorno, y *matplotlib* para la configuración de gráficos.

- *Tokenizers*: Se configuran varios *tokenizers* para diferentes modelos de lenguaje, como GPT-3.5 (*tiktoken*), GPT-2 (*GPT2Tokenizer*), y LLAMA-3 (*AutoTokenizer*). Un *tokenizer* (tokenizador en español) es una herramienta o proceso utilizado en el procesamiento de lenguaje natural (NLP) que divide un texto en unidades más pequeñas llamadas tokens. Los tokens pueden ser palabras individuales, subpalabras, caracteres o incluso símbolos de puntuación, dependiendo del tipo de “tokenizador” y su configuración. Los modelos de lenguaje, como BERT, GPT, o LLAMA, no pueden procesar texto en bruto directamente. En su lugar, necesitan que el texto sea convertido en una secuencia de tokens, que son representaciones más manejables del texto original. Estos tokens se convierten en vectores numéricos que el modelo puede interpretar y procesar. Existen varios tipos de *tokenizers*:
 1. *Word Tokenizer* que divide el texto en palabras individuales.
 2. *Subword Tokenizer* el cual divide las palabras en subunidades más pequeñas, que pueden ser combinaciones de letras que tienen significado en diferentes contextos. Este enfoque es útil para manejar palabras desconocidas o raras.
 3. *Character Tokenizer* que divide el texto en caracteres individuales.
 4. *Sentence Tokenizer* el cual divide el texto en oraciones completas. Este tipo de “tokenización” es útil cuando el contexto de toda la oración es necesario para la tarea.

Los *tokenizers* se entrenan junto con los modelos de lenguaje. Por ejemplo, el “tokenizador” de BERT está diseñado para dividir el texto en subpalabras, lo que le permite manejar vocabularios más pequeños, pero suficientemente expresivos para capturar una amplia gama de contextos lingüísticos (Jurafsky D. &, 2021).

En el caso, por ejemplo, del modelo BERT, su *tokenizer* utiliza un enfoque basado en subpalabras, conocido como *WordPiece*.

Por el contrario, el modelo GPT, usa un “tokenizador” con un enfoque de subpalabras conocido como *Byte Pair Encoding* (BPE).

Ambos algoritmos de “tokenización” han sido explicados con mas detalle en los anexos III y IV del presente trabajo.

Posteriormente, dentro del script se define la clase *LabelGenerator*, que es la clase principal encargada de gestionar la clasificación de los documentos del BOE.

Entre sus atributos están:

- **LABELS**: Una lista de etiquetas que representan las categorías posibles en las que se pueden clasificar los documentos.
- **model_label**: Indica el modelo de lenguaje que se utilizará para la clasificación (ej. GPT, LLAMA).

Jorge Resino Martin

- `tokenizer`: El *tokenizer* específico que corresponde al modelo seleccionado.
- `prompt`: Plantilla de *prompt* utilizada para generar las instrucciones que se pasan al modelo de lenguaje para la clasificación.
- `chain`: Este atributo combina el *prompt*, el modelo de lenguaje seleccionado, y un *parser* de salida JSON para obtener los resultados de la clasificación.

Los métodos de esta clase incluyen:

- `__init__`: Constructor de la clase que inicializa el modelo, el *prompt*, y las etiquetas.
- `_get_tokens`: Calcula el número de tokens de un texto utilizando el *tokenizer* configurado.
- `invoke`: Este es el método central que procesa una lista de documentos. Para cada documento:
 1. Se calculan los tokens y se actualizan los metadatos añadiendo el número de tokens de cada *chunk* como metadato del objeto *document*.
 2. Se llama al modelo y se genera la clasificación del *chunk*.
 3. Se asignan las etiquetas generadas por el modelo a los metadatos del objeto *document*.

Se configuran diferentes modelos de lenguaje a través de la librería *langchain_community*, que permite integrar modelos como GPT, LLAMA, y NVIDIA-LLAMA3. El modelo seleccionado por el usuario se utiliza para procesar y clasificar los textos según las instrucciones definidas en los *prompts*. Un *prompt* es una secuencia de texto que se utiliza como entrada para un modelo de lenguaje, como GPT-3, con el fin de guiar o iniciar la generación de texto por parte del modelo. En términos técnicos, un *prompt* es la información inicial que el modelo recibe para comenzar a producir una respuesta o continuación de texto. El diseño y contenido del *prompt* pueden influir significativamente en la calidad, relevancia y precisión de la salida generada por el modelo (Brown, 2020).

Se definen varios *templates* de *prompts* que son instrucciones textuales que se envían al modelo de lenguaje para guiar su respuesta. Estos *prompts* especifican que el modelo debe clasificar el texto en base a las etiquetas proporcionadas en la variable `LABELS` y devolver un resultado en formato JSON, para que pueda ser recuperado el output del modelo en código.

En los archivos de configuración, el usuario puede definir el modelo LLM para clasificar cada *chunk* y el *tokenizer*, solo que se ha establecido por coste GPT-3.5 y su *tokenizer*.

Dentro del proyecto, este script tiene la función crítica de automatizar la clasificación de documentos del BOE en categorías predefinidas. Este proceso es esencial para organizar la información de manera eficiente y facilitar su recuperación posterior. Los modelos de lenguaje empleados permiten una clasificación precisa y escalable, adecuada para manejar grandes volúmenes de documentos oficiales.

Como resumen de la funcionalidad del script, este es una implementación sofisticada que combina procesamiento de lenguaje natural, clasificación automática, y uso de modelos avanzados para estructurar y organizar documentos oficiales en categorías significativas. Esto

no solo mejora la eficiencia del manejo de la información, sino que también sienta las bases para tareas más avanzadas de análisis y recuperación de datos.

Archivos de configuración del módulo ETL

Como se ha mencionado anteriormente, en los archivos de configuración de tipo JSON, el usuario puede definir los atributos de cada una de estas clases y con ello variar el tipo procesamiento, extracción, división de texto, descarga... de los PDFs del BOE. En los anexos del presente trabajo se exponen los diferentes archivos de configuración para este y demás módulos de la herramienta.

Después, todas estas clases son instanciadas en un *script* llamado *pipeline.py* que se encarga de (gracias al archivo de configuración) instanciar cada clase y ejecutar los métodos de estas de manera ordenada para procesar los PDFs y obtener una serie de objetos de tipo *Document* que almacenan el fragmento de texto (también conocido en la literatura como *chunk*) y los metadatos asociados a cada uno de estos fragmentos. Tras el procesado de los PDFs y como *output* final de este módulo, se genera una lista de otro tipo de objetos llamado *ClassifyChunk* con los siguientes atributos:

Atributos	Descripción
text	Contenido de texto del fragmento a clasificar
file_path	Ruta del archivo del cual se ha extraído el fragmento
file_name	Nombre del archivo del cual se ha extraído el fragmento
file_type	Tipo de archivo (por ejemplo, PDF, DOCX, etc.)
file_size	Tamaño del archivo en bytes
creation_date	Fecha de creación del archivo
last_modified_date	Fecha de la última modificación del archivo
fecha_publicacion_boe	Fecha de publicación del documento en el Boletín Oficial del Estado (BOE)
orden	Orden relacionada con el fragmento, puede ser una lista de <i>strings</i> o un <i>string</i> único
real_decreto	Referencias a Reales Decretos relacionadas con el fragmento, puede ser una lista de <i>strings</i> o un <i>string</i> único

ministerios	Ministerios relacionados con el fragmento, puede ser una lista de <i>strings</i> o un <i>string</i> único
pdf_id	Identificador único del PDF del cual se ha extraído el fragmento
chunk_id	Identificador único del fragmento de texto
num_tokens	Número de tokens (unidades léxicas) en el fragmento de texto
num_caracteres	Número de caracteres en el fragmento de texto
label2id	Diccionario que mapea etiquetas a identificadores numéricos enteros, basado en una lista de etiquetas generada por la clase <i>LabelGenerator</i>
label	Lista de etiquetas iniciales asignadas al fragmento, por defecto contiene tres valores ("999", "999", "999"), que se utilizan para etiquetas por defecto

Tabla 2: Atributos de cada objeto chunk tras ser procesado.

Una vez que se tiene esta lista de objetos de la clase *ClassifyChunk* se convierten de nuevo a objetos *Document* y el *pipeline* instancia la clase *Storer* con la información de los archivos de configuración JSON.

La clase *Storer* es una utilidad diseñada para almacenar documentos procesados en un formato estructurado, como un archivo CSV, *Parquet* o *Feather*. Esta clase facilita la conversión de una lista de objetos *Document* en un *DataFrame* de la librería pandas y luego guarda ese *DataFrame* en un archivo con el formato especificado por el usuario.

Los métodos de la Clase *Storer* son:

- `__init__`: Este es el constructor de la clase. Se encarga de inicializar los atributos principales:
 1. `store_path`: Es la ruta donde se almacenará el archivo.
 2. `file_name`: El nombre del archivo (sin la extensión).
 3. `file_format`: El formato del archivo, que por defecto es CSV. Puede ser 'csv', 'parquet' o 'feather'.
- `_document_to_dataframe`: Este método privado convierte una lista de objetos *Document* en un *DataFrame* de pandas. Cada *Document* tiene un contenido (`page_content`) y metadatos asociados. El método crea un diccionario para cada documento, donde el texto del documento se almacena bajo la clave "text" y los metadatos se añaden como otras columnas del *DataFrame*. El resultado es un *DataFrame* donde cada fila representa un documento y las columnas representan el contenido del documento y sus metadatos.

- `_get_id`: Este es otro método privado que genera un identificador único (UUID). Esto es útil para crear identificadores únicos para los documentos o archivos que se están almacenando.
- `_store_dataframe`: Este método privado guarda el *DataFrame* en un archivo, en el formato especificado por *file_format*. Dependiendo del valor de *file_format*, el *DataFrame* se guarda como un archivo CSV, *Parquet* o *Feather*. Si el formato no es soportado, se lanza una excepción *ValueError* y se registra el error.
- `invoke`: Este es el método principal que se llama para ejecutar la funcionalidad de la clase. Asegura que el directorio de salida especificado por *store_path* exista, creando los directorios necesarios si no están presentes. Convierte la lista de documentos en un *DataFrame* utilizando el método `_document_to_dataframe`. Genera el nombre del archivo utilizando la fecha y hora actuales, junto con el *file_name* proporcionado y el *file_format*. Guarda el *DataFrame* en el formato especificado y en la ruta determinada mediante el método `_store_dataframe`.

Por todo esto, la clase *Storer* está diseñada para pasar de unos fragmentos de texto del BOE semejantes a nivel semántico y con ciertos metadatos asociados que los identifican y definen a archivos de tipo CSV, *Parquet* o *Feather* los cuales se guardan de manera local y son mucho óptimos de manejar en fases posteriores del proyecto o de la aplicación.

Script *hg_push.py*

La última parte del proceso y del módulo ETL consiste en convertir a un objeto *dataset* y subir estos archivos mencionados a un repositorio personal y publico de *Hugging Face*.

Hugging Face es una plataforma líder en el campo del procesamiento de lenguaje natural (NLP) y el aprendizaje automático. Ofrece herramientas y bibliotecas que permiten a los desarrolladores y científicos de datos trabajar con modelos de lenguaje avanzados, como *transformers*. Uno de los componentes más conocidos de *Hugging Face* es su biblioteca *transformers* ya mencionada en el presente trabajo, que proporciona acceso a modelos preentrenados y facilita la implementación de tareas como clasificación de texto, generación de texto, traducción automática, entre otros.

Además de las bibliotecas, *Hugging Face* también incluye la *Hugging Face Hub*, una plataforma de repositorios donde los usuarios pueden compartir y colaborar en modelos de *machine learning* y *datasets*. Esta plataforma permite a la comunidad acceder a modelos de última generación, entrenados y compartidos por otros usuarios, lo que acelera la innovación y el desarrollo en el campo del *machine learning* (Hugging Face, s.f.).

Enunciada esta información previa y necesaria, este script de Python define una clase llamada *HGDataset*, la cual está diseñada para manejar operaciones relacionadas con *datasets* en la plataforma de *Hugging Face* (HG). La clase permite leer, combinar, limpiar y subir conjuntos de datos a la *Hugging Face Hub*.

Los atributos de la Clase *HGDataset* son:

Jorge Resino Martin

- `data_dir_path`: Especifica el directorio donde se encuentran los archivos de datos, ya sean en formato CSV o *Parquet*.
- `hg_api_token`: Almacena el token de autenticación para interactuar con la API de *Hugging Face*.
- `repo_id`: Identifica el repositorio en *Hugging Face* donde se subirá el *dataset*.
- `from_date` y `to_date`: Definen el rango de fechas de los archivos que se quieren procesar y subir a la plataforma.
- `desire_columns`: Una lista opcional de columnas que se desean conservar en el *dataset* procesado.
- `data`: Atributo opcional donde se almacenará el *DataFrame* combinado de todos los archivos de datos.

Los métodos de esta clase son:

- `initialize_data`: Inicializa el atributo `data` combinando y limpiando los archivos de datos en el directorio especificado. Llama a los métodos `_get_data` y `_clean_data` para realizar estas operaciones.
- `_get_data`: Lee y combina los datos de los archivos CSV y *Parquet* dentro del rango de fechas especificado. Utiliza `os.listdir` para obtener la lista de archivos en el directorio, filtra los archivos por nombre usando la fecha y luego los lee en *DataFrames* individuales, que son combinados en un único *DataFrame*.
- `_clean_data`: Limpia los datos manteniendo solo las columnas especificadas en `desire_columns`. Si no se especifican columnas, se mantiene todo el *DataFrame*.
- `parse_date`: Método estático que convierte una cadena de texto en un objeto de fecha *datetime*. Es útil para analizar las fechas en los nombres de los archivos.
- `_get_hg_dataset`: Divide los datos en tres subconjuntos: entrenamiento, validación y prueba. Estos se almacenan en un objeto *DatasetDict*, que es una estructura de datos utilizada en *Hugging Face* para manejar *datasets* de manera eficiente. También se asegura de eliminar columnas innecesarias de los *datasets* antes de subirlos.
- `push_to_hub`: Este método sube el *dataset* a *Hugging Face Hub*. Puede especificar si el repositorio debe ser privado, cuál es el token de autenticación y en qué rama del repositorio se debe hacer el *commit*.

Dentro del proyecto, esta clase desempeña un papel crucial en el proceso de Extracción, Transformación y Carga (ETL) de datos. La clase extrae los datos de archivos almacenados localmente en un directorio específico. Estos archivos contienen información relevante que se desea procesar y subir a *Hugging Face*. Los datos extraídos son limpiados y transformados. Por ejemplo, se filtran las columnas que no son necesarias y se dividen los datos en conjuntos de entrenamiento, validación y prueba. Este paso asegura que los datos estén en un formato adecuado para ser utilizados en modelos de *machine learning*. Finalmente, los datos transformados se cargan en *Hugging Face Hub*, donde pueden ser accedidos y utilizados por otros modelos o usuarios.

Este script automatiza gran parte del proceso ETL, facilitando la gestión de grandes volúmenes de datos y asegurando que los *datasets* estén siempre actualizados y disponibles en una

plataforma centralizada como *Hugging Face Hub*. Esto es especialmente útil en proyectos que requieren la integración continua de datos y modelos, permitiendo a los equipos de desarrollo y machine *learning* colaborar de manera más eficiente y, como se ha mencionado, es el último paso en el módulo y proceso ETL. Una vez se tienen los *chunk* clasificados y subidos a *Hugging Face Hub* se pueden recuperar para el siguiente paso del proyecto: *Fine tuning* del modelo DeBERTa V3 para la clasificación de fragmentos de texto del BOE.

4.2.2 Ajuste fino (*Fine-tuning*) del modelo DeBERTa V3 para la clasificación de fragmentos de texto del BOE

Modelo DeBERTa V3

DeBERTaV3 es una versión mejorada del modelo DeBERTa, diseñada para abordar las limitaciones en la eficiencia del preentrenamiento y mejorar el rendimiento en tareas de comprensión del lenguaje natural (NLU). Este modelo se construye sobre los cimientos de DeBERTa, incorporando innovaciones clave del modelo ELECTRA y mejorando el proceso de preentrenamiento mediante técnicas avanzadas de compartición de *embeddings* y mecanismos de atención.

El modelo original DeBERTa se caracteriza por su uso de la atención des entrelazada (*Disentangled Attention, DA*), donde se separan los vectores que representan el contenido y la posición de cada palabra en una secuencia. Esto permite al modelo capturar mejor las relaciones entre palabras considerando tanto su contenido como sus posiciones relativas. DeBERTa también utiliza un decodificador de máscara mejorado que incorpora información de posición absoluta, lo que mejora su capacidad para predecir palabras enmascaradas.

Sin embargo, a pesar de estas innovaciones, DeBERTa seguía dependiendo de la tarea de preentrenamiento de *Masked Language Modeling (MLM)*, que, aunque efectiva, no es la más eficiente en términos de uso de datos y convergencia del modelo.

Para mejorar la eficiencia del preentrenamiento, DeBERTaV3 reemplaza MLM por RTD, una técnica introducida por ELECTRA. RTD es un enfoque más eficiente que consiste en entrenar un generador para producir reemplazos ambiguos de tokens en una secuencia y luego entrenar un discriminador para identificar si cada token es original o ha sido reemplazado.

Este cambio de paradigma permite a DeBERTaV3 aprender de manera más efectiva a partir de las secuencias de entrada, ya que el modelo no solo se centra en predecir los tokens correctos, sino también en discernir entre originales y reemplazos. Esto resulta en un uso más eficiente de los datos de entrenamiento y una convergencia más rápida del modelo.

En ELECTRA, tanto el generador como el discriminador comparten los mismos *embeddings* de tokens, lo que introduce un problema de "tug-of-war" o conflicto de gradientes. El generador, entrenado con MLM, intenta acercar los *embeddings* de tokens semánticamente similares, mientras que el discriminador, entrenado con RTD, busca separarlos para mejorar la clasificación binaria. Este conflicto afecta negativamente la calidad del preentrenamiento.

Para resolver este problema, DeBERTaV3 introduce el método *de Gradient-Disentangled Embedding Sharing* (GDES). GDES permite que el generador y el discriminador compartan *embeddings*, pero desacopla los gradientes que afectan estos *embeddings*. Específicamente, GDES reparametriza los *embeddings* del discriminador como una suma de los *embeddings* del generador y un término de corrección que se actualiza únicamente con los gradientes del discriminador, evitando que estos afecten directamente a los *embeddings* del generador.

Este enfoque asegura que el modelo se beneficie de la riqueza semántica de los *embeddings* generados, sin los efectos negativos del conflicto de gradientes, mejorando así la eficiencia del preentrenamiento y la calidad del modelo final.

DeBERTaV3 se entrenó utilizando los mismos datos que DeBERTa, incluyendo Wikipedia, BookCorpus, CC-News, OpenWebText y Stories, lo que suma un total de 160 GB de texto. Se entrenaron varias configuraciones del modelo, incluyendo versiones grandes, base y pequeñas, para evaluar su rendimiento en diferentes tareas de NLU.

Los resultados muestran que DeBERTaV3 supera consistentemente a modelos previos, incluidos DeBERTa y ELECTRA, en una variedad de *benchmarks*. Por ejemplo, en el conjunto de tareas de GLUE, DeBERTaV3-large logra un puntaje promedio de 91.37%, que es 1.37% más alto que DeBERTa y 1.91% más alto que ELECTRA, estableciendo un nuevo estado del arte (SOTA) para modelos con estructuras similares.

Además de su implementación en inglés, DeBERTaV3 también se ha entrenado como un modelo multilingüe, denominado mDeBERTaV3. Este modelo se entrenó con el conjunto de datos CC100, que abarca múltiples idiomas, y mostró mejoras significativas sobre otros modelos multilingües como XLM-R. Por ejemplo, mDeBERTaV3-base logró una precisión del 79.8% en la tarea XNLI de transferencia cruzada de idiomas, superando a XLM-R base en 3.6%.

DeBERTaV3 es una mejora significativa en la serie DeBERTa, que integra de manera efectiva innovaciones de modelos anteriores como ELECTRA y refina el proceso de preentrenamiento para mejorar tanto la eficiencia como el rendimiento. Con su enfoque en la resolución de conflictos de gradientes y la incorporación de técnicas avanzadas de compartición de *embeddings*, DeBERTaV3 establece un nuevo estándar en modelos de lenguaje natural, especialmente en su capacidad para manejar tareas multilingües y comprender mejor el lenguaje con menos datos de entrenamiento (He P. G., 2023).

Notebook training.ipynb

Este notebook está diseñado para realizar el *fine-tuning* del modelo DeBERTaV3 de *Hugging Face* para una tarea de clasificación de textos del BOE, que previamente y mediante el método anterior han sido divididos en fragmentos, preprocesados y clasificados bajo ciertas categorías. El objetivo es tener un modelo tipo BERT que, llegado un fragmento de texto del BOE le asigne una etiqueta o clase y lo clasifique para automatizar el proceso y no tener que utilizar continuamente o llamar a un LLM tipo *decoder* (GPT). Esto abarata o busca abaratar costes, aunque inicialmente la inversión del ajuste fino del modelo no lo sea.

A continuación, se describe el funcionamiento del notebook dividido en secciones clave:

- Configuración Inicial y Preparación del Entorno
 1. Instalación de Dependencias: El notebook instala las librerías necesarias como *transformers*, *datasets*, *torch*, *accelerate*, entre otras. Estas son fundamentales para el manejo de modelos preentrenados y la ejecución eficiente en GPU.
 2. Montaje de Google Drive: En caso de ejecutarse en Google Colab, se monta Google Drive para acceder a los archivos y guardar resultados.
 3. Información del Sistema: Se verifican las versiones de PyTorch, CUDA y se obtienen detalles del entorno de ejecución, como el nombre del dispositivo (GPU o CPU) y el sistema operativo.
- Autenticación en Hugging Face
 1. Autenticación: Se utiliza un token de autenticación para conectarse a *Hugging Face Hub*, lo que permite cargar y guardar modelos desde y hacia el repositorio de *Hugging Face*.
- Carga y Preparación del *Dataset*
 1. Carga del *Dataset*: Se carga un conjunto de datos específico usando la función `load_dataset` de la librería *datasets*. En este caso, el *dataset* se obtiene de un repositorio en *Hugging Face*. Este repositorio es el que ha sido previamente utilizado y tras la clasificación de los *chunks*, es decir, especificando la fecha de se pueden obtener los *chunks* clasificados que se quiera.
 2. Tokenización: Se “tokenizan” los textos del *dataset* usando un *tokenizer* preentrenado, adaptando las secuencias de texto para que sean compatibles con la entrada del modelo. Esto incluye la adición de *padding* y truncamiento para gestionar las longitudes variables de las secuencias.
- Definición de Etiquetas
 1. Gestión de Etiquetas: Se crea un diccionario de etiquetas (`label2id` e `id2label`) que asigna cada etiqueta a un identificador numérico. Además, se normalizan las etiquetas para eliminar acentos y otros caracteres especiales.
- Preprocesamiento del Dataset
 1. Preprocesamiento de las Entradas: Se define una función de preprocesamiento que aplica la tokenización y convierte las etiquetas de texto en un formato adecuado para el entrenamiento, utilizando codificación *multi-hot* para la clasificación “multi-etiqueta”.
- Configuración y Carga del Modelo
 1. Configuración del Modelo: Se carga una configuración (*AutoConfig*) para el modelo DeBERTaV3, especificando el número de etiquetas y los mapeos `id2label` y `label2id`. La configuración completa se adjunta en los anexos del presente trabajo.
 2. Carga del Modelo Preentrenado: Se carga el modelo DeBERTaV3 preentrenado con la configuración definida, adaptándolo para la tarea de clasificación de secuencias.
- Arquitectura de DeBERTa V3

A continuación, se muestra la arquitectura del modelo utilizado dividida en las partes principales que lo conforman y dentro de estas las subpartes.

- *Embeddings*:
 - *word_embeddings*: *Embedding* de palabras con una dimensión de entrada de 128,100 y una dimensión de salida de 768. Se utiliza un índice de *padding* (*padding_idx=0*).
 - *LayerNorm*: Normalización de capas (*LayerNorm*) con una dimensión de 768 y un valor de epsilon de 1e-07 para evitar divisiones por cero. Incluye ajuste de parámetros (*elementwise_affine=True*). La normalización de capas (*Layer Normalization*) es una técnica que se aplica en redes neuronales para estabilizar el proceso de entrenamiento, particularmente útil en redes recurrentes. A diferencia de la normalización por lotes (*Batch Normalization*), que normaliza las entradas de una neurona utilizando la media y la varianza calculadas sobre un mini-lote de datos, la normalización de capas calcula estos valores a partir de todas las entradas sumadas a las neuronas dentro de una misma capa para un solo caso de entrenamiento. Esto elimina la dependencia de las estadísticas del mini-lote, permitiendo una normalización más consistente, independientemente del tamaño del lote, y mejora la estabilidad en redes recurrentes al manejar la dinámica de estados ocultos, lo que puede reducir significativamente el tiempo de entrenamiento y mejorar el rendimiento general del modelo (Ba, 2016).
 - *dropout*: Se utiliza un mecanismo de *StableDropout* para regularización. El *dropout* es una técnica de regularización en redes neuronales diseñada para prevenir el sobreajuste. Durante el entrenamiento, se apagan aleatoriamente un subconjunto de neuronas en cada iteración, lo que fuerza a la red a aprender representaciones más robustas y generales de los datos. Esto mejora la capacidad del modelo para generalizar a datos nuevos, ya que evita que la red dependa excesivamente de neuronas específicas. El *dropout* se configura mediante un hiperparámetro que define la probabilidad de desactivación de las neuronas en cada iteración, siendo común un valor de 0.5 (Srivastava, 2014). El *Stable Dropout* es una variante del método tradicional de *dropout* en redes neuronales, diseñada para mejorar la estabilidad del proceso de entrenamiento. A diferencia del *dropout* convencional, que apaga completamente un subconjunto aleatorio de neuronas durante cada iteración del entrenamiento, el *stable dropout* modifica las activaciones neuronales de manera más suave, evitando apagarlas por completo. Este enfoque reduce la variabilidad introducida por

el *dropout*, lo que a su vez disminuye el ruido en los gradientes y mejora la estabilidad durante el entrenamiento, especialmente en redes profundas. Esto es particularmente útil en situaciones donde la estabilidad es crítica para la convergencia del modelo (Labach, 2019).

- *Encoder*:
 - *layer*: El *encoder* está compuesto por 12 capas (0-11) del tipo DebertaV2Layer, cada una de las cuales incluye:
 - *Attention* (Atención):
 - DisentangledSelfAttention: Atención auto-desentrelazada con proyecciones lineales para las consultas (query_proj), claves (key_proj) y valores (value_proj), cada una con una dimensión de entrada y salida de 768.
 - pos_dropout y dropout: Dropout estable aplicado para regularización durante el proceso de atención.
 - SelfOutput:
 - dense: Capa densa que transforma la salida de la atención de 768 dimensiones a 768.
 - LayerNorm: Normalización de capas con parámetros similares a los de las *embeddings*.
 - dropout: Dropout estable para regularización.
 - Intermediate:
 - dense: Capa densa que aumenta la dimensionalidad de 768 a 3072.
 - intermediate_act_fn: Función de activación GELU (*Gaussian Error Linear Unit*) para no linealidad. Esta función y sus ventajas frente a otras funciones de activación se describe en los anexos del presente trabajo.
 - Output:
 - dense: Capa densa que reduce la dimensionalidad de 3072 a 768.
 - LayerNorm: Normalización de capas con los mismos parámetros utilizados anteriormente.
 - dropout: Dropout estable para regularización.
 - rel_embeddings: *Embedding* relativo con 512 dimensiones de entrada y 768 dimensiones de salida.
 - LayerNorm: Aplicación de normalización de capas similar a la de las *embeddings*.
- *Pooler*:

- ContextPooler: Agregador de contexto que incluye:
 - dense: Capa densa que transforma la representación final con 768 dimensiones.
 - dropout: *Dropout* estable para regularización.
- *Classifier*:
 - classifier: Capa final de clasificación, que toma las 768 dimensiones de entrada y las transforma en 41 salidas, correspondientes al número de clases en la tarea de clasificación. En este caso las 41 categorías elegidas del BOE.
 - dropout: Dropout estable adicional para regularización.
- Entrenamiento del Modelo
 1. Definición de Hiperparámetros de Entrenamiento: Se configuran los parámetros de entrenamiento utilizando *TrainingArguments*. Esta es una instancia de una clase la cual después se pasará al *Trainer* para llevar a cabo el ajuste fino del modelo. En la siguiente tabla se pueden observar todos estos hiperparámetros utilizados durante el ajuste fino del modelo.

```
output_dir=output_dir, # Nombre del modelo (se crea un directorio con este nombre)
overwrite_output_dir=True, # Sobrescribir el directorio de salida si existe
evaluation_strategy="epoch", # Evaluamos y guardamos en cada época
report_to="tensorboard", # Reportar a TensorBoard para monitorizar el progreso del
entrenamiento
save_strategy="epoch", # Guardamos un checkpoint del modelo en cada época
learning_rate=3e-05, # Tasa de aprendizaje usada en el optimizador Adam
per_device_train_batch_size=1, # Tamaño de batch en entrenamiento
per_device_eval_batch_size=1, # Tamaño de batch en evaluación
num_train_epochs=1, # Número de épocas para entrenar
weight_decay=0.01, # Decaimiento de peso en el optimizador Adam
adam_epsilon=1e-08, # Valor para el parámetro epsilon de Adam
load_best_model_at_end=True, # Cargar el mejor modelo al final del entrenamiento
metric_for_best_model="f1", # Métrica para escoger el mejor modelo
gradient_accumulation_steps=10, # Número de pasos en los que se acumula gradiente
warmup_ratio=0.03, # Porcentaje de pasos de entrenamiento para el "warmup" del learning rate
```



```
fp16_full_eval=False, # Evaluación en precisión mixta no completa
fp16=False, # Desactivar la precisión mixta
push_to_hub=True, # Habilitar la subida al hub
hub_model_id="koke143/boe_classifier", # Nombre del repositorio del modelo en el hub
hub_strategy="checkpoint", # Estrategia de subida al hub
```

Tabla 3: Hiperparámetros para el ajuste fino (entrenamiento de un modelo pre entrenado) del modelo.

2. Función de Evaluación: Se define una función para calcular métricas de rendimiento, como el F1 score, utilizando la biblioteca *evaluate*.
3. Entrenamiento con *Trainer*: Se utiliza la clase *Trainer* de *transformers* para entrenar el modelo. *Trainer* facilita el proceso de entrenamiento al manejar automáticamente la optimización, evaluación y registro de métricas.
- Evaluación y Guardado del Modelo
 1. Evaluación: Después del entrenamiento, se evalúa el modelo en el conjunto de datos de validación para obtener las métricas de rendimiento.
 2. Guardado y Publicación: Finalmente, el modelo entrenado se guarda y se publica en el *Hugging Face Hub*, permitiendo su uso futuro.

En conclusión, ejecutando celda a celda de este notebook, se automatiza el proceso de *fine-tuning* del modelo DeBERTaV3 para una tarea de clasificación de texto, en este caso, del BOE. El código utiliza o se basa en la infraestructura proporcionada por *Hugging Face*, lo que facilita el acceso a modelos preentrenados, el manejo de datos y el entrenamiento eficiente en hardware acelerado como GPUs, evitando así los grandes costes que supone guardar en local un modelo de este calibre y ajustarlo a un *dataset* específico.

4.2.3 RAPTOR modificado

En el anexo X del presente trabajo se detalla y describe, basándose en el *paper* original, RAPTOR. Este sistema consiste en, de manera resumida, dividir un corpus de texto en fragmentos mediante alguna técnica, crear los *embeddings* de estos fragmentos, usar un algoritmo de “clusterización” para agrupar los fragmentos en *clusters* y después generar resúmenes de estos grupos de fragmentos de texto (en teoría de similar significado semántico) con algún modelo LLM.

Sabiendo esto y con la explicación mas detallada del anexo, en el presente trabajo, se va a realizar una adaptación de este algoritmo o sistema RAPTOR. Se ha tomado como inspiración este sistema que, según el *paper*, tan buen resultado ha dado en las aplicaciones que implementan RAG.

Antes de describir el sistema RAPTOR modificado es necesario recordar que en este punto se tienen los fragmentos de texto del BOE y recordar en qué formato se encuentran. En este punto de la herramienta, se cuenta con los fragmentos semánticamente semejantes o cercanos de cada archivo del BOE clasificados con la misma etiqueta o clase. Tomando como referencia el

algoritmo de RAPTOR, se van a tomar los fragmentos de la misma clase o etiqueta y usar un LLM para resumir ese conjunto de textos o *chunks*. Además, se añade como metadato de este *chunk* este resumen. Cada resumen, para cada una de las clases, se ha llamado con la clave (dentro del diccionario de metadatos): *cluster_summary*.

De esta forma se busca mejorar el posterior proceso de aportar contexto a los modelos del grafo. Debido a que se añade un nuevo filtro en la búsqueda de *chunks* similares en la base de datos vectorial. Ya no solo se buscará por métricas relacionadas con la distancia geométrica entre vectores densos o *embeddings* sino también por estos *clusters*. En el grafo habrá un modelo LLM que a la pregunta o *query* del usuario le asignará una de las clases o categorías de entre todas las 41 existentes elegidas para clasificar los textos del BOE. Una vez validada esta categoría se cruzarán los metadatos de los *chunks* en la base de datos vectorial con la misma categoría y posteriormente se aplicarán métricas típicas que miden la distancia geométrica entre los *embeddings* de la *query* y los *chunks* filtrados por categoría.

Para lograr incorporar este nuevo metadato al *dataset*, formado por los similarmente semánticos fragmentos del BOE, se han creado y diseñado dos *scripts* de Python dentro del modulo RAPTOR de la herramienta. Uno se encarga de procesar los *chunks* y llamar a un LLM que genere en resumen por *cluster* y otro de conectarse con la base de datos vectorial, crear los *embeddings* de los fragmentos de texto y almacenar este vector y los metadatos de cada *chunk*. A continuación, se definen con más detalle ambos.

Script RAPTOR BOE.py

En este script se ha materializado el algoritmo anteriormente descrito, añadiendo en los metadatos de los *chunks* dichos resúmenes de los *clusters* generados por un modelo LLM.

Mas en detalle, este script en Python ha sido diseñado para procesar y analizar estos fragmentos textuales del Boletín Oficial del Estado (BOE) en España. En este se utilizan una variedad de bibliotecas y herramientas, incluyendo Pandas para la manipulación de datos, LangChain para la interacción con modelos de lenguaje, y *Pydantic* para la validación de datos. El script está estructurado en torno a dos clases principales: *ClusterSummaryGenerator* y *RaptorDataset*.

La clase *ClusterSummaryGenerator* es responsable de generar resúmenes de *clusters* de *chunks* o fragmentos clasificados dentro de la misma categoría del BOE, pudiendo utilizar diferentes modelos de lenguaje. Los resúmenes se crean basándose en un *prompt* que se ha diseñado para reflejar con precisión el contenido original, sin añadir comentarios o interpretaciones adicionales en el resumen.

La clase se inicializa el modelo (por defecto es 'GPT') su “tokenizador” usando la biblioteca *tiktoken* y prepara el *prompt* utilizando la clase *PromptTemplate* que define cómo se formateará el texto de entrada para el modelo.

La clase admite varios modelos (por ejemplo, GPT-3.5, GPT-4o-mini, NVIDIA-LLAMA3, etc.), y según el modelo seleccionado, construye una cadena de procesamiento (*chain* de *LangChain*) que incluye el modelo, el *prompt* y un *parser* de salida.

La clase *RaptorDataset* lee, limpia y procesa datos de archivos (CSV o Parquet) e inicializa un objeto *ClusterSummaryGenerator* que se invoca y genera resúmenes para los *clusters*.

El método *initialize_data* orquesta el proceso de inicialización de datos. Lee los datos de los archivos especificados, los limpia, procesa los metadatos, genera resúmenes de texto para los *clusters* y, finalmente, convierte los datos procesados en una lista de objetos *Document* de *LangChain*. La razón de esta conversión de un archivo CSV a un *Dataframe* y por último a una lista de *Document* es la versatilidad que da este tipo de objetos y la integración con la propia librería *LangChain*.

Script raptor_vectordb.py

Este script en Python está diseñado para interactuar con una base de datos vectorial, específicamente *Pinecone*, almacenar y recuperar objetos de tipo *Document* utilizando *embeddings* generados por modelos de lenguaje como los proporcionados por *HuggingFace*. El script incluye la configuración del modelo de *embeddings*, la conexión a la base de datos vectorial, y la ejecución de operaciones de almacenamiento y recuperación de documentos basados en consultas.

Es decir, la clase *RaptorVectorDB* permite, almacenar los metadatos de cada objeto de la lista de documentos anteriormente procesados, generar con el modelo de *embedding: sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2* (de tipo SBERT) el vector denso que representa al texto del *Document* y almacenar todo ello en la base de datos vectorial.

También la clase *RaptorVectorDB* gestiona la conexión y la interacción con la base de datos vectorial (en este caso, *Pinecone*).

El constructor de la clase toma como parámetros una clave API, el nombre del índice en *Pinecone*, y un modelo de *embeddings*. Carga variables de entorno, inicializa el modelo de *embeddings* y establece una conexión con el índice de *Pinecone* para crear un objeto *retriever* y un objeto *vectorstore*. Ambos objetos son claves y muy utilizados por la librería *LangChain* para

El *retriever*, realmente, es un objeto *VectorStoreRetriever*, que es utilizado para recuperar documentos similares basados en la consulta del usuario sobre la base de datos vectorial. Utiliza la estrategia de distancia geométrica del coseno para medir la similitud entre *embeddings*. Primero genera el *embedding* de la consulta y calcula la mínima distancia (el coseno entre vectores) entre este vector y todos los de la base de datos vectorial mediante el algoritmo HNSW (detallado en el anexo XI) para después retornar los de menor distancia o mayor coseno, que es equivalente a nivel matemático.

Una funcionalidad de esta clase muy importante es que permite realizar consultas textuales contra la base de datos para recuperar documentos relevantes, utilizando filtros específicos de metadatos para afinar los resultados. Es decir, complementando la mínima distancia geométrica entre *embeddings* con filtros en determinados valores de sus metadatos.

4.2.4 Grafo multi Agente con sistema RAG mejorado

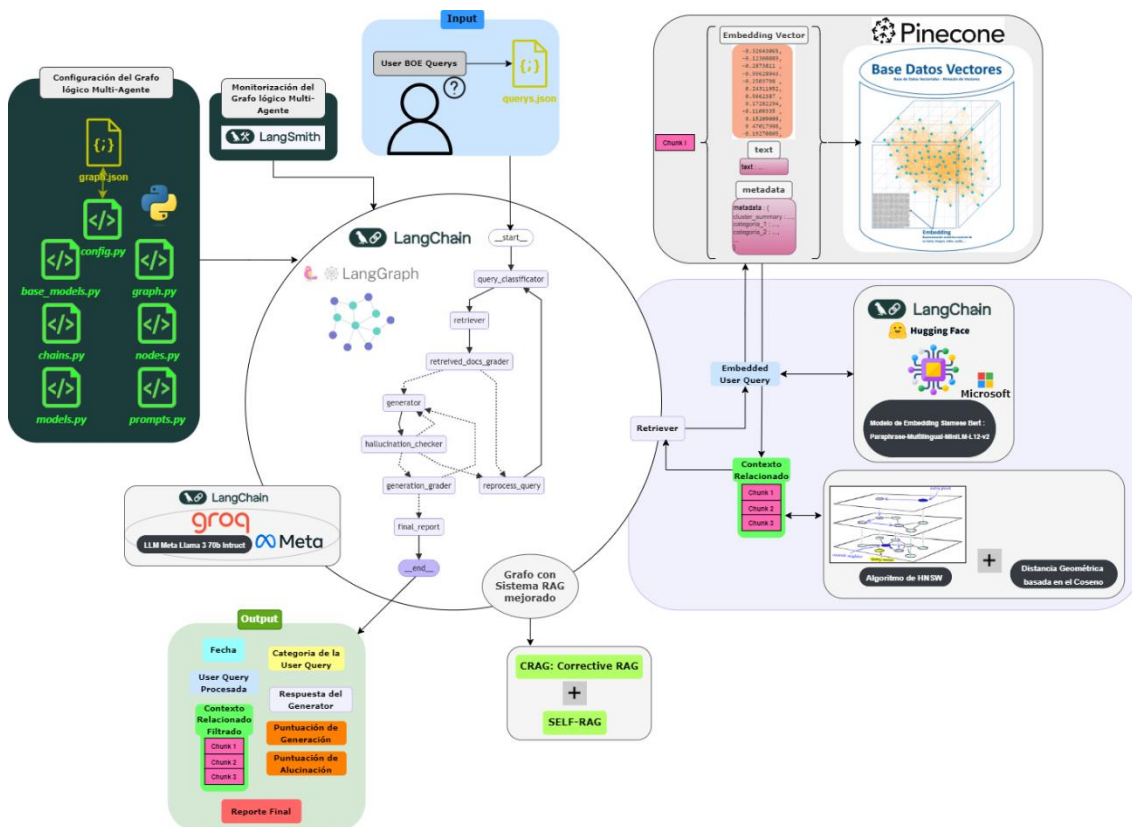


Ilustración 10: Diagrama de la lógica seguida por el grafo multi-agente.

Una vez se tienen los *chunks* clasificados, sus metadatos y su *embeddings* almacenados en una base de datos vectorial como *Pinecone*, el siguiente paso es diseñar la parte de la herramienta encargada de gestionar las *queries* o consultas de los usuarios, obtener el contexto adecuado de la base de datos vectorial y proporcionar una respuesta adecuada y precisa utilizando un LLM. Todo, en un proceso iterativo y autocorrectivo para prevenir fallos de contexto y respuesta. Este tipo de sistemas RAG se basan en grafos lógicos formados por nodos interconectados entre sí. A cada nodo se le asigna una función que recibe un input, por ejemplo, la consulta del usuario, que sirve como entrada al LLM. Después, este LLM es llamado utilizando un *prompt* donde se incrusta el input anterior y el modelo genera una salida estructurada (formato JSON) o no (tipo *string*), y en función de este output, que actualiza el estado del grafo y sus atributos, se inserta el estado del grafo en un nodo u otro según la lógica diseñada.

Por otra parte, y técnicamente hablando, un agente es un modelo LLM con la capacidad de, en función del input recibido, decidir si usar una serie de herramientas o no para resolver la tarea definida. En este trabajo no se han proporcionado herramientas a los LLM, dando más prioridad a la lógica del grafo, a la ingeniería en el diseño de *prompts* y a la conexión entre cada nodo de este.

La función óptima de cada nodo del grafo (es decir, qué se hace en cada nodo o función en cuestión del estado del grafo que recibe) y la lógica del diseño del grafo (es decir, como se interconectan los nodos entre sí), hoy, aún es motivo de investigación por los desarrolladores. Sin embargo, en el presente trabajo, se va a tomar como referencia dos *papers* para diseñar este módulo de la herramienta los cuales mejoran la técnica básica o clásica de un sistema RAG: Self-RAG y Corrective RAG.

CRAG (Corrective RAG)

El método *Corrective Retrieval-Augmented Generation* (CRAG) es una mejora sobre los modelos tradicionales de generación aumentada por recuperación de información (RAG). CRAG aborda las limitaciones de RAG al corregir automáticamente los errores en los documentos recuperados que se utilizan para generar respuestas en un modelo de lenguaje.

CRAG es un marco diseñado para mejorar la robustez de los modelos de generación de lenguaje, como los modelos grandes de lenguaje (LLMs). Aunque los LLMs pueden generar texto fluido y coherente, a menudo sufren de "alucinaciones", es decir, errores factuales que ocurren cuando el modelo confía demasiado en el conocimiento paramétrico incorporado en sus pesos. RAG intenta mitigar esto al proporcionar documentos relevantes recuperados de una base de datos para enriquecer la entrada del modelo. Sin embargo, RAG depende en gran medida de la relevancia y precisión de los documentos recuperados, lo que puede llevar a errores si la recuperación no es adecuada.

CRAG introduce varios mecanismos correctivos para mejorar la precisión de los documentos recuperados antes de que se utilicen en la generación de texto:

- Evaluador de Recuperación: CRAG incorpora un evaluador ligero que revisa la calidad y relevancia de los documentos recuperados. Este evaluador asigna una puntuación de confianza a los documentos recuperados, determinando si son correctos, incorrectos o ambiguos.
- Acciones de Recuperación: Basado en la evaluación, CRAG desencadena diferentes acciones:
 - Correcto: Si al menos un documento recuperado es altamente relevante, se realiza una refinación del conocimiento para extraer la información más crítica.
 - Incorrecto: Si todos los documentos recuperados son irrelevantes, se descartan y se realiza una búsqueda en la web para obtener nuevas fuentes de conocimiento.
 - Ambiguo: Si la evaluación es incierta, se combinan tanto los documentos refinados como la información obtenida de la web para generar una respuesta más robusta.

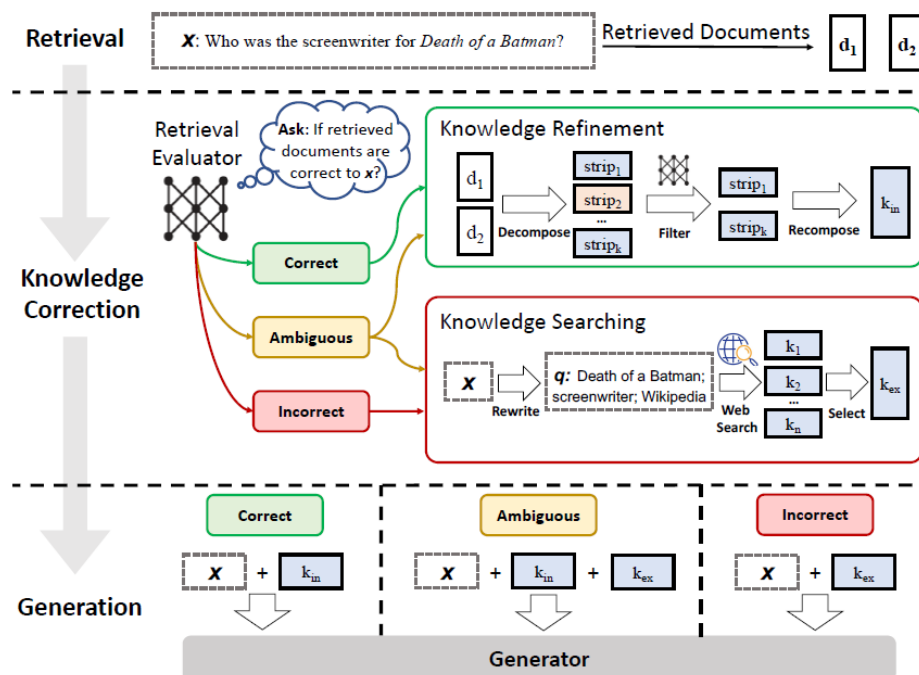


Ilustración 11: CRAG durante la inferencia. Evaluador de recuperación para evaluar la relevancia de los documentos recuperados en relación con la entrada, y estimar un grado de confianza basado en el cual se pueden desencadenar diferentes acciones de recuperación de conocimiento, como {Correcto, Incorrecto, Ambiguo}.

- Refinamiento del Conocimiento: CRAG aplica un proceso de descomposición y recomposición a los documentos relevantes para enfocar la información clave y filtrar los elementos irrelevantes.
- Búsqueda en la Web: Cuando se determina que los documentos recuperados son incorrectos, CRAG realiza búsquedas web a gran escala para complementar la información faltante.

CRAG se puede integrar fácilmente con métodos basados en RAG, como Self-RAG, y ha demostrado mejorar significativamente la precisión y robustez de los modelos de generación en diversas tareas, desde generación de textos cortos hasta largos (Yan, 2024).

Self-RAG

El método Self-RAG (*Self-Reflective Retrieval-Augmented Generation*) es, también, una mejora del enfoque tradicional de RAG (*Retrieval-Augmented Generation*). Este método combina la generación de texto con la recuperación de información relevante y una etapa de autorreflexión crítica.

Self-RAG introduce un proceso de generación de texto que no solo recupera información relevante cuando es necesario, sino que también reflexiona sobre la información recuperada y sobre la propia generación de texto para mejorar la calidad del contenido producido. El modelo realiza tres tareas principales durante su funcionamiento:

- Recuperación bajo demanda: En lugar de recuperar un número fijo de documentos como en los modelos tradicionales de RAG, Self-RAG decide dinámicamente si es necesario recuperar información adicional de la base de datos vectorial. Si el modelo determina que la recuperación es necesaria, se utiliza un modelo de recuperación para obtener los documentos más relevantes.
- Generación concurrente y crítica: Una vez que se han recuperado los documentos, Self-RAG genera el contenido y simultáneamente evalúa la relevancia y la exactitud de la información generada. Esto se hace mediante "tokens de reflexión", que permiten al modelo criticar su propia salida en términos de relevancia, soporte por parte de los documentos recuperados y utilidad general de la respuesta.
- Selección del mejor resultado: Tras generar varias versiones del contenido y evaluarlas, Self-RAG selecciona la mejor opción basada en criterios como la relevancia, el soporte documental y la precisión factual.

El funcionamiento de Self-RAG puede dividirse en los siguientes pasos:

1. Predicción de Recuperación: El modelo predice si es necesario recuperar documentos adicionales para generar una respuesta de calidad.
2. Recuperación de Documentos: Si la predicción anterior indica que se requiere recuperación, se obtienen los documentos más relevantes utilizando un modelo de recuperación.
3. Generación y Crítica: El modelo genera una respuesta basada en los documentos recuperados y, al mismo tiempo, evalúa esta generación utilizando tokens de reflexión para determinar si la información es relevante, está completamente soportada por los documentos, y si la respuesta es útil.
4. Selección de Respuesta: Finalmente, se selecciona la mejor respuesta basada en las evaluaciones críticas realizadas en el paso anterior.

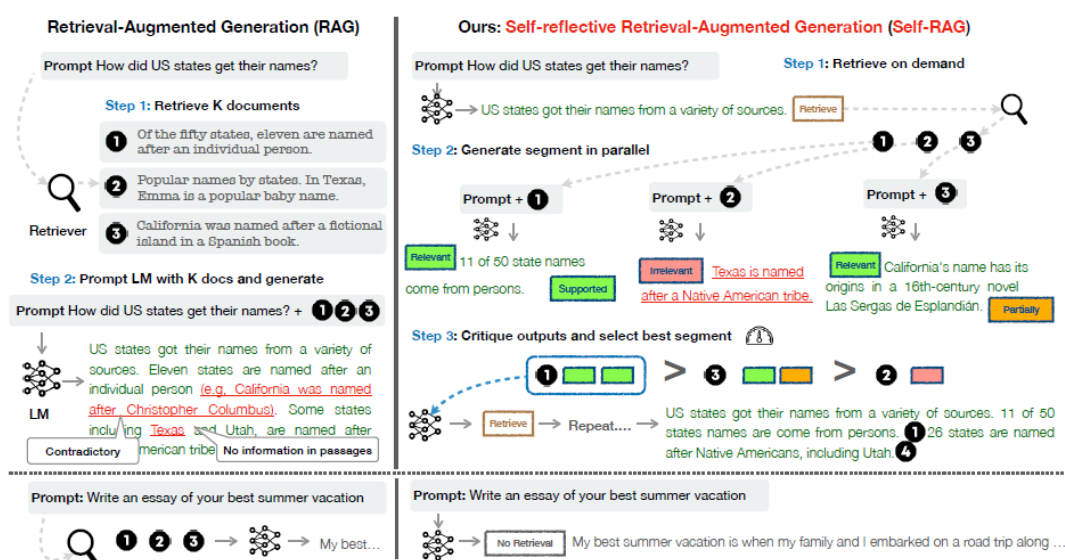


Ilustración 12: Funcionamiento de Self-RAG.

Los resultados empíricos muestran que Self-RAG supera significativamente a los modelos tradicionales de generación aumentada con recuperación y a modelos de lenguaje preentrenados (LLMs) en diversas tareas, incluyendo:

- Mejora en la precisión factual: Self-RAG demuestra una mayor precisión en la generación de contenido basado en hechos, superando a modelos como ChatGPT y Llama2-chat en tareas de verificación de hechos y generación de respuestas basadas en conocimiento abierto.
- Mejor precisión en las citas: En tareas que requieren la generación de texto con citas precisas, Self-RAG muestra una mejora significativa en la precisión y el recuerdo de las citas, superando a otros modelos que utilizan la recuperación de documentos.
- Flexibilidad y personalización: El método permite ajustar la frecuencia de recuperación y la importancia de los tokens de reflexión durante la inferencia, lo que facilita la adaptación del modelo a diferentes tipos de tareas y preferencias del usuario.

Self-RAG es un enfoque avanzado que combina la recuperación dinámica de información con una evaluación crítica del contenido generado, lo que resulta en mejoras significativas en la precisión factual y la calidad general de las respuestas generadas por modelos de lenguaje (Asai, 2023).

Grafo lógico utilizando *LangGraph*

En el módulo GRAPH_RAG de la herramienta, se encuentran los *scripts* de Python que se encargan de definir el grafo lógico, las funciones asociadas a sus nodos, los *prompts* utilizados y que definen a cada agente, la conexión con la API de los diferentes modelos LLM, la creación y compilación del grafo lógico de la biblioteca de *LangChain LangGraph* o la lógica de las conexiones del grafo entre otras características y funciones. A continuación, se explica más en detalle cada uno de ellos.

En este módulo se han implementado dentro de la lógica de las funciones e interconexiones de cada nodo y a través de los *prompts* de cada agente tanto el método Self-RAG como CRAG dentro del grafo lógico de *LangGraph*.

LangGraph es una biblioteca diseñada para construir aplicaciones con modelos de lenguaje (LLMs) que involucran múltiples actores o agentes, permitiendo la creación de flujos de trabajo complejos con agentes individuales o múltiples. A diferencia de otros marcos para LLMs, *LangGraph* se destaca por ofrecer tres beneficios clave: ciclos, controlabilidad y persistencia.

- Ciclos y Ramificaciones: *LangGraph* permite definir flujos de trabajo que incluyen ciclos y condicionales, lo cual es esencial para arquitecturas de agentes avanzados y que lo diferencia de soluciones basadas en grafos acíclicos dirigidos (DAG).
- Controlabilidad y Persistencia: Proporciona un control detallado tanto sobre el flujo como sobre el estado de la aplicación, permitiendo la creación de agentes confiables. Además, incluye persistencia incorporada, lo que significa que el estado de la aplicación se guarda automáticamente después de cada paso, facilitando la recuperación ante

errores, la inclusión de humanos en el proceso de toma de decisiones, y la capacidad de "viajar en el tiempo" en el flujo de trabajo.

- Integración y Soporte para *Streaming*: Aunque está construido por *LangChain* Inc., *LangGraph* puede integrarse fácilmente con *LangChain* y *LangSmith*, pero no es obligatorio usarlos. Además, soporta la transmisión de datos en tiempo real, lo que permite ver los resultados a medida que se generan en cada nodo.

LangGraph se inspira en tecnologías como *Pregel* y *Apache Beam*, y su interfaz pública toma elementos de *NetworkX*, proporcionando un marco de trabajo poderoso y flexible para el desarrollo de aplicaciones avanzadas con LLMs (*LangGraph GitHub Documentation*, 2024).

Script base models.py

En este script se han definido los modelos (son clases como *BaseModel* de la librería *Pydantic*, *TypeDict*, *Dataclasses* o la clase normal de Python) que se van a usar en el grafo con el objetivo de que sigan todos estos una estructura definida, con los mismos atributos y métodos. Esto es necesario y beneficioso para manejar preguntas y respuestas de manera automatizada utilizando un modelo de lenguaje. Además, es muy útil para aplicaciones que requieren seguimiento del contexto y verificación de la calidad de las respuestas.

Este tipo de modelos son comunes en aplicaciones de asistentes virtuales, *chatbots* avanzados, o cualquier otra solución que requiera un manejo automatizado de preguntas y respuestas con integración a bases de datos vectoriales y modelos de lenguaje.

La clase *Question*, que hereda de la clase *BaseModel*, representa la *query* hecha por el usuario.

Tiene tres atributos:

- *id*: Identificador único de la *query*.
- *user_question*: La *query* en sí misma.
- *date*: Fecha en que se realizó la *query*.

Después se define la clase *State* que hereda de la clase *TypeDict* y representa el estado del grafo lógico. Esto significa que es como si se definiese una plantilla para el estado del grafo que es un diccionario de Python, pero con las claves predefinidas y sin valores iniciales, solo con el tipo de dato que cada valor de clave debe almacenar.

Las claves o atributos del estado del grafo son:

- *date*: Fecha del momento que se inicializa un nuevo estado del grafo lógico.
- *question*: *Query* del usuario.
- *query_label*: Etiqueta o categoría del BOE asignada a la *query* del usuario.
- *generation*: Respuesta a la *query* del usuario generada por el modelo LLM tras obtener el contexto del *retriever* de la base de datos vectorial.
- *documents*: Documentos recuperados que pueden ser útiles para responder la pregunta y que aportan el contexto al LLM para responder adecuadamente.

- `fact_based_answer`: Es un tipo de dato *string* que indica si la respuesta o generación del modelo estaba o no basada en hechos o fundamentada en el contexto aportado por el sistema RAG. Tiene dos valores posibles: *yes* o *no*.
- `useful_answer`: Es un tipo de dato *string* que indica si la respuesta es útil o no para resolver la *query* del usuario. Tiene dos valores posibles: *yes* o *no*.
- `report`: Informe final generado tras una ejecución completa y correcta del grafo para una *query* del usuario.

La clase *Agent* representa a cada objeto agente en cada nodo que utiliza un modelo de lenguaje, un *prompt* y un *parser* para llevar a cabo una tarea de generación de texto.

Sus atributos son:

- `agent_name`: Nombre del agente.
- `model`: Nombre del modelo de lenguaje utilizado.
- `get_model`: Función que se conecta a la API del modelo, lo configura y devuelve el modelo como un objeto.
- `temperature`: Parámetro de temperatura del modelo (controla la aleatoriedad de las respuestas).
- `prompt`: Plantilla de *prompt* utilizada para generar las respuestas.
- `parser`: *Parser* que transforma la salida del modelo en el formato deseado.

La clase *VectorDB* maneja la interacción con varias bases de datos vectoriales. Sus atributos son:

- `client`: Cliente de la base de datos (*Pinecone*, *ChromaDB*, *Qdrant*).
- `hg_embedding_model`: Modelo de *embeddings* de *Hugging Face* utilizado por la base de datos vectorial.
- `k`: Número de vecinos más cercanos a recuperar en la consulta a la base de datos.

Su método `get_retriever_vstore` devuelve dos objetos "retriever" y "vector store" basados en el cliente seleccionado (*Pinecone*, *ChromaDB*, *Qdrant*).

Script models.py

Este script de Python está diseñado para configurar modelos de lenguaje natural (LLMs) de diferentes proveedores, incluyendo *OpenAI*, *NVIDIA*, *Ollama*, y *HuggingFace*.

Este script está diseñado para proporcionar una interfaz unificada para acceder a varios modelos de lenguaje y *embeddings*. Cada función crea un modelo específico, lo configura con los parámetros proporcionados en el archivo de configuración del grafo *graph.json*, y lo retorna listo para su uso en la aplicación. Esto es útil en aplicaciones que requieren cambiar dinámicamente entre diferentes modelos de lenguaje o estrategias de *embeddings*, dependiendo de la tarea o del entorno en el que se esté trabajando.

Script prompts.py

Este script de Python está diseñado para crear varias plantillas de *prompts* (instrucciones) que se utilizarán por los modelos de lenguaje natural del grafo, principalmente para la clasificación

de textos, evaluación de documentos y generación de respuestas basadas en el contenido del "Boletín Oficial del Estado" (BOE) de España.

Las plantillas definidas aquí están diseñadas para ser usadas con modelos de NVIDIA y OpenAI:

- `query_classify_prompt`: Se encarga de categorizar un texto del BOE en una de las etiquetas proporcionadas. Si el texto no corresponde a ninguna etiqueta, se clasifica como "Otra".
- `grader_docs_prompt`: Evalúa la relevancia de un documento recuperado en relación con una pregunta del usuario, dando un resultado binario (sí o no).
- `gen_prompt`: Genera una respuesta basada en el contexto proporcionado para responder una pregunta de manera concisa.
- `query_process_prompt`: Reescribe una pregunta del usuario para optimizar su recuperación en una base de datos vectorial.
- `hallucination_prompt`: Evalúa si una respuesta está respaldada por los hechos, proporcionando una puntuación binaria (sí o no).
- `grade_answer_prompt`: Evalúa si una respuesta es útil para resolver una pregunta, también con una puntuación binaria.

En resumen, este script está diseñado para crear instrucciones (*prompts*) personalizadas que guían a diferentes modelos de lenguaje (de NVIDIA y OpenAI) en tareas específicas. Estos prompts después se unirán a la cadena que corresponda dentro de cada nodo.

Script chains.py

En este script se define una única función `get_chain` que es una función diseñada para crear y devolver una cadena (*LLMChain*) en el contexto de la biblioteca *LangChain*.

Los argumentos de esta función y su definición son:

- `prompt_template`: Este parámetro recibe una plantilla de *prompt* que define cómo se estructurará la entrada que se enviará al modelo de lenguaje. Es un texto que contiene variables que se pueden llenar con datos específicos durante la ejecución.
- `Parser`: Un *parser* que se encarga de transformar la salida del modelo de lenguaje en un formato deseado. Este *parser* puede convertir la salida en diferentes formatos, como JSON, texto, etc.
- `get_model`: Este parámetro es una función que devuelve un modelo de lenguaje configurado. Por defecto, está configurado para usar `get_nvdi`, que es una función que carga un modelo específico de Nvidia. Sin embargo, se puede pasar cualquier otra función que devuelva un modelo de lenguaje.
- `temperature`: La temperatura es un parámetro que controla la aleatoriedad en la generación de texto del modelo de lenguaje. Un valor bajo de temperatura (cerca de 0) hace que el modelo sea más determinista y predecible, mientras que un valor alto introduce más variabilidad en las respuestas.

La cadena (*LLMChain*) se construye utilizando el operador de canalización (que es el carácter: `|`), que combina el `prompt_template`, el modelo de lenguaje, y el `parser` de salida en un flujo

continuo. Este operador crea una secuencia de procesamiento donde la entrada pasa por el *prompt*, luego por el modelo, y finalmente por el *parser*, que transforma la salida.

La función *get_chain* se utiliza como argumento de cada función de cada nodo del grafo para obtener su cadena (*LLMChain*). Esto hace que en cada nodo su cadena pueda ser utilizada en la generación de texto, procesamiento de lenguaje natural, o cualquier otra tarea definida por la cadena.

Script nodes.py

En este script de Python se definen todas las funciones que definen los dos elementos clave del grafo lógico de *LangGraph*:

- Nodos.
- Enrutamientos condicionales entre nodos.

Las funciones de los nodos encapsulan la lógica del grafo en un flujo de trabajo y se encargan de tareas específicas dentro de ese flujo. Estas funciones son:

- *query_classifier*: Clasifica la consulta del usuario en una de las etiquetas disponibles del BOE utilizando su modelo, *prompt* y *parser* específico. Esto se logra llamando a una cadena (*LLMchain*) que procesa la consulta y devuelve una etiqueta.
- *retriever*: Recupera documentos relevantes de una base de datos vectorial basándose en la consulta del usuario y la etiqueta generada anteriormente.
- *retrieved_docs_grader*: Evalúa la relevancia de los documentos recuperados en relación con la pregunta del usuario. Si los documentos no son relevantes, se marca para reprocesar la consulta.
- *generator*: Genera una respuesta utilizando un modelo de lenguaje basado en el contenido de los documentos recuperados (técnica conocida como RAG - Retrieval-Augmented Generation).
- *process_query*: Reprocesa o mejora la consulta del usuario para optimizar la recuperación de documentos en futuras búsquedas.
- *hallucination_checker*: Verifica si la respuesta generada está basada en los hechos presentados en los documentos, ayudando a identificar posibles alucinaciones (respuestas que parecen correctas, pero no están basadas en datos reales).
- *generation_grader*: Evalúa la utilidad de la respuesta generada para resolver la pregunta del usuario.
- *final_report*: Compila un informe final con la pregunta, los documentos recuperados, la respuesta generada y las evaluaciones realizadas.

Las funciones de enrutamiento condicional determinan el siguiente paso en el flujo de trabajo basado en el estado actual (objeto *State*) del sistema. Estas son:

- *route_generate_requery*: Decide si se debe generar una respuesta o reprocesar la consulta.
- *route_generate_gradegen_requery*: Decide si se debe calificar la generación, generar una nueva respuesta o reprocesar la *query* del usuario.

- `route_generate_final`: Decide si se debe generar una nueva respuesta o generar el informe final.

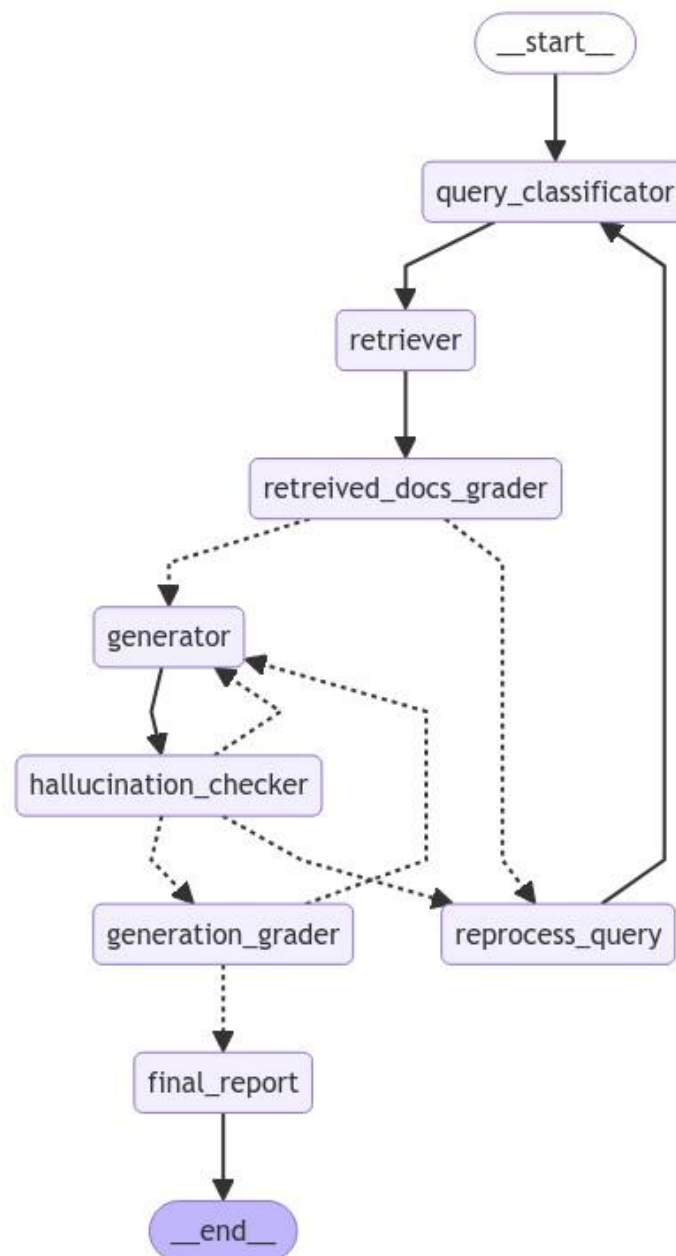


Ilustración 13: Representación del grafo lógico multi-agente creado con LangGraph, sus nodos, sus conexiones y enrutamientos condicionales.

Este script junto con el script de *graph.py* y el archivo de configuración crean y compilan el grafo que se expone en la ilustración anterior y que según se ha comentado con anterioridad es el *core* del sistema RAG mejorado diseñado. Mejorado porque en el grafo y sus nodos se han incorporado las técnicas de CRAG y Self-RAG.

Script graph.py

Este script de Python ha sido diseñado para crear, compilar y guardar el gráfico lógico de flujo de trabajo utilizando la biblioteca *LangGraph*. En este script se importan todas las funciones nodales del script *nodes.py* y, junto con el script de configuración *config.py*, y los archivos de configuración se crea el grafo.

En concreto la función *create_graph* crea un *StateGraph*, que es el objeto que define el gráfico aun sin la capacidad de ejecutarse mediante las funciones nodales y sus conexiones o enrutamientos, también conocidos como aristas (*edges*) que también son funciones como se ha comentado en el script *nodes.py*.

Cada nodo representa una tarea específica, como clasificar una consulta (*query_classifier*), recuperar documentos (*retriever*), o generar un reporte final (*final_report*). Los nodos, como ya se ha mencionado, son funciones que toman el estado actual del flujo de trabajo (*state*) y ejecutan una acción en base a este estado. Después dichas funciones o nodos modifican el estado del grafo o *workflow* y lo retornan.

Las aristas (*Edges*) conectan los nodos, definiendo un enrutamiento en el flujo de ejecución del gráfico. Además de las conexiones directas entre nodos, el gráfico también incluye aristas o enrutamientos condicionales que permiten bifurcaciones en el flujo de trabajo basadas en el resultado de una tarea.

Como se muestra en la ilustración del grafo, se define un nodo de entrada (*query_classifier*) y un nodo de salida (*final_report*), lo que marca el inicio y el fin del flujo de trabajo.

La función *compile_graph* compila el gráfico creado utilizando un objeto *MemorySaver* para guardar el estado del gráfico. La compilación prepara el gráfico estático creado para su ejecución, devuelve un *CompiledGraph*, que es la versión compilada del gráfico lista para ejecutarse.

Script config.py

Este script de Python ha sido diseñado para integrar en un mismo objeto todos los modelos LLM, *prompts* y configuraciones de estos modelos para todos los agentes del grafo lógico de *LangGraph*. Por otra parte, también gestiona el cliente de la base de datos vectorial que se va a utilizar y su configuración (modelo de *embedding*, k vecinos más cercanos...). El lugar donde el usuario de la herramienta define cada una de estas configuraciones es en los archivos JSON dentro de la carpeta *config* y subcarpeta *graph* del proyecto. Estos archivos se explican después.

La clase que se ha diseñado en este script, *ConfigGraph*, maneja la configuración del gráfico de procesamiento de datos. Como se ha adelantado, define los modelos, los agentes, y las bases de datos vectoriales (vector DBs) que se utilizarán en el flujo de trabajo mediante variables de clase que luego el usuario seleccionara mediante los archivos de configuración. Después de instancia esta clase, se cargan esta configuración desde un archivo JSON y se cargan las *queries* que el usuario quiere hacer al grafo, también, mediante otro archivo JSON: *queries.json*.

Posterior a esto y con la configuración cargada se crean los agentes y bases de datos vectoriales basados en la configuración cargada.

En resumen y por matizar, en el diseño de este script se ha buscado aunar en un mismo objeto toda la configuración personalizable de la herramienta encargada de dar respuesta a las *queries* de un usuario, pudiendo después ejecutar el grafo asegurando que cada parte del proceso esté bien definida y controlada.

Archivo de configuración y de consultas

El archivo de configuración *graph.json* configura seis agentes, todos utilizando el modelo de lenguaje "OPENAI" con una temperatura de 0, lo que indica que el modelo se ejecutará con alta determinación y menor variabilidad en las respuestas.

Los agentes definidos son:

- `query_classifier`: Clasifica las consultas de los usuarios en categorías predefinidas.
- `docs_grader`: Evalúa la relevancia de los documentos recuperados en relación con la consulta del usuario.
- `query_processor`: Procesa o reformula la consulta del usuario para mejorar la recuperación de documentos.
- `generator`: Genera respuestas basadas en los documentos recuperados y la consulta del usuario.
- `hallucination_grader`: Evalúa si la respuesta generada está basada en hechos, es decir, si está bien fundamentada en los documentos.
- `answer_grader`: Evalúa si la respuesta es útil y adecuada para resolver la consulta del usuario.

También se incluye en el archivo la configuración de la base de datos vectorial que se utilizará para recuperar documentos relevantes en función de las consultas del usuario.

Los parámetros que se configuran son:

- `client`: Se especifica *pinecone* como el cliente de base de datos vectorial, que es un servicio para búsqueda de similitudes de vectores.
- `hg_embedding_model`: Especifica el modelo de *embeddings* "*sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2*", que se utilizará para convertir texto en representaciones vectoriales.
- `k`: Define el número de vecinos más cercanos ($k=3$) que se recuperarán de la base de datos vectorial durante las consultas.

Y, por último, el archivo permite configurar otros parámetros relacionados con el grafo lógico:

- `iteraciones`: Define el número de iteraciones que se ejecutará en el grafo lógico, en este caso, 10 iteraciones.
- `thread_id`: Especifica un identificador para el hilo de ejecución, que aquí es "4".
- `verbose`: Define el nivel de detalle en los registros de ejecución, donde 0 indica un nivel bajo de verbosidad.

```
{
  "agents":{
    "query_classifier":{
      "name":"GROQ",
      "temperature":0
    },
    "docs_grader": {
      "name":"GROQ",
      "temperature":0
    },
    "query_processor": {
      "name":"GROQ",
      "temperature":0
    },
    "generator": {
      "name":"GROQ",
      "temperature":0
    },
    "hallucination_grader": {
      "name":"GROQ",
      "temperature":0
    },
    "answer_grader": {
      "name":"GROQ",
      "temperature":0
    }
  },
  "vector_db": {
```



```
"client": "pinecone",  
  "hg_embedding_model": "sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2",  
  "k": 3  
,  
  "iteraciones": 100,  
  "thread_id": "4",  
  "verbose": 0  
}
```

Tabla 4: Archivo de configuración del grafo: *graph.json*.

El archivo de consultas incluye la fecha de la pregunta del usuario, la *query* o consulta (*user_question*) en sí que se quiere realizar al grafo, el *boe_id* que identifica el archivo PDF de donde procede la *query* y donde se encuentra la respuesta y, con objetivos de evaluación, la respuesta que se espera del grafo o *ground_truth*.

```
[  
  {  
    "user_question": "¿Cuál es la duración total de las enseñanzas del ciclo de grado medio para Técnico Deportivo en Atletismo?",  
    "ground_truth": "La duración total es de 1.005 horas, organizadas en dos ciclos.",  
    "date": "2024-04-15",  
    "boe_id": "BOE-A-2024-7296"  
  },  
  {  
    "user_question": "¿Qué pruebas son necesarias para acceder al ciclo inicial de grado medio en atletismo?",  
    "ground_truth": "Es necesario superar la prueba de carácter específico RAE-ATAT101 descrita en el anexo VI del Real Decreto 427/2023.",  
    "date": "2024-04-15",  
    "boe_id": "BOE-A-2024-7296"  
  }  
]
```

--

Tabla 5: Archivo de queries o consultas: *queries.json*.

Script main graph.py

Por último, dentro de este módulo (aunque al mismo nivel que la carpeta *src*) está este script, el cual es el que se debe ejecutar para lanzar la aplicación y con ello ejecutar el grafo para que resuelva la consulta definida en los archivos por parte del usuario.

Primero se cargan las variables de entorno desde un archivo *.env*. Estas variables incluyen claves API necesarias para interactuar con diferentes servicios como *LangChain*, *Pinecone*, *OpenAI*, etc.

Después, se leen los parámetros del script o si estos parámetros no se especifican, el script intenta obtenerlos desde variables de entorno.

Acto seguido se instancia la clase *ConfigGraph*, que se importa del script *graph.py*, con las rutas a los archivos de configuración y datos. Esta clase configura el grafo lógico y los agentes que lo compondrán.

Se añaden como atributos de la clase *RunnableConfig* los parámetros de ejecución del grafo, incluyendo el número de iteraciones y el ID del hilo de ejecución. Después, guarda el grafo compilado como una imagen en local.

Una vez se tiene el grafo preparado para su ejecución, itera sobre la lista de *queries* del usuario definidas en la configuración *queries.json*. Para cada pregunta, construye un diccionario de entradas (*inputs*) y lo pasa al grafo compilado ejecutándolo mediante el método *invoke*. Durante la ejecución se observan por la consola las respuestas de cada uno de los agentes y como interactúan con la *query* del usuario.

Tras procesar el grafo las *queries*, se crea un conjunto de datos de prueba (*RagasDataset*) con las preguntas del usuario, las respuestas generadas por el grafo, los contextos retonados por el sistema RAG (los fragmentos de texto o *chunks*) y las respuestas esperadas o *ground truth*.

Una vez se tiene este conjunto de datos que ha proporcionado el grafo y que sirve para medir su rendimiento, se convierte en un formato compatible con *Hugging Face* mediante el método *to_dataset* que transforma los atributos del objeto *RagasDataset* en un diccionario de Python y después, este diccionario, en un objeto *Dataset*.

Por último, se publica este *dataset* en el repositorio de *Hugging Face Hub* usando el método *push_to_hub* de la clase *RagasDataset*.

4.2.5 Evaluación del sistema RAG con la librería RAGAS

Ragas es un marco de trabajo que ayuda a evaluar *pipelines* o aplicaciones basadas en un sistema de *Retrieval Augmented Generation* (RAG) como la diseñada en el presente trabajo. Como definen los desarrolladores de esta librería, RAG se refiere a una clase de aplicaciones de modelos de lenguaje que utilizan datos externos para mejorar el contexto del modelo. Existen

herramientas y marcos que ayudan a construir estos pipelines como *LangChain*, pero evaluarlos y cuantificar su rendimiento puede ser complicado. Aquí es donde entra esta librería para la evaluación de RAG (RAGAS, s.f.).

En el presente trabajo se ha desarrollado un nuevo módulo de la herramienta o la aplicación que se integra con el módulo anterior del grafo lógico y, si se desea, se puede utilizar para evaluar, mediante diferentes métricas, el sistema RAG mejorado diseñado.

Este módulo llamado RAG_EVAL está formado por una serie de *scripts* de Python cuya funcionalidad se describe a continuación.

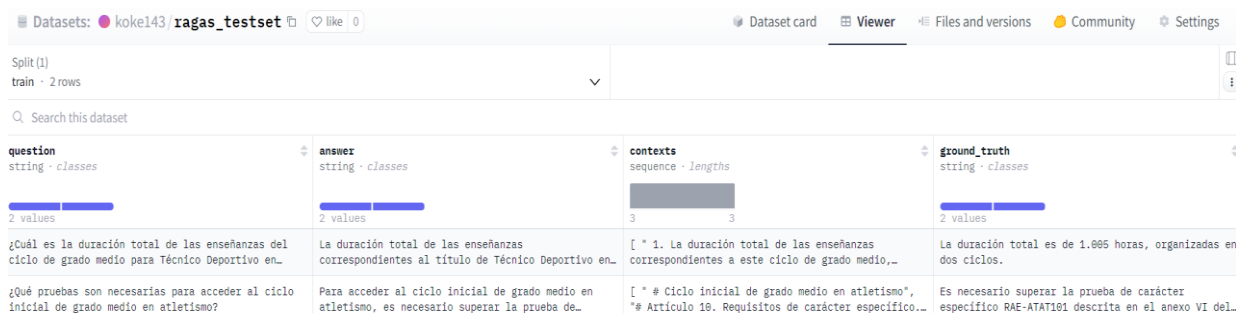
Script base model.py

Este script de Python define una clase llamada *RagasDataset* que está diseñada para manejar un conjunto de datos utilizado en la evaluación de un sistema RAG.

La clase está diseñada para crear objetos que contengan como atributos:

- *Queries* o preguntas del usuario relacionadas con el BOE.
- Respuestas del sistema RAG a cada una de las preguntas que, en el presente trabajo, se proporcionan como atributo del *State* del grafo de agentes tras las iteraciones internas de este.
- Los contextos utilizados por el agente o modelo LLM generador y recuperados de la base de datos vectorial de *Pinecone* donde se almacenan los fragmentos o *chunks* de texto semánticamente semejantes del BOE.
- *Ground_truth* o respuestas correctas: Son las respuestas que se espera que proporcione el sistema RAG para cada pregunta y contexto correcto. Esta serie de respuestas esperadas se pueden crear de varias formas, en este caso, se han creado de manera manual.

La clase contiene un método que, una vez obtenidos los anteriores atributos para una serie de preguntas tras ejecutar el grafo, convierte este conjunto de datos a un objeto *Dataset* del módulo *datasets* de *Hugging Face*. Este objeto es adecuado para ser manipulado, almacenado en *Hugging Face Hub* y recuperado después para aplicar las métricas de evaluación de la librería RAGAS a la herramienta o aplicación.



question	answer	contexts	ground_truth
¿Cuál es la duración total de las enseñanzas del ciclo de grado medio para Técnico Deportivo en...	La duración total de las enseñanzas correspondientes al título de Técnico Deportivo en...	[" 1. La duración total de las enseñanzas correspondientes a este ciclo de grado medio,...	La duración total es de 1.005 horas, organizadas en dos ciclos.
¿Qué pruebas son necesarias para acceder al ciclo inicial de grado medio en atletismo?	Para acceder al ciclo inicial de grado medio en atletismo, es necesario superar la prueba de...	[" # Ciclo inicial de grado medio en atletismo", " # Artículo 10. Requisitos de carácter específico...	Es necesario superar la prueba de carácter específico RAE-ATAT101 descrita en el anexo VI del...

Ilustración 14: Repositorio en Hugging Face Hub donde se encuentra subido el dataset para ser recuperado y utilizado posteriormente para la evaluación del sistema RAG.

Script testset.py

Este script de Python es el que se encarga en sí de la evaluación del sistema RAG diseñado gracias a la recuperación del *testset* creado y subido al repositorio remoto de *Hugging Face Hub*.

En él se definen, entre otras clases, una llamada *RagasEval* que está diseñada para cargar, evaluar y guardar los resultados de este conjunto de datos de evaluación o *testset* de modelos de Recuperación y Generación Automática de Respuestas (RAG).

Esta clase está diseñada para una vez cargado o recuperado el, ejecutar una evaluación usando métricas específicas de la librería RAGAS, guardar los resultados en un archivo CSV y generar un informe gráfico con estos resultados.

El flujo típico de uso de la clase *RagasEval* sería crear una instancia de *RagasEval*, proporcionando el token de autenticación de *Hugging Face* y el nombre del *testset* en el repositorio del conjunto de datos que se quiere evaluar. Después, ejecutar el método *run* de la clase, pasando la ruta donde se quieren almacenar los resultados de la evaluación e indicando si se quiere generar un reporte grafico con los resultados de las métricas.

El script cargará el conjunto de datos, ejecutará la evaluación con las métricas definidas, y luego guardará los resultados y las gráficas en la ubicación especificada.

Métricas de evaluación

Las métricas utilizadas de la librería RAGAS para la evaluación del sistema RAG diseñado son:

- *Answer relevancy*
- *Faithfulness*
- *Context recall*
- *Context precision*

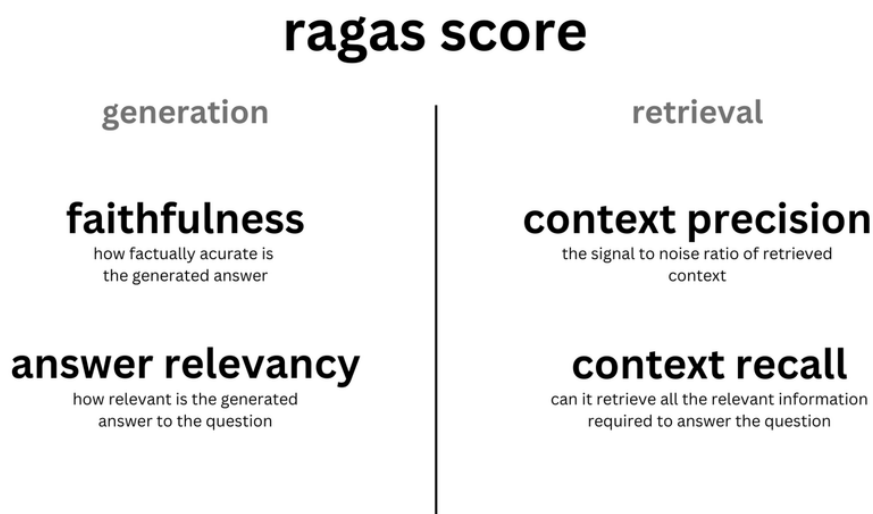


Ilustración 15: Métricas de evaluación de la librería RAGAS.

Faithfulness o Fidelidad

Esta métrica mide la consistencia factual de la respuesta generada por el sistema RAG, y en concreto por el agente generador, en comparación con el contexto dado. Se calcula a partir de la respuesta y el contexto recuperado. La respuesta se escala en un rango de (0,1). Cuanto más alto, mejor (Ragas, 2024).

La respuesta generada se considera fiel si todas las afirmaciones hechas en la respuesta pueden inferirse del contexto dado. Para calcular esto, primero se identifica un conjunto de afirmaciones de la respuesta generada. Luego, cada una de estas afirmaciones se verifica con el contexto dado para determinar si puede inferirse del contexto. La puntuación de fidelidad se obtiene de la siguiente manera:

$$\text{Fidelidad} = \frac{|\text{Número de afirmaciones en la respuesta generada que pueden deducirse del contexto dado}|}{|\text{Número total de afirmaciones en el contexto dado}|}$$

Un ejemplo de uso y cálculo de esta métrica se expone a continuación para mayor comprensión.

- Pregunta: ¿Dónde y cuándo nació Einstein?
- Contexto: Albert Einstein (nacido el 14 de marzo de 1879) fue un físico teórico nacido en Alemania, considerado ampliamente como uno de los científicos más grandes e influyentes de todos los tiempos.
- 1. Ejemplo de una respuesta de alta fidelidad: Einstein nació en Alemania el 14 de marzo de 1879.
- 2. Ejemplo de una respuesta de baja fidelidad: Einstein nació en Alemania el 20 de marzo de 1879.

Cómo se calcularía la fidelidad utilizando la respuesta de baja fidelidad:

- Paso 1: Divide la respuesta generada en declaraciones individuales.
 - Declaración 1: "Einstein nació en Alemania."
 - Declaración 2: "Einstein nació el 20 de marzo de 1879."
- Paso 2: Para cada una de las declaraciones generadas, verifica si puede inferirse del contexto dado.
 - Declaración 1: Sí
 - Declaración 2: No
- Paso 3: Utiliza la fórmula descrita anteriormente para calcular la fidelidad:
-

$$\text{Fidelidad} = \frac{1}{2}$$

Answer relevancy o relevancia de respuesta

La métrica de evaluación, Relevancia de la Respuesta, se centra en evaluar cuán pertinente es la respuesta generada en relación con la pregunta dada. Se asigna una puntuación más baja a las respuestas que son incompletas o contienen información redundante, y puntuaciones más altas indican mejor relevancia. Esta métrica se calcula utilizando la pregunta, el contexto y la respuesta (Ragas, 2024).

La Relevancia de la Respuesta se define como la media de la similitud coseno entre la pregunta original y un número de preguntas artificiales, que fueron generadas (ingeniería inversa) basadas en la respuesta:

$$\text{answer relevancy} = \frac{1}{N} \sum_{i=1}^N \cos(E_{g_i}, E_o)$$
$$\text{answer relevancy} = \frac{1}{N} \sum_{i=1}^N \frac{E_{g_i} \cdot E_o}{\|E_{g_i}\| \|E_o\|}$$

Ilustración 16: Expresión de la métrica de relevancia de la respuesta.

Dónde:

- E_{g_i} : es el *embedding* o vector denso de la pregunta generada cuya dimensión dependerá del modelo de *embedding* utilizado (en el presente trabajo serán vectores de 384 dimensiones).
- E_o : es el *embedding* o vector denso de la pregunta original.
- N : es el número de preguntas generadas, que por defecto es 3.

Es importante destacar que, aunque en la práctica la puntuación generalmente oscilará entre 0 y 1, esto no está matemáticamente garantizado debido a que la similitud coseno puede variar entre -1 y 1.

Esta es una métrica independiente de la referencia. Si se está buscando comparar la respuesta correcta (*ground truth*) con la respuesta generada, se deben usar otras métricas como la de corrección de la respuesta.

Una respuesta se considera relevante cuando aborda de manera directa y adecuada la pregunta original. Es importante señalar que la evaluación de la relevancia de la respuesta, por parte de la librería RAGAS, no considera la veracidad, sino que penaliza los casos en los que la respuesta carece de completitud o contiene detalles redundantes. Para calcular esta puntuación, se solicita al modelo de lenguaje que genere una pregunta adecuada para la respuesta generada varias veces, y se mide la similitud coseno media entre estas preguntas generadas y la pregunta original. La idea subyacente es que, si la respuesta generada aborda con precisión la pregunta inicial, el modelo de lenguaje debería ser capaz de generar preguntas a partir de la respuesta que se alineen con la pregunta original.

A continuación, se expone un ejemplo de aplicación de esta métrica.

- Pregunta: ¿Dónde está Francia y cuál es su capital?
- Respuesta de baja relevancia: Francia está en Europa occidental.
- Respuesta de alta relevancia: Francia está en Europa occidental y París es su capital.

Para calcular la relevancia de la respuesta respecto a la pregunta dada, se siguen dos pasos:

- Paso 1: la librería crea de forma inversa 'n' variantes de la pregunta a partir de la respuesta generada, utilizando un modelo de lenguaje grande (LLM). Por ejemplo, para la primera respuesta de baja relevancia, el LLM podría generar las siguientes posibles preguntas:
 - Pregunta 1: "¿En qué parte de Europa se encuentra Francia?"
 - Pregunta 2: "¿Cuál es la ubicación geográfica de Francia dentro de Europa?"
 - Pregunta 3: "¿Puedes identificar la región de Europa donde está situada Francia?"
- Paso 2: Se calcula la similitud coseno media entre las preguntas generadas y la pregunta original.

El concepto subyacente es que, si la respuesta aborda correctamente la pregunta, es muy probable que la pregunta original pueda ser reconstruida únicamente a partir de la respuesta.

Context precision o precisión en el contexto

La Precisión de contexto es una métrica que evalúa si todos los elementos relevantes de la verdad fundamental o *ground truth* presentes en los contextos están clasificados en posiciones superiores o no. Idealmente, todos los fragmentos o *chunks* relevantes deben aparecer en los primeros lugares del ranking. Esta métrica se calcula utilizando la pregunta, la verdad fundamental y los contextos, con valores que oscilan entre 0 y 1, donde puntuaciones más altas indican mejor precisión (Ragas, 2024).

$$\text{Context Precision@K} = \frac{\sum_{k=1}^K (\text{Precision@k} \times v_k)}{\text{Total number of relevant items in the top } K \text{ results}}$$

$$\text{Precision@k} = \frac{\text{true positives@k}}{(\text{true positives@k} + \text{false positives@k})}$$

Ilustración 17: Expresiones de la métrica de precisión del contexto.

Donde:

- K: es el número total de *chunks* o fragmentos de texto en el contexto.
- V_k: es el indicador de relevancia en el rango k. Su valor esta acotado entre 0 y 1.

Un ejemplo sencillo para esta métrica sería:

- Pregunta: ¿Dónde está Francia y cuál es su capital?
- Verdad fundamental o *ground truth*: Francia está en Europa Occidental y su capital es París.
- Ejemplo de una alta precisión de contexto (dos fragmentos de texto o *chunks*):
 - "Francia, en Europa Occidental, abarca ciudades medievales, pueblos alpinos y playas mediterráneas. París, su capital, es famosa por sus casas de moda, museos de arte clásico como el Louvre y monumentos como la Torre Eiffel"
 - "El país también es conocido por sus vinos y cocina sofisticada. Los antiguos dibujos rupestres de Lascaux, el teatro romano de Lyon y el vasto Palacio de Versalles son testimonio de su rica historia."
- Ejemplo de una baja precisión de contexto (dos fragmentos de texto o *chunks*):
 - "El país también es conocido por sus vinos y cocina sofisticada. Los antiguos dibujos rupestres de Lascaux, el teatro romano de Lyon y"
 - "Francia, en Europa Occidental, abarca ciudades medievales, pueblos alpinos y playas mediterráneas. París, su capital, es famosa por sus casas de moda, museos de arte clásico como el Louvre y monumentos como la Torre Eiffel."

Un ejemplo del cálculo de la precisión de contexto utilizando el ejemplo de baja precisión de contexto:

- Paso 1: Para cada fragmento en el contexto recuperado, se verifica si es relevante o no para la verdad fundamental dada la pregunta.
- Paso 2: Se calcula la precisión@k para cada fragmento en el contexto.

$$\text{Precision@1} = \frac{0}{1} = 0$$

$$\text{Precision@2} = \frac{1}{2} = 0.5$$

Ilustración 18: Cálculo de precisión para cada chunk del contexto.

- Paso 3: se calcula la media de precisión@k para obtener la puntuación final de precisión de contexto.

$$\text{Context Precision} = \frac{(0+0.5)}{1} = 0.5$$

Ilustración 19: Cálculo de la métrica de precisión del contexto.

Context recall o recuperación de contexto

Esta métrica de recuperación de contexto mide en qué medida el contexto recuperado se alinea con la respuesta anotada, tratada como la verdad fundamental. Se calcula utilizando la pregunta, la verdad fundamental o *ground truth* y el contexto recuperado, y los valores oscilan entre 0 y 1, con valores más altos que indican un mejor rendimiento. Para estimar la recuperación de contexto a partir de la respuesta de la verdad fundamental, se analiza cada afirmación en la respuesta de la verdad fundamental para determinar si puede atribuirse al contexto recuperado o no. En un escenario ideal, todas las afirmaciones en la respuesta de la verdad fundamental deberían ser atribuibles al contexto recuperado (Ragas, 2024).

La fórmula para el cálculo de la recuperación de contexto es:

$$\text{Context recall} = \frac{|\text{Número de afirmaciones en la verdad fundamental que pueden atribuirse al contexto}|}{|\text{Número total de afirmaciones en la verdad fundamental}|}$$

Un ejemplo para una mayor comprensión del funcionamiento esta métrica es:

- Pregunta: ¿Dónde está Francia y cuál es su capital?
- Verdad fundamental: Francia está en Europa Occidental y su capital es París.
- Alta recuperación de contexto: Francia, en Europa Occidental, abarca ciudades medievales, pueblos alpinos y playas mediterráneas. París, su capital, es famosa por sus casas de moda, museos de arte clásico como el Louvre y monumentos como la Torre Eiffel.
- Baja recuperación de contexto: Francia, en Europa Occidental, abarca ciudades medievales, pueblos alpinos y playas mediterráneas. El país también es conocido por

sus vinos y cocina sofisticada. Los antiguos dibujos rupestres de Lascaux, el teatro romano de Lyon y el vasto Palacio de Versalles son testimonio de su rica historia.

Ahora se muestra cómo la librería calculó la recuperación de contexto utilizando el ejemplo de baja recuperación de contexto:

- Paso 1: Divide la respuesta de la verdad fundamental en declaraciones individuales.
 - Declaración 1: "Francia está en Europa Occidental."
 - Declaración 2: "Su capital es París."
- Paso 2: Para cada una de las declaraciones de la verdad fundamental, verifica si puede atribuirse al contexto recuperado.
 - Declaración 1: Sí
 - Declaración 2: No
- Paso 3: Usa la fórmula descrita anteriormente para calcular la recuperación de contexto:

$$\text{context recall} = \frac{1}{2} = 0.5$$

Ilustración 20: Cálculo de context recall.

4.3 Recursos requeridos

La aplicación ha sido desarrollada con Python y un gran número de sus librerías o paquetes o módulos relacionados con la inteligencia artificial, los grandes modelos de lenguaje, *el machine learning*, técnicas del procesamiento del lenguaje natura (NLP)...

En concreto, se han utilizado *langchain* y *llamaindex* que son *frameworks* para construir componentes y aplicaciones basadas en LLMs interoperables. Se ha utilizado *LangGraph* que es el *framework* que ha permitido la construcción de flujos de trabajo o grafos lógicos controlables formados por agentes. Se ha utilizado *langsmith*, otra librería o *framework*, que permite el despliegue, el monitoreo, la colaboración, y el *testing* de la aplicación LLM, tanto si está construida con *LangChain* como si no.

También se ha hecho uso de librerías de *Huggingface* como *Transformers* y *datasets* las cuales proporcionan APIs y herramientas para descargar y entrenar fácilmente modelos pre-entrenados de última generación en tareas como:

- Procesamiento del Lenguaje Natural
- Visión por ordenador
- Audio
- Multimodales

Se han utilizado también librerías como *NumPy*, *Pandas*, *Scikit-learn*, otras de visualización como *Matplotlib* y *Seaborn*.

En el módulo de nlp de la herramienta se han utilizado librerías de procesamiento del lenguaje natura como NLTK y la librería RE que permite el uso de expresiones regulares para localización de ciertos patrones en los textos del BOE.

Para el ajuste fino del modelo DebertaV3 se han utilizado librerías de ML y *Deep learning* como *Pytorch*, *TensorFlow* y *Transformers*.

A nivel de *debugging* o monitoreo de la aplicación se han utilizado librerías para el control de *logs* o registros como *Logging*.

A nivel de aplicación, se han utilizado librerías como *Streamlit* un *framework* de Python que permite la creación del *front-end* de la aplicación web con Python nativo.

También, se han utilizado librerías como *Typing*, para definir el tipo de dato de los objetos y evitar errores de la aplicación, y *Pydantic*, que es la librería de validación de datos más utilizada para Python.

Para el módulo de descarga se ha utilizado *Requests* que es una librería HTTP de Python que permite enviar peticiones HTTP de forma sencilla.

Para la configuración de la aplicación se ha utilizado *Python-dotenv* que lee pares clave-valor de un archivo “.env” y puede establecerlos como variables de entorno. En este archivo se guardan los API token de todas las tecnologías utilizadas, así como nombres de repositorios remotos, nombres de las *collections* e *index* de las bases de datos vectoriales, nombres de modelos de *embeddings* y nombres de los modelos LLMs utilizados, así como otras variables necesarias para la configuración de la aplicación. Esta librería *dotenv* ayuda en el desarrollo de la aplicación siguiendo los principios de los 12 factores.

Para el módulo de resultados y evaluación, se ha utilizado la librería RAGAS que permite la aplicación de métricas específicas que miden el rendimiento de un sistema o un pipeline de RAG.

En lo que respecta al desarrollo de código de manera genérica, se ha hecho uso de un entorno virtual gracias *venv* de Python. Este módulo *venv* permite crear entornos virtuales ligeros, cada uno con su propio conjunto independiente de paquetes Python instalados en sus directorios de sitio.

Para el control de versiones de la aplicación se ha usado *Git* y *GitHub*. *Git* es un sistema de control de versiones distribuido, gratuito y de código abierto y GitHub es una plataforma donde puedes almacenar, compartir y trabajar junto con otros usuarios para escribir código. Almacenar el código en un repositorio en GitHub permite presentar, compartir el trabajo, y administrar los cambios en el código a lo largo del tiempo. Todo el código, por ello, se ha ido subiendo a un repositorio de *GitHub* público con el objetivo de que el trabajo desarrollado sea utilizado por la comunidad dado que no existe, según la investigación realizada, ningún trabajo de esta índole a nivel nacional, al menos *open source*. Es decir, una aplicación que se enfoque en el procesamiento de textos legales gubernamentales españoles mediante LLMs y NLP y esté disponible de manera pública.

Otras herramientas utilizadas han sido los servicios de *OpenAI*, en concreto vía API, para la clasificación automatizada de fragmentos de texto del BOE.

También se ha hecho uso, mediante el *wrapper* de *langchain*, de *Pinecone* es una plataforma de gestión de bases de datos vectoriales diseñada específicamente para trabajar con modelos de *machine learning* y búsqueda semántica. Destacando por su conocido uso en aplicaciones de IA. Se usa en casos de uso avanzados como búsqueda semántica, *chatbots*, recomendadores, sistemas de clasificación, entre otros.

Y, por último, los recursos técnicos y la documentación en su mayoría han sido obtenidos de los sitios web oficiales de estas tecnologías, del sitio oficial de Arxiv, donde publican *papers* de diferentes indoles, de las referencias citadas en su respectivo apartado y de la página oficial del gobierno de España para la obtención de los PDF del BOE.

4.4 Presupuesto

Tipo de coste	Valor	Comentarios
Horas de trabajo en el proyecto	500 horas aproximadamente y según el cronograma	Tiempo invertido en el desarrollo, pruebas, y ajuste del proyecto.
Equipo técnico utilizado	1700 [€]	Desktop/Laptop: Equipo Workstation Portátil HP ZBook Firefly G10 con las especificaciones necesarias para el desarrollo (CPU potente, GPU adecuada para entrenamiento de modelos).
Servidor en la nube	13[€] / mes	Google Colab Pro: Fine-Tuning DebertaV3
Software utilizado	12 [€]	Python: 0€ (software de código abierto). VSCode IDE: 0€ (Editor de Código). LlamaParse: 0 € (licencia gratuita). Utilización del <i>parser</i> en la nube. GroqCloud: 0 € Para inferencia usando el modelo de meta llama3 de 70

		<p>billones de parámetros y una <i>context window</i> de 8,192 tokens.</p> <p>Huggingface Transformers: 0€ (licencia gratuita).</p> <p>GitHub: 0€ (plan gratuito para repositorios públicos).</p> <p>OpenAI API: Coste de 12 € por la utilización de modelos vía API como: GPT-4o mini, GPT-3.5 Turbo y GPT-4o</p>
Estudios e informes	0 [€]	<p><i>Papers</i> de <i>Arxiv</i>, documentación de tecnologías y archivos del BOE</p>
Librerías y <i>frameworks</i> adicionales	0[€]	<p>Todas las librerías utilizadas (LangChain, Pydantic, LangSmith, Streamlit, PyTorch, NumPy, etc.) son de uso libre y gratuito.</p>
Espacio de almacenamiento	2[€] / mes	<p>Coste aproximado de almacenamiento en la nube (Google Drive) 2[€] / mes.</p> <p><i>Pinecone</i> como base de datos vectorial para el almacenamiento de los fragmentos de texto o <i>chunks</i> del BOE, sus metadatos y sus <i>embeddings</i> en una <i>collection</i> de <i>pinecone</i> (servicio gratuito hasta 2GB de almacenamiento).</p>

Tabla 6: Presupuesto para el proyecto.

4.5 Resultados del proyecto

Conforme a los objetivos fijados para el presente proyecto los resultados han sido positivos.

En primer lugar, conforme al primer objetivo, se ha logrado diseñar una herramienta modularizada que constituye una aplicación basada en inteligencia artificial para el procesamiento de los archivos o documentos del boletín oficial del estado español. Permitiendo a los usuarios no expertos en textos técnicos o legales obtener información de esta índole y de estos archivos mediante preguntas en lenguaje natural utilizando LLMs.

En segundo lugar, la gran ventaja de esta herramienta es que es altamente escalable y mejorable. Se pueden variar, por ejemplo, en los archivos de configuración del módulo ETL, las diferentes técnicas de NLP que incorpora la herramienta buscando, mediante la experimentación, mejorar el procesamiento de los textos del BOE. También, en un futuro, se puede escalar la interacción actual usuario-herramienta para crear una aplicación web o un SaaS, por ejemplo, con UI constituyendo una aplicación web basada en IA y NLP para procesar el BOE.

Conforme al objetivo fijado inicialmente de realizar el ajuste fino de un modelo *transformer-encoder* tipo BERT para la clasificación “multi-etiqueta” de fragmentos semánticos del BOE, los resultados son más negativos. Realizar el ajuste fino de estos grandes modelos con tantos parámetros los cuales ocupan un gran espacio en memoria y, aunque, solo se actualicen durante en ajuste fino los correspondientes a la última capa, es complejo. Se necesitan muchos recursos computacionales que, para un desarrollador independiente, requiere mucha inversión. Aun así, los resultados tras el *fine-tuning*, a pesar de no lograr haber obtenido un modelo “experto” en el BOE, no son malos. De nuevo, el lado positivo, es que se ha desarrollado toda la arquitectura de ajuste fino de un modelo tipo BERT para la clasificación de texto del Boletín Oficial del Estado español, pudiendo cuando se tengan los recursos utilizarla y obtener el primer modelo BERT de esta índole.

4.5.1 Resultados y evaluación en el procesamiento del BOE por parte de la herramienta desarrollada

Para evaluar la eficiencia y precisión de la arquitectura diseñada, tanto en el procesamiento de texto mediante NLP como en la generación de una respuesta precisa de los modelos LLMs y la recuperación de información y contexto adecuada por parte del sistema RAG, como se ha mencionado, se han utilizado métricas adecuadas del procesamiento de lenguaje natural.

El procedimiento ha sido, en primer lugar, seleccionar un PDF concreto del BOE suficientemente extenso para contener variedad de elementos de donde extraer información y poner a prueba el sistema. Estos elementos son tablas, texto guionizado, texto jerarquizado, texto con palabras técnicas o con siglas, así como otro tipo de elementos.

En segundo lugar y localizado este PDF, explorar y generar manualmente un *dataset* basado en su contenido. Este *dataset* cuenta con cuatro columnas por cada muestra creada que ya se han comentado anteriormente, y son los atributos del archivo *queries.json*. Es decir, se han creado 10 muestras de preguntas (*user_question*) con su respuesta esperada asociada (*ground_truth*),

junto con la fecha (*date*) y el identificador del archivo del BOE (*boe_id*) como se muestra en la siguiente tabla.

```
[
  {
    "user_question": "¿Cuál es la duración total de las enseñanzas del ciclo de grado medio para Técnico Deportivo en Atletismo?",
    "ground_truth": "La duración total es de 1.005 horas, organizadas en dos ciclos.",
    "date": "2024-04-15",
    "boe_id": "BOE-A-2024-7296"
  },
  {
    "user_question": "¿Qué pruebas son necesarias para acceder al ciclo inicial de grado medio en atletismo?",
    "ground_truth": "Es necesario superar la prueba de carácter específico RAE-ATAT101 descrita en el anexo VI del Real Decreto 427/2023.",
    "date": "2024-04-15",
    "boe_id": "BOE-A-2024-7296"
  },
  {
    "user_question": "¿Qué se menciona sobre la accesibilidad universal en relación con los espacios y equipamientos necesarios para la enseñanza del grado medio en atletismo?",
    "ground_truth": "Los espacios y equipamientos deben cumplir con la normativa sobre igualdad de oportunidades, diseño para todos y accesibilidad universal, prevención de riesgos laborales, así como con la normativa sobre seguridad y salud en el puesto de trabajo.",
    "date": "2024-04-15",
    "boe_id": "BOE-A-2024-7296"
  },
  {
    "user_question": "¿Qué competencias desarrolla el módulo de 'Tecnificación y perfeccionamiento técnico en carreras y marcha' en el ciclo final de grado medio en atletismo?",
```

```
"ground_truth": "Este módulo desarrolla competencias relacionadas con la valoración de la ejecución técnica, la elaboración de secuencias de aprendizaje, y la dirección de sesiones de entrenamiento técnico, táctico y de acondicionamiento físico.",
```

```
"date": "2024-04-15",
```

```
"boe_id": "BOE-A-2024-7296"
```

```
},
```

```
{
```

```
"user_question": "¿Qué aspectos se analizan en la técnica de la zancada en las carreras de velocidad?",
```

```
"ground_truth": "Se analizan los parámetros de la carrera, como la amplitud y la frecuencia de zancadas, la posición del tronco y la acción de los brazos, y se utilizan ciclogramas y kinogramas para la corrección de errores técnicos.",
```

```
"date": "2024-04-15",
```

```
"boe_id": "BOE-A-2024-7296"
```

```
},
```

```
{
```

```
"user_question": "¿Qué técnicas de transmisión del testigo se emplean en el relevo corto?",
```

```
"ground_truth": "Se emplean técnicas como el cambio 'por abajo', 'por arriba' y 'de empuje', cada una con su propio movimiento de entrega y recepción del testigo.",
```

```
"date": "2024-04-15",
```

```
"boe_id": "BOE-A-2024-7296"
```

```
},
```

```
{
```

```
"user_question": "¿Cuál es la duración total del ciclo inicial de grado medio en atletismo según la distribución horaria del currículo?",
```

```
"ground_truth": "La duración total del ciclo inicial de grado medio en atletismo es de 430 horas.",
```

```
"date": "2024-04-15",
```

```
"boe_id": "BOE-A-2024-7296"
```

```
},
```

```
{
```



```
"user_question": "¿Qué requisitos deben cumplir los integrantes del tribunal evaluador de las pruebas de carácter específico en atletismo?",

"ground_truth": "Las personas integrantes del tribunal evaluador de las pruebas de carácter específico deberán acreditar la titulación de Técnico Deportivo Superior en Atletismo o estar en posesión de la homologación al título de Técnico Deportivo Superior en Atletismo. Además, como mínimo una de las personas integrantes del tribunal será propuesto por la Real Federación Española de Atletismo.",

"date": "2024-04-15",

"boe_id": "BOE-A-2024-7296"

},

{

"user_question": "¿Cuáles son los factores determinantes del rendimiento en el triple salto según el modelo conceptual?",

"ground_truth": "Los factores determinantes del rendimiento en el triple salto incluyen la relación entre la velocidad de carrera y la distancia saltada, la distribución del esfuerzo en cada salto, la relación entre la altura del salto y el frenado horizontal, la evolución de la técnica en la formación atlética a largo plazo, y las variaciones asociadas a la maduración y competencia técnica.",

"date": "2024-04-15",

"boe_id": "BOE-A-2024-7296"

},

{

"user_question": "¿Cuánto deben o pueden pesar los balones medicinales?",

"ground_truth": "Los balones medicinales deben pesar 3 kg, 4 kg o 5 kg",

"date": "2024-04-15",

"boe_id": "BOE-A-2024-7296"

}

]
```

Tabla 7: Archivo queries.json utilizado para la evaluación del sistema RAG.

El archivo PDF usado para la evaluación tiene un id de BOE-A-2024-7296 y contiene la normativa y directrices oficiales sobre el currículo para los ciclos inicial y final del grado medio de Técnico Deportivo en Atletismo. Es decir, el documento establece las directrices oficiales para la formación de Técnicos Deportivos en Atletismo en el grado medio, incluyendo la estructura del

currículo, los requisitos de acceso, la metodología, la evaluación, y las infraestructuras necesarias.



BOLETÍN OFICIAL DEL ESTADO



Núm. 92

Lunes 15 de abril de 2024

Sec. I. Pág. 41281

I. DISPOSICIONES GENERALES

MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES

7296 Orden EFD/322/2024, de 8 de abril, por la que se establece el currículo de los ciclos inicial y final de grado medio correspondiente al título de Técnico Deportivo en Atletismo.

De acuerdo con la Ley Orgánica 2/2006, de 3 de mayo, de Educación, las Administraciones educativas establecerán el currículo de las enseñanzas deportivas de régimen especial, del que formarán parte los aspectos básicos de los currículos establecidos en los correspondientes títulos. Los centros docentes desarrollarán y completarán, en su caso, el currículo de las diferentes etapas y ciclos en uso de su autonomía y tal como se recoge en el capítulo II del título V de la citada Ley.

Con la publicación del Real Decreto 427/2023, de 6 de junio, por el que se establece el título de Técnico Deportivo en Atletismo y se fijan su currículo básico y los requisitos de acceso, el Gobierno ha fijado el perfil profesional y aquellos otros aspectos de la ordenación

Ilustración 21: Archivo PDF BOE-A-2024-7296.

Elegida la fuente de información, en este caso el PDF BOE-A-2024-7296, y generado el *dataset* con las preguntas y respuestas basadas en su contenido, se procede a ejecutar la herramienta pasando por todos sus módulos ya descritos. El comportamiento de cada uno se rige por los archivos de configuración presentes en los anexos donde se definen las técnicas de NLP usadas y las características del grafo “multi-agente” con RAG mejorado usado entre otras funcionalidades de la herramienta.

La ejecución del módulo RAG_EVAL permite generar un reporte con las métricas de rendimiento de la herramienta y en concreto del grafo lógico. Estas graficas se exponen y explican a continuación.

La siguiente tabla muestra la *query*, la respuesta generada por el sistema RAG, el contexto recuperado y utilizado por el sistema para responder y la respuesta esperada o *ground truth*.

Query ID	Query	RAG Answer	Contexts	Ground Truth
----------	-------	------------	----------	--------------

0	¿Cuál es la duración total de las enseñanzas del ciclo de grado medio para Técnico Deportivo en Atletismo?	La duración total de las enseñanzas correspondientes al título de Técnico Deportivo en Atletismo es de 1.005 horas.	<p><i>Chunk 1:</i> 1. La duración total de las enseñanzas correspondientes a este ciclo de grado medio, incluido el respectivo módulo de formación práctica, correspondiente al título de Técnico Deportivo en Atletismo es de 1.005 horas. Estas enseñanzas se organizan en dos ciclos: - Ciclo inicial de grado medio en atletismo, con una duración de 430 horas. - Ciclo final de grado medio en atletismo, con una duración de 575 horas.</p> <p><i>Chunk 2:</i> 92 lunes I. 41349 Criterios de organización del entrenamiento referidos a la duración de la vida deportiva.</p> <p><i>Chunk 3:</i> Actividades docentes, cursos y sesiones técnicas. - Servicios: documentación, información y publicaciones. - Los títulos y ciclos de enseñanza deportiva en atletismo: - Normativa.</p>	La duración total es de 1.005 horas, organizadas en dos ciclos.
1	¿Qué pruebas son necesarias para acceder al ciclo inicial de grado medio en atletismo?	Para acceder al ciclo inicial de grado medio en atletismo, es necesario superar la prueba de carácter específico RAE-ATAT101.	<p><i>Chunk 1:</i> # Ciclo inicial de grado medio en atletismo.</p> <p><i>Chunk 2:</i> # Artículo 10. Requisitos de carácter específico. 1. Para acceder al ciclo inicial de grado medio en atletismo será necesario superar la prueba de carácter específico RAE-ATAT101 descrita en el anexo VI del Real Decreto 427/2023, de 6 de junio. 2. El requisito de carácter específico del ciclo inicial tiene asignada una carga horaria de formación de 120 horas sobre la duración total del ciclo inicial.</p> <p><i>Chunk 3:</i> Módulos específicos de enseñanza deportiva del ciclo inicial en atletismo Módulo específico de enseñanza deportiva: Metodología de la iniciación en atletismo Código: MED-ATAT105.</p>	Es necesario superar la prueba de carácter específico RAE-ATAT101 descrita en el anexo VI del Real Decreto 427/2023.
2	¿Qué se menciona sobre la accesibilidad universal en	Se menciona que los espacios y equipamientos necesarios para la	<i>Chunk 1:</i> Los espacios y equipamientos que deben reunir los centros autorizados para impartir las enseñanzas deportivas son los	Los espacios y equipamientos deben cumplir con la normativa sobre

	relación con los espacios y equipamientos necesarios para la enseñanza del grado medio en atletismo?	enseñanza del atletismo deben cumplir con la normativa sobre accesibilidad universal.	establecidos en los anexos V-A y V-B, y deben permitir el desarrollo de las actividades de enseñanza cumpliendo con la normativa sobre igualdad de oportunidades, diseño para todos y accesibilidad universal, prevención de riesgos laborales, así como con la normativa sobre seguridad y salud en el puesto de trabajo. # Artículo 7. Titulaciones y acreditación de requisitos del profesorado. <i>Chunk 2:</i> - El contexto y las limitaciones originadas por la falta de acceso a la información de la oferta de la práctica del atletismo adaptado y la difusión de la misma. <i>Chunk 3:</i> - Barreras arquitectónicas en las instalaciones de atletismo adaptado y espacios sociales. Tipos y características. - Las actitudes en el entorno de las personas con discapacidad hacia la práctica del atletismo adaptado.	igualdad de oportunidades, diseño para todos y accesibilidad universal, prevención de riesgos laborales, así como con la normativa sobre seguridad y salud en el puesto de trabajo
3	¿Qué competencias desarrolla el módulo de 'Tecnificación y perfeccionamiento técnico en carreras y marcha' en el ciclo final de grado medio en atletismo?	El módulo de "Tecnificación y perfeccionamiento técnico en carreras y marcha" desarrolla competencias para valorar la ejecución técnica y el rendimiento del atleta en el entrenamiento y durante la competición, analizando aspectos fundamentales de la técnica y situaciones tácticas de las carreras y la marcha.	<i>Chunk 1:</i> La formación del módulo contribuye a completar en un entorno deportivo y profesional real los objetivos generales y las competencias del ciclo final de grado medio en atletismo. # Finalidad del módulo de Formación práctica: - Completar la adquisición de competencias profesionales y deportivas propias del ciclo final de grado medio de enseñanza deportiva, alcanzadas en el centro educativo. --- # <i>Chunk 2:</i> 1. Valora la ejecución técnica y el rendimiento del atleta en el entrenamiento y durante la competición en cada una de las etapas de la fase de tecnificación y perfeccionamiento, analizando los aspectos fundamentales de la técnica y las situaciones tácticas de las carreras (velocidad, vallas, medio fondo, fondo, obstáculos y trail) y la marcha. Etapa de entrenamiento básico de la marcha y las carreras	Este módulo desarrolla competencias relacionadas con la valoración de la ejecución técnica, la elaboración de secuencias de aprendizaje, y la dirección de sesiones de entrenamiento técnico, táctico y de acondicionamiento físico.

			<p>(12-15 años): Características y factores que condicionan la tecnificación de la marcha y la carrera. </p> <p><i>Chunk 3:</i> La formación del módulo contribuye a completar en un entorno deportivo y profesional real los objetivos generales y las competencias del ciclo inicial de grado medio en atletismo. # B) Finalidad del módulo de Formación práctica: - Completar la adquisición de competencias profesionales y deportivas propias del ciclo inicial de grado medio de enseñanza deportiva, alcanzadas en el centro educativo.</p>	
4	¿Qué aspectos se analizan en la técnica de la zancada en las carreras de velocidad?	Según el contexto, en el análisis mecánico de la zancada de carrera se analizan los parámetros de la carrera, específicamente la relación velocidad de carrera / distancia saltada.	<p><i>Chunk 1:</i> 92 Lunes I. 41328 # Enseñanza de los ejercicios de asimilación técnica de las carreras de velocidad: - Ejercicios para el aprendizaje de los fundamentos técnicos de la salida de tacos.</p> <p><i>Chunk 2:</i> - Objetivos de la corrección de la técnica de carrera. Etapas en la adquisición del dominio técnico. - Motivación hacia la correcta ejecución de la carrera a alta velocidad. - Análisis mecánico de la zancada de carrera: - - Los parámetros de la carrera:</p> <p><i>Chunk 3:</i> - Relación velocidad de carrera / distancia saltada.</p>	Se analizan los parámetros de la carrera, como la amplitud y la frecuencia de zancadas, la posición del tronco y la acción de los brazos, y se utilizan ciclogramas y kinogramas para la corrección de errores técnicos.
5	¿Qué técnicas de transmisión del testigo se emplean en el relevo corto?	Según el contexto, se emplean tres técnicas de transmisión del testigo en el relevo corto: el cambio "por abajo", "por arriba" y "de empuje".	<p><i>Chunk 1:</i> - La velocidad del testigo en la zona. Posición de salida y puesta en acción de los diferentes relevistas. - La distancia de hándicap o referencia. Criterios para su determinación. - Las técnicas de transmisión del testigo. El cambio «por abajo», «por arriba» y «de empuje». Descripción del movimiento de entrega y recepción del testigo en las diferentes técnicas. - Los tres tiempos de la entrega del testigo. - Las tácticas de equipo en el relevo corto.</p>	Se emplean técnicas como el cambio 'por abajo', 'por arriba' y 'de empuje', cada una con su propio movimiento de entrega y recepción del testigo.

6	¿Cuál es la duración total del ciclo inicial de grado medio en atletismo según la distribución horaria del currículo?	La duración total del ciclo inicial de grado medio en atletismo es de 430 horas.	<p><i>Chunk 1:</i> 1. La duración total de las enseñanzas correspondientes a este ciclo de grado medio, incluido el respectivo módulo de formación práctica, correspondiente al título de Técnico Deportivo en Atletismo es de 1.005 horas. Estas enseñanzas se organizan en dos ciclos: - Ciclo inicial de grado medio en atletismo, con una duración de 430 horas. - Ciclo final de grado medio en atletismo, con una duración de 575 horas.</p> <p><i>Chunk 2:</i> # Ciclo inicial de grado medio en atletismo.</p> <p><i>Chunk 3:</i> "Total del ciclo: 430 h # Ciclo final de grado medio en atletismo.</p>	La duración total del ciclo inicial de grado medio en atletismo es de 430 horas.
7	¿Qué requisitos deben cumplir los integrantes del tribunal evaluador de las pruebas de carácter específico en atletismo?	Los integrantes del tribunal evaluador de las pruebas de carácter específico en atletismo deben acreditar la titulación de Técnico Deportivo Superior en Atletismo o estar en posesión de la homologación al título de Técnico Deportivo Superior en Atletismo.	<p><i>Chunk 1:</i> 1. El tribunal estará formado por un/a presidente/a, un/a secretario/a y tres personas evaluadoras. 2. Las personas integrantes del tribunal evaluador de las pruebas de carácter específico deberán acreditar la titulación de Técnico Deportivo Superior en Atletismo o estar en posesión de la homologación al título de Técnico Deportivo Superior en Atletismo. Además, como mínimo una de las personas integrantes del tribunal será propuesto por la Real Federación Española de Atletismo. 3. La Presidencia designará a una de las personas evaluadoras para ejercer la vicepresidencia de dicho tribunal.</p> <p><i>Chunk 2:</i> - Supuestos prácticos basados en los conceptos trabajados para el asesoramiento del atleta en la competición analizando el reglamento general (vinculado a R.A.1).</p> <p><i>Chunk 3:</i> 1. Colabora en la preparación de eventos y actividades de iniciación en atletismo, identificando los requisitos de la actividad y</p>	Las personas integrantes del tribunal evaluador de las pruebas de carácter específico deberán acreditar la titulación de Técnico Deportivo Superior en Atletismo. Además, como mínimo una de las personas integrantes del tribunal será propuesto por la Real Federación Española de Atletismo.

			describiendo los procesos de inscripción y las características organizativas y reglamentarias. - Las pruebas alternativas para categorías menores: reglamentación RFEA y sus características básicas: - Categorías por edades.	
8	¿Cuáles son los factores determinantes del rendimiento en el triple salto según el modelo conceptual?	Según el contexto, los factores determinantes del rendimiento en el triple salto incluyen la distribución del esfuerzo, la relación altura del salto/frenado horizontal, la técnica del triple salto en sus fases (carrera, 1º/Hop, 2º/Step y 3º/Jump), la coordinación de los brazos y la motivación hacia la ejecución técnica.	<p><i>Chunk 1:</i> - Distribución del esfuerzo. Ratios o porcentajes de cada salto. - Relación altura del salto/frenado horizontal. Concepto básico. - La evolución de la técnica de los multisaltos y del triple salto en la formación atlética a largo plazo. - Variaciones asociadas a la maduración y a la competencia técnica. - Limitación de la carrera. - Esquema técnico del 1º, 2º y 3er salto - Coordinación de los brazos. - La técnica del triple salto y cada una de sus fases: la carrera, 1º/Hop, 2º/Step y 3º/Jump. - Variabilidad óptima, soluciones o respuestas adecuadas inadecuados. Estrategias de modelado. Refuerzo y corrección. - Motivación hacia la ejecución técnica.</p> <p><i>Chunk 2:</i> - Características y factores que condicionan la tecnificación de los saltos atléticos.</p> <p><i>Chunk 3:</i> "# El triple salto:.</p>	Los factores determinantes del rendimiento en el triple salto incluyen la relación entre la velocidad de carrera y la distancia saltada, la distribución del esfuerzo en cada salto, la relación entre la altura del salto y el frenado horizontal, la evolución de la técnica en la formación atlética a largo plazo, y las variaciones asociadas a la maduración y competencia técnica.
9	¿Cuánto deben o pueden pesar los balones medicinales?	Los balones medicinales pueden pesar 3 kg, 4 kg o 5 kg.	<p><i>Chunk 1:</i> - Combas, balones blandos, picas, balón medicinal, bancos suecos, vallas de diferentes alturas, colchonetas, aros. - Material de cada especialidad. ---</p> <p><i>Chunk 2:</i> - Pulsímetros. - Balones medicinales (3 kg, 4 kg, 5 kg). - Colchonetas. - Mancuernas de diferentes pesos. </p> <p><i>Chunk 3:</i> - El lanzamiento de peso:.</p>	Los balones medicinales deben pesar 3 kg, 4 kg o 5 kg

Tabla 8: Resultados obtenidos tras la ejecución del sistema RAG: query, response, context y ground truth.

	context_precision	faithfulness	answer_relevancy	context_recall
count	10.0	10.0	10.0	10.0
mean	0.82	0.95	0.93	0.86
std	0.23	0.16	0.05	0.25
min	0.5	0.5	0.84	0.33
25%	0.58	1.0	0.89	0.85
50%	0.92	1.0	0.94	1.0
75%	1.0	1.0	0.97	1.0
max	1.0	1.0	1.0	1.0

Ilustración 22: Estadísticas de las métricas de la librería RAGAS.

En la ilustración anterior, se muestran las estadísticas de las cuatro métricas de evaluación del sistema RAG en el conjunto de 10 muestras del *testset*: *context precision*, *faithfulness*, *answer relevancy* y *context recall*.

Si se analiza esta ilustración que contine una tabla con las estadísticas de las métricas se puede observar que:

Para el *context precision* se tiene una media de 0.82 lo que indica que, en promedio, el 82% del contexto recuperado es relevante para responder la pregunta. Sin embargo, la desviación estándar de 0.23 indica una variabilidad considerable en la precisión del contexto recuperado, lo que sugiere que, en algunas muestras, como son *query* ID 1 y 4, el sistema está fallando más en la recuperación de contexto relevante (se observa un valor mínimo de 0.5).

Para la métrica de *faithfulness* se tiene una media de 0.95 que muestra que las respuestas generadas son altamente fieles al contexto en la mayoría de las ocasiones. Sin embargo, la desviación estándar de 0.16 y un valor mínimo de 0.5 reflejan que, en la *query* de ID 5, la respuesta generada no fue del todo fiel al contexto.

La métrica de *answer relevancy* cuya media es de 0.93 y cuya desviación estándar de tan solo 0.05, muestran que la relevancia de las respuestas es bastante consistente y alta. Esto indica que el sistema generalmente genera respuestas que están alineadas con las preguntas, con muy poca variabilidad entre las muestras.

Para la métrica de *context recall* se ha obtenido una media de 0.86 indicando que, en promedio, el sistema recupera el 86% del contexto relevante disponible. Sin embargo, la desviación estándar de 0.25 es alta, lo que indica que el sistema tiene problemas significativos en algunos casos (mínimo de 0.33) para recuperar todo el contexto relevante, lo que podría afectar la calidad de la respuesta generada.

Como conclusiones a los reportes de la ilustración 18 y, también los diagramas de cajas de la ilustración 20, tanto el *context precision* como el *context recall* presentan una alta variabilidad, como lo indican las desviaciones estándar de 0.23 y 0.25 respectivamente. Aunque los valores medios son buenos (0.82 y 0.86), la dispersión muestra que el sistema no siempre logra recuperar el contexto más relevante, lo cual es crítico en un sistema RAG. Para el sistema diseñado, el mejorar la precisión y cobertura del contexto debería ser una prioridad.

En el caso de la fidelidad, esta es mayoritariamente alta. Aunque la *faithfulness*, tiene una media alta (0.95), la desviación estándar de 0.16 y el mínimo de 0.5 revelan que en una muestra el sistema no logró ser fiel al contexto. Este es un problema, ya que una falta de fidelidad puede generar respuestas incorrectas o engañosas, afectando la confiabilidad del sistema.

La *answer relevancy* tiene la desviación estándar más baja (0.05), lo que indica que el sistema genera respuestas consistentemente relevantes. Esto es positivo y sugiere que, en términos de relevancia, el sistema está funcionando de manera robusta.

Este análisis revela que el sistema RAG tiene un buen rendimiento en general, pero que aún existen áreas clave donde se puede mejorar, especialmente en la recuperación de contexto y la fidelidad.

	context_precision	faithfulness	answer_relevancy	context_recall
0	1.0	1.0	0.95	1.0
1	0.5	1.0	0.87	1.0
2	0.83	1.0	0.95	1.0
3	1.0	1.0	0.93	0.33
4	0.5	1.0	0.94	0.5
5	1.0	0.5	0.84	1.0
6	0.83	1.0	0.88	1.0
7	1.0	1.0	1.0	1.0
8	1.0	1.0	0.97	0.8
9	0.5	1.0	0.99	1.0

Ilustración 23: Valor de cada métrica de evaluación del sistema RAG para cada una de las 10 muestras del dataset de evaluación.

Las métricas presentadas en la tabla de la ilustración 19 son *context_precision*, *faithfulness*, *answer_relevancy* y *context_recall*.

Como se ha definido anteriormente, la métrica de *context precision*, mide la proporción de los fragmentos de contexto recuperados que son en realidad relevantes para responder correctamente la pregunta del usuario.

En la tabla, los valores de esta métrica varían entre 0.5 y 1.0. Esto indica que, en algunos casos, el sistema recupera fragmentos de texto o *chunks* que son completamente relevantes (1.0), mientras que, en otros casos, solo la mitad de los fragmentos recuperados son relevantes (0.5) como es el caso de la pregunta de ID 4 y de ID 1.

En lo que respecta a la métrica de *faithfulness*, que evalúa si la respuesta generada es fiel a la información contenida en los *chunks* contextuales recuperados, en la tabla, todos los valores son 1.0 excepto en la pregunta de ID 5 donde la fidelidad es 0.5. Esto se debe a que el número de afirmaciones presentes en la respuesta del sistema RAG dividido entre el número de afirmaciones en el único *chunk* de contexto recuperado da un valor de 0.5. Es decir, el LLM encargado de generar la respuesta no ha utilizado todo el contexto recuperado para generar la respuesta. Esto no es negativo para el sistema RAG siempre que, por ejemplo, otras métricas como *anwer relevancy* tengan una puntuación alta. En este caso se cumple que para esta *query*

la *answer relevancy* es de 0.84, un valor aceptable que se podría mejorar forzando al LLM a usar todo el contexto disponible dado que los valores de *context recall* y *context precision* son de 1.0.

Para la métrica de *answer relevancy*, que mide cuán relevante es la respuesta generada en relación con la pregunta formulada, los valores oscilan entre 0.84 y 1.0, lo que indica que, en la mayoría de los casos, las respuestas generadas son bastante relevantes, aunque no siempre alcanzan el máximo posible.

Y, por último, en la métrica *context recall*, que mide la proporción del contexto relevante que fue efectivamente recuperado en comparación con todo el contexto relevante disponible, los valores de esta métrica van desde 0.33 hasta 1.0, lo que indica una variabilidad significativa en la capacidad del sistema para recuperar todo el contexto necesario para generar una buena respuesta. En concreto muestra que ciertos *chunks* recuperados son inútiles o no son relevantes para generar una respuesta correcta. Por ejemplo, para la pregunta de ID 3, solo 1/3 (0.33) de su contexto (uno de los tres *chunks*) ha sido relevante.

Como conclusiones de estos resultados se tiene:

- Variabilidad en la Recuperación de Contexto: La métrica de *context recall* muestra una alta variabilidad, sugiriendo que en algunos casos el sistema no logra recuperar todo el contexto necesario, lo que podría limitar la calidad de la respuesta generada. Este problema del sistema RAG se podría corregir cambiando el modelo que genera los *embeddings* del texto, dado que tal vez, este no crea vectores geoméricamente cercanos para oraciones semánticamente semejantes. O tal vez el problema pueda solucionarse variando las técnicas de NLP, para evitar ciertas palabras vacías en oraciones que hayan podido modificar este vector denso porque no se alinean con el

significado del resto de la oración. Cabe mencionar que la técnica de *splitting* de los textos también es muy sensible a una buena recuperación de contexto.

- Fidelidad Mayormente Alta: La métrica de *faithfulness* es mayoritariamente alta, lo que indica que el sistema generalmente genera respuestas alineadas con la información recuperada, aunque con alguna excepción notable.
- Relevancia Consistente: *Answer relevancy* es consistente, con valores cercanos a 1.0 en la mayoría de los casos, lo que sugiere que el sistema es efectivo en generar respuestas que son relevantes para la pregunta planteada.
- Precisión de Contexto: La *context precision* también muestra variabilidad, lo que indica que en algunos casos el sistema recupera información que no es completamente relevante, lo que podría afectar la calidad final de la respuesta.

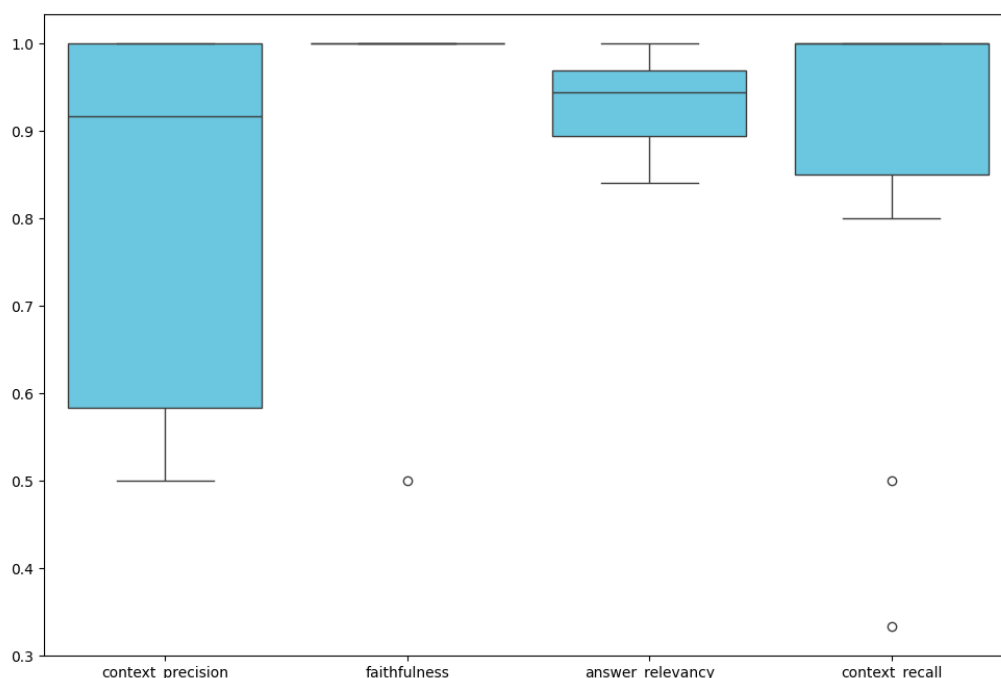


Ilustración 24: Diagrama de cajas para cada métrica de evaluación calculada sobre el testset.

En la ilustración 20 se presenta un diagrama de cajas (*box plot*) para cada una de las métricas de evaluación del sistema RAG. Este tipo de gráfico ayuda a visualizar la distribución de los datos, identificando la mediana, los cuartiles y posibles valores atípicos (*outliers*).

Para la métrica de *context precisión* se observa:

- Mediana: La línea negra en el centro de la caja indica que la mediana de esta métrica es aproximadamente 0.92.
- Rango Inter cuartil (IQR): La caja abarca desde el percentil 25 (alrededor de 0.58) hasta el percentil 75 (1.0). Esto significa que el 50% de las observaciones de esta métrica están en este rango.

Jorge Resino Martin

- Extremos: Los bigotes del gráfico indican que los valores mínimos y máximos (sin considerar *outliers*) están en torno a 0.5 y 1.0, respectivamente.
- Valores atípicos: No hay valores atípicos significativos en esta métrica, ya que no se ven puntos fuera de los bigotes.

Es decir, la métrica de *context precision* tiene una dispersión considerable. La mitad de las muestras tienen valores cercanos a 1.0, pero el otro 50% tiene una dispersión considerable hacia valores más bajos, lo que indica variabilidad en la precisión con la que el sistema recupera contexto relevante, una conclusión recurrente en los anteriores análisis realizado.

Para la métrica de *faithfulness* se observa:

- Mediana: La mediana es exactamente 1.0, lo que indica que, en la mayoría de los casos, la respuesta generada es completamente fiel al contexto recuperado.
- Rango Inter cuartil (IQR): La caja es extremadamente pequeña, lo que muestra que el 50% de los valores están cercanos a 1.0.
- Extremos: Los bigotes indican que casi todos los valores están cercanos a 1.0, pero hay al menos un *outlier* con un valor muy bajo (0.5), que es un caso en el que la respuesta no fue fiel al contexto.
- Valores atípicos: Se observa un *outlier* debajo de los bigotes, representando una instancia en la que la fidelidad fue significativamente menor (alrededor de 0.5).

Como conclusión el diagrama de cajas de *faithfulness* muestra lo ya mencionado que el valor de esta métrica e el *dataset* generado es alta y consistente en casi todos los casos, excepto por un único valor atípico que podría reflejar un error o una desviación significativa en el sistema.

Para la métrica de *answer relevancy*:

- Mediana: La mediana es aproximadamente 0.94, lo que significa que la mayoría de las respuestas son altamente relevantes.
- Rango Inter cuartil (IQR): La caja abarca desde el percentil 25 (0.89) hasta el percentil 75 (0.97), lo que muestra una distribución muy estrecha y concentrada hacia valores altos.
- Extremos: Los bigotes cubren un rango que va desde alrededor de 0.84 hasta 1.0.
- Valores atípicos: No se observan valores atípicos en esta métrica.

La métrica de *answer relevancy* es alta y consistente en todas las muestras, con una variabilidad muy baja, lo que sugiere que el sistema genera respuestas bastante relevantes para las preguntas.

Para la métrica de *context recall*:

- Mediana: La mediana es de 1.0, lo que indica que, en al menos la mitad de los casos, el sistema recupera todo el contexto relevante.
- Rango Inter cuartil (IQR): La caja abarca desde aproximadamente 0.85 hasta 1.0, lo que indica que el 50% de los valores se encuentran dentro de este rango.
- Extremos: Los bigotes cubren desde un mínimo de aproximadamente 0.33 hasta 1.0.
- Valores atípicos: Se observa un *outlier* en la parte baja (alrededor de 0.33), lo que indica un caso en el que el sistema recuperó una proporción muy baja del contexto relevante.

Es decir, la *context recall* es alta en la mayoría de los casos, pero existen algunas muestras donde el sistema no logra recuperar suficiente contexto, lo que podría estar afectando la calidad de las respuestas en esos casos, aunque según se ha analizado antes esto no sucede.

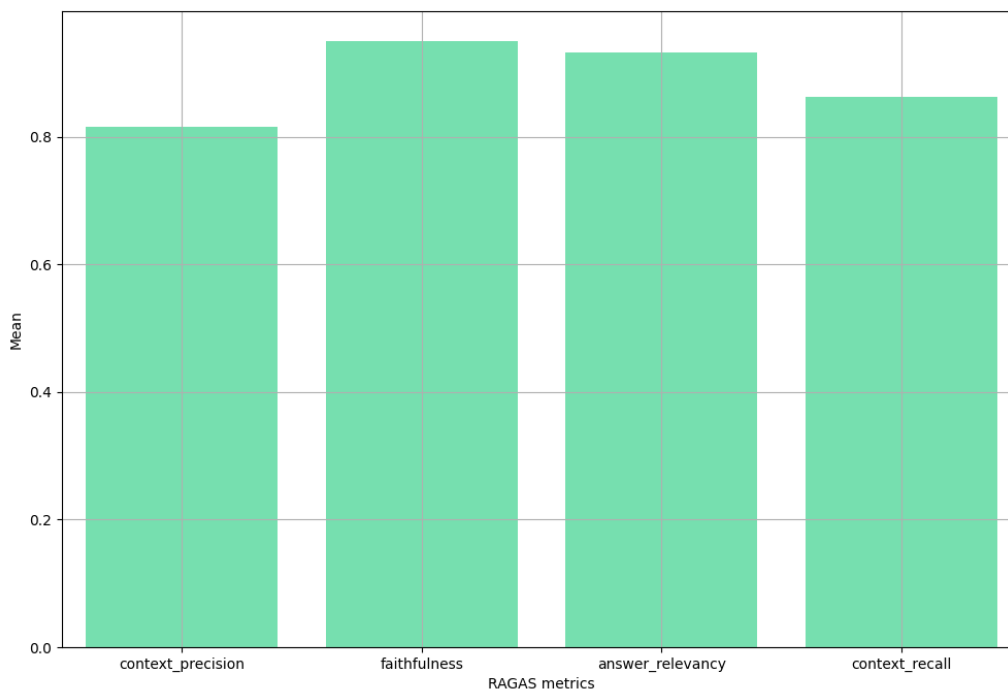


Ilustración 25: Valor medio para cada métrica de evaluación calculada sobre el testset.

En la ilustración 21 se muestra un gráfico de barras que representa las medias de las métricas de evaluación del sistema RAG calculadas para las 10 muestras ejecutadas sobre el grafo lógico.

Debido al extenso análisis ya realizado, esta grafica no aporta una nueva información n de los resultados. Según se ha comentado, la precisión del contexto tiene una media menor en comparación con las otras métricas. Esto indica que, en promedio, el 82% del contexto recuperado es relevante, pero hay margen de mejora en la precisión del sistema al recuperar los fragmentos de información correctos.

La métrica de *faithfulness* tiene la media más alta del conjunto, lo que indica que el sistema RAG genera respuestas que, en general, son fieles al contexto recuperado. Esto es un indicador muy positivo, ya que la fidelidad es crucial para mantener la exactitud de las respuestas.

La relevancia de las respuestas también tiene una media alta, lo que significa que, en promedio, las respuestas generadas son altamente relevantes para la pregunta formulada. Esto refleja que el sistema genera respuestas adecuadas y coherentes.

El *context recall* tiene una media ligeramente inferior a la relevancia y fidelidad, lo que indica que, en promedio, el sistema recupera el 86% del contexto relevante. Esto sugiere que, aunque

Jorge Resino Martin

el sistema es relativamente eficiente en la recuperación de información, hay casos en los que no logra recuperar todo el contexto necesario para una respuesta óptima.

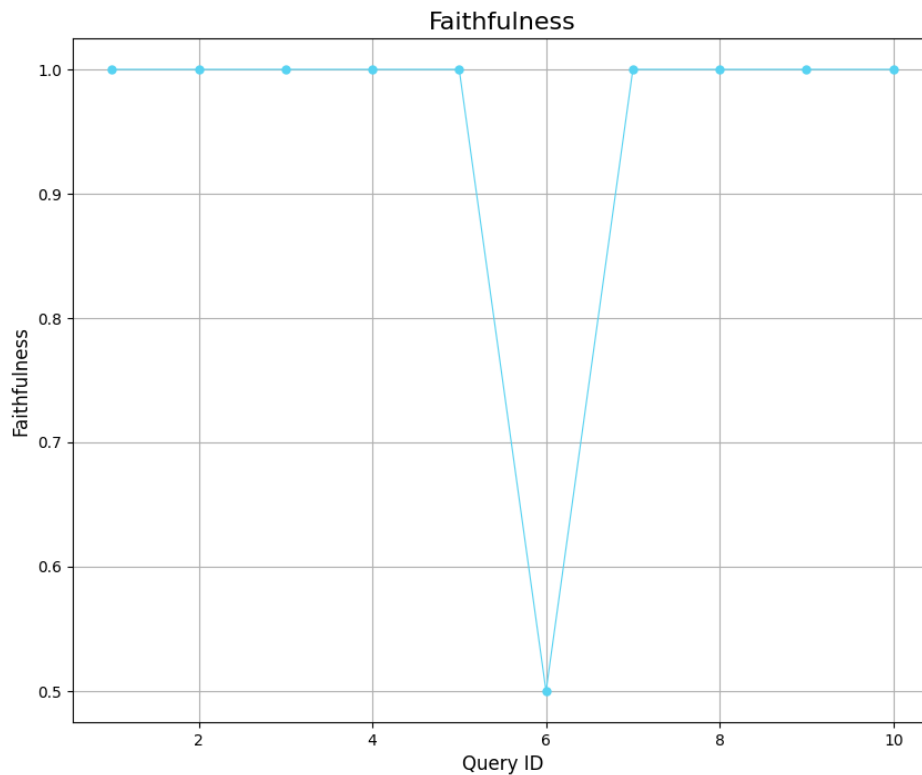


Ilustración 26: Valor de la métrica faithfulness para cada query del testset.

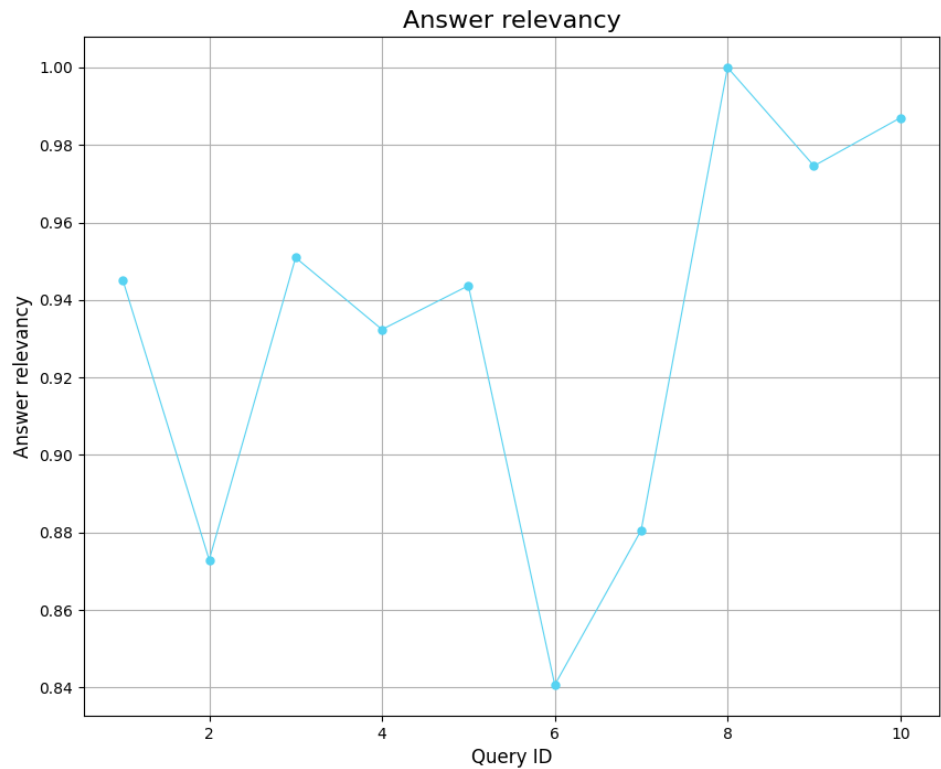


Ilustración 27: Valor de la métrica answer relevancy para cada query del testset.

Jorge Resino Martin

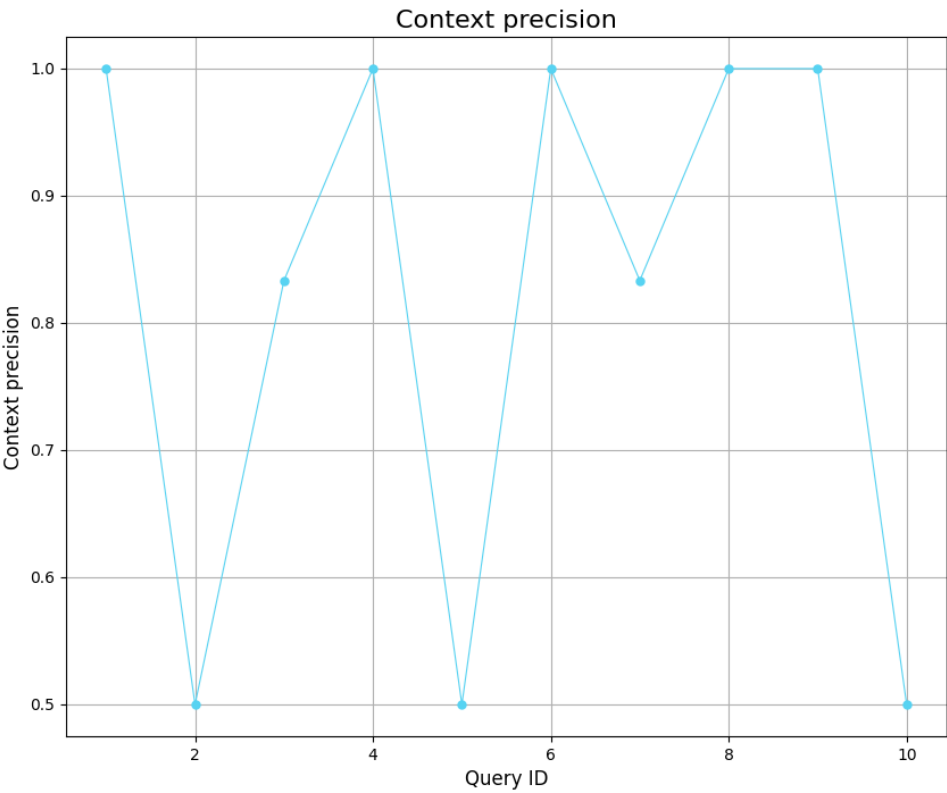


Ilustración 28: Valor de la métrica context precision para cada query del testset.

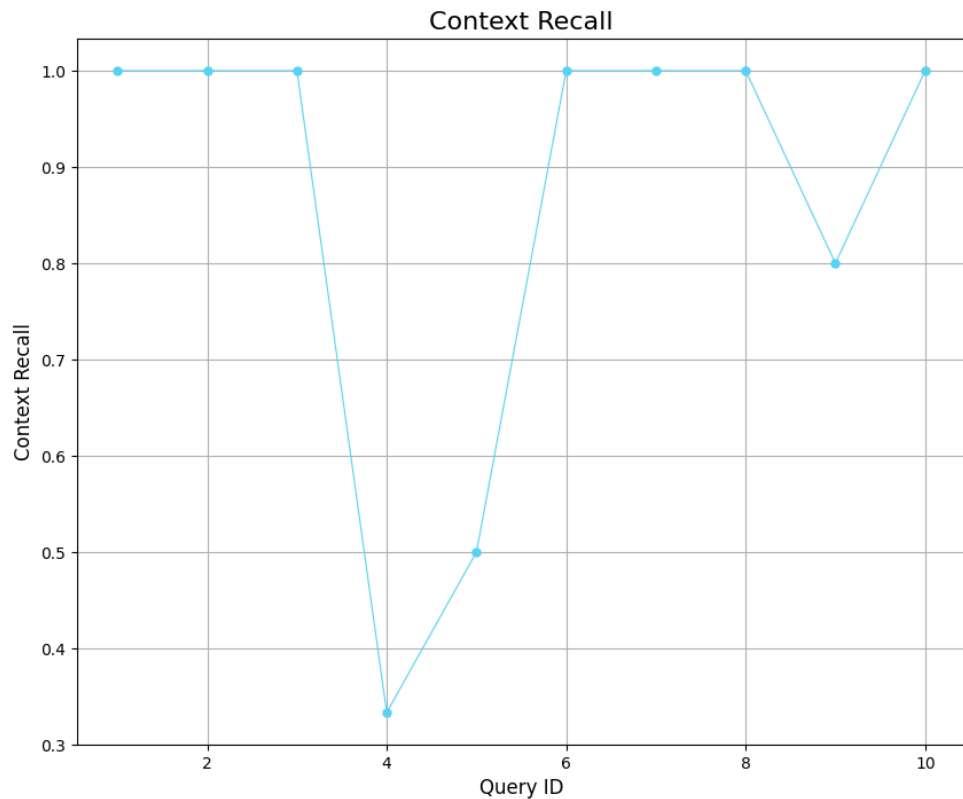


Ilustración 29: Valor de la métrica context recall para cada query del testset.

Por último, estas 4 ilustraciones muestran los gráficos de líneas con los valores de las cuatro métricas para cada consulta o *query* del *testset*. La razón de estas graficas es puramente de mejorar la explicación de los resultados además de, mediante un objeto visual y la tabla que mapea cada *query* con su ID, obtener una mejor comprensión del rendimiento del sistema RAG diseñado. Pero no es necesario comentarlas debido a que ya han sido desglosados los resultados en los análisis realizados.

Capítulo 5. DISCUSIÓN

5.1 Resultados principales

Los resultados obtenidos en el proyecto confirman la viabilidad de la herramienta desarrollada para procesar y extraer información del Boletín Oficial del Estado (BOE). La aplicación ha demostrado ser funcional, permitiendo la recuperación eficiente de información y la generación de respuestas relevantes a partir de textos complejos y técnicos en formato PDF.

Uno de los logros más destacables es la implementación del sistema de Recuperación Aumentada con Generación (RAG), que ha demostrado su eficacia en generar respuestas fieles al contexto recuperado. Sin embargo, se han identificado áreas de mejora, particularmente en la recuperación de contexto, donde en ciertos casos el sistema no logra seleccionar fragmentos lo suficientemente precisos para optimizar la respuesta final.

5.2 Discusión sobre la metodología

La metodología inicialmente propuesta ha resultado útil para el desarrollo del proyecto. El enfoque modular ha facilitado la implementación de los distintos componentes, desde la extracción y procesamiento de los datos hasta la generación de respuestas basadas en modelos de lenguaje (LLMs).

No obstante, durante el proyecto surgieron desafíos inesperados relacionados con la capacidad computacional requerida para realizar ajustes finos de los modelos *transformer-encoder* tipo BERT. El entrenamiento de estos modelos requiere recursos que no estaban disponibles en su totalidad, lo que afectó la precisión esperada de los modelos ajustados para la clasificación semántica de fragmentos del BOE. A pesar de ello, el desarrollo de la arquitectura permite escalar el proyecto en el futuro, cuando los recursos computacionales sean más accesibles.

5.3 Limitaciones del estudio

Existen varias limitaciones identificadas durante el desarrollo del proyecto:

1. Limitaciones de recursos computacionales: El ajuste fino de modelos grandes, como los utilizados en este proyecto, requiere una cantidad significativa de memoria y procesamiento. Debido a estas limitaciones, los resultados obtenidos en algunas áreas del proyecto no alcanzaron los niveles de precisión deseados, especialmente en la clasificación “multi-etiqueta” de fragmentos semánticos.
2. *Recall* del contexto: Aunque el sistema de RAG ha sido exitoso en general, hay margen de mejora en la precisión y cobertura de los fragmentos de texto recuperados. En ciertos casos, el sistema no logra capturar todo el contexto necesario para una respuesta óptima, lo que afecta la relevancia de las respuestas generadas.

5.4 Adaptación a cambios durante el proyecto

El desarrollo del proyecto requirió adaptaciones sobre la marcha. En particular, la limitación de recursos llevó a cambios en la metodología aplicada. Originalmente, se planeaba realizar un ajuste fino completo del modelo BERT, pero debido a las restricciones, se decidió optimizar solo ciertos parámetros. Además, la implementación de una arquitectura más flexible permitió realizar ajustes en tiempo real sin comprometer la funcionalidad de la herramienta.

Estas adaptaciones incluyeron la creación de un *testset* específico para evaluar la herramienta en escenarios prácticos. Este *testset* permitió evaluar con mayor precisión la capacidad de la herramienta para generar respuestas relevantes a consultas sobre el BOE, facilitando el ajuste del sistema a lo largo del proyecto

5.5 Impacto del proyecto

El impacto del proyecto ha sido positivo, tanto en el ámbito práctico como en el académico. A nivel práctico, la herramienta proporciona una solución accesible para usuarios no expertos en textos legales, permitiendo la extracción y generación de información relevante de manera automatizada. Esto representa una contribución significativa para aquellos que necesitan acceder rápidamente a información específica dentro de grandes volúmenes de documentos oficiales.

A nivel académico, el proyecto contribuye al avance en el procesamiento de lenguaje natural (NLP) aplicado a contextos legales. La metodología implementada y los resultados obtenidos proporcionan una base sólida para futuras investigaciones y mejoras en sistemas de RAG y en la optimización de la generación de texto basada en grandes modelos de lenguaje.

En resumen, los resultados obtenidos cumplen en gran medida con los objetivos planteados al inicio del proyecto, aunque existen áreas que requieren mayor optimización, especialmente en la recuperación precisa de contexto y la mejora del rendimiento del sistema RAG.

Capítulo 6. CONCLUSIONES

6.1 Conclusiones del trabajo

El objetivo general del proyecto fue desarrollar un sistema automatizado y eficiente capaz de extraer y generar información relevante a partir de consultas realizadas por el usuario basadas en los documentos del Boletín Oficial del Estado (BOE), utilizando una arquitectura modular basada en técnicas de procesamiento del lenguaje natural (NLP) y un sistema de Recuperación Aumentada con Generación (RAG).

El resultado obtenido cumple de manera satisfactoria con este objetivo. Se ha creado una herramienta funcional que permite procesar documentos del BOE en formato PDF, extrayendo información precisa y relevante mediante un sistema basado en modelos grandes de lenguaje (LLMs). La herramienta implementa técnicas avanzadas de NLP para procesar los textos del BOE y un grafo lógico con RAG mejorado para generar respuestas fundamentadas y contextualizadas a estas consultas de usuarios no expertos.

El sistema RAG ha demostrado ser eficaz en la recuperación de fragmentos relevantes del BOE y en la generación de respuestas útiles, aunque se han identificado áreas de mejora en la precisión y *recall* del contexto recuperado. A pesar de las limitaciones encontradas, especialmente en la capacidad computacional para ajustar completamente los modelos de clasificación de texto de tipo BERT, el desarrollo de una arquitectura escalable y flexible ha permitido adaptarse a estos desafíos y cumplir con los objetivos establecidos.

En resumen, el proyecto ha logrado desarrollar una aplicación automatizada capaz de cumplir con el objetivo general planteado, ofreciendo una solución innovadora para el procesamiento de grandes volúmenes de datos legales.

6.2 Conclusiones personales

El proyecto ha sido percibido como una experiencia enriquecedora y desafiante tanto a nivel personal como profesional. A lo largo del proceso, se ha adquirido un conocimiento profundo en el campo del procesamiento de lenguaje natural (NLP) y en la implementación de sistemas basados en modelos grandes de lenguaje (LLMs). La creación de una herramienta capaz de procesar documentos del Boletín Oficial del Estado (BOE) ha permitido aplicar de manera práctica conceptos complejos usando tecnologías actuales en cuanto a inteligencia artificial y procesamiento de lenguaje se refiere.

La importancia del tema ha sido destacada, ya que, además de los desafíos técnicos, se ha abordado una necesidad crítica en la automatización y mejora del acceso a la información legal, algo que puede tener un impacto significativo tanto para profesionales como para ciudadanos comunes. A lo largo del desarrollo, se ha aprendido a manejar, optimizar e iterar sobre el diseño de una herramienta compleja y extensa contando además con recursos limitados en la capacidad computacional.

Jorge Resino Martin

Se ha valorado este proyecto no solo como una oportunidad de crecimiento profesional, sino también como un esfuerzo y una oportunidad de crear un producto que haga frente a una necesidad real: procesamiento y acceso automatizado vía modelos grandes de lenguaje a información contenida en el BOE. De tal forma que, en un futuro, la evolución y mejora de esta aplicación creada mejore o permita una mayor eficiencia en la toma de decisiones legales y administrativas, así como un ahorro de tiempo de profesionales a la hora de consultar grandes cantidades de textos legales.

Capítulo 7. FUTURAS LÍNEAS DE TRABAJO

1. **Mejora del sistema RAG y evolución del sistema RAG hacia *GraphRAG*:** Aunque el sistema de Recuperación Aumentada con Generación (RAG) ha mostrado un buen rendimiento, especialmente en la generación de respuestas relevantes y fieles al contexto, se han identificado áreas de mejora. Por ejemplo, mejorar la precisión del contexto recuperado y el *recall* del contexto son áreas clave. Esto puede lograrse mediante el ajuste fino de modelos de recuperación o la inclusión de técnicas adicionales para mejorar la relevancia de los fragmentos recuperados. Un ejemplo de una de estas resonadas y actuales tecnologías es *GraphRAG*.

GraphRAG es un enfoque estructurado y jerárquico para la Generación Aumentada con Recuperación (RAG), que mejora sobre los métodos tradicionales que utilizan solo fragmentos de texto en búsquedas semánticas simples. A diferencia de estos enfoques más básicos, *GraphRAG* se basa en la construcción de un grafo de conocimiento a partir del texto, lo que permite una comprensión más profunda y estructurada de la información (dge, 2024).

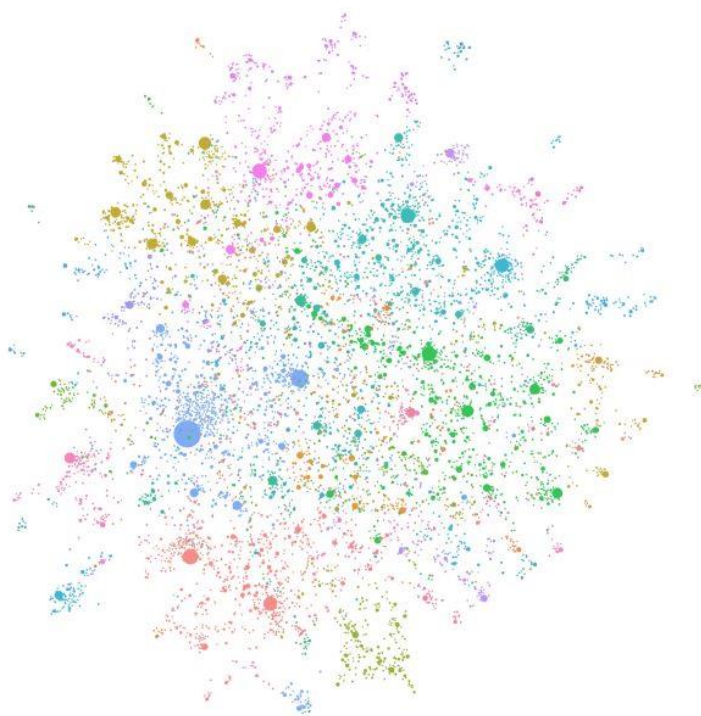


Ilustración 30: Un grafo de conocimiento generado por LLM construido utilizando GPT-4 Turbo.

El proceso de *GraphRAG* implica varios pasos clave:

- Extracción de un grafo de conocimiento: Se parte de un corpus de texto, del cual se extraen entidades (personas, lugares, organizaciones), relaciones y afirmaciones clave.

- Construcción de una jerarquía comunitaria: Mediante técnicas como el algoritmo de *clustering* de Leiden, se agrupan las entidades relacionadas dentro de comunidades, lo que permite organizar la información de manera jerárquica.
- Generación de resúmenes: Para cada comunidad, se generan resúmenes que facilitan una comprensión holística del conjunto de datos.

En el momento de realizar una consulta (*query*), *GraphRAG* utiliza estas estructuras para ofrecer mejores resultados:

- Búsqueda global: Se responde a preguntas generales sobre el corpus utilizando los resúmenes generados de las comunidades.
- Búsqueda local: Se realiza un análisis más detallado de entidades específicas, extendiéndose a los conceptos y relaciones asociadas.

GraphRAG mejora significativamente las capacidades de los modelos de lenguaje (LLMs) para razonar sobre información compleja y privada, superando a los enfoques tradicionales de RAG que se basan en la similitud vectorial de texto (*Baseline* RAG). En situaciones donde RAG tradicional tiene dificultades, como conectar fragmentos de información dispersos o comprender conceptos abstractos en colecciones grandes de datos, *GraphRAG* destaca.

Este enfoque es especialmente útil en contextos empresariales o legales, como el BOE, donde los LLMs necesitan trabajar con datos privados no entrenados previamente, como documentos comerciales o investigaciones propietarias (Microsoft, 2024).

2. **Incremento en la validación del *output* de los modelos LLMs para aumentar la confianza en las respuestas generadas:** Una de las limitaciones identificadas en el sistema es la variabilidad en la fidelidad de las respuestas generadas. Es fundamental incluir mecanismos automáticos de validación del contenido generado para asegurar que las respuestas sean consistentes y precisas en todos los casos. Esto podría implicar el desarrollo de evaluadores automáticos o el uso de técnicas de aprendizaje por refuerzo para garantizar una mayor y más robusta corrección de alucinaciones o errores factuales a las que ya implementa la aplicación.
3. **Mejoras en la interfaz de usuario (*Front-end*):** Aunque el núcleo del proyecto se ha centrado en la arquitectura y procesamiento del lenguaje, una mejora en la interfaz de usuario podría hacer que la herramienta sea más accesible y fácil de usar por parte de usuarios no expertos en tecnología, en el manejo de los archivos de configuración de tipo JSON que rigen el comportamiento de cada módulo de la aplicación o en la aplicación de técnicas de procesamiento de lenguaje natural. El *front-end* podría ser optimizado para presentar los resultados de una manera más clara y comprensible, permitiendo una interacción más intuitiva con el sistema. Una opción que se está

estudiando es la migración de una interfaz de *Streamlit* a una interfaz creada con *React (javascript)*.

4. **Ampliación del corpus y escalabilidad:** Actualmente, el sistema trabaja con un conjunto limitado de documentos del BOE. Una futura línea de trabajo sería la ampliación del corpus procesado, permitiendo que la herramienta funcione con otros tipos de documentos técnicos y gubernamentales, de otras fechas, buscando que sea capaz de posicionarse a nivel temporal y discernir o conocer los cambios legales y gubernamentales y su evolución a la hora de generar una respuesta. Además, se podría explorar la escalabilidad del sistema para manejar volúmenes aún mayores de datos sin afectar el rendimiento.

Capítulo 8. REFERENCIAS

- Asai, A. W. (2023). *SELF-RAG: Learning to Retrieve, Generate, and Critique Through Self-Reflection*. Arxiv. Obtenido de <https://arxiv.org/abs/2310.11511>
- Ba, J. L. (2016). *Layer normalization*. arXiv.
- Brown, T. M. (2020). *Language Models are Few-Shot Learners*. arXiv. Obtenido de <https://arxiv.org/abs/2005.14165>
- Demis Hassabis, c. y. (s.f.).
- Devlin, J. C.-W. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv. Obtenido de <https://arxiv.org/abs/1810.04805>
- dge, D. T. (2024). *From local to global: A Graph RAG approach to query-focused summarization*. arXiv. Obtenido de <https://arxiv.org/abs/2404.16130>
- Gage, P. (1994). A New Algorithm for Data Compression. . *C Users Journal*.
- Harris, Z. S. (1954). Distributional Structure. *Word*, 146-162.
- He, P. G. (2023). *DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing*. *Proceedings of the International Conference on Learning Representations (ICLR)*. arXiv. Obtenido de <https://arxiv.org/abs/2111.09543>
- He, P. L. (2021). *DeBERTa: Decoding-enhanced BERT with disentangled attention*. *Proceedings of the International Conference on Learning Representations (ICLR)*. arXiv. Obtenido de <https://arxiv.org/abs/2006.03654>
- Hendrycks, D. &. (2023). *Gaussian Error Linear Units (GELUs)*. arXiv. Obtenido de <https://arxiv.org/abs/1606.08415>
- Hugging Face. (s.f.). *The AI community building the future*. Obtenido de <https://huggingface.co/>.
- Jurafsky, D. &. (2009). *Speech and Language Processing (2nd ed.)*. Prentice Hall.
- Jurafsky, D. &. (2021). *Speech and Language Processing (3rd ed.)*. Pearson.
- Kamradt, G. (8 de Enero de 2024). *YouTube*. Obtenido de <https://www.youtube.com/watch?v=8OJC21T2SL4&t=1930s>
- KeepCoding. (2023). *¿Qué es y en qué consiste la Ley de Zipf?* Obtenido de <https://keepcoding.io/blog/que-es-y-en-que-consiste-la-ley-de-zipf/>
- Labach, A. S. (2019). *Survey of dropout methods for deep neural networks*. . arXiv.
- LangGraph GitHub Documentation*. (2024). Obtenido de <https://github.com/langchain-ai/langgraph>

- Liu, Y. O. (2019). *RoBERTa: A robustly optimized BERT pretraining approach*. arXiv. Obtenido de <https://arxiv.org/abs/1907.11692>
- LlamaIndex. (s.f.). *LlamaParse: Servicio de Análisis de Documentos. Documentación de LlamaIndex*. Obtenido de https://docs.llamaindex.ai/en/stable/module_guides/loading/connector/llama_parse/
- Manning, C. D. (2008). *Introduction to Information Retrieval*. Cambridge University Press.
- Microsoft. (2024). *GraphRAG: A structured, hierarchical approach to retrieval augmented generation*. Microsoft. Obtenido de <https://microsoft.github.io/graphrag/>
- Pinecone. (August de 2024). *Hierarchical Navigable Small World (HNSW) Graphs*. Obtenido de <https://www.pinecone.io/learn/series/faiss/hnsw/>
- Ragas. (2024). *Ragas Documentation*. Obtenido de <https://docs.ragas.io/en/stable/concepts/metrics/index.html>
- RAGAS. (s.f.). *RAGAS documentation*. Obtenido de <https://docs.ragas.io/en/stable/index.html>
- Reimers, N. &. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. arXiv. Obtenido de <https://arxiv.org/abs/1908.10084>
- Salton, G. &. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill.
- Sarathi, P. A. (2024). *RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval*. Proceedings of the International Conference on Learning Representations (ICLR). Obtenido de <https://arxiv.org/abs/2401.18059>
- Schuster, M. &. (2012). Japanese and Korean voice search., (págs. 5149-5152). Obtenido de <https://doi.org/10.1109/ICASSP.2012.6289079>
- Sennrich, R. H. (2016). *Neural Machine Translation of Rare Words with Subword Units*. arXiv. Obtenido de <https://arxiv.org/abs/1508.07909>
- Srivastava, N. H. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 1929-1958.
- The AI Forum. (11 de julio de 2023). *RAG en PDF Complejos usando LlamaParse, LangChain, y Groq*. Obtenido de <https://medium.com/the-ai-forum/rag-on-complex-pdf-using-llamaparse-langchain-and-groq-5b132bd1f9f3>
- Vaswani, A. S. (2017). Attention Is All You Need. arXiv. Obtenido de <https://arxiv.org/abs/1706.03762>
- Yan, S.-Q. G.-C.-H. (2024). Corrective Retrieval Augmented Generation. *Proceedings of the 40th International Conference on Machine Learning*. National Engineering Research Center of Speech and Language Information Processing, University of Science and Technology of China; Department of Computer Science, University of California, Los Angeles; Google Research.

Procesamiento del BOE mediante una arquitectura compuesta por un sistema de RAG y en un agente basado en un modelo grande de lenguaje (LLM)



**Universidad
Europea**

Jorge Resino Martin

Capítulo 9. ANEXOS

9.1 Anexo I. BERT: *Pre-training of Deep Bidirectional Transformers for Language Understanding*

BERT, que significa "*Bidirectional Encoder Representations from Transformers*", es un modelo de representación de lenguaje desarrollado por Google que utiliza transformadores bidireccionales profundos para aprender representaciones contextuales de texto. A diferencia de los modelos de lenguaje tradicionales que se entrenan de manera unidireccional (por ejemplo, de izquierda a derecha o de derecha a izquierda), BERT se entrena bidireccionalmente, lo que significa que considera el contexto de ambas direcciones (izquierda y derecha) simultáneamente en todas sus capas.

9.1.1 Innovación y Contribuciones Clave

1. **Modelo Bidireccional:** La principal innovación de BERT es su capacidad para preentrenar representaciones bidireccionales profundas de texto, lo que permite que el modelo comprenda mejor el contexto de las palabras en relación con sus vecinos. Esto se logra utilizando un "modelo de lenguaje enmascarado" (*Masked Language Model*, MLM) que enmascara algunas palabras en una oración y entrena al modelo para predecirlas en función del contexto bidireccional.
2. **Preentrenamiento y Ajuste Fino:** BERT se entrena primero en grandes cantidades de datos no etiquetados (preentrenamiento) y luego se ajusta con tareas específicas con datos etiquetados (ajuste fino). Durante el preentrenamiento, BERT utiliza dos tareas: el modelo de lenguaje enmascarado (MLM) y la predicción de la siguiente oración (*Next Sentence Prediction*, NSP). Este enfoque permite que BERT se adapte a una amplia gama de tareas de procesamiento de lenguaje natural (NLP) sin necesidad de arquitecturas específicas para cada tarea.
3. **Estado del Arte en Tareas de NLP:** BERT ha establecido nuevos estándares en múltiples tareas de NLP, como la respuesta a preguntas y la inferencia de lenguaje natural. Sus resultados superan significativamente a modelos anteriores, incluyendo mejoras sustanciales en *benchmarks* como GLUE, SQuAD y otros.

9.1.2 Arquitectura y Entrenamiento

BERT utiliza una arquitectura de transformadores, basada en la implementación original descrita por Vaswani et al. (2017). En su versión básica, BERT cuenta con 12 capas de transformadores con un tamaño de ocultación de 768 y 12 cabezas de autoatención, lo que suma un total de 110 millones de parámetros. En su versión más grande, BERTLARGE, el modelo cuenta con 24 capas y un total de 340 millones de parámetros (Devlin, 2019).

9.1.3 Tareas de Preentrenamiento

1. *Masked Language Model* (MLM): Durante el preentrenamiento, BERT enmascara aleatoriamente un 15% de las palabras en una oración y entrena al modelo para predecir esas palabras basándose en el contexto de ambas direcciones.
2. *Next Sentence Prediction* (NSP): Para aprender la relación entre dos oraciones, BERT entrena también con una tarea de predicción de la siguiente oración, donde el modelo debe determinar si una oración sigue a otra en el texto original o si es una oración aleatoria.

9.1.4 Impacto y Uso de BERT

BERT ha revolucionado el campo del procesamiento del lenguaje natural al permitir que modelos generales de lenguaje se adapten eficazmente a tareas específicas con un mínimo ajuste fino. Esto ha permitido su adopción generalizada en diversas aplicaciones, desde motores de búsqueda hasta *chatbots* y sistemas de respuesta a preguntas.

9.2 Anexo II. *Sentence-BERT* (SBERT): *Embeddings* de Oraciones Usando Redes Siamese BERT.

Sentence-BERT (SBERT) es una modificación de la arquitectura BERT (*Bidirectional Encoder Representations from Transformers*), diseñada para crear *embeddings* (representaciones vectoriales densas) semánticamente significativos de oraciones de manera eficiente. BERT, por sí mismo, fue revolucionario por su rendimiento en tareas de procesamiento de lenguaje natural, pero no estaba optimizado para tareas que requieren *embeddings* de oraciones, como la similitud semántica, el agrupamiento o la recuperación de información. Esto se debe a que BERT se utiliza típicamente en una configuración "*cross-encoder*" donde ambas oraciones en un par se ingresan en el modelo simultáneamente, lo que es computacionalmente costoso.

9.2.1 Enfoque de SBERT

SBERT aborda esta ineficiencia utilizando una estructura de red siamesa, donde dos modelos BERT idénticos (con pesos compartidos) procesan cada oración de manera independiente. La innovación clave en SBERT es el uso de estas redes siamesas para producir *embeddings* de tamaño fijo para oraciones individuales, que luego se pueden comparar utilizando medidas de similitud, como la similitud del coseno.

9.2.2 Funcionamiento de SBERT

1. Estructuras de Redes Siamesas y Tripletes: En SBERT, se utilizan dos redes BERT idénticas para codificar dos oraciones por separado. Las salidas (*embeddings*) de estas redes se comparan utilizando la similitud del coseno para medir la similitud semántica. Esta estructura permite que SBERT maneje eficientemente tareas de similitud semántica a gran escala.
2. Estrategias de *Pooling*: SBERT añade una capa de *pooling* a la salida de BERT para crear un *embedding* de oración de tamaño fijo. Se experimenta con tres estrategias de

pooling: usando el token CLS, calculando la media de todos los vectores de salida (estrategia MEAN), y calculando el máximo a lo largo del tiempo de los vectores de salida (estrategia MAX).

3. Ajuste Fino en Datos de NLI: SBERT se ajusta en conjuntos de datos de Inferencia de Lenguaje Natural (NLI) para producir *embeddings* de oraciones que superan a otros métodos de última generación. El ajuste fino con NLI ayuda a generar *embeddings* donde las oraciones semánticamente similares están cercanas en el espacio vectorial.
4. Eficiencia: La eficiencia de SBERT se destaca por su capacidad para reducir significativamente la carga computacional. Por ejemplo, mientras que BERT tradicional requeriría aproximadamente 65 horas para encontrar el par más similar en una colección de 10,000 oraciones, SBERT reduce este tiempo a solo 5 segundos.
5. Aplicaciones: SBERT es particularmente útil para tareas que implican comparaciones de oraciones a gran escala, como el agrupamiento, la similitud textual semántica (STS) y la recuperación de información. Ha mostrado un rendimiento sólido en varios *benchmarks*, superando a modelos anteriores como InferSent y *Universal Sentence Encoder* en muchos casos.

9.2.3 Ventajas sobre BERT tradicional

- Eficiencia Computacional: SBERT reduce drásticamente el tiempo necesario para tareas que requieren comparar pares de oraciones.
- Mejores *Embeddings* de Oraciones: SBERT produce *embeddings* de oraciones de alta calidad que son semánticamente significativos, lo que lo hace adecuado para una amplia gama de tareas de NLP.

En resumen, *Sentence-BERT* representa un avance significativo en la generación de *embeddings* de oraciones al combinar las fortalezas de BERT con una arquitectura de red siamesa eficiente, permitiendo que funcione bien en tareas que requieren similitud semántica, agrupamiento y recuperación, siendo al mismo tiempo computacionalmente eficiente (Reimers, 2019).

9.3 Anexo III. *Byte Pair Encoding* (BPE)

Byte Pair Encoding (BPE) es un algoritmo de compresión de datos adaptado para el procesamiento de lenguaje natural (NLP) para realizar la “tokenización”, dividiendo palabras en subunidades más pequeñas llamadas subpalabras o tokens. Este enfoque es especialmente útil en modelos de lenguaje como GPT y BERT para manejar palabras desconocidas, reducir el tamaño del vocabulario, y mejorar la generalización (Sennrich, 2016).

9.3.1 Funcionamiento de BPE

1. Frecuencia de Pares: El algoritmo comienza por descomponer cada palabra en su secuencia de caracteres individuales. Luego, se identifican los pares de caracteres más frecuentes en el corpus de texto.
2. Unión de Pares: El par de caracteres más frecuente se combina en un solo token o subpalabra. Esta nueva secuencia se considera una nueva unidad en el vocabulario.

3. Repetición del Proceso: Este proceso de identificación y combinación se repite iterativamente hasta que se alcanza un número predeterminado de tokens o subpalabras en el vocabulario.
4. Manejo de Palabras Desconocidas: En inferencia o uso práctico, si una palabra completa no está presente en el vocabulario, el algoritmo puede dividirla en las subunidades más pequeñas posibles que existen en el vocabulario, lo que permite manejar palabras desconocidas o raras.

9.3.2 Ejemplo de aplicación simplificado de BPE

Se supone que se tienen las palabras "low", "lower", "newest", y "widest". Al comenzar, cada palabra se divide en caracteres:

- "low" -> l o w
- "lower" -> l o w e r
- "newest" -> n e w e s t
- "widest" -> w i d e s t

El algoritmo observa que el par más frecuente es "e s". Por lo tanto, combina "e s" en un solo token:

- "newest" -> n e w e s t
- "widest" -> w i d e s t

Luego, el proceso continúa con la siguiente pareja más frecuente.

9.3.3 Ventajas del BPE en NLP

1. Reducción del Tamaño del Vocabulario: Permite mantener un vocabulario compacto, ya que no necesita incluir cada palabra completa, sino solo las subpalabras o tokens más frecuentes.
2. Flexibilidad: Puede manejar palabras nuevas o raras al dividir las en subunidades conocidas.
3. Mejor Generalización: Como las subpalabras se reutilizan en diferentes palabras, el modelo puede generalizar mejor.

Byte Pair Encoding se ha convertido en un método estándar en muchos modelos de lenguaje modernos, incluyendo los modelos como GPT y BERT, debido a su capacidad para equilibrar la eficiencia de la "tokenización" y la cobertura de vocabulario (Gage, 1994).

9.4 Anexo IV. Algoritmo de "tokenización" *WordPiece*

WordPiece es un algoritmo de "tokenización" que se utiliza comúnmente en modelos de procesamiento de lenguaje natural (NLP), como BERT (*Bidirectional Encoder Representations from Transformers*). Este algoritmo descompone palabras en subunidades más pequeñas llamadas subpalabras o tokens, lo que permite manejar un vocabulario más eficiente y abordar palabras desconocidas o raras. *WordPiece* es similar al algoritmo *Byte Pair Encoding* (BPE) pero con algunas diferencias clave en cómo se construye el vocabulario (Schuster, 2012).

9.4.1 Funcionamiento de *WordPiece*

1. Inicialización del Vocabulario:
WordPiece comienza con un vocabulario inicial que contiene todos los caracteres individuales presentes en el corpus de entrenamiento, junto con un token especial [UNK] que se utiliza para representar cualquier subpalabra desconocida.
2. Construcción Iterativa del Vocabulario:
 - El algoritmo analiza el corpus de entrenamiento y evalúa todas las posibles subpalabras (pares de caracteres o subpalabras ya existentes) que se pueden formar combinando los elementos del vocabulario actual.
 - En cada iteración, WordPiece selecciona la subpalabra que maximiza la probabilidad de la secuencia de subpalabras en el corpus de entrenamiento. Esta probabilidad se calcula utilizando un modelo estadístico que tiene en cuenta la frecuencia de aparición de las subpalabras.
 - La subpalabra seleccionada se añade al vocabulario, y el proceso se repite hasta que se alcanza un tamaño de vocabulario predefinido.
3. Tokenización de Palabras Nuevas:
 - Durante la inferencia, cuando se encuentra una palabra que no está en el vocabulario completo, WordPiece descompone la palabra en la secuencia de subpalabras más larga posible que exista en el vocabulario.
 - Si no se puede dividir la palabra en subpalabras conocidas, el algoritmo recurre al token [UNK] para representar la palabra completa.

9.4.2 Ejemplo de aplicación simplificado de *WordPiece*

Suponiendo que se tiene la palabra "playing". Si "playing" no está en el vocabulario, pero las subpalabras "play" y "##ing" sí lo están (donde "##" indica un fragmento que no es una palabra completa):

- La palabra "playing" se dividiría en los tokens: ["play", "##ing"].

Este enfoque permite que el modelo maneje variaciones de palabras y formas desconocidas de manera más eficiente que si dependiera de un vocabulario basado en palabras completas.

9.4.3 Ventajas de *WordPiece*

Reducción del Tamaño del Vocabulario: Al representar palabras mediante subpalabras comunes, WordPiece permite que un modelo mantenga un vocabulario compacto.

Manejo de Palabras Desconocidas: Las palabras raras o desconocidas pueden ser descompuestas en subpalabras que tienen significado, lo que mejora la capacidad del modelo para generalizar.

Optimización de la Probabilidad: WordPiece selecciona las subpalabras que maximizan la probabilidad de la secuencia en el corpus de entrenamiento, lo que ayuda a mejorar la eficiencia y precisión del modelo.

9.4.4 Diferencias con BPE

Aunque WordPiece y BPE comparten similitudes, una diferencia clave es cómo seleccionan las subpalabras durante la construcción del vocabulario. BPE se basa en la frecuencia de los pares de caracteres, mientras que *WordPiece* utiliza la probabilidad conjunta de las subpalabras en el corpus de entrenamiento, lo que puede conducir a un vocabulario más optimizado (Schuster, 2012).

9.5 Anexo V. DeBERTa: *Decoding-enhanced BERT with disentangled attention*

El modelo DeBERTa (*Decoding-enhanced BERT with disentangled attention*) es una arquitectura de lenguaje natural basada en Transformers que mejora los modelos BERT y RoBERTa mediante dos técnicas novedosas: la atención desentrelazada (*disentangled attention*) y un decodificador de máscara mejorado (*enhanced mask decoder*).

9.5.1 Atención des entrelazada (*Disentangled Attention*)

A diferencia de BERT, donde cada palabra se representa con un vector que combina su contenido y su posición, DeBERTa utiliza dos vectores separados para representar el contenido y la posición de cada palabra. Estos vectores permiten calcular los pesos de atención entre palabras mediante matrices des entrelazadas que consideran tanto el contenido como las posiciones relativas. Esto es particularmente útil porque la dependencia entre palabras no solo se basa en su contenido, sino también en su proximidad relativa en la oración. Esta atención des entrelazada se descompone en cuatro componentes principales:

- Contenido a Contenido (Content-to-Content): Considera cómo un par de palabras se influyen mutuamente basándose en su significado.
- Contenido a Posición (Content-to-Position): Evalúa cómo la posición de una palabra afecta la influencia de su contenido en otra palabra.
- Posición a Contenido (Position-to-Content): Considera cómo la posición de una palabra afecta su relevancia para otras palabras.
- Posición a Posición (Position-to-Position): Evalúa la interacción entre las posiciones relativas de dos palabras (este componente se elimina en la implementación de DeBERTa por ser menos relevante).

Esta separación permite que DeBERTa capture mejor las relaciones sintácticas complejas entre palabras, mejorando la capacidad del modelo para entender y generar lenguaje natural.

9.5.2 Decodificador de máscara mejorado (*Enhanced Mask Decoder*)

Similar a BERT, DeBERTa se entrena utilizando la técnica de *masked language modeling* (MLM), donde se predicen palabras ocultas basándose en el contexto proporcionado por el resto de la oración. Sin embargo, DeBERTa introduce mejoras al incorporar las posiciones absolutas de las palabras justo antes de la capa de *softmax* en el decodificador. Esto es crucial porque las

posiciones absolutas pueden aportar información sintáctica importante, como en el caso de determinar el sujeto de una oración. Esta mejora permite que DeBERTa distinga mejor entre palabras con contextos similares, pero funciones sintácticas diferentes.

9.5.3 Entrenamiento adversarial virtual para afinado (*Virtual Adversarial Training*)

DeBERTa también incorpora un método de entrenamiento adversarial virtual (SiFT, *Scale-invariant Fine-Tuning*) durante la fase de afinado del modelo. Este método mejora la generalización del modelo al hacerlo más robusto frente a perturbaciones adversariales en los datos de entrada, lo que es especialmente útil para modelos grandes con miles de millones de parámetros.

9.5.4 Resultados y Desempeño

DeBERTa muestra mejoras significativas en diversas tareas de procesamiento del lenguaje natural (NLP), superando a modelos como RoBERTa y XLNet en *benchmarks* clave como MNLI, SQuAD y SuperGLUE. En particular, DeBERTa se destaca al superar por primera vez el rendimiento humano en el *benchmark* SuperGLUE con un modelo de 1.5 mil millones de parámetros.

En resumen, DeBERTa representa un avance significativo en el diseño de modelos de lenguaje natural al mejorar la forma en que los modelos representan y utilizan la información posicional y de contenido, lo que se traduce en un rendimiento superior en una amplia gama de tareas de NLP (He, 2021).

9.6 Anexo VI. RoBERTa: *Robustly Optimized BERT Pretraining Approach*

El modelo RoBERTa (*A Robustly Optimized BERT Pretraining Approach*) es una variante optimizada de BERT (*Bidirectional Encoder Representations from Transformers*) que se creó para abordar algunas limitaciones del entrenamiento original de BERT y mejorar su rendimiento en tareas de procesamiento del lenguaje natural (NLP).

9.6.1 Arquitectura de RoBERTa

RoBERTa mantiene la misma arquitectura básica de BERT, utilizando la estructura de *transformer* introducida por Vaswani et al. (2017). Este transformador es un modelo basado en autoatención que tiene varias capas de autoatención, seguidas de capas *feedforward*. La arquitectura de BERT, y por ende la de RoBERTa, consta de múltiples capas de *transformer*, cada una de las cuales aplica autoatención a los tokens de entrada y luego pasa la información a través de una red neuronal totalmente conectada.

Específicamente, en RoBERTa, se utilizan varias configuraciones del modelo, como RoBERTa-BASE y RoBERTa-LARGE, que difieren en el número de capas (12 y 24 respectivamente), el tamaño de los vectores de representación (768 y 1024 dimensiones), y el número de cabezas de atención.

9.6.2 Entrenamiento de RoBERTa

RoBERTa introduce varias mejoras clave en el proceso de entrenamiento en comparación con BERT:

- Eliminación de la predicción de la siguiente oración (NSP): En BERT, una de las tareas de preentrenamiento es la predicción de si dos oraciones son consecutivas en el texto original. RoBERTa elimina esta tarea, ya que estudios posteriores han sugerido que no contribuye significativamente al rendimiento y podría incluso limitar la capacidad del modelo.
- Entrenamiento con secuencias más largas: RoBERTa se entrena utilizando secuencias más largas que las usadas en BERT. Esto permite al modelo captar dependencias a largo plazo en el texto, lo que es esencial para entender contextos complejos.
- Mayor tamaño del conjunto de datos: Una de las diferencias más notables es que RoBERTa se entrena en un conjunto de datos significativamente más grande. Además de los datos originales utilizados para entrenar BERT (BookCorpus y Wikipedia), RoBERTa utiliza datos adicionales como CC-News, OpenWebText, y Stories, sumando un total de 160 GB de texto.
- Ajuste dinámico de la máscara: En lugar de usar un enmascaramiento estático, donde las palabras seleccionadas para ser enmascaradas se fijan al inicio del entrenamiento, RoBERTa emplea un enmascaramiento dinámico, lo que significa que el conjunto de palabras enmascaradas se selecciona de nuevo en cada época del entrenamiento. Esto introduce una variabilidad que mejora la robustez del modelo.
- Uso de mini-lotes grandes: RoBERTa entrena con mini-lotes significativamente más grandes que los de BERT, lo que, combinado con el uso de una tasa de aprendizaje ajustada, mejora tanto la velocidad de convergencia como el rendimiento final del modelo.

9.6.3 Resultados y Contribuciones

Gracias a estas optimizaciones, RoBERTa logra superar los resultados de BERT en varias tareas estándar de NLP, incluyendo los *benchmarks* GLUE, RACE, y SQuAD, alcanzando o superando el estado del arte en múltiples tareas. Las pruebas realizadas demostraron que RoBERTa, a pesar de utilizar la misma arquitectura y objetivo de preentrenamiento que BERT, obtiene mejoras significativas gracias a las estrategias de entrenamiento optimizadas.

En resumen, RoBERTa es un modelo mejorado y robusto que refina la metodología de preentrenamiento de BERT, optimizando diversos aspectos clave del entrenamiento para lograr un rendimiento superior en tareas de NLP (Liu, 2019).

9.7 Anexo VII. Función GELU (*Gaussian Error Linear Unit*)

La función GELU (*Gaussian Error Linear Unit*) es una función de activación utilizada en redes neuronales que ha mostrado un rendimiento superior en comparación con otras funciones de activación más tradicionales como ReLU (*Rectified Linear Unit*) y ELU (*Exponential Linear Unit*). La GELU se caracteriza por su capacidad para realizar una transformación no lineal que tiene en cuenta la naturaleza probabilística de los datos de entrada.

9.7.1 Definición y Funcionamiento de GELU

La función GELU se define matemáticamente como:

$$\text{GELU}(x) = x \cdot \Phi(x)$$

$\Phi(x)$ es la función de distribución acumulativa de la distribución normal estándar. Esto significa que en lugar de simplemente truncar o activar las entradas basándose en un umbral fijo, como lo hace la ReLU, la GELU pondera las entradas por su valor, aplicando una transformación suave que toma en cuenta cuán probable es que el valor de entrada sea positivo o negativo según una distribución normal.

La versión aproximada de la GELU se puede expresar como:

$$\text{GELU}(x) \approx 0.5x(1 + \tanh[\sqrt{\pi/2}(x + 0.044715x^3)])$$

Esta aproximación es más rápida de calcular y es comúnmente utilizada en implementaciones prácticas.

9.7.2 Utilidad y Beneficios

- **Curvatura y No Linealidad:** A diferencia de ReLU, que es lineal en su dominio positivo, la GELU introduce curvatura en toda la función, lo que permite a las redes neuronales modelar funciones más complejas y no lineales con mayor precisión.
- **Probabilística y Suavidad:** La GELU no es una función monotónica ni convexa, lo que significa que puede capturar relaciones más complejas en los datos. Además, su suavidad y la dependencia de una distribución normal la hacen menos propensa a problemas de saturación que otras funciones como la ReLU.
- **Mejor Rendimiento en Tareas de NLP:** La GELU ha demostrado ser particularmente efectiva en modelos de lenguaje natural, como BERT y GPT, donde ha reemplazado a funciones de activación más tradicionales y se ha convertido en el estándar para los transformadores de última generación.

9.7.3 Comparación con otras Funciones de Activación

- **ReLU:** Mientras que ReLU simplemente activa valores positivos y deja a cero los negativos, la GELU pondera los valores de entrada de forma probabilística, permitiendo una activación más suave y gradual.
- **ELU:** Aunque ELU también permite valores negativos, la GELU ofrece una suavidad adicional y un comportamiento no lineal que puede mejorar la capacidad de generalización de la red.

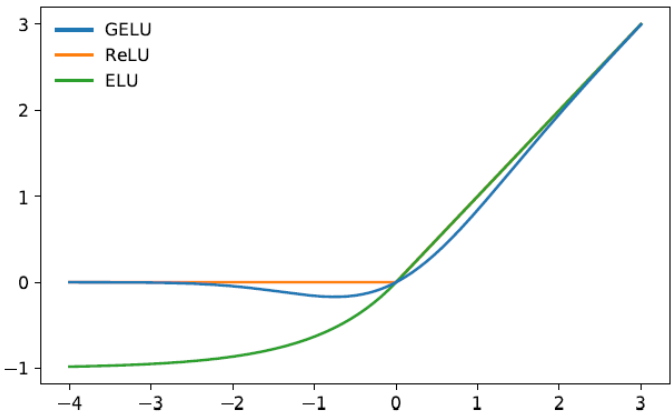


Ilustración 31: Comparativa de las funciones de activación RELU, GELU y ELU.

En resumen, la función GELU proporciona una alternativa más sofisticada y efectiva a las funciones de activación tradicionales, especialmente en arquitecturas de redes profundas utilizadas en procesamiento del lenguaje natural y visión por computadora. Su adopción en modelos avanzados como BERT subraya su eficacia y capacidad para mejorar el rendimiento en tareas complejas (Hendrycks, 2023).

9.8 Anexo VIII. Configuración del modelo DeBERTa V3

La configuración utilizada del modelo LLM tipo BERT ha sido la siguiente. Cabe destacar las 41 categorías elegidas para la clasificación de textos del BOE.

```
DebertaV2Config {
  "_name_or_path": "microsoft/deberta-v3-base",
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "Leyes Orgánicas",
    "1": "Reales Decretos y Reales Decretos-Leyes",
    "2": "Tratados y Convenios Internacionales",
    "3": "Leyes de Comunidades Autónomas",
    "4": "Reglamentos y Normativas Generales",
    "5": "Nombramientos y Ceses",
```

"6": "Promociones y Situaciones Especiales",
"7": "Convocatorias y Resultados de Oposiciones",
"8": "Anuncios de Concursos y Adjudicaciones de Plazas",
"9": "Ayudas",
"10": "Subvenciones y Becas",
"11": "Convenios Colectivos y Cartas de Servicio",
"12": "Planes de Estudio y Normativas Educativas",
"13": "Convenios Internacionales y Medidas Especiales",
"14": "Edictos y Notificaciones Judiciales",
"15": "Procedimientos y Citaciones Judiciales",
"16": "Licitaciones y Adjudicaciones Públicas",
"17": "Avisos y Notificaciones Oficiales",
"18": "Anuncios Comerciales y Convocatorias Privadas",
"19": "Sentencias y Autos del Tribunal Constitucional",
"20": "Orden de Publicaciones y Sumarios",
"21": "Publicaciones por Órgano Emisor",
"22": "Jerarquía y Autenticidad de Normativas",
"23": "Publicaciones en Lenguas Cooficiales",
"24": "Interpretaciones y Documentos Oficiales",
"25": "Informes y Comunicaciones de Interés General",
"26": "Documentos y Estrategias Nacionales",
"27": "Medidas de Emergencia y Seguridad Nacional",
"28": "Anuncios de Regulaciones Específicas",
"29": "Normativas Temporales y Urgentes",
"30": "Medidas y Políticas Sectoriales",
"31": "Todos los Tipos de Leyes (Nacionales y Autonómicas)",
"32": "Todos los Tipos de Decretos (Legislativos y no Legislativos)",
"33": "Convocatorias y Resultados Generales (Empleo y Educación)",
"34": "Anuncios y Avisos (Oficiales y Privados)",
"35": "Judicial y Procedimientos Legales",

```
"36": "Sentencias y Declaraciones Judiciales",
"37": "Publicaciones Multilingües y Cooficiales",
"38": "Informes y Estrategias de Política",
"39": "Emergencias Nacionales y Medidas Excepcionales",
"40": "Documentos y Comunicaciones Específicas"
},
"initializer_range": 0.02,
"intermediate_size": 3072,
"label2id": {
  "Anuncios Comerciales y Convocatorias Privadas": 18,
  "Anuncios de Concursos y Adjudicaciones de Plazas": 8,
  "Anuncios de Regulaciones Específicas": 28,
  "Anuncios y Avisos (Oficiales y Privados)": 34,
  "Avisos y Notificaciones Oficiales": 17,
  "Ayudas": 9,
  "Convenios Colectivos y Cartas de Servicio": 11,
  "Convenios Internacionales y Medidas Especiales": 13,
  "Convocatorias y Resultados Generales (Empleo y Educación)": 33,
  "Convocatorias y Resultados de Oposiciones": 7,
  "Documentos y Comunicaciones Específicas": 40,
  "Documentos y Estrategias Nacionales": 26,
  "Edictos y Notificaciones Judiciales": 14,
  "Emergencias Nacionales y Medidas Excepcionales": 39,
  "Informes y Comunicaciones de Interés General": 25,
  "Informes y Estrategias de Política": 38,
  "Interpretaciones y Documentos Oficiales": 24,
  "Jerarquía y Autenticidad de Normativas": 22,
  "Judicial y Procedimientos Legales": 35,
  "Leyes Orgánicas": 0,
  "Leyes de Comunidades Autónomas": 3,
```

```
"Licitaciones y Adjudicaciones Públicas": 16,  
"Medidas de Emergencia y Seguridad Nacional": 27,  
"Medidas y Políticas Sectoriales": 30,  
"Nombramientos y Ceses": 5,  
"Normativas Temporales y Urgentes": 29,  
"Orden de Publicaciones y Sumarios": 20,  
"Planes de Estudio y Normativas Educativas": 12,  
"Procedimientos y Citaciones Judiciales": 15,  
"Promociones y Situaciones Especiales": 6,  
"Publicaciones Multilingües y Cooficiales": 37,  
"Publicaciones en Lenguas Cooficiales": 23,  
"Publicaciones por Órgano Emisor": 21,  
"Reales Decretos y Reales Decretos-Leyes": 1,  
"Reglamentos y Normativas Generales": 4,  
"Sentencias y Autos del Tribunal Constitucional": 19,  
"Sentencias y Declaraciones Judiciales": 36,  
"Subvenciones y Becas": 10,  
"Todos los Tipos de Decretos (Legislativos y no Legislativos)": 32,  
"Todos los Tipos de Leyes (Nacionales y Autonómicas)": 31,  
"Tratados y Convenios Internacionales": 2  
},  
"layer_norm_eps": 1e-07,  
"max_position_embeddings": 512,  
"max_relative_positions": -1,  
"model_type": "deberta-v2",  
"norm_rel_ebd": "layer_norm",  
"num_attention_heads": 12,  
"num_hidden_layers": 12,  
"pad_token_id": 0,  
"pooler_dropout": 0,
```



```
"pooler_hidden_act": "gelu",
"pooler_hidden_size": 768,
"pos_att_type": [
  "p2c",
  "c2p"
],
"position_biased_input": false,
"position_buckets": 256,
"relative_attention": true,
"share_att_key": true,
"transformers_version": "4.42.4",
"type_vocab_size": 0,
"vocab_size": 128100
}
```

9.9 Anexo IX. Archivos de configuración tipo JSON de la herramienta.

9.9.1 Archivo *download.json*

Archivo de configuración que se encarga de definir en que ruta de un repositorio local depositar los archivos descargados de la web del gobierno, qué cantidad máxima de archivos PDF del BOE descargar y entre que fechas descargar estos.

```
{
  "local_dir" : "./data",
  "fecha_desde" : "2024-04-15",
  "fecha_hasta": "2024-04-21",
  "batch":999
}
```

9.9.2 Archivo *etl.json*

Archivo de configuración que se encarga de orquestar el tipo de proceso de extracción, transformación, procesamiento y carga de los textos en los archivos PDF del BOE.

```
{
  "parser": {
    "directory_path": "./data/boe/dias/2024/04/prueba",
    "file_type": ".pdf",
    "recursive_parser": true,
    "result_type": "markdown",
    "verbose": true
  },
  "splitter": {
    "chunk_size": 200,
    "storage_path": "./data/figures/splitter",
    "embedding_model": "sentence-transformers/paraphrase-multilingual-MiniLM-L12-
v2",
    "tokenizer_model": "meta-llama/Meta-Llama-3-8B",
    "threshold": 60,
    "max_tokens": 500,
    "verbose": 1,
    "buffer_size": 3,
    "max_big_chunks": 10,
    "splitter_mode": "CUSTOM",
    "min_initial_chunk_len": 100
  },
  "TextPreprocess": {
    "spc_characters": [
      "!", "@", "$", "%", "^", "&", "*", "(", ")", "-", "_", "=", "+",
      "{", "}", "[", "]", "|", "\\", ":", ";", "\"", "'", "<", ">", ",", ".",
      "?", "/", "~", "`", "\n", "\r", "\t", "\b", "\f", "___"
    ],
    "spc_words": [
      ""
    ]
  }
}
```

```
    ],
    "task_name": "classification",
    "methods": {
        "del_stopwords": {
            "apply": false,
            "lang": "Spanish"
        },
        "del_urls": {"apply": false},
        "del_html": {"apply": false},
        "del_emojis": {"apply": false},
        "del_special": {"apply": false},
        "del_special_words": {"apply": true},
        "del_digits": {"apply": false},
        "del_chinese_japanese": {"apply": false},
        "del_extra_spaces": {"apply": false},
        "get_lower": {"apply": false},
        "get_alphanumeric": {"apply": false},
        "stem": {"apply": false},
        "lemmatize": {"apply": false},
        "custom_del": {
            "apply": true,
            "delete": false,
            "plot": true,
            "storage_path": "./data/figures/text"
        },
        "bow": {
            "apply": true,
            "storage_path": "./data/figures/text/bow"
        },
        "bow_tf_idf": {
```

```
        "apply":true,
        "storage_path":"./data/figures/text/bow_tf_idf"
    }
}
},
"label_generator": {
    "tokenizer_model": "GPT35",
    "labels": null,
    "model": "GPT",
    "max_samples": 9999
},
"storer": {
    "store_path": "./data/boedataset",
    "file_name":"boe_id_BOE-A-2024-7296",
    "file_format": "csv"
},
"google_sheet_database": {
    "api_call_max_tries": 10
}
}
```

9.9.3 Archivo *graph.json*

Archivo de configuración que se encarga de definir los modelos LLM para cada agente o nodo del grafo lógico, así como de la temperatura de cada modelo para lograr una mayor o menor objetividad de estos en las respuestas. En este archivo también se definen los parámetros de la base de datos vectorial utilizada y el modelo de *embeddings* de esta.

```
{
  "agents":{
    "docs_grader": {
      "name":"OPENAI",
      "temperature":0
```

```
    },
    "query_processor": {
        "name": "OPENAI",
        "temperature": 0
    },
    "generator": {
        "name": "OPENAI",
        "temperature": 0
    },
    "hallucination_grader": {
        "name": "OPENAI",
        "temperature": 0
    },
    "answer_grader": {
        "name": "NVIDIA",
        "temperature": 0
    }
},
"vector_db": {
    "client": "qdrant",
    "hg_embedding_model": "sentence-transformers/paraphrase-multilingual-
MiniLM-L12-v2",
    "k": 3
},
"iteraciones": 10,
"thread_id": "4",
"verbose": 0
}
```

9.10 Anexo X. RAPTOR (*Recursive Abstractive Processing for Tree-Organized Retrieval*).

RAPTOR (*Recursive Abstractive Processing for Tree-Organized Retrieval*) es un sistema novedoso diseñado para mejorar la recuperación de información en modelos de lenguaje, especialmente en contextos donde es necesario manejar documentos largos o complejos. El objetivo principal de RAPTOR es superar las limitaciones de los métodos de recuperación tradicionales que suelen trabajar con fragmentos de texto contiguos y cortos, lo cual puede limitar la comprensión del contexto completo de un documento.

9.10.1 Funcionamiento de RAPTOR

RAPTOR construye un árbol jerárquico de resúmenes a partir de un corpus de texto. El proceso comienza con la segmentación del texto en fragmentos pequeños, que son representados como nodos hoja en el árbol. Estos fragmentos son luego incrustados en vectores utilizando un modelo de lenguaje como SBERT. A partir de estos vectores, RAPTOR agrupa los fragmentos similares utilizando un algoritmo de *clustering*, y cada grupo es resumido usando un modelo de lenguaje. Este proceso de agrupación y resumen se repite de manera recursiva, creando múltiples niveles de abstracción en el árbol, desde detalles granulares en las hojas hasta conceptos más generales en los nodos superiores.

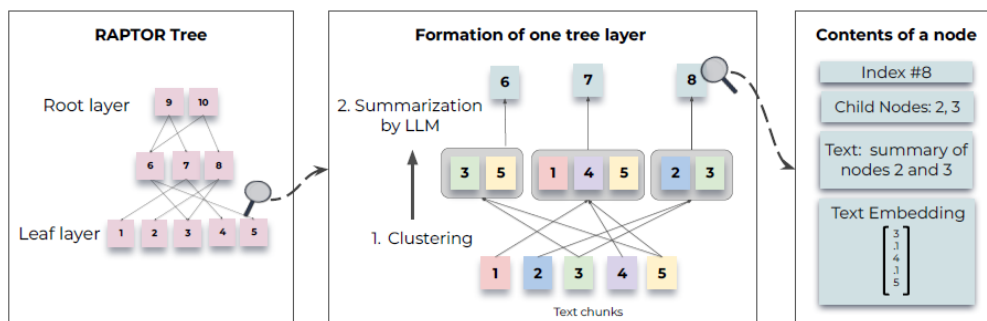


Ilustración 32: Tree construction process.

9.10.2 Recuperación de Información

Cuando se realiza una consulta, RAPTOR ofrece dos estrategias principales para la recuperación de información: la travesía del árbol y el árbol colapsado. La travesía del árbol implica recorrer el árbol desde la raíz hasta las hojas, seleccionando los nodos más relevantes en cada nivel basándose en la similitud con la consulta. Por otro lado, la estrategia del árbol colapsado implica tratar todos los nodos del árbol como si estuvieran en un solo nivel, lo que permite una comparación directa y simultánea de todos los nodos. Este enfoque ha demostrado ser más flexible y efectivo en la recuperación de información relevante.

9.10.3 Beneficios

RAPTOR permite integrar información de diferentes niveles de detalle en una consulta, lo cual es particularmente útil para preguntas que requieren una comprensión profunda y multiescalar de un texto. Los experimentos han demostrado que RAPTOR supera los métodos tradicionales de recuperación, logrando mejoras significativas en tareas de preguntas y respuestas que requieren razonamiento complejo.

Este sistema es especialmente útil en dominios como el análisis de documentos extensos (por ejemplo, libros o artículos científicos), donde es crucial tener en cuenta tanto los detalles finos como las ideas generales para proporcionar respuestas precisas y contextualmente relevantes (Sarthi, 2024).

9.11 Anexo XI. HNSW (*Hierarchical Navigable Small World*).

El algoritmo de *Hierarchical Navigable Small World* (HNSW) es uno de los métodos más avanzados y eficientes para la búsqueda de similitud de vectores, específicamente en el contexto de búsquedas de vecinos más cercanos aproximados (ANN, por sus siglas en inglés). Este algoritmo se utiliza ampliamente en bases de datos vectoriales como *Pinecone* debido a su capacidad para ofrecer velocidades de búsqueda extremadamente rápidas y altos niveles de precisión en la recuperación de información basada en vectores.

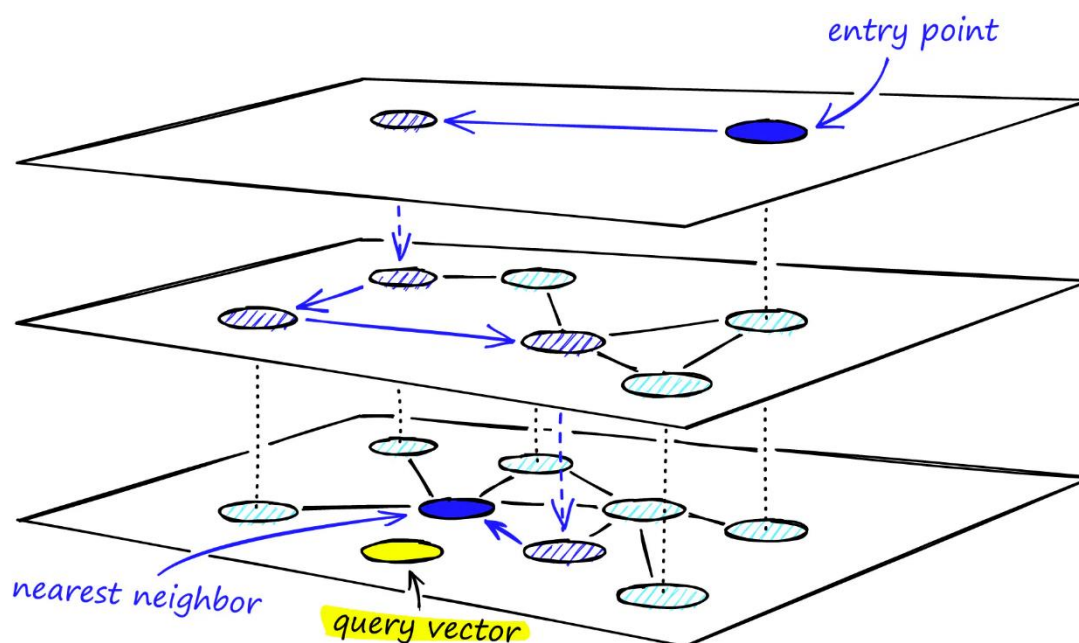


Ilustración 33: Algoritmo HNSW.

9.11.1 Fundamentos de HNSW

HNSW se enmarca en la categoría de algoritmos de búsqueda aproximada que utilizan grafos de proximidad. En estos grafos, los vértices están conectados entre sí en función de su proximidad, generalmente medida en términos de distancia euclidiana. Sin embargo, HNSW va más allá de

un simple grafo de proximidad al incorporar dos técnicas fundamentales: las listas de omisión probabilísticas y los grafos de mundo pequeño navegables.

- *Probability Skip List* (Lista de omisión probabilística):
Esta estructura de datos fue introducida en 1990 y permite realizar búsquedas rápidas similares a las de un array ordenado, pero utilizando una estructura de lista enlazada que facilita la inserción rápida de nuevos elementos. HNSW hereda esta estructura de capas donde las capas superiores contienen enlaces más largos (para búsquedas rápidas) y las capas inferiores tienen enlaces más cortos (para búsquedas más precisas).
- *Navigable Small World Graphs* (Grafos de mundo pequeño navegables):
En estos grafos, cada vértice está conectado a varios otros vértices, denominados "amigos", y el proceso de búsqueda comienza en un punto de entrada predefinido. El algoritmo busca de manera codiciosa el vértice más cercano al vector de consulta, moviéndose a través del grafo hasta que no encuentra un vértice más cercano, lo que marca el punto de detención.

9.11.2 Creación y Estructura de HNSW

HNSW es una evolución natural de los grafos de mundo pequeño navegables (NSW), a los cuales se les añade una jerarquía de capas inspirada en la estructura de las listas de omisión probabilísticas. En HNSW, las conexiones se dividen en diferentes capas: en las capas superiores se encuentran los enlaces más largos, mientras que en las inferiores se ubican los enlaces más cortos y numerosos.

Durante la búsqueda, el algoritmo comienza en la capa superior, que actúa como un "zoom-out", permitiendo movimientos rápidos a través del grafo mediante vértices con enlaces más largos. A medida que desciende a capas inferiores, el algoritmo entra en una fase de "zoom-in", donde los movimientos son más precisos, pero más lentos, hasta encontrar el mínimo local en la capa más baja.

9.11.3 Construcción del Grafo HNSW

La construcción del grafo HNSW es un proceso iterativo donde los vectores se insertan uno a uno. El número de capas (L) y la probabilidad de inserción en una capa específica están controlados por funciones probabilísticas, ajustadas para balancear la precisión de búsqueda y la complejidad del grafo.

El proceso de inserción comienza en la capa superior, y tras encontrar los vecinos más cercanos a un nuevo vector, se agregan enlaces en todas las capas hasta la capa de inserción seleccionada. Los parámetros como M , M_{\max} , y $M_{\max 0}$ determinan el número máximo de enlaces que puede tener un vértice, afectando la estructura y el rendimiento del grafo.

9.11.4 Desempeño de HNSW

HNSW es conocido por su excelente desempeño en términos de rapidez de búsqueda y alta precisión (*recall*), aunque este rendimiento viene acompañado de un uso considerable de

memoria. Para optimizar el uso de memoria y los tiempos de búsqueda, se pueden aplicar técnicas como la cuantización de producto (PQ) o la adición de componentes IVF.

En resumen, HNSW combina conceptos avanzados de grafos y estructuras jerárquicas para lograr búsquedas vectoriales altamente eficientes, siendo una de las opciones más robustas para aplicaciones de búsqueda de similitud de alta precisión (Pinecone, 2024).

[PÁGINA INTENCIONADAMENTE EN BLANCO]