GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

Desarrollo Sistema RAG

Autor: Sofía Negueruela Avellaneda

Director: Nicolás Pérez Beltrán

Madrid

Mayo de 2026

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Desarrollo Sistema RAG

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2025/26 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Sofía Negueruela Avellaneda Fecha: 25/05/2026

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Nicolás Pérez Beltrán Fecha: 25/05/2026

GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

Desarrollo Sistema RAG

Autor: Sofía Negueruela Avellaneda

Director: Nicolás Pérez Beltrán

Madrid

DESARROLLO SISTEMA RAG

Autor: Negueruela Avellaneda, Sofía.
Director: Pérez Beltrán, Nicolás.
Entidad Colaboradora: EY

RESUMEN DEL PROYECTO

Este trabajo implementa y evalúa un sistema *Retrieval-Augmented Generation* (RAG) modular y multiidioma para mejorar la fiabilidad de asistentes basados en LLMs en un contexto empresarial. El punto de partida es una limitación observada en un caso real: respuestas parciales y alucinaciones al consultar repositorios documentales heterogéneos. La propuesta combina un RAG clásico con dos extensiones orientadas a problemas concretos: **grafos de conocimiento**, para información relacional distribuida; y **procesamiento en paralelo**, para preguntas que exigen respuestas extensas y completas. Los resultados muestran mejoras en completitud y trazabilidad con un coste computacional controlado.

Palabras clave: *Retrieval-Augmented Generation*, GraphRAG, sistema multiidioma, recuperación híbrida, Azure, Neo4j, evaluación experimental

1. Introducción

El trabajo se enmarca en el ámbito de la inteligencia artificial generativa aplicada a recuperación de información. Aunque los modelos de lenguaje de gran tamaño (LLMs) ofrecen alta capacidad de redacción y síntesis, presentan algunas limitaciones: conocimiento potencialmente desactualizado, baja especialización en datos internos y dificultades de trazabilidad [2] [4]. Los sistemas RAG abordan parte de este problema al incorporar evidencia externa en tiempo de consulta antes de generar la respuesta [1] [2]. En escenarios reales, no obstante, aún existen dos retos: la recuperación top-k puede ser insuficiente para preguntas de alta cobertura y la similitud vectorial no siempre reconstruye bien relaciones distribuidas entre documentos [2] [3] [8] [13]. Por ello, este trabajo propone y evalúa una arquitectura modular que permite comparar estrategias RAG y mejorar calidad de respuesta sin perder viabilidad operativa [9] [10] [11].

2. Objetivos

El objetivo general es desarrollar y evaluar un sistema RAG extensible y multiidioma que permita comparar distintas estrategias de recuperación y generación en varios tipos de documentos, priorizando tres criterios: calidad de respuesta, trazabilidad y eficiencia [2] [4]. Los objetivos específicos son:

- Implementar un flujo de ingesta documental robusto con un proceso de *chunking* adaptado por caso de uso
- Habilitar la recuperación híbrida (léxica, vectorial y semántica) con filtrado por idioma y configuración por caso de uso [5] [6] [7] [9]
- Incorporar una estrategia GraphRAG basada en Neo4j y Graphiti para recuperar evidencia estructural cuando la pregunta del usuario requiere relacionar documentos inconexos y obtener información distribuida [8] [12] [13]
- Diseñar un procesamiento en paralelo para preguntas que exigen alta cobertura, reduciendo la pérdida de información que suelen presentar los enfoques top-k

- Evaluar y comparar RAG clásico, GraphRAG y procesamiento en paralelo mediante protocolos replicables y *gold standards*
- Registrar métricas (número de *chunks* consultados, tiempo, modelo empleado y coste) para facilitar trazabilidad y análisis posteriores

La pregunta central que guía el trabajo es: ¿qué combinación de estrategias de recuperación mejora la completitud y la fiabilidad de las respuestas en escenarios empresariales reales, manteniendo un coste computacional razonable? [2] [4]

3. Descripción del sistema

La solución se estructura en dos fases desacopladas: fase offline (ingesta e indexación) y fase online (consulta, recuperación y generación) [2] [4]. En la fase **offline**, se generan o descargan los documentos según el caso de uso (CVs, Wikipedia y Legislación UE), se normalizan y dividen en *chunks* con metadatos (idioma, fuente, página, *timestamp* y atributos de dominio). Los *chunks* se almacenan en Azure Cosmos DB y se sincronizan hacia Azure AI Search mediante *indexers*, con un índice único por caso de uso [9] [10]. En la fase **online**, el *backend* recibe la pregunta del usuario, utiliza el idioma seleccionado y aplica una de las tres estrategias disponibles:

- **RAG clásico**: recuperación híbrida en Azure AI y generación de la respuesta [1] [5] [6] [7] [9]
- **GraphRAG**: consulta el grafo de conocimiento en Neo4j mediante Graphiti, recuperando episodios, entidades y hechos para aportar contexto relacional [8] [12] [13]
- **Procesamiento en paralelo**: ampliación del conjunto recuperado en la búsqueda, agrupación por tramos de fiabilidad y validación en paralelo con modelos ligeros (GPT-5-mini), seguida de consolidación final con un LLM de mayor capacidad

La implementación se desarrolla con Python y FastAPI para el *backend* junto con orquestación basada en LangChain; y con un *frontend* de React para pruebas funcionales. A nivel de infraestructura, se emplean Azure Blob Storage como capa de persistencia, Azure Cosmos DB como repositorio de datos, Azure AI Search como plano de consulta y servicios de modelos desplegados en Azure AI Foundry o Azure AI OpenAI [9] [10] [11].

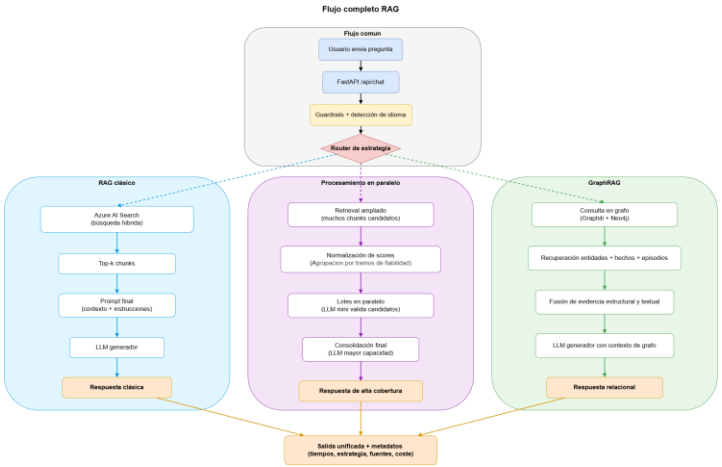

Ilustración 1 - Diagrama del flujo completo con las tres estrategias

La arquitectura modular permite desacoplar persistencia, recuperación y generación, simplificando el mantenimiento y facilitando la experimentación [2] [4]. Además, el sistema incorpora capacidades multiidioma [2] [9].

4. Resultados

La validación se plantea como una comparativa experimental entre estrategias sobre tres casos de uso y varios idiomas. Para cada escenario se definen preguntas de prueba y respuestas de referencia (*gold standard*), con evaluación de calidad de respuesta y análisis de coste y latencia [2] [4]. Los resultados principales son:

- En preguntas de cobertura, la estrategia RAG con procesamiento en paralelo mejora la exhaustividad frente al RAG clásico, pero presenta mayor latencia y coste [2] [3]

- Por su parte, GraphRAG aporta mayor coherencia global y mejor justificación estructural, al incorporar relaciones explícitas entre entidades. Mejora la calidad de respuesta, tiempos y coste [8] [12] [13]

- El registro de métricas de ejecución mejora la auditabilidad del proceso y facilita explicar por qué una respuesta fue generada de una determinada forma

En conjunto, la evidencia experimental respalda una conclusión clave: no existe una única estrategia óptima para todas las consultas. La mejora más robusta aparece cuando el sistema selecciona la estrategia en función del tipo de pregunta y del patrón de distribución del conocimiento en los documentos [2] [4] [8].

5. Conclusiones

Este trabajo aporta una arquitectura RAG aplicada y evaluable en contexto empresarial, con tres contribuciones principales. Primero, un marco modular que separa con claridad ingesta, indexación, recuperación y generación, permitiendo reproducibilidad y evolución incremental [2] [4]. Desarrolla dos extensiones orientadas a las limitaciones concretas del RAG clásico: GraphRAG para consultas relacionales y procesamiento paralelo para consultas de alta cobertura [3] [8] [13]. Una metodología de evaluación comparativa con trazabilidad, útil para tomar decisiones de arquitectura basadas en evidencias [2] [4].

Desde una perspectiva práctica, el trabajo demuestra que mejorar la calidad de un asistente no depende solo del modelo generativo final, sino del diseño completo de la cadena de recuperación. Cuando la recuperación aporta evidencia suficiente, diversa y bien estructurada, disminuyen las respuestas incompletas y se reduce el riesgo de alucinaciones [1] [2] [4].

Como líneas futuras de investigación, se propone: ampliar la evaluación con métricas automáticas, optimizar políticas de enrutamiento dinámico entre estrategias según el tipo de consulta, y profundizar en mecanismos de control de coste para despliegues continuos a gran escala.

6. Referencias

[1] Lewis, P. et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems, vol. 33, pp. 9459-9474, 2020. https://arxiv.org/abs/2005.11401

[2] Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, H. "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv preprint arXiv:2312.10997, 2023. https://arxiv.org/abs/2312.10997

[3] Asai, T. et al. "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," arXiv preprint arXiv:2310.11511, 2023. https://arxiv.org/abs/2310.11511

[4] Mialon, N. F. N. N. et al. "Augmented Language Models: A Survey," arXiv preprint arXiv:2302.07842, 2023. https://arxiv.org/abs/2302.07842

[5] Cormack, G.; Clarke, C. L. A.; Buettcher, S. "Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods," in Proceedings of the 32nd International ACM SIGIR Conference, 2009, pp. 758-759. https://dl.acm.org/doi/10.1145/1571941.1572114

[6] Dhakal, A.; Neupane, S. S.; Chaulagain, N. "A Comparative Study of Information Retrieval Models: BM25 versus Hybrid Retrieval on the Cranfield Collection," 2026. https://n9.cl/2k4d6

[7] Zhang, H.; Liu, J.; Zhu, Z.; Zeng, S.; Sheng, M.; Yang, T.; Dai, G.; Wang, Y. "Efficient and Effective Retrieval of Dense-Sparse Hybrid Vectors using Graph-based Approximate Nearest Neighbor Search," arXiv preprint arXiv:2410.20381, 2024. https://arxiv.org/abs/2410.20381

[8] Nguyen, C. D. M.; French, T.; Stewart, M.; Hodkiewicz, M.; Liu, W. "Representation Learning in Complex Logical Query Answering on Knowledge Graphs: A Survey," ACM Computing Surveys, vol. 58, no. 5, Article No. 112, pp. 1-36, 2025. https://dl.acm.org/doi/full/10.1145/3771692

[9] Microsoft. "Azure AI Search documentation," 2026. https://learn.microsoft.com/azure/search/

[10] Microsoft. "Azure Cosmos DB documentation," 2026. https://learn.microsoft.com/azure/cosmos-db/

[11] Microsoft. "Azure AI Foundry documentation," 2026. https://learn.microsoft.com/azure/ai-foundry/

[12] Neo4j. "Neo4j Graph Database documentation," 2026. https://neo4j.com/docs/

[13] Zep AI. "Graphiti documentation and repository," 2026. https://github.com/getzep/graphiti

RAG SYSTEM DEVELOPMENT

Author: Negueruela Avellaneda, Sofía.
Supervisor: Pérez Beltrán, Nicolás.
Collaborating Entity: EY

ABSTRACT

This work implements and evaluates a modular, multilingual Retrieval-Augmented Generation (RAG) system to improve the reliability of LLM-based assistants in an enterprise context. The starting point is a limitation observed in a real situation: partial responses and hallucinations when asking questions about heterogeneous document repositories. The proposal combines a classic RAG approach with two extensions that aim to solve specific problems: **knowledge graphs**, for distributed relational information; and a **parallel processing** pipeline for questions that require long and complete responses. The results show improvements in completeness and traceability with controlled computarional costs.

Keywords: Retrieval-Augmented Generation, GraphRAG, multilingual system, hybrid retrieval, Azure, Neo4j, experimental evaluation

1. Introduction

The work is framed within generative artificial intelligence applied to information retrieval. Although large language models (LLMs) offer high writing and synthesis capabilities, they have some limitations: potentially outdated knowledge, low specialization in internal data, and traceability difficulties[2] [4]. RAG systems address part of this problem by incorporating external evidence at query time before generating the answer [1] [2]. In real-world scenarios, however, there are two challenges: top-k retrieval may be insufficient for extensive high-coverage questions, and vector similarity does not always reconstruct complex or distributed relationships between documents [2] [3] [8] [13]. Therefore, this work proposes and evaluates a modular architecture that allows comparing RAG strategies and improving response quality without losing operational viability [9] [10] [11].

2. Objectives

The general objective is to develop and evaluate an extensible, multilingual RAG system that enables comparison of different retrieval and generation strategies across several document types, prioritizing three criteria: response completeness, traceability, and efficiency [2] [4]. The specific objectives are:

- Implement a robust document ingestion pipeline with a chunking process adapted to each use case
- Enable hybrid retrieval (lexical, vector, and semantic) with language filtering and use-case-specific configuration [5] [6] [7] [9]
- Incorporate a GraphRAG strategy based on Neo4j and Graphiti to retrieve structural evidence when the query requires relating distributed facts [8] [12] [13]
- Design parallel processing for high-coverage queries, reducing the information loss typical of strict top-k approaches

- Evaluate and compare classic RAG, GraphRAG, and parallel processing through replicable protocols and gold standards
- Log metrics (number of documents consulted, time, model used, and cost) for easier traceability and later analysis

The central research question guiding this work is: which combination of retrieval strategies improves the completeness and reliability of answers in real enterprise scenarios, while maintaining reasonable computational cost? [2] [4]

3. Description of the proposed system

The solution is structured into two decoupled phases: offline phase (ingestion and indexing) and online phase (query, retrieval, and generation) [2] [4]. In the **offline phase**, documents are generated or downloaded according to the use case (CVs, Wikipedia, and EU legislation), normalized, and split into chunks with metadata (language, source, page, timestamp, and domain attributes). Chunks are stored in Azure Cosmos DB and synchronized to Azure AI Search via indexers, with one specialized index per use case [9] [10]. In the **online phase**, the backend receives the user question, uses the selected language, and applies one of the three available strategies:

- Classic RAG: hybrid retrieval in Azure AI Search and answer generation[1] [5] [6] [7] [9]

- GraphRAG: queries the knowledge graph in Neo4j through Graphiti, retrieving episodes, entities, and facts to provide relational context [8] [12] [13]

- Parallel processing: expansion of the retrieved set, grouping by reliability bands, and distributed validation with lightweight models, followed by final consolidation with a higher-capacity model

The implementation uses Python and FastAPI in the backend, with LangChain-based orchestration and a React frontend for functional testing. At the infrastructure level, Azure Blob Storage is used as the persistence layer, Azure Cosmos DB as the document repository, Azure AI Search as the query layer, and model services deployed in Azure AI Foundry or Azure AI OpenAI [9] [10] [11].

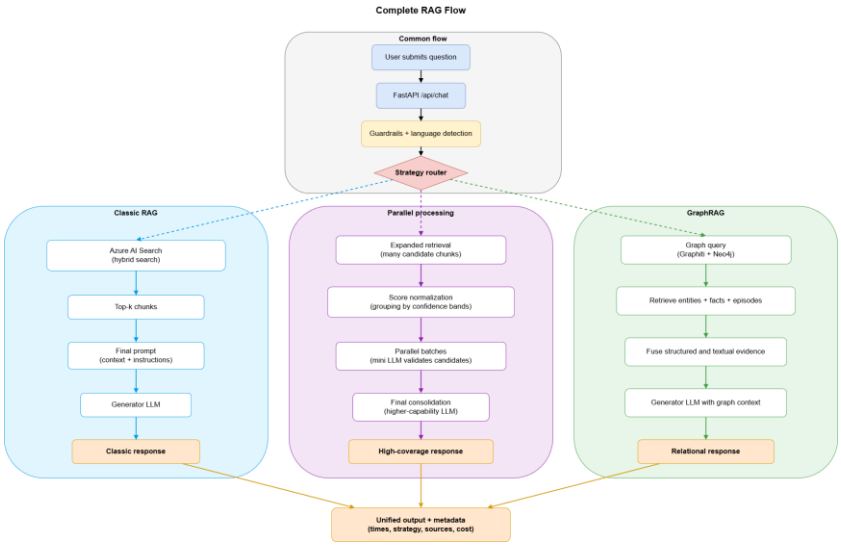

Illustration 1 - Diagram of the complete pipeline with the three strategies

The modular architecture allows persistence, retrieval, and generation to be decoupled, simplifying maintenance and facilitating experimentation [2] [4]. In addition, the system incorporates multilingual capabilities in indexing, filtering, and generation [2] [9].

4. Results

Validation is approached as an experimental comparison among strategies across three use cases and multiple languages. For each scenario, test questions and reference answers (gold standard) are defined, with evaluation of response quality and analysis of cost and latency [2] [4]. The main results are:

- For coverage-oriented questions, parallel processing improves exhaustiveness and completeness compared to classic RAG, but presents higher latency and cost [2] [3]

- GraphRAG provides greater global coherence and better structural justification by incorporating explicit relationships between entities. It improves response quality, timing, and cost [8] [12] [13]

- Execution metric logging improves process auditability and makes it easier to explain why an answer was generated in a particular way

Overall, the experimental evidence supports one key conclusion: there is no single optimal strategy for all queries. The most robust improvement appears when the system selects the strategy according to the type of question and the knowledge distribution pattern in the documents [2] [4] [8].

5. Conclusions

This work provides an applied and evaluable RAG architecture in an enterprise context, with three main contributions. First, a modular framework that clearly separates ingestion, indexing, retrieval, and generation, enabling replication and evolution [2] [4]. Two extensions aimed at specific limitations of classic RAG: GraphRAG for relational queries and parallel processing for high-coverage queries [3] [8] [13]. A comparative evaluation methodology with traceability, useful for making architecture decisions based on evidence [2] [4].

From a practical perspective, the work demonstrates that improving assistant quality does not depend only on the final generative model, but on the complete design of the retrieval chain. When retrieval provides sufficient, diverse, and well-structured evidence, incomplete responses decrease and the risk of hallucinations is reduced [1] [2] [4].

As future work, the following are proposed: expanding evaluation with automatic metrics, optimizing dynamic routing policies among strategies according to query type, and deepening cost-control mechanisms for large-scale continuous deployments.

6. References

[1] Lewis, P. et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems, vol. 33, pp. 9459-9474, 2020. https://arxiv.org/abs/2005.11401

[2] Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, H. "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv preprint arXiv:2312.10997, 2023. https://arxiv.org/abs/2312.10997

[3] Asai, T. et al. "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," arXiv preprint arXiv:2310.11511, 2023. https://arxiv.org/abs/2310.11511

[4] Mialon, N. F. N. N. et al. "Augmented Language Models: A Survey," arXiv preprint arXiv:2302.07842, 2023. https://arxiv.org/abs/2302.07842

[5] Cormack, G.; Clarke, C. L. A.; Buettcher, S. "Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods," in Proceedings of the 32nd International ACM SIGIR Conference, 2009, pp. 758-759. https://dl.acm.org/doi/10.1145/1571941.1572114

[6] Dhakal, A.; Neupane, S. S.; Chaulagain, N. "A Comparative Study of Information Retrieval Models: BM25 versus Hybrid Retrieval on the Cranfield Collection," 2026. https://n9.cl/2k4d6

[7] Zhang, H.; Liu, J.; Zhu, Z.; Zeng, S.; Sheng, M.; Yang, T.; Dai, G.; Wang, Y. "Efficient and Effective Retrieval of Dense-Sparse Hybrid Vectors using Graph-based Approximate Nearest Neighbor Search," arXiv preprint arXiv:2410.20381, 2024. https://arxiv.org/abs/2410.20381

[8] Nguyen, C. D. M.; French, T.; Stewart, M.; Hodkiewicz, M.; Liu, W. "Representation Learning in Complex Logical Query Answering on Knowledge Graphs: A Survey," ACM Computing Surveys, vol. 58, no. 5, Article No. 112, pp. 1-36, 2025. https://dl.acm.org/doi/full/10.1145/3771692

[9] Microsoft. "Azure AI Search documentation," 2026. https://learn.microsoft.com/azure/search/

[10] Microsoft. "Azure Cosmos DB documentation," 2026. https://learn.microsoft.com/azure/cosmos-db/

[11] Microsoft. "Azure AI Foundry documentation," 2026. https://learn.microsoft.com/azure/ai-foundry/

[12] Neo4j. "Neo4j Graph Database documentation," 2026. https://neo4j.com/docs/

[13] Zep AI. "Graphiti documentation and repository," 2026. https://github.com/getzep/graphiti

Índice de la memoria

Índice de figuras

Índice de tablas

1. INTRODUCCIÓN

Durante las prácticas en consultoría tecnológica, se comenzó a desarrollar una nueva herramienta a nivel interno basada en un sistema RAG (*Retrieval-Augmented Generation*). Su objetivo inicial era resolver un problema recurrente: encontrar perfiles dentro de la empresa que cumplieran los requisitos de un nuevo proyecto, facilitando la elaboración de propuestas y justificando que cada persona aportaba la experiencia necesaria. Sin embargo, el sistema no ofrecía los resultados esperados: muchas respuestas eran parciales o incorrectas. Entender por qué ocurría esto se convirtió en el punto de partida de este trabajo.

1.1 CONTEXTO Y MOTIVACIÓN

En los últimos años, la inteligencia artificial generativa se ha incorporado progresivamente en el entorno empresarial. Sin embargo, el uso de modelos de lenguaje de gran tamaño (LLMs) presenta una limitación crítica: la respuesta se apoya en información adquirida durante el entrenamiento, que puede estar desactualizada, incompleta o no alineada con el conocimiento concreto de una organización. Esta limitación se traduce en respuestas plausibles, pero potencialmente incorrectas, con problemas de trazabilidad y verificabilidad.

Los sistemas RAG surgen como solución a estos retos: antes de generar la respuesta final, recuperan evidencia documental relevante y la incorporan al contexto (*prompt*) de la llamada al LLM. Con ello se mejora la precisión de la respuesta, se facilita citar las fuentes y se reduce la dependencia sobre la memoria interna del modelo que se use. Aun así, el estado actual de los sistemas RAG muestra algunas limitaciones en escenarios reales. En primer lugar, muchos flujos de trabajo de los sistemas RAG siguen orientados a una búsqueda vectorial única y eficaz para preguntas acotadas, aunque es menos robusta en consultas que, por ejemplo, requieren enumerar todos los elementos que cumplen una o varias condiciones. En estos casos, recuperar solo un subconjunto de documentos suele producir respuestas parciales e incompletas. A esto se le añade que la ventana de contexto de los modelos de

lenguaje es limitada, y no permiten recuperar demasiada información sin perder calidad en la respuesta. En segundo lugar, las búsquedas por similitud semántica priorizan la proximidad entre *embeddings* (representación vectorial de un fragmento de texto), pero no siempre son correctas cuando la información necesaria para responder a una pregunta está distribuida en fragmentos inconexos. Esta situación aparece con frecuencia en documentación normativa extensa, compleja o con contenidos muy diversos.

A partir de las limitaciones, este Trabajo de Fin de Grado plantea una arquitectura RAG modular y escalable que permita comparar de forma controlada diferentes estrategias de recuperación y generación multiidioma sobre varios casos de uso. La motivación principal, además de construir una solución con valor empresarial, es diseñar e implementar un sistema experimental replicable para investigar qué combinaciones de técnicas de recuperación mejoran la completitud y la fiabilidad de la respuesta sin disparar el coste computacional. El trabajo se desarrolla, además, en un entorno *cloud* realista, con servicios gestionados de Azure para almacenamiento, base de datos, indexación y búsqueda híbrida. Este enfoque aporta valor empresarial al proyecto: en contextos profesionales, la utilidad del sistema depende mucho de su viabilidad operativa. Por ello, el proyecto combina rigor técnico con criterios de despliegue empresarial.

1.2 OBJETIVOS

El objetivo general de este trabajo es desarrollar y evaluar un sistema RAG modular, extensible y multiidioma que permita comparar estrategias de recuperación y generación en distintos tipos de documentos; priorizando exhaustividad, trazabilidad y eficiencia operativa.

Los objetivos específicos planteados son los siguientes:

1. **Procesamiento multiidioma de documentos.** Se busca incorporar capacidades de detección de idioma, permitiendo que el sistema reconozca el idioma antes de procesar la petición del usuario. El sistema almacena el idioma detectado como metadato asociado a cada documento, facilitando así el filtrado y la búsqueda por idioma. Adicionalmente,

el sistema debe ser capaz de procesar diferentes tipos de documento (PDF, JSON, etc.) con estrategias de extracción especializadas para cada formato, utilizando *Document Intelligence*[1] para obtener el contenido estructurado con información acerca de páginas, párrafos y relaciones explícitas

2. **Sistema de *embeddings*.** Se busca integrar un sistema de *embeddings* que soporte la recuperación vectorial dentro de un flujo de búsqueda híbrida que combine búsqueda léxica y vectorial, junto con reordenación semántica. En el proyecto se utiliza el modelo *text-embedding-ada-002* de Azure OpenAI[33]. Este enfoque permite evaluar el impacto del procesamiento de los datos en la recuperación y en la calidad de respuesta

3. **Recuperación Avanzada y Representación del Conocimiento.** Este objetivo contempla la implementación de múltiples estrategias de recuperación y búsqueda de documentos, en concreto:

    a. RAG Fusión con reformulación de consultas
    b. Recuperación vectorial y semántica (búsqueda híbrida), combinada mediante *Reciprocal Rank Fusion* (RRF)
    c. Recuperación basada en grafos de conocimiento (GraphRAG) con Neo4j y Graphiti para documentos

    El objetivo es validar experimentalmente cuándo cada estrategia es superior y cómo su combinación mejora la completitud y rendimiento del sistema sin disparar costes

4. **Sistema de *Keywords* y Metadatos.** Se requiere enriquecer cada documento con metadatos estructurados: idioma detectado, información de página (para PDFs), documento fuente, y *timestamps*. El índice de Azure AI Search debe adaptarse para soportar filtrado por idioma, facilitando las búsquedas multiidioma. Los analizadores del índice se configuran de acuerdo con el tipo de documento y caso de uso, mejorando la

[1] *Document Intelligence* describe el uso de la IA para que un sistema entienda documentos completos (texto, diseño de página, tablas e imágenes), saque la información importante y la use para responder preguntas o automatizar tareas.[34]

precisión de búsqueda léxica y semántica sin requerir procesamiento específico por idioma de los documentos

5. **Generación con LLMs.** Este objetivo integra modelos GPT-5 de Azure AI Foundry para generación, configurados mediante *prompting* adaptado a múltiples idiomas. El sistema incluye plantillas de *prompting* que especifican el idioma de respuesta esperado y control de formato según el caso de uso. Se implementan *guardrails* de entrada/salida y detección de alucinaciones para mejorar la fiabilidad de las respuestas sin optimización local de los modelos usados

6. **Evaluación y *Benchmarking*.** El objetivo comprende la definición de *datasets*[2] de evaluación con preguntas en varios idiomas. Se implementan métricas de calidad de respuesta (relevancia léxica, precisión semántica, completitud) y se realiza un análisis detallado del rendimiento por estrategia de recuperación, por idioma y por tipo de consulta. Se incluye medición de costes (llamadas a LLMs, *tokens*[3] consumidos, latencia) para validar otras mejoras además de la precisión de la respuesta

7. **Estrategias Experimentales de Recuperación y Generación.** Se persigue la exploración de flujos de recuperación alternativos y más sofisticados, incluyendo búsquedas iterativas o paralelas que permitan aumentar la calidad y completitud de la respuesta. El objetivo contempla la comparación experimental entre estrategias de recuperación vectorial frente a estrategias avanzadas en distintos escenarios

1.3 ALINEACIÓN CON LOS ODS

El proyecto se alinea con varios Objetivos de Desarrollo Sostenible como consecuencia directa de las decisiones técnicas y de diseño que abarca.

[2] Un *dataset* es un conjunto organizado de datos relacionados, recopilados y preparados para analizarlos o para entrenar, validar y evaluar modelos de inteligencia artificial. [35]
[3] Un *token* es una "pieza" de texto que procesa el modelo (puede ser una palabra, parte de palabra, un signo o un espacio). **¡Error! No se encuentra el origen de la referencia.**[36]

- **ODS 4. Educación de calidad**

La propuesta favorece la transferencia de conocimiento al construir una arquitectura RAG multiidioma documentada y replicable. Esto facilita que estudiantes y profesionales puedan aprender, reproducir y extender los experimentos de recuperación y generación descritos en este documento sobre conjuntos de datos reales. El valor educativo reside en que el proyecto expone de forma transparente todas las capas del flujo de trabajo: ingesta, *chunking*, indexación, recuperación, generación y evaluación.

- **ODS 9. Industria, innovación e infraestructura**

El desarrollo de un *backend* modular basado en componentes desacoplados permite escalar la solución implementada sin rediseñarla por completo. La separación entre estrategias de recuperación, configuración por caso de uso e infraestructura *cloud* soporta innovación incremental y transferencia industrial. Además, la comparación explícita entre enfoques vectoriales y grafos de conocimiento aporta evidencia útil para decisiones de arquitectura en organizaciones que gestionan documentación compleja.

- **ODS 16. Paz, justicia e instituciones sólidas**

Aunque el proyecto no aborda directamente el ámbito jurídico-institucional, sí contribuye a la transparencia tecnológica mediante mecanismos de trazabilidad del proceso y reducción de respuestas opacas inverificables. En sistemas de IA generativa, la capacidad de justificar la respuesta obtenida con evidencia clara es una condición necesaria para construir confianza y uso responsable.

1.4 ESTRUCTURA DEL TRABAJO

La memoria se organiza en cinco capítulos principales, además de Bibliografía y anexos.

El Capítulo 1, Introducción, presenta el contexto empresarial y tecnológico del problema junto con la motivación del trabajo, los objetivos generales y específicos, y su alineación con los Objetivos de Desarrollo Sostenible. Este capítulo delimita el alcance del proyecto y

justifica por qué resulta relevante estudiar mejoras sobre arquitecturas RAG en escenarios reales. El Capítulo 2, Estado del Arte, recoge los fundamentos teóricos y tecnológicos que sustentan la propuesta. Se revisan conceptos clave de LLMs y RAG, estrategias de recuperación avanzada, grafos de conocimiento y tecnologías *cloud* utilizadas en la implementación. El objetivo de este capítulo es identificar las limitaciones actuales y establecer la base conceptual para las decisiones de diseño posteriores. El Capítulo 3, Sistema Desarrollado, describe el diseño e implementación de la solución propuesta. Se detallan la arquitectura general del sistema, el flujo de ingesta e indexación de documentos, la extensión con grafos de conocimiento y la estrategia de procesamiento en paralelo. También se documenta la estructura del código y la integración de servicios en Azure. El Capítulo 4, Resultados, expone el diseño experimental, la construcción de los *gold standards*, el protocolo de ejecución y las métricas empleadas para evaluar calidad, completitud y coste operativo. Se analizan los resultados, explicando los casos de uso aplicados para comparar el comportamiento del RAG clásico frente a las estrategias avanzadas implementadas. El Capítulo 5, Conclusiones y Trabajos Futuros, sintetiza las aportaciones del proyecto, discute las principales limitaciones observadas y propone líneas de mejora para iteraciones posteriores.

Finalmente, la memoria incluye la bibliografía utilizada y anexos con material complementario como diagramas, esquemas de datos o ejemplos de código y diseño.

2. ESTADO DEL ARTE

Este capítulo presenta el marco teórico que sustenta el proyecto, con el objetivo de situar la propuesta dentro del estado del arte de la investigación y de las soluciones utilizadas en entornos profesionales. La revisión se centra en los sistemas *Retrieval-Augmented Generation* (RAG), en las estrategias avanzadas de recuperación de información y en el papel de infraestructuras *cloud* en entornos de despliegue reales. Se busca responder a tres preguntas clave: qué fundamentos conceptuales son necesarios para entender la arquitectura implementada, qué técnicas y herramientas han demostrado utilidad recientemente, y qué limitaciones presentan las soluciones existentes. El capítulo cubre conceptos fundamentales, técnicas y modelos relevantes, trabajos relacionados y limitaciones observadas en la práctica. Esta estructura permite conectar los antecedentes con las decisiones de diseño del proyecto y sirve como base para la justificación metodológica de los capítulos siguientes.

2.1 CONCEPTOS PREVIOS

Antes de describir la arquitectura de los sistemas RAG y las técnicas de recuperación aplicadas en este trabajo, conviene definir brevemente un conjunto de conceptos que aparecen a lo largo del documento y son claves para comprender el funcionamiento de la aplicación desarrollada.

- **Modelo de lenguaje de gran tamaño (LLM).** Un *Large Language Model* es una red neuronal con muchos parámetros, entrenada sobre grandes bloques de texto para ser capaz de generar uno nuevo modelando la probabilidad de secuencias de palabras [23]. Este entrenamiento permite al modelo capturar hechos y relaciones gramaticales y semánticas del lenguaje, lo que se traduce en la capacidad de generar texto coherente, responder preguntas, resumir documentos o traducir entre idiomas. Sin embargo, todo su conocimiento proviene de los datos con los que fue entrenado; por lo que contiene

información estática, probablemente desactualizada e incompleta para responder con exactitud al usuario en todos los casos[23]. Esto implica tres limitaciones importantes [2]:

i. El modelo no puede actualizar su conocimiento sin ser reentrenado
ii. Carece de información privada o específica de una organización que no estuviese en los datos de entrenamiento
iii. No dispone de mecanismos internos para justificar de dónde proviene cada afirmación de su respuesta, lo que dificulta la trazabilidad y puede dar lugar a alucinaciones – respuestas plausibles pero incorrectas

- ***Chunk***. Un *chunk* es un pequeño fragmento de texto (de entre decenas y cientos de palabras) que pertenece a un documento más largo. Dividir documentos muy extensos en trozos más pequeños es necesario, ya que los LLMs tienen una ventana de contexto limitada; es decir, solo pueden procesar una cantidad finita de texto en cada llamada. Al fragmentar los documentos en *chunks*, el sistema puede seleccionar solo los trozos relevantes para cada consulta. Cada *chunk* almacena el contenido del documento junto con metadatos descriptivos (documento de origen, idioma, página, fecha, etc.) que facilitan su filtrado y trazabilidad

- ***Embedding***. Un *embedding* es una representación numérica de una cadena de caracteres en forma de vector de alta dimensión. Los modelos de *embeddings* transforman palabras, frases o párrafos en vectores de números reales, de manera que textos con significado similar producen vectores que se encuentran cerca en el espacio vectorial. Esta propiedad es la que permite realizar búsquedas por similitud semántica: dado el *embedding* de una consulta, se pueden encontrar los *chunks* cuyo significado sea más parecido [3] [4]

- **Índice**. Un índice es una estructura de datos optimizada para buscar información de forma rápida sin necesidad de recorrer todos los documentos uno a uno. En este trabajo, se utilizan dos tipos de índices: **índices léxicos**, que permiten buscar por coincidencia de palabras (similares a un índice de un libro); e **índices vectoriales**, que organizan *embeddings* en estructuras matemáticas para encontrar los vectores más cercanos a una

consulta dada de forma eficiente[25]¡Error! No se encuentra el origen de la referencia.. La combinación de ambos tipos de índice es lo que se conoce como búsqueda híbrida

Prompt. Un *prompt* es el texto completo que se envía a un LLM como entrada para que genere una respuesta. La calidad y estructura del *prompt* influyen directamente en la respuesta generada. En un sistema RAG, el *prompt* se compone de tres partes:

a) Una instrucción de sistema que establece el rol del modelo, idioma de respuesta y restricciones (por ejemplo, "responde solo usando la información de los documentos proporcionados")
b) Los fragmentos de evidencia recuperados, que se insertan como contexto
c) La pregunta del usuario

2.2 AZURE

En el *paper* de Praveen Borra (2022) [5] y en la documentación técnica[9], se describe Azure como una plataforma *cloud* que desde 2010 opera con un "modelo de nube híbrida" y un catálogo amplio de servicios IaaS (Infraestructura como Servicio), PaaS (Plataforma como Servicio) y SaaS (Software como Servicio). Desde una perspectiva empresarial, esta combinación permite cubrir necesidades de infraestructura base, desarrollo de aplicaciones y consumo de software sin imponer un único patrón técnico.

Además, Azure "ha permitido alta disponibilidad y acceso a recursos *cloud* con baja latencia" gracias a su despliegue global en múltiples regiones, lo que se traduce en continuidad operativa y menor latencia para organizaciones distribuidas. Esta herramienta se suele usar en el ámbito empresarial por tres características clave:

- **Escalabilidad.** Azure permite ampliar o reducir los recursos que se usan en función de las necesidades del usuario. También es posible activar el autoescalado para absorber cargas de trabajo variables sin sobredimensionar recursos, con el objetivo de mantener el rendimiento y evitar costes innecesarios [5]

- **Seguridad y cumplimiento.** Incluye un enfoque de varias capas de seguridad, combinando seguridad física del centro de datos, control de identidad y acceso, protección de red y detección avanzada de amenazas, junto con autenticación multifactor, inicio de sesión único y certificaciones regulatorias (HIPAA, GDPR, ISO/IEC 27001). Por ello, se presenta como "una solución muy fiable para las grandes empresas" [5]

- **Integración de IA.** Azure integra servicios como Azure Machine Learning, OpenAI Service y Cognitive Services para construir aplicaciones inteligentes dentro de procesos de negocio, destacando además su ajuste con entornos que ya operan con Windows Server, Active Directory y Office 365 [5]

Microsoft Azure proporciona una infraestructura *cloud* gestionada para desplegar sistemas de datos e inteligencia artificial con escalabilidad, observabilidad y seguridad empresarial. En trabajos de IA aplicada, el uso de plataforma *cloud* no solo resuelve aspectos operativos; también condiciona la arquitectura de datos, la gobernanza y la reproducibilidad del ciclo de vida del modelo[5]. En este proyecto, Azure se utiliza como entorno de ejecución para integrar almacenamiento documental (*Azure Cosmos* DB), indexación híbrida (*Azure AI* Search) y consumo de modelos (*Azure AI* Foundry). Esta decisión permite que el desarrollo se lleve a cabo en un entorno realista, usando una herramienta común en el ámbito empresarial que aporta valor a la solución implementada.

2.2.1 AZURE BLOB STORAGE

Azure Blob Storage es un servicio de almacenamiento de objetos en la nube que permite guardar grandes volúmenes de datos no estructurados, como documentos, imágenes o archivos de respaldo[6]. Su propuesta de valor se basa en tres características principales:

- **Escalabilidad y elasticidad.** Permite almacenar y gestionar cantidades masivas de datos, adaptándose automáticamente al crecimiento de la información sin necesidad de aprovisionar recursos manualmente

- **Acceso global y alta disponibilidad.** Los datos almacenados en Blob Storage pueden ser accedidos desde cualquier ubicación con baja latencia, gracias a la infraestructura distribuida de Azure
- **Seguridad y economía.** Ofrece cifrado en reposo y en tránsito, control de acceso granular y opciones de almacenamiento optimizadas para reducir costes según el uso

En este proyecto, Blob Storage se utiliza como repositorio para documentos fuente, facilitando la ingesta de datos en el flujo RAG. Su integración con otros servicios de Azure permite automatizar la ingesta y asegurar la persistencia de los datos originales.

2.2.2 AZURE COSMOS DB

Azure Cosmos DB es una base de datos NoSQL distribuida globalmente diseñada para aplicaciones de misión crítica. El término "misión crítica" hace referencia a sistemas que si no están disponibles o fallan pueden causar un impacto operativo grave: interrupciones de servicio afectan la experiencia de usuarios, pérdida de datos financieros, o incapacidad para ejecutar procesos empresariales esenciales[7]. Por ello, Cosmos DB está diseñado con tolerancia a fallos, redundancia geográfica y recuperación automática para garantizar que el sistema permanece disponible incluso durante eventos adversos (fallos de hardware, desastres regionales, picos inesperados de carga).

Su propuesta de valor se fundamenta en tres pilares técnicos: distribución global, escalabilidad horizontal elástica, y soporte nativo para múltiples modelos de datos[7]. El servicio garantiza acuerdos de nivel de servicio (SLAs) integrales que abarcan cuatro dimensiones críticas:

- ***Throughput***: capacidad de procesar transacciones por segundo. Se mide en *Request Units* (RU/s), una métrica normalizada independiente del hardware. En Cosmos DB, se provisiona el *throughput* deseado y el sistema garantiza esos valores de manera consistentemente, escalando automáticamente si la demanda aumenta

- **Latencia**: tiempo transcurrido desde que se envía una solicitud hasta que se recibe respuesta. Cosmos DB garantiza latencia bajo 10ms en percentil 99 para lecturas y bajo 15ms para escritas indexadas
- **Disponibilidad**: porcentaje de tiempo operativo del servicio. Cosmos DB garantiza disponibilidad del 99.99%, lo que implica que, en todo un año, el servicio está parado durante menos de cuatro minutos
- **Consistencia**: garantía sobre cómo se ven los datos después de escribirlos. Cosmos DB ofrece múltiples modelos predefinidos (*strong*, *bounded staleness, session*, *consistent prefix*, *eventual*) en lugar de solo dos extremos.

Cosmos DB se diferencia como el primer y único servicio de base de datos distribuida en ofrecer SLAs comprensivos y respaldados por contrato en estas cuatro dimensiones simultáneamente[4]. En el proyecto, Cosmos DB actúa como capa de persistencia documental aprovechando dos características:

1. **Esquema agnóstico** que permite almacenar documentos JSON con estructura variable, conteniendo *chunks*, metadatos e índices. Esta estructura de los documentos (*chunks*) se define durante la ingesta documental
2. **Indexación automática y particionamiento** transparente, que facilitan escalabilidad sin aumentar latencia[7]. Cosmos proporciona menos de 15 ms (milisegundos) de latencia incluso en operaciones de lectura o escritura de larga distancia, lo que implica recuperación rápida durante la ingesta y consulta de *chunks*

Los contenedores de Cosmos DB actúan como fuentes de datos para Azure AI Search, que se sincronizan mediante *indexers*. De este modo, la arquitectura separa claramente las responsabilidades: Cosmos DB actúa como capa de almacenamiento persistente de documentos, *chunks* y metadatos; mientras que Azure AI Search proporciona el plano de consulta optimizado para la recuperación léxica, vectorial e híbrida.

2.2.3 AZURE AI SEARCH

Azure AI Search es un motor de búsqueda gestionado que combina capacidades de indexación documental con búsqueda léxica, vectorial y semántica. Aunque la implementación concreta varía, sus fundamentos se apoyan en técnicas clásicas de recuperación de información. Desde un punto de vista técnico, la búsqueda **léxica** se basa en variantes de BM25 (*Best Matching 25*) para *ranking* por coincidencia de términos[8]. La búsqueda **vectorial** utiliza estructuras ANN (*Approximate Nearest Neighbor*) como HNSW (*Hierarchical Navigable Small World*) para búsqueda aproximada eficiente en alta dimensión[25]. Por su parte, la búsqueda **semántica** incorpora modelos de lenguaje para reordenar resultados según intención y significado contextual[10]. Sobre estos resultados puede aplicarse fusión de *rankings*, donde enfoques como *Reciprocal Rank Fusion* (RRF) han mostrado robustez práctica [10].

En el proyecto, la capa de Azure AI Search se organiza por caso de uso mediante índices específicos, cada uno con campos, analizadores y configuración semántica propios. Esta especialización por dominio sigue una recomendación frecuente en sistemas de búsqueda: adaptar el esquema y la señal de *ranking* al tipo documental mejora significativamente la utilidad de recuperación. En consecuencia, la búsqueda híbrida (léxica + vectorial + semántica) se ejecuta en Azure AI Search, no en Cosmos DB. El papel de los *indexers* es sincronizar automáticamente documentos desde Cosmos DB hacia el índice de búsqueda, aplicando mapeos de campos y permitiendo operar con flujos de trabajo reproducibles de ingesta-indexación. Arquitectónicamente, esto separa responsabilidades:

1. Cosmos DB como almacenamiento persistente
2. Azure AI Search como plano de consulta optimizado
3. *Backend* RAG como orquestador de recuperación y generación

Este desacoplamiento facilita mantenimiento, actualización selectiva de índices y experimentación con diferentes estrategias de recuperación sin rediseñar la capa de persistencia.

2.2.4 AZURE AI FOUNDRY

Azure AI Foundry se orienta al ciclo de vida de aplicaciones de IA generativa: selección y despliegue de modelos, gestión de credenciales, evaluación, observabilidad y gobierno del sistema. Aunque parte del desarrollo de este TFG se implementa directamente en código Python, el marco de Foundry resulta relevante porque formaliza prácticas de LLMOps y *Responsible AI* en entornos empresariales [11] [20]. Desde el punto de vista metodológico, AI Foundry aporta valor en cuatro ejes [11]:

1. Gestión de modelos y despliegues, ya que permite tratar modelo, versión y *endpoint* como activos controlados
2. Evaluación continua, facilitando la comparación de *prompts*, configuraciones y resultados bajo criterios repetibles
3. Trazabilidad y observabilidad, pues ayuda a registrar entradas, salidas, costes y comportamiento en producción
4. Gobernanza y seguridad, integrando controles de acceso y políticas alineadas con buenas prácticas de *Machine Learning*

En una arquitectura RAG como la de este trabajo, estos ejes son críticos porque el rendimiento no depende de un único modelo, sino de la interacción entre ingesta, recuperación, *prompting* y generación. Por tanto, la aportación de una plataforma de este tipo mejora la calidad del proceso experimental y la fiabilidad del despliegue.

2.3 SISTEMA RAG

Los LLM han demostrado una capacidad notable para generar texto coherente, responder preguntas y sintetizar información. Sin embargo, presentan una limitación fundamental: todo su conocimiento procede de los datos con los que fueron entrenados. Esto significa que pueden estar desactualizados, carecer de información necesaria para responder al usuario, y no disponer de mecanismos para justificar de dónde proviene cada afirmación ni trazabilidad sobre las fuentes que respaldan su respuesta [2] [23]. El paradigma *Retrieval-Augmented Generation* surge como respuesta a estos problemas. En lugar de depender exclusivamente

del conocimiento almacenado en los parámetros del LLM, un sistema RAG recupera los *chunks* relevantes en tiempo de consulta e incorpora su contenido al *prompt* antes de generar la respuesta final. De este modo, el modelo responde con lo que encuentra en una base de documentos externa y actualizable. Esto reduce el riesgo de alucinaciones, mejora la precisión de la respuesta y permite al usuario verificar las fuentes consultadas[12] [31]. La arquitectura del flujo RAG suele descomponerse en cinco etapas [2]:

1. **Ingesta documental.** Los documentos fuente (PDFs, archivos JSON, etc.) se procesan para extraer su contenido y se dividen en *chunks* con sus metadatos asociados. La calidad de esta fase condiciona el rendimiento de todo el sistema: un *chunking* mal diseñado puede provocar que información relevante quede dividida en varios fragmentos o sea irrecuperable
2. **Indexación**. Cada *chunk* se representa de forma léxica y vectorial, y se almacena en índices que permiten realizar operaciones de recuperación de información de manera eficiente. La representación léxica permite encontrar *chunks* por coincidencia de palabras mediante algoritmos como BM25[8], mientras que la representación vectorial, basada en *embeddings*, permite encontrar *chunks* por similitud semántica, aunque no compartan las mismas palabras [3]
3. **Recuperación**. Cuando el usuario formula una pregunta, el sistema necesita encontrar los *chunks* que contienen la información relevante para responderla. Si se realiza una búsqueda léxica, se comparan las palabras exactas de la pregunta y del *chunk*. Esto funciona bien cuando el usuario emplea los mismos términos que aparecen en los documentos, pero falla si usa sinónimos o reformulaciones. En cambio, si se realiza una búsqueda vectorial, se comparan los *embeddings* de la pregunta y el contenido del *chunk*. Así se logra capturar el significado, aunque se usen palabras distintas; pero se puede perder precisión con nombres propios, siglas o tecnicismos muy específicos. La búsqueda híbrida combina ambas para cubrir los posibles fallos de cada una por separado. Es el enfoque más habitual en los sistemas RAG actuales [12] [3]
4. **Enriquecimiento del contexto.** La búsqueda inicial suele devolver muchos *chunks* candidatos (por ejemplo, 50 o 100), pero no todos son igual de útiles ni caben en la

ventana de contexto del modelo. En esta etapa se refinan los resultados mediante dos mecanismos.

5. **Generación condicionada**. El LLM recibe dicho *prompt* con la consulta del usuario, los *chunks* seleccionados y las instrucciones de formato definidas previamente. Con esta información, el LLM genera una respuesta apoyada en los documentos proporcionados, en lugar de recurrir exclusivamente a su conocimiento interno. Esto es lo que distingue a un sistema RAG de un modelo generativo convencional: la respuesta está respaldada por fragmentos de documentos reales[2] [31]

La literatura reciente destaca que la calidad final depende tanto del generador como de la recuperación, y en muchos escenarios este último es el factor limitante[12] [13]. Por ello, han proliferado diversas técnicas orientadas a mejorar la fase de recuperación [3] [12] [22]. Entre estas técnicas, **RAG Fusión** se ha consolidado como un método eficaz para mejorar la recuperación de documentos[12] [5]. Su funcionamiento sigue estos cuatro pasos:

i. Dada la consulta original del usuario, un LLM genera automáticamente N reformulaciones que expresan la misma necesidad de información desde perspectivas distintas. Por ejemplo, ante la pregunta "¿qué perfiles del equipo tienen experiencia en desarrollo cloud?", el sistema podría generar variantes como "profesionales con proyectos en AWS o Azure", "consultores especializados en infraestructura en la nube" o "personas con certificaciones de plataformas cloud". Cada variante busca captar matices, sinónimos o formulaciones alternativas que una única consulta no cubriría

ii. La búsqueda de cada una de las N variantes se ejecuta de forma independiente, obteniendo N listas de *chunks* separadas. Dado que cada formulación enfatiza aspectos diferentes de la consulta, las listas pueden solaparse o recuperar documentos complementarios

    Cuando se obtienen los resultados de las N búsquedas, no es posible simplemente sumar las puntuaciones que cada tipo de búsqueda asigna por defecto: cada una usa su propia escala, y una puntuación de 0.95 en una búsqueda vectorial no es comparable con un 0.88 en una búsqueda semántica. La solución que aporta **RRF**

(*Reciprocal Rank Fusion*) es ignorar completamente las puntuaciones numéricas y usar únicamente la posición (rango) de cada resultado en su lista[32]. La fórmula para cada aparición de un documento es:

$$\text{RRF\_score}(d, q) = \frac{1}{k + \text{rank}(d, q)}$$

Donde:

- $\text{rank}(d, q)$ es la posición del documento d en la lista de la búsqueda q
- $k = 60$ es una constante de suavizado estándar entre los rangos altos y bajos; evita que los primeros puestos dominen demasiado.

La puntuación final de un documento (*chunk)* es la suma de sus contribuciones en todas las listas donde aparece. El *chunk* consistentemente relevante en múltiples reformulaciones de la consulta gana sobre el que solo es relevante para una formulación concreta. Esto reduce el ruido y aumenta la precisión final. En el Anexo A.1 se detalla su implementación.

iii. Del *ranking* fusionado se seleccionan los top-k documentos para pasarlos al generador como evidencia

La ventaja principal de RAG Fusión frente a una búsqueda simple es que reduce la sensibilidad a la formulación exacta de la pregunta: en lugar de depender de que una única pregunta sea suficientemente buena para recuperar todos los documentos necesarios, el sistema lanza múltiples variantes y premia la consistencia entre resultados[5]. Esto es especialmente útil en escenarios multiidiomas o con vocabulario técnico variado, donde una sola formulación rara vez cubre todas las formas en que la información relevante está expresada. Los sistemas RAG presentan algunos problemas recurrentes [2] [13]:

- Cuando la pregunta requiere listar todos los elementos que cumplen una condición (por ejemplo, "¿qué perfiles del equipo tienen experiencia en Python y Azure?"), el sistema tiende a devolver solo un subconjunto de *chunks*, omitiendo otros igualmente válidos que quedaron fuera del top-k más relevantes para la pregunta. Esto produce **respuestas parciales** que no recogen la totalidad de la información disponible

- El tamaño del *chunk*, la estrategia de solapamiento entre fragmentos y los metadatos asociados condicionan directamente la búsqueda. *Chunks* demasiado grandes diluyen la señal de relevancia; *chunks* demasiado pequeños pierden contexto. Así mismo, metadatos mal definidos impiden filtrar resultados de manera eficiente
- Muchos de los modelos de *embeddings* y los analizadores léxicos están optimizados para un idioma principal (habitualmente inglés). Cuando la fuente de datos contiene documentos con formatos muy distintos, la calidad de la recuperación empeora
- Cuando la respuesta requiere conectar información repartida en varios documentos, la recuperación vectorial tiende a tratar cada fragmento de forma aislada sin reconstruir la relación entre ellos. Esto limita la capacidad del sistema para resolver preguntas que exigen razonamiento relacional o síntesis de fuentes diversas

Un sistema RAG funciona como una cadena de tres pasos: primero busca información relevante, después la incorpora como contexto y, por último, genera una respuesta apoyada en esos documentos consultados. La idea central es que el modelo de lenguaje responde con lo que recupera en cada consulta desde una base documental externa[2] [31]. Esto reduce el riesgo de respuestas inventadas y permite que la salida esté mejor alineada con documentos concretos. De forma simplificada, el proceso es el siguiente:

1. La pregunta del usuario se recibe e interpreta.
2. Se buscan documentos de la base de datos que respondan a esa pregunta
3. Se seleccionan los documentos más útiles usando RAG Fusión
4. Se construye un *prompt* combinando la pregunta del usuario, el contenido de los *chunks* y algunas instrucciones predefinidas
5. La llamada al LLM produce una respuesta

Aunque este esquema parece simple, su comportamiento real depende mucho de factores como el tamaño de los *chunks*, la forma de indexación, el tipo de búsqueda y la combinación de resultados. Si la recuperación falla, el LLM recibe contexto incompleto o poco útil; si la recuperación es fiable, la calidad de la respuesta mejora de forma notable. Por eso, en este tipo de sistemas es importante recuperar bien la información adecuada [2] [13].

2.4 GRAFOS DE CONOCIMIENTO

Un grafo de conocimiento representa entidades (documentos) y relaciones semánticas entre ellas, generalmente mediante tuplas formadas por sujeto, relación y objeto[17] [18]. Este modelo permite capturar la estructura explícita de la información y habilitar operaciones de inferencia, navegación contextual y recuperación relacional [16] [17].

Frente a índices vectoriales, los grafos presentan dos ventajas conceptuales [16] [18]:

- ✓ Las relaciones entre documentos son explícitas; no solo almacenan similitud textual, sino también las conexiones semánticas interpretables entre *chunks*
- ✓ Mejoran trazabilidad estructural, puesto que la respuesta puede justificarse por relaciones o subgrafos, no únicamente por proximidad de los *embeddings*

Los fundamentos técnicos de los grafos de conocimiento se apoyan en tres bloques [17]:

- Extracción de entidades y relaciones. Se identifican nodos y aristas a partir de texto no estructurado
- Enriquecimiento y resolución de entidades. Se normalizan variantes y se reducen ambigüedades entre menciones
- Consulta y *ranking* sobre grafo. Se recuperan subestructuras relevantes para la pregunta

En aplicaciones reales, los grafos han funcionado bien cuando el problema exige relacionar hechos distribuidos o mantener consistencia entre múltiples fuentes[15] [16] [19]. En esos escenarios, el enfoque vectorial puro puede recuperar fragmentos relevantes de forma aislada, pero no siempre reconstruye el contexto necesario para dar una respuesta completa. Trabajos recientes en GraphRAG muestran mejoras en algunos tipos de preguntas al combinar recuperación basada en estructura de grafo con síntesis generativa[19]. La clave no es sustituir por completo la búsqueda vectorial, sino complementarla con una capa estructural que aumente cobertura y coherencia cuando los hechos son más dispersos [19].

2.4.1 GRAPHITI

Graphiti es una librería orientada a construir y consultar grafos de conocimiento dinámicos para aplicaciones con llamadas a LLMs, normalmente empleando Neo4j como capa de persistencia[24]. Su propuesta práctica es ofrecer una interfaz unificada para ingesta episódica de texto, extracción de entidades/relaciones y búsqueda híbrida sobre nodos, aristas y episodios. El diseño de Graphiti se apoya en principios ampliamente documentados: *embeddings* para recuperación semántica[4], *reranking* para mejorar relevancia[20], y modelado de conocimiento relacional en grafos [16] [17].

A alto nivel, el flujo operativo de Graphiti puede resumirse en:

1. **Ingesta**. Cada *chunk* se añade como episodio contextualizado
2. **Extracción**. El sistema identifica entidades y relaciones, creando nodos y aristas
3. **Consolidación**. Los elementos se persisten en Neo4j

En el contexto del proyecto, esta lógica encaja de forma natural con la estrategia GraphRAG implementada. La búsqueda devuelve una combinación de episodios, hechos y entidades, que posteriormente se transforma en contexto que se añade al *prompt* final. Además, Graphiti aporta tres ventajas principales:

- ✓ Permite contrastar búsqueda vectorial clásica frente a recuperación estructural
- ✓ Facilita el análisis cualitativo de por qué se recuperan ciertos documentos
- ✓ Mejora la cobertura en escenarios donde la información es inconexa o está fragmentada en múltiples documentos

Como limitación, el coste de construcción y mantenimiento del grafo suele ser mayor que en un flujo de trabajo puramente vectorial; y la calidad final depende de la precisión de extracción de entidades/relaciones.

3. SISTEMA DESARROLLADO

En este capítulo se describe el núcleo del proyecto: el problema abordado, el diseño de la solución y su implementación. La estructura se adapta al objetivo del trabajo, que no es solo construir un RAG funcional, sino analizar sus límites y proponer mejoras específicas para cada tipo de problema detectado.

3.1 PLANTEAMIENTO DEL PROBLEMA

El proyecto parte de dos limitaciones recurrentes de los sistemas RAG en escenarios reales.

La primera aparece cuando una pregunta exige **cobertura completa**, es decir, la respuesta requiere consultar un gran número de documentos distintos. Esto ocurre, por ejemplo, al pedir un listado de personas que cumplan varios criterios. En estos casos, un flujo RAG clásico (recuperación top-k y una única generación final) suele priorizar documentos con alta relevancia, pero pierde muchos resultados válidos que quedan fuera de los primeros *chunks* recuperados. Esto se traduce en una respuesta incompleta: faltan elementos válidos y, en ocasiones, el LLM rellena "lo que no sabe" con inferencias no respaldadas por el contexto.

La segunda limitación surge cuando la información necesaria está distribuida en **documentos inconexos**. Aunque cada *chunk* sea correcto por separado, el sistema tiene dificultades para reconstruir adecuadamente las relaciones globales entre entidades, hechos y fuentes. Un ejemplo de esto puede ser consultar sobre un requisito normativo de una institución. Si un documento establece la creación de una ley de manera detallada y otro documento posterior modifica algún aspecto de esa normativa, es probable que los *chunks* recuperados procedan del primer documento, lo que conduce a información desactualizada y respuestas erróneas. Un RAG clásico puede recuperar fragmentos relevantes, pero no integra sus relaciones jurídicas de manera consistente. El mismo patrón se observa con

artículos de Wikipedia, cuando la respuesta depende de enlazar información de artículos diferentes.

Con el objetivo de investigar soluciones a las limitaciones detectadas, en el proyecto se desarrollan dos estrategias RAG diferentes (Grafos de Conocimiento y Procesamiento en Paralelo), una para cada limitación detectada. Para ello, el sistema debe cumplir un conjunto de requisitos que garanticen tanto su viabilidad funcional como su robustez operacional.

- **Requisitos funcionales**: El sistema debe implementar un RAG operativo con búsqueda híbrida y generación con LLM, añadir variantes específicas por problema mantener trazabilidad completa desde ingesta hasta respuesta, soportar ejecución multiidioma en español e inglés y permitir comparación experimental entre RAG clásico y las estrategias implementadas
- **Requisitos no funcionales**: La reproducibilidad se garantiza mediante *scripts* desacoplados por fase. La escalabilidad debe ser razonable para colecciones heterogéneas, y el coste debe mantenerse controlado usando modelos ligeros en etapas intermedias cuando sea posible

La evaluación se basa en un *gold standard* por caso de uso e idioma. No se busca solo medir exactitud superficial, sino verificar si las modificaciones del sistema RAG superan efectivamente las limitaciones identificadas. El diseño e implementación de la evaluación del sistema se desarrolla detalladamente en el 4.

3.2 DISEÑO DE LA SOLUCIÓN

El diseño del sistema se plantea de forma incremental y orientado al problema. En lugar de buscar una única arquitectura universal, se parte de un RAG clásico y se introducen cambios específicos según el tipo de limitación observada en cada caso de uso.

La arquitectura final combina tres capas:

- Un sistema RAG clásico, con búsqueda híbrida y generación

- Una extensión Graph RAG para reconstruir relaciones entre documentos inconexos
- Una modificación, procesamiento en paralelo, para máxima cobertura en la respuesta

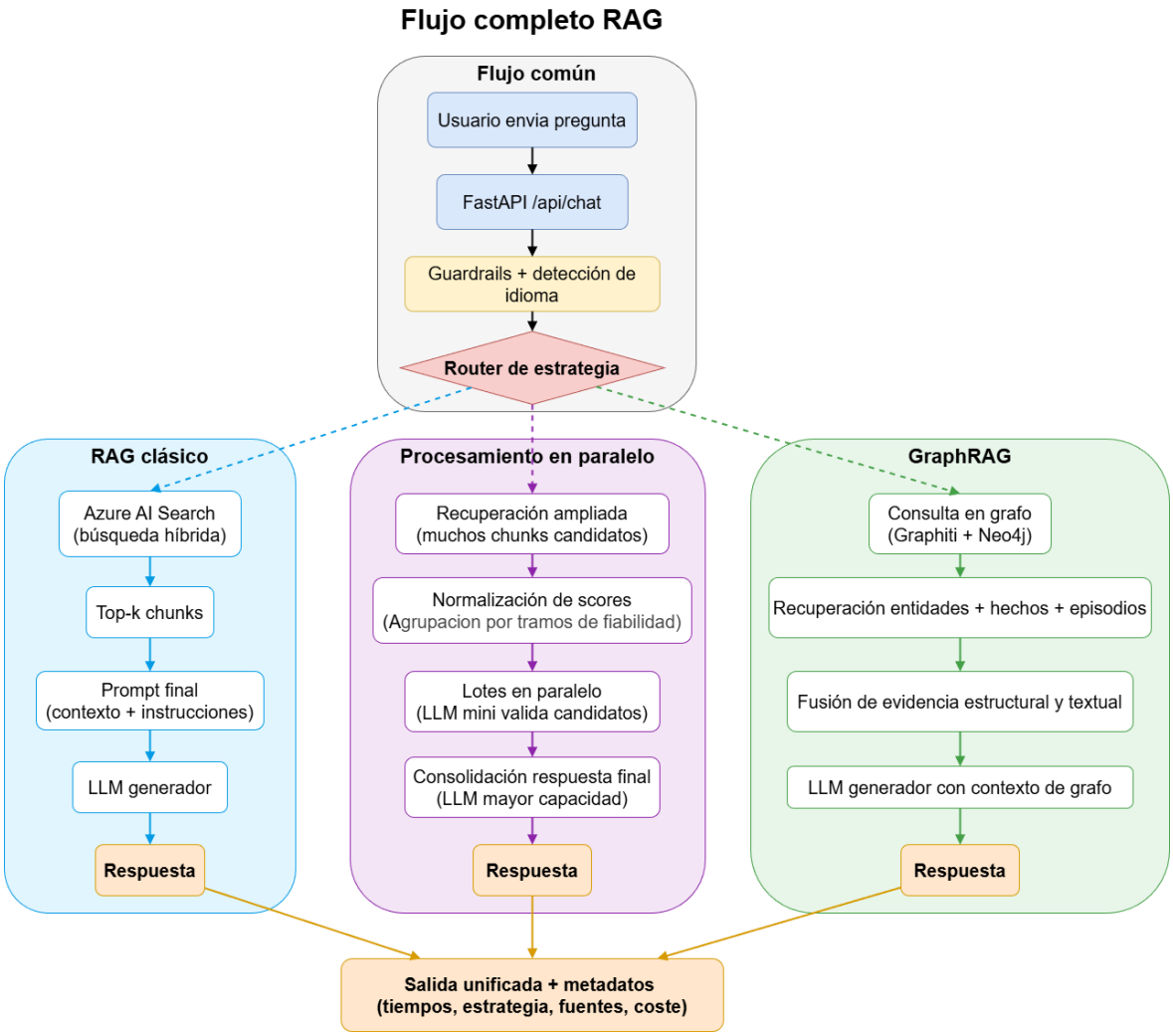

Figura 1. Arquitectura general del sistema con las tres estrategias RAG implementadas

Además, la arquitectura incorpora una dimensión multiidioma operativa en toda la cadena de consulta: el idioma seleccionado por el usuario condiciona el filtrado de chunks recuperados y las instrucciones de generación, de forma que búsqueda y respuesta usan el mismo idioma.

3.2.1 INGESTA DOCUMENTAL

En este apartado se describe todo el proceso de ingesta documental para los tres casos de uso: generación o descarga de datos, chunking, subida a Cosmos DB y creación de índices e *indexers* en Azure AI Search. Primero, veamos el origen de cada conjunto de documentos seleccionados:

- **CVs** (dataset sintético): se generaron 300 CVs por idioma (español, inglés) en formato JSON mediante un *script* en Python, seleccionando los campos de forma aleatoria a partir de un conjunto de opciones para cada campo: nombre, puesto, experiencia, estudios y habilidades
- **Wikipedia**: se descargaron 135 artículos en cada idioma (español, inglés) con la API de Wikipedia y gestión de cookies de navegador, almacenándolos en formato JSON y por idioma
- **Legislación UE**: se descargaron 20 PDFs por idioma, en cinco idiomas distintos (español, inglés, francés, portugués, italiano), desde la web EUR-Lex[27] con un flujo de *web scraping* robusto

Durante la descarga de documentos apareció una limitación práctica: en algunos entornos, peticiones HTTP simples fueron bloqueadas por mecanismos *anti-bot* (*CloudFront WAF*). Esto obligó a reforzar la adquisición de sesión/cookies y los reintentos de descarga. Aunque se trata de un detalle técnico del proceso de ingesta, debe tenerse en cuenta para la reproducibilidad experimental. Los documentos se almacenan en Azure Blob Storage. Para garantizar consistencia en los datos, se validan los documentos de CVs y Wikipedia con estos esquemas de entrada. Además, se define una estructura homogénea de *chunk* para almacenar cada fragmento en Cosmos, con metadatos específicos de cada caso de uso.

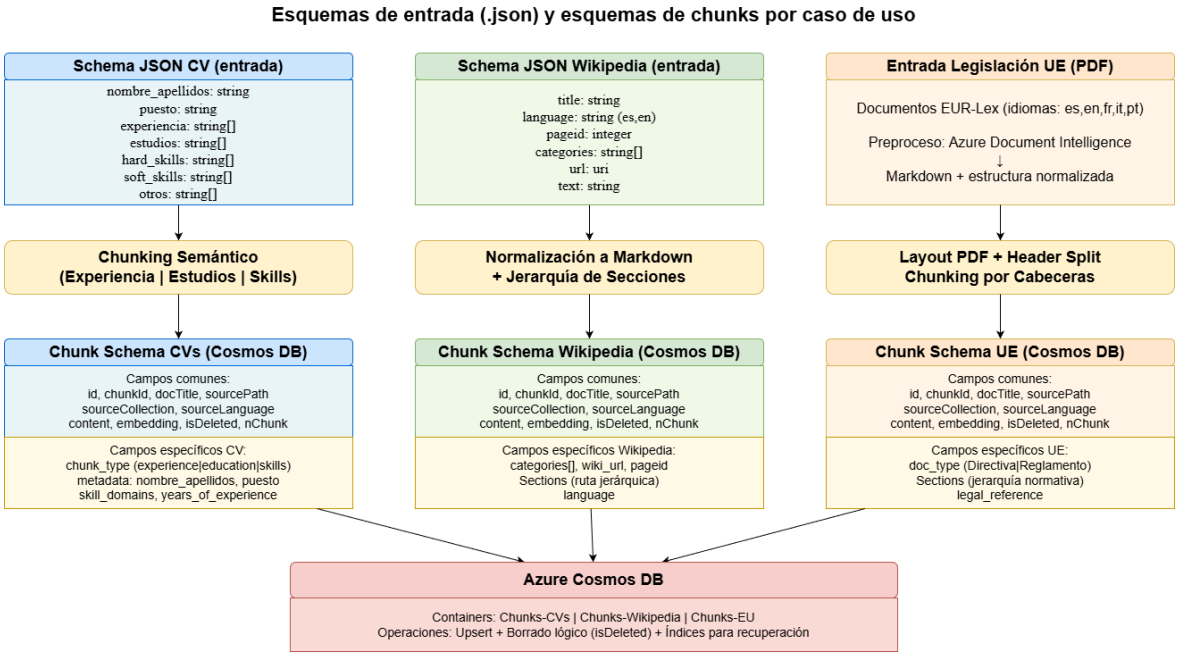

Figura 2. Schemas de entrada y estructura de los chunks por caso de uso. Ver Anexo A.2

La estrategia de *chunking* se adapta a la naturaleza de cada caso de uso. Para **CVs**, se aplica *chunking* semántico fijo dividiendo cada documento en tres bloques funcionales – experiencia, estudios y *skills* – enriqueciendo cada *chunk* con metadatos del documento original para mejorar la trazabilidad y el filtrado posterior. En **Wikipedia**, los encabezados se normalizan a formato *markdown;* y los artículos se dividen respetando la jerarquía de secciones y subsecciones, fusionando fragmentos pequeños que perderían contexto y subdividiendo aquellos que superen un umbral de tokens definido previamente. Finalmente, los PDFs de la **Legislación de la UE** se extraen también a formato *markdown* estructurado con Azure Document Intelligence; se limpia el contenido repetido (encabezados y pies de página) y se aplica *chunking* por encabezados con ajuste dinámico del tamaño objetivo del *chunk*, evitando cortes arbitrarios que comprometan la coherencia normativa.

El flujo completo para crear la base documental es el siguiente:

1. Leer los documentos originales almacenados en Azure Blob Storage
2. Transformar cada documento en *chunks*
3. Generar un *embedding* por *chunk*.

4. Subir y añadir cada *chunk* al contenedor de Cosmos DB correspondiente
5. Crear índices e *indexers* para cada caso de uso

Una vez todos los documentos han sido procesados e ingestados en Cosmos DB, cada caso de uso crea su índice y su *indexer* sobre su base de datos. Cada índice habilita la búsqueda híbrida: léxica (BM25) y vectorial (HNSW), con filtros por idioma y metadatos específicos de cada caso de uso.

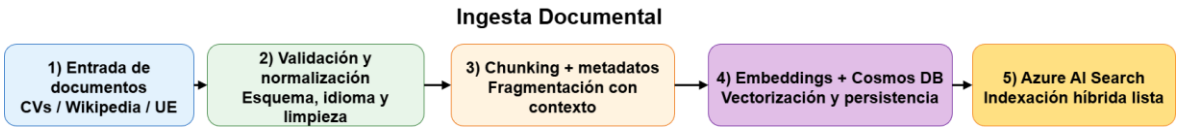

Figura 3. Flujo simplificado de la ingesta documental

3.2.2 GRAFOS DE CONOCIMIENTO

Para gestionar cualquier base de datos con documentos inconexos se añade una capa de grafo de conocimiento con Neo4j y Graphiti. Esta extensión responde a una limitación importante de la búsqueda textual clásica: aunque las búsquedas semánticas son efectivas para recuperar fragmentos con contenido similar, a menudo pierden de vista las relaciones explícitas que conectan esos fragmentos. Dicho de otra forma, un documento puede contener información sobre una entidad X y otro documento sobre la entidad Y, pero la relación "X está conectada con Y" no es obvia, y solo se ve cuando está presente explícitamente en la base de datos; pero un grafo sí es capaz de plasmar las relaciones de sus nodos.

El proceso de migración a Neo4j comienza reutilizando los *chunks* existentes en Cosmos DB. Para cada caso de uso, se crea una base de datos de grafos dentro de una misma instancia de Neo4j. Una vez establecida la conexión, se inicializa un esquema de índices y restricciones en el grafo para garantizar que las entidades se identifiquen unívocamente y las relaciones se normalizasen sin duplicados. Cada *chunk* se ingiere como un episodio textual en el grafo. Durante esta ingesta, Graphiti realiza automáticamente la extracción de entidades y relaciones. El motor de extracción recorre el texto, identifica entidades nominales (personas, organizaciones, conceptos), detecta las relaciones implícitas o explícitas entre

ellas, y consolida todo en una estructura de grafo. Para garantizar una ingesta robusta, en contextos con un gran número de documentos se aplica concurrencia controlada mediante semáforos y estrategias de reintento exponencial, evitando así los bloqueos por *timeout* que ocurren muchas veces en operaciones masivas.

El cambio en la estrategia de búsqueda es radical. En lugar de ejecutar una búsqueda textual clásica y luego intentar razonar sobre los resultados, el sistema ahora realiza una consulta estructural sobre el grafo. Esta consulta devuelve tres categorías de evidencia complementarias: **hechos**, representados como aristas del grafo (conexiones entre entidades); **nodos de entidades** que son relevantes para la pregunta del usuario; y **episodios** o *chunks* que fundamentan esas entidades. Estos tres niveles de evidencia se fusionan en un contexto enriquecido que se pasa finalmente al LLM que genera la respuesta, permitiendo que razone sobre datos textuales y estructurales.

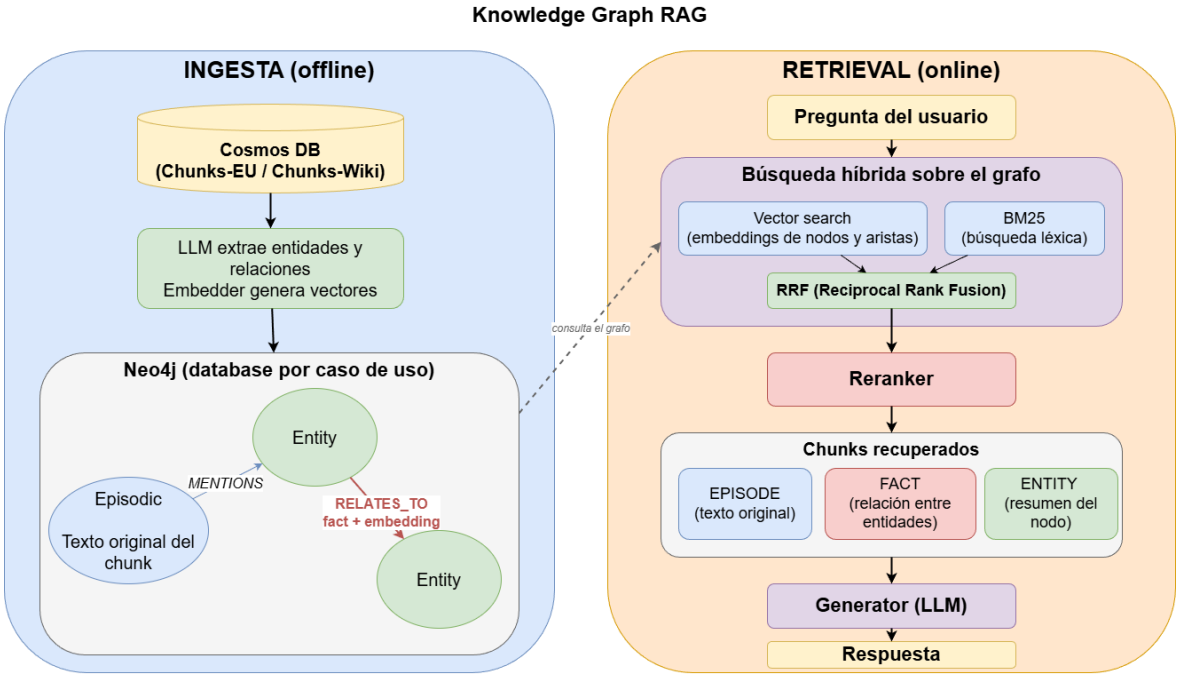

Figura 4. Flujo del sistema Graph RAG

3.2.3 PROCESAMIENTO EN PARALELO

La segunda estrategia propuesta aborda un problema distinto: cuando una consulta busca enumerar múltiples perfiles (por ejemplo, "buscar todos los perfiles con experiencia en AWS y Kubernetes"), el sistema RAG clásico presenta dos cuellos de botella.

- El primero surge en la recuperación de los *chunks*. Un motor de búsqueda típicamente devuelve un número pequeño de resultados ordenados por relevancia - los 20 o 30 primeros. Pero una pregunta sobre perfiles cualificados podría tener 100 o 200 respuestas válidas en la base de datos. Recuperar solo los 20 primeros significa perder el 80% de la respuesta, aunque los perfiles excluidos sean perfectamente válidos
- El segundo emerge en la generación de la respuesta final. Incluso si se recuperaran 100 perfiles, añadirlos todos al *prompt* no es eficiente. La ventana de contexto del LLM es limitada, y si se rellena con una lista inmensa de perfiles, el modelo leerá los del principio y final, pero no tendrá en cuenta la mayor parte del contenido central. Esto se conoce como *Lost in the Middle* [28]

La solución es eliminar o reducir ambos cuellos de botella. En el primero, se propone que el número de *chunks* que se pueden recuperar sea mucho mayor de lo normal, priorizando cantidad en vez de calidad. Una vez se tiene este gran conjunto de documentos, se normalizan los *scores* de relevancia de cada *chunk* a una escala entre 0% y 100%. Basándose en esos *scores*, los perfiles se agrupan en cinco tramos de fiabilidad. Los perfiles en el tramo de fiabilidad más alto probablemente sean respuestas válidas por sí solos; los de los tramos intermedios requieren validación adicional; los del tramo más bajo son perfiles marginales que podrían no contener información relevante para la respuesta. Para todos los perfiles, se ejecutan consultas en paralelo a un LLM mini. En estas llamadas se define un número máximo de *chunks* por llamada, agrupando los *chunks* de cada tramo de fiabilidad en lotes. Finalmente, todos los resultados se condensan con una llamada a un LLM – idealmente de mayor capacidad – que realiza una selección final y ordena los elementos de la respuesta por fiabilidad.

Durante todo el proceso, se mantiene un registro detallado de cómo se llegó a cada resultado: qué tramo produjo cada perfil, qué modelo lo validó, qué puntuación tiene. Este histórico es fundamental para la auditabilidad y reproducibilidad del sistema.

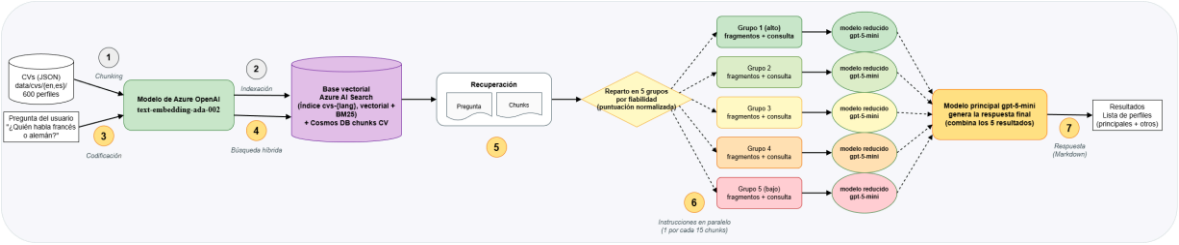

Figura 5. Flujo de estrategia RAG procesamiento en paralelo

3.3 IMPLEMENTACIÓN

La implementación concreta aterriza el diseño anterior sobre una pila mixta de software y servicios cloud. El objetivo no fue solo tener un prototipo funcional, sino un sistema modular, auditable y reproducible para comparar variantes de RAG. A continuación, se muestra una lista con las tecnologías y recursos empleados:

- *Backend*: Python, FastAPI, LangGraph/LangChain
- Datos: Azure Blob Storage, Azure Cosmos DB
- *Retrieval*: Azure AI Search (léxica + vectorial)
- Modelos: Azure OpenAI (embeddings + generación)
- Extracción documental: Azure Document Intelligence
- Conocimiento estructurado: Neo4j + Graphiti
- Frontend de pruebas: React

Independientemente de la estrategia elegida (clásica, basada en grafos o paralela), el ciclo completo del proyecto sigue una estructura clara que separa las fases de preparación, ejecución y validación. En primer lugar, se realiza el preprocesado de datos: generación o descarga de documentos de fuentes primarias, chunking adaptado a cada caso de uso, y carga en las bases de datos correspondientes. Esta fase es desacoplada del resto; sus *scripts* se

ejecutan una sola vez o periódicamente, pero no interactúan directamente con la lógica de consulta.

El sistema RAG recibe una pregunta del usuario, elige la estrategia de adecuada según el caso de uso y el idioma, y ejecuta el proceso correspondiente. En todos los casos, el sistema registra metadatos intermedios: qué documentos se recuperaron, qué modelo se usó, cuánto tiempo tardó la ejecución.

Finalmente, la fase de evaluación compara las respuestas generadas contra *gold standards* predefinidos. Estos archivos son conjuntos de preguntas y respuestas para poner a prueba el sistema. La reproducibilidad se mantiene porque cada *script* de generación de datos produce salidas deterministas o auditables, la configuración del sistema está centralizada, y los resultados intermedios se almacenan, permitiendo examinar cualquier punto del proceso. Esta estructura modular y trazable es crítica para comparar objetivamente las tres estrategias de RAG y entender qué mejoras proporciona cada extensión.

3.3.1 ESTRUCTURA DEL CÓDIGO

La organización del repositorio sigue una estructura basada en responsabilidades. A continuación, se describe cada módulo en detalle:

- ***src/generate_data.*** Contiene *scripts* independientes de generación y descarga de datos por caso de uso: *generate_cvs.py* coordina la generación sintética de currículums mediante Azure OpenAI a partir de plantillas estructuradas; *generate_wiki.py* y *generate_eu.py* descargan y normalizan artículos de Wikipedia y documentos de legislación de la UE respectivamente. *cv_dataloader.py* gestiona la carga incremental desde Azure Blob Storage, *validate_data.py* verifica la integridad y coherencia del conjunto antes de la ingesta en Azure, y *runner.py* ofrece un punto de entrada unificado para ejecutar el flujo de generación completo
- ***src/document_ingestion:*** Abarca el proceso de *chunking*, generación de *embeddings* y carga de documentos a las bases de datos. Se organiza en tres subcarpetas por caso de uso, cada una con cuatro *scripts* especializados: *doc_chunking_*.py* implementa la

estrategia de *chunking* adaptada al dominio; *processchunks_*.py* genera los *embeddings* con Azure OpenAI y enriquece los metadatos de cada *chunk*; *create_index_*.py* crea y configura los índices en Azure AI Search con los campos necesarios para las búsquedas vectorial y semántica; y *runner_*.py* orquesta las fases anteriores. Además, *processchunks.py* es un *script* común dentro de la carpeta que contiene funciones de preprocesado compartidas entre los tres dominios, mientras que *graphiti_wiki.py* y *graphiti_eu.py* se encargan de migrar los *chunks* de Azure a un grafo de conocimiento en Neo4j mediante Graphiti. De esta forma, los objetivos de *embeddings*, indexación híbrida y enriquecimiento con metadatos no quedan enunciados de forma teórica, sino implementados como fases explícitas y trazables del flujo de ejecución.

- ***src/services.*** Contiene *scripts* para facilitar el uso de los servicios de Azure. *openai_service.py* centraliza las llamadas a Azure OpenAI para *embeddings* y chat; *cosmos_service.py* gestiona la lectura y escritura en Azure Cosmos DB, incluyendo historial de conversaciones y metadatos de consulta; *azure_storage_service.py* proporciona acceso a Azure Blob Storage para la carga y descarga de documentos originales; y *docintelligence_service.py* encapsula las llamadas a Azure Document Intelligence para la extracción de texto de PDFs estructurados
- ***test.*** La carpeta ***gold_standard*** contiene ficheros *.xlsx* con pares pregunta-respuesta de referencia para los tres casos de uso, junto con los *scripts* de generación por idioma y dominio, *generate_gold_standard_*.py*. En la carpeta ***scripts***, *execute_tests.py* ejecuta el conjunto de pruebas, *evaluation.py* evalúa los resultados generados y *pricing.py* centraliza el cálculo de coste económico por tokens para ambas fases. Esta capa de *testing* proporciona una forma de *benchmarking* reproducible, ya que permite comparar variantes del sistema bajo condiciones similares junto con métricas de calidad y eficiencia homogéneas
- ***frontend.*** Es la parte visible de la aplicación, y se organiza en componentes independientes: *ChatInterface* gestiona el panel principal de conversación; *Sidebar* permite navegar por el historial de conversaciones; *UseCaseSelector* y *LanguageSelector* exponen los parámetros de configuración del modo de RAG al usuario; *LoginScreen* autentica la sesión; y *AssistantManager* coordina la comunicación

con el *backend* vía las llamadas *REST* definidas en *config.js*. A continuación, se muestra una imagen del diseño web del sistema. En el Anexo A.3 se muestran más ejemplos

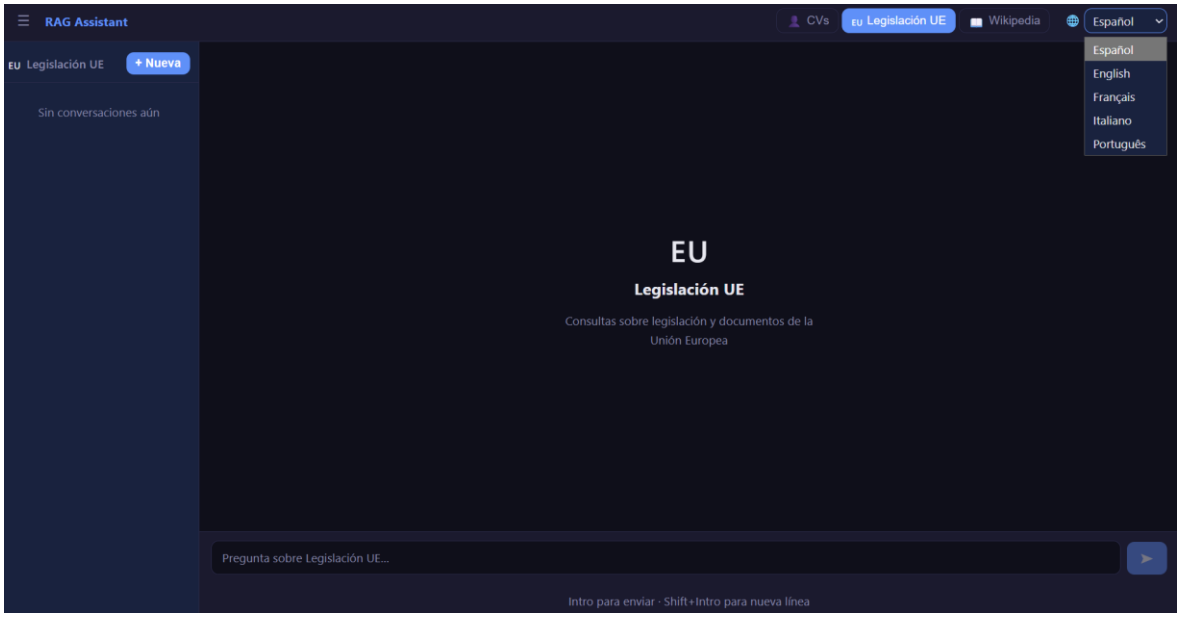

Figura 6. Interfaz del asistente RAG — caso de uso Legislación UE, con el selector de idioma desplegado mostrando las cinco opciones disponibles (Español, English, Français, Italiano, Português)

- ***src/rag.*** Es el módulo más complejo, articula la lógica de los tres modos de RAG sobre una base común de LangGraph y FastAPI. Contiene los siguientes elementos y submódulos:
  - ***main.py:*** servidor FastAPI que expone los *endpoints REST* del *backend*, como */chat*, */history* y */reset*, gestiona CORS (*Cross-Origin Resource Sharing*, mecanismo de seguridad de los navegadores web) y controla la concurrencia mediante hilos
  - ***app_langgraph.py:*** define el grafo de ejecución LangGraph. declara el tipo de estado compartido, *RAGState*, registra los nodos del grafo (validación de entrada, clasificación de contexto, recuperación, etc), y compila el grafo guardando el estado de la conversación en memoria
  - ***guardrails.py:*** módulo de validación de entradas y salidas. *InputGuardRails* analiza la consulta del usuario antes de llegar a la fase de búsqueda, detectando contenido fuera de dominio, intentos de inyección de *prompt* y peticiones inapropiadas. *OutputGuardRails* revisa la respuesta generada antes de devolverla al cliente

- ***handler/:*** implementa el patrón *Template Method* para los tres casos de uso, que permite definir el esqueleto de un algoritmo en una clase "padre" y que otras subclases sobrescriban algunos pasos del algoritmo[30]. base.py define el esqueleto del sistema, incluyendo la construcción del *prompt* de generación y configuración de la búsqueda; y el resto de los archivos modifican los métodos base en función de cada caso de uso
- ***strategies/:*** encapsulan la lógica de cada estrategia de RAG. base.py define la interfaz abstracta *BaseStrategy*. *basic_fusion.py* implementa la estrategia clásica: genera preguntas sintéticas mediante RAG Fusión y lanza una búsqueda híbrida en Azure AI Search. *graph_rag.py* dirige la búsqueda sobre el grafo de conocimiento Neo4j a través de Graphiti, combinando búsqueda por proximidad de nodos, recorrido trasversal de relaciones y recuperación de episodios; y normaliza el resultado de la búsqueda al formato de *chunk* interno. *cvs_parallel.py* implementa la estrategia de procesamiento en paralelo para el caso de uso CVs
- ***prompts/:*** los ficheros de esta carpeta construyen de forma modular los *prompts* usados en cada parte del sistema
- ***models/:*** capa de acceso a los LLMs. *generator.py* construye el mensaje final y llama a Azure OpenAI. *retriever.py* ejecuta las consultas en Azure AI Search, gestionando la paginación, el filtrado por metadatos y la fusión de resultados cuando la búsqueda es híbrida
- ***helpers/:*** *helper_azure_search.py* contiene funciones auxiliares para construir filtros, formatear los resultados de búsqueda y gestionar la selección de campos devueltos por el índice
- ***utils/:*** contiene ficheros con funcionalidad adicional necesaria. *pricing.py* y *pricing_config.json* implementan el cálculo de coste por consulta en función del modelo utilizado y los *tokens* consumidos, permitiendo comparar el coste económico de las tres estrategias de RAG

Esta estructura permite que la misma infraestructura soporte tres flujos distintos de RAG simultáneamente. La coexistencia de estas tres estrategias dentro de una misma interfaz es

clave para el objetivo experimental del trabajo: demostrar en qué casos funciona mejor una estrategia u otra según el tipo de pregunta y caso de uso

3.3.2 SISTEMA RAG CLÁSICO

El RAG clásico actúa como punto de partida de la investigación porque implementa el flujo estándar, permitiendo comparar objetivamente las mejoras implementadas. Su arquitectura es simple: el usuario formula una pregunta, se valida la petición, se recuperan los *chunks* relevantes y, a partir de ellos, se genera la respuesta final.

En tiempo de ejecución, la consulta se envía al servicio de chat, que primero comprueba que la entrada sea válida y que el estado conversacional esté listo para continuar. A continuación, el orquestador construye el estado inicial del flujo y activa el grafo de ejecución. Ese grafo encadena las etapas de validación, selección de estrategia, clasificación del contexto y búsqueda de documentos. En el caso del sistema RAG clásico, la estrategia elegida es la que incluye RAG Fusión y búsqueda híbrida (enriquecida con el contexto relevante de la conversación), y se ejecuta para los tres casos de uso cuando el idioma seleccionado es español. Esta simplicidad estructural convierte al RAG clásico en un punto de partida útil: ofrece una referencia estable frente a la cual medir el valor añadido de los grafos de conocimiento y del procesamiento paralelo.

El diagrama siguiente muestra la secuencia completa de interacciones entre componentes del RAG desde que el usuario envía una consulta hasta que recibe la respuesta generada, detallando las llamadas a cada servicio externo en el orden en que se producen durante la ejecución.

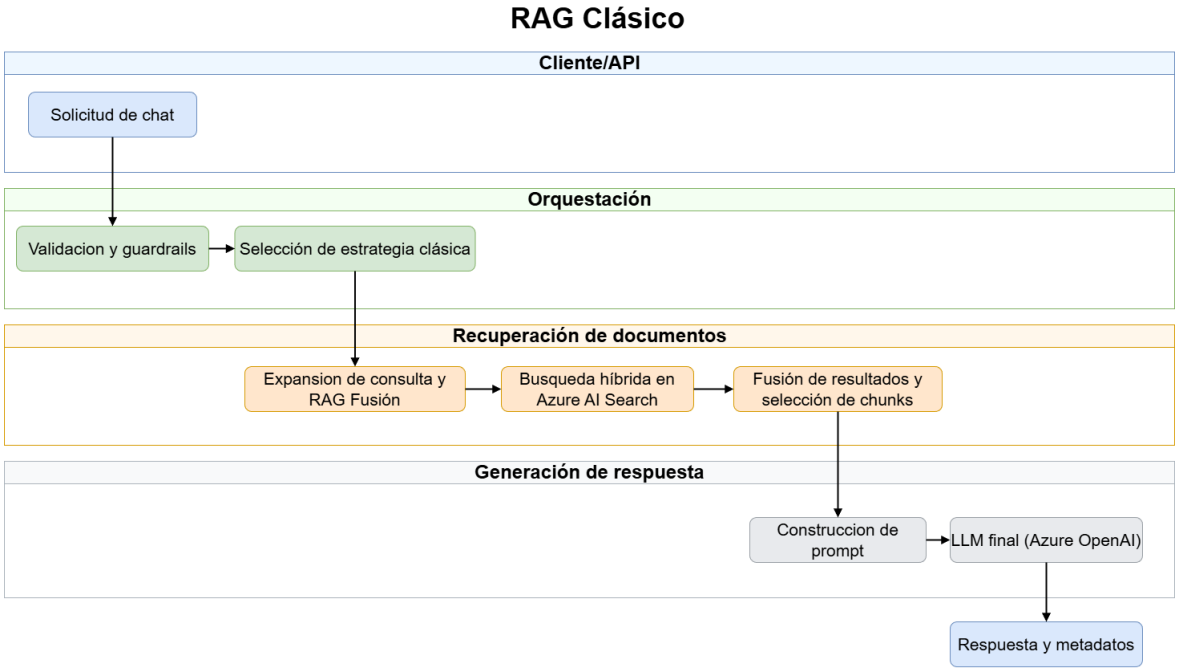

Figura 7. Flujo de llamadas en tiempo de ejecución del sistema RAG clásico

3.3.3 GRAFOS DE CONOCIMIENTO

La estrategia RAG basada en grafos mantiene la misma fase final de generación que el RAG clásico, pero cambia la forma de recuperar los chunks. En lugar de depender solo de la similitud textual entre la pregunta y documento, explota una capa de conocimiento estructurado construida sobre Neo4j y Graphiti. El objetivo es capturar relaciones, entidades y episodios que no siempre aparecen de forma evidente en la búsqueda híbrida original. En ese punto se determina qué base de datos debe consultarse y se accede al grafo de conocimiento estructurado mediante la librería Graphiti. La búsqueda sobre el grafo se ejecuta de forma asíncrona y puede combinar distintos modos de exploración en función de lo que se busque: conexiones entre nodos, relaciones explícitas o episodios documentales. El resultado devuelto por Graphiti no se utiliza de forma directa, sino que se normaliza para que encaje con el formato definido en el prompt de la llamada final al LLM. Así, la etapa de generación permanece unificada mientras que la recuperación gana expresividad semántica.

Esta arquitectura resulta especialmente útil cuando la respuesta depende de relaciones entre documentos, no solo de coincidencias locales. En vez de limitarse a encontrar el fragmento

más parecido, el sistema puede recorrer documentos conectados por entidades compartidas, referencias cruzadas o vínculos temáticos. El diagrama siguiente ilustra el proceso del sistema implementado a alto nivel. Además, se incluye la representación de la base de documentos de Wikipedia en el Anexo A.4.

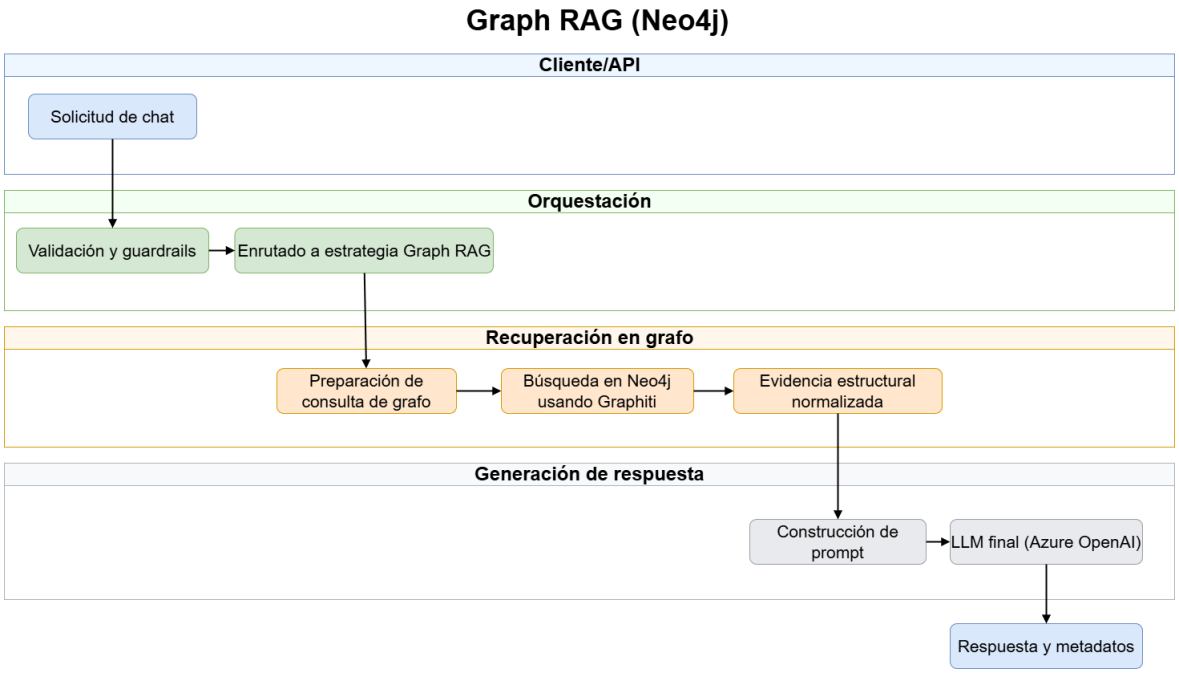

Figura 8. Flujo de llamadas en tiempo de ejecución del sistema Graph RAG con recuperación de documentos sobre grafo de conocimiento en Neo4j

3.3.4 PROCESAMIENTO EN PARALELO

La estrategia de procesamiento en paralelo se implementa con un flujo de ejecución propio. Como he comentado anteriormente, y a diferencia del RAG clásico y GraphRAG, esta estrategia no sigue el flujo estándar de ejecución de LangGraph hasta la generación final, sino que una parte de la funcionalidad del sistema se resuelve con una lógica totalmente personalizada.

El flujo operativo es el siguiente: primero, se realiza una búsqueda amplia sin RAG Fusión para obtener la mayor cantidad posible de *chunks*. Después, se normalizan los scores recibidos por Azure AI Search, mapeando cada uno a un percentil y clasificando los perfiles

en cinco tramos de fiabilidad. Cada tramo de fiabilidad se distribuye entre múltiples procesos de trabajo concurrentes. Cada proceso gestiona un lote pequeño de perfiles y ejecuta una llamada a un modelo de lenguaje pequeño (GPT-5-mini en este caso), enviando en cada llamada los *chunks* de cada lote junto con la pregunta del usuario y un *prompt* de instrucciones para decidir qué perfiles cumplen la consulta. Tras esta ejecución en paralelo, un LLM final recibe los resultados consolidados de esas llamadas, los ordena por fiabilidad y genera la respuesta final. Durante todo el proceso, además, se recuperan las trazas de las llamadas a los LLMs registrando métricas como la fuente de cada resultado, fiabilidad, tiempo de ejecución y coste. Veamos un ejemplo de ejecución de una pregunta real: "Busca todos los perfiles con experiencia en AWS y *Kubernetes*". El proceso es el siguiente:

1. Se recuperan 200 perfiles, priorizando cualquier CV que mencione al menos uno de estos términos
2. Los *scores* de Azure AI Search se transforman en una escala de 0 a 1 (en porcentaje)
3. Los 200 perfiles se distribuyen en cinco tramos de fiabilidad. Por ejemplo: 30 en el tramo 1 (el de mayor fiabilidad), 50 en el tramo 2, 40 en el tramo 3, 45 en el tramo 4 y 35 en el tramo 5
4. En cada tramo, los *chunks* se envían por lotes a llamadas al mini-LLM junto con la pregunta del usuario y el *prompt* de instrucciones, para extraer los perfiles que sí responden a la consulta
5. Los perfiles de todos los tramos se distribuyen entre ocho procesos de trabajo concurrentes. Cada proceso recibe lotes pequeños (10-15 perfiles por lote) y lanza esas llamadas al mini-LLM en paralelo
6. Una vez que todos los procesos terminan, se consolidan los resultados. Se eliminan duplicados, se reordenan por fiabilidad y se genera la respuesta final
7. El historial captura que la consulta produjo 45 perfiles válidos, distribuidos por tramo de fiabilidad y con trazas completas de cómo se identificó cada uno

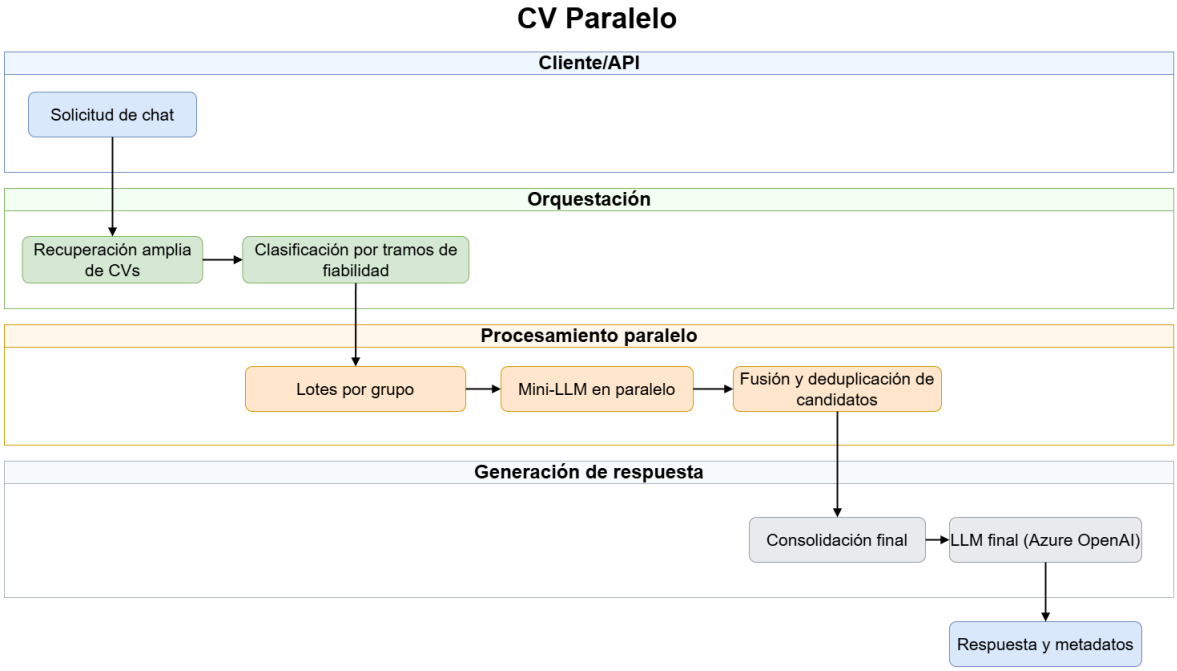

Figura 9. Flujo de llamadas en tiempo de ejecución de la estrategia de procesamiento en paralelo

En conjunto, esta estrategia transforma un problema de cobertura en un proceso controlado de recuperación y validación progresiva. Al separar recuperación masiva, clasificación por fiabilidad y consolidación final, el sistema mejora la completitud sin saturar la ventana de contexto del modelo principal, y mantiene trazabilidad sobre cada decisión.

4. RESULTADOS

Este capítulo describe cómo se han validado experimentalmente las soluciones desarrolladas y cómo deben interpretarse los resultados obtenidos. El objetivo de la evaluación no fue únicamente comprobar si el sistema era capaz de responder preguntas de forma aparentemente correcta, sino determinar de manera reproducible si las mejoras introducidas resuelven las limitaciones que motivaron el trabajo: respuestas incompletas cuando se piden listas extensas y dificultades para conectar información dispersa en varios documentos.

Para ello se diseñó un proceso de evaluación. Se construyó un conjunto de preguntas y respuestas de referencia para cada caso de uso, y se ejecutó sobre todas las estrategias RAG del sistema. El objetivo es analizar si, en general, las estrategias implementadas mejoran las respuestas del sistema en todos los casos; teniendo en cuenta que había fallos sistemáticos en cada caso de uso y que lo que se busca es corregirlos sin generar otros nuevos. La evaluación se articula en dos niveles complementarios. En primer lugar, se mide la calidad de la respuesta mediante señales semánticas y de cobertura. En segundo lugar, se registran métricas operativas, como tiempo de respuesta, volumen de contexto recuperado, número de llamadas al modelo y coste aproximado por consulta. De este modo, el capítulo no se limita a responder si una variante acierta más, sino también si lo hace con un coste y una complejidad razonables. Este enfoque está directamente relacionado con los objetivos definidos al inicio del trabajo: no se busca validar solo la precisión de las respuestas, sino también demostrar trazabilidad, robustez y viabilidad operativa de la arquitectura.

4.1 DISEÑO

El diseño de la evaluación parte de una limitación habitual en proyectos de IA aplicada: es fácil sobreestimar la calidad del sistema dependiendo de las pruebas realizadas. Un sistema RAG puede responder con gran fluidez y, sin embargo, omitir partes de la respuesta, mezclar documentos incorrectamente o producir una salida que parezca correcta sin serlo. Por este

motivo, la evaluación del sistema implementado comienza definiendo casos de uso y *gold standards* específicos para cada uno. En este contexto, un *gold standard* es un conjunto estructurado de preguntas y sus respuestas correctas definidas a partir de los datos originales. La necesidad de crear *gold standards* en este trabajo responde a cuatro motivos:

- **Reproducibilidad.** Un conjunto fijo de preguntas y respuestas esperadas permite ejecutar muchas veces las mismas pruebas, comparar versiones del sistema y recalcular métricas con diferentes umbrales sin alterar la validez del experimento
- **Comparabilidad entre estrategias.** El trabajo analiza varias formas de búsqueda: RAG Fusión, grafos de conocimiento y una búsqueda de máxima cobertura
- **Cobertura de los fallos relevantes.** Uno de los problemas de partida del proyecto era que los sistemas RAG convencionales responden razonablemente bien a preguntas acotadas, pero empeoran cuando se les pide enumerar todos los elementos que cumplen una condición, recuperar hechos repartidos entre varios documentos o mantener consistencia entre idiomas. Por tanto, el *gold standard* incluye todo tipo de preguntas, priorizando aquellas en las que el sistema mostraba debilidades
- **Reducción de subjetividad.** En preguntas abiertas, decidir si una respuesta es correcta puede depender de la persona que lo evalúa, del estilo de redacción o del nivel de detalle admitido. Al construir respuestas esperadas directamente a partir de los datos originales, y al combinar después esa referencia con métricas automáticas de comparación semántica, la evaluación es más robusta

Los *gold standards* se diseñaron siguiendo tres principios metodológicos comunes a todo el proyecto. El **primero** fue que la respuesta esperada proviene del dato original. Esto reduce el riesgo de introducir errores humanos en la referencia y permite que la batería de preguntas crezca con coherencia cuando aumenta el tamaño de la base de datos. El **segundo** fue cubrir distintos rangos de dificultad en cada conjunto de preguntas: algunas exigen localizar una entidad concreta, otras obligan a enumerar múltiples resultados, combinar criterios lógicos, cruzar varios documentos o recuperar estructura documental. El **tercer principio** consistió en preservar el carácter multiidioma del experimento. La evaluación no podía construirse solo en español y después asumir que el sistema generalizaba al resto de idiomas. Se

generaron versiones lingüísticamente equivalentes para cada idioma, garantizando robustez de la evaluación.

El diseño basado en *gold standards* aporta varias ventajas. La más importante es que obliga a evaluar el sistema sobre preguntas difíciles y no solo sobre ejemplos favorables. Además, separa claramente las fases de construcción y ejecución del *testing*, lo que mejora la trazabilidad experimental. No obstante, es importante recordar que ni con un *gold standard* es posible abarcar todos los escenarios reales, por lo que será necesario implementar un sistema de recolección de feedback si el sistema se despliega en producción para mantener la precisión de las respuestas. También existe un conflicto entre exhaustividad y naturalidad: cuanto más precisa es la respuesta esperada, más estricto se vuelve el criterio de evaluación frente a respuestas parcialmente correctas o redactadas de otra forma. El proyecto incluye un *benchmark* más exigente para evitar falsas mejoras en el rendimiento del sistema.

4.2 IMPLEMENTACIÓN

El diseño del sistema descrito en el capítulo anterior fue pensado precisamente para facilitar la evaluación: para cada caso de uso, en español, está implementado un sistema RAG clásico; en inglés (y en el resto de los idiomas) está implementada la mejora correspondiente. De esta forma, al comparar los resultados en español e inglés, realmente se compara la efectividad de cada estrategia. En concreto, se ha generado un *gold standard* para cada caso de uso de aproximadamente 100 preguntas y respuestas con el objetivo de garantizar la fiabilidad de los resultados. Estos conjuntos han sido generados mediante *scripts* construidos con ayuda de IA; y revisados posteriormente para garantizar que las preguntas de todos los ficheros están bien formuladas, y que sus respuestas son coherentes, correctas y completas.

Una vez definidos estos conjuntos de preguntas y respuestas, la evaluación se ejecutó mediante un protocolo en dos fases independientes. Esta separación permite distinguir entre el comportamiento del sistema durante la ejecución y el proceso posterior de evaluación de métricas. Además, cualquier error que pudiera ocurrir no afectaría a todo el proceso, reduciendo posibles fallos durante la ejecución del *testing*.

En la primera fase, cada pregunta del *gold standard* se lanzó de forma independiente contra el sistema. Dicha independencia entre consultas es importante para evitar que el historial de conversación contamine los resultados, y para asegurar que cada pregunta se evalúa exclusivamente en función de su propia capacidad de recuperación y generación. En esta etapa, el sistema selecciona la estrategia RAG correspondiente al caso de uso y al idioma, recupera los *chunks* relevantes, genera la respuesta final y registra una traza estructurada de la consulta completada. La traza incluye la respuesta generada, así como tiempo total de respuesta, número de fragmentos recuperados, número de fragmentos finalmente utilizados como contexto, número de llamadas realizadas al modelo, volumen de *tokens* consumidos de entrada y salida, y coste estimado de la consulta; es decir, información suficiente para analizar el rendimiento del sistema desde varias perspectivas. También se registran los errores de ejecución, si los hubiera. Esta trazabilidad convierte la evaluación en algo más que un registro de las respuestas: permite estudiar el comportamiento interno del sistema y cuantificar el precio de cada consulta.

En la segunda fase, las respuestas generadas por el sistema se evalúan automáticamente utilizando ciertos criterios y métricas. Esta decisión tiene dos ventajas. Por un lado, evita tener que repetir consultas costosas cada vez que se modifica un criterio de evaluación. Por otro, permite aplicar de manera uniforme las mismas métricas a todos los resultados ya generados.

Gracias a este diseño, el objetivo de evaluación sistemática no se limita a una foto puntual del sistema, sino que queda preparado para repetir los análisis de manera consistente cuando cambian umbrales, métricas o hipótesis de comparación.

4.2.1 MÉTRICAS DE EVALUACIÓN

La calidad de la respuesta se midió mediante dos señales cuantitativas complementarias.

- **Coincidencia con la respuesta esperada**. Esta métrica intenta responder a la siguiente pregunta: ¿hasta qué punto la salida del sistema cubre las mismas entidades, hechos o elementos que la respuesta esperada? Para estimarla se realiza una llamada a un LLM

que compara, para cada pregunta, las respuestas esperada y generada; asignando un porcentaje de cobertura. Este porcentaje valora qué cantidad de la información que presentan ambas respuestas coincide.

- **Relevancia semántica**. En este caso se evalúa si la respuesta obtenida responde de forma coherente y pertinente a la pregunta formulada, usando la llamada anterior al LLM

De esta forma, se prioriza completitud y fiabilidad por encima de respuestas que parecen correctas, que era precisamente una de las debilidades del sistema de partida.

Sobre estas dos métricas continuas se estableció un mecanismo de evaluación binario (OK/KO). Una respuesta se considera correcta (OK) únicamente cuando supera simultáneamente el umbral fijado en coincidencia y en relevancia semántica. En la evaluación final se empleó un umbral del 65 %. Este valor establece una frontera homogénea para comparar las distintas estrategias. La clave está en que el sistema no obtiene un OK por ser parcialmente correcto en una de las dos dimensiones: necesita ser suficientemente próximo a la referencia y, al mismo tiempo, cubrir la información esperada con una calidad mínima.

En este trabajo, la completitud se evalúa como una propiedad incorporada en el diseño del *benchmark*. Esto es especialmente visible en las preguntas tipo lista. Cuando la respuesta esperada es un conjunto de entidades, pero el sistema devuelve solo una parte de ellas, la relevancia semántica puede seguir siendo relativamente alta porque se está respondiendo a la pregunta de forma razonable. Sin embargo, la coincidencia disminuye, ya que la respuesta obtenida es incompleta. Esta distinción es importante porque refleja el problema original del proyecto: si una consulta pide todos los perfiles que cumplan ciertas características y el sistema devuelve solo algunos, el resultado deja de ser útil incluso si los elementos mostrados son correctos y coherentes con la pregunta.

Además de las métricas de calidad, se registraron otras variables que influyen para determinar qué estrategia es mejor. Entre ellas destacan el tiempo total por consulta, el número de documentos recuperados, el número de *chunks* realmente utilizados para generar la respuesta, el número de llamadas a LLMs junto con métricas específicas de cada llamada,

y el consumo de *tokens* de entrada y salida. A partir de estos datos se estimó el coste económico tanto de la generación de respuestas como de la propia evaluación. Estas son relevantes porque una mejora en precisión que multiplicara desproporcionadamente el coste o la latencia tendría un valor limitado en un entorno profesional. El objetivo del trabajo no era únicamente aumentar el porcentaje de aciertos, sino analizar si las estrategias que mejoran la respuesta son viables para implementarlas en un entorno empresarial.

4.3 ANÁLISIS DE RESULTADOS

Una vez ejecutado el protocolo experimental, los resultados deben interpretarse desde una perspectiva que combine calidad de respuesta, comportamiento operativo y adecuación de cada estrategia al problema que pretende resolver. Un porcentaje global de aciertos ofrece una primera aproximación útil, pero no basta para explicar por qué una variante funciona mejor que otra ni en qué condiciones compensa asumir un mayor coste computacional. En este trabajo, el interés no reside solo en determinar qué sistema obtiene más respuestas aceptadas, sino en analizar si las mejoras introducidas corrigen fallos concretos del RAG clásico y si lo hacen de una manera coherente con el tipo de documento y de consulta evaluados, teniendo siempre presente el valor empresarial de las soluciones implementadas.

Este apartado de análisis se centra, por tanto, en evaluar las dos estrategias implementadas. Por un lado, la recuperación basada en grafos de conocimiento se estudia en los casos de Wikipedia y Legislación UE, donde el reto principal consiste en conectar información distribuida entre documentos y capturar relaciones que una búsqueda puramente semántica puede pasar por alto. Por otro lado, el procesamiento en paralelo se analiza en el caso de CVs, donde el problema dominante es la necesidad de recuperar de forma exhaustiva todos los perfiles que cumplen una o varias condiciones. Ambas estrategias persiguen mejorar el rendimiento del sistema, pero se centran en limitaciones diferentes; por ello, su evaluación debe leerse en términos de adecuación al caso de uso.

4.3.1 GRAFOS DE CONOCIMIENTO

La estrategia basada en grafos de conocimiento se diseñó para sustituir la búsqueda híbrida clásica por recuperación estructural sobre nodos y relaciones. El objetivo no era únicamente mejorar el acierto en consultas directas, sino reducir fallos en preguntas que exigen conectar información distribuida entre documentos, especialmente en tareas de referencia cruzada y relaciones entre entidades. Para evaluar esta hipótesis, se analizan los resultados obtenidos sobre los siguientes casos de uso:

- **Wikipedia.** Se escogió un conjunto cualquiera de artículos de Wikipedia para analizar la capacidad de Graphiti de inferir relaciones entre documentos sencillos, de conocimiento general y conceptos "fáciles de entender". Con estos datos se busca analizar la mejora que suponen los grafos de conocimiento en un contexto sencillo. En el *gold standard*, se combinan preguntas de definición de conceptos, verificación de existencia, relaciones entre artículos, búsqueda por contenido y preguntas cruzadas entre varios documentos. Esta mezcla permite evaluar desde tareas sencillas, como recuperar la definición inicial de un documento; hasta tareas más exigentes, como relacionar conceptos
- **Legislación UE.** Para probar el sistema en un entorno complejo, se seleccionaron algunas entradas del Diario Oficial de la Unión Europea[27]. Se escogieron tres días aleatorios con documentos lo suficientemente válidos para hacer esta prueba. Estos documentos son totalmente inconexos e incompletos, además de incluir términos muy específicos sobre la normativa europea, dificultando la inferencia de las relaciones entre los nodos del grafo de conocimiento. En este caso, el objetivo de la evaluación no es solo comprobar si el sistema recupera los documentos correctos, sino si también su capacidad para manejar estructura documental compleja y referencias cruzadas. No se esperan buenos resultados, ya que se consultan documentos sueltos y sin contexto. Sin embargo, con este caso de uso se busca llevar al sistema al límite; es decir, se evalúa si los grafos de conocimiento mejoran las respuestas incluso cuando la base de datos es extremadamente inconexa y las relaciones entre documentos no existen o es muy

complicado inferirlas. La falta de un experimento con todos los documentos de la fuente citada se debe a la alta carga computacional y coste que eso supondría.

Las preguntas generadas y sus respectivas respuestas se derivan de propiedades observables en los propios documentos: tipo de disposición, organismo emisor, fecha, número de artículos, longitud aproximada, presencia de referencias a otros actos y contenido textual específico. A partir de estos elementos se formularon consultas que obligan al sistema a operar en varios niveles: identificación de metadatos, recuperación de contenido concreto, comprensión de la estructura interna y relación entre documentos distintos. Este diseño es clave para abordar las limitaciones del RAG clásico, que aparecen en preguntas que exigen conectar información distribuida. Una consulta sobre referencias cruzadas o sobre relaciones entre actos legislativos no se resuelve necesariamente con el *chunk* más parecido en términos semánticos; requiere capturar las dependencias entre los documentos relevantes.

A continuación, se analizan en detalle los resultados obtenidos:

Idioma	Nº preguntas	OK	KO	Tiempo medio (segundos)	Coste medio (USD)[4]	% OK
ES	100	78	22	25,077	0,01010362	78.0%
EN	100	82	18	16,94	0,00932473	82.0%

Tabla 1. Resultados del caso de uso Wikipedia: comparación entre RAG clásico (ES) y Graph RAG (EN), con métricas de calidad, latencia media y coste por consulta

En Wikipedia se observan muy buenos resultados en ambos idiomas. Se observa un mayor porcentaje de respuestas correctas en el idioma que implementa la estrategia de grafos de conocimiento (EN) frente al sistema RAG clásico (ES). En términos de calidad, pasa de un 78,0 % de OK a un 82,0 %, lo que supone una mejora de 4 puntos porcentuales. Además, la

[4] USD: dólares estadounidenses. Coste estimado basado en los precios de Azure OpenAI al momento de la evaluación

latencia media desciende de 25,08 s a 16,94 s, una reducción del 32,4 %; y el coste medio baja de 0,01010362 USD a 0,00932473 USD, aproximadamente un 7,7 % menos.

Estos resultados son coherentes y esperables: la información de los artículos descargados es de conocimiento general, y las relaciones entre los documentos son sencillas de inferir. Cuando existen relaciones semánticas relativamente evidentes entre documentos, la estructura de grafo permite seleccionar mejor el contexto útil y evita parte del ruido de recuperación. En consecuencia, no solo aumenta la tasa de respuestas aceptadas, sino que además disminuye el coste por pregunta. La mejora implementada afecta tanto a la calidad de la respuesta como al tiempo que tarda en generarse, demostrando que los grafos de conocimiento son una mejora respecto a la búsqueda híbrida dentro de los sistemas RAG.

Idioma	Nº preguntas	OK	KO	Tiempo medio (segundos)	Coste medio (USD)	% OK
ES	91	47	44	28,217	0,00566158	51.6%
EN	98	60	38	30,346	0,00630499	61.2%
FR	98	54	44	30,513	0,00668518	55.1%
IT	99	56	43	32,083	0,0068847	56.6%
PT	91	54	37	31,062	0,00649948	59.3%

Tabla 2. Resultados del caso de uso Legislación UE: comparación entre RAG clásico (ES) y Graph RAG (EN, FR, IT, PT), con métricas de calidad, tiempo medio y coste por consulta

En Legislación UE, el resultado también es positivo, pero más moderado y con un compromiso de eficiencia más exigente. El sistema RAG clásico (ES) alcanza un 51,6 % OK, mientras que el resto de los idiomas, que implementan la estrategia de grafos, obtienen desde un 55,1 % y hasta un 61,2 %. Estos valores muestran un promedio del 58,0 % de aciertos, que representa una mejora media de 6,4 puntos porcentuales frente al sistema base. Sin embargo, esta mejora de calidad viene acompañada de mayor coste computacional. El tiempo medio ponderado con grafos asciende a 31,00 s, frente a 28,22 s en español, y el coste medio ponderado sube hasta 0,00659605 USD, frente a 0,00566158 USD. No obstante, estos incrementos son pequeños comparado con la mejora de las respuestas. Por tanto, en un

dominio normativo complejo, los grafos mejoran la robustez de recuperación, pero con un coste añadido. Este balance entre calidad y eficiencia muestra una discusión frecuente en cualquier entorno empresarial: un sistema que responde mejor a cambio de más recursos por consulta. Cabe resaltar que este incremento de costes es asumible en este caso: es preferible un sistema que tarda 3 segundos más en responder pero que aumenta la probabilidad de aciertos en un 6 %. Finalmente, se observa una variación de hasta 6 puntos porcentuales entre los distintos idiomas, demostrando que el rendimiento del sistema también depende de este factor.

La lectura conjunta de ambos casos de uso permite sostener, en primer lugar, que la estrategia de grafos aporta una mejora neta de precisión en ambos escenarios y, por tanto, su valor no depende de un único dominio. El beneficio es más claro en preguntas relacionales que en consultas puramente extractivas de un solo *chunk*, donde el RAG clásico ya era competitivo. Por último, en escenarios muy inconexos, el grafo ayuda, pero no compensa por completo la falta de contexto global: cuando faltan documentos clave o relaciones explícitas, alcanzar un rendimiento alto es poco factible. También deben mencionarse dos hechos que dificultan la evaluación de la estrategia. La primera es la posible confusión entre estrategia e idioma: en este diseño, el español actúa como RAG clásico y los otros idiomas incorporan la mejora, por lo que una parte de la diferencia observada puede estar influida por variaciones lingüísticas y no solo por la arquitectura. La segunda es la cobertura documental parcial en UE: al trabajar con muestras de días concretos y no con un período completo del Diario Oficial, muchas referencias externas no están presentes, lo que reduce artificialmente la capacidad del grafo para cerrar cadenas de evidencia.

En conjunto, los resultados sostienen que los grafos de conocimiento mejoran la calidad de respuesta de forma consistente en tareas de recuperación relacional, especialmente cuando la pregunta exige conectar trozos de información dispersos. No obstante, el beneficio depende de la complejidad de los documentos originales, y de su procesamiento e ingesta.

4.3.2 PROCESAMIENTO EN PARALELO

La estrategia de procesamiento en paralelo se aplica al caso de uso de CVs. Esta elección viene del objetivo inicial del proyecto: solucionar un problema interno del sistema RAG que utiliza una empresa para redactar listados de perfiles que comparten una o varias aptitudes.

Por ello, se construyó un *gold standard* orientado explícitamente al problema de la completitud. La base de datos está formada por CVs sintéticos y estructurados, lo que permite conocer con precisión qué perfiles cumplen cada condición. Sobre este conjunto se generó una batería de preguntas y respuestas esperadas en español e inglés, alineadas entre sí en contenido y orden, de modo que ambas versiones evalúan exactamente el mismo fenómeno de recuperación. Las preguntas incluyen categorías como búsqueda por *skill*, puesto, idiomas o experiencia; otros elementos mencionados en el CV y preguntas con múltiples criterios lógicos (por ejemplo, "dime todos los perfiles que cumplen X o Y"). Las preguntas más sencillas permiten comprobar si el sistema localiza correctamente una característica concreta, mientras que las preguntas multicriterio fuerzan a que la búsqueda sea exhaustiva y que se recuperen todos los perfiles válidos.

Idioma	Nº preguntas	OK	KO	Tiempo medio (segundos)	Coste medio (USD)	% OK
EN	86	64	22	65,462	0,05374904	74.4%
ES	86	37	49	36,52	0,0050176	43.0%

Tabla 3. Resultados del caso de uso CVs: comparación entre RAG clásico (ES) y pipeline paralelo (EN) sobre 86 preguntas, con métricas de calidad, tiempo medio y coste por consulta

La Tabla 3 muestra una mejora muy clara en calidad, pero acompañada de un incremento elevado de coste computacional. En inglés (EN), donde se implementa el procesamiento en paralelo, el sistema alcanza un 74,4 % de OK. En español (ES), con la estrategia base, el resultado cae a un 43,0 %. El incremento es de 31,4 puntos porcentuales. Esta diferencia es demasiado grande como para atribuirla a una fluctuación menor del experimento y apunta a que la estrategia ataca de forma efectiva el problema de completitud para el que fue diseñada. En CVs, muchas preguntas no consisten en identificar un único dato puntual, sino en

enumerar todos los perfiles que cumplen una condición o una combinación de condiciones. En este tipo de consultas, una respuesta parcialmente correcta sigue siendo insuficiente desde el punto de vista del proyecto. Precisamente por eso, el procesamiento en paralelo aporta valor: al lanzar varias búsquedas en paralelo y condensar después la información, se reduce la probabilidad de que perfiles válidos queden fuera de la respuesta final. Dicho de otro modo, la estrategia mejora la completitud de las respuestas, que era la debilidad más importante del sistema RAG en este caso de uso.

Sin embargo, esta mejora tiene una gran desventaja. El tiempo medio por consulta pasa de 36,52 s a 65,462 s, lo que supone un incremento aproximado del 79,2 %. El coste medio también crece de forma significativa: de 0,0050176 USD a 0,05374904 USD por consulta, es decir, diez veces más. Esto confirma que la estrategia logra una mejora clara en exhaustividad, pero lo hace a costa de un aumento notable en el consumo de recursos. Todos los LLMs tienen un sistema de coste por uso, es decir, se cobra por cada llamada y en función del número de tokens de entrada y salida. Para mejorar la completitud, en esta estrategia se elevó el límite de tokens de salida respecto al flujo base, permitiendo que la respuesta final enumerase todos los perfiles posibles sin aumentar en exceso el coste computacional. El resultado es sólido, pero plantea una decisión de diseño importante: si se valora especialmente la completitud, como ocurre en la generación de listados de perfiles, el sobrecoste puede estar justificado. En cambio, si el objetivo fuera responder preguntas simples donde la completitud no fuera un requisito indispensable, probablemente este aumento de costes no compensaría. Con todo, su comportamiento experimental es coherente con el objetivo para el que fue implementado.

Igual que ocurre en el apartado anterior, la arquitectura está acoplada al idioma. Aunque las preguntas y respuestas del *gold standard* estén alineadas, esto introduce una amenaza a la validez que conviene reconocer explícitamente: una parte del efecto observado podría verse influida por diferencias lingüísticas o por el comportamiento del modelo entre idiomas, y no exclusivamente por la arquitectura. Aun así, la magnitud de la diferencia, junto con la coherencia entre el objetivo de diseño y el patrón de resultados, hace razonable concluir que el procesamiento en paralelo sí aporta una mejora sustantiva y no meramente marginal.

5. CONCLUSIONES Y TRABAJOS FUTUROS

Este trabajo parte de un problema real: los sistemas RAG convencionales ofrecen respuestas razonables a preguntas sencillas, pero fallan cuando se exige completitud en respuestas largas o cuando la información está distribuida en varios documentos inconexos. A partir de ese problema se ha diseñado, implementado y validado una arquitectura RAG modular en un entorno *cloud* real, con tres estrategias RAG multiidioma comparables y una metodología experimental reproducible basada en *gold standards*.

El grado de cumplimiento de los objetivos es alto y consistente con el alcance definido al inicio del proyecto. El objetivo general, centrado en desarrollar y evaluar un sistema RAG multiidioma para comparar diferentes estrategias, se alcanzó mediante una arquitectura unificada que permitió comparar las distintas variantes implementadas y evaluar su eficacia por caso de uso e idioma, logrando también el objetivo de procesamiento multiidioma. En paralelo, se materializó el sistema de *embeddings* y búsqueda híbrida, organizada en índices por caso de uso y con mecanismos de sincronización reproducibles que facilitaron la trazabilidad experimental. Para la recuperación avanzada y representación del conocimiento, se implementaron y compararon técnicas como RAG Fusión, RRF y GraphRAG, incorporando la búsqueda de documentos contenidos en grafos. Esta capacidad se reforzó con el enriquecimiento de *chunks* mediante *keywords* y metadatos, lo que mejoró el filtrado, la trazabilidad y la recuperación de documentos tanto en calidad como en tiempos. Del mismo modo, la fase de generación con LLM quedó integrada con un control de salida basado en plantillas de *prompt*, *guardrails* y trazas de ejecución. El objetivo de evaluación y benchmarking integral se alcanzó con un proceso en dos fases que incluye métricas cuantitativas de calidad y eficiencia, desagregadas por caso de uso, idioma, estrategia y categoría de pregunta. Se exploraron estrategias experimentales de recuperación y generación, aunque existe posibilidad de mejora en optimización de costes de la estrategia de procesamiento paralelo para que un despliegue continuo a gran escala sea viable.

La aportación principal del proyecto reside en haber construido una arquitectura RAG modular y replicable, capaz de ejecutar tres estrategias diferentes en un mismo sistema, evitando sesgos de infraestructura y permitiendo comparaciones justas. A ello se suma una contribución enfocada en resolver dos limitaciones existentes de estos sistemas: la completitud de los listados en las respuestas extensas y la inferencia y recuperación de relaciones entre documentos inconexos. Otra aportación de peso es el marco de evaluación reproducible y exigente, desacoplado de la ejecución principal del sistema y basado en *gold standards* por caso de uso e idioma, usando criterios de aceptación explícitos, ajustables y auditables. Este enfoque permitió integrar en un mismo análisis varias métricas, de modo que cada mejora puede evaluarse no solo por su impacto en el porcentaje de aciertos, sino también por su efecto en tiempo y coste. En consecuencia, el trabajo ofrece evidencia cuantitativa de las estrategias implementadas: mejorar cobertura y capacidad de razonamiento puede requerir un incremento de recursos por consulta. Finalmente, el desarrollo sobre servicios gestionados y mecanismos reproducibles refuerza la transferibilidad del sistema al entorno profesional y facilita su adopción en contextos empresariales.

La principal conclusión es que no existe una única estrategia de recuperación óptima para todas las casuísticas. El estudio realizado confirma que el rendimiento depende del tipo de pregunta, del dominio documental y del compromiso aceptado entre cobertura, coste y latencia. Desde el punto de vista metodológico, el trabajo demuestra que el uso de *gold standards* junto con métricas de coincidencia, relevancia semántica y señales operativas (tiempo, *tokens*, coste), ha permitido analizar el comportamiento real del sistema y evitar conclusiones sesgadas por un mal diseño de *testing*.

Aunque los resultados obtenidos cumplen los objetivos planteados, el trabajo presenta algunas limitaciones que conviene abordar de formga explícita. La cobertura del *benchmark* es amplia y exigente, pero no captura toda la variedad de preguntas que se podrían hacer en un entorno empresarial. Además, persisten errores de completitud en preguntas especialmente demandantes, y se mantiene una sensibilidad significativa al tipo de pregunta, a la configuración de *chunking* y al diseño de *prompt*. Todo ello, junto con el sobrecoste de

las estrategias más avanzadas, confirma que la mejora de calidad debe gestionarse siempre con criterios de coste-beneficio para cada contexto de negocio. Estas limitaciones no invalidan la aportación del trabajo; al contrario, delimitan con precisión su alcance y orientan los próximos pasos a seguir.

Comenzando por la estrategia de grafos de conocimiento, una continuación natural es ampliar la cantidad de documentos descargados del Diario Oficial de la UE para incrementar la conectividad del grafo y medir de forma más realista la mejora en preguntas de referencias cruzadas. En el proyecto se usaron documentos de tres días distintos por limitaciones de coste computacional, pero se considera que con al menos todos los documentos de un mes se puede empezar a apreciar la mejora real del sistema. Junto a esta mejora, es relevante explorar otras técnicas de representación del conocimiento. En esta dirección, el trabajo reciente Corpus2Skill[29], heredero conceptual de la idea *Scatter/Gather*, propone compilar el conjunto inicial de documentos en una jerarquía navegable de "*skills*" y sustituir la recuperación puntual por la navegación agéntica guiada. Su aportación central es dar al modelo una vista estructural del espacio documental con exploración, retroceso y salto entre ramas; en lugar de limitarlo a resultados top-k de recuperación. La evidencia presentada en el *paper* indica mejoras de calidad y de fundamentación en la evidencia en documentos de un mismo dominio con taxonomía recuperable, pero también muestra que no es un reemplazo universal: en dominios abiertos o en colecciones de tablas muy parecidas, la recuperación plana puede seguir siendo preferible. Por tanto, una extensión razonable del proyecto sería comparar GraphRAG y Corpus2Skill bajo el mismo protocolo experimental, para decidir de forma empírica qué representación conviene por tipo de corpus y consulta.

Respecto al procesamiento en paralelo, resulta prioritario un despliegue preproductivo con usuarios reales en un entorno controlado. Esto permitiría construir una batería de preguntas más representativa e incorporar un circuito de *feedback* continuo, registrando por conversación si la respuesta se considera válida y con la posibilidad de que el usuario añada justificación textual. Con ello, sería posible mejorar el sistema a partir del uso real, del diagnóstico de fallos y de la recalibración continua. Por otra parte, esta estrategia debe evolucionar desde una activación fija hacia una activación adaptativa por tipo de consulta.

En la práctica, esto implica incorporar un selector de estrategia (RAG clásico o procesamiento paralelo) que clasifique la pregunta y reserve el proceso más costoso para casos con alto riesgo de incompletitud como enumeraciones, operaciones lógicas o búsquedas multicriterio. Sobre esta base, las mejoras técnicas más prometedoras son: optimizar el número de llamadas a LLMs y la agregación final para reducir redundancia, estudiando cómo influye el número de tokens de salida en la completitud de la respuesta; e incorporar un verificador de completitud previo a la respuesta final que detecte omisiones y lance una recuperación dirigida para cubrir los huecos de información detectados. Por su parte, la optimización coste-latencia puede reforzarse con inferencia en cascada y caché semántica: uso de modelos ligeros en etapas intermedias y escalado selectivo a modelos más costosos solo cuando exista alta incertidumbre, junto con reutilización de resultados parciales en consultas recurrentes. Esta línea ya se ha iniciado con el historial de llamadas, pero todavía hay margen para mejorar tiempos y coste por consulta en despliegue continuo.

Finalmente, en la dimensión de evaluación conviene ampliar el marco actual con métodos complementarios como RAGAs, un *framework* de evaluación automatizada que, en muchos casos, permite evaluar el sistema sin necesidad de contar con respuestas correctas definidas previamente[26]. RAGAs separa de forma explícita la calidad de recuperación y la de generación mediante métricas como fidelidad, precisión del contexto, cobertura del contexto y relevancia de la respuesta, permitiendo ciclos de evaluación más rápidos sin depender siempre de anotación manual exhaustiva. Una implementación directa en este proyecto consistiría en: registrar para cada ejecución la pregunta del usuario, contexto recuperado y respuesta generada; ejecutar un evaluador RAGAs desacoplado del proceso principal; y agregar resultados por estrategia, idioma y categoría de pregunta para obtener alertas tempranas de degradación y apoyar decisiones de enrutado adaptativo basadas en evidencia[26].

6. BIBLIOGRAFÍA

[1] Yan, S.-Q.; Gu, J.-C.; Zhu, Y.; Ling, Z.-H. (2024). Corrective Retrieval Augmented Generation. arXiv:2401.15884. https://arxiv.org/abs/2401.15884

[2] Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, H. "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv preprint arXiv:2312.10997, 2023. https://arxiv.org/abs/2312.10997

[3] Zhang, Y.; Li, M.; Long, D.; Zhang, X.; Lin, H.; Yang, B.; Xie, P.; Yang, A.; Liu, D.; y otros (2025). Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. arXiv:2506.05176. https://arxiv.org/abs/2506.05176

[4] Shukla, S., Thota, S., Raman, K., y otros (2017). Azure Cosmos DB: A Globally Distributed, Multi-Model Database Service. SIGMOD 2017. https://petri.com/microsoft-azure-cosmos-db-globally-distributed-multi-model-database-service/

[5] Raudaschl, A. (2023). Forget RAG, the Future is RAG-Fusion. arXiv:2402.03367. https://arxiv.org/abs/2402.03367

[6] Microsoft. "Azure Blob Storage documentation," 2026. https://learn.microsoft.com/azure/storage/blobs/

[7] Microsoft. "Azure Cosmos DB documentation," 2026. https://learn.microsoft.com/azure/cosmos-db/

[8] Dhakal, A.; Neupane, S. S.; Chaulagain, N. "A Comparative Study of Information Retrieval Models: BM25 versus Hybrid Retrieval on the Cranfield Collection," 2026. https://n9.cl/2k4d6

[9] Microsoft. "Azure Architecture Center documentation," 2026. https://learn.microsoft.com/azure/architecture/

[10] Microsoft. "Azure AI Search documentation," 2026. https://learn.microsoft.com/azure/search/

[11] Sonkamble, S. (2025). The Role of Microsoft Azure in Modern Cloud Computing: Scalability, Security, and AI Integration (PDF). https://ijirt.org/publishedpaper/IJIRT180561_PAPER.pdf

[12] Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandrahas, A., Guo, W., Ren, H., Huang, A., y otros (2024). Searching for Best Practices in Retrieval-Augmented Generation. arXiv:2407.01219. https://arxiv.org/abs/2501.12948

[13] Oche, O. O., Ayanponle, L. O., y Oche, V. O. (2025). A Systematic Review of Key Retrieval-Augmented Generation Systems: Progress, Gaps, and Future Directions. https://arxiv.org/html/2507.18910v1

[14] Singh, A.; Ehtesham, A.; Kumar, S.; Khoei, T. T.; Vasilakos, A. V. (2025). Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG. arXiv:2501.09136. https://arxiv.org/abs/2501.09136

[15] Edge, D.; Trinh, H.; Cheng, N.; Bradley, J.; Chao, A.; Mody, A.; Truitt, S.; Metropolitansky, D.; y otros (2024). From Local to Global: A Graph RAG Approach to Query-Focused Summarization. arXiv:2404.16130. https://arxiv.org/abs/2404.16130

[16] Nguyen, C. D. M.; French, T.; Stewart, M.; Hodkiewicz, M.; Liu, W. (2025). Representation Learning in Complex Logical Query Answering on Knowledge Graphs: A Survey. ACM Computing Surveys, 58(5). https://dl.acm.org/doi/full/10.1145/3771692

[17] Pan, Z., y colaboradores (2024). Knowledge Graph Enhanced Retrieval-Augmented Generation: A Survey. https://arxiv.org/abs/2408.08921

[18] Nickel, M., Murphy, K., Tresp, V., y Gabrilovich, E. (2016). A Review of Relational Machine Learning for Knowledge Graphs. Proceedings of the IEEE, 104(1). https://arxiv.org/abs/1503.00759

[19] Zep AI. "Graphiti documentation and repository," 2026. https://github.com/getzep/graphiti

[20] Microsoft. "Azure AI Foundry documentation," 2026. https://learn.microsoft.com/azure/ai-foundry/

[21] Friel, R.; Belyi, M.; Sanyal, A. (2024). RAGBench: Explainable Benchmark for Retrieval-Augmented Generation Systems. arXiv:2407.11005. https://arxiv.org/abs/2407.11005

[22] Asai, T. et al. "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," arXiv preprint arXiv:2310.11511, 2023. https://arxiv.org/abs/2310.11511

[23] Mialon, N. F. N. N. et al. "Augmented Language Models: A Survey," arXiv preprint arXiv:2302.07842, 2023. https://arxiv.org/abs/2302.07842

[24] Neo4j. "Neo4j Graph Database documentation," 2026. https://neo4j.com/docs/

[25] Zhang, H.; Liu, J.; Zhu, Z.; Zeng, S.; Sheng, M.; Yang, T.; Dai, G.; Wang, Y. "Efficient and Effective Retrieval of Dense-Sparse Hybrid Vectors using Graph-based Approximate

Nearest Neighbor Search," arXiv preprint arXiv:2410.20381, 2024. https://arxiv.org/abs/2410.20381

[26] RAGAs: Automated Evaluation of Retrieval Augmented Generation - ACL Anthology. https://aclanthology.org/2024.eacl-demo.16/

[27] Publications Office of the European Union. "Official Journal of the European Union: direct access," 2026. https://eur-lex.europa.eu/oj/direct-access.html

[28] Liu, N. F.; Lin, K.; Hewitt, J.; Paranjape, A.; Bevilacqua, M.; Petroni, F.; Liang, P. (2023). Lost in the Middle: How Language Models Use Long Contexts. arXiv:2307.03172. https://arxiv.org/abs/2307.03172

[29] Sun, Y.; Wei, P.; Hsieh, L. B. (2026). Don't Retrieve, Navigate: Distilling Enterprise Knowledge into Navigable Agent Skills for QA and RAG. arXiv:2604.14572v3. https://arxiv.org/html/2604.14572v3

[30] Refactoring.Guru. "Template Method," 2026. https://refactoring.guru/es/design-patterns/template-method

[31] Lewis, P. et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems, vol. 33, pp. 9459-9474, 2020. https://arxiv.org/abs/2005.11401

[32] Cormack, G. V.; Clarke, C. L. A.; Buettcher, S. "Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods," in Proceedings of the 32nd International ACM SIGIR Conference, 2009, pp. 758-759. https://dl.acm.org/doi/10.1145/1571941.1572114

[33] OpenAI. "La guía del modelo text-embedding-ada-002," 2026. https://zilliz.com/es/ai-models/text-embedding-ada-002

[34] Ke, W., Zheng, Y., Li, Y., Xu, H., Nie, D., Wang, P., and He, Y., "Large Language Models in Document Intelligence: A Comprehensive Survey, Recent Advances, Challenges, and Future Trends," ACM Transactions on Information Systems, vol. 44, no. 1, Art. 18, pp. 1-64, 2026. https://dl.acm.org/doi/full/10.1145/3768156

[35] IBM, "What is a dataset?", IBM Think, 2026. https://www.ibm.com/think/topics/dataset

[36] OpenAI, "What are tokens and how to count them?", OpenAI Help Center, 2026. https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them

ANEXO I

A.1. IMPLEMENTACIÓN RRF

Este apartado muestra el fragmento de implementación de *Reciprocal Rank Fusion* (RRF):

```
# RRF scoring
chunk_scores: dict[str, dict] = {}
for rank, chunk in enumerate(all_chunks, 1):
    cid      = chunk.get("chunk_id") or chunk.get("id") or str(rank)
    rrf_score = 1 / (60 + rank)
    if cid in chunk_scores:
        chunk_scores[cid]["rrf_score"] += rrf_score
    else:
        chunk["rrf_score"] = rrf_score
        chunk_scores[cid]  = chunk
ranked   = sorted(chunk_scores.values(), key=lambda x: x.get("rrf_score", 0), reverse=True)
filtered = [c for c in ranked if c.get("reranker_score", 0) >= min_score]
return filtered[:max_chunks]
```

A.2. ESTRUCTURA DE LOS SCHEMAS Y CHUNKS

Las tablas siguientes resumen los campos más relevantes de los documentos de entrada y de los *chunks* generados en la ingesta, con el objetivo de facilitar trazabilidad y mantenimiento del pipeline.

Campo	**Descripción**
nombre_apellidos	Nombre completo
puesto	Puesto actual
experiencia[]	Experiencia laboral
estudios[]	Formación académica

hard_skills[]	Habilidades técnicas
soft_skills[]	Habilidades blandas
otros[]	Otros datos

Tabla 4. Estructura principal de datos para CVs

Campo	**Descripción**
title	Título del artículo
language	Idioma
pageid	Identificador de página
categories[]	Categorías
url	URL
text	Contenido textual

Tabla 5. Estructura principal de datos para Wikipedia

Campo	**Descripción**
id	Identificador
chunkId	ID del chunk
docTitle	Título del documento
sourcePath	Ruta
sourceCollection	Colección

sourceLanguage	Idioma
content	Contenido
embedding	Vector embedding
isDeleted	Indicador de borrado
nChunk	Número de chunk

Tabla 6. Campos comunes de chunk para todos los casos de uso

Campo	**Descripción**
chunk_type	Tipo (experience/education/skills)
metadata.nombre_apellidos	Nombre
metadata.puesto	Puesto

Tabla 7. Campos específicos de chunk para CVs

Campo	**Descripción**
categories	Categorías
wiki_url	URL
pageid	ID página
Sections	Secciones

Tabla 8. Campos específicos de chunk para Wikipedia

Campo	**Descripción**
Sections	Secciones

doc_type	Tipo documento
metadatos normativos	Datos regulatorios

Tabla 9. Campos específicos de chunk para Legislación UE

A.3. EJEMPLOS INTERFAZ DE USUARIO

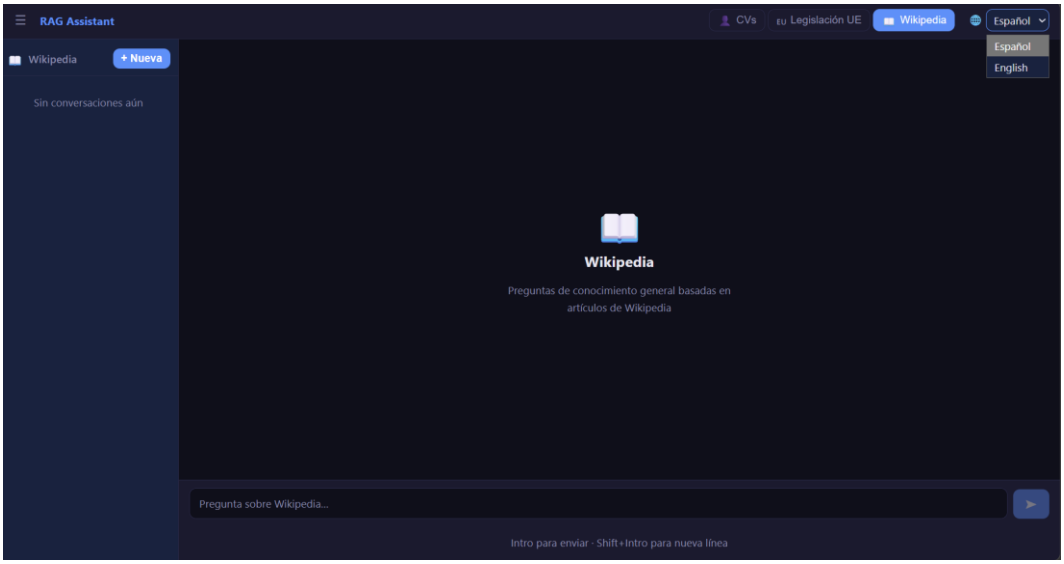

Figura 10. Interfaz del asistente RAG para el caso de uso Wikipedia, con el selector de idioma desplegado

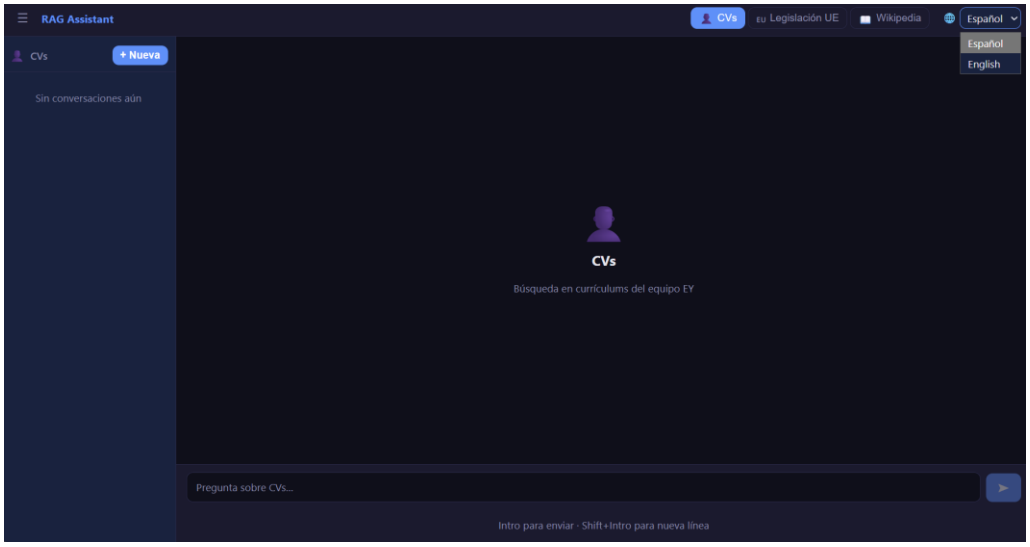

Figura 11. Interfaz del asistente RAG para el caso de uso CVs, con el selector de idioma desplegado.

A.4. GRAFO DE CONOCIMIENTO DE WIKIPEDIA

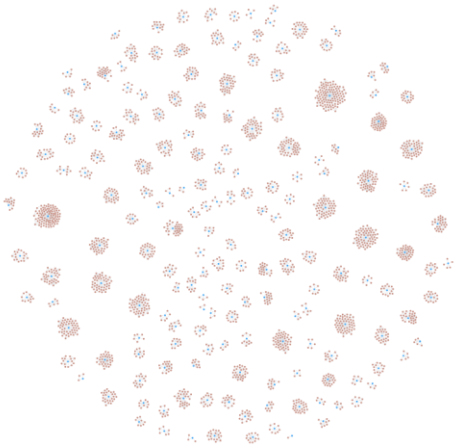

Figura 12. Grafo de conocimiento completo generado para el caso de uso Wikipedia, mostrando nodos de entidades y aristas de relaciones en Neo4j

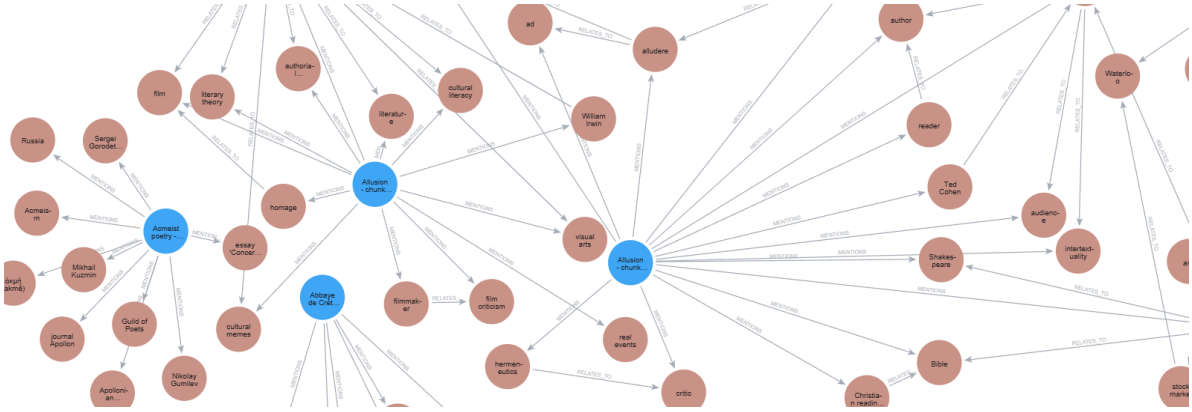

Figura 13. Muestra del grafo de conocimiento generado para el caso de uso Wikipedia, mostrando nodos de entidades y aristas de relaciones en Neo4j