ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

INTEGRACIÓN DE MODELOS GENERATIVOS PARA LA RECUPERACIÓN ACADÉMICA

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO DE CIENCIAS DE LA COMPUTACION

ALEJANDRO SEBASTIAN CHAVEZ VEGA
chavezalejo85@gmail.com

Director: DRA. GABRIELA SUNTAXI
gabriela.suntaxi@epn.edu.ec

QUITO, JULIO 2025

DECLARACIÓN

Yo ALEJANDRO SEBASTIAN CHAVEZ VEGA, declaro bajo juramento que el trabajo aquí escrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual, correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normatividad institucional vigente.

Alejandro Sebastian Chavez Vega

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por ALEJANDRO SEBASTIAN CHAVEZ VEGA, bajo mi supervisión.

Dra. Gabriela Suntaxi
Director del Proyecto

AGRADECIMIENTOS

A todos.

DEDICATORIA

A Georg Ferdinand Ludwig Philipp Cantor,
pues nadie nos expulsará del paraíso que creó para nosotros.

Índice general

Resumen viii

Abstract ix

1. Introducción 1

1.1. Planteamiento del problema 1

1.2. Justificación 1

1.3. Justificación Metodológica 2

1.4. Objetivos 2

1.4.1. Objetivo general 2

1.4.2. Objetivos específicos 2

1.5. Alcance 2

1.6. Marco Teórico 3

1.7. Revisión de literatura 3

1.7.1. Propósito y objetivos de la revisión 4

1.7.2. Criterios de inclusión y exclusión 4

1.7.3. Identificación del estudio semilla y selección de revisiones relevantes 4

1.7.4. Valoración de las evidencias y extracción de la información . . 5

1.7.5. Síntesis y representación de resultados 5

2. Metodología 38

2.1. Revisión sistemática 38

2.2. Enfoque Design Science Research (DSR) 40

2.3. Diseño y desarrollo del artefacto 44
2.3.1. Ingesta 44
2.3.2. Preprocesamiento 45
2.3.3. Vectorizacion 52
2.3.4. Vector DB 53
2.3.5. Recuperación 54
2.3.6. Re-ranking 54
2.3.7. Construcción de Contexto 56
2.3.8. Generación (LLM) 57
2.3.9. Evaluación 58

3. Pruebas, Resultados, Conclusiones y Recomendaciones **59**
3.1. Demostración 59
3.2. Evaluación del desempeño 59
3.3. Resultados 59
3.4. Conclusiones 59
3.5. Recomendaciones 59

Bibliografía **60**

Resumen

En el presente trabajo...

Abstract

In this paper...

Capítulo 1

Introducción

1.1. Planteamiento del problema

Los buscadores modernos como Google y Bing integran técnicas tipo RAG para entregar respuestas rápidas y de calidad que resumen temas sin obligar a abrir cada artículo, apoyándose en fragmentos de texto con citas verificables y entendiendo mejor el contexto de las consultas. Centinela es una plataforma web para la comunidad académica ecuatoriana que permite buscar artículos y autores, explorar perfiles de expertos y conectar colaboradores a través de grupos y discusiones; sin embargo, su buscador basado en enfoques léxicos no captura adecuadamente la semántica en consultas largas, mezcla de idiomas, presencia o ausencia de tildes y variaciones de mayúsculas y minúsculas, lo que reduce el recall y desordena la relevancia en escenarios con sinónimos, parafraseo y entidades. Al incorporar RAG, Centinela se alineará con las tendencias actuales, comprenderá mejor el contexto que el usuario desea y ofrecerá resultados más precisos con resúmenes útiles y evidencia trazable.

1.2. Justificación

La mejora del buscador de Centinela se fundamenta en ofrecer una experiencia de usuario más fluida y confiable al entender mejor el contexto de consultas largas y producir respuestas trazables que disminuyen alucinaciones a diferencia de los enfoques tradicionales, que sufren desajustes de vocabulario (polisemia, sinonimia y brechas léxicas) afectando el recall y el ranking, especialmente en varios idiomas. Adicionalmente, Centinela puede escalar el volumen de documentos y concurrencia de usuarios mediante bases de datos vectoriales que almacenan embeddings y per-

miten búsqueda semántica en alta dimensión con baja latencia y buenas propiedades de costo-desempeño, un pilar técnico probado para aplicaciones RAG.

1.3. Justificación Metodológica

1.4. Objetivos

1.4.1. Objetivo general

Desarrollar e implementar un sistema RAG que mejore el desempeño del buscador de la plataforma Centinela, permitiendo recuperar información científica relevante y generar respuestas automáticas de valor para el usuario.

1.4.2. Objetivos específicos

- Realizar una revisión sistemática de la literatura sobre metodologías y/o frameworks para la implementación de RAG.
- Diseñar e implementar la arquitectura técnica del sistema RAG utilizando modelos de recuperación y generación de texto.
- Evaluar el sistema RAG desarrollado mediante métricas estándar.

1.5. Alcance

El alcance del componente de mejora del buscador abarca la investigación, selección, validación e implementación de un buscador avanzado orientado a optimizar la precisión y la relevancia de los resultados para investigadores ecuatorianos, garantizando además trazabilidad de la evidencia y mitigación de alucinaciones mediante un esquema RAG. Primero, se realizará una revisión del estado actual de Centinela para establecer la línea base, documentando las limitaciones de los enfoques léxicos (polisemia, sinonimia y brechas léxicas) y reforzando prácticas de normalización del texto para reducir ruido. Luego, se analizarán enfoques recientes de IR y RAG para definir criterios técnicos de adopción, integración de evidencia y flujo clásico de RAG (recuperación de fragmentos top-k y generación) con énfasis en faithfulness y uso de citas. Finalmente, se implementará e integrará el modelo seleccionado en Centinela

con indexación semántica en una base de datos vectorial (índices ANN y filtrado por metadatos) manteniendo latencia baja y trazabilidad de los fragmentos citados.

1.6. Marco Teórico

1.7. Revisión de literatura

En los últimos años, la evolución de los modelos de lenguaje de gran escala (Large Language Models, LLM) ha redefinido el procesamiento del lenguaje natural e impulsado nuevas líneas de investigación. Sin embargo, estos modelos dependen únicamente de los datos empleados durante su entrenamiento, lo que limita su capacidad para ofrecer respuestas actualizadas, verificables y contextualizadas. En respuesta a esta limitación surge el enfoque de Retrieval-Augmented Generation (RAG), el cual combina la recuperación de información con la generación de lenguaje natural, logrando mejorar la precisión, la coherencia y la actualidad de las respuestas producidas por los modelos.

Dada la creciente relevancia de los LLM, resulta necesario llevar a cabo una revisión exhaustiva de la literatura que permita consolidar los avances recientes y evaluar los desafíos aún presentes. Para ello, esta revisión se apoya en Umbrella SLR y propagacion de citaciones usando el protocolo propuesto en la Sección 2.1; En esta sección se presenta un análisis estructurado de la literatura disponible, considerando tanto los fundamentos conceptuales de RAG como sus fases de desarrollo, aplicaciones y el futuro. Para ello, el proceso de revisión se organiza en las fases que se presentan a continuación:

- Propósito y objetivos de la revisión
- Criterios de inclusión y exclusión
- Identificación del estudio semilla y selección de revisiones relevantes
- Valoración de las evidencias y extracción de la información
- Síntesis y representación de resultados

Las cuales buscan garantizar la consistencia, validez y pertinencia de la evidencia obtenida.

1.7.1. Propósito y objetivos de la revisión

El propósito de esta revisión es consolidar la información disponible sobre los RAG, abordando su estudio desde los fundamentos hasta las fases de desarrollo. Se inicia con su definición y arquitectura, para luego profundizar en las etapas clave del proceso: extracción del corpus, preprocesamiento, vectorización, recuperación de información, evaluación, almacenamiento en bases vectoriales y generación de resultados. Asimismo, se examinan los paradigmas, las métricas de evaluación y el futuro de RAG. Durante esta revisión se busca lograr el objetivo general de proporcionar un panorama global y actualizado sobre los RAG, exponiendo sus fundamentos, desarrollo y aplicación.

1.7.2. Criterios de inclusión y exclusión

Se incluyen únicamente revisiones sistemáticas y metaanálisis publicados entre 2018 y 2025, en inglés o español, dado que la producción científica en el área comenzó a incrementarse a partir de 2018, con base en información de Lens.org[1], este incremento coincide con la popularización de los modelos de lenguaje basados en transformers[2]. Los estudios deben provenir de fuentes confiables y ser, a su vez, revisados por un experto. Se da preferencia a aquellos que presenten una cobertura amplia de los temas más relevantes para el objeto de estudio.

Se excluyen las revisiones narrativas, los documentos que carezcan de transparencia en sus métodos de búsqueda o síntesis, así como las publicaciones que no estén directamente relacionadas con el objeto de estudio delimitado.

1.7.3. Identificación del estudio semilla y selección de revisiones relevantes

El proceso de búsqueda se inicia con la identificación de dos estudios semilla, extraídos de Google Scholar mediante los parámetros “Retrieval Information” y “Retrieval Augmented Generation”. Debido al análisis realizado en Lens.org, se estableció el filtro de 2018 a 2025, ya que se observa que a partir de 2018 el término retrieval-augmented generation comenzó a adquirir una relevancia en la literatura científica,

[1]Es una plataforma abierta para la búsqueda, análisis y visualización de literatura científica y patentes. Accesible en: Lens.org

[2]Se atribuye a hitos como BERT (2018), GPT-2 (2019) y T5 (2020), que impulsaron un avance en la investigación del procesamiento del Lenguaje Natural

mostrando interés de la comunidad investigadora hasta la actualidad.

El primer estudio seleccionado fue Information Retrieval: Recent Advances and Beyond Hambarde y Hugo [8], publicado en IEEE Access. Este trabajo constituye una revisión exhaustiva de la recuperación de información, abarcando desde los métodos tradicionales hasta los enfoques basados en deep learning y transformers, por lo que resulta un punto de partida principal para explorar la literatura reciente y relevante.

El segundo estudio semilla corresponde al artículo Retrieval-Augmented Generation for Large Language Models Gao et al. [7], publicado en arXiv, el cual presenta un marco conceptual y aplicado sobre la integración de recuperación de información y modelos generativos de gran escala. Su incorporación permite establecer una base teórica para contextualizar el análisis de las revisiones seleccionadas.

A partir de estos dos estudios semilla, y aplicando los criterios de inclusión y exclusión previamente definidos, se identificaron 25 revisiones relevantes que cumplen con los criterios establecidos. Estas revisiones constituyen la base para el análisis y síntesis en el presente trabajo.

1.7.4. Valoración de las evidencias y extracción de la información

De los estudios seleccionados se procede a realizar un análisis, con el fin de excluir aquellos artículos que no cumplen con los criterios establecidos o que presentan un nivel de profundidad insuficiente para los objetivos de la revisión. La selección final de los estudios se realiza en consenso con expertos en el área, garantizando así la pertinencia y relevancia de la evidencia incluida. Para la organización, codificación y síntesis de la información se utiliza ATLAS.ti[3], que facilitará la estructuración de los hallazgos.

1.7.5. Síntesis y representación de resultados

Con la literatura seleccionada se identificó la hoja de ruta que se presenta a continuación en la Figura 1.1.

[3]Scientific Software Development GmbH. Disponible en: Atlas.ti

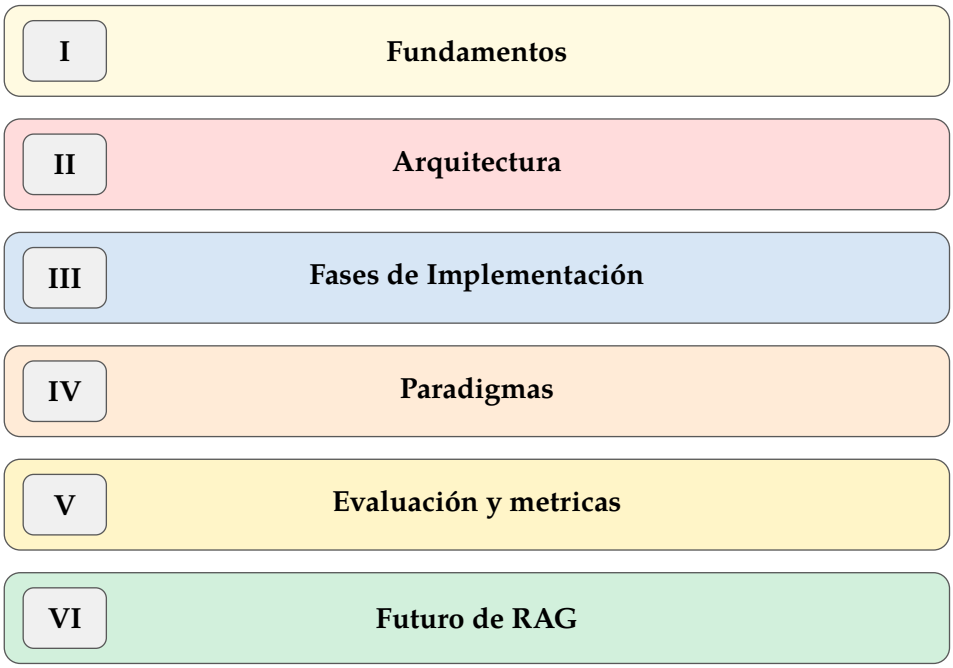

Figura 1.1: Resumen esquemático de RAG

A partir de esta hoja de ruta se desarrolla un esquema más detallado, en el que primero se exploran la teoría, las características y las aplicaciones, como se muestra en la Figura. 1.2. Posteriormente, se profundiza en la arquitectura, Figura. 1.3, donde se describe cada uno de los componentes que la conforman (Retriever, Augmented y Generation), así como las variantes y mejoras que existen en cada uno. Más adelante, se detalla el proceso de implementación (ver la Figura. 1.4), desde la preparación de los datos hasta el componente de generación, incluyendo las técnicas y herramientas más relevantes. En la Figura. 1.5 correspondiente se examinan los paradigmas de RAG, presentando sus tipos y clases; seguidamente, se introducen las métricas y evaluadores automáticos empleados en la evaluación de sistemas RAG, junto con las consideraciones éticas y de equidad que deben tenerse en cuenta. Finalmente, se discuten las tendencias emergentes, los desafíos actuales y las posibles direcciones futuras que podrían orientar la evolución de los sistemas RAG.

Fundamentos de RAG

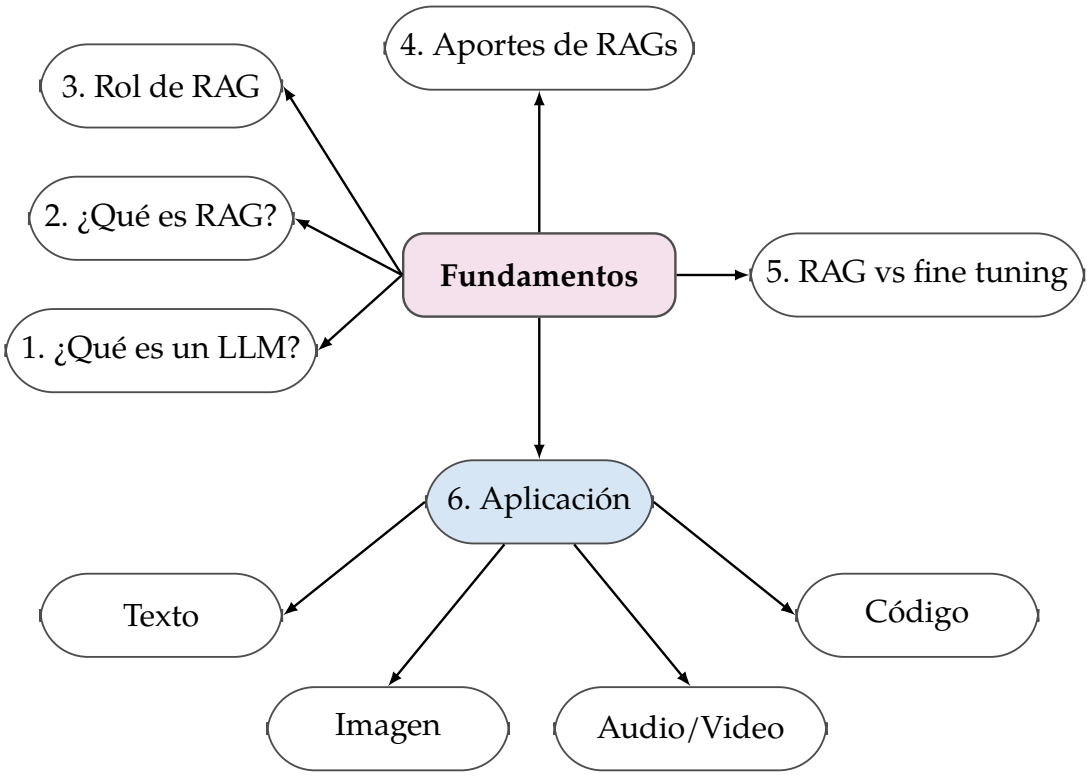

Figura 1.2: Fundamentos de RAG

En esta subsección se presentan los fundamentos teóricos de Retrieval-Augmented Generation (RAG), comenzando con la definición de los modelos de lenguaje de gran escala (LLMs) y su relación. Se expone también el papel que desempeña RAG, los principales aportes que ha generado en distintos ámbitos y su diferenciación frente al *fine-tuning*. Finalmente, se introduce su aplicación práctica, lo que permite comprender la importancia y el impacto que RAG tiene en la actualidad.

¿Qué es un LLM? Son modelos de inteligencia artificial (IA) basados en la arquitectura *transformer*[4], entrenados con grandes volúmenes de datos textuales con el objetivo de aprender representaciones contextuales del lenguaje. Según Casola, Lauriola y Lavelli [4], estos modelos utilizan técnicas de preentrenamiento no supervisado para captar patrones lingüísticos y semánticos, lo que permite que posteriormente puedan ajustarse a tareas específicas como clasificación de texto, análisis de senti-

[4]es una arquitectura para modelar secuencias que reemplaza la recurrencia y la convolución por auto-atención multi-cabeza y capas feed-forward por posición, con conexiones residuales, normalización y codificaciones posicionales; puede usarse como codificador, decodificador o codificador-decodificador

mientos, traducción automática, reconocimiento de entidades o respuesta a preguntas. Ejemplos destacados son *BERT, RoBERTa, ALBERT, XLNet, DistilBERT y GPT-3*, que han mostrado rendimientos sobresalientes en diversas aplicaciones de procesamiento de lenguaje natural (NLP).

De acuerdo con Ramdurai [21], los LLMs también se definen como una clase de modelos de IA capaces de procesar y generar texto de forma similar al lenguaje humano, gracias al uso de redes neuronales profundas y la capacidad de aprender no solo gramática y relaciones entre palabras, sino también aspectos más complejos como humor, tono emocional y contexto. Entrenados en enormes corpus de datos provenientes de libros, artículos y sitios web, estos modelos pueden responder preguntas, redactar ensayos, traducir, resumir y crear contenido de manera autónoma. Ejemplos recientes incluyen *GPT-4, T5, XLNet y PaLM*, los cuales demuestran su versatilidad en tareas avanzadas de NLP y en sistemas aplicados en diferentes industrias.

¿Qué es un RAG? Según Han, Susnjak y Mathrani [9], Retrieval-Augmented Generation (RAG) es una técnica que integra la capacidad generativa de los modelos de lenguaje con la precisión de la recuperación de información en tiempo real. En lugar de basarse únicamente en el conocimiento almacenado en los parámetros durante el entrenamiento, RAG permite consultar repositorios externos como bases de datos o motores de búsqueda para obtener documentos relevantes y actualizados. Estos se incorporan al prompt del usuario, lo que fundamenta la respuesta en fuentes verificables y disminuye los problemas de errores y alucinaciones que suelen presentarse en los modelos de lenguaje de gran escala.

Rol del RAG El rol de RAG es abordar los 3 límites existentes de los LLM: alucinaciones, desactualización del conocimiento y falta de trazabilidad. Como menciona Zhai [27] no es realista esperar que los LLM sustituyan al buscador; más bien, los futuros sistemas integrarán LLM + búsqueda/RAG, de modo que el modelo "aprenda a usar" la recuperación como herramienta fiable.

Aportes de RAG El aporte de los modelos RAG no se limita a añadir texto al prompt, también consiste en seleccionar evidencia relevante, integrarla mediante filtros adecuados y exponer las fuentes utilizadas para que el usuario pueda verificar la confiabilidad de la información. De acuerdo con la revisión exhaustiva de la literatura los RAGs aportan:

1. **Mejora de la exactitud**: en tareas intensivas en conocimiento, al fundamentar las respuestas en documentos recuperados y relevantes [7, 29].

2. **Mitigación de alucinaciones** al condicionar la salida en evidencias recuperadas, exigir consistencia con estas y habilitar la verificación/citación de fuentes por parte del usuario [28, 5].

3. **Reducción de costos y actualización ágil del conocimiento**: en lugar de reentrenar o usar fine-tuning el LLM, basta con actualizar el corpus o el índice, lo que permite incorporar información reciente de manera eficiente [7, 27].

4. **Cobertura y ampliación de conocimiento especializado**: los RAG permiten abordar información de long-tail[5] que no suele estar bien representada en los datos de entrenamiento de un LLM, al tiempo que extienden su utilidad a campos especializados como derecho, medicina, finanzas y automatización de revisiones sistemáticas de literatura (SLR), apoyando procesos de búsqueda, filtrado, extracción y síntesis de evidencia [7, 10, 27].

5. **Manejo de documentos largos**: técnicas como RAPTOR organizan documentos extensos en árboles de resúmenes que permiten recuperar a distintos niveles de abstracción, mejorando el rendimiento en QA (Question Answering)[6] complejo y multi-hop [7] [22].

6. **Aportes en evaluación**: se han desarrollado métricas específicas para auditar la calidad de un pipeline RAG (p. ej., context relevance, faithfulness, answer correctness, citation quality), hoy ya sistematizadas en la literatura [15].

RAG vs Fine Tuning La diferencia entre ambos es en el tipo de problema que resuelven y en la manera en que se maneja la información. De acuerdo con Gao et al. [7], RAG añade conocimiento no paramétrico, lo que lo hace especialmente útil para datos cambiantes, dominios con documentación extensa y situaciones donde se requiere trazabilidad de las fuentes. Por otro lado, fine-tuning introduce conocimiento paramétrico dentro de los pesos del modelo durante un proceso adicional de entrenamiento. Así, el modelo está adaptado a generar salidas con el estilo, formato o razonamiento esperado, incluso sin consultar fuentes externas. Esto es muy útil para

[5] Información poco frecuente, rara o muy especializada que casi no aparece en los datos de entrenamiento de un modelo.

[6] Es la tarea de responder preguntas a partir de evidencia extraída de uno o varios textos.

[7] La respuesta a una pregunta no se puede obtener de una única fuente, sino que requiere combinar la evidencia de varias fuentes distintas.

casos donde se requiere que el modelo siga protocolos internos, produzca salidas con plantillas específicas o razone con un estilo particular.

Aplicación RAG puede usarse en 4 modalidades:

- **Texto**: RAG sustenta tareas de QA, asistentes de búsqueda, resúmenes y revisiones sistemáticas de literatura (SLR). Para documentos largos se apoya en estructuras jerárquicas que organizan el corpus en un árbol construido de abajo hacia arriba donde se agrupan trozos de texto por similitud, se resumen para formar nodos padre y se crean niveles de abstracción (fragmentos específicos hasta resúmenes globales), los cuales recupera el contexto más pertinente para responder preguntas con evidencia dispersa [22]. Además, se complementa con técnicas de *text-matching* y *re-ranking* para elevar la pertinencia del contexto recuperado [12].
- **Código**: conecta el modelo con documentación, repositorios y bases de conocimiento para explicar fragmentos, ubicar APIs y resolver errores, integrando recuperadores y bases vectoriales en el flujo del desarrollador [10].
- **Imagen**: mediante el uso de embeddings multimodales y bases vectoriales, permite tareas de descripción asistida, búsqueda cruzada y grounding de respuestas, vinculando consultas textuales con representaciones visuales [13, 17].
- **Audio y Video**: generalmente se parte de la transcripción, que luego se indexa en segmentos con marcas de tiempo. De esta manera, RAG posibilita responder, resumir o enlazar afirmaciones con evidencias auditables en el propio material [13, 29].

Arquitectura RAG

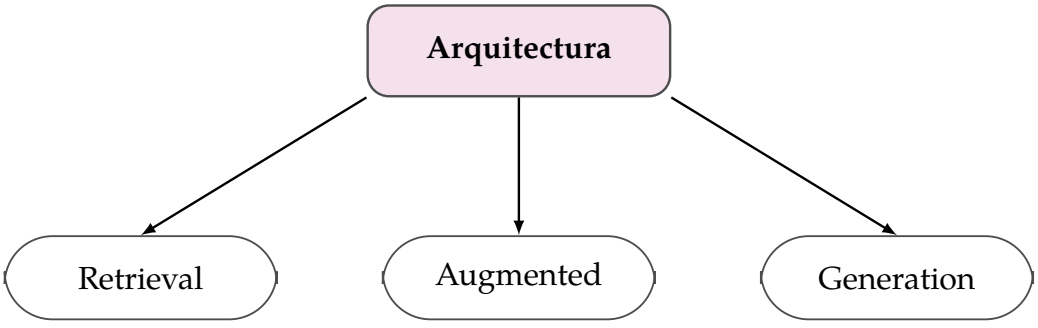

Figura 1.3: Componentes de RAG

RAG se compone de tres fases: recuperación, augmentation y generación (ver Figura 1.3). Como lo menciona Gao et al. [7], primero se localizan documentos relevantes para la consulta (Recuperación); luego, se enriquece la entrada del usuario con esos textos (augmentation); y finalmente, el modelo produce una respuesta basada tanto en su conocimiento interno como en la información recuperada (Generación). Gracias a este enfoque, RAG incrementa la exactitud de las respuestas, facilita la actualización del conocimiento sin necesidad de reentrenar el modelo y mejora la transparencia al permitir la cita de fuentes. Se trata de una de las técnicas más relevantes en tareas que requieren una gran cantidad de conocimiento, como en los ámbitos médico, legal o de investigación científica.

La Tabla 1.2 resume el componente retriever en tres categorías: (i) **indexación**, que incluye bases vectoriales y algoritmos de búsqueda ; (ii) **enhancements**, conjunto de técnicas que mejoran la calidad del contexto recuperado y su ordenación; y (iii) **tipos de retrievers**, que abarcan enfoques sparse, dense e híbridos.

Categoría	**Subcategoría / Tipo**	**Técnicas**
Indexing	Vector DB	FAISS, Annoy, Milvus, Weaviate, Pinecone, Qdrant
	Algoritmos de búsqueda	Approximate Nearest Neighbors, Locality-Sensitive Hashing
Enhancements	Reranking	CEDR, DuoBERT, DRMM
	Retriever Finetuning	REPLUG, Learning to retrieve ICL, UDAPDR
	Hybrid Search	RAP-Gen, BlendedRAG, ReACC
	Chunk Optimization	RAPTOR, LlamaIndex
	Recursive Retrieval	ReAct, RATP
	RAG Pipeline Enhancement	*Rule-based*, *Model-based*
	Query Reformulation	HyDE, Multi-query, Query2Doc
Tipos de retrievers	Sparse	BM25, TF-IDF
	Dense	Embeddings (ej. BERT, OpenAI, etc.)
	Híbridos/otros	Modelos híbridos u otros

Tabla 1.2: Componente Retriever

Indexing: Es el encargado de organizar y representar la información de forma eficiente. En este sentido, los vector databases (VDBs) han aumentado su popularidad, ya que permiten almacenar vectores de alta dimensionalidad y realizar búsquedas por similitud semántica. Como explica Joshi [14], estas bases de datos resultan esenciales para la aplicación de inteligencia artificial generativa, ya que superan las limitaciones de las bases relacionales en el manejo de datos no estructurados. Los VDBs integran mecanismos de búsqueda aproximada de vecinos más cercanos (ANNS), lo que posibilita consultas rápidas incluso sobre miles de objetos. Además, como lo menciona Ma et al. [17], incorporan técnicas de optimización como particionamiento, sharding [8], cachés y replicación para garantizar escalabilidad y baja latencia en entornos distribuidos.

Locality-Sensitive Hashing (LSH) es una técnica de indexación ampliamente utilizada en VDBs para acelerar la búsqueda de vecinos aproximados en espacios de alta dimensionalidad. A diferencia de los esquemas de hashing tradicionales, cuyo objetivo es dispersar uniformemente los datos para minimizar colisiones, LSH está diseñado para maximizar la probabilidad de que vectores similares se asignen al mismo bucket de hash [17]. Según esta técnica, la preservación de la localidad se consigue mediante funciones de hash que reflejan la similitud entre vectores, generando colisiones más frecuentes dentro del espacio reducido. De esta manera, al realizar una consulta, el sistema solo necesita comparar el vector de entrada con aquellos almacenados en el mismo o en buckets cercanos, reduciendo drásticamente la complejidad computacional de la búsqueda [17]. Estas características explican por qué los VDBs se han consolidado como una infraestructura fundamental en el soporte de sistemas RAG.

Enhancements: Son técnicas complementarias que se añaden antes, durante o después de la recuperación para subir la calidad sin reemplazar al recuperador base.

Técnicas	**Descripción**
Reranking	
CEDR (Contextualized Embeddings for Document Ranking)	Re-ranker basado en BERT que concatena q y el fragmento d para obtener representaciones contextualizadas y estimar directamente la relevancia del par.

[8] Es una técnica de partición horizontal que divide una base de datos en fragmentos distribuidos entre múltiples nodos, lo que permite escalar de forma eficiente y balancear la carga en bases de datos vectoriales

DuoBERT	Re-ranker *pairwise* que compara explícitamente la *query*, el candidato positivo y el negativo $(q,\ d^+,\ d^-)$ en una sola secuencia para aprender preferencias y producir el ordenamiento final; muy usado en *passage ranking*.
DRMM (Deep Relevance Matching Model)	Modelo clásico de interacción que construye histogramas de similitud (consulta-documento) por término y usa *term gating* (softmax) para ponderar la contribución de cada término al puntaje final.
Retriever Finetuning	
REPLUG	Ajuste que trata al LLM como caja negra y entrena el recuperador para que su distribución de documentos coincida con la inducida por el modelo generador, reduciendo divergencia y mejorando *recall@k*.
Learning to Retrieve ICL	Método que entrena un recuperador ligero capaz de seleccionar ejemplos útiles para *in-context learning*, afinando qué documentos se muestran al LLM.
UDAPDR	Estrategia de entrenamiento con pseudo-relevancia y expansiones automáticas de consultas, reduciendo dependencia de grandes conjuntos anotados.
Hybrid Search	
RAP-Gen	Combina recuperación dispersa (BM25) y densa (embeddings neuronales) en pipelines generativos, equilibrando eficiencia y cobertura semántica.
BlendedRAG	Integra resultados de varios recuperadores (sparse y dense) y los pondera para mitigar sesgos de un único método.
ReACC	Búsqueda híbrida adaptativa que alterna y ajusta dinámicamente el peso entre consultas dispersas y densas según la necesidad de la tarea.
Chunk Optimization	
RAPTOR	Técnica jerárquica–recursiva que organiza documentos largos en niveles de representación (chunks y resúmenes) para optimizar recuperación y ranking.

LlamaIndex	Framework flexible que segmenta y organiza documentos en chunks ajustables, permitiendo índices eficientes y contextos más ricos para RAG.
Recursive Retrieval	
ReAct	Algoritmo que alterna pasos de razonamiento (*thoughts*) y acciones de búsqueda (*actions*), refinando la recuperación de forma iterativa.
RATP	Método recursivo que planifica múltiples rondas de recuperación, integrando razonamiento y búsqueda para preguntas complejas de varios saltos.
RAG Pipeline Enhancement	
Rule-based	Mejoras basadas en reglas predefinidas: filtrado por metadatos, deduplicación, control de fechas o tipos de documentos.
Model-based	Módulos aprendidos que deciden qué recuperar, cómo fusionar y qué citar, optimizando el pipeline extremo a extremo mediante aprendizaje.
Query Reformulation	
HyDE (Hypothetical Document Embeddings)	Genera un documento hipotético a partir de la consulta y lo proyecta en el espacio de embeddings, mejorando la recuperación semántica.
Multi-query	Expande la consulta en varias variantes equivalentes, aumentando la cobertura del recuperador frente a sinonimia y polisemia.
Query2Doc	Convierte la consulta en un pseudo-documento que actúa como proxy para recuperar pasajes más relevantes y completos.

Tabla 1.3: Técnicas de Enhancements

Definiciones clave:

- **Cross-encoder.** Arquitectura de re-ranking que concatena la consulta q y el pasaje d en una sola secuencia para codificarlos conjuntamente y producir un puntaje directo de relevancia del par (q, d).

- **Pairwise.** Esquema de entrenamiento para re-rankers que recibe $(q,\ d^+,\ d^-)$ y optimiza para que el modelo puntúe d^+ por encima de d^-, generando el ordenamiento final.

- **Term gating.** Mecanismo de ponderación por término (softmax) que modula la contribución de cada término de la consulta al puntaje global de similitud; característico de modelos de interacción.

- **Agregación en *passage ranking*.** Cuando los documentos largos se segmentan en fragmentos $d = \{p_1, \ldots, p_n\}$, el sistema puntúa pares (q, p_i) y luego agrega esos puntajes a nivel documento. Dos heurísticas comunes son:

$$\text{score}(d) = \begin{cases} \text{máx}_i\, s(q, p_i) & (\textit{maxP}) \\ \sum_i s(q, p_i) & (\textit{sumP}) \end{cases}$$

  Además, existen modelos de *agregación aprendida* como PARADE, que primero codifican cada fragmento y luego aprenden a combinarlos (p. ej., con otra capa/transformer) para producir score(d), superando a estas heurísticas en diversos escenarios [6].

De acuerdo con la Tabla 1.3, resume un conjunto de técnicas de mejora del pipeline RAG: *Reranking* (segunda etapa de ordenamiento), *Retriever finetuning* (ajuste del recuperador), *Hybrid search* (combinación dispersa-densa), *Chunk optimization* (segmentación/jerarquización de documentos largos), *Recursive retrieval* (búsqueda iterativa con razonamiento), *RAG pipeline enhancement* (módulos basados en reglas y en modelos) y *Query reformulation* (reescritura/expansión de consultas). De acuerdo con la literatura [8, 6, 9, 25], estas categorías agrupan las técnicas más empleadas y reportadas en revisiones exhaustivas de la literatura. No obstante, existen otras variantes y propuestas que, aunque menos citadas, complementan este panorama.

Tipos de retrievers: Los retrievers se agrupan en tres familias: sparse, dense y híbridos/otros. Los métodos sparse representan consultas y documentos como vectores dispersos de términos ponderados, son eficientes, interpretables y dependen del matching léxico. Los dense usan representaciones continuas aprendidas (embeddings) para medir similitud semántica, capturando sinonimia y paráfrasis con mejor recall semántico, aunque con mayor costo de indexación y búsqueda. Los enfoques híbridos/otros combinan señales léxicas y semánticas fusionando resultados sparse y dense o usando interacciones tardías y re-rankers para equilibrar precisión, cobertura y eficiencia.

- **Sparse**
  - **BM25**: Modelo probabilístico que pondera frecuencia de términos, IDF y normaliza por longitud. A diferencia de otros enfoques dispersos, BM25 introduce una función de saturación que evita que la alta repetición de un término en un documento incremente indefinidamente su peso. Es la línea base más usada como primer filtro eficiente en RAG y RI a gran escala Hambarde y Hugo [8] y Han, Susnjak y Mathrani [9].
  - **TF-IDF**: Representación dispersa clásica que pondera términos por frecuencia local y rareza global, es decir, cada vez que un término aparece, su peso crece linealmente. Sin embargo, es ampliamente reportada como recuperador base en RAG Jiang y Cai [12] y Zhai [27].
- **Dense**
  - **DPR (Dense Passage Retrieval)**: Bi-*encoder* BERT (un codificador para la consulta y otro para fragmentos de texto) que compara embeddings por producto punto; superó a BM25 en open-domain QA[9] y es referente en dense retrieval Fan et al. [6] y Han, Susnjak y Mathrani [9].
  - **Contriever**: Retriever auto-supervisado con un solo BERT *encoder* para consultas y documentos; entrena con aprendizaje contrastivo y es robusto y adaptable sin grandes datasets etiquetados Hambarde y Hugo [8].
- **Híbridos / Otros**
  - **Hybrid Search**: Combinación de señales *sparse* y *dense* para equilibrar precisión léxica y cobertura semántica en el pipeline RAG Gao et al. [7].
  - **Motores de Internet (Bing/Google)**: Funcionan como recuperadores dinámicos para información actualizada sin mantener índice propio dependiendo de un servicio externo, la variabilidad en la calidad de resultados y el riesgo de inconsistencias en la recuperación. Hambarde y Hugo [8].

[9]Open-domain QA (open-domain question answering) es el planteamiento de preguntas-respuestas donde el sistema debe responder sobre cualquier tema apoyándose en un corpus grande y general

Categoría	Subcategoría / Tipo	Técnicas / Ejemplos
Tipos	Pre-training	Knowledge Graph Embeddings
	Fine-tuning	SFT (supervised fine-tuning), PEFT (parameter-efficient fine-tuning), ajuste del retriever/reranker
	Inference	Retrieve-then-Read-then-Revise (RARR), mitigación de Lost in the Middle
Data	Structured	Knowledge Graphs
	Unstructured	open corpus (Common Crawl, PubMed, etc.)
	LLM generated content	self-retrieval
Process	Once	document augmentation
	Iterative	pseudo-relevance feedback (PRF)
	Adaptative	SKR (Selective Knowledge Retrieval)

Tabla 1.5: Componente Augmentation

Augmentation es el proceso el cual un modelo de lenguaje incorpora información adicional ya sea externa, como documentos, bases de conocimiento o corpus abiertos, o procesada internamente en diferentes etapas de su funcionamiento. Los autores coinciden en que esta integración cumple objetivos clave: mejorar la precisión de las respuestas al aportar evidencia relevante, reducir las alucinaciones al contrastar el conocimiento implícito del modelo con fuentes verificables, y actualizar el conocimiento de los LLMs sin necesidad de reentrenarlos desde cero, ya que el acceso a información recuperada permite mantenerlos al día en dominios dinámicos como ciencia, medicina o derecho. La Tabla 1.5 contiene tres ejes del componente augmentation: Tipos, Data y Process. Los cuales enriquecen el contexto disponible y mejoran la pertinencia de la recuperación.

Tipos de augmentation: Zhao et al. [29] señalan que se puede aplicar en tres momentos distintos. En el pre-training, se integran representaciones estructuradas como knowledge graph embeddings que dotan al modelo de memoria explícita sobre entidades y relaciones. En la fase de fine-tuning, se ajustan los parámetros del modelo (y/o del recuperador) con datos del dominio para alinear formato y criterios de evaluación con la tarea. Para lograr este objetivo, existen estrategias como SFT el cual

consiste en volver a entrenar el modelo usando datos etiquetados de la tarea objetivo para alinear el formato y criterios de salida. Este ajuste del modelo es efectivo, sin embargo resulta costoso y sensible a los hiperparámetros [10]e incluso a la semilla inicial (números aleatorios del entorno de entrenamiento , por ejemplo, la inicialización de pesos y el orden de muestreo de los datos), lo que impacta la selección de modelo y la fiabilidad de la evaluación Casola, Lauriola y Lavelli [4]. Por otra parte, PEFT modifican muy pocos parámetros usando prompt tunning (soft prompts) donde se aprenden vectores continuos que se anteponen al input para guiar la generación, sin cambiar la base del modelo y prefix tunning es una variante de prompt tunning el cual agrega prefijos de claves o valores en las capas de atención del transformer y a su vez puede anteponer vectores al input. Asi, cada capa contiene el contexto que dirige la atención del modelo hacia patrones deseados, manteniendo intactos los pesos base del modelo. El ajuste del recuperador en el método REPLUG se realiza mediante la divergencia de Kullback–Leibler (KL), que compara la distribución de probabilidad de documentos estimada por el recuperador $P_R(d \mid x)$ con la distribución inducida por el modelo de lenguaje $Q_{\text{LM}}(d \mid x, y)$. La función objetivo es:

$$\text{KL}\big(P_R(d \mid x) \,\|\, Q_{\text{LM}}(d \mid x, y)\big) = \sum_{d \in \mathcal{C}(x)} P_R(d \mid x) \log \frac{P_R(d \mid x)}{Q_{\text{LM}}(d \mid x, y)}.$$

Minimizar esta divergencia significa entrenar el recuperador para que su distribución de documentos recuperados se acerque a la que el propio LLM, tratado como caja negra, considera más adecuada. De este modo, el recuperador aprende a imitar las preferencias del LLM respecto a qué documentos son más relevantes, logrando así seleccionar de manera más efectiva aquellos fragmentos que aportan información útil. Este alineamiento mejora el *recall@k* y asegura que el contexto suministrado al generador no solo sea pertinente, sino también consistente con la forma en que el LLM procesa y utiliza la evidencia externa Gao et al. [7].

UPRISE, por su parte, entrena un prompt retriever usando a un LLM congelado como supervisor: el LLM puntúa qué prompts (de un banco de plantillas) ayudan más a producir buenas respuestas para cada entrada; esos puntajes se convierten en señales de entrenamiento para que el retriever aprenda a seleccionar, en inferencia y sin reentrenar el LLM, los prompts más útiles en zero-shot[11], elevando la calidad del contexto que finalmente consume el generador Zhao et al. [29] y Fan et al. [5].

[10]Variables de configuración externas al modelo que controlan cómo aprende durante el entrenamiento (e.g., tasa de aprendizaje, batch size, warmup, regularización, scheduler y número de épocas), regulando el tamaño y el ritmo de las actualizaciones y la duración del entrenamiento.

[11]Resolver una tarea sin ejemplos etiquetados de esa tarea durante el ajuste

Finalmente, en la inferencia, se aplican métodos sin reentrenamiento: por ejemplo, RARR (Retrieve-then-Read-then-Revise) refina las respuestas con evidencia recuperada, mientras que la mitigación de Lost in the Middle reorganiza documentos recuperados para que el modelo aproveche mejor la ventana de contexto.

Datos: Fan et al. [5] destacan tres clases. La información estructurada, como los Knowledge Graphs, es fundamental para tareas de razonamiento factual ya que permite modelar entidades y relaciones explícitas. La información no estructurada, como corpus abiertos (Common Crawl, PubMed, Wikipedia), se ha vuelto estándar en open-domain QA. Tal como explican Gao et al. [7] estos corpus aportan amplitud temática, pero también requieren mecanismos de filtrado, chunking y re-ranking para evitar ruido y mitigar el problema de Lost in the Middle. Finalmente, surge la categoría de contenido generado por LLM, donde el propio modelo actúa como fuente en esquemas de self-retrieval, generando y reutilizando conocimiento de manera autónoma. En este caso, los LLMs generan documentos intermedios, hipótesis o representaciones que se utilizan posteriormente como consultas o evidencia, empleando un módulo crítico que evalúa si es necesario recuperar información externa o si el propio contenido generado basta para resolver la tarea. Gracias a este enfoque, como subraya Fan et al. [5], se abre la posibilidad de que los modelos se autocomplementen y reutilicen su conocimiento previo sin depender exclusivamente de bases externas.

Procesos de augmentation: se clasifican en tres modalidades según Zhao et al. [29]. Augmentation puede aplicarse una sola vez (Once), como en el caso de la document augmentation, donde se enriquece directamente la entrada con información adicional antes de la generación. Otra modalidad es la iterativa, que emplea técnicas como el pseudo-relevance feedback (PRF), mediante el cual la consulta inicial se reformula a partir de los resultados recuperados, repitiendo el ciclo para refinar la relevancia. La modalidad adaptativa es cuando el sistema decide dinámicamente si conviene recuperar información o no. Un ejemplo de esto es Selective Knowledge Retrieval (SKR), que evita búsquedas innecesarias cuando el modelo ya posee el conocimiento suficiente, reduciendo costes y minimizando la incorporación de ruido.

El componente generador (ver Tabla 1.7) es el encargado de producir texto, imágenes u otro tipo de contenido. La capacidad del generador no depende únicamente de su arquitectura sino de un conjunto de estrategias que optimizan su rendimiento y controlan la calidad de las salidas. Estas mejoras incluyen prompt engineering, el fine-tuning especializado en dominios y ajustes en los métodos de decodificación pa-

Categoría	Subcategoría / Tipo	Ejemplos
Tipos	Transformers	GPT, BART, T5
	LSTM	Modelos secuencia a secuencia tradicionales
	GANs	Generación adversarial en imágenes y texto
	Diffusion Models	Imagen, audio y video
Enhancements	Prompt Engineering	Diseño de instrucciones, chain of thought, step-back prompts
	Generator Fine-tuning	Ajuste del modelo al dominio específico
	Decoding Tuning	Beam search, nucleus sampling, top-k sampling

Tabla 1.7: Componente Generator

ra equilibrar coherencia y diversidad en la generación. En este sentido, comprender tanto los tipos de modelos como las técnicas de optimización resulta fundamental para evaluar el papel del *Generator* en sistemas avanzados como los de Retrieval-Augmented Generation (RAG), donde la combinación de arquitectura y optimización garantiza la generación de respuestas más fiables, contextualizadas y relevantes [4, 5, 28]

Los transformers son la arquitectura más influyente en NLP ya que, según lo menciona Casola, Lauriola y Lavelli [4], su capacidad de manejar dependencias a largo plazo mediante mecanismos de auto-atención ha permitido el desarrollo de modelos como GPT, BART y T5, los cuales han superado ampliamente a enfoques previos y marcado un cambio de paradigma en la generación de lenguaje. Los LSTM (Long Short-Term Memory) se empleaban en arquitecturas secuencia a secuencia, resolviendo problemas de memoria en redes recurrentes, aunque presentaban limitaciones en el escalamiento y en la captura de dependencias largas [13]. Por otro lado, introdujeron un enfoque basado en el enfrentamiento entre un generador y un discriminador, logrando avances en la creación de imágenes y, posteriormente, en la generación de texto. Al mismo tiempo, se desarrollaron las GANs (Generative Adversarial Networks) las cuales plantean un aprendizaje adversarial [12] entre un generador y discriminador, donde modelan distribuciones complejas y producir datos sintéticos realistas, inicialmente en imágenes, pero posteriormente también en texto y audio. Más recientemente los Diffusion Models se han consolidado en el ámbito multimodal (imagen, audio y video), gracias a su capacidad de producir datos de

[12]Dos modelos con objetivos opuestos se enfrentan para la mejora mutua

alta calidad a partir de procesos de ruido inverso, ampliando las fronteras de la generación más allá del texto.

Con respecto a la mejora, el Prompt Engineering ha demostrado ser una técnica central para guiar los modelos hacia respuestas más precisas y controladas. Según Zhang y Zhang [28] incluye el diseño de instrucciones específicas, así como métodos como chain of thought o step-back prompting, que mejoran la coherencia y reducen alucinaciones en los resultados. Otra estrategia es el fine-tuning que consiste en adaptar un modelo a un dominio concreto, optimizando su capacidad de manejar información especializada y aumentando su fiabilidad en tareas críticas, por ejemplo en contextos científicos o legales. El Decoding Tuning se refiere al ajuste de los métodos utilizados por un modelo generativo para decidir qué palabra o token producir a continuación. En lugar de limitarse a elegir siempre la opción con mayor probabilidad, lo cual puede generar textos repetitivos y poco naturales, se aplican estrategias que permiten modular el balance entre coherencia y diversidad. Entre ellas, el beam search explora varias rutas posibles de generación en paralelo para seleccionar la más prometedora, garantizando coherencia aunque sacrificando creatividad. Por su parte, el nucleus sampling restringe las opciones a un conjunto dinámico de palabras que concentran la mayor parte de la probabilidad acumulada y selecciona una de ellas de manera aleatoria, lo que produce textos más naturales y variados. Finalmente, el top-k sampling limita las elecciones a las k palabras más probables y elige una de ellas de acuerdo con su distribución de probabilidad, lo que ofrece un equilibrio controlado entre precisión y diversidad. Estas estrategias permiten modular el equilibrio entre coherencia y diversidad en la generación de texto, adaptando el comportamiento del modelo sin necesidad de modificar su arquitectura [10].

Fases de Implementación

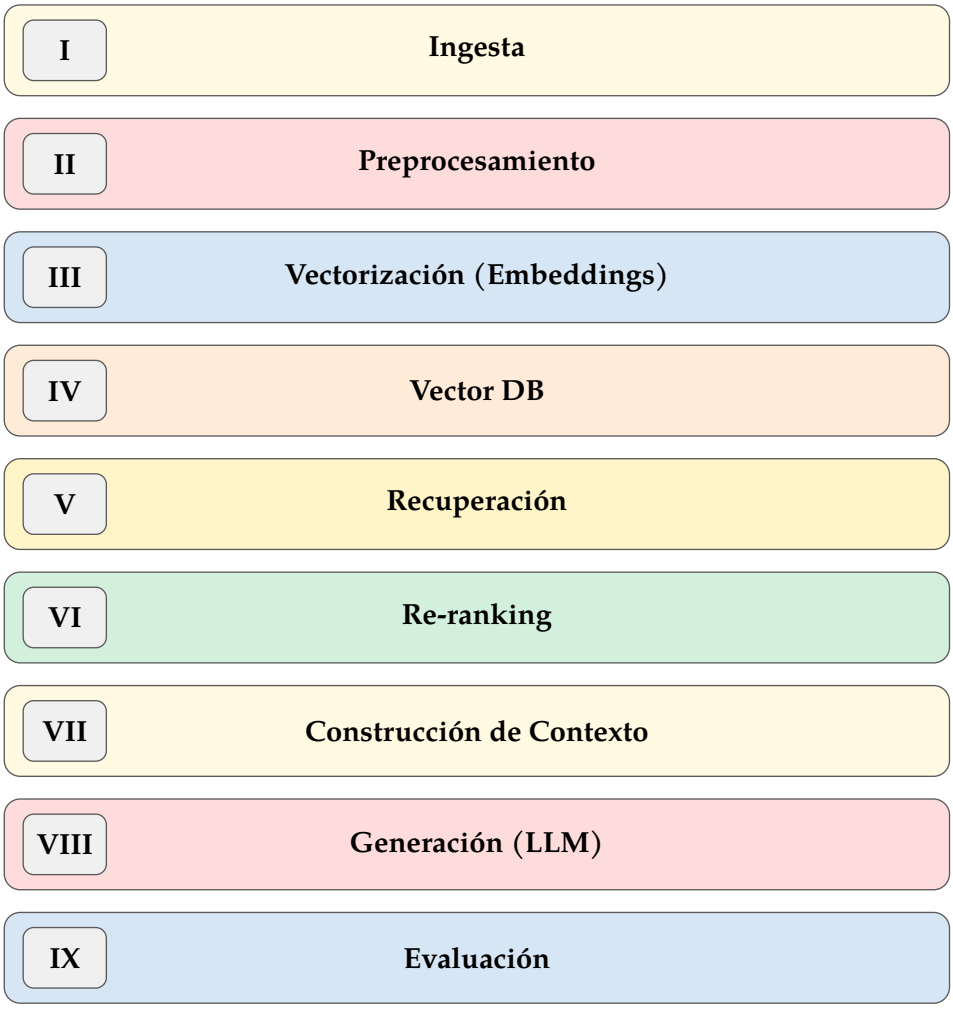

Figura 1.4: Pipeline de RAG

Las fases de implementación de los sistemas RAG son una secuencia de pasos (Ver Figura. 1.4) diseñados para enriquecer la generación de respuestas con información externa y actualizada de estos sistemas. Según Tabassum y Patil [23], todo inicia con la ingesta y el preprocesamiento de los datos, donde las técnicas de limpieza y segmentación garantizan que el texto sea utilizable para fases posteriores. Luego se vectoriza mediante embeddings los cuales transforman los fragmentos de texto en representaciones numéricas para la recuperación semántica tal como lo expresa Minaee et al. [18]. Para Hu y Lu [10], estos vectores se almacenan en una base de datos vectorial la cual permite realizar búsquedas eficientes por similitud semántica entre la consulta y los fragmentos de texto indexados. Además, esta VDB permite trabajar en conjunto con índices tradicionales como BM25, dando lugar a la recuperación hí-

brida. Con los datos obtenidos, como lo señala Sarthi et al. [22], es necesario aplicar procesos de re-ranking y filtrado que permiten priorizar las partes más relevantes dando lugar a la construcción de contexto, en la que los fragmentos de texto seleccionados se organizan, se condensan para reducir su extensión y se adaptan al límite de entrada del modelo para conformar un contexto optimizado y manejable que servirá de entrada al modelo generativo. En la fase final, Knollmeyer et al. [15] sostiene que el LLM debe complementarse con mecanismos de evaluación para asegurar la fidelidad y coherencia de las respuestas generadas.

Ingesta: Es el comienzo del pipeline de RAG, su objetivo es recopilar, normalizar y preparar las fuentes de información que luego serán usadas para los procesos de recuperación y generación. La ingesta implica recolectar distintas fuentes ya sean libros, literatura científica, corpus abiertos, etc., y convertirlos en un formato manejable por el sistema. De acuerdo con Gao et al. [7], Ibrihich et al. [11] y Jing, Su y Han [13] estos datos se almacenan como texto estructurado, semiestructurado, sin estructura o heterogéneos, dependiendo de su naturaleza y del tipo de procesamiento requerido.

Preprocesamiento: Dentro de la ingesta de datos es necesario limpiar y estructurar el texto en bruto para que sea utilizable en etapas posteriores. De acuerdo con Tabassum y Patil [23] los pasos a seguir son los siguientes:

1. **Normalización:** Estandarizar la forma del texto (codificación, idioma, minúsculas, puntuación no informativa, espacios, símbolos y números aislados) para asegurar consistencia a lo largo del corpus.
2. **Segmentación en oraciones:** Delimitar correctamente las oraciones.
3. **Tokenización:**[13]. Perrmite transformar datos no estructurados en unidades coherentes y consistentes, reduciendo el ruido que podría propagarse en fases posteriores.
4. **Eliminación de *stopwords* :** Retirar términos funcionales poco informativos cuando no aportan al objetivo de la tarea.
5. **Stemming o Lematización:** Ambas técnicas buscan reducir la variabilidad morfológica de las palabras para unificar formas derivadas y mejorar la coherencia

[13]Proceso de segmentar el texto en unidades básicas (palabras, signos de puntuación o incluso caracteres), por ejemplo: “NLP”, “is”, “the”, “future”, etc

del corpus. El *stemming* consiste en reducir las palabras a una raíz mediante reglas que recortan sufijos, agrupando variantes flexivas como (*compute, computing, computed* $\rightarrow$ *comput*). Es un método rápido y agresivo que puede generar formas no canónicas, pero resulta útil cuando la pérdida morfológica no afecta el rendimiento, como en clasificación o búsqueda léxica a gran escala. Por su parte, la *lematización* mapea cada término a su forma canónica del diccionario (p. ej., *caring* $\rightarrow$ *care*), preservando el significado y la información gramatical. Suele preferirse cuando la morfología y el contexto sintáctico influyen en la efectividad del modelo.

6. **Fragmentación en *chunks* :** Adaptar el texto procesado a las limitaciones de contexto de los LLM mediante partición en segmentos manejables.

Este paso es importante ya que el texto procesado debe adaptarse a las limitaciones de los LLM, lo cual implica fragmentar en chunks y generar representaciones vectoriales, siendo así que Knollmeyer et al. [15] afirma que la calidad del preprocesamiento incide directamente en dimensiones de evaluación como la relevancia contextual y la fidelidad de las respuestas. Una preparación deficiente del texto puede introducir ruido en el contexto recuperado y favorecer inconsistencias en la generación, lo cual incrementa el riesgo de alucinaciones; en este sentido, se entiende que el preprocesamiento actúa como una primera barrera de mitigación de errores (Zhang y Zhang [28]).

Vectorización: Los fragmentos de texto una vez preprocesados se transforman en *embeddings*[14], permitiendo capturar la semántica de tal manera que los textos similares estén próximos en el espacio vectorial. De acuerdo con Hu y Lu [10], existen dos enfoques principales:

- **Recuperación dispersa (sparse).** Métodos como TF-IDF y BM25 modelan la relevancia a partir de la frecuencia estadística de términos en los documentos [6]. Posteriormente, técnicas como *Latent Semantic Analysis* (LSA) introducen reducción dimensional (SVD) sobre la matriz término–documento para identificar estructuras semánticas latentes, superando limitaciones basadas únicamente en frecuencias; varios trabajos la consideran un punto de partida hacia los embeddings neuronales [6, 18].

[14] Representación numérica que codifica el significado del texto en un espacio vectorial

- **Recuperación densa.** Basada modelos preentrenados, como BERT, que ofrecen representaciones más contextuales y expresivas. Con el auge del *deep learning* surgieron representaciones densas como Word2Vec y GloVe, capaces de capturar relaciones semánticas en espacios continuos; los *transformers* preentrenados, al combinar atención y preentrenamiento masivo, producen vectores contextualizados robustos que capturan dependencias de largo alcance [4].

Vector DB: Los VDBs son importantes dentro del desarrollo de sistemas RAG, pues permiten almacenar y organizar representaciones densas de texto para su posterior recuperación eficiente. Estos se emplean en lugar de los índices invertidos tradicionales, que basan la búsqueda en coincidencias exactas de palabras o estadísticas de frecuencia, lo cual dificulta la detección de sinónimos y relaciones semánticas. De acuerdo con Jing, Su y Han [13], los VDBs están diseñados para manejar grandes volúmenes de representaciones de alta dimensionalidad y soportar búsquedas por similitud mediante métricas como la distancia euclidiana o el coseno. Además, como menciona Fan et al. [6], el uso de representaciones preentrenadas mejora la calidad de los índices, ya que los embeddings capturan relaciones semánticas profundas entre consultas y documentos. Para optimizar estas búsquedas se incorporan técnicas de indexación, que organizan los embeddings en estructuras preparadas para la búsqueda aproximada de vecinos más cercanos [17].

Para llevar a cabo estas operaciones, los VDBs recurren a diferentes enfoques de búsqueda, tales como:

- **Nearest Neighbor Search (NNS):** compara de manera exhaustiva el vector de consulta con todos los vectores almacenados. Garantiza la máxima precisión, pero resulta poco escalable con colecciones masivas de datos [17].
- **Graph-Based Approach:** emplea estructuras de grafos, como HNSW, que permiten navegar de forma eficiente entre vectores y localizar vecinos cercanos en pocas iteraciones, reduciendo considerablemente los tiempos de búsqueda [13].
- **Tree-Based Approach:** organiza los vectores en estructuras jerárquicas (ej. KD-Tree, Ball-Tree), dividiendo recursivamente el espacio. Es eficiente en dimensionalidades bajas o medias, pero pierde rendimiento en espacios de muy alta dimensión [6].

- **Approximate Nearest Neighbor Search (ANNS):** sacrifica una mínima exactitud para ganar eficiencia y escalabilidad. Métodos como IVF o PQ permiten realizar búsquedas rápidas en grandes colecciones, siendo muy usados en producción [17].

Cuando una consulta es transformada en vector, el sistema no necesita recorrer todo el corpus, sino que localiza la región del espacio donde se ubica y devuelve los vecinos más próximos. Para lograr esta eficiencia se emplean técnicas de indexación que organizan los embeddings en estructuras optimizadas para la búsqueda aproximada. Se implementa mediante enfoques como:

- **HNSW (Hierarchical Navigable Small World graphs):** construye un grafo jerárquico navegable que permite acceder rápidamente a los vecinos más cercanos, reduciendo la complejidad de la búsqueda [17].
- **IVF (Inverted File Index):** agrupa los vectores en listas invertidas de centroides, de modo que la comparación se restringe a las regiones más prometedoras del espacio [17].
- **PQ (Product Quantization):** divide los vectores en subespacios y los representa de forma comprimida, lo que acelera significativamente el cálculo de distancias en colecciones de gran tamaño [17].

De acuerdo con las referencias Ma et al. [17] y Wang et al. [24], la Tabla 1.8 muestra las herramientas y sistemas para recuperación vectorial en RAG, separando ANN (implementación de índices) de VDBs (servicios de almacenamiento/consulta y capacidades de producción). Las librerías ANN aportan el motor algorítmico que calcula vecinos cercanos en espacios vectoriales usando métodos y estructuras de indexación para búsqueda por similitud (IVF, PQ, HNSW). En general no resuleven tareas de gestion datos, control de concurrencia, replicación , etc. Por lo que son considerados componentes de bajo nivel que se agregan en sistemas mayores. Por otro lado, las bases de datos vectoriales (VDBs) integran esos mismos índices como motores internos y añaden la capa de gestión necesaria para operar a escala

Tipo	Nombre	Descripción breve
Librería ANN		
	FAISS	Librería de Meta para búsqueda de vecinos más cercanos con soportes clásicos (IVF, PQ) y aceleración CPU/GPU; ampliamente usada como backend en VDBs.
	ANNOY	Índices basados en árboles para ANN, común en escenarios de baja/mediana dimensionalidad y escenarios con índices mayormente inmutables. Prioriza bajo uso de memoria y latencias de lectura estables
	SPTAG	Librería de Microsoft para ANN orientada a colecciones extensas combinando estructuras basadas en particionamiento y grafos para mejorar el recupero por similitud y la escalabilidad.
VDB (Sistema)		
	Milvus	VDB distribuido; soporta múltiples índices (IVF, PQ, HNSW, RNSG), CPU/GPU, y arquitectura de almacenamiento/consulta escalable.
	Weaviate	VDB gestionado con índices ANN y consultas híbridas (atributos + semántica).
	Qdrant	VDB orientado a ANN (HNSW), filtrado por metadatos y despliegue gestionado.
Softwate as a Service	Pinecone	VDB gestionado orientado a baja latencia y operación a gran escala. Soporta escalado horizontal y vertical y tres funciones de distancia soportadas (inner product, cosine, euclidean).
Extensión DB relacional	pgvector	Extensión de PostgreSQL para vectores; habilita ANN/NNS dentro de un RDBMS(Relational DataBase Management System).
Motor de búsqueda	Elasticsearch	Motor de búsqueda generalista que opera con un índice invertido, en versiones recientes cuenta con capacidades vectoriales(dense/sparse). Carece de algunas funciones propias de VDB especializadas.(p. ej., GPU y multi-vector query nativas)

Tabla 1.8: Librerías ANN y bases de datos vectoriales (VDBs)

Recuperación: Esta fase es el punto donde el usuario interactúa con el conocimiento almacenado y su eficiencia y eficacia se reflejan en la calidad de la respuesta obtenida. Como señalan Fan et al. [6], la recuperación en sistemas de IR se centra en estimar la relevancia entre la consulta y los documentos, devolviendo una lista ordenada de resultados que responden a la necesidad de información. De acuerdo con Hambarde y Hugo [8], los enfoques de recuperación se dividen en cuatro categorías principales:

- **Convencional**: se basa en modelos clásicos como el booleano, el de espacio vectorial o BM25, donde la similitud depende de coincidencias exactas de términos y estadísticas de frecuencia. Estos métodos son eficientes para recuperar información en grandes colecciones, aunque limitados ante problemas de sinonimia o polisemia [8].
- **Dispersa (sparse)**: representa consultas y documentos como vectores escasos. Una línea reciente ha mejorado este enfoque mediante esquemas neuronales que aprenden a ponderar términos o expandir documentos, como DeepCT o Doc2Query, integrando así señales semánticas en modelos léxicos tradicionales [8, 7].
- **Densa (dense)**: emplea embeddings generados por modelos preentrenados (p. ej., BERT o variantes de transformers), que permiten medir similitud en espacios vectoriales continuos. Según Hu y Lu [10], este enfoque captura relaciones semánticas profundas y resulta más eficaz en tareas de recuperación semántica abierta. Ejemplos destacados son Dense Passage Retrieval (DPR) y ANCE [8].
- **Híbrida**: combina la precisión léxica de los enfoques dispersos con la riqueza semántica de los densos. Como indican Zhai [27], este tipo de estrategias incrementa tanto la cobertura como la pertinencia, al integrar coincidencias exactas y relaciones semánticas, lo que resulta fundamental para sistemas RAG modernos.

Re-Ranking: Una vez recuperada la información requerida, el sistema aplica técnicas de ranking para refinar el orden y priorizar los fragmentos más relevantes. Este proceso puede apoyarse en métodos tradicionales como BM25, en algoritmos de learning to rank (LTR)[15]. Como explican Bernard y Balog [2], LTR permitió combinar múltiples señales de ranking en un modelo unificado, superando las limitaciones

[15]LTR se refiere al uso de algoritmos de aprendizaje supervisado para optimizar el orden de los resultados en un sistema de recuperación de información.

de los métodos heurísticos. Entre los algoritmos más representativos se encuentran RankNet, LambdaRank y LambdaMART, que introdujeron redes neuronales y técnicas de gradiente boosting para mejorar la calidad del ordenamiento. Según Hambarde y Hugo [8], el LTR representa un punto de transición entre los métodos tradicionales y los basados en deep learning, abriendo el camino hacia arquitecturas neuronales.

Dentro de los enfoques de deep learning para re-ranking, se distinguen los siguientes componentes principales:

- **Representación basada en modelos**: codifican la consulta y el documento de manera independiente en un espacio vectorial, y luego miden la similitud entre ambas representaciones. Son eficientes, aunque limitados en capturar interacciones finas entre texto y consulta [18].
- **Interacción basada en modelos**: en lugar de representaciones independientes, modelan explícitamente las relaciones término a término entre consulta y documento. Esto permite capturar señales de correspondencia más precisas, a costa de mayor costo computacional [12].
- **Modelos attention-based**: utilizan mecanismos de atención (como el self-attention de los transformers) para ponderar dinámicamente qué partes del texto y la consulta son más relevantes, mejorando la identificación de dependencias contextuales [8].
- **Transformers preentrenados**: arquitecturas como BERT, RoBERTa o T5 han revolucionado el re-ranking al permitir un entendimiento contextual profundo. Estos modelos, ajustados con tareas de recuperación, han demostrado superar ampliamente métodos previos [27].

Construcción de Contexto: La Tabla 1.10 resume los componentes de la construcción de contexto en RAG junto con su respectiva referencia.

Aspecto	Descripción breve	Referencia
Integración del contexto	Unir fragmentos reordenados; elegir lo relevante, quitar redundancias y dejar evidencia lista para el generador.	Gao et al. [7]
Agregación y filtrado	Priorizar con re-rank/umbrales y deduplicar para evitar sobrecarga y reducir ruido.	Sarthi et al. [22]
Restricciones del LLM	Respetar la ventana: fragmentar, resumir o comprimir; mala preparación aumenta alucinaciones.	Gao et al. [7]
Enfoques avanzados	Usar jerarquías y resúmenes intermedios para maximizar relevancia y coherencia global.	Sarthi et al. [22]

Tabla 1.10: Construcción de contexto en RAG

Generación (LLM): Durante esta fase, el modelo de lenguaje no solo transforma los fragmentos de texto en un texto general, sino que también decide cómo integrar la información recuperada con su conocimiento preentrenado. En la literatura, este proceso se conceptualiza mediante diferentes estrategias de fusión, que buscan equilibrar el aporte externo con las capacidades internas del modelo [29, 8]. Las técnicas de fusión son:

- **Query-based Fusions**: la información recuperada se incorpora desde la etapa de entrada, concatenando la consulta con los pasajes relevantes para que el modelo los procese conjuntamente. Este enfoque, también denominado *early fusion*, es sencillo de implementar y aprovecha la arquitectura del transformador para contextualizar documentos y consulta en un mismo espacio semántico [7].
- **Logits-based Fusions**: en este caso, la fusión ocurre en la etapa de salida, combinando las distribuciones de probabilidad (logits) producidas por el modelo generador con las provenientes de la información recuperada. De esta forma, se refuerzan las respuestas que cuentan con evidencia externa, reduciendo la probabilidad de alucinaciones [27].

- **Latent Fusions**: aquí la integración se realiza en un espacio latente intermedio, donde los embeddings de la consulta y los documentos recuperados se mezclan con las representaciones internas del LLM. Según Hu y Lu [10], esto permite un balance más fino entre lo que sabe el LLM y lo que aporta la recuperación externa.

En conjunto, estas estrategias reflejan que la generación no es un proceso lineal, sino una etapa flexible donde se decide cómo y en qué nivel se integra la recuperación con el modelo. Como señalan Zhao et al. [29], la elección del mecanismo de fusión es determinante para la fidelidad y coherencia de las respuestas generadas.

Evaluación: La evaluación en sistemas RAG debe abordarse de forma multifásica, verificando tanto la recuperación (calidad y suficiencia del contexto) como la generación (fidelidad y atribución), ya que el desempeño global depende de ambas etapas del pipeline Knollmeyer et al. [15].
Enfoque por fases. La evaluación en los sistemas RAG es un proceso donde no se centra únicamente en evaluar la calidad de la respuesta generada, sino que debe evaluarse en todas las fases de implementación.
Verificación del *retriever*. Empezando por el retriever, donde se verifica que esté encontrando evidencia relevante y suficiente.
Métricas de recuperación. Para llevar a cabo esta verificación se utilizan métricas tradicionales como Recall@k, nDCG o MRR que permiten cuantificar qué proporción de los documentos útiles fueron efectivamente recuperados.
Impacto sobre la fidelidad. Es importante verificar que estén recuperando los documentos adecuadamente, ya que la etapa de generación no podrá producir respuestas fieles si carece de evidencia adecuada, por más sofisticado que sea el generador.
Generación y *groundedness*. En segundo lugar, medimos la generación más allá de indicadores como BLEU, ROUGE o F1, introduciendo el concepto de fidelidad o groundedness, que examina si el texto producido está sustentado en las fuentes recuperadas, reduciendo el riesgo de alucinaciones.
Atribución y trazabilidad. Este criterio debe complementarse con la atribución de citas, es decir, si el sistema enlaza explícitamente sus afirmaciones con los fragmentos de textos extraídos, lo que facilita la trazabilidad [28].
Evaluación general del sistema. Finalmente, es necesaria una evaluación general del sistema donde se considere la utilidad práctica en un escenario real. Aquí entran en juego dimensiones como eficiencia (latencia y costo), robustez frente a consultas adversas y adaptabilidad en dominios específicos. Knollmeyer et al. [15] destacan que

requiere combinar métricas automáticas con juicios humanos o LLM-as-a-judge calibrados.

Criterios FATE. Sin embargo, en los últimos años se ha propuesto integrar criterios de Fairness, Accountability, Transparency and Ethics (FATE) como parte de la evaluación. Según Bernard y Balog [2], la confianza en los sistemas de recuperación y generación depende no solo de su precisión, sino también de que sean justos (sin sesgos desproporcionados hacia ciertos grupos), responsables (documentando el procedimiento de evaluación y anotadores), transparentes (explicando cómo se recuperó y procesó la información) y éticos (respetando la privacidad, la diversidad de fuentes y evitando desinformación).

Iteración y multimodalidad. Incluir FATE como un parámetro de evaluación permite generar confianza social y regulatoria, además de estar alineado con la calidad técnica. Como sugiere Ramdurai [21], la evaluación debe ser iterativa y multimodal: incluir tanto experimentos controlados en benchmarks como pruebas de usabilidad en escenarios reales. De esta manera se garantiza que el sistema sea confiable, útil, socialmente responsable y no solamente funcione bien en métricas numéricas.

Paradigmas

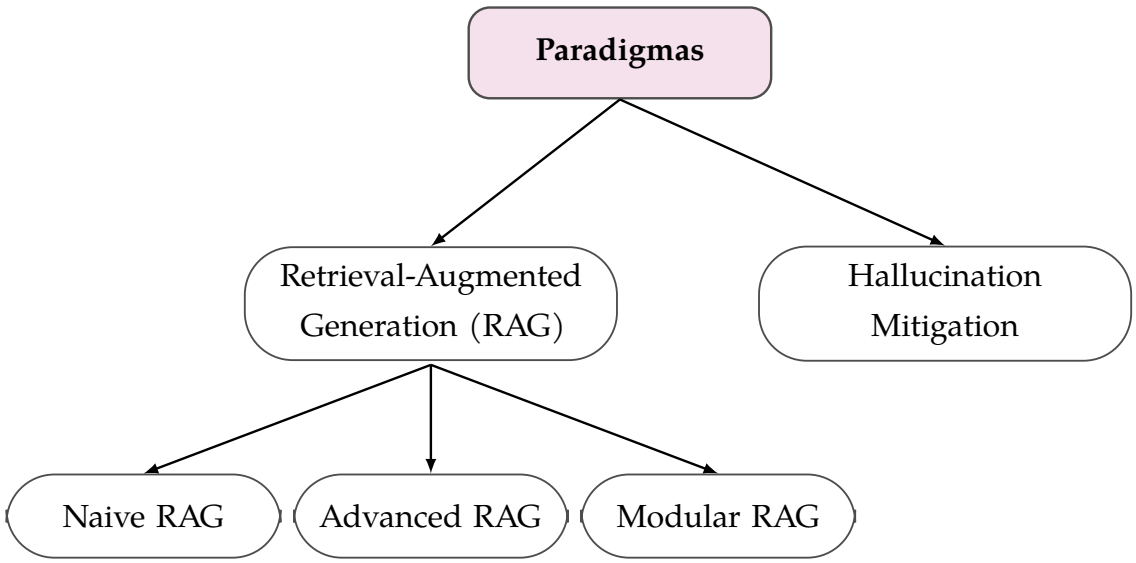

Figura 1.5: Paradigmas principales en RAG

La Figura 1.5 muestra el campo de Retrieval-Augmented Generation (RAG) que, según Gao et al. [7] distinguen los paradigmas Naive RAG, Advanced RAG y Modular RAG, que representan un progreso desde enfoques básicos de recuperación y

generación hasta arquitecturas modulares y flexibles. A su vez, Zhang y Zhang [28] clasifican los problemas en el área de Hallucination Mitigation en dos ejes: retrieval failure (fallos en fuentes, consultas o recuperadores) y generation deficiency (ruido o conflicto contextual y límites de capacidad). Finalmente, Zhao et al. [29] proponen fundaciones de RAG según la forma en que el retriever complementa al generador: Query-based RAG, Latent Representation-based RAG, Logit-based RAG y Speculative RAG. De esta forma se expande la aplicación de RAG a múltiples dominios y modalidades.

Tipos de Paradigmas

- **Retrieval-Augmented Generation (RAG) — Gao et al. (2023)**
  - Naive RAG: enfoque básico de recuperación y generación.
  - Advanced RAG: incorpora optimizaciones como segmentación fina, reranking y recuperación iterativa.
  - Modular RAG: paradigma flexible con módulos especializados para búsqueda, memoria, alineación y validación.
- **Tipos de RAG — Zhao et al. (2024)**
  - Query-based RAG: integra directamente la consulta y la información recuperada en la entrada del generador.
  - Latent Representation-based RAG: incorpora la información recuperada como representaciones latentes en el modelo generativo.
  - Logit-based RAG: combina la información de recuperación en la fase de decodificación a nivel de logits.
  - Speculative RAG: sustituye pasos de generación por recuperación para acelerar y reducir costes.
- **Hallucination Mitigation — Zhang & Zhang (2025)**
  - Retrieval failure: fallos en las fuentes, consultas, recuperadores o estrategias de recuperación.
  - Generation deficiency: deficiencias en la generación como ruido, conflictos contextuales, middle curse, problemas de alineación y límites de capacidad.

Evaluación y Métricas

La evaluación de los sistemas de Recuperación de Información (IR) y de Retrieval-Augmented Generation (RAG) ha sido objeto de un creciente interés en la literatura reciente. En el ámbito de IR, Bernard y Balog [2] destacan que las nociones de equidad, transparencia y responsabilidad requieren enfoques diversos: mientras la *fairness* se mide mayormente con métricas automáticas, la *transparency* y la *accountability* suelen evaluarse mediante auditorías y estudios con usuarios. En contraste, la dimensión ética carece de métricas claras y se asocia más a aspectos como privacidad y seguridad.

En el área de RAG, Knollmeyer et al. [15] identifican cinco dimensiones clave de evaluación: relevancia del contexto, fidelidad (*faithfulness*), relevancia de la respuesta, corrección y calidad de las citas. Estas dimensiones permiten evaluar de manera más integral los sistemas que combinan recuperación y generación. Asimismo, Gao et al. [7] señalan que, además de las métricas automáticas tradicionales de IR (precisión, recall, nDCG), resulta fundamental incluir evaluaciones humanas que capten aspectos como coherencia, transparencia y verificabilidad.

Categorías de Evaluación

- **Enfoques de Evaluación — Gao et al. (2023)**
    - Evaluación independiente por etapas: análisis separado de retrieval, augmentation y generation, con métricas específicas en cada módulo.
    - Evaluación End-to-End: valoración directa de la salida final del sistema RAG, con dos variantes:
        - Evaluación End-to-End automática: frameworks que miden habilidades clave como exactitud, fidelidad (*faithfulness*), atribución de fuentes, reducción de alucinaciones y transparencia.
        - Evaluación End-to-End con juicio humano: juicios expertos que valoran coherencia, verificabilidad, utilidad práctica y confianza.
    - Combinación de métricas: integración de métricas clásicas de recuperación (precisión, recall, nDCG), métricas de generación (coherencia, verificabilidad, calidad narrativa) y evaluación humana.
- **Evaluación en IR con FATE — Bernard & Balog (2025)**
    - Fairness: métricas automáticas como *top-k*, *exposure* y *pairwise metrics*.

- Transparency y Accountability: auditorías y estudios de usuarios, centrados en interpretabilidad y trazabilidad.
- Ethics: enfoques cualitativos vinculados a privacidad y seguridad.
- Tensiones: conflictos entre fairness individual vs. grupal y entre transparencia excesiva vs. carga cognitiva.

- **Evaluación Clásica en IR — Hambarde & Proença (2023)**
  - Métricas de recuperación: precisión, recall, F1, MAP (Mean Average Precision), MRR (Mean Reciprocal Rank).
  - Métricas de ranking: nDCG (Normalized Discounted Cumulative Gain) y calidad del ordenamiento.
  - Evaluación de modelos neuronales: comparación con benchmarks tradicionales (TREC, MS MARCO).
  - Dimensión multimodal: evaluación de IR en escenarios que integran texto, imágenes y audio.

- **Evaluación en RAG — Knollmeyer et al. (2024); Gao et al. (2023)**
  - *Fase de Recuperación*:
    - Context relevance: grado de pertinencia de los documentos recuperados.
    - Dataset quality: uso de bases de datos curadas y representativas (ej. Wikipedia, MS MARCO).
    - Métricas aplicadas: precisión, recall, nDCG, cobertura de conocimiento.
  - *Fase de Generación*:
    - Faithfulness: consistencia factual entre recuperación y generación.
    - Answer relevance: utilidad de la respuesta respecto a la consulta.
    - Correctness: exactitud de la información producida y reducción de alucinaciones.
    - Métricas aplicadas: BLEU, ROUGE, métricas de consistencia factual y evaluaciones humanas.
  - *Fase de Integración*:
    - Citation quality: precisión, trazabilidad y cobertura de las fuentes citadas.

- *Evaluación Global*:
    - Evaluadores mixtos: combinación de métricas automáticas (similaridad semántica, BLEU, ROUGE) con juicios humanos (coherencia, verificabilidad, utilidad práctica).
    - Datasets: necesidad de colecciones específicas adaptadas a RAG, más allá de benchmarks tradicionales de IR.

Futuro de RAG

Debido al poder de otorgar documentos como evidencia, aportando contexto y trazabilidad de las fuentes, RAG se consolidará como una capa estructural de acceso conversacional al conocimiento, donde LLM e IR cooperan para entregar respuestas verificables y reducir alucinaciones. De esta forma, avances como la recuperación jerárquica y el uso de bases de datos vectoriales fortalecen la integración de contexto y la gobernanza de la evidencia, habilitando sistemas que citan, planifican y mantienen conocimiento actualizado [27].

Desafíos Actuales:
De acuerdo con Zhai [27], uno de los principales desafíos de los sistemas RAG es la generación de alucinaciones, que comprometen la confianza y limitan su uso en aplicaciones críticas. Asimismo, persisten limitaciones en la calidad de la recuperación y en la eficiencia computacional, especialmente en dominios donde se requiere información actualizada y especializada [10]. Además, Ramdurai [21] señala que la integración de RAG con otras arquitecturas, como redes neuronales convolucionales, enfrenta problemas de escalabilidad y de adaptación a contextos heterogéneos. También es clave subrayar que persisten problemas éticos y de transparencia (FATE) en los sistemas de recuperación y generación. En la literatura de Bernard y Balog [2] muestra que aún no existen marcos estandarizados para evaluar imparcialidad, transparencia y responsabilidad en IR, lo que limita la confianza social en los sistemas RAG. Otro reto es la reproducibilidad y robustez de los modelos preentrenados, ya que la variabilidad en hiperparámetros y semillas afecta la consistencia de los resultados [6].

Direcciones Potenciales:
Las líneas de avance incluyen el desarrollo de retrievers más robustos, la optimización de las interacciones entre modelo y recuperación, así como mecanismos de

evaluación que prioricen la relevancia y la coherencia. Ramdurai [21] enfatiza la sinergia con arquitecturas multimodales y con sistemas capaces de combinar información textual, visual y estructurada. Asimismo, enfoques como RAPTOR, que introduce resúmenes recursivos y jerárquicos en la recuperación, muestran cómo superar la fragmentación del contexto y mejorar la integración semántica [22]. Otra dirección destacada es la incorporación de bases de datos vectoriales como memorias semánticas persistentes, que permiten gestionar grandes volúmenes de embeddings, mitigar el olvido catastrófico de los modelos y ofrecer personalización en la recuperación [13]. Finalmente, cabe mencionar que los pre-trained transformers aplicados a IR han abierto el camino hacia modelos más sofisticados para tareas de ranking y matching semántico, aunque todavía requieren diseños de preentrenamiento adaptados a los desafíos específicos de la recuperación de información [6].

Perspectivas:
A medio plazo, RAG y los LLM convergerán con la IR clásica hacia sistemas de acceso conversacional al conocimiento: motores de búsqueda capaces de generar respuestas enriquecidas con análisis propio, pero con respaldo en documentación verificable, capaces de planificar, citar y aprovechar el historial conversacional. Incluso si los LLM siguen mejorando, la recuperación seguirá siendo un componente clave porque permite reducir costos, mantener la información siempre actualizada, control y gobernanza de la evidencia usada y en consecuencia se espera la aparición de arquitecturas integradas donde RAG funcione junto con herramientas externas y módulos memoria trabajen de forma coordinada. Esta coordinación hará posible planificar y ejecutar tareas más largas o compuestas. La evaluación de estos sistemas ya no se limitará a medir métricas técnicas como la exactitud, sino que pondrá el énfasis en dos partes clave: la utilidad práctica y la veracidad, especialmente en contextos de alta exigencia.

Capítulo 2

Metodología

2.1. Revisión sistemática

Umbrella Review, según los lineamientos del Instituto Joanna Briggs (JBI), es un tipo de revisión sistemática que recopila y analiza evidencia secundaria, es decir, revisiones sistemáticas y metaanálisis ya publicados. Su propósito es consolidar el conocimiento disponible, identificar coincidencias y contradicciones en la literatura existente, así como señalar vacíos de evidencia. Para ello, requiere la elaboración de un protocolo previo que establezca criterios de inclusión y exclusión, estrategias de búsqueda y métodos de síntesis, garantizando un proceso transparente y riguroso.

Por otra parte, la estrategia de propagación de citaciones (Back-and-Forward Citation Propagating) complementa este enfoque al permitir encontrar dinámicamente la literatura. A través de la propagación de citaciones se amplía y actualiza la literatura encontrada en las bases de datos tradicionales. De este modo, se superan limitaciones como la indexación incompleta, las variaciones en el uso de palabras clave o la exclusión de ciertas publicaciones.

Metodología: Umbrella Review con Propagación de Citaciones

Como parte de la metodología Umbrella Review es necesario establecer un protocolo para ejecutar la revisión. Se han considerado las siguientes fases para dicho protocolo:

1. **Propósito de la revisión**

   La revisión se justifica en la necesidad de consolidar evidencia secundaria de

calidad, aprovechando el enfoque de propagación de citaciones para garantizar una búsqueda amplia, estructurada y actualizada.

2. **Objetivos específicos**

   Se definen los objetivos generales y específicos que guiarán la identificación de literatura mediante la propagación de citaciones, así como el proceso de síntesis de resultados.

3. **Criterios de inclusión y exclusión**

   Se definen de manera general como la incorporación de revisiones y metaanálisis que sean pertinentes, de calidad y relacionados con el tema de estudio, y la exclusión de aquellos trabajos que no cumplan con estos requisitos de relevancia.

4. **Identificación del estudio semilla y propagación de citaciones**

   La búsqueda se inicia en bases de datos académicas como *Scopus*, *Web of Science*, *IEEE Xplore* o *Google Scholar*, a fin de localizar un estudio semilla (revisión o resumen amplio) que ofrezca una cobertura representativa del tema. A partir de este estudio, se aplica la estrategia de Back-and-Forward Citation Propagation, que combina:

   - *Backward citation:* revisión de las referencias citadas en el estudio semilla.
   - *Forward citation:* identificación de trabajos más recientes que citan al estudio semilla.

   De este modo, el corpus de literatura se amplía progresivamente hasta alcanzar un punto de saturación en el que la propagación deja de aportar nueva evidencia relevante.

5. **Selección de revisiones relevantes**

   A partir de la propagación de citaciones, se aplican los criterios de inclusión - exclusión para determinar qué revisiones serán incorporadas al análisis.

6. **Valoración de la calidad de la evidencia**

   La calidad de los estudios se evalúa según los criterios definidos, garantizando su consistencia al tema de estudio. Para apoyar este proceso se usa una herramienta de análisis que facilite la organización y valoración sistemática de la evidencia.

7. **Extracción de información clave**

   De cada revisión seleccionada se extraerán datos esenciales, organizados en una tabla de extracción que incluirá:

   - Autor y año de publicación
   - Objetivo del estudio
   - Tipo de revisión
   - Número de estudios primarios incluidos
   - Principales hallazgos
   - Conclusiones generales
   - Limitaciones reportadas

8. **Síntesis y representación de resultados**

   Los hallazgos se organizarán en dos niveles complementarios:

   - **Tabular:** tablas comparativas de las revisiones incluidas.
   - **Narrativo:** síntesis descriptiva de los principales hallazgos.
   - **Temático y visual:** mapas de evidencia y esquemas que reflejen la propagación de citaciones, mostrando las conexiones entre estudios clave.

9. **Discusión y conclusiones**

   Los resultados se interpretan desde una perspectiva crítica, destacando fortalezas, limitaciones y la evolución de la evidencia en el tiempo. Se identifican coincidencias y divergencias entre revisiones, así como vacíos de conocimiento, y se proponen líneas de investigación futura.

En esta metodología, el Umbrella Review se utiliza como marco general para sintetizar evidencia secundaria a partir de revisiones de exhaustivas de la literatura, complementándose con la propagación de citaciones para integrar aportes recientes y reflejar la evolución del conocimiento disponible.

2.2. Enfoque Design Science Research (DSR)

De acuerdo con vom Brocke et al. Brocke, Hevner y Maedche [3], Design Science Research, desarrollada en 1969, es un paradigma de resolución de problemas que

busca mejorar el conocimiento humano mediante la creación de artefactos innovadores. En otras palabras, es una metodología que crea soluciones a problemas reales y, al mismo tiempo, genera conocimiento útil y aplicable sobre cómo diseñar estas soluciones. Las etapas que se aplicarán en el presente trabajo son las siguientes:

- **Identificación del problema y motivación** En esta etapa se precisa el problema y se justifica por qué es necesaria una solución. De acuerdo con Peffers et al. (2008), esta etapa exige analizar el problema en detalle, descomponiéndolo en sus partes clave para identificar sus causas, efectos y alcance. Además, es crucial justificar la relevancia del problema, tanto desde una perspectiva teórica (es decir, cómo contribuye al conocimiento académico) como desde una perspectiva práctica (cómo afecta a organizaciones, usuarios o sistemas reales). También implica explorar la literatura para verificar que el problema es relevante, desafiante y nuevo, lo que permite definir los límites del proyecto de investigación.

- **Definir los objetivos para la solución** Se plantean los criterios que debe cumplir una solución exitosa basándose en el conocimiento existente y en la factibilidad técnica y organizacional. Los objetivos deberán permitir construir algo efectivo y deseable, no solamente desde el ámbito académico sino también en el entorno en que se aplicará. Estos pueden expresarse en términos cualitativos o cuantitativos; el investigador establece aquí la meta hacia donde se dirigirá el artefacto.

- **Diseño y desarrollo del artefacto** En esta etapa se construye una solución concreta, como un modelo, software o sistema, que responde directamente a los objetivos planteados. Para ello, se utiliza el conocimiento existente que fundamenta las decisiones del diseño y la estructura del artefacto. No solo se trata de crear algo, sino de asegurar que pueda ser comprendido, evaluado y replicado por otros.

- **Demostración del uso del artefacto para resolver el problema** Se muestra cómo se usa el artefacto en un escenario real o simulado. Esta demostración no valida científicamente su efectividad, sino que muestra su aplicabilidad, evidenciando que el artefacto propuesto puede operar de forma efectiva. Por su parte, vom Brocke et al. (2020) destacan que esta etapa es fundamental para conectar el diseño teórico con la realidad del usuario o del entorno organizacional, permitiendo detectar oportunidades de mejora antes de una evaluación rigurosa.

- **Evaluación del desempeño del artefacto** Se busca medir su efectividad, eficiencia e impacto, aportando evidencia que justifique su valor y utilidad. Además, según vom Brocke et al. (2020), esta puede asumirse de forma continua mediante una evaluación formativa que permita ciclos iterativos de rediseño y mejora a lo largo del proceso de investigación.
- **Comunicación de los resultados al público académico y profesional** Finalmente, esta etapa consiste en difundir de forma clara los resultados del diseño y de la investigación realizada.

Estos pasos están basados en el modelo clásico de DSR de Peffers (2008), que vom Brocke adapta y expande en su guía. En la Figura 2.1 muestra el ciclo de DSR dedicado a Centinela. La fila superior muestra las 6 fases y los recuadros inferiores brindan mayor informacion acerca de cada fase. Además, el contenedor inferior resume tres puntos de entrada de investigación que guían las decisiones de diseño y delimitan el alcance. En conjunto, conectan el marco teórico con las decisiones técnicas y la evaluación, asegurando coherencia a lo largo del proyecto.

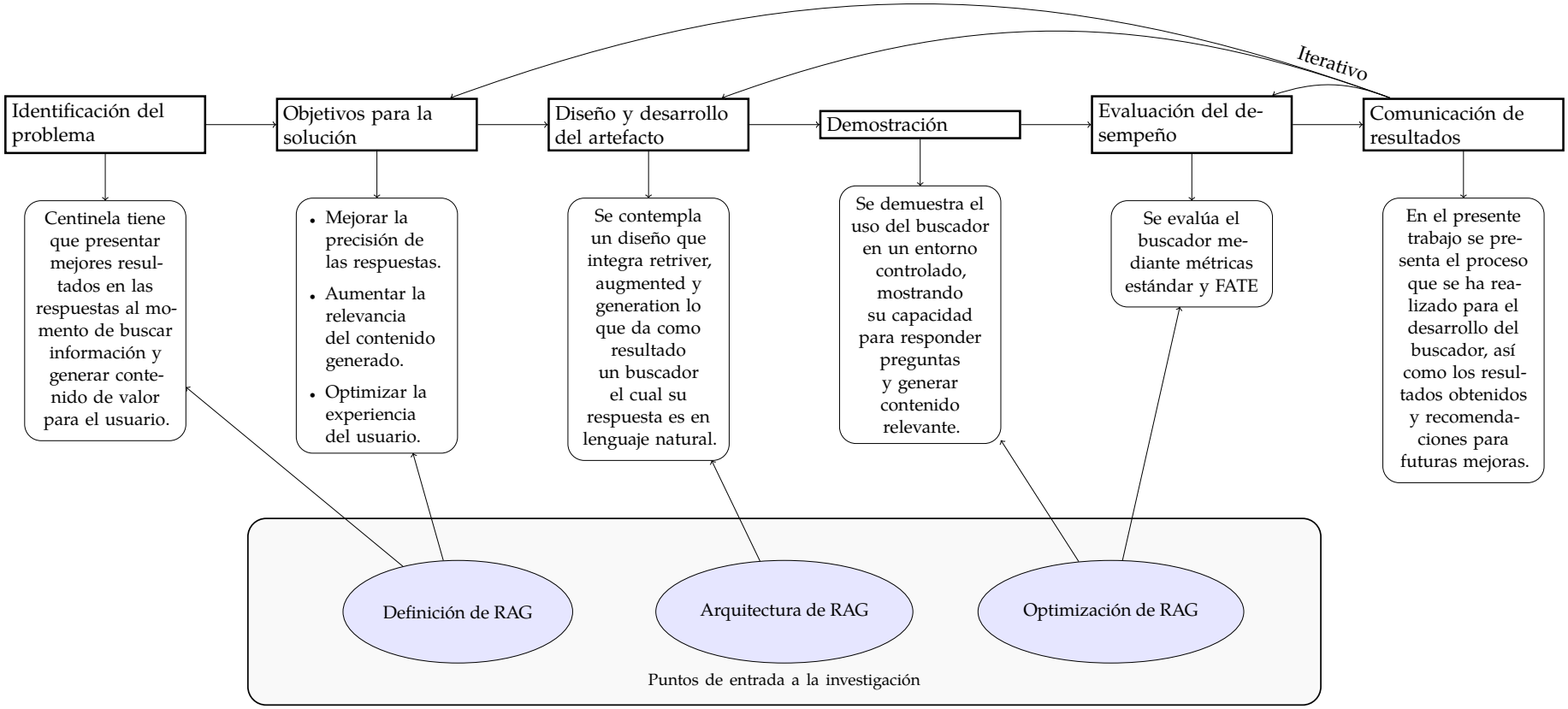

Figura 2.1: Proceso de Diseño de Investigación para el desarrollo de RAG en Centinela

2.3. Diseño y desarrollo del artefacto

De acuerdo con a la literatura, el desarrollo de RAG para Centinela se realiza usando las fases expuestas en la Figura 1.1. La implementación, en `Jupyter Notebook`, está disponible en el repositorio de GitHub kalech7/RAG-Centinela.

2.3.1. Ingesta

De acuerdo al estudio exhaustiva de la literatura, el primer paso para la construccion de RAG es la obtencion de los datos. En nuestro caso los datos provienen de Scopus donde se extrae los metadatos de los articulos, en particular título y abstract, los cuales concentran la representación semántica principal del artículo y son los puntos de entrada más frecuentes para consultas y embeddings[27]. De acuerdo con la literatura [27, 15, 6, 11, 2, 17], se han saleccionado los siguientes campos que contienen la información necesaria para representar el contenido del documento e identificarlo.

- **Título y abstract**: condensan la información temática central del articulo y son la base principal para la recuperación y la generación de embeddings.
- **DOI**: garantiza trazabilidad y constata la fuente, enfrenta a los problemas de confianza y alucinación en RAG.
- **Autores**: son un metadato clave ya que permite analizar impacto académico y coautorías.
- **Afiliaciones**: aportan contexto institucional y geográfico, lo cual resulta útil para filtrar información por país o institución.
- **Número de citaciones**: es un metadato adicional para priorizar documentos de mayor impacto en la fase de ranking.
- **ScopusID**: asegura que cada documento tenga un identificador único, evitando duplicados en la VDB.

De esta forma los campos título y abstract concentran la mayor parte de semántica relevante para crear embeddings y recuperar documentos, por lo que conviene tratarlos como el bloque principal de indexación. En cambio, DOI, ScopusID, Autores y Afiliaciones funcionan como metadatos de soporte: ayudan a auditar la procedencia, filtrar por institución o región, destingir entre entidades y evitar duplicados en

la base vectorial. Separar título/resumen de metadatos de control (DOI, ScopusID, autores, afiliaciones) mejora la interpretabilidad del sistema y evita sobrecargar el espacio vectorial con señales que no aportan significado textual directo.

Con la siguiente función se extrae el corpus y se guardan en un CSV (UTF-8) para su procesamiento en fases posteriores.

Algorithm 1 Export_Articles_to_CSV

Input: NEO4J_URI, NEO4J_AUTH, OUT_CSV=`scopusdata.csv`
Output: CSV (UTF-8, `sep="|"`, no index) with flattened metadata

Query Neo4j (filtered + ordered).
$query \leftarrow$ CypherForArticles ▷ filter by non-empty `scopus_id`, `title`, ...
$rows \leftarrow$ RunCypher($NEO4J\_URI$, $NEO4J\_AUTH$, $query$)
$df \leftarrow$ DataFrame($rows$) ▷ tabularize query results
Join list-valued columns
Function JoinList(x)
    return IsList(x) ? Join(x, `", "`) : x
for all $c \in$ Cols(df) **do** ▷ for all c in columns
    for all $r \in$ Rows(df) **do**
        if IsList(df[c][r]) **then**
            $df[c][r] \leftarrow$ Join($df[c][r]$, `", "`)
Set final column order.
$cols \leftarrow$ [`title`, `abstract`, `doi`, `authors`, ...]
$df \leftarrow$ ReindexColumns(df, $cols$)
Export CSV.
WriteCSV(df, OUT_CSV, sep=`"|"`, encoding=`utf-8`, index=`False`)

2.3.2. Preprocesamiento

Una vez completada la ingesta de datos provienientes de Scopus, es fundamental preprocesarlos para normalizar y limpiar el texto, reducir ruido y mejorar la calidad semántica de las representaciones. En esta etapa aplicare el siguiente flujo reproducible compuesto por estos pasos:

1. **Normalización**. Consiste en estandarizar la forma del texto mediante UTF-8, la corrección de caracteres extraños, la conversión a minúsculas, la eliminación de puntuación no informativa, símbolos y números aislados, así como la normalización de espacios y se etiqueta el idioma de cada documento; no se fuerza la traducción, preservando la variación lingüística para análisis por idioma. para garantizar la coherencia. Cuando corresponde, se desduplican registros o

resúmenes (comparación por DOI o por título+autor+año) con el fin de homogeneizar el vocabulario y reducir sesgos por repeticiones. Con el siguiente pseudocodigo se procedera a normalizar las columnas que contienen toda la semantica:

Algorithm 2 Process and Label_Language

Input: IN_CSV=`scopusdata.csv`
Input: PARQ_1=`processed.parquet`
Input: SRC=`abstract_norm`
Output: `processed_lbl.parquet` (con `lang`)

Load CSV

$df \leftarrow$ ReadCSV($IN\_CSV,\ sep = "|"$)

Normalize text fields

Function Normalize(s)
    $s \leftarrow$ FixText(s)
    $s \leftarrow$ Lower(s); $s \leftarrow$ UnicodeNFC(s) ▷ lowercase and canonical Unicode
    $s \leftarrow$ CleanSymbols(s); $s \leftarrow$ CollapseWhitespace(s) ▷ punctuation; spaces
    return s
df[`title_norm`] $\leftarrow$ Map(Normalize, df[`title`])
df[`abstract_norm`] $\leftarrow$ Map(Normalize, df[`abstract`])

Label language from abstract_norm

SetLangDetectSeed(0)
Function DetectLangSafe(t)
    if $\neg$IsString(t) $\vee$ IsBlank(t) **then**
        return `und`
    try
    return DetectLang(t)
    catch
    return `und`
df[`lang`] $\leftarrow$ Map(DetectLangSafe, $df[SRC]$)

Save labeled table

SaveParquet(df, OUT, compression=`gzip`)

Se lee el CSV (separador "|") y se generan title-norm y abstract-norm corrigiendo codificación, pasando a minúsculas y limpiando caracteres no informativos; luego se guarda todo en formato Parquet (columna-orientado y comprimido) que conserva el esquema, reduce tamaño y acelera lecturas/filtrado frente a CSV. A continuación se carga el Parquet procesado, se detecta el idioma de abstract-norm con langdetect (columna lang, usando "und" si es indeterminado) y se guarda un nuevo Parquet. La detección de idioma permite tratar corpus multilingües de forma óptima (stopwords/tokenización por lengua y elección del modelo de embeddings multilingüe).

2. **Segmentación en oraciones**. Dividir el texto en oraciones completas utilizando delimitadores válidos (puntos, signos de cierre), con el fin de estructurar unidades coherentes para el análisis posterior.

Algorithm 3 Segment Text

Input: IN(parquet),OUT(parquet-out),COL=`abstract_norm`,LANG_COL=`lang`
Output: Table with a sentence segmentation derived from `COL` using `LANG_COL`

$df \leftarrow$ ReadParquet(IN) ▷ load table
$segs \leftarrow \{$ `en` $\mapsto$ PySBD(en), `es` $\mapsto$ PySBD$(es)\}$ ▷ load language segmenters
Function SegmentByLang(text, lg)
    $k \leftarrow$ LowerPrefix(lg) ▷ e.g., "en", "es"
    $seg \leftarrow$ Get$(segs,\ k)$
    return seg.Segment$(text)$
for all $r \in$ Rows(df) **do** ▷ iterate over rows
    $text \leftarrow df[COL][r]$
    $lg \leftarrow df[LANG\_COL][r]$
    df[`sentences`]$[r] \leftarrow$ SegmentByLang$(text,\ lg)$ ▷ segment in its language
One row per sentence:
$out \leftarrow$ ExplodeRename$(df,$ `sentences` $\rightarrow$ `sentence`$)$
SaveParquet$(out,\ OUT)$

Se lee el Parquet de entrada, elige la columna abstract-norm y segmenta sus textos en oraciones usando pysbd según el idioma que se detecto previamente. Devuelve una lista de oraciones por fila; luego crea una fila por oración, conserva metadatos útiles añadiendo el índice de oración sentence-idx dentro de cada documento y escribe el resultado en processed-sentences.parquet.

3. **Tokenización**. Segmentar cada oración en tokens (palabras o subpalabras) preservando frases clave frecuentes (p. ej.,*aprendizaje automático*), a fin de mantener unidades semánticas estables.

Algorithm 4 Tokenize

Input: IN = Parquet path, COL = `sentence`

Output: Table with column `tokens_gram`

1: $df \leftarrow$ ReadParquet(`IN`) ▷ load
2: df[`tokens_base`] $\leftarrow$ SimpleTokenizeColumn(df, `COL`) ▷ create
3: **Load Gensim phrase models.**
4: $S \leftarrow$ ToList(df[`tokens_base`]) ▷ prepare list of token sequences
5: $bigP \leftarrow$ Phrases(S, *min_count* $= 5$, *threshold* $= 10.0$) ▷ train bigram model
6: $bigF \leftarrow$ Phraser($bigP$) ▷ wrap bigram
7: $S\_bi \leftarrow$ TransformAll($bigF$, S) ▷ apply bigram
8: $triP \leftarrow$ Phrases(S_bi, *min_count* $= 5$, *threshold* $= 10.0$) ▷ train trigram model
9: $triF \leftarrow$ Phraser($triP$) ▷ wrap trigram
10: df[`tokens`] $\leftarrow$ TransformAllChain([$bigF$, $triF$], df[`tokens_base`]) ▷ apply
11: df[`tokens_gram`] $\leftarrow$ JoinTokensByComma(df[`tokens`]) ▷ create
12: SaveParquet(df, `OUT`, *engine*=`fastparquet`, *comp*=`gzip`) ▷ save

Carga el parquet y tokeniza con una expresión regular y entrena detectores de bigramas y trigramas con Gensim usando mincount=5 y threshold=10. Luego aplica ambos phrasers para unir collocations frecuentes en un solo token el cual a su vez genera tokens. Los tokens son unidos por comas y se guardan en el nuevo parquet.

4. **Eliminación de stop words**. Remover términos de alta frecuencia y escaso valor semántico.

Algorithm 5 Remove Stopwords by Language

Input: IN = Parquet path, COL = `tokens`, LANGCOL = `lang` (`"es"` or `"en"`)

Output: Table with `tokens_nostop` and `text_for_embed`

1: $df \leftarrow$ ReadParquet(`IN`) ▷ load input

2: $LANGMAP \leftarrow \{$`"es"` $\rightarrow$ `"spanish"`, `"en"` $\rightarrow$ `"english"`$\}$

Stopword sets

3: $S_{\text{es}} \leftarrow$ NLTK.Stopwords(`spanish`)

4: $S_{\text{en}} \leftarrow$ NLTK.Stopwords(`english`)

5: norm$(s) \leftarrow$ Lower(StripDiacritics(s))

6: **Define** IsStop$(t,\ c)$:

$S \leftarrow$ LangStop(c); $u \leftarrow$ norm(t)

return $(u \in S)$

7: **Define** FilterTokens$(xs,\ c,\ \mathit{min_len} = 2)$:

return $[\, t \in xs \mid \text{Len}(t) \geq \mathit{min\_len} \wedge \neg\text{IsNum}(t) \wedge \neg\text{IsStop}(t, c)\,]$

8: df[`tokens_nostop`] $\leftarrow$ Map(FilterTokens, df[`COL`], df[`LANGCOL`])

9: df[`text_for_embed`] $\leftarrow$ JoinWithSpace(df[`tokens_nostop`])

SaveParquet(df, *comp*=`gzip`)

Una vez tokenizado eliminamos las stop words asegura las listas, normaliza a minúsculas y quita tildes. La función is-stop considera tokens compuestos (espacios/guiones) y solo los descarta si todas sus partes son stopwords; además, filter-tokens aplica filtros básicos (longitud mínima, quitar numéricos)

5. **Aplicación de Lematización o stemming**. Preferir lematización sobre *stemming* para conservar la forma canónica y reducir la varianza léxica sin pérdida de significado [23].

Algorithm 6 Lemmatize_spaCy

Input: IN=`corpus_stop.parquet`, TEXT_COL=`text_for_embed`

Output: Parquet with `text_lemma`

Load table

$df \leftarrow$ ReadParquet(IN);

Load spaCy pipelines

$nlp\_es \leftarrow$ LoadSpacy(`es_core_news_sm`)

$nlp\_en \leftarrow$ LoadSpacy(`en_core_web_sm`)

Build language masks from `lang` column

$idx\_es \leftarrow [\,$ LowerPrefix$(lg) =$ `es` **for** $lg \in df[$`LANG_COL`$]\,]$

$idx\_en \leftarrow [\,$ LowerPrefix$(lg) =$ `en` **for** $lg \in df[$`LANG_COL`$]\,]$

Function to lemmatize a string: return space-separated lowercase lemmas

Process ES batch

$stream\_es \leftarrow nlp\_es$.Pipe$(\dots)$ ▷ batched iterator of Docs

$lem\_es \leftarrow [\,$ LemmaJoin(d) **for** $d \in stream\_es\,]$

Process EN batch

$docs\_en \leftarrow nlp\_en$.Pipe$(df[$`TEXT_COL`$][idx_en],\ batch_size{=}512)$

$lem\_en \leftarrow [\,$ LemmaJoin(d) **for** $d \in docs\_en\,]$

Assemble output column

$text\_lemma[idx\_es] \leftarrow lem\_es$; $\quad text\_lemma[idx\_en] \leftarrow lem\_en$

$df[$`text_lemma`$] \leftarrow$ FillNA$(text_lemma,$ `""`$)$ ▷ replace nulls with empty string

Save

SaveParquet(df, OUT, compression=`gzip`)

Iniciamos leyendo el parquet de entrada de la fase anterior y y lematiza en lote con spaCy usando dos modelos (ES/EN) dependiendo de la etiqueta que habiamos usado agregado anteriormente. Genera tokens lematizados en minúsculas, separados por espacios y guarda el resultado en el nuevo parquet.

6. ***Chunking* para LLM/RAG**. Dividir documentos largos en fragmentos ($\sim$ 200–400 tokens) con solape (15–30 %) para respetar límites de contexto de los transformadores y mejorar la recuperación a nivel de pasaje (Hambarde & Proença, 2023).

Algorithm 7 Chunking `text_lemma`

Input: Parquet $IN\_PARQUET$ with `scopus_id`, `text_lemma`

Load *tokenizer* E5 (*multilingual-e5-base*)
$df \leftarrow$ ReadParquet($IN\_PARQUET$)
Group and clean text
$agg\_text \leftarrow$ GroupByJoin(df, $key =$ `scopus_id`, $col =$ `text_lemma`, $sep =$ "")
$doc\_df \leftarrow$ ResetIndex(agg_text); asignar `doc_id` consecutivo
doc_df.`text_for_chunk` $\leftarrow$ NormalizeSpaces(doc_df.`text_lemma`)
Chunking por tokens
Function ChunkByTokens($text$, MAX_TOKENS, $STRIDE$)
    $ids \leftarrow$ TokEncode($text$); $n \leftarrow |ids|$; $chunks \leftarrow [\,]$ ▷ map text → token IDs
    for $start \leftarrow 0$ **step** $STRIDE$ **to** n **do**
        $end \leftarrow \min(start+MAX\_TOKENS,\ n)$
        $chunks$.Append$\big(\langle start,\ end,\ \text{TokDecode}(ids[start{:}end])\rangle\big)$ ▷ decode token slice → text
        if $end = n$ **then**
            break
    return $chunks$
Build chunk table
$rows \leftarrow [\,]$
for all document d in doc_df **do**
    $chunks \leftarrow$ ChunkByTokens(d.`text_for_chunk`, MAX_TOKENS, $STRIDE$)
    for each chunk c with index j in $chunks$ **do**
        AppendColumns($rows$, {`doc_id`, `chunk_id`, ..., `text_chunk`})
$CHUNKS \leftarrow$ ToDataFrame($rows$)
Save
$save\_cols \leftarrow$ [`doc_id`, `chunk_id`, `chunk_uid`, `scopus_id`, ...]
$out \leftarrow chunks[save\_cols]$
ToParquet(out, `pyarrow`)

Finalmente, para fragmentar el texto tomamos el Parquet lematizado, consolidamos un solo documento por artículo y, con el tokenizer E5 (bi-encoder basado en transformer), lo separamos por conteo de tokens (tamaño = 300) con un solapamiento de 0.2.

Con el uso de este pipeline se disminuye el ruido de contexto suministrado al LLM en RAG, lo que reduce inconsistencias y riesgo de alucinaciones en la generación.

En términos de evaluación, una mejor normalización y segmentación favorecen la relevancia contextual del material recuperado y la fidelidad de las respuestas, al entregar fragementos más coherentes, comparables y medibles permitiendo calcular su similitud y una mejor relevancia contextual del material recuperado.

2.3.3. Vectorizacion

Para el diseño de este sistema se usa un retriever denso de arquitectura bi-encoder tipo BERT para la generación de embeddings, ya que permite codificar por separado la consulta y los fragmentos en un espacio vectorial de baja dimensión (vector corto y continuo de tamaño fijo) y comparar por similitud, capturando relaciones semánticas más allá de coincidencias léxicas y mejorando la recuperación en escenarios RAG y open-domain QA. Además, estos embeddings se indexan en bases de datos vectoriales con búsqueda aproximada de vecinos cercanos (ANN), lo que viabiliza el escalamiento eficiente del sistema[14].

Algorithm 8 Vectorizacion

Input: `chunks_df[text_chunk]`, `EMB_MODEL`, `FAISS_PATH`, `PKL_PATH`, `INIT_BATCH`, `EMB_MAX_SEQ_LEN`

$model \leftarrow$ LoadModel(`EMB_MODEL`, cpu) ▷ Load E5 on CPU
SetMaxSeqLen($model$, min(`EMB_MAX_SEQ_LEN`, 512)) ▷ Bound to 512
$P \leftarrow$ Prefix(`chunks_df.text_chunk`, $"passage: "$) ▷ E5 passage prefix
$index \leftarrow \varnothing$ ▷ Will hold IndexFlatIP
for $batch \in$ Batches(P, `INIT_BATCH`) **do** ▷ Encode in mini-batches
    $E \leftarrow$ Encode($model$, $batch$, *normalize* $= True$)
    if $index = \varnothing$ **then**
        $index \leftarrow$ NewIndexFlatIP(d)
    Add($index$, E) ▷ Append vectors to FAISS
WriteFAISS($index$, `FAISS_PATH`) ▷ Persist FAISS
EnsureVecID(`chunks_df`, $0..|P|-1$) ▷ 0-based contiguous ids
$cols \leftarrow$ [`vec_id`, `chunk_uid`, `doc_id`, `chunk_id`, ..., `scopus_id`]
$meta\_min \leftarrow$ Select(`chunks_df`, $cols$)
WritePKL({`model`, `device`, `dim`, `meta_min`}, `PKL_PATH`)

Toma los textos del archivo parquet, los limpia y les asigna identificadores (vec-id, chunk-uid), luego usa un modelo SentenceTransformer (E5) en CPU para convertir cada fragmento en un vector numérico normalizado (embedding). Procesa los

textos por lotes (reduce el tamaño si hay errores de memoria), junta todos los embeddings en una matriz (N, dim) y finalmente guarda en dos archivos .npy (donde estan los vectores) y .pkl con la información mínima (modelo, dimensiones e índices que vinculan cada vector con su texto original).

2.3.4. Vector DB

Algorithm 9 Build Vector Index from Embedding Matrix

Input: Paths: `IN_EMB`, `IN_META`, `OUT_INDEX`, `OUT_INFO`

Output: Serialized vector index and metadata file

1: $X \leftarrow$ LoadArray(`IN_EMB`) ▷ embedding matrix
2: $meta \leftarrow$ LoadObject(`IN_META`) ▷ load metadata info
3: $index \leftarrow$ InitIndex(*dim* $= meta.dim$, *metric* $=$ `inner_product`)
4: AddVectors($index$, X)
5: SaveIndex($index$, `OUT_INDEX`)
6: $info \leftarrow$ BuildRecord($model$, dim, $count$, $paths$)
7: SaveObject(`OUT_INFO`, $info$)

Carga los embeddings normalizados verifica que las dimensiones coincidan, construye un índice FAISS y el resultado se guarda en disco (.bin)

2.3.5. Recuperación

Algorithm 10 Query Search over Vector Index

Input: Paths: `PKL_MIN_PATH`, `FAISS_PATH`, `SCOPUS_CSV`

Output: Table of top-k retrieved documents with metadata

load artifacts:

$meta \leftarrow$ LoadPKL(`PKL_MIN_PATH`)[`"meta_min"`]

$index \leftarrow$ ReadFAISS(`FAISS_PATH`)

$model \leftarrow$ LoadModel(LoadPKL(`PKL_MIN_PATH`)[`"model"`], *device* = `cpu`)

encode query and search:

$q\_emb \leftarrow$ Encode($model$, [$"query:" + Q$], *normalize* = `True`)

($scores$, ids) $\leftarrow$ Search($index$, q_emb, k) ▷ retrieve top-k similar vectors

assemble top-k results:

$hits \leftarrow$ SelectRows($meta$, $ids[0]$)

AddColumn($hits$, `"score"`, $scores[0]$)

merge with external CSV:

$sc \leftarrow$ ReadCSV(`SCOPUS_CSV`, *sep* = `"|"`) ▷ load full Scopus metadata

EnsureType(sc[`"scopus_id"`], `str`)

$hits \leftarrow$ MergeLeft($hits$, sc, *on* = `"scopus_id"`)

Primero cargamos la vector DB, luego vectorizamos la consulta (query) y normaliza el embedding con ese vector hace búsqueda k-NN en FAISS y obtiene scores e ids. Para obtener un contexto mejorado se une el resultado con el CSV de Scopus por scopus-id para agregar la demas metadata que contiene el csv.

2.3.6. Re-ranking

Se implementa un cross encoder entrenado de segunda capa que re-ordena los candidatos recuperados en la primera etapa densa. El puntaje final se obtiene por fusión lineal (min-max + suma ponderada) entre el score denso de la recuperación inicial y el score del cross-encoder. Esta estrategia equilibra eficacia (precisión en el top-k) y eficiencia (coste computacional), y se ejecuta en CPU con MiniLM por ser un modelo compacto (menor latencia y memoria).

Algorithm 11 CE Re-Ranking (concise, with brief comments)

Input: query q, table D, text cols `text_cols`, model `CE_MODEL`, device, batch B, weights (w_{ce}, w_d)

Output: D' sorted by `score_final`

Function SELECTTEXT($r,\ C$) ▷ pick first non-empty text field
  for $c \in C$ **do**
    if NONEMPTY($r[c]$) **then return** $r[c]$
  ERROR ▷ no valid text
Function MINMAX(x) ▷ normalize to [0,1] per query
  $m \leftarrow \min(x),\ M \leftarrow \max(x)$
  if $M \leq m$ **then**
    ERROR ▷ constant vector
  return $(x - m)/(M - m)$
Function FUSE(z_{ce}, z_d, w_{ce}, w_d) ▷ simple convex fusion
  return $w_{ce} z_{ce} + w_d z_d$
ASSERT($|D| > 0 \wedge |\texttt{text\_cols}| > 0$) ▷ basic checks
$CE \leftarrow$ CROSSENCODER(`CE_MODEL`, `device`) ▷ load CE (once)
$P, I \leftarrow [\,], [\,]$ ▷ (q,doc) pairs and index map
for row r at idx i in D **do**
  append $(q,$ SELECTTEXT$(r, \texttt{text_cols}))$ to P ▷ build CE input
  append i to I ▷ keep original index
$S_{ce} \leftarrow$ BATCHEDPREDICT(CE, P, B) ▷ batched CE inference
$D' \leftarrow D$ ▷ work on a copy
for j **do** $D'[\texttt{score\_ce}]_{I[j]} \leftarrow S_{ce}[j]$ ▷ write CE scores
$z_{ce} \leftarrow$ MINMAX$(D'[\texttt{score_ce}])$ ▷ normalize CE
if `score_dense` $\in$ COLS(D') **then**
  $z_d \leftarrow$ MINMAX$(D'[\texttt{score_dense}])$ ▷ normalize dense
  $D'[\texttt{score\_final}] \leftarrow$ FUSE$(z_{ce}, z_d, w_{ce}, w_d)$ ▷ fuse signals
else
  $D'[\texttt{score\_final}] \leftarrow z_{ce}$ ▷ no dense: CE only
return SORTBY$(D',$ `score_final`, *desc* $= True)$ ▷ final ranking

Primero toma una lista de candidatos recuperadaos para una consulta y los reordena usando un cross encoder. Construye pares (consulta, texto) a partir de las columnas de texto prioritarias, infiere puntajes del CE por lotes, normaliza esos punta-

jes (y, si existe, el puntaje denso del recuperador) con min-max y luego los fusiona mediante una suma ponderada convexa.

2.3.7. Construcción de Contexto

La evidencia recolectada en el paso anterior se convierte en un bloque de evidencia utilizable por el LLM, equilibrando cobertura semántica y límite de longitud. Proporcionar cada texto en chunks manejables (longitud fija), preservando su trazabilidad (título, autores, año, DOI); y ensamblar un "paquete" de contexto con instrucciones de uso para el modelo

Algorithm 12 BuildContextBlocks

Input: reranked table D; top-k; character limit L; flag $trim$

Output: list $blocks$ with fields `cite_id`, `title`, `text`, `authors_mention`, `authors_raw`, `year`, `doi_raw`

$T \leftarrow$ first column present in `[title, chunk_title]` ▷ pick title source

$B \leftarrow$ first column present in `[abstract, chunk_text, summary]` ▷ pick body source

$n \leftarrow \min(k, |D|)$; $blocks \leftarrow [\,]$ ▷ cap k and init output

for $i \leftarrow 0..n-1$ **do** ▷ iterate top-n rows

  $r \leftarrow D[i]$ ▷ current row

  $title \leftarrow$ FirstNonEmpty$(r, [T])$ **or** "Untitled!" ▷ get title

  $body \leftarrow$ FirstNonEmpty$(r, [B])$ ▷ get body text

  if $trim$ **then** ▷ optional clipping

    $body \leftarrow$ Shorten$(body, L)$ ▷ enforce char limit

  $auth\_raw \leftarrow r$[`authors`] **or** "" ▷ raw authors

  $mention \leftarrow$ FormatAuthorsForMention$(auth_raw)$ ▷ short citation string

  $year \leftarrow$ ExtractYear(r) ▷ publication year

  $doi \leftarrow r$[`doi`] **or** "" ▷ DOI if present

  $cid \leftarrow$ RowId(r) ▷ prefer scopus_id"

  append { `cite_id`: cid, `title`: $title$, `text`: $body$, `authors_mention`: $mention$, `authors_raw`: $auth\_raw$, `year`: $year$, `doi_raw`: doi } to $blocks$

return $blocks$ ▷ final list

A partir de la tabla reranqueada, se eligen las columnas para título (title/chunk title) y cuerpo (abstract/chunktext/summary), se limita el número de filas a top-k y

se crea la lista blocks. Para las n primeras filas se obtienen el título y el texto; si trim está activo, el cuerpo se recorta hasta L caracteres. Se capturan los autores completos y se genera una mención abreviada, se extraen el año, el DOI y el identificador (scopusid). Con estos campos se arma cada bloque y devuelve blocks, listo para alimentar el prompt.

2.3.8. Generación (LLM)

En esta penultima etapa el LLM produce una respuesta dependiendo de la evidencia externa extraida previamente. recivbe un prompt estruturado que incluye la instruccion de estilo y formato la pregunta del usuario y los fragmentos selecionados tras la recuperacion y reraking el llm cita explícitamente las fuentes dentro de cada párrafo para asegurar trazabilidad y verificabilidad

Algorithm 13 LLMGenerateViaHTTP

Input: query q; list $blocks$; `MODEL`; `BASE_URL`; `TEMPERATURE`; `MAX_NEW_TOKENS`; `TIMEOUT`; `MAX_INPUT_CHARS`

Output: text *answer_text*

1: $prompt \leftarrow$ ComposePrompt(q, $blocks$, `MAX_INPUT_CHARS`)
2: $prompt \leftarrow$ SafeStr($prompt$) ▷ Max input chars
3: $url \leftarrow$ `BASE_URL` ‖ `"/api/chat"` ▷ chat endpoint
4: $payload \leftarrow$ {
`"model": MODEL,`
`"messages": [{role:"system", content:` "You are an academic assistant. Answer only using the provided context."}, {`role:"user", content:` prompt}],
`"options": {temperature: TEMPERATURE, num_predict: MAX_NEW_TOKENS},`
`"stream": false`
}
5: *resp* $\leftarrow$ HTTP_POST(url, *json* $= payload$, *timeout* $=$ `TIMEOUT`) ▷ call to the LLM backend
6: $data \leftarrow$ ParseJSON(*resp*)
7: *answer_text* $\leftarrow$ SafeStr($data$[`"message"`][`"content"`])
8: **return** *answer_text*

Para iniciar la generación, se construye el prompt a partir de la consulta y de los bloques de contexto, ajustándolos a un tope de caracteres y limpiandolos con

SafeStr. Ese prompt se envía como payload al endpoint de Ollama[1] (api/chat) junto con las opciones del modelo. Luego, se parsea el JSON de respuesta, se extrae message.content, se limpia nuevamente con SafeStr para asegurar texto válido y, finalmente, devuelve la respuesta.

2.3.9. Evaluación

[1]Ollama es un servidor para ejecutar modelos de lenguaje localmente (CPU o GPU).

Capítulo 3

Pruebas, Resultados, Conclusiones y Recomendaciones

3.1. Demostración

3.2. Evaluación del desempeño

3.3. Resultados

3.4. Conclusiones

3.5. Recomendaciones

Bibliografía

[1] Edoardo Aromataris et al. *Methodology for JBI Umbrella Reviews*. Adelaide: Joanna Briggs Institute, 2014. URL: https://ro.uow.edu.au/smhpapers/3344.

[2] Nolwenn Bernard y Krisztian Balog. «A Systematic Review of Fairness, Accountability, Transparency, and Ethics in Information Retrieval». En: *ACM Computing Surveys* 57.6 (feb. de 2025), 136:1-136:29. DOI: 10.1145/3637211.

[3] Jan vom Brocke, Alan Hevner y Alexander Maedche. «Introduction to Design Science Research». En: *Design Science Research. Cases*. Ed. por Jan vom Brocke, Alan Hevner y Alexander Maedche. Cham: Springer, 2020, págs. 1-13. DOI: 10.1007/978-3-030-46781-4_1.

[4] Silvia Casola, Ivano Lauriola y Alberto Lavelli. «Pre-trained transformers: An empirical comparison». En: *Machine Learning with Applications* 9 (2022), pág. 100334.

[5] Wenqi Fan et al. «A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models». En: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining KDD '24*. Barcelona, Spain: ACM, 2024, págs. 6491-6501. DOI: 10.1145/3637528.3671470.

[6] Yixing Fan et al. «Pre-training Methods in Information Retrieval». En: *arXiv preprint arXiv:2111.13853* (2021).

[7] Yunfan Gao et al. *Retrieval Augmented Generation for Large Language Models: A Survey*. 2023. arXiv: 2312.10997 [cs.CL].

[8] Kailash A. Hambarde y Proen Hugo. «Information Retrieval: Recent Advances and Beyond». En: *IEEE Access* 11 (2023), pág. 76581 76620. DOI: 10.1109/ACCESS.2023.3295776.

[9] Binglan Han, Teo Susnjak y Anuradha Mathrani. «Automating Systematic Literature Reviews with Retrieval Augmented Generation: A Comprehensive Overview». En: *Applied Sciences* 14.19 (2024), pág. 9103. DOI: 10.3390/app14199103.

[10] Yucheng Hu y Yuxing Lu. «RAG and RAU: A Survey on Retrieval-Augmented Language Models in Natural Language Processing». En: *arXiv preprint arXiv:2404.19543* (2024).

[11] S. Ibrihich et al. «A Review on recent research in information retrieval». En: *Procedia Computer Science*. Vol. 201. Elsevier, 2022, págs. 777-782. DOI: 10.1016/j.procs.2022.03.106.

[12] Peng Jiang y Xiaodong Cai. «A Survey of Text-Matching Techniques». En: *Information* 15.6 (2024), pág. 332. DOI: 10.3390/info15060332.

[13] Zhi Jing, Yongye Su y Yikun Han. «When Large Language Models Meet Vector Databases: A Survey». En: *arXiv preprint arXiv:2402.01763* (2024).

[14] Satyadhar Joshi. «Introduction to Vector Databases for Generative AI: Applications, Performance, Future Projections, and Cost Considerations». En: *International Advanced Research Journal in Science, Engineering and Technology* (*IARJSET*) 12.2 (2025), págs. 79-89. DOI: 10.17148/IARJSET.2025.12210.

[15] Simon Knollmeyer et al. «Benchmarking of Retrieval Augmented Generation: A Comprehensive Systematic Literature Review on Evaluation Dimensions, Evaluation Metrics and Datasets». En: *Proceedings of the 16th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management* (*IC3K 2024*) - *Volume 3: KMIS*. SCITEPRESS – Science y Technology Publications, Lda, 2024, págs. 137-148. ISBN: 978-989-758-716-0. DOI: 10.5220/0013065700003838.

[16] Tianyang Lin et al. «A survey of transformers». En: *AI Open* 3 (2022), págs. 111-132.

[17] Le Ma et al. «A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge». En: *arXiv preprint arXiv:2310.11703* (2025).

[18] Shervin Minaee et al. «Deep Learning-based Text Classification: A Comprehensive Review». En: *ACM Computing Surveys* 54.3 (2021), 62:1-62:40. ISSN: 0360-0300. DOI: 10.1145/3439726.

[19] Stefania Papatheodorou. «Umbrella reviews: what they are and why we need them». En: *European Journal of Epidemiology* 34.6 (2019), págs. 543-546. DOI: 10.1007/s10654-019-00505-6.

[20] Ken Peffers et al. «A design science research methodology for information systems research». En: *Journal of Management Information Systems* 24.3 (2008), pág. 45 77. DOI: 10.2753/MIS0742-1222240302.

[21] Balagopal Ramdurai. «Large Language Models (LLMs), Retrieval-Augmented Generation (RAG) systems, and Convolutional Neural Networks (CNNs) in Application systems». En: *International Journal of Marketing and Technology* 15.01 (2025). URL: https://www.researchgate.net/publication/387128512.

[22] Parth Sarthi et al. «RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval». En: *Proceedings of the International Conference on Learning Representations (ICLR)*. OpenReview, 2024.

[23] Ayisha Tabassum y Rajendra R. Patil. «A Survey on Text Pre-Processing & Feature Extraction Techniques in Natural Language Processing». En: *International Research Journal of Engineering and Technology (IRJET)* 7.6 (2020), págs. 4864-4870. ISSN: 2395-0056.

[24] Jianguo Wang et al. «Milvus: A Purpose-Built Vector Data Management System». En: *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. Industrial Track Paper. Virtual Event, China: Association for Computing Machinery, 2021. ISBN: 978-1-4503-8343-1. DOI: 10.1145/3448016.3457550. URL: https://doi.org/10.1145/3448016.3457550.

[25] Shangyu Wu et al. «Retrieval-Augmented Generation for Natural Language Processing: A Survey». En: *arXiv preprint arXiv:2407.13193* (2025).

[26] Shunyu Yao et al. «ReAct: Synergizing Reasoning and Acting in Language Models». En: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2023. URL: https://arxiv.org/abs/2210.03629.

[27] ChengXiang Zhai. «Large Language Models and Future of Information Retrieval: Opportunities and Challenges». En: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Washington, DC, USA: ACM, 2024, págs. 1-10. DOI: 10.1145/3626772.3657848.

[28] Wan Zhang y Jing Zhang. «Hallucination Mitigation for Retrieval-Augmented Large Language Models: A Review». En: *Mathematics* 13.5 (2025), pág. 856. DOI: 10.3390/math13050856.

[29] P. Zhao et al. «Retrieval-Augmented Generation for AI-Generated Content: A Survey». En: *arXiv* (2024). DOI: 10.48550/arXiv.2402.19473.