Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales

Generación automática de preguntas basada en grafos de conocimiento para optimización de sistemas de recuperación aumentada

Tesis de Licenciatura en Ciencias de Datos

Teo Gutter

Director: Luciano Del Corro

Buenos Aires, 2025

GENERACIÓN AUTOMÁTICA DE PREGUNTAS BASADA EN GRAFOS DE CONOCIMIENTO PARA OPTIMIZACIÓN DE SISTEMAS DE RECUPERACIÓN AUMENTADA

Los sistemas de Generación por Recuperación Aumentada (RAG) permiten a los grandes modelos de lenguaje (LLMs) acceder a información externa en tiempo real, superando así las limitaciones impuestas por su propio entrenamiento, como la dificultad para controlar con exactitud qué información conoce el modelo. Este enfoque no solo mejora significativamente la calidad de las respuestas generadas por asistentes basados en LLMs, sino que también permite incorporar conocimiento específico, confidencial o ausente en su entrenamiento. Como consecuencia, los sistemas RAG se están adoptando de manera cada vez más extendida y horizontal en la industria.

Los sistemas basados en RAG, sin embargo, enfrentan desafíos significativos relacionados con la latencia y el costo computacional. Además, estos sistemas suelen recuperar documentos que presentan alta similitud superficial pero baja diversidad semántica, lo que reduce la cobertura del contexto relevante y limita la capacidad del modelo para generar respuestas completas y bien fundamentadas.

Para mitigar estas limitaciones, esta tesis presenta KRAQ (*Knowledge-graph Representative Automatic Questions*), un sistema que permite precomputar conjuntos de preguntas representativas para un corpus determinado a partir de un grafo de conocimiento. Para ello, se realiza la detección de entidades y relaciones presentes en los textos con las cuales se construye el grafo. Luego, se identifican comunidades "semánticas" dentro del grafo que permiten la generación de resúmenes textuales. Finalmente, con un LLM *fine-tuneado* se genera un conjunto de preguntas representativas a partir de dichos resúmenes. La principal ventaja de este enfoque es su capacidad para generar preguntas que capturan relaciones profundas presentes en el corpus, incluso cuando dichas relaciones se extienden a través de múltiples documentos o no están formuladas explícitamente en el texto. De este modo, se obtiene un conjunto de preguntas verdaderamente representativas del contenido, que refleja de manera más fiel la estructura semántica subyacente.

Este listado de preguntas permite optimizar sistemas RAG: por un lado, (i) incrementando la precisión mediante estrategias de recuperación combinada, donde se enriquece el conjunto de documentos recuperados; y por otro, (ii) mejorando la latencia de sistemas de RAG como Speculative RAG, utilizando las preguntas generadas para pre-computar los embeddings necesarios para separar en subconjuntos los documentos.

La validación experimental, realizada en múltiples datasets estándar como TriviaQA, BioASQ, PubHealth y HotPotQA, demostró la efectividad de KRAQ. Los resultados evidencian que las preguntas representativas superan a los baselines por hasta 49 puntos porcentuales, mientras que su aplicación en sistemas RAG impulsó mejoras de hasta un 3 % en la precisión y reducciones de hasta un 11.8 % en la latencia. La incorporación de estas preguntas no solo mejora la eficiencia y precisión de RAG, sino que abre un camino prometedor para escalar esta tecnología.

Palabras claves: Generación Aumentada por Recuperación (RAG), Modelos de Lenguaje de Gran Escala (LLMs), Generación Automática de Preguntas (QG), Grafos de Conocimiento (KG), Clustering Semántico, Optimización de RAG.

AGRADECIMIENTOS

Agradezco a la universidad pública, para mí la institución más valiosa que tiene este país, por brindarme la oportunidad de estudiar y formarme de manera gratuita.

A mi familia, gracias por inculcarme la motivación al estudio y por hacer posible que pudiera dedicarme a aprender.

También quiero agradecer a los amigos que tuve e hice a lo largo de la carrera, quienes hicieron que todo el recorrido fuera hermoso. En especial, a Pablito Groisman, con quien compartí el 95 % de las materias lo que para mi tiene que romper algún record y que, además, me dio una mano en un momento de crisis durante la tesis.

A Luciano Del Corro, mi director, gracias por su tiempo, su escucha y por acompañarme en el proceso de este trabajo.

Y finalmente, a Nicolás Palermo, por haberme mostrado papers que fueron una fuente de inspiración para el desarrollo de esta tesis.

Índice general

1. INTRODUCCIÓN

1.1. Motivación

En los últimos años, el desarrollo de los modelos de lenguaje de gran escala (LLMs, por sus siglas en inglés) ha transformado profundamente las capacidades de los sistemas de procesamiento del lenguaje natural, permitiendo avances notables en tareas de generación, resumen, traducción y razonamiento [8, 55, 75]. Sin embargo, estos modelos presentan una limitación estructural clave: su conocimiento está restringido a la información contenida en los datos de entrenamiento y al momento temporal en que estos fueron recopilados. Además, la combinación de su compleja arquitectura y su escala masiva los convierte en sistemas inherentemente opacos, lo que dificulta controlar la información que realmente poseen [35]. Esto conlleva desafíos significativos en dominios donde se requiere acceso a conocimiento actualizado, especializado o confidencial [34].

Ante esta problemática, los sistemas de Generación por Recuperación Aumentada (Retrieval-Augmented Generation, RAG) han emergido como una solución efectiva que permite a los LLMs consultar información externa en tiempo real mediante la recuperación de documentos relevantes, que luego se utilizan como contexto adicional para la generación de respuestas. Este enfoque híbrido combina la capacidad generativa y de razonamiento contextual de los LLMs con mecanismos de recuperación, permitiendo no solo mejorar la precisión y factualidad de las respuestas, sino también incorporar información no presente en el pre-entrenamiento, algo esencial en contextos dinámicos o sensibles [40]. Como resultado, los sistemas RAG han sido ampliamente adoptados en aplicaciones industriales y por proveedores de servicios en la nube que ofrecen servicios integrales de RAG [1].

No obstante, a pesar de sus ventajas, los sistemas RAG enfrentan limitaciones persistentes. En primer lugar, el proceso de recuperación e integración de información externa conlleva una carga computacional significativa, lo cual incrementa la latencia y el costo operativo del sistema. En segundo lugar, los mecanismos de recuperación tradicionales suelen basarse en medidas de similitud en el espacio de embeddings, lo que resulta en la selección de documentos con alta similitud superficial pero baja diversidad semántica [2]. Esto afecta negativamente la cobertura de aspectos relevantes del corpus, comprometiendo la calidad de las respuestas generadas. Además, se suma el problema del sesgo de posición [69], una consecuencia de los mecanismos de atención que favorecen la información localizada al inicio del contexto, reduciendo la equidad en la integración de evidencia y afectando a la precisión en las respuestas. Estas limitaciones son especialmente críticas en dominios especializados como la medicina o el derecho, donde se exige precisión contextual, razonamiento multihop y cobertura informativa amplia.

1.2. Contribuciones

Para abordar las limitaciones de los sistemas de Generación por Recuperación Aumentada (RAG) (como la latencia, el costo computacional y la baja diversidad semántica en la recuperación), esta tesis introduce una nueva metodología centrada en la generación automática de preguntas. Proponemos *Knowledge-graph Representative Automatic Questions* (KRAQ), una herramienta diseñada para construir un conjunto de preguntas que

capten la estructura semántica profunda del corpus. La hipótesis es que este conjunto de preguntas, cuidadosamente generado a partir de comunidades semánticas identificadas en un grafo de conocimiento, actúa como un activo valioso. Específicamente, estas preguntas pueden enriquecer la fase de recuperación, mejorando la diversidad y precisión, y optimizar la eficiencia computacional en el pipeline de ciertos sistemas RAG. Estas preguntas actúan como puentes semánticos para enriquecer la recuperación y, a la vez, como proxies precalculados que optimizan la eficiencia del pipeline.

A diferencia de enfoques previos de generación de preguntas en RAG que construyen las preguntas directamente a partir de documentos individuales [25], esta tesis propone un enfoque estructurado que modela el contenido del corpus como un grafo de conocimiento. En esta representación, las entidades y relaciones extraídas del texto se conectan explícitamente, lo que permite capturar vínculos profundos entre los conceptos presentes en distintos documentos. A continuación, se aplica un algoritmo de clustering [72] para identificar comunidades semánticas de nodos, a partir de las cuales se generan resúmenes representativos en lenguaje natural. Sobre estos resúmenes se aplica un modelo de lenguaje *fine-tuneado*, capaz de producir preguntas que condensan las relaciones latentes y significativas del corpus.

La herramienta se implementa utilizando el framework GraphRAG [19], que permite construir el grafo de conocimiento a partir del corpus, identificar comunidades semánticas y generar resúmenes textuales representativos. Además, se introduce un benchmark específico para evaluar la relevancia del conjunto de preguntas generadas.

Adicionalmente, para demostrar la eficacia de KRAQ, esta tesis presenta dos aplicaciones concisas (que sirven como métricas indirectas del método) orientadas a mitigar limitaciones críticas en los sistemas RAG actuales. La primera consiste en la creación de un algoritmo de recuperación combinada, donde las preguntas generadas se utilizan para enriquecer el conjunto de documentos recuperados, complementando las búsquedas realizadas con la pregunta original del usuario. Esta estrategia se basa en la hipótesis de que los documentos recuperados utilizando preguntas similares a la pregunta original obtenidas con KRAQ cubren una mayor diversidad informativa, disminuyendo la redundancia contextual observada en los métodos estándar de recuperación. [2].

La segunda aplicación apunta a mejorar la eficiencia computacional de Speculative RAG [78], empleando las preguntas generadas por KRAQ para realizar un pre-cómputo de los embeddings necesarios para crear los subconjuntos de documentos que luego serán enviados a los RAG Drafters.

Para validar la efectividad de nuestro enfoque, se realizaron experimentos extensivos sobre cuatro benchmarks estándar ampliamente utilizados en la comunidad: TriviaQA, BioASQ, PubHealth y HotPotQA. Los resultados muestran mejoras consistentes. En la evaluación directa de la calidad de las preguntas, KRAQ superó a los baselines de referencia por hasta 49 puntos porcentuales en métricas de relevancia semántica. Adicionalmente, su aplicación práctica demostró ser beneficiosa: se observó un incremento de hasta un 3 % en la precisión del RAG tradicional usando la técnica de recuperación combinada, y una reducción de hasta el 11.8 % en la latencia del Speculative RAG. Estos resultados validan empíricamente la cobertura semántica de las preguntas generadas por KRAQ.

Así, esta tesis realiza una doble contribución: por un lado, a nivel teórico, propone una arquitectura novedosa para la generación representativa de preguntas mediante grafos de conocimiento, y por otro, a nivel experimental, demuestra su eficacia en mejorar el desempeño de sistemas RAG existentes bajo distintos escenarios de evaluación. En conjunto,

estos aportes abren una vía prometedora hacia la construcción de sistemas de recuperación más robustos y escalables, particularmente valiosos en dominios donde la precisión contextual y la eficiencia computacional son críticas.

1.3. Estructura de la tesis

La presente tesis se organiza en los siguientes capítulos para desarrollar de manera progresiva la problemática, las soluciones propuestas y su validación:

- **Capítulo 2 (Marco teórico y trabajos relacionados):** Proporciona los fundamentos conceptuales necesarios para comprender las metodologías desarrolladas. Se abordan los principios de los modelos de lenguaje modernos, incluyendo la arquitectura Transformer y los embeddings. Se profundiza en los sistemas RAG, sus componentes, ventajas y desafíos, así como en la optimizacion de Speculative Rag. Se revisan trabajos previos en generación automática de preguntas y la representación estructurada del conocimiento mediante grafos y análisis comunitario. Finalmente, se analizan trabajos similares a las propuestas de esta tesis, estableciendo un diálogo con el estado del arte.

- **Capítulo 3 (KRAQ):** Introduce el núcleo de la contribución de esta tesis: el sistema KRAQ. Se detalla exhaustivamente su metodología, desde la extracción de conocimiento y construcción de un grafo, pasando por la detección de comunidades semánticas y la síntesis de resúmenes, hasta la generación de preguntas representativas mediante un modelo de lenguaje *fine-tuneado*. Posteriormente, se presenta la experimentación y los resultados de KRAQ, incluyendo los datasets utilizados, las decisiones generales de implementación, el diseño de evaluación específico, y un análisis de su rendimiento en la generación de preguntas de alta calidad semántica.

- **Capítulo 4 (Combined Retrieve RAG):** Explora la primera aplicación práctica de las preguntas generadas por KRAQ. Se presenta la metodología de *Combined Retrieve RAG*, un algoritmo diseñado para enriquecer la diversidad de los documentos recuperados y mejorar la precisión de los sistemas RAG. A continuación, se detallan los experimentos realizados para evaluar esta propuesta, analizando su impacto en la calidad de las respuestas y presentando estudios de ablación.

- **Capítulo 5 (Efficient Speculative RAG):** Describe la segunda aplicación de KRAQ, orientada a mejorar la eficiencia computacional del *framework* Speculative RAG. Se introduce la metodología de *Efficient Speculative RAG*, que utiliza las preguntas de KRAQ para permitir el pre-cómputo de embeddings instruidos. La sección de experimentación evalúa la reducción de latencia obtenida y la preservación de la calidad de respuesta, complementada con estudios de ablación.

- **Capítulo 6 (Conclusiones generales):** Sintetiza los hallazgos principales de la investigación, resume las contribuciones teóricas y prácticas, y discute las implicaciones de los resultados obtenidos. Finalmente, se proponen líneas de trabajo futuro que podrían expandir las ideas y metodologías presentadas.

Adicionalmente, la tesis incluye un Apéndice con los *prompts* detallados utilizados en las diversas etapas de generación y evaluación, así como la bibliografía consultada.

2. MARCO TEÓRICO

2.1. Fundamentos de modelos de lenguaje modernos

Los avances contemporáneos en el Procesamiento del Lenguaje Natural (PLN) se fundamentan en una rápida evolución de las arquitecturas de aprendizaje profundo. Esta sección traza el recorrido desde los modelos pioneros de redes neuronales hasta la llegada de la arquitectura Transformer, que sentó las bases para los potentes modelos de lenguaje que se utilizan en la actualidad.

2.1.1. De redes neuronales a la revolución transformer

Las Redes Neuronales Artificiales (ANNs, por sus siglas en inglés) constituyen la base computacional del aprendizaje profundo moderno. Estos modelos, inspirados vagamente en la estructura del cerebro humano, son capaces de aprender funciones de mapeo complejas directamente a partir de grandes volúmenes de datos. Mediante la optimización de sus parámetros internos a través de algoritmos como la retropropagación y el descenso por gradiente, las redes neuronales pueden identificar patrones intrincados y realizar tareas sofisticadas en diversos dominios, incluyendo el PLN [24].

Un hito fundamental en la evolución del PLN fue la introducción de la arquitectura Transformer por Vaswani et al. [75]. El Transformer se distingue por su innovador mecanismo de auto-atención (*self-attention*), que permite al modelo ponderar la importancia de diferentes tokens de la secuencia entre si. Esta capacidad para capturar dependencias a largo plazo y contextualizar la información de manera flexible, junto con su diseño inherentemente paralelizable, superó muchas de las limitaciones de arquitecturas secuenciales previas como las Redes Neuronales Recurrentes (RNNs). La arquitectura Transformer original comprende componentes de codificador (*encoder*), diseñados para procesar la secuencia de entrada y generar representaciones contextualizadas, y componentes de decodificador (*decoder*), orientados a generar una secuencia de salida, siendo ambos cruciales para tareas de secuencia a secuencia como la traducción automática.

El impacto de la arquitectura Transformer ha sido transformador, sentando las bases para el desarrollo de los actuales LLMs. Al escalar la profundidad y el ancho de los modelos Transformer y entrenarlos sobre corpus textuales masivos, los investigadores han logrado crear sistemas con una comprensión y capacidad de generación de lenguaje sin precedentes. Estos LLMs, que se explorarán con más detalle en secciones posteriores, han redefinido el estado del arte en una multitud de tareas de PLN y son un componente central de las metodologías investigadas en esta tesis.

2.1.2. Embeddings: representando el significado

En el contexto del aprendizaje automático y, de manera crucial, en el PLN, los *embeddings* constituyen una técnica fundamental para representar datos categóricos discretos en espacios vectoriales continuos y de menor dimensión. Antes de que el texto pueda ser procesado por modelos de aprendizaje automático, este debe ser segmentado en unidades más pequeñas llamadas **tokens**. Un token puede ser una palabra, una sub-palabra (e.g.,

"embedding" podría dividirse en tokens como "embed" y "ding") o incluso un carácter individual, dependiendo del algoritmo de tokenización utilizado. Una vez fragmentado el texto, los *embeddings* se utilizan para mapear estos tokens (o secuencias de tokens como frases y documentos) a representaciones vectoriales densas. Estas representaciones permiten capturar similitudes y relaciones semánticas entre las unidades del lenguaje, habilitando su procesamiento eficiente por redes neuronales y otros algoritmos de aprendizaje [3, 50].

Representación vectorial densa

Tradicionalmente, las palabras eran representadas mediante codificaciones locales como el *one-hot encoding*. En esta representación, cada palabra de un vocabulario $\mathcal{V}$ se mapea a un vector binario de dimensión $|\mathcal{V}|$ (donde $|\mathcal{V}|$ es el tamaño total del vocabulario), en el cual solo una entrada es igual a uno y todas las demás son cero. Si bien es simple, esta representación presenta dos grandes desventajas: primero, genera vectores de muy alta dimensión y extremadamente dispersos (es decir, con una abrumadora mayoría de ceros), lo que es ineficiente computacionalmente; segundo, y más importante, carece de información semántica intrínseca, ya que los vectores de palabras diferentes son ortogonales entre sí, sin reflejar ningún tipo de similitud o relación (e.g., los vectores de "rey" y "reina" serían tan distintos como los de "rey" y "manzana").

Los *embeddings* superan estas limitaciones proyectando las palabras (u otras unidades textuales) en un espacio vectorial denso de una dimensión d significativamente menor que $|\mathcal{V}|$. Un vector se considera denso porque la mayoría de sus elementos son valores de punto flotante distintos de cero, cada uno contribuyendo a la representación del significado. Formalmente, una función de *embedding* es un mapeo:

$$\text{Embedding} : \mathcal{V} \to \mathbb{R}^d$$

donde $d \ll |\mathcal{V}|$. En este espacio de *embedding*, la proximidad geométrica (medida, por ejemplo, por la distancia euclidiana o la similitud coseno) entre vectores busca reflejar relaciones semánticas o contextuales entre las palabras correspondientes. Una propiedad deseable y a menudo observada en buenos espacios de *embedding* es la capacidad de capturar analogías mediante aritmética vectorial, como la famosa relación:

$$\text{vec(rey)} - \text{vec(hombre)} + \text{vec(mujer)} \approx \text{vec(reina)}$$

Este tipo de relaciones vectoriales fue popularizado por modelos pioneros como Word2Vec [50], que utiliza tareas de predicción contextual (como Skip-Gram, que predice palabras de contexto dada una palabra central, o CBOW, Continuous Bag-of-Words, que predice una palabra central a partir de su contexto) para aprender representaciones que preservan la estructura semántica y sintáctica del lenguaje a partir de grandes corpus textuales.

Medición de similitud: Similitud Coseno

Una vez que las unidades textuales (palabras, frases, documentos) han sido representadas como vectores en un espacio de *embedding*, es fundamental poder cuantificar su similitud o diferencia. Una de las métricas más utilizadas para este propósito es la **similitud coseno**.

Dados dos vectores de *embedding* no nulos, $\vec{A}$ y $\vec{B}$, la similitud coseno mide el coseno del ángulo entre ellos. Esta métrica evalúa la orientación de los vectores, independientemente

de su magnitud. Su valor varía entre -1 (vectores exactamente opuestos) y 1 (vectores con la misma orientación), donde 0 indica ortogonalidad (sin similitud de orientación). Un valor más cercano a 1 indica una mayor similitud semántica.

La fórmula para la similitud coseno entre dos vectores $\vec{A}$ y $\vec{B}$ de n dimensiones es:

$$\cos(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.1}$$

donde $\vec{A} \cdot \vec{B}$ es el producto punto de los vectores $\vec{A}$ y $\vec{B}$, y $\|\vec{A}\|$ y $\|\vec{B}\|$ son sus respectivas magnitudes euclidianas (normas L2).

En el contexto del PLN, la similitud coseno es ampliamente utilizada para:

- Encontrar las palabras más similares a una palabra dada.
- Comparar la similitud semántica entre frases o documentos.
- En sistemas de recuperación de información (como los RAG), para encontrar los documentos o pasajes más relevantes para una consulta del usuario, comparando el *embedding* de la consulta con los *embeddings* de los documentos indexados.

Su popularidad se debe a su eficacia para capturar la similitud semántica y a su relativa insensibilidad a la longitud de los documentos (ya que la magnitud de los vectores se normaliza).

Tipos de embeddings: estáticos vs. contextuales

Existen dos enfoques principales para obtener y utilizar *embeddings*:

Embeddings estáticos (o preentrenados): Modelos como Word2Vec [50], GloVe (Global Vectors for Word Representation) [60], y FastText [6] aprenden representaciones vectoriales fijas para cada palabra del vocabulario sobre grandes corpus textuales. En estos modelos, cada palabra tiene un único vector asociado, independientemente del contexto específico en el que aparezca. FastText, además, aprende *embeddings* para n-gramas de caracteres, lo que le permite generar vectores para palabras fuera del vocabulario (OOV) y capturar mejor información morfológica.

Embeddings contextuales (o dinámicos): Con la llegada de arquitecturas más profundas y contextuales, como las basadas en Transformers, surgieron modelos capaces de generar *embeddings* que varían según el contexto en el que aparece una palabra. ELMo (Embeddings from Language Models) [61] fue uno de los pioneros, utilizando LSTMs bidireccionales. Posteriormente, modelos como BERT (Bidirectional Encoder Representations from Transformers) [15] y los propios LLMs generan *embeddings* profundamente contextuales para cada token en una secuencia, lo que permite resolver ambigüedades léxicas (e.g., la palabra ”banco” tendrá diferentes *embeddings* en ”banco de peces” vs. ”banco financiero”).

Rol crítico de los embeddings en LLMs y RAG

En los LLMs, los *embeddings* son un componente crítico. Constituyen la primera capa del modelo, transformando la secuencia de tokens de entrada en representaciones numéricas

que el resto de la red puede procesar. Además, durante el preentrenamiento y el *fine-tuning*, estos *embeddings* se ajustan para capturar las complejas relaciones semánticas y sintácticas presentes en los datos.

En los sistemas RAG, los *embeddings* desempeñan un papel doblemente esencial:

1. **Para la indexación del corpus:** Los documentos o *chunks* de la base de conocimiento externa se convierten en vectores de *embedding* y se almacenan en una base de datos vectorial, creando un índice semántico.

2. **Para la recuperación en tiempo de consulta:** Cuando un usuario formula una pregunta, esta también se convierte en un vector de *embedding* utilizando el mismo modelo. Luego, se compara este *embedding* de consulta con los *embeddings* de los documentos indexados (usualmente mediante similitud coseno) para identificar y recuperar los fragmentos de texto más relevantes que servirán de contexto al LLM para generar la respuesta.

La calidad de los *embeddings* y la efectividad de la métrica de similitud son, por lo tanto, cruciales para el rendimiento de los sistemas RAG, ya que determinan la relevancia del contexto proporcionado al LLM.

2.1.3. Modelos de lenguaje de gran escala (LLMs)

Los Modelos de Lenguaje de Gran Escala (LLMs) representan un avance significativo en el Procesamiento del Lenguaje Natural como resultado de escalar la arquitectura Transformer (predominantemente las variantes *decoder-only* como LLaMA [70] o arquitecturas *encoder-decoder*) a un número masivo de parámetros, desde miles de millones hasta billones y entrenarlos en cantidades masivas de datos textuales, a menudo extraídos de la web [8, 11].

Principios arquitectónicos y preentrenamiento El objetivo de preentrenamiento más común para los LLMs generativos (típicamente *decoder-only*) es la predicción del siguiente token: dado un fragmento de texto, el modelo aprende a predecir el token más probable que sigue en la secuencia. Es crucial destacar que, en lugar de predecir un único token determinista, el LLM en realidad genera una distribución de probabilidad sobre todo el vocabulario de posibles tokens para la siguiente posición. Durante la generación de texto (inferencia), en lugar de elegir siempre el token más probable, se utiliza un proceso de muestreo (*sampling*) para seleccionar el siguiente token a partir de esta distribución de probabilidad. Este proceso introduce un grado de aleatoriedad controlada, lo que permite generar un texto más variado y natural. Las estrategias de muestreo van desde la selección determinista del token más probable (*greedy decoding*) hasta técnicas más sofisticadas que consideran un subconjunto de los tokens más plausibles (e.g., *top-k*, *nucleus sampling*) para aumentar la creatividad de la respuesta. A través de este simple pero potente objetivo auto-supervisado, y gracias a la escala sin precedentes de los datos y del propio modelo, los LLMs desarrollan una comprensión sorprendentemente profunda de la sintaxis, la semántica, el conocimiento del mundo incorporado en los textos de entrenamiento, y ciertas capacidades de razonamiento y abstracción [7].

Capacidades emergentes Uno de los hallazgos más notables en el desarrollo de LLMs es el fenómeno de las **capacidades emergentes** [80]. Se trata de habilidades complejas que no están presentes (o lo están de forma muy rudimentaria) en modelos más pequeños

de la misma familia arquitectónica, pero que emergen de manera no lineal una vez que el tamaño del modelo (número de parámetros) y/o la cantidad de datos de entrenamiento superan ciertos umbrales críticos.

Ejemplos de capacidades emergentes incluyen:

- **Aprendizaje en pocos ejemplos (*Few-Shot Learning*):** La capacidad de realizar nuevas tareas con un rendimiento razonable después de ver solo unas pocas demostraciones (ejemplos de entrada-salida de la tarea) proporcionadas en el *prompt* de entrada, sin necesidad de reentrenar o ajustar los pesos del modelo. Brown et al. [8] demostraron esto extensamente con GPT-3.
- **Razonamiento aritmético y simbólico:** Habilidad para resolver problemas matemáticos simples o seguir cadenas de razonamiento lógico.
- **Generación de código:** Capacidad para escribir código funcional en diversos lenguajes de programación a partir de descripciones en lenguaje natural.
- **Comprensión avanzada de instrucciones:** Modelos como GPT-4 [55] muestran una habilidad sofisticada para seguir instrucciones complejas y matizadas en lenguaje natural, incluso para tareas para las cuales no fueron explícitamente entrenados.

Limitaciones fundamentales de los LLMs

A pesar de los avances extraordinarios y las capacidades impresionantes demostradas por los LLMs contemporáneos, como GPT-4 [55], PaLM [11] y LLaMA [70], estos sistemas presentan una serie de limitaciones estructurales, operativas y conceptuales. Estas restricciones pueden mermar su aplicabilidad y fiabilidad en escenarios que demandan alta precisión factual, conocimiento actualizado, interpretabilidad o razonamiento complejo. Comprender estas limitaciones es crucial para el desarrollo de aplicaciones robustas y para guiar la investigación futura [34].

Conocimiento estático y desactualizado. Una de las limitaciones más fundamentales de los LLMs preentrenados radica en la naturaleza estática de su conocimiento. Estos modelos aprenden a partir de un corpus de datos masivo que, una vez completado el preentrenamiento, permanece fijo. Esto implica que:

- **Desactualización temporal:** El conocimiento del LLM está intrínsecamente ligado al corte temporal de sus datos de entrenamiento. No pueden incorporar información sobre eventos, descubrimientos o desarrollos ocurridos posteriormente sin un reentrenamiento o mecanismos de actualización, lo cual es costoso.
- **Ausencia de conocimiento específico o confidencial:** Por diseño, no tienen acceso a información específica de un dominio particular que no estuviera presente en su corpus de preentrenamiento público, ni a datos privados o confidenciales de un usuario u organización.

Esta amnesia respecto a la información nueva o no vista durante el preentrenamiento ha sido una de las principales motivaciones para el desarrollo de arquitecturas híbridas como RAG, que buscan complementar el conocimiento paramétrico del LLM con información obtenida en tiempo real de fuentes externas [40].

Alucinaciones y fiabilidad factual. Los LLMs, a pesar de su fluidez y coherencia aparente, pueden generar respuestas que son plausibles y gramaticalmente correctas, pero fácticamente incorrectas, inconsistentes con el contexto proporcionado, o incluso inventadas. Este fenómeno se conoce comúnmente como alucinación (*hallucination*) [31]. Las alucinaciones pueden surgir por diversas razones, incluyendo:

- Errores o sesgos en los datos de entrenamiento.
- La naturaleza probabilística de la generación de texto, que optimiza la verosimilitud de la secuencia en lugar de la veracidad factual.
- Dificultad para distinguir entre información memorizada y conocimiento inferido.

La propensión a las alucinaciones dificulta enormemente el uso de LLMs en dominios donde la precisión y la veracidad son críticas (e.g., medicina, finanzas, derecho), y subraya la necesidad de mecanismos de verificación y validación.

Opacidad e interpretabilidad. Dada su vasta escala (miles de millones de parámetros) y la complejidad de sus arquitecturas internas, los LLMs operan en gran medida como cajas negras. Resulta extremadamente difícil:

- **Determinar el conocimiento específico:** Precisar qué información exacta conoce el modelo, cómo está representada internamente, y cuáles son las fuentes originales de ese conocimiento.
- **Auditar el proceso de generación:** Entender por qué el modelo genera una respuesta particular en lugar de otra, o trazar el razonamiento (si lo hubiera) que condujo a una conclusión específica.
- **Identificar y Corregir Errores Sistemáticos:** La opacidad dificulta el diagnóstico y la corrección de sesgos o patrones de error recurrentes.

Esta falta de interpretabilidad y auditabilidad es una preocupación central, especialmente en aplicaciones sensibles, y es un área activa de investigación [7].

Costos computacionales y sostenibilidad. El entrenamiento y, en menor medida, la inferencia de los LLMs más grandes requieren recursos computacionales masivos:

- **Entrenamiento:** Modelos como PaLM y GPT-4 necesitan clústeres de miles de GPUs de alto rendimiento (o TPUs equivalentes) y semanas o meses de entrenamiento, utilizando técnicas de paralelización de datos y modelos complejas como FSDP (Fully Sharded Data Parallel) o ZeRO (Zero Redundancy Optimizer) [11, 68]. Esto implica un costo económico y energético considerable.
- **Inferencia:** Aunque menos intensiva que el entrenamiento, la ejecución de LLMs grandes para generar respuestas también demanda hardware especializado y optimizaciones para lograr latencias aceptables, especialmente a escala.

Estos altos costos limitan el acceso a la investigación y desarrollo de LLMs de vanguardia a un pequeño número de grandes corporaciones y consorcios, planteando cuestiones sobre la democratización de la IA y la sostenibilidad ambiental.

Limitaciones en razonamiento complejo. Si bien los LLMs pueden exhibir capacidades de razonamiento sorprendentes en ciertas tareas, especialmente con técnicas de *prompting* como *chain-of-thought* [62], todavía muestran debilidades en formas de razonamiento más complejas o estructuradas:

- **Razonamiento multi-salto (*Multi-hop Reasoning*):** Dificultad para integrar consistentemente información proveniente de múltiples fragmentos de texto o realizar múltiples pasos inferenciales para llegar a una conclusión.
- **Razonamiento simbólico y numérico:** Limitaciones en la manipulación precisa de símbolos, la realización de operaciones aritméticas exactas (especialmente con números grandes o múltiples pasos), y el seguimiento de reglas lógicas formales.
- **Razonamiento de sentido común robusto:** Aunque han mejorado, los LLMs aún pueden fallar en tareas que requieren una comprensión profunda del sentido común del mundo físico o social.

Estas limitaciones indican que, si bien los LLMs son excelentes modelos de lenguaje, su capacidad para un razonamiento robusto y generalizable sigue siendo un área de desarrollo activo [62].

Sesgo de posición y sensibilidad al orden del contexto. Incluso cuando se les proporciona información relevante, los LLMs pueden no utilizarla de manera uniforme o equitativa. Se ha observado que muchos modelos exhiben un sesgo de posición (*positional bias*), lo que significa que la ubicación de la información dentro del *prompt* de entrada (el contexto) puede influir desproporcionadamente en la respuesta generada [69]. Este fenómeno se manifiesta de varias maneras:

- **Preferencia por información al inicio o al final:** A menudo, la información presentada al principio o al final del contexto tiene una mayor probabilidad de ser utilizada o recordada por el modelo en su respuesta, mientras que la información en el medio del contexto ("lost in the middle") puede ser ignorada o subutilizada, incluso si es crucial [45].
- **Impacto en tareas de múltiples documentos:** En sistemas RAG, donde se proporcionan múltiples documentos recuperados como contexto, el orden en que se presentan estos documentos puede afectar significativamente la respuesta final.
- **Inconsistencias en las respuestas:** El mismo conjunto de información, presentado en diferente orden, podría llevar a respuestas diferentes.

Fine-tuning de LLMs para adaptación y especialización

El *fine-tuning* (ajuste fino) es una técnica fundamental en el ciclo de vida de los LLMs. Consiste en adaptar un modelo preentrenado sobre vastos corpus de datos genéricos a una tarea específica o a un dominio particular mediante un entrenamiento adicional sobre un conjunto de datos más reducido y especializado para dicha tarea. Este procedimiento se enmarca en el paradigma del aprendizaje por transferencia (*transfer learning*), permitiendo aprovechar el conocimiento general del lenguaje, la sintaxis, la semántica y cierto grado

de conocimiento del mundo que el modelo adquirió durante su costosa fase de preentrenamiento [15, 64].

Formulación general. Dado un modelo de lenguaje preentrenado cuyos parámetros se denotan por θ_{pre}, el proceso de *fine-tuning* busca encontrar una nueva configuración de parámetros θ' que minimice una función de pérdida específica para la tarea objetivo, $\mathcal{L}_{\text{task}}$. Formalmente, el objetivo es:

$$\theta' = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{task}}} \left[\mathcal{L}_{\text{task}}(f(x;\theta), y)\right]$$

donde $\mathcal{D}_{\text{task}}$ es el *dataset* específico de la tarea y $f(x;\theta)$ es la salida del modelo.

Estrategias principales de Fine-tuning. Existen diversas estrategias para realizar el fine-tuning de LLMs:

Full Fine-tuning. Consiste en actualizar todos los parámetros del modelo preentrenado. Aunque teóricamente ofrece la máxima flexibilidad, es computacionalmente muy costoso y puede ser propenso al sobreajuste en *datasets* pequeños.

Instruction Tuning. Es una forma de *fine-tuning* supervisado donde el modelo se entrena con ejemplos formulados como instrucciones en lenguaje natural [56]. El modelo aprende a generar respuestas apropiadas condicionadas a una amplia variedad de *prompts* que describen la tarea (e.g., "Resume el siguiente texto:", "¿Cuál es la capital de Y?"). Este enfoque mejora la capacidad de los LLMs para seguir instrucciones y generalizar a tareas no vistas. En esta tesis, el modelo generador de preguntas de KRAQ se beneficia de esta técnica (ver Sección 3.1.5).

Parameter-Efficient Fine-tuning (PEFT). Dada la carga computacional del *full fine-tuning*, las técnicas de PEFT buscan adaptar los LLMs modificando solo una pequeña fracción de los parámetros o añadiendo un número reducido de parámetros entrenables.

LoRA (Low-Rank Adaptation). Propuesta por Hu et al. [27], LoRA mantiene congeladas las matrices de pesos originales W de ciertas capas e introduce dos matrices pequeñas, A y B, cuyo producto BA representa la actualización $\Delta W = BA$. Solo se entrenan A y B. El rango r de estas matrices es mucho menor que las dimensiones originales, reduciendo drásticamente los parámetros entrenables y los requisitos de memoria, logrando un rendimiento comparable al *full fine-tuning* en muchas tareas. La Ecuación 2.2 describe esta actualización.

$$W' = W + \Delta W = W + BA \tag{2.2}$$

QLoRA (Quantized LoRA). Introducida por Dettmers et al. [14], QLoRA optimiza LoRA para mayor eficiencia en memoria. Permite el *fine-tuning* de LLMs muy grandes en hardware con VRAM limitada. Combina:

- **Cuantización del Modelo Base a 4 bits:** Los pesos del modelo preentrenado se cuantizan a 4 bits usando el formato **NF4 (NormalFloat4)**, optimizado para distribuciones de pesos neuronales, reduciendo drásticamente la huella de memoria.

- **Doble Cuantización:** Se aplica una segunda cuantización a las constantes de cuantización de la primera etapa.
- **Aplicación de LoRA:** Las matrices de adaptación LoRA (A y B) se mantienen en mayor precisión (e.g., BFloat16) y se aplican sobre el modelo base cuantizado. Solo los parámetros de LoRA se actualizan.
- **Optimizadores Paginados (*Paged Optimizers*):** Para manejar picos de memoria.

QLoRA ha demostrado ser muy efectiva para el *fine-tuning* de LLMs con decenas de miles de millones de parámetros en GPUs individuales. En esta tesis, tanto el modelo generador de preguntas de KRAQ como el modelo M_{Drafter} (en una fase exploratoria) fueron ajustados utilizando QLoRA (ver Secciones 3.2.5 y 5.2.5).

2.2. Generación aumentada por recuperación (RAG)

Los sistemas de Generación por Recuperación Aumentada (Retrieval-Augmented Generation, RAG) han emergido como una arquitectura efectiva para mitigar algunas de las limitaciones inherentes a los LLMs, en particular su dependencia de un conocimiento paramétrico estático y su incapacidad para acceder a información externa o actualizada posterior a su fase de preentrenamiento [40]. Al integrar un componente de recuperación de información con un modelo generativo potente, los sistemas RAG pueden producir respuestas más precisas, factuales y contextualmente relevantes, basándose en evidencia obtenida en tiempo real de una base de conocimiento externa.

2.2.1. Principios y componentes de los sistemas RAG

En tareas que requieren un conocimiento intensivo, donde la respuesta no puede derivarse únicamente del conocimiento general del LLM, una instancia de RAG puede entenderse como el procesamiento de una consulta Q del usuario, utilizando un conjunto de documentos de evidencia $\mathcal{D} = \{d_1, d_2, \ldots, d_k\}$ recuperados de un corpus externo E, para generar una respuesta A. El objetivo es que la respuesta generada $\hat{A}$ sea coherente, precisa y esté fundamentada en la evidencia $\mathcal{D}$.

Desde una perspectiva computacional, un sistema RAG estándar consta de dos componentes principales que operan secuencialmente:

1. **Recuperador (*Retriever*, R):** Dada una consulta del usuario Q, este módulo es responsable de buscar y seleccionar dentro de un corpus de documentos E un subconjunto $\mathcal{D}$ de k documentos o fragmentos de texto (*chunks*) que se consideran los más relevantes para responder a Q. Formalmente, $\mathcal{D} = R(Q, E, k)$.
2. **Generador (*Generator*, G):** Este módulo, típicamente un LLM, recibe la consulta original Q y el conjunto de documentos recuperados $\mathcal{D}$ como contexto adicional. Utiliza esta información combinada para generar la respuesta final $\hat{A}$. Formalmente, $\hat{A} = G(Q, \mathcal{D})$.

La formulación general del proceso RAG puede, por lo tanto, expresarse como:

$$\hat{A} = G(Q, R(Q, E, k))$$

Durante la evaluación de un sistema RAG (si se realiza de forma *end-to-end* o si se ajusta el generador), se busca minimizar una función de pérdida $\mathcal{L}(\hat{A}, A_{\text{ref}})$ que mida la discrepancia entre la respuesta generada $\hat{A}$ y una respuesta de referencia A_{ref} [29].

Se explicarán las evaluaciones usadas en esta tesis en la Sección 4.2.1

El proceso de recuperación detallado

El componente de recuperación es crucial para el éxito de un sistema RAG. En la mayoría de las implementaciones modernas, y consistentemente en esta tesis, la recuperación se basa en la similitud semántica en un espacio de *embeddings*:

1. **Indexación del corpus (Offline):**
   - El corpus documental E se divide primero en fragmentos manejables (*chunks*), c_i.
   - Cada *chunk* c_i se codifica en un vector de *embedding*, $\vec{e}_i = \text{Emb}(c_i)$, utilizando un modelo de *embeddings* preentrenado.
   - Estos vectores $\vec{e}_i$, junto con sus *chunks* originales c_i (o referencias a ellos), se almacenan en una base de datos vectorial especializada, que permite búsquedas eficientes por similitud.
2. **Recuperación en yiempo de consulta (Online):**
   - La consulta del usuario Q se codifica en un vector de *embedding* $\vec{q} = \text{Emb}(Q)$ utilizando el mismo modelo de *embeddings* que se usó para indexar el corpus.
   - Se calcula la similitud entre el vector de consulta $\vec{q}$ y todos los vectores de *chunk* $\vec{e}_i$ en la base de datos indexada. La métrica de similitud más comúnmente empleada es la **similitud coseno** (explicada en detalle en la Sección 2.1.2, Ecuación 2.1).
   - Los k *chunks* c_j cuyos vectores $\vec{e}_j$ presentan la mayor similitud coseno con $\vec{q}$ se seleccionan como el conjunto de documentos relevantes $\mathcal{D} = \{d_1, \ldots, d_k\}$.

La calidad de los *embeddings* y la elección del umbral k son hiperparámetros críticos que impactan directamente la relevancia del contexto proporcionado al generador.

Pseudocódigo de un RAG Tradicional

El flujo de trabajo de un sistema RAG tradicional puede resumirse en el Algoritmo 1.

Algorithm 1 Algoritmo de RAG Tradicional (con Búsqueda Vectorial Eficiente)

Require: Consulta del usuario Q, Base de Datos Vectorial $\mathcal{DB}_{\text{vectorial}}$ (contiene embeddings y documentos/chunks), Modelo de Embedding $\text{Emb}(\cdot)$, Número de documentos a recuperar k, Modelo Generador LLM $G(\cdot, \cdot)$

Ensure: Respuesta generada $\hat{A}$

$\vec{q} \leftarrow \text{Emb}(Q)$ ▷ Codificar la consulta del usuario en un embedding

$\mathcal{D} \leftarrow \mathcal{DB}_{\text{vectorial}}.\text{Search}(\vec{q}, k)$ ▷ Recupera k chunks relevantes a q

$\textit{contexto} \leftarrow \text{ConcatenateDocuments}(\mathcal{D})$ ▷ Formar el contexto para el LLM

$\hat{A} \leftarrow G(Q, \textit{contexto})$ ▷ Generar respuesta con el LLM y el contexto

return $\hat{A}$

Nótese que, en la práctica, las bases de datos vectoriales realizan el paso de cálculo de similitud y ordenamiento de manera mucho más eficiente utilizando estructuras de datos como HNSW [28] o IVFADC, en lugar de una búsqueda exhaustiva.

2.2.2. Ventajas y desafíos de RAG

Ventajas de los sistemas RAG

Los sistemas RAG presentan múltiples beneficios que los posicionan como una alternativa robusta a los LLMs autónomos:

- **Conocimiento actualizable y específico:** Al recuperar documentos de un corpus externo, los RAG pueden incorporar información reciente o específica de un dominio sin necesidad de reentrenar costosamente el LLM base. El corpus puede actualizarse independientemente.
- **Reducción de alucinaciones:** Al condicionar la generación en evidencia recuperada, se reduce la tendencia de los LLMs a inventar información, aumentando la factualidad de las respuestas.
- **Explicabilidad y trazabilidad:** Dado que las respuestas están (idealmente) fundamentadas en los documentos recuperados, es posible citar las fuentes, lo que aumenta la transparencia y permite al usuario verificar la información.
- **Control sobre la información:** Permite restringir las respuestas a un corpus de conocimiento específico y confiable, lo cual es crucial en entornos empresariales o dominios sensibles.
- **Potencial de personalización:** El corpus de recuperación puede adaptarse a usuarios o contextos específicos.

Limitaciones y desafíos inherentes

No obstante, los RAG también presentan desafíos importantes que deben ser considerados:

- **Dependencia de la calidad de recuperación (*Retrieval Quality*):** Si el recuperador falla en encontrar los documentos verdaderamente relevantes, o si recupera información ruidosa, contradictoria o sesgada, la calidad de la respuesta generada por el LLM se verá comprometida, incluso si el LLM es muy potente [2].
- **Sensibilidad al número y orden de documentos:** La elección de cuántos documentos recuperar (k) es crítica. Demasiados documentos pueden abrumar al LLM o exceder su ventana de contexto; muy pocos pueden omitir información vital. Además, muchos LLMs exhiben un sesgo de posición, dando más importancia a la información al inicio o al final del contexto [69].
- **Costo computacional en inferencia:** Aunque se evita el reentrenamiento, cada consulta requiere ejecutar el *pipeline* completo de recuperación (búsqueda en base de datos vectorial) y generación (inferencia del LLM), lo que puede introducir latencia y ser costoso a gran escala.

- **Desafíos en la síntesis de evidencia múltiple:** Integrar información de múltiples documentos recuperados de manera coherente, resolver contradicciones entre ellos, o asegurar que toda la evidencia relevante sea utilizada por el LLM, sigue siendo un desafío abierto intrínseco de los LLMS.

Estas limitaciones impulsan la investigación activa en la mejora de cada componente y en el desarrollo de estrategias más sofisticadas, algunas de las cuales se exploran en la presente tesis.

2.2.3. Speculative RAG

Para abordar las limitaciones explicadas en la seccionan anterior, una línea de investigación relevante se ha centrado en arquitecturas que mejoran la eficiencia y exploran una mayor diversidad semántica. Un ejemplo paradigmático de esta dirección es *Speculative RAG*, introducido por Wang et al. [78]. Este enfoque adapta la estrategia de *draft-then-verify*, popularizada por técnicas de *speculative decoding* en la generación de LLMs [82], al contexto específico de los sistemas RAG.

Principios fundamentales del paradigma Draft-then-Verify en RAG. En lugar de que un único LLM, usualmente grande y costoso, procese todo el contexto recuperado para generar la respuesta, Speculative RAG emplea un sistema de dos etapas. Primero, múltiples modelos "borrador" (M_{Drafter}), que pueden ser LLMs más pequeños y, por ende, más rápidos, generan en paralelo un conjunto de respuestas candidatas. Cada uno de estos modelos borradores opera sobre un subconjunto diferente de los documentos recuperados, donde cada subconjunto está diseñado para capturar distintas perspectivas respecto a la consulta. Posteriormente, un modelo "verificador" (M_{Verifier}), que puede ser un LLM más potente, evalúa estos borradores y selecciona el que se considera de mayor calidad como la respuesta final. Esta paralelización de la generación de borradores y la especialización de tareas pueden conducir a una reducción en la latencia y a una exploración más amplia del espacio de posibles respuestas.

Construcción de subconjuntos de documentos con embeddings instruidos. Un aspecto crucial de Speculative RAG es cómo se construyen los subconjuntos de documentos que se proporcionan a cada modelo borrador. Dado un conjunto de documentos $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$ recuperados inicialmente para una pregunta original Q, el sistema primero genera representaciones vectoriales (*embeddings*) de estos documentos que están explícitamente "instruidos" o condicionados por la pregunta Q. Para ello, se emplea un modelo de *embeddings* con un encoder especializado, como InBedder-RoBERTa [59], que produce una representación $\mathcal{E}(d_i \mid Q)$ para cada documento d_i. La instrucción proporcionada por la pregunta Q al *encoder* de *embeddings* tiene como objetivo mejorar la agrupación semántica de los documentos en relación con la consulta específica, un aspecto fundamental para la posterior generación de perspectivas diversas en los borradores. Una vez obtenidos estos *embeddings* instruidos, los documentos se agrupan en k clústeres, mediante el algoritmo K-Means [47]. A partir de estos clústers, que representan diferentes perspectivas temáticas de la información recuperada, se generan m subconjuntos de documentos $\delta_j \subset \mathcal{D}$. Cada subconjunto δ_j se construye muestreando un único documento de cada uno de los k clústeres. Esta estrategia de muestreo tiene un doble propósito: reducir la redundancia de información dentro de cada subconjunto y asegurar que cada uno de ellos cubra, idealmen-

te, todas las perspectivas identificadas frente a la pregunta del usuario.

Generación paralela de borradores y proceso de verificación. Cada subconjunto de documentos δ_j se proporciona como entrada a una instancia del modelo borrador M_{Drafter}, junto con la pregunta original Q. Para cada subconjunto, M_{Drafter} genera una respuesta tentativa α_j y su correspondiente justificación o razonamiento (racional) β_j, que explica cómo se llegó a α_j a partir de δ_j. Wang et al. [78] señalan que los modelos M_{Drafter} pueden ser *Fine-Tuneados* para generar esta dupla de respuesta y racional. Este proceso de generación de m pares (α_j, β_j) se puede realizar en paralelo, lo que constituye una de las fuentes de optimización de tiempo. Ver prompt exacto para la generación de borradores en el Apéndice 7.5

Posteriormente, el modelo generalista M_{Verifier}, que no requiere un *fine-tuning* específico para esta tarea de evaluación, procesa cada par (α_j, β_j). Para cada uno, se computan típicamente tres puntuaciones (*scores*) distintas que contribuyen a un *score* de confianza global ρ_j:

- ρ_j^{draft}: Mide la confianza o probabilidad con la que M_{Drafter} generó el par (α_j, β_j) dado Q y δ_j. Formalmente $P_{M_{\text{Drafter}}}(\beta_j \mid Q, \delta_j) + P_{M_{\text{Drafter}}}(\alpha_j \mid Q, \delta_j, \beta_j)$.

- $\rho_j^{\text{self-contain}}$: Evalúa la coherencia interna del par (α_j, β_j) generado por M_{Drafter} $P_{M_{\text{Drafter}}}(\alpha_j, \beta_j \mid Q, \delta_j)$.

- $\rho_j^{\text{self-reflect}}$: Captura la evaluación que hace M_{Verifier} sobre la calidad del racional β_j como soporte para la respuesta α_j. Esto se obtiene de la probabilidad con la que M_{Verifier} genera una afirmación positiva (e.g., "Sí") en respuesta a una meta-pregunta como: "¿Considera que la justificación β_j apoya adecuadamente la respuesta α_j?", condicionada por Q, α_j, y β_j. Ver prompt exacto en Apéndice 7.6.

El *score* final para cada borrador j se calcula como el producto de estas tres probabilidades:

$$\rho_j = \rho_j^{\text{draft}} \cdot \rho_j^{\text{self-contain}} \cdot \rho_j^{\text{self-reflect}}$$

La respuesta final $\hat{A}$ del sistema se selecciona como aquella respuesta borrador α_j que maximiza este *score* combinado ρ_j.

Pseudocódigo. El procedimiento completo de Speculative RAG se detalla en el Algoritmo 2

Algorithm 2 Algoritmo de Speculative RAG (Adaptado de Wang et al. [78])

Require: Pregunta Q, conjunto de documentos recuperados $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$
Require: Número de clústeres k, número de borradores m
Ensure: Respuesta predicha $\hat{A}$

function SPECULATIVERAG($Q, \mathcal{D}, k, m$)
  $E \leftarrow \{\mathcal{E}(d_i \mid Q)$ para cada $d_i \in \mathcal{D}\}$ ▷ Embeddings instruidos
  $\{C_1, \ldots, C_k\} \leftarrow \text{KMeans}(E, k)$ ▷ Clustering en k grupos
  $\Delta \leftarrow \emptyset$ ▷ Subconjuntos de documentos
  for $j = 1$ **to** m **do**
    $\delta_j \leftarrow \{\text{SampleOne}(C_l)$ para cada $l = 1 \ldots k\}$ ▷ Un doc por clúster
    $\Delta \leftarrow \Delta \cup \{\delta_j\}$
  $\mathcal{B} \leftarrow \emptyset$ ▷ Lista de borradores con scores
  for all $\delta_j \in \Delta$ **in parallel do**
    $(\alpha_j, \beta_j) \leftarrow M_{\text{Drafter}}(Q, \delta_j)$ ▷ Respuesta y racional
    $\rho_j^{\text{draft}} \leftarrow P(\beta_j \mid Q, \delta_j) + P(\alpha_j \mid Q, \delta_j)$
    $\rho_j^{\text{self-contain}} \leftarrow P(\alpha_j, \beta_j \mid Q, \delta_j)$
    $\rho_j^{\text{self-reflect}} \leftarrow P_{M_{\text{Verifier}}}(\text{“Yes”} \mid Q, \alpha_j, \beta_j)$
    $\rho_j \leftarrow \rho_j^{\text{draft}} \cdot \rho_j^{\text{self-contain}} \cdot \rho_j^{\text{self-reflect}}$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\alpha_j, \rho_j)\}$
  $(\hat{A}, \_) \leftarrow \arg\max_{(\alpha_j, \rho_j) \in \mathcal{B}} \rho_j$ ▷ Selección final
  return $\hat{A}$

Limitaciones computacionales de Speculative Rag

A pesar de sus beneficios en términos de eficiencia en la etapa de generación y la promoción de la diversidad semántica, Speculative RAG presenta una limitación computacional clave. Como se mencionó, el uso de *embeddings* instruidos por la pregunta Q (e.g., $\mathcal{E}(d_i \mid Q)$ con InBedder-RoBERTa) es fundamental para la calidad del *clustering* y, por ende, para la efectividad del muestreo de documentos y el razonamiento posterior. La importancia de este condicionamiento se demuestra empíricamente en Wang et al. [78, Sección 3.2].

El problema principal radica en que el cálculo de estos *embeddings* instruidos debe realizarse *en línea* (online), es decir, por cada nueva consulta Q que formula el usuario y para la totalidad de los documentos d_i recuperados para esa consulta. Esto se debe a que los *embeddings* son, por diseño, dependientes de Q. A diferencia de los sistemas de recuperación tradicionales donde los *embeddings* de los documentos se precomputan y se almacenan en un índice, aquí el condicionamiento por Q impide este pre-cómputo directo. En términos prácticos, esto significa que cada vez que un usuario formula una nueva pregunta, el sistema se ve obligado a ejecutar el costoso modelo de *embeddings* instruidos sobre todos los documentos recuperados. En escenarios con múltiples usuarios concurrentes o un alto volumen de consultas, y especialmente si el número de documentos recuperados N es grande, la carga computacional de esta etapa puede volverse prohibitiva y constituir el principal cuello de botella del *pipeline*. Este costo computacional afecta negativamente la aplicabilidad del método en contextos que demandan baja latencia, como asistentes conversacionales o sistemas de búsqueda interactiva. La propuesta de *Efficient Speculative RAG* en esta tesis (detallada en la Sección 5.1) busca mitigar precisamente este cuello de botella mediante la integración de las preguntas representativas generadas por KRAQ,

permitiendo un pre-cómputo efectivo de los *embeddings* instruidos.

2.3. Generación automática de preguntas

La Generación Automática de Preguntas (QG, por sus siglas en inglés) es una tarea del PLN que consiste en crear preguntas textuales a partir de diversas modalidades de entrada, tales como texto, datos estructurados o incluso imágenes, las cuales denotamos genéricamente como $\mathcal{X}$ [25]. Dado un input $\mathcal{X}$, y opcionalmente una respuesta específica A (ya que algunas tareas de QG son conscientes de la respuesta y otras no), el objetivo de la QG es aprender una función de mapeo f_θ que genere una pregunta textual Q. Formalmente, la tarea puede expresarse como:

$$f_\theta : (\mathcal{X}, A^*) \rightarrow Q \tag{2.3}$$

donde A^* indica que la respuesta es un input opcional. La pregunta generada $Q = \langle q_1, q_2, \ldots, q_n \rangle$ está compuesta por una secuencia de n tokens q_i, seleccionados de un vocabulario predefinido $\mathcal{V}$. En la práctica, la función f_θ se implementa comúnmente mediante arquitecturas de redes neuronales, tales como RNNs [18], modelos basados en la arquitectura Transformer o LLMs [17, 75]

2.3.1. Aplicaciones

La QG ha demostrado ser una herramienta versátil en múltiples áreas del procesamiento del lenguaje natural, extendiendo su utilidad desde sistemas tradicionales de pregunta-respuesta hasta entornos educativos y conversacionales.. A continuación, se describen en detalle las aplicaciones en las que QG ha generado impacto, culminando con la propuesta central de esta tesis: su incorporación estratégica en *pipelines* de RAG.

Datasets QA. Uno de los usos más consolidados de QG es en la generación de datos sintéticos para entrenar sistemas de QA. Dado que la creación manual de pares pregunta–respuesta es costosa y limitada en escala, la generación automática permite ampliar significativamente los *datasets* disponibles [18]. Esta estrategia permite, en la práctica, construir asistentes y *chatbots* más robustos y precisos, al nutrir los sistemas con datos de entrenamiento más diversos y abundantes.

Educación personalizada y tutoría inteligente. En el ámbito educativo, QG permite adaptar la generación de preguntas al contenido y nivel del estudiante. Los sistemas de tutoría inteligente (Intelligent Tutoring Systems, ITS) pueden utilizar QG para generar encuestas, ejercicios o evaluaciones alineadas con el progreso individual del alumno [54]. Por ejemplo, Bull2Sum [23] no solo produce preguntas relevantes para textos educativos, sino que también contribuye a la construcción de *datasets* pedagógicos significativos. Esto se traduce en experiencias de aprendizaje más dinámicas y personalizadas, con un impacto positivo en la comprensión y retención del contenido.

Sistemas conversacionales. Los asistentes virtuales, *bots* de atención al cliente y plataformas de diálogo interactivo se benefician de QG para mantener conversaciones fluidas y naturales al proponer siguientes preguntas posibles del usuario. La capacidad de generar preguntas de seguimiento pertinentes y contextualizadas mejora significativamente la

calidad de la interacción [16]. El trabajo de Pan et al. [57] sobre generación de preguntas conversacionales (Conversational Question Generation, CQG) resalta la importancia de incorporar el historial del diálogo y el contexto semántico en la generación de preguntas, haciendo que las respuestas y preguntas formuladas por el sistema sean más coherentes y humanas, y evitando así interacciones repetitivas o poco atractivas.

Optimización de sistemas RAG. Una de las aplicaciones clave es el uso de la generación de preguntas para optimizar sistemas RAG. El sistema KRAQ, propuesto en este trabajo, produce preguntas representativas que condensan temáticamente el contenido del corpus y permiten mejorar los sistemas RAG en dos dimensiones principales:

- **Mejora de la precisión y cobertura semántica:** Mediante la estrategia de recuperación combinada (Combined Retrieve RAG, detallada en la Sección 4.1), las preguntas de KRAQ se utilizan como consultas complementarias que amplían y diversifican el conjunto de documentos recuperados. Se hipotetiza que esto proporciona al LLM un contexto más rico y variado, lo que conduce a una mejora en la precisión y la cobertura semántica de las respuestas generadas.
- **Mejora de la eficiencia computacional:** En el contexto de Speculative RAG (explicado en la Sección 5.1), las preguntas representativas de KRAQ permiten el pre-cómputo de *embeddings* instruidos. Esta optimización, denominada Efficient Speculative RAG, está diseñada para reducir la latencia del algoritmo al evitar el costoso cálculo de *embeddings* en línea para cada consulta del usuario.

Este enfoque posiciona a la QG no solo como una herramienta generativa *per se*, sino como un componente estratégico dentro del *pipeline* de recuperación y generación, orientado a mejorar tanto la calidad como la eficiencia de los sistemas RAG.

2.3.2. Avances recientes

La generación automática de preguntas (QG) puede abordarse a partir de distintos tipos de datos de entrada, como información estructurada (proveniente de grafos de conocimiento o bases de datos), elementos visuales (imágenes, videos) o, como es más frecuente y relevante para este trabajo, texto no estructurado [25]. Mientras que la QG a partir de entradas estructuradas suele emplear modelos como los Graph2Seq [4, 10] y la QG visual se apoya en arquitecturas multimodales [9, 83], esta tesis se enfoca exclusivamente en la generación de preguntas desde texto libre, modalidad conocida como Text-based Question Generation (TQG).

La TQG ha evolucionado significativamente en los últimos años, impulsada por el avance de los modelos de lenguaje preentrenados (PLMs) y, más recientemente, por los LLMs. Inicialmente, los modelos de TQG seguían el paradigma clásico de secuencia a secuencia (Seq2Seq), empleando arquitecturas basadas en RNNs (como LSTMs o GRUs) [18]. Posteriormente, la arquitectura Transformer [75] fue adoptada para mejorar la captura de dependencias de largo alcance en el texto de entrada [76]. Sin embargo, estos enfoques, aunque pioneros, presentaban limitaciones al procesar documentos extensos debido a la complejidad cuadrática de la auto-atención o a la dificultad de las RNNs para mantener información a través de secuencias muy largas. Además, a menudo sufrían problemas de sobreajuste debido a la relativa escasez de datos de entrenamiento específicos y de alta calidad para QG.

El surgimiento de modelos preentrenados masivos como T5 [65], BART [39], y UNILM [17], entre otros, marcó un punto de inflexión. Estos modelos, entrenados en vastos corpus textuales con objetivos auto-supervisados (como la predicción de palabras enmascaradas o la decodificación de secuencias), aprenden representaciones lingüísticas ricas y generalizables. Esto permitió su posterior fine-tuning sobre tareas de QG con conjuntos de datos más pequeños, logrando mejoras sustanciales en la fluidez, coherencia y precisión semántica de las preguntas generadas.

Dentro de esta línea, trabajos recientes han explorado arquitecturas más sofisticadas y mecanismos de control. Por ejemplo, SG-CQG [16] para la generación de preguntas conversacionales y MultiFactor [81] para QG con planificación de contenido multinivel, investigan arquitecturas modulares, el control explícito del tipo de pregunta y la incorporación de mecanismos de planificación semántica. Estos esfuerzos buscan aumentar la relevancia, la diversidad y la complejidad de las preguntas generadas, yendo más allá de la simple transformación sintáctica del texto de entrada.

Paralelamente, se ha observado una tendencia creciente hacia el uso de representaciones estructuradas intermedias, como grafos semánticos o redes de entidades extraídas del texto fuente, para enriquecer el contexto y mejorar la selección de contenido relevante para la QG [21, 58]. Esta integración de representaciones estructuradas (derivadas del texto no estructurado) con modelos de generación constituye una línea de trabajo que conecta directamente con el enfoque propuesto en esta tesis, donde el grafo de conocimiento juega un papel central.

Con la llegada de los LLMs de última generación, como los de la familia GPT [8, 55], se ha explorado la generación de preguntas bajo esquemas de *zero-shot* (sin ejemplos específicos de la tarea) y *few-shot* (con unos pocos ejemplos) utilizando técnicas de *in-context learning* y *prompt engineering* [42]. Estos modelos ofrecen ventajas notables en términos de generalización y flexibilidad, ya que pueden generar preguntas para una amplia variedad de contextos sin necesidad de un fine-tuning específico. No obstante, aún presentan desafíos en el control fino de la generación (e.g., asegurar que la pregunta sea sobre un aspecto particular del texto), en la cobertura temática exhaustiva de corpus extensos, y en la interpretabilidad del proceso generativo.

En esta tesis, nos situamos dentro de esta última ola de desarrollos, proponiendo un enfoque híbrido. Se aprovecha la potencia generativa de los modelos preentrenados (mediante fine-tuning específico), pero como novedad se incorpora un *pipeline* estructurado que utiliza grafos de conocimiento y la detección de comunidades semánticas. Esta estructuración previa del corpus permite guiar la generación de preguntas de manera que se logre una cobertura temática más balanceada y representativa del contenido global, optimizando así su utilidad posterior en la mejora de sistemas RAG, como se detalla en el Capítulo 3.

2.4. Grafos de Conocimiento y detección de comunidades

Los métodos propuestos en esta tesis, particularmente el sistema KRAQ, se fundamentan en la capacidad de transformar corpus textuales en representaciones estructuradas que facilitan un análisis semántico profundo y la identificación de agrupaciones temáticas. En esta sección, se describen dos pilares conceptuales para este proceso: los Grafos de Conocimiento (KGs) como formalismo para representar información semantica, y el Clustering en Grafos como técnica para descubrir comunidades cohesivas dentro de estas estructuras.

2.4.1. Grafos de Conocimiento

Los Grafos de Conocimiento (Knowledge Graphs, KGs) se han consolidado como una poderosa herramienta para representar información de manera estructurada, capturando entidades del mundo real y las complejas relaciones que existen entre ellas. A diferencia de las bases de datos relacionales tradicionales o el texto no estructurado, los KGs ofrecen una representación semántica rica, interpretable y fácilmente navegable, lo que facilita el razonamiento y el descubrimiento de conocimiento [30, 53].

Definición y estructura

Formalmente, un grafo de conocimiento se define comúnmente como un conjunto de tripletas factuales de la forma (h, r, t), donde h (entidad cabeza o *head*) y t (entidad cola o *tail*) son nodos que representan entidades (e.g., personas, organizaciones, lugares, conceptos, eventos), y r es una arista dirigida y etiquetada que representa la relación semántica que vincula a h con t (e.g., *nació_en*, *es_miembro_de*, *causa_de*). Por lo tanto, un KG puede modelarse como un multigrafo dirigido y etiquetado $G = (V, E, L)$, donde V es el conjunto de nodos (entidades), E es el conjunto de aristas (relaciones), y L es un conjunto de etiquetas para esas relaciones.

Esta estructura permite una representación explícita del conocimiento que va más allá de la simple coocurrencia de palabras, capturando el significado y el contexto de las interacciones entre entidades.

Construcción de Grafos de Conocimiento

La construcción de un KG, también conocida como población de KG, puede realizarse a partir de diversas fuentes:

- **Fuentes estructuradas y semi-estructuradas:** Utilizando bases de datos existentes, hojas de cálculo, o información de la web estructurada (e.g., tablas HTML, infoboxes de Wikipedia). KGs a gran escala como DBpedia, YAGO o Wikidata se han construido en gran medida a partir de este tipo de fuentes.
- **Texto no Estructurado:** Mediante técnicas de PLN, que incluyen:
  - **Reconocimiento de entidades nombradas (NER):** Para identificar menciones de entidades en el texto y clasificarlas (e.g., Persona, Organización, Lugar).
  - **Extracción de relaciones (RE):** Para identificar relaciones semánticas entre las entidades detectadas. Esto puede hacerse con enfoques basados en patrones, aprendizaje automático supervisado, o más recientemente, mediante Open Information Extraction (OpenIE) o el uso de LLMs para generar tripletas (h, r, t) directamente a partir de oraciones. Por ejemplo, de la oración "Albert Einstein formuló la teoría de la relatividad", se podría extraer la tripleta (*Albert Einstein, formuló, teoría de la relatividad*).

En los últimos años, ha habido un interés creciente en la construcción de KGs "orientados a la tarea" o "contextuales", donde el grafo se construye o se adapta dinámicamente para ser relevante para una consulta o un conjunto de documentos específico, en lugar de intentar

modelar todo el conocimiento universal [87]. Esta aproximación es particularmente relevante para aplicaciones como RAG, donde se busca sintetizar un contexto estructurado a partir de un subconjunto de documentos recuperados. En esta tesis, como se detallará en la Sección 3.1, se utiliza el *framework* GraphRAG [19], que emplea LLMs para la extracción de entidades, relaciones y afirmaciones a partir de un corpus textual para construir un KG.

Integración con LLMs y sistemas RAG

La integración de KGs con LLMs y, específicamente, con sistemas RAG, es un área de investigación activa y prometedora que busca combinar lo mejor de ambos mundos: la capacidad de razonamiento estructurado y el conocimiento factual explícito de los KGs con la fluidez generativa y la comprensión del lenguaje natural de los LLMs. Varias estrategias de integración han sido exploradas:

- **KGs como fuente de recuperación en RAG:** En lugar de (o además de) recuperar pasajes de texto plano, el componente de recuperación puede consultar un KG para obtener hechos relevantes, entidades relacionadas o subgrafos que proporcionen un contexto estructurado al LLM generador [85, 86].

- **Construcción de KGs a partir de documentos recuperados:** El sistema RAG puede construir dinámicamente un KG local a partir de los documentos recuperados para una consulta específica. Este KG contextual puede luego ser utilizado para refinar la comprensión, identificar entidades clave o guiar la generación de la respuesta.

- **Uso de KGs para mejorar la Generación:** La información del KG puede utilizarse para controlar o restringir la generación del LLM, asegurando que las respuestas sean consistentes con los hechos del KG o que se enfoquen en entidades/relaciones particulares.

- **GraphRAG y enfoques similares:** El trabajo de Edge et al. [19], denominado GraphRAG, propone un *pipeline* donde se construye un KG global a partir de un corpus. Luego, se utilizan técnicas de detección de comunidades en este grafo para identificar agrupaciones temáticas. A partir de estas comunidades (y sus resúmenes textuales generados por LLMs), se pueden realizar tareas como la sumarización global del corpus o la respuesta a preguntas que requieren información de múltiples documentos. Este enfoque, que sirve de base para la primera parte del *pipeline* de KRAQ, demuestra cómo la estructura del KG puede guiar la síntesis de información a diferentes niveles de granularidad.

En el contexto de esta tesis, la construcción de un grafo de conocimiento a partir del corpus documental es el primer paso fundamental del sistema KRAQ. Este KG permite luego identificar comunidades semánticas, generar resúmenes representativos por comunidad y, finalmente, producir un conjunto de preguntas que capturen la esencia temática del corpus. Estas preguntas son luego utilizadas para optimizar los sistemas RAG, demostrando cómo los KGs pueden actuar como una capa intermedia crucial para una interacción más explicable, eficiente y controlada entre la recuperación de información y la generación de lenguaje.

2.4.2. Clustering en grafos

El *clustering* en grafos, más comúnmente conocido en el análisis de redes como detección de comunidades (*community detection*), es una tarea fundamental que busca particionar los nodos de un grafo en grupos o clústeres. El principio es que los nodos dentro de una misma comunidad deben estar más densa o fuertemente conectados entre sí que con los nodos pertenecientes a otras comunidades [22, 32].

Modularidad: una métrica para la calidad de las comunidades

Una de las métricas más influyentes y ampliamente utilizadas para cuantificar la calidad de una partición de un grafo en comunidades es la **modularidad**, introducida por Newman [52]. La modularidad Q mide la fracción de las aristas que caen dentro de las comunidades dadas, menos la fracción esperada si las aristas se distribuyeran al azar manteniendo los grados de los nodos. Un valor de modularidad positivo y alto indica una estructura comunitaria fuerte y bien definida.

Para un grafo no dirigido con m aristas y una partición en c comunidades, la modularidad se define como:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(C_i, C_j)$$

donde:

- A_{ij} es un elemento de la matriz de adyacencia del grafo (1 si hay una arista entre el nodo i y el nodo j, 0 en caso contrario).
- k_i es el grado (número de conexiones) del nodo i.
- $m = \frac{1}{2} \sum_i k_i$ es el número total de aristas en el grafo.
- C_i es la comunidad a la que pertenece el nodo i.
- $\delta(C_i, C_j)$ es la función delta de Kronecker, que es 1 si $C_i = C_j$ (es decir, si los nodos i y j están en la misma comunidad) y 0 en caso contrario.

El término $\frac{k_i k_j}{2m}$ representa la probabilidad esperada de que exista una arista entre los nodos i y j en un grafo aleatorio con la misma distribución de grados (modelo nulo de configuración). Muchos algoritmos de detección de comunidades buscan maximizar esta métrica Q.

Algoritmos destacados: Louvain y Leiden

Entre la vasta gama de algoritmos de detección de comunidades, dos métodos heurísticos basados en la optimización de la modularidad han ganado gran popularidad debido a su eficiencia y efectividad en grafos grandes:

Algoritmo de Louvain: Propuesto por Blondel et al. [5], el método de Louvain es un algoritmo aglomerativo y jerárquico que opera en dos fases iterativas:

1. **Optimización local de la modularidad:** Para cada nodo, se considera moverlo a cada una de las comunidades de sus nodos vecinos. El nodo se asigna a la comunidad que resulta en el mayor incremento positivo de la modularidad

global. Esta fase se repite para todos los nodos hasta que no se puedan realizar más movimientos que mejoren la modularidad.

2. **Agregación de la red:** Se construye un nuevo grafo donde cada comunidad identificada en la fase anterior se convierte en un único supernodo. Las aristas entre los nuevos supernodos se ponderan según la suma de las aristas entre los nodos de las comunidades correspondientes.

Estas dos fases se repiten hasta que la modularidad ya no puede incrementarse significativamente. La eficiencia y simplicidad del método de Louvain lo han convertido en un estándar de facto para el análisis de comunidades en redes grandes.

Algoritmo de Leiden: Introducido por Traag et al. [72] como una mejora significativa sobre el algoritmo de Louvain, el método de Leiden fue diseñado para abordar algunas de las limitaciones teóricas y prácticas de su predecesor, resultando en particiones comunitarias de mayor calidad y más robustas. Si bien Leiden también es un algoritmo aglomerativo jerárquico que busca optimizar la modularidad, introduce modificaciones cruciales en sus fases operativas:

1. **Movimiento local de nodos:** Similar a Louvain, los nodos se mueven inicialmente a comunidades vecinas si tal movimiento incrementa la modularidad. Sin embargo, Leiden pone más énfasis en explorar el vecindario de cada nodo de manera más exhaustiva.
2. **Refinamiento de la partición (fase clave de mejora):** Esta es una de las innovaciones principales. Después de una fase inicial de movimiento de nodos, Leiden toma cada comunidad formada y intenta subdividirla recursivamente. El algoritmo intenta refinar cada comunidad individualmente, buscando subestructuras que puedan ser comunidades por sí mismas. Solo si una comunidad no puede ser dividida de forma que se incremente la modularidad, se considera una unidad cohesiva. Este paso ayuda a evitar que los nodos queden "atrapados" en comunidades grandes pero poco densas, un problema que a veces afecta a Louvain.
3. **Agregación de la red basada en particiones refinadas:** Una vez que las comunidades han sido refinadas (y posiblemente divididas), la red se agrega. Los nodos se agrupan en estas comunidades refinadas, y solo aquellas comunidades que no pudieron ser descompuestas más (es decir, son bien cohesivas) se consideran para la siguiente iteración de agregación de la red.

Ambos algoritmos son altamente eficientes y escalables, lo que permite su aplicación a grafos con millones de nodos y aristas, como los que pueden derivarse de grandes corpus textuales.

2.5. Trabajos relacionados a la tesis

En esta sección, se analizan con mayor detalle los trabajos previos que guardan una relación más cercana con las problemáticas abordadas y las soluciones propuestas en esta tesis. Se busca establecer un diálogo con el estado del arte, identificando tanto las inspiraciones como los puntos de divergencia que definen la originalidad y el aporte de KRAQ y sus aplicaciones.

2.5.1. Trabajos similares en generación de preguntas

Si bien la generación automática de preguntas (QG) es un campo amplio, varios trabajos recientes se alinean con los objetivos y/o metodologías de esta tesis, particularmente aquellos que utilizan representaciones estructuradas del conocimiento o buscan generar preguntas representativas de un corpus. A continuación, analizamos dos trabajos relevantes que abordan la QG desde perspectivas metodológicamente afines: SG-CQG [16] y Savaal [54].

SG-CQG: generación de preguntas conversacionales sin respuesta conocida Do et al. [16] presentan SG-CQG, un *framework* para la generación de preguntas conversacionales (CQG) en un escenario *answer-unaware*. Su método se desarrolla en dos etapas:

1. **Selección del Contenido a Preguntar (*what-to-ask*):** Se construye un grafo semántico local a partir de un documento para identificar y seleccionar una oración pertinente, denominada *racional*, como base para la pregunta.

2. **Formulación de la Pregunta (*how-to-ask*):** Con el *racional* seleccionado, se utiliza un clasificador para determinar el tipo de pregunta y luego modelos T5 generan la pregunta final, asegurando la coherencia conversacional.

Diferencias procedimentales con KRAQ: A pesar de la inspiración compartida en el uso de grafos, los procedimientos divergen. SG-CQG opera sobre un grafo de un *documento individual* para seleccionar una *oración* específica. KRAQ (como veremos en el Capitulo 2, en cambio, construye un grafo *global* del corpus para identificar *comunidades temáticas enteras*, generando preguntas a partir de resúmenes de estas, lo que representa una abstracción de mayor nivel.

Savaal: generación escalable de preguntas orientadas a conceptos Noorbakhsh et al. [54] proponen Savaal, un sistema de QG escalable para material educativo. Su *pipeline* consiste en:

1. **Identificación de conceptos clave:** Se extraen y clasifican ideas centrales del corpus mediante un proceso distribuido.

2. **Recuperación de evidencia contextual:** Para cada concepto, se recuperan pasajes textuales relevantes con un *retriever* denso.

3. **Generación de preguntas pedagógicas:** Un LLM genera preguntas a partir de los conceptos y los pasajes recuperados para evaluar la comprensión.

Diferencias procedimentales con KRAQ: Si bien ambos buscan generar preguntas a partir de unidades temáticas, los métodos difieren. Savaal identifica conceptos clave sin depender de un grafo global ni de clustering. KRAQ, por el contrario, define sus unidades temáticas a través de la detección de comunidades en el grafo de conocimiento del corpus. Además, KRAQ genera preguntas a partir de un resumen cohesivo de la comunidad, un nivel de agregación superior al de Savaal, que combina un concepto aislado con pasajes de texto.

Posicionamiento de KRAQ Los trabajos de SG-CQG y Savaal ilustran el potencial de usar abstracciones semánticas para guiar la QG. KRAQ se distingue por su *pipeline*

integral que parte de un grafo de conocimiento global. La identificación de comunidades, la síntesis de resúmenes y la posterior generación de preguntas con un LLM *fine-tuneado* constituyen un flujo metodológico específico. La principal novedad de KRAQ es cómo este conjunto de preguntas se convierte en un activo para optimizar sistemas RAG (Combined Retrieve RAG y Efficient Speculative RAG), un diferenciador clave de esta tesis.

2.5.2. Trabajos similares en RAG

Para incrementar la robustez y la cobertura semántica en la fase de recuperación, una línea de investigación significativa se ha centrado en el uso de múltiples formulaciones de la consulta original. Al diversificar las búsquedas, se busca construir un contexto documental más completo. A continuación, se revisan trabajos clave en esta línea.

RAG-Fusion Propuesto por Rackauckas [63], este enfoque sigue los siguientes pasos:

1. Se generan automáticamente diversas reformulaciones de la pregunta original del usuario utilizando un LLM.
2. Cada reformulación se emplea para realizar una búsqueda independiente en el corpus documental.
3. Los conjuntos de documentos recuperados se consolidan y se reordenan mediante Reciprocal Rank Fusion (RRF) para destacar la evidencia más robusta.

Diferencias procedimentales con Combined Retrieve RAG: La principal diferencia radica en el origen y la naturaleza de las consultas adicionales. RAG-Fusion genera reformulaciones de la pregunta del usuario *en tiempo real (online)*, lo que incrementa la latencia. Combined Retrieve RAG, en cambio, utiliza un conjunto de preguntas representativas de KRAQ que han sido *pre-generadas offline*, lo que optimiza la eficiencia al evitar la generación de consultas en tiempo de ejecución.

DMQR-RAG Presentado por Li et al. [41], este método busca aprovechar múltiples consultas de la siguiente manera:

1. Se aplica un conjunto controlado de transformaciones semánticas (generalización, especificación, etc.) sobre la consulta original para inducir diversidad.
2. Las variantes más prometedoras son seleccionadas de forma adaptativa.
3. Estas variantes se emplean en las etapas de recuperación y generación.

Diferencias procedimentales con Combined Retrieve RAG: Mientras DMQR-RAG aplica un conjunto de *transformaciones genéricas* predefinidas a la pregunta del usuario, Combined Retrieve RAG utiliza consultas que se derivan de la *estructura semántica intrínseca* del corpus. Las preguntas de KRAQ no son simples reformulaciones, sino representaciones de los temas centrales del corpus, aspirando a una cobertura temática más significativa.

Posicionamiento de *Combined Retrieve RAG* (contribución de esta tesis) El método *Combined Retrieve RAG*, detallado en la Sección 4.1, se inspira en la idea central de estos enfoques de múltiples consultas, pero se diferencia fundamentalmente en su

estrategia. En lugar de generar reformulaciones en línea (como RAG-Fusion) o aplicar transformaciones genéricas (como DMQR-RAG), nuestra propuesta utiliza un conjunto de preguntas representativas $\mathcal{Q}^{\mathcal{K}}$ generadas por el sistema KRAQ (ver Capítulo 3). Estas preguntas, al derivarse de la estructura semántica profunda del corpus, ofrecen una diversidad semántica controlada y temáticamente alineada. Se busca así un balance entre la diversificación contextual y la eficiencia computacional, proporcionando una cobertura temática más significativa que las alternativas puramente sintácticas o genéricas.

3. KRAQ

Como se introdujo en el Capítulo 2, y para abordar las limitaciones discutidas en la Introducción (Capítulo 1), esta tesis propone KRAQ (*Knowledge-graph Representative Automatic Questions*). Este capítulo detalla la metodología detrás de KRAQ, un sistema diseñado para construir un conjunto de preguntas semánticamente representativas a partir de un corpus textual, sentando las bases para las optimizaciones de RAG que se explorarán en capítulos posteriores. Primero, se describirá su arquitectura y componentes metodológicos, seguido de una presentación exhaustiva de su diseño experimental, implementación y los resultados obtenidos en la generación de dichas preguntas.

3.1. Metodología de KRAQ

El sistema KRAQ (*Knowledge-graph Representative Automatic Questions*), núcleo de esta tesis, introduce una arquitectura novedosa para optimizar los sistemas RAG. La intuición fundamental es que, al modelar la estructura semántica profunda de un corpus y generar a partir de ella un conjunto de preguntas representativas, podemos anticipar y alinearnos mejor con las posibles consultas de un usuario. Estas preguntas, extraídas de la esencia temática del corpus, no solo ofrecen una forma de condensar el conocimiento documental, sino que actúan como herramientas estratégicas. Específicamente, permiten enriquecer la diversidad y precisión de la información recuperada y optimizar la eficiencia computacional de los pipelines RAG, abordando así algunas de sus limitaciones clave.

KRAQ, cuyo pipeline general se ilustra en la Figura 3.1, transforma un corpus documental en este valioso conjunto de preguntas representativas a través de la siguiente secuencia de etapas:

1. **Extracción de conocimiento:** Se procesa el texto para identificar y extraer entidades, relaciones y afirmaciones clave, sentando las bases para una representación estructurada.

2. **Construcción del grafo de conocimiento:** La información extraída pasa por un proceso de desambiguación y se integra en un grafo, donde las entidades son nodos y las relaciones aristas, capturando las conexiones del corpus.

3. **Detección de comunidades semánticas:** Sobre este grafo, se aplican algoritmos de clustering (Leiden) para identificar comunidades de nodos densamente conectados, que representan agrupaciones temáticas coherentes.

4. **Síntesis de resúmenes comunitarios:** Para cada comunidad detectada, se genera un resumen textual que condensa su contenido temático principal.

5. **Generación de preguntas representativas:** Finalmente, estos resúmenes comunitarios se utilizan como entrada para un modelo de lenguaje *fine-tuneado*, que produce una pregunta representativa para cada comunidad.

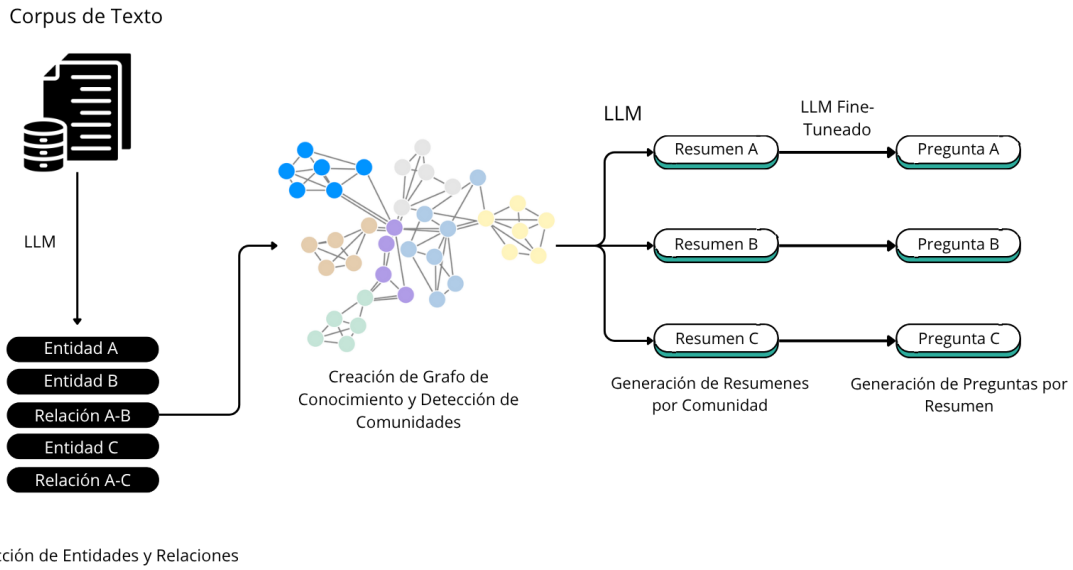

Fig. 3.1: Diagrama del pipeline metodológico de KRAQ: desde el corpus de documentos hasta la generación de preguntas representativas por comunidad.

Es fundamental destacar que las primeras cuatro etapas de este pipeline (desde la extracción de conocimiento (1ra) hasta la síntesis de resúmenes comunitarios (4ta)) se implementan utilizando y adaptando el robusto *framework* GraphRAG, propuesto por Edge et al. [19]. La adopción de GraphRAG para estos pasos iniciales nos proporciona una base metodológica sólida, reproducible y validada para la construcción del grafo de conocimiento, la identificación de comunidades y la generación de sus correspondientes resúmenes textuales.

La **contribución central y novedosa de esta tesis, y por ende el núcleo distintivo de KRAQ**, reside en la **quinta y última etapa**: la generación de un conjunto de preguntas representativas a partir de dichos resúmenes comunitarios. Esta transformación de resúmenes en preguntas se realiza mediante un modelo de lenguaje específicamente *fine-tuneado* para esta tarea. Mientras GraphRAG utiliza sus resúmenes para responder directamente a consultas externas, KRAQ va un paso más allá al convertir estos resúmenes en un nuevo activo que como se demostrara posteriormente, puede ser utilizado para la optimización de sistemas RAG.

En las siguientes secciones, se detallará cada uno de los componentes y procesos que conforman el sistema KRAQ. Para ilustrar de manera concreta cómo opera KRAQ, seguiremos un ejemplo temático centrado en el ecosistema de la educación superior y la investigación en Argentina, cuyas entidades principales son la Universidad de Buenos Aires (UBA) y el Premio Nobel.

3.1.1. Extracción de entidades y relaciones con GraphRAG

La primera etapa del pipeline, llevada a cabo mediante el *framework* GraphRAG, consiste en convertir un corpus documental en una representación semántica estructurada mediante la extracción de entidades, relaciones y afirmaciones relevantes. Esta representa-

ción constituye la base para la posterior construcción del grafo de conocimiento.

El proceso, tal como lo implementa GraphRAG, inicia con la fragmentación del corpus en segmentos de longitud fija, denominados *chunks*. Esta división busca asegurar una cobertura contextual adecuada y, al mismo tiempo, respetar las restricciones inherentes al tamaño de contexto de los LLMs [37, 45]. Cada uno de estos *chunks* es tratado como una unidad semántica fundamental de la cual se extraerá conocimiento estructurado.

Sobre cada *chunk*, GraphRAG aplica un LLM mediante el uso de *prompts* específicamente diseñados para la detección y caracterización de tres tipos principales de elementos semánticos:

- **Entidades:** Se identifican junto con su nombre, su tipología (e.g., persona, organización, lugar) y una descripción concisa de sus atributos y actividades relevantes dentro del contexto del fragmento.
- **Relaciones:** Se detectan las conexiones significativas entre pares de entidades previamente identificadas. Estas relaciones se representan formalmente como triplas dirigidas (entidad origen, tipo de relación, entidad destino), acompañadas de una descripción textual que explica la naturaleza del vínculo.
- **Afirmaciones (*claims*):** Se extraen proposiciones fácticas que condensan hechos, eventos o condiciones notables mencionados en el texto y asociados a las entidades o sus relaciones.

La utilización de LLMs para esta tarea ha demostrado ser una aproximación eficiente [90]. Los *prompts* exactos empleados por GraphRAG se detallan en el apéndice de su publicación original [19].

Texto Original del Chunk	
"La Universidad de Buenos Aires (UBA), fundada en 1821, es una de las instituciones educativas más prestigiosas de América Latina. Varios premios Nobel han sido egresados de esta universidad pública argentina."	
Tipo de Elemento	**Información Extraída**
Entidades	*Nombre:* Universidad de Buenos Aires *Tipo:* Institución Educativa *Descripción:* Universidad pública argentina.
	Nombre: Premio Nobel *Tipo:* Distinción *Descripción:* Galardón que ha sido recibido por egresados de la UBA.
Relaciones	*Cabeza (Head):* Universidad de Buenos Aires *Cola (Tail):* Premio Nobel *Relación:* ha formado egresados que han recibido
Afirmaciones (Claims)	• La Universidad de Buenos Aires (UBA) fue fundada en 1821. • La UBA es una de las instituciones educativas más prestigiosas de América Latina. • Egresados de la UBA han recibido premios Nobel. • La UBA es una universidad pública argentina.

Fig. 3.2: Ejemplo ilustrativo de la extracción de información semántica a partir de un *chunk* sobre la UBA y Premios Nobel, tal como la realizaría GraphRAG. Se identifican entidades clave, sus relaciones y afirmaciones.

Continuando con nuestro ejemplo conductor, la Figura 3.2 muestra cómo, a partir de un *chunk* que menciona a la "Universidad de Buenos Aires (UBA)" y los "Premios Nobel", GraphRAG extraería las entidades correspondientes, sus tipos, descripciones, la relación "ha formado egresados que han recibido", y varias afirmaciones relevantes. Dicho formato de salida captura la información semántica clave que establece las bases para la siguiente etapa.

Reflexión iterativa para la mejora de cobertura en GraphRAG. Es común que en la extracción inicial de entidades, ciertos elementos menos evidentes sean omitidos. Para mitigar esto, GraphRAG emplea una técnica de *self-reflection* [48]. Una vez realizada la extracción, la salida se reenvía al LLM, solicitándole que identifique y justifique elementos relevantes omitidos, mediante *prompts* como:

> "¿Qué entidades o relaciones relevantes no fueron extraídas previamente? Justifique su relevancia."

Este ciclo puede repetirse. Para un análisis detallado de esta técnica en GraphRAG, consúltese [19], Apéndice A.2.

3.1.2. Construcción del grafo de conocimiento con GraphRAG

Una vez extraídas las entidades, relaciones y afirmaciones desde los *chunks* de texto, el siguiente paso, también gestionado por GraphRAG, consiste en integrar esta información en un grafo de conocimiento. Este grafo, que para KRAQ es fundamental, servirá de base para el clustering y la generación de resúmenes.

Abstracción semántica y agregación en GraphRAG La información extraída por GraphRAG (entidades, relaciones, afirmaciones) puede considerarse una forma de resumen abstracto del contenido de los *chunks* [19]. Dado que los documentos se fragmentan, una misma entidad o relación puede detectarse múltiples veces.

Para construir el grafo, GraphRAG primero realiza un proceso de **desambiguación de entidades** (*entity disambiguation*), crucial para asegurar que cada entidad única corresponda a un único nodo. Esta implementación en GraphRAG utiliza estrategias de normalización y unificación, comenzando con la coincidencia de cadenas (*string matching*) [12, 20]. Así, para nuestro ejemplo, si "Universidad de Buenos Aires" y "UBA" aparecen, GraphRAG evalúa su unificación bajo un solo nodo si su similitud léxica o semántica (potencialmente verificada por un LLM o reglas heurísticas) supera cierto umbral.

Luego, GraphRAG agrega y combina las descripciones asociadas a cada entidad desambiguada. Retomando el ejemplo, las diversas menciones descriptivas de la "Universidad de Buenos Aires" se consolidan en una descripción concisa para el nodo UBA. Las relaciones extraídas se convierten en aristas dirigidas, y sus descripciones se agregan para enriquecer semánticamente el grafo. La Figura 3.3 ilustra cómo las entidades y la relación de nuestro ejemplo ("UBA ha formado egresados que han recibido Premio Nobel") se representan gráficamente.

Finalmente, las afirmaciones relevantes (*claims*) extraídas por GraphRAG, como "la UBA fue fundada en 1821", se asocian a los nodos o aristas correspondientes del grafo, sirviendo como anotaciones para la posterior generación de resúmenes.

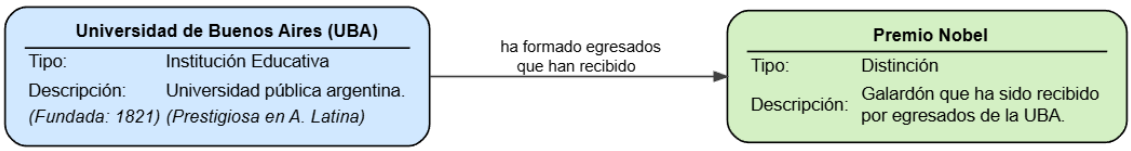

Fig. 3.3: Ilustración del grafo de conocimiento simplificado para el ejemplo de la Universidad de Buenos Aires y el Premio Nobel, mostrando las entidades extraídas (detalladas en la Figura 3.2) y la relación que las vincula, como lo construiría GraphRAG.

Representación final

El resultado de este proceso es un grafo dirigido $G = (V, E)$, donde:

- Cada nodo $v \in V$ representa una entidad única, acompañada por su descripción.
- Cada arista $e = (v_i, v_j) \in E$ representa una relación dirigida entre dos entidades.
- Las afirmaciones extraídas se almacenan como propiedades adicionales, útiles para tareas posteriores de resumen.

3.1.3. Detección de comunidades en el grafo de GraphRAG

Una vez construido el grafo de conocimiento por GraphRAG, el siguiente paso en su pipeline (y crucial para KRAQ) es la identificación de comunidades semánticas. Estas son subconjuntos de entidades (nodos) fuertemente conectadas entre sí que comparten una temática o dominio común, permitiendo organizar el contenido en regiones cohesivas.

Desde el punto de vista teórico, los grafos de conocimiento a menudo exhiben propiedades de redes de **mundo pequeño** (*small-world networks*) [79], con alto coeficiente de clustering, lo que favorece la emergencia de estructuras comunitarias. La calidad de estas estructuras puede medirse con la modularidad (ver Sección 2.4.2) [52]. Estas agrupaciones, o comunidades, representan conjuntos de nodos con alta densidad de conexiones internas y baja conectividad externa, interpretables como focos temáticos.

Para realizar esta partición comunitaria, GraphRAG emplea el algoritmo Leiden [72]. Como se detalló en la Sección 2.4.2, Leiden optimiza la modularidad y garantiza comunidades con fuerte conectividad interna. GraphRAG aplica Leiden de forma jerárquica para descubrir la estructura comunitaria a múltiples niveles:

1. **Detección de comunidades de nivel superior (Nivel 0):** Leiden se ejecuta sobre el grafo completo.
2. **Subdivisión recursiva:** Cada comunidad se trata como un subgrafo y Leiden se aplica nuevamente para subdividirla.
3. **Criterio de detención:** La recursión se detiene al alcanzar comunidades ”hoja” (no divisibles sin pérdida de cohesión/modularidad) o un nivel de profundidad predefinido.

Para una descripción detallada del algoritmo Leiden, ver Sección 2.4.2.

Esta estructura jerárquica es valiosa, ya que cada nivel proporciona una partición exhaustiva y mutuamente excluyente de nodos [19], facilitando un enfoque de ”divide y

vencerás" para la generación de resúmenes. La Figura 3.4 muestra un ejemplo genérico de esta detección jerárquica. Aplicado a nuestro ejemplo conductor, la "UBA" y el "Premio Nobel", aunque relacionados, podrían asignarse a diferentes comunidades o a la misma en distintos niveles jerárquicos, según la densidad de sus conexiones con el resto del grafo.

Fig. 3.4: Ejemplo ilustrativo de la detección jerárquica de comunidades de nivel superior (Nivel 0) utilizando el algoritmo Leiden, tal como lo aplicaría GraphRAG. (Adaptado de Edge et al. [19], Fig. 4, Apéndice B).

En la práctica, este proceso de GraphRAG transforma el grafo global en un conjunto de particiones jerárquicas. Cada comunidad en un nivel seleccionado se considera una unidad semántica, base para la posterior generación de resúmenes por parte de GraphRAG, y luego de preguntas por parte de KRAQ.

3.1.4. Generación de resúmenes

Una vez detectadas las comunidades semánticas dentro del grafo de conocimiento, KRAQ genera un resumen textual por comunidad. Estos resúmenes actúan como condensaciones interpretables del contenido temático de cada grupo de entidades, relaciones y afirmaciones. Son fundamentales para reducir la redundancia en la representación del conocimiento y habilitar la posterior generación de preguntas representativas.

La motivación principal de esta etapa, tal como la implementa GraphRAG y la aprovecha KRAQ, es la de facilitar un resumen textual y escalable del corpus. Mientras que enfoques tradicionales de resumen (query-focused o extractivos) se enfrentan a límites contextuales de los LLMs o a redundancia en los pasajes recuperados, el enfoque de KRAQ permite resumir por partes coherentes y con semántica local fuerte aprovechando las comunidades en el grafo. Siguiendo un enfoque jerárquico, los resúmenes de comunidades hoja sirven como base para construir resúmenes de niveles superiores, logrando una visión global por agregación local [19].

Para cada comunidad identificada, se construye un resumen utilizando un LLM, alimentado con información de los nodos (entidades), las aristas (relaciones) y las afirmaciones asociadas.

El proceso varía ligeramente según el nivel jerárquico de la comunidad:

- **Comunidades hoja:** se priorizan las entidades y relaciones de mayor centralidad (calculada por grado), y se seleccionan en orden descendente hasta llenar el límite de tokens de entrada del modelo.

- **Comunidades de nivel superior:** si los elementos individuales no caben todos en el contexto del modelo, se reemplazan por los resúmenes ya generados de sus subcomunidades. De este modo, se logra una agregación bottom-up, donde los niveles altos reflejan progresivamente una visión más global del corpus.

En ambos casos, se utiliza un prompt de resumen (template prompt, ver 19), que guía al modelo para que incluya temas clave, relaciones centrales y afirmaciones relevantes.

Para ilustrar cómo GraphRAG genera estos resúmenes, consideremos que, para nuestro ejemplo conductor, se ha identificado una comunidad semántica que agrupa a la "Universidad de Buenos Aires" con entidades relacionadas a su prestigio y los "Premios Nobel" obtenidos por sus egresados. La Figura 3.5 detalla cómo, a partir de descripciones de nodos y afirmaciones pertinentes a esta comunidad UBA-Nobel, GraphRAG sintetizaría un resumen.

Ejemplo: Generación de Resumen para la Comunidad UBA-Premio Nobel	
Componentes de la Comunidad	**Información de Input al LLM (GraphRAG)**
Nodos Clave	
Entidad:	"Universidad de Buenos Aires (UBA)"
Descripción del Nodo:	"Institución educativa pública argentina, fundada en 1821, reconocida por su prestigio en América Latina y por la formación de múltiples personalidades destacadas."
Entidad:	"Premio Nobel"
Descripción del Nodo:	"Máximo galardón internacional otorgado anualmente por contribuciones excepcionales en diversas áreas del conocimiento y la paz."
Relaciones Clave	
Tipo de Relación:	"egresados_han_recibido"
Descripción de la Relación:	"La UBA se destaca porque varios de sus egresados han sido galardonados con el Premio Nobel a lo largo de su historia."
Afirmaciones (Claims)	
• "La UBA es una de las universidades más antiguas y prestigiosas de Argentina."	
• "Cinco ciudadanos argentinos han recibido el Premio Nobel, varios de ellos vinculados a la UBA."	
• "Bernardo Houssay, egresado y profesor de la UBA, fue el primer latinoamericano en recibir un Premio Nobel en ciencias (Medicina, 1947)."	
Resumen de Comunidad Generado (GraphRAG)	
"Esta comunidad temática se centra en la Universidad de Buenos Aires (UBA) y su distinguida conexión con los Premios Nobel. Se resalta el prestigio histórico de la UBA como formadora de figuras galardonadas con este reconocimiento internacional, subrayando su impacto en el ámbito académico y científico, como en el caso de Bernardo Houssay."	

Fig. 3.5: Ejemplo ilustrativo de la generación de un resumen por parte de GraphRAG para una comunidad temática centrada en la UBA y su vinculación con los Premios Nobel. El LLM sintetiza la información clave en un texto cohesivo.

De esta manera, el proceso implementado por GraphRAG culmina con la obtención de un resumen textual R_i, como el mostrado en la Figura 3.5, para cada comunidad C_i. Estos resúmenes R_i son el punto de partida para la etapa de generación de preguntas de KRAQ.

3.1.5. Generación de preguntas con KRAQ

El paso final y distintivo de KRAQ consiste en transformar cada resumen comunitario R_i (obtenido de GraphRAG) en una pregunta representativa Q_i, que capture el núcleo

temático y conceptual de dicha comunidad.

Dado un conjunto de resúmenes textuales R_i asociados a cada comunidad C_i, la función generadora de KRAQ es:

$$Q_i = f_\theta(R_i)$$

donde f_θ es un modelo de lenguaje específicamente *fine-tuneado* por nosotros con parámetros θ. Es crucial destacar que, en esta etapa, KRAQ opera exclusivamente sobre el resumen textual R_i, ya que se considera que este condensa eficientemente la información relevante de la comunidad para el propósito de generar una pregunta representativa.

Por ejemplo, tomando el resumen generado para la comunidad UBA-Premio Nobel (mostrado en la Figura 3.5):

> *"Esta comunidad temática se centra en la Universidad de Buenos Aires (UBA) y su distinguida conexión con los Premios Nobel. Se resalta el prestigio histórico de la UBA como formadora de figuras galardonadas con este reconocimiento internacional, subrayando su impacto en el ámbito académico y científico, como en el caso de Bernardo Houssay."*

A partir de este resumen, nuestro modelo f_θ de KRAQ podría generar una pregunta representativa como:

> *"¿Cómo se relaciona el prestigio de la Universidad de Buenos Aires con los Premios Nobel obtenidos por sus egresados, y qué ejemplos lo ilustran?"*

Esta pregunta ejemplifica el tipo de salida que KRAQ busca producir: una interrogante natural y temáticamente alineada con el contenido condensado del resumen comunitario.

Entrenamiento del modelo generador

Para construir el modelo f_θ, se utiliza un enfoque de *fine-tuning*, que consiste en ajustar un modelo preentrenado para que aprenda a generar preguntas significativas a partir de resúmenes de comunidades. El objetivo es que el modelo pueda transformar de manera controlada un resumen en una pregunta representativa del contenido.

La construcción del dataset de fine-tuning se inspiró en los pares triples comúnmente disponibles en datasets de pregunta-respuesta, típicamente de la forma:

$$(Q, A, E)$$

donde Q es la pregunta, A la respuesta, y E la evidencia o documento fuente. Sin embargo, dado que en KRAQ se desea generar preguntas sin depender directamente de una respuesta, se aplica un procedimiento de transformación del dataset para crear pares de entrenamiento de la forma (R, Q), donde R es un resumen generado a partir de E y Q, y Q es la pregunta original.

Este proceso se realiza en dos pasos:

1. **Síntesis del resumen (g)**
   Se utiliza un modelo LLM (e.g., GPT-4o) con ejemplos few-shot para sintetizar un resumen R a partir del contexto y la pregunta:

$$R = g(Q, E)$$

El modelo recibe como input la evidencia E y la pregunta Q, y se le instruye para generar un texto que resuma los conceptos centrales de la evidencia, sin hacer referencia explícita a la pregunta. El resultado es un resumen temático general similar a los obtenidos de las comunidades, que captura el contenido de la evidencia pero puede utilizarse de forma autónoma para generar nuevas preguntas.

2. **Fine-tuning del generador (f)**
   Una vez generados los pares (R, Q), se fine-tunea un modelo de lenguaje f_θ para predecir Q dado R. El objetivo de entrenamiento es maximizar la verosimilitud logarítmica del conjunto:

$$\mathbb{E}_{(R,Q)} \left[\log P_\theta(Q \mid R) \right]$$

   Este procedimiento sigue el esquema clásico de modelado de lenguaje condicional y permite que el modelo aprenda a inferir las posibles preguntas que emergen naturalmente de un resumen de comunidad.

Para ver los prompts especificos del entrenamiento ir a Sección de Implementacion 3.2.5

La decisión de utilizar resúmenes R, generados por un LLM, como base para el fine-tuning de nuestro modelo f_θ se valida con las conclusiones de Lampinen et al. [38]. Su investigación valida que el fine-tuning de modelos se beneficia de la aumentación del dataset con información inferida o sintetizada por LLMs (similar a nuestros resúmenes R), lo que resulta en un mejor entrenamiento, además de lograr asi trabajar sobre un input ajustado a nuestro objetivo de generación de preguntas.

Entonces, como resultado final de KRAQ, obtenemos un listado de preguntas representativas $Q^K = \{Q_1^K, Q_2^K, \ldots, Q_k^K\}$, una por cada comunidad detectada en el grafo de conocimiento (de todos los niveles).

3.2. Experimentación y resultados de KRAQ

Habiendo detallado la arquitectura metodológica de KRAQ en la sección anterior, esta sección se dedica a la validación empírica de su capacidad para generar preguntas representativas. Se describirán los datasets y las decisiones generales de implementación que sustentan todos los experimentos de esta tesis, para luego enfocarse en el diseño de evaluación específico para KRAQ, los detalles de su implementación utilizando GraphRAG, el proceso de fine-tuning del modelo generador de preguntas, y finalmente, se presentarán y analizarán los resultados obtenidos.

3.2.1. Datasets

La validación empírica de las metodologías propuestas en esta tesis se llevó a cabo utilizando un conjunto de cuatro benchmarks estándar, ampliamente reconocidos en la comunidad de investigación de procesamiento del lenguaje natural. Cada uno de estos datasets presenta características y desafíos particulares, lo que permite evaluar el rendimiento de los sistemas en diversos escenarios. A continuación, se describe cada dataset y sus particularidades.

TriviaQA

TriviaQA [33] es un dataset de pregunta-respuesta (QA) de dominio abierto donde las preguntas fueron redactadas por anotadores humanos de forma independiente a los documentos de evidencia, basándose en trivias preexistentes. Las evidencias provienen de diversas fuentes web y artículos de Wikipedia.

Estructura de instancia. Cada instancia en TriviaQA se modela como una tripleta (Q, A, E):

- Q: Una pregunta en lenguaje natural.
- A: Una respuesta corta y concisa (e.g., una entidad nombrada, una frase nominal).
- E: Un conjunto de documentos o pasajes textuales que contienen la evidencia para responder a Q.

Ejemplo: TriviaQA

Pregunta (Q): "Who wrote the novel Pride and Prejudice?"

Respuesta (A): "Jane Austen"

Desafíos. La principal particularidad de TriviaQA es el grado de desalineación entre la pregunta y el contexto evidencial, que puede incluir información distractora o redundante. Las respuestas no siempre se encuentran explícitamente en una única frase, lo que exige una robusta capacidad de recuperación y síntesis. Su diversidad temática y la naturalidad de sus preguntas lo consolidan como un benchmark de referencia para evaluar la robustez de los sistemas en entornos realistas y potencialmente ruidosos.

HotPotQA

HotPotQA [84] es un dataset diseñado específicamente para evaluar la capacidad de razonamiento multihop, que implica responder preguntas que requieren la combinación e integración de información proveniente de múltiples pasajes de texto.

Estructura de instancia. Cada instancia en HotPotQA se presenta como una tripleta (Q, A, E):

- Q: Una pregunta escrita por humanos, recolectada a través de tareas de crowdsourcing, y diseñada explícitamente para requerir múltiples saltos inferenciales.
- A: Una respuesta puntual y concisa, típicamente un nombre propio, una fecha, o alguna otra entidad nombrada.
- E: Generalmente contiene dos o más artículos de Wikipedia (o fragmentos de ellos) que, en conjunto, incluyen la información necesaria para derivar la respuesta A.

Tipos de Razonamiento Multihop. HotPotQA se distingue por la inclusión de dos tipos principales de preguntas que exigen un razonamiento multihop:

Ejemplo: Pregunta Puente (Bridge Question)

Este tipo de preguntas requiere que el sistema identifique una entidad o concepto en un fragmento de evidencia y luego utilice esa información como ”puente” para encontrar la respuesta final en otro fragmento.

Pregunta (Q): “What is the birth year of the author of The Selfish Gene?”
Respuesta (A): “1941”
Razonamiento Implicado:

1. *Salto 1 (Identificación):* Identificar que el autor de ”The Selfish Gene” es Richard Dawkins.
2. *Salto 2 (Búsqueda con Puente):* Usar ”Richard Dawkins” para buscar y encontrar su año de nacimiento.

Ejemplo: Pregunta Comparativa (Comparative Question)

Estas preguntas requieren la recuperación de atributos para dos o más entidades, a menudo de diferentes fragmentos de evidencia, y luego la realización de una comparación directa para determinar la respuesta.

Pregunta (Q): “Which city has a larger population: Milan or Turin?”
Respuesta (A): “Milan”
Razonamiento Implicado:

1. *Recuperación Entidad 1:* Encontrar la población de Milán.
2. *Recuperación Entidad 2:* Encontrar la población de Turín.
3. *Comparación:* Comparar ambos valores y determinar el mayor.

Desafíos. A diferencia de otros conjuntos de datos de QA donde las respuestas suelen encontrarse mediante búsquedas superficiales o patrones léxicos directos, HotPotQA presenta un desafío más complejo que requiere una lectura estructurada y razonamientos inferenciales encadenados. Esta complejidad lo posiciona como un benchmark exigente para evaluar modelos que combinan la recuperación de información con la generación de lenguaje natural.

BioASQ

BioASQ [74] es un benchmark de referencia para QA en el dominio biomédico, desarrollado en el contexto del BioASQ Challenge y utilizando literatura médica de PubMed.
Estructura de instancia. Cada instancia en BioASQ está compuesta por una tripleta (Q, A, E):

- Q: Una pregunta formulada por expertos (médicos, biólogos) sobre temas biomédicos.
- A: Una respuesta de referencia, típicamente presentada como una lista de entidades (e.g., genes, medicamentos, procedimientos) o sinónimos que responden directamente a la pregunta. Por ejemplo, para una pregunta sobre tratamientos, la respuesta podría ser una lista de fármacos específicos.

- E: Un conjunto de abstracts científicos de PubMed que contienen la evidencia para responder a Q.

Ejemplo: BioASQ

Pregunta (Q): "¿Qué medicamentos se utilizan en el tratamiento de la enfermedad de Crohn?"

Respuesta de Referencia (A): Una lista como: [infliximab, adalimumab, vedolizumab, ustekinumab, corticosteroides, metotrexato, azatioprina].

Desafíos. Muchas preguntas en BioASQ requieren la integración de información de múltiples documentos. El lenguaje técnico y la terminología especializada del dominio biomédico representan un desafío considerable para la comprensión y la síntesis semántica. El formato de las respuestas de referencia, al consistir en listas de entidades o términos, si bien busca la factualidad, presentará ciertas complicaciones para la evaluación mediante métricas de coincidencia exacta, como se detallará y abordará en secciones posteriores al analizar los resultados específicos para este dataset.

PubHealth

PubHealth [36] es un dataset diseñado para la verificación automática de hechos (*fact-checking*) en el dominio de la salud pública, utilizando evidencia de fuentes confiables como PubMed.

Estructura de instancia. Cada instancia en PubHealth se presenta como una tripleta (Q, A, E):

- Q: Una afirmación factual (un *claim*) en lenguaje natural.
- A: Una etiqueta de veracidad: `true`, `false`, o `mixture`.
- E: Fragmentos de texto de fuentes autorizadas que justifican la etiqueta de veracidad.

Ejemplo: PubHealth

Afirmación (Q): "Beber agua con limón alcaliniza el cuerpo."

Etiqueta (A): `false`.

Desafíos. PubHealth evalúa la capacidad de los modelos para determinar la veracidad de afirmaciones y justificarla. Requiere la síntesis de evidencia dispersa, especialmente para la categoría `mixture`. El lenguaje combina terminología técnica con discurso público, exigiendo una comprensión profunda.

3.2.2. Decisiones generales de implementación

Antes de detallar los experimentos específicos para KRAQ y sus aplicaciones, es pertinente describir las decisiones de implementación comunes que se adoptaron para la mayoría de los componentes y procesos evaluados en esta tesis. Estas elecciones, relativas a modelos de lenguaje, servidores de inferencia, y herramientas de gestión de datos, proporcionan el contexto técnico para la reproducibilidad y comprensión de los resultados experimentales presentados en este y los capítulos siguientes.

Modelo de lenguaje utilizado

Para la generación de texto a través de modelos de LLM, se utilizó el modelo LLaMA 3.1–8B-Instruct, desarrollado por Meta AI [71]. Este modelo forma parte de la familia de LLMs LLaMA (Large Language Model Meta AI), y corresponde a una versión optimizada para tareas de seguimiento de instrucciones (*instruct-tuning*). Consta de aproximadamente 8 mil millones de parámetros y ha sido preentrenado sobre un corpus multilingüe de alta calidad, seguido de un fine-tuning para mejorar la utilidad y seguridad en contextos conversacionales.

La elección de este modelo responde a una necesidad de equilibrio entre capacidad expresiva, eficiencia computacional y adaptabilidad a recursos de hardware limitados. Estudios recientes han mostrado que modelos de esta escala, cuando están debidamente utilizados, pueden alcanzar un rendimiento comparable al de modelos más grandes en tareas de pregunta-respuesta, razonamiento y generación de lenguaje controlado [44].

Cuando se tuvo que realizar alguna tarea de fine-tuning, se utilizo a este modelo (LLaMA 3.1–8B Instruct) como modelo base.

Servidor de inferencia

Para realizar la inferencia del modelo LLM, se utilizó el servidor vLLM [67], una arquitectura diseñada específicamente para acelerar la inferencia de modelos de lenguaje a gran escala. vLLM se basa en un enfoque de *paged attention*, una técnica que permite reutilizar eficientemente los *key-values* en memoria sin necesidad de recomputarlos en cada paso de generación. Esto resulta fundamental para tareas de generación paralela, donde múltiples entradas deben procesarse de forma simultánea, como es el caso de la indexación de *chunks* y resúmenes en el *pipeline* de GraphRAG.

Una de las ventajas clave de vLLM en este proyecto fue su capacidad para manejar llamadas concurrentes al modelo con mínimos costos de latencia, lo que permitió acelerar significativamente la generación de resúmenes y preguntas en grandes volúmenes. Además, al ser compatible con el cliente de OpenAI, resulta idóneo para integrarse con el *framework* de GraphRAG, y su interfaz facilita su utilización posterior en el desarrollo.

La elección de vLLM también respondió a la necesidad de optimizar el uso de memoria VRAM en una GPU limitada (RTX 3090 de 24 GB), maximizando el *throughput*.

Como servidor de Inferencia para el modelo de embeddings, se utilizo Ollama.

Cuantización del modelo

Para asegurar la viabilidad de ejecutar el modelo LLaMA 3.1–8B Instruct en una GPU con 24 GB de VRAM (RTX 3090) en el framework de vLLM, se recurrió a un modelo previamente cuantizado mediante la técnica AWQ (*Activation-aware Weight Quantization*) a 4 bits, descargado desde de Hugging Face (hugging-quants/Meta-Llama-3.1-8B-Instruct-AWQ-INT4). Esta técnica permite reducir significativamente el tamaño del modelo y su requerimiento de memoria sin afectar de forma sustancial la calidad de las respuestas generadas.

AWQ se basa en una estrategia de cuantización post-entrenamiento que preserva las activaciones más relevantes del modelo al momento de decidir los rangos de cuantización, lo que mejora la estabilidad y el rendimiento frente a métodos de cuantización más simples como la cuantización uniforme a int8 [43]. Al centrarse en preservar la distribución de

activaciones críticas, AWQ logra mantener la fidelidad del modelo incluso bajo formatos de 4 bits, permitiendo un uso más eficiente de la memoria y mayor *throughput* de inferencia.

Una de las ventajas clave de esta elección fue su compatibilidad nativa con el servidor vLLM, que ofrece soporte completo para modelos AWQ. Esto evitó la necesidad de conversiones adicionales o ajustes manuales en el *pipeline* de inferencia, facilitando una integración fluida y eficiente del modelo cuantizado en todo el sistema de generación.

Generador de embeddings

Para la representación vectorial de los textos en el *pipeline* de GraphRAG, tanto en la etapa de indexación como en la comparación semántica de preguntas y recuperacion de documentos, se utilizó el modelo `nomic-embed-text`, desarrollado por Nomic AI [26]. Este modelo fue seleccionado por su excelente rendimiento en tareas de recuperación semántica, evaluado en *benchmarks* estándar como MTEB (*Massive Text Embedding Benchmark*) [51], donde se posiciona como uno de los mejores modelos *open-source* de generación de *embeddings* para textos en inglés.

El modelo `nomic-embed-text` fue entrenado específicamente para capturar similitudes semánticas a nivel de documento y frase, lo que lo hace especialmente adecuado para tareas de *retrieval*. Su arquitectura se basa en una variante optimizada del *encoder* BERT-like con *pooling* por *token* [CLS], y es capaz de proyectar entradas de texto de longitud variable en un espacio vectorial denso y consistente, de manera eficiente y estable.

Base de datos vectorial

Para el almacenamiento y la recuperación eficiente de *embeddings* generados durante el *pipeline* de KRAQ, se utilizó Qdrant, un motor de búsqueda vectorial optimizado para consultas por similitud semántica. Qdrant permite realizar búsquedas aproximadas en espacios vectoriales de alta dimensión mediante algoritmos eficientes como HNSW (*Hierarchical Navigable Small World*), y soporta operaciones complejas como filtrado condicional y metadatos adjuntos a los vectores [28]. Esto lo convierte en una herramienta ideal para sistemas RAG donde se requiere recuperar documentos relevantes a partir de *queries* embebidas.

En el contexto de esta tesis, se desplegó una instancia local de Qdrant para garantizar control total sobre la persistencia de los vectores, optimizar el rendimiento en entornos con recursos limitados, y evitar dependencias externas. La base de datos almacenó tanto los *embeddings* de los *chunks* documentales como los *embeddings* de las preguntas generadas, permitiendo realizar comparaciones semánticas durante la evaluación.

Tamaños de datasets

Para la realización de los experimentos descritos en esta tesis, se procedió a un ajuste en el tamaño de los datasets empleados. Específicamente, los corpus originales fueron submuestreados de manera que, tras el proceso de fragmentación (*chunking*) implementado en el *pipeline* de GraphRAG, se obtuviera un volumen de aproximadamente 15,000 *chunks* por dataset. Cada uno de estos *chunks* fue configurado para tener una longitud máxima de 300 *tokens*.

Esta reducción controlada del tamaño de los datos fue una consideración pragmática, impuesta por las limitaciones temporales y los recursos computacionales disponibles en el

marco de esta tesis de licenciatura. El objetivo principal de este ajuste fue asegurar la viabilidad de ejecutar la totalidad de los experimentos y las correspondientes evaluaciones en un plazo razonable, permitiendo así una exploración exhaustiva de las metodologías propuestas.

Por lo tanto, el tamaño final del conjunto de preguntas seleccionadas para cada dataset se definió de modo que la totalidad de los documentos de evidencia asociados sumaran aproximadamente **5.000.000** de tokens por corpus. El número de preguntas de referencia utilizadas para esta evaluación de KRAQ fue:

- **TriviaQA:** 300 preguntas de referencia.
- **HotPotQA:** 2000 preguntas de referencia.
- **BioASQ:** 1000 preguntas de referencia.
- **PubHealth:** 2400 preguntas de referencia (afirmaciones).

El número de preguntas utilizadas para la evaluación de los sistemas RAG (Combined Retrieve RAG y Efficient Speculative RAG) varía y se especificará en las secciones correspondientes a cada uno, reflejando también las limitaciones temporales para la ejecución de dichos experimentos.

3.2.3. Diseño de evaluación para KRAQ

La evaluación de la calidad de las preguntas generadas por KRAQ se centra en su capacidad para representar semánticamente las posibles consultas que un usuario podría formular sobre un corpus. Para cuantificar esto, se adoptó un enfoque basado en métricas de similitud semántica, ya que las métricas tradicionales basadas en N-gramas (como BLEU o ROUGE) presentan limitaciones importantes en el contexto de QG. Estas últimas penalizan formulaciones sintáctica o léxicamente diferentes pero semánticamente correctas y relevantes, lo que a menudo resulta en una baja correlación con la evaluación humana en tareas donde la creatividad y la variación son deseables [25].

En esta tesis, optamos por **BERTScore** [88], una métrica que utiliza *embeddings* contextuales para calcular la similitud semántica entre la pregunta generada y una de referencia, superando las limitaciones de la simple coincidencia léxica. Específicamente, se utiliza el F1 de BERTScore, que combina precisión y recall. Este enfoque permite cuantificar la relevancia y la cobertura semántica de las preguntas de KRAQ en relación con un conjunto de preguntas de referencia existentes en *datasets* estándar, sin exigir una correspondencia léxica exacta.

Es crucial destacar una distinción metodológica fundamental de KRAQ respecto a muchos trabajos previos en QG. Un número considerable de sistemas de QG evaluados con métricas tradicionales operan bajo un paradigma *answer-aware*, incorporando la respuesta objetivo A como entrada para generar la pregunta Q (Ecuación 2.3). En contraste, KRAQ adopta un paradigma *answer-unaware*, operando únicamente a partir del corpus $\mathcal{X}$ sin requerir respuestas predefinidas. Esta elección deliberada responde a la necesidad de aplicabilidad en escenarios realistas donde no siempre se dispone de pares (pregunta, respuesta) exhaustivos. Por ello, métricas que presuponen un modelo *answer-aware* son menos pertinentes para evaluar la contribución central de KRAQ. Nuestro enfoque de evaluación,

centrado en la cobertura semántica de preguntas relevantes (sin asumir que KRAQ conozca sus respuestas) y en el impacto funcional en tareas *downstream*, favorece la valoración de la flexibilidad y utilidad práctica del sistema.

Adicionalmente, dado que el propósito fundamental del conjunto de preguntas generado por KRAQ es optimizar los sistemas RAG, los resultados de los experimentos de aplicación (Combined Retrieve RAG y Efficient Speculative RAG, descritos en las Secciones 4.2 y 5.2.7) funcionan como una métrica indirecta de la utilidad de KRAQ. Una generación de preguntas efectiva por parte de KRAQ debería traducirse en mejoras observables en la precisión o eficiencia de dichos sistemas RAG (como se analiza en la Sección 5.2.9).

Para llevar a cabo esta evaluación, se utilizaron los datasets estándar previamente descritos (Sección 3.2.1), donde cada instancia provee una tripleta (Q, A, E). En esta estructura, Q es un conjunto de preguntas de referencia formuladas por humanos, A sus correspondientes respuestas correctas, y E la colección de documentos o evidencia textual que conforma el corpus del dataset y sobre la cual opera KRAQ. El protocolo de evaluación de KRAQ, diseñado para medir su capacidad de cubrir semánticamente estas preguntas Q, se desarrolla de la siguiente manera:

1. Inicialmente, el pipeline completo de KRAQ procesa el corpus de evidencia E para generar un conjunto de preguntas representativas, denotado como $\mathcal{Q}^{\mathrm{K}} = \{Q_1^{\mathrm{K}}, Q_2^{\mathrm{K}}, \ldots, Q_k^{\mathrm{K}}\}$. Cada pregunta $Q_i^{\mathrm{K}} \in \mathcal{Q}^{\mathrm{K}}$ es generada a partir de una comunidad semántica distinta, detectada previamente en el grafo de conocimiento construido sobre E.

2. Posteriormente, se establece una correspondencia entre las preguntas generadas $\mathcal{Q}^{\mathrm{K}}$ y el conjunto de preguntas de referencia del dataset, $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_N\}$. Para cada pregunta de referencia $Q_j \in \mathcal{Q}$, se identifica su contraparte generada más similar, $Q_j^* \in \mathcal{Q}^{\mathrm{K}}$, mediante la maximización de la similitud coseno entre sus respectivas representaciones vectoriales (embeddings). Formalmente:

$$Q_j^* = \arg\max_{Q_i \in \mathcal{Q}^{\mathrm{K}}} \left(\cos(\mathrm{emb}(Q_j), \mathrm{emb}(Q_i))\right) \tag{3.1}$$

   donde $\mathrm{emb}(\cdot)$ es la función que proyecta una pregunta a su embedding, y $\cos(\cdot)$ es la funcion que calcula la similitud coseno entre dos embeddings.

A partir de estos pares (Q_j, Q_j^*), se procede a evaluar la calidad de la generación utilizando dos métricas complementarias, ambas fundamentadas en BERTScore.

Métrica 1: Cobertura semántica mediante BERTScore

La primera métrica tiene como objetivo evaluar la fidelidad con la que el contenido semántico de una pregunta de referencia Q_j se encuentra reflejado en su correspondiente pregunta generada más similar Q_j^*. Para cada par (Q_j, Q_j^*), se calcula el BERTScore (específicamente, la puntuación F1 de BERTScore, que combina precisión y recall):

$$\mathrm{BERTScore}(Q_j, Q_j^*)$$

El resultado final de esta métrica se obtiene promediando las puntuaciones de BERTScore sobre todas las N preguntas de referencia del dataset.

$$\mathrm{Relevance} = \frac{1}{N} \sum_{j=1}^{N} \mathrm{BERTScore}(Q_j, Q_j^*)$$

Esta métrica proporciona una medida de la cercanía semántica global, siendo robusta a variaciones léxicas o estructurales entre las preguntas comparadas.

Métrica 2: Umbral de relevancia semántica (Relevance@τ)

La segunda métrica propuesta es de naturaleza binaria y busca estimar la utilidad práctica de las preguntas generadas por KRAQ. Se considera que una pregunta generada Q_j^* es relevante como sustituto o representante de Q_j si el BERTScore entre ellas supera un umbral predefinido τ (e.g., $\tau = 0{,}75$). Esta métrica permite cuantificar la proporción de preguntas generadas por KRAQ que alcanzan un grado aceptable de alineación semántica con las preguntas de referencia.

La métrica se define como la proporción de pares (Q_j, Q_j^*) que satisfacen esta condición:

$$\text{Relevance@}\tau = \frac{1}{N}\sum_{j=1}^{N} \mathbb{I}\left[\text{BERTScore}(Q_j, Q_j^*)) \geq \tau\right]$$

donde N es el número total de preguntas de referencia en $\mathcal{Q}$, $\mathbb{I}[\cdot]$ es la función indicadora (que toma valor 1 si la condición entre corchetes es verdadera, y 0 en caso contrario).

La aplicación conjunta de estas dos métricas permite una evaluación eficiente, determinando si las preguntas generadas por KRAQ logran una cobertura semántica completa y un grado de relevancia suficiente respecto a las preguntas relevantes que pueden formularse sobre el corpus analizado.

Baseline

Para contextualizar el rendimiento de KRAQ, se implementó un baseline que simula una estrategia de generación de preguntas más directa, sin el análisis estructural profundo ni el conocimiento global del corpus que caracteriza a KRAQ. Este enfoque genera preguntas a partir de un número variable de *chunks* seleccionados aleatoriamente del corpus, combinados para formar un contexto a partir del cual un modelo de lenguaje formula una pregunta.

El procedimiento es el siguiente:

1. Se elige un número entero aleatorio m dentro de un rango predefinido $[m_1, m_2]$. Este valor m determinará cuántos *chunks* se utilizarán para generar la pregunta actual.

2. Se seleccionan m *chunks* $C_1^{(i)}, \ldots, C_m^{(i)}$ de manera uniforme y aleatoria del corpus E.

3. Estos m *chunks* se concatenan para formar un único contexto $C_{\text{input}}^{(i)}$.

4. Se utiliza este contexto $C_{\text{input}}^{(i)}$ para que un modelo de lenguaje genere una pregunta Q_i^{b}.

5. El proceso se repite K veces (donde K es el número de preguntas generadas por KRAQ en el mismo corpus) para construir el conjunto de preguntas del baseline, denotado como $\mathcal{Q}^{\text{b}} = \{Q_1^{\text{b}}, Q_2^{\text{b}}, \ldots, Q_K^{\text{b}}\}$.

Algorithm 3 Baseline: Generación de preguntas a partir de un número variable de fragmentos aleatorios

Require: Corpus E, rango de número de chunks $[m_1, m_2]$, número de preguntas K, modelo LLM LLM

Ensure: Conjunto de preguntas generadas $\mathcal{Q}^{\mathrm{b}}$

$\mathcal{Q}^{\mathrm{b}} \leftarrow \emptyset$
for $i = 1$ **to** K **do**
  $m \leftarrow \mathrm{RandomInt}(m_1, m_2)$
  $E^{(i)} \leftarrow \mathrm{SampleRandomChunks}(E, m)$
  $C^{(i)}_{\mathrm{input}} \leftarrow \mathrm{Concatenate}(E^{(i)})$
  $Q^{\mathrm{b}}_i \leftarrow \mathrm{LLM}(\mathrm{prompt}_{\mathrm{QG}}, C^{(i)}_{\mathrm{input}})$
  $\mathcal{Q}^{\mathrm{b}} \leftarrow \mathcal{Q}^{\mathrm{b}} \cup \{Q^{\mathrm{b}}_i\}$
return $\mathcal{Q}^{\mathrm{b}}$

Ver el prompt para generar las preguntas $\mathrm{prompt}_{\mathrm{QG}}$ en Apéndice 7.3. En la implementación realizada para esta tesis, se utilizaron los valores $m_1 = 3$ y $m_2 = 7$ para definir el rango del número de *chunks* aleatorios a concatenar.

Este baseline, al operar sobre muestras aleatorias locales de tamaño variable, puede generar preguntas coherentes pero frecuentemente redundantes, sin una estrategia para cubrir adecuadamente las distintas áreas temáticas del corpus de manera sistemática.

Comparar KRAQ con este baseline permite aislar el valor agregado por el análisis estructurado inherente a KRAQ, que incluye la construcción del grafo de conocimiento, la detección de comunidades semánticas y la generación de resúmenes representativos antes de la formulación de preguntas.

Cabe destacar que muchas líneas base alternativas de tipo Graph-to-Text no son comparables directamente, ya que típicamente generan texto a partir de tripletas o subgrafos condicionados explícitamente por una respuesta conocida. En nuestro caso, el objetivo es formular preguntas abiertas sin información de respuesta, por lo que esos métodos no resultan adecuados como punto de comparación directa (Ver Sección 2.3).

3.2.4. Configuración de GraphRAG

Para la construcción del grafo de conocimiento, la ejecución del clustering semántico y la posterior generación de resúmenes por comunidad, se empleó el *framework* GraphRAG, desarrollado por Microsoft [19]. GraphRAG hizo su framework OpenSource lo que facilita y permite su utilización [49].

La elección de GraphRAG se fundamenta en su capacidad para ejecutar estos procesos de manera controlada y sistemática. La utilización de este *framework* establecido contribuye a la replicabilidad y al rigor científico del método KRAQ propuesto, al tiempo que se aprovechan las validaciones y el consenso existentes en la comunidad científica en torno a GraphRAG. Para los propósitos de esta tesis, el *framework* fue adaptado con el fin de optimizar tanto el rendimiento computacional como la compatibilidad con los recursos de hardware disponibles.

La totalidad de los experimentos descritos en este trabajo se llevaron a cabo utilizando una única unidad de procesamiento gráfico (GPU) NVIDIA GeForce RTX 3090, equipada con 24 GB de VRAM.

Parámetros elegidos

Con el objetivo de adaptar la ejecución de GraphRAG a los datasets y recursos mencionados, se realizaron ajustes específicos sobre los parámetros base del *framework*. A continuación, se detallan las configuraciones más relevantes modificadas.
Configuración de (*chunks*): La fragmentacion del corpus de entrada se configuró con los siguientes parámetros:

- `size`: 300 (*tokens* por *chunk*).
- `overlap`: 50 (*tokens* de solapamiento entre *chunks* consecutivos).

Configuración del modelo del lenguaje en GraphRAG: Para las tareas de extracción y resumen dentro de GraphRAG, se especificó el siguiente modelo y parámetros de inferencia:

- `model`: `hugging-quants/Meta-Llama-3.1-8B-Instruct-AWQ-INT4` (refiriéndose al modelo LLaMA 3.1 8B Instruct cuantizado con AWQ, previamente discutido en la Sección 3.2.2).
- `concurrent_requests`: 15 (número de solicitudes paralelas permitidas al servidor del modelo, ajustado para maximizar el *throughput* (tokens generados por segundo) en la GPU disponible).

Obtención de resúmenes comunitarios desde GraphRAG

Una vez ejecutado el *pipeline* de GraphRAG sobre un corpus, el *framework* genera diversos archivos que contienen el conocimiento estructurado extraído y los análisis realizados sobre el grafo. Para los propósitos de esta tesis, el archivo de principal interés que se utiliza en etapas posteriores es el archivo denominado `community_reports.parquet` que se encuentra en la carpeta `output` al correr el CLI de GraphRAG.

Este archivo, almacenado en formato Apache Parquet para un manejo eficiente de datos tabulares, contiene una representación detallada de todas las comunidades semánticas identificadas por GraphRAG. Para cada comunidad detectada en los diferentes niveles jerárquicos del grafo, el archivo `community_reports.parquet` incluye tanto el resumen textual como una lista de observaciones llamadas *findings* que destacan hechos o puntos claves dentro de la información contenida en la comunidad.

Si bien en la presente tesis el sistema KRAQ se enfoca en utilizar los resúmenes textuales comunitarios como base para la generación de preguntas (ver Sección 3.1.5), la existencia de estos *findings* detallados por comunidad abre una vía interesante para trabajos futuros. Se podría explorar:

- Utilizar directamente los *findings* (o una selección de ellos) como una forma alternativa o complementaria de resumen para la generación de preguntas, lo que podría llevar a preguntas más granulares o específicas.
- Emplear los *findings* para enriquecer el *prompt* enviado al modelo generador de preguntas, proporcionando así un contexto adicional más allá del resumen narrativo, con el objetivo de mejorar la calidad o diversidad de las preguntas generadas.

No obstante, para el alcance actual, el resumen comunitario principal es el objeto fundamental que se extrae de GraphRAG para el *pipeline* de KRAQ.

Optimización de prompts en graphRAG

El *framework* GraphRAG ofrece la capacidad de adaptar sus *prompts* para dominios específicos, con el fin de mejorar su alineación con las características particulares de un corpus. Esta optimización puede realizarse mediante una funcionalidad integrada que ajusta los *prompts* de manera automática llamada **PromptTuning**. Para evaluar el impacto de esta funcionalidad en el contexto de la presente tesis, se compararon las dos modalidades de configuración de *prompts*:

Modalidad sin Ajuste de Prompts (NoPromptTuning): En esta configuración, se emplearon los *prompts* predeterminados que provee GraphRAG. Estos *prompts* están diseñados para funcionar bien en una amplia variedad de un ajuste adicional específico al dominio.

Modalidad con Ajuste de Prompts (PromptTuning): Esta modalidad explora la capacidad de GraphRAG para optimizar sus *prompts* internos. En particular, el sistema parte del muestreo de un subconjunto de *chunks*. Luego, se analiza el contenido mediante un *LLM* para identificar los tipos de entidades y demas caracteristicas del corpus, lo cual permite adaptar los *prompts* de manera automatica. [19].

Para determinar la configuración más adecuada para los experimentos siguientes de esta tesis, se realizó un estudio comparativo entre estas dos modalidades utilizando el dataset BioASQ. Se usaron las metricas explicadas para KRAQ (ver Sección 3.2.3). Los resultados obtenidos se presentan en la Tabla 3.1.

Tab. 3.1: Comparación de metricas sobre KRAQ con y sin ajuste de prompts (*PromptTuning*) en GraphRAG sobre el dataset BioASQ.

Métrica de Relevancia	***NoPromptTuning***	***PromptTuning***
Relevance	**79.0**	78.8
Relevance@0.70	**93.1**	92.9
Relevance@0.75	73.8	**74.8**
Relevance@0.80	**42.6**	40.6

A partir de los resultados presentados en la Tabla 3.1, se observa que la modalidad sin ajuste de *prompts* (*NoPromptTuning*) ofrece un rendimiento ligeramente superior o comparable en la mayoría de las métricas de relevancia evaluadas. Aunque la modalidad con ajuste (*PromptTuning*) muestra una leve ventaja en el umbral de Relevance@0.75, las diferencias generales no son significativas y, en los demas casos, el no ajuste resulta marginalmente mejor.

Considerando estos hallazgos y la complejidad adicional que implica el proceso de *PromptTuning*, se concluyo que, para los fines de esta tesis, la utilización de los *prompts* predeterminados ofrecía un balance más favorable entre rendimiento y simplicidad metodológica. Por consiguiente, para el resto de los experimentos detallados que involucran GraphRAG, se optó por la modalidad sin ajuste de *prompts* (*NoPromptTuning*).

Los prompt predeterminados de GraphRag pueden encontrarse en el Apendice E de [19]

3.2.5. Modelo generador de preguntas

Tras la identificación de comunidades semánticas y la generación de resúmenes para cada una de ellas mediante el *pipeline* de GraphRAG (como se detalló en la Sección 3.2.4), la siguiente etapa del sistema KRAQ consiste en transformar estos resúmenes en un conjunto de preguntas representativas. El objetivo de esta fase es derivar, a partir de cada resumen comunitario, una pregunta que capture la esencia del mismo y que pueda actuar como *proxy* de las posibles intenciones de búsqueda de un usuario interesado en el corpus.

Este proceso de generación de preguntas se fundamenta en la utilización de un LLM específicamente *fine-tuneado*. Dicho modelo está entrenado para realizar un mapeo desde un resumen textual de entrada (proveniente de una comunidad semántica) hacia una pregunta significativa y contextualmente relevante. La pregunta generada busca encapsular la interrogante más probable o pertinente que un individuo formularía con respecto a la información contenida en ese resumen específico. En las subsecciones siguientes, se describirá el proceso de fine-tuning de este modelo generador y se analizarán los resultados obtenidos.

Fine-tuning del modelo

Como se introdujo en la Sección 3.1.5, la generación de preguntas representativas a partir de los resúmenes comunitarios se basa en un modelo de lenguaje f_θ específicamente entrenado para esta tarea. Este proceso de *fine-tuning*, tiene como objetivo entrenar un modelo LLM para que, dado un resumen R , pueda generar una pregunta Q^K que sea semántica y contextualmente relevante para dicho resumen.

Dataset y metodología de entrenamiento. La generación de los pares de entrenamiento (R, Q) para el ajuste fino del modelo f_θ se fundamentó en la utilización de dos datasets externos, reconocidos en el ámbito de QA: Dolly-v2 y MusiQue. La selección de estos datasets específicos respondió a la necesidad de emplear corpus distintos a aquellos que se utilizarían posteriormente para la evaluación final del sistema KRAQ, evitando así cualquier posible sesgo o sobreajuste a los datos de prueba.

El dataset Dolly-v2 [13] esta compuesto por aproximadamente 15,000 ejemplos de seguimiento de instrucciones generados por humanos, resulta idóneo por su diversidad y calidad. Por su parte, MusiQue [73] es un dataset diseñado para evaluar el razonamiento multi-salto en tareas de QA, proporcionando preguntas complejas que requieren la integración de información de múltiples fuentes. De ambos datasets, se extrajeron las tripletas (Q, A, E) (pregunta de referencia, respuesta, evidencia) que sirvieron como base para la síntesis de los resúmenes R y la conformación de los pares (R, Q) para el entrenamiento de nuestro modelo generador de preguntas.

1. **Síntesis de resúmenes (g):** Para cada instancia (Q, E) de los datasets, se utilizó un modelo LLM (GPT-4o) en un esquema *few-shot* para generar un resumen temático $R = g(Q, E)$. Se instruyó al modelo para que resumiera los conceptos centrales de la evidencia E sin hacer referencia explícita a Q, produciendo así un resumen general del contenido de una supuesta comunidad semantica relacionada. El *prompt* específico utilizado para esta síntesis de resúmenes se detalla en el Apéndice 7.1.

2. **Fine-tuning del generador (f_θ):** Con los pares (R, Q) generados, se procedió al finetuning del modelo LLaMA 3.1–8B-Instruct. El objetivo fue maximizar la verosimilitud logarítmica de predecir la pregunta original Q dado el resumen R, $\mathbb{E}_{(R,Q)} [\log P_\theta(Q \mid R)]$.

Este enfoque permite al modelo aprender a inferir las preguntas más probables y significativas que podrían surgir naturalmente de un resumen temático, como los generados para las comunidades. El *prompt* diseñado para el proceso de *fine-tuning* se detalla en el Apéndice 7.2. Cabe destacar que esta misma estructura de *prompt* se mantuvo para la generación de preguntas durante la fase de inferencia de KRAQ.

Configuración y parámetros del entrenamiento. El proceso de *fine-tuning* se llevó a cabo utilizando la técnica QLoRA (*Quantized Low-Rank Adaptation*) para una adaptación eficiente en memoria del modelo LLaMA 3.1–8B Instruct. A continuación, se resumen los hiperparámetros y configuraciones clave empleadas.

- **Modelo base:** `meta-llama/Llama-3.1-8B-Instruct`.
- **Cuantización (BitsAndBytes):** Carga en 4-bit (`load_in_4bit=True`), tipo de cómputo `torch.float16`, doble cuantización (`bnb_4bit_use_double_quant=True`), tipo de cuantización `nf4`.
- **Configuración PEFT (LoRA):** Rango (`r=64`), alfa LoRA (`lora_alpha=16`), dropout LoRA (`lora_dropout=0.05`), `bias=''none''`, tipo de tarea `CAUSAL_LM`.
- **Dataset:** Partición 90 % entrenamiento / 10 % evaluación.
- **Argumentos de entrenamiento (Principales):**
  - Lote por dispositivo (entrenamiento y evaluación): 2.
  - Pasos de acumulación de gradiente: 4.
  - Épocas de entrenamiento: 3.
  - Tasa de aprendizaje: 2×10^{-4}.
  - Precisión mixta: `bf16=True`.
  - Optimizador: `paged_adamw_8bit`.
  - Longitud máxima de secuencia: 2048 *tokens*.

El entrenamiento se realizó sobre una GPU NVIDIA GeForce RTX 3090 con 24 GB de VRAM.

Resultados del entrenamiento. El proceso de finetuning del modelo generador de preguntas mostró una convergencia estable. La Figura 3.6 ilustra la curva de pérdida (*loss*) durante el entrenamiento sobre el conjunto de validación. Se observa una rápida disminución inicial de la pérdida, seguida de una estabilización alrededor de un valor de 0.65-0.70, lo que indica que el modelo aprendió efectivamente a mapear los resúmenes a las preguntas correspondientes. La tendencia suavizada (ventana de 50 pasos) confirma esta estabilización.

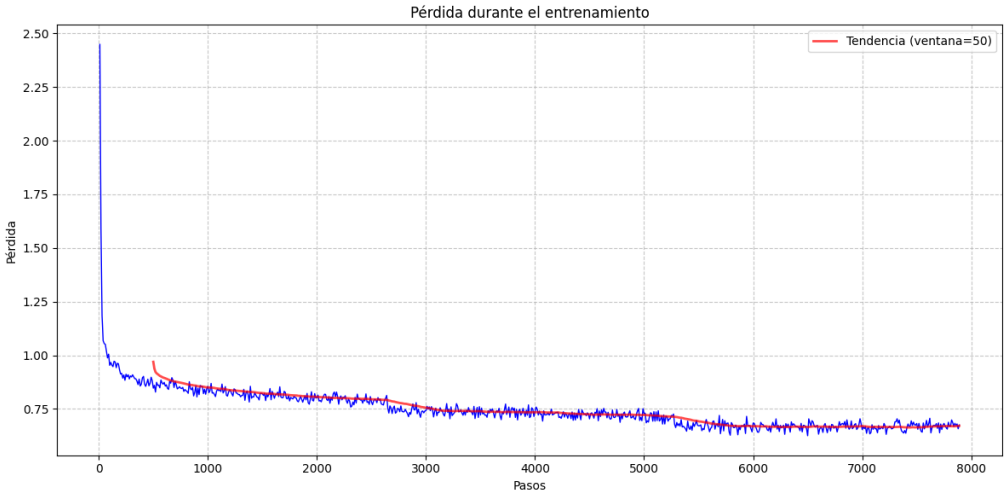

Fig. 3.6: Curva de pérdida en el conjunto de validación durante el proceso de finetuning del modelo generador de preguntas de KRAQ. La línea azul muestra la pérdida por paso y la línea roja una tendencia suavizada con una ventana de 50 pasos.

Como resultado final del *pipeline* completo de KRAQ, se obtiene un listado de preguntas representativas $\{Q_1^K, Q_2^K, \ldots, Q_k^K\}$, una por cada comunidad semántica detectada en el grafo de conocimiento (considerando todos los niveles jerárquicos de comunidades).

3.2.6. Complejidad Computacional de KRAQ

La complejidad computacional del *pipeline* completo de KRAQ está dominada en gran medida por las etapas iniciales gestionadas por el *framework* GraphRAG, específicamente la extracción de entidades y relaciones, la construcción del grafo de conocimiento y la detección jerárquica de comunidades. Estas operaciones, si bien intensivas, se realizan una única vez por corpus durante una fase de preprocesamiento.

Una vez que se han identificado las comunidades semánticas y se han generado sus respectivos resúmenes textuales (como se describe en la Sección 3.2.4), la etapa final de KRAQ, que es la generación de una pregunta representativa por cada resumen comunitario, presenta una complejidad que es lineal con respecto al número total de comunidades detectadas en todos los niveles jerárquicos. Es decir, por cada resumen de comunidad R_i obtenido de GraphRAG, se realiza una única inferencia con el modelo de lenguaje f_θ (ajustado como se detalla en la Sección 3.2.5) para producir la pregunta Q_i^K. Si N_c es el número total de comunidades identificadas, la generación de preguntas requerirá N_c inferencias del LLM.

El número de comunidades y, por lo tanto, el número de preguntas generadas, depende de la estructura inherente del corpus, su tamaño, la diversidad temática y los parámetros del algoritmo de detección de comunidades (e.g., el algoritmo Leiden). Para los *datasets* utilizados en esta tesis, cada uno compuesto por aproximadamente 5 millones de tokens (resultando en unos 15,000 *chunks* de 300 tokens, como se describe en la Sección 3.2.2), se observó la siguiente cantidad de preguntas generadas (equivalente al número de comunidades detectadas en todos los niveles):

- **TriviaQA:** 17,378 preguntas.

- **PubHealth:** 10,412 preguntas.
- **HotPotQA:** 25,660 preguntas.
- **BioASQ:** 19,100 preguntas.

A modo ilustrativo, la distribución de las preguntas generadas por KRAQ para el *dataset* HotPotQA a través de los diferentes niveles jerárquicos de comunidades se presenta en la Tabla 3.2. Los niveles más bajos (e.g., Nivel 0, Nivel 1) representan comunidades más amplias y generales, mientras que los niveles superiores (e.g., Nivel 3, Nivel 4) corresponden a comunidades más pequeñas, específicas y granulares.

Tab. 3.2: Distribución de preguntas generadas por KRAQ (y, por ende, comunidades detectadas) por nivel jerárquico para el dataset HotPotQA.

Nivel Jerárquico	Número de Preguntas/Comunidades
0	111
1	2,392
2	11,468
3	10,717
4	961
5	11
Total	**25,660**

La capacidad de KRAQ para generar un conjunto tan extenso y jerarquizado de preguntas representativas es una de sus características distintivas. Este conjunto no solo busca una cobertura semántica amplia, sino que también ofrece preguntas a diferentes niveles de granularidad temática, lo que podría ser explotado en futuras aplicaciones avanzadas de RAG que requieran un entendimiento contextual a múltiples escalas.

3.2.7. Resultados de KRAQ

Tras haber detallado el protocolo experimental para la evaluación de KRAQ (Sección 3.2.3) y el proceso completo de generación de sus preguntas representativas, desde la construcción del grafo de conocimiento (Sección 3.2.4) hasta el finetuning del modelo generador (Sección 3.2.5), esta sección se enfoca en presentar los hallazgos empíricos.

El rendimiento de KRAQ se evaluará comparándolo con dos alternativas:

- **KRAQ-Instruct:** Para aislar el impacto del *fine-tuning* específico realizado en KRAQ, se considera esta variante. KRAQ-Instruct sigue el mismo *pipeline* de GraphRAG para obtener los resúmenes comunitarios, pero, para la generación de preguntas a partir de estos, utiliza el modelo LLaMA 3.1–8B Instruct (sin el *fine-tuning* de la Sección 3.2.5).
- **Baseline Aleatorio (*Random*):** Este enfoque, ya descrito en la Sección 3.2.3, genera preguntas a partir de *chunks* de texto seleccionados aleatoriamente, sirviendo como punto de referencia para métodos sin análisis estructural profundo.

A continuación, se presentan los resultados cuantitativos de estas tres aproximaciones, utilizando las métricas de Relevance y Relevance@τ (con $\tau \in \{0{,}70, 0{,}75, 0{,}80\}$) definidas en el diseño de evaluación.

Resultados

Los resultados obtenidos para cada dataset y cada variante del sistema se presentan en la Tabla 3.3. Los valores indican el porcentaje de preguntas generadas que cumplen con los criterios de cada métrica.

Tab. 3.3: Comparación del rendimiento de KRAQ (con modelo *fine-tuned*), KRAQ-Instruct (modelo preentrenado) y el Baseline Aleatorio en la generación de preguntas representativas. Las métricas son Relevancia promedio (BERTScore F1) y Relevance@τ para diferentes umbrales τ.

Dataset	Métrica	KRAQ (Fine-tuned)	KRAQ-Instruct	Baseline Aleatorio
TriviaQA				
	Relevance	**78.1**	75.5	72.2
	Relevance@0.70	**93.0**	90.6	72.0
	Relevance@0.75	**71.0**	50.3	22.3
	Relevance@0.80	**33.0**	15.0	3.6
HotPotQA				
	Relevance	**74.2**	72.8	69.5
	Relevance@0.70	**84.0**	75.0	42.7
	Relevance@0.75	**40.4**	29.9	5.6
	Relevance@0.80	**10.0**	4.7	0.3
PubHealth				
	Relevance	**68.5**	68.0	66.7
	Relevance@0.70	**33.6**	30.3	15.6
	Relevance@0.75	**4.8**	3.4	1.1
	Relevance@0.80	**0.4**	0.26	0.03
BioASQ				
	Relevance	**79.0**	77.9	74.1
	Relevance@0.70	93.1	**93.6**	84.3
	Relevance@0.75	**73.8**	70.8	42.7
	Relevance@0.80	**42.6**	34.9	8.6

Análisis de los resultados

La Tabla 3.3 evidencia consistentemente la superioridad del enfoque KRAQ, particularmente en su versión con el modelo generador de preguntas fine-tuneado, en comparación tanto con el uso del modelo instruct sobre los mismos resúmenes comunitarios (KRAQ-Instruct) como con el baseline de generación a partir de *chunks* aleatorios.

Impacto del ajuste fino (*Fine-tuning*): En la mayoría de los datasets y métricas, KRAQ con el modelo *fine-tuned* supera a KRAQ-Instruct. Por ejemplo, en TriviaQA, la Relevance aumenta de 75.5 % a 78.1 %, y la proporción de preguntas que superan un umbral de relevancia de 0.75 (Relevance@0.75) pasa de un 50.3 % a un notable 71.0 %. Similarmente, en HotPotQA, el incremento en Relevance@0.75 es de 29.9 % a 40.4 %. Estos resultados sugieren que el proceso de *fine-tuning*, al entrenar el modelo específicamente para mapear resúmenes comunitarios a preguntas relevantes, logra capturar de manera más efectiva la intención semántica subyacente en comparación con un modelo generalista. La única excepción notable se observa en BioASQ para Relevance@0.70, donde KRAQ-Instruct obtiene un rendimiento ligeramente superior (93.6 % vs 93.1 %), aunque las diferencias en otros umbrales y en la relevancia promedio para BioASQ siguen favoreciendo al modelo ajustado.

Superioridad sobre el baseline aleatorio: Ambas variantes de KRAQ demuestran una mejora sustancial sobre el baseline que genera preguntas a partir de *chunks* aleatorios. Este último, si bien puede producir preguntas localmente coherentes, falla en capturar la estructura temática global del corpus. Esto es particularmente evidente en las métricas de Relevance@τ con umbrales más altos. Por ejemplo, en TriviaQA para Relevance@0.75, KRAQ (*fine-tuned*) alcanza un 71.0 % mientras que el baseline aleatorio solo llega al 22.3 %. En HotPotQA, la diferencia es aún más pronunciada: 40.4 % para KRAQ frente a un escaso 5.6 % para el baseline. Esto subraya el valor de la estructuración del conocimiento mediante grafos y la generación de resúmenes comunitarios que realiza KRAQ antes de la etapa de generación de preguntas, asegurando una cobertura semántica más amplia y representativa.

Rendimiento en PubHealth: Es interesante notar que, si bien KRAQ sigue superando a los otros métodos en PubHealth, las mejoras y los valores absolutos de las métricas son consistentemente más bajos en comparación con los otros datasets. Por ejemplo, Relevance@0.75 para KRAQ (*fine-tuned*) es solo del 4.8 %. Esto puede atribuirse a la naturaleza particular de las "preguntas" de referencia en PubHealth, que originalmente son afirmaciones o *claims* que requieren verificación. La tarea de generar una pregunta a partir de un resumen temático podría no alinearse tan directamente con la formulación de un *claim* factual como lo hace con las preguntas más abiertas o de búsqueda de información de TriviaQA o HotPotQA. No obstante, incluso en este escenario más desafiante, la estructuración de KRAQ sigue ofreciendo un rendimiento superior al baseline.

Conclusiones del análisis: Los resultados validan empíricamente la efectividad de la arquitectura KRAQ para generar preguntas representativas que cubren semánticamente el contenido de un corpus. El proceso de *fine-tuning* del modelo generador de preguntas demuestra ser beneficioso, mejorando la calidad de las preguntas generadas a partir de los resúmenes comunitarios. La comparación con el baseline aleatorio resalta la importancia fundamental del análisis estructural y la síntesis temática que KRAQ introduce antes de la generación final de preguntas.

4. COMBINED RETRIEVE RAG

Este capítulo introduce y evalúa la primera aplicación práctica de las preguntas representativas generadas por KRAQ (descrito en el Capítulo 3): la estrategia de *Combined Retrieve RAG*. El objetivo principal de esta técnica es abordar la limitación de los sistemas RAG tradicionales en cuanto a la diversidad semántica de los documentos recuperados, proponiendo un método para enriquecer el contexto proporcionado al LLM y, consecuentemente, mejorar la precisión de las respuestas. Primero se detallará la metodología del algoritmo de recuperación combinada, y luego se presentará su evaluación experimental.

4.1. Metodologia de Combined Retrieve RAG

Los sistemas RAG tradicionales, si bien son efectivos, a menudo producen un conjunto de documentos recuperados que, aunque superficialmente similares a la consulta del usuario, pueden carecer de diversidad semántica y omitir facetas relevantes del conocimiento disponible. Para abordar esta limitación y mejorar la cobertura temática, esta tesis propone *Combined Retrieve RAG*, una estrategia de recuperación que integra el conjunto de preguntas representativas generadas por KRAQ. La idea central es utilizar estas preguntas precomputadas, derivadas de la estructura profunda del corpus, como consultas complementarias para enriquecer y diversificar el contexto proporcionado al LLM.

4.1.1. Algoritmo

La estrategia de Recuperación Combinada, denominada Combined Retrieve RAG, busca enriquecer el conjunto de documentos recuperados para una consulta de usuario Q. Para ello, aprovecha un conjunto precomputado de preguntas representativas $\mathcal{Q}^{\mathcal{K}} = \{Q_1^K, Q_2^K, \ldots, Q_k^K\}$, generadas previamente por el sistema KRAQ (como se detalló en el Capítulo 3). El objetivo es obtener un total de M documentos relevantes para la generación final de la respuesta.

El proceso se desarrolla en las siguientes etapas, formalizadas posteriormente en el Algoritmo 4:

1. **Selección de preguntas auxiliares de KRAQ:** Dada la consulta original Q, el primer paso consiste en identificar, dentro del conjunto $\mathcal{Q}^{\mathcal{K}}$, las n preguntas que exhiben la mayor similitud semántica con Q. Esta similitud se mide típicamente mediante la similitud coseno entre sus respectivas representaciones vectoriales (embeddings). El hiperparámetro n define cuántas de estas preguntas de KRAQ, que denominaremos Q_{sim}^K, se utilizarán como consultas complementarias.

2. **Distribución del esfuerzo de recuperación:** Se establece una proporción $\alpha \in (0, 1)$ para determinar cómo se distribuye la recuperación de los M documentos. Una porción de $\lfloor \alpha \cdot M \rfloor$ documentos se recuperará utilizando la consulta original Q. El presupuesto restante, $M' = M - \lfloor \alpha \cdot M \rfloor$, se asignará equitativamente entre las n preguntas seleccionadas de Q_{sim}^K, de modo que cada una de ellas buscará recuperar $\lfloor M'/n \rfloor$ documentos.

3. **Recuperación documental y consolidación:** La recuperación se efectúa mediante una función Retrieve(q, m, D), que retorna los m documentos más relevantes para una consulta q, asegurando que no se incluyan documentos previamente recuperados y listados en D.

   - Primero, se recuperan $\lfloor \alpha \cdot M \rfloor$ documentos utilizando la consulta original Q, sin exclusiones iniciales: $D_Q = \text{Retrieve}(Q, \lfloor \alpha \cdot M \rfloor, \emptyset)$. Estos forman el conjunto base de documentos.
   - Luego, para cada pregunta $Q_i^K \in Q_{\text{sim}}^K$, se recuperan $\lfloor M'/n \rfloor$ documentos adicionales: $D_{K_i} = \text{Retrieve}(Q_i^K, \lfloor M'/n \rfloor, D_Q \cup (\bigcup_{j<i} D_{K_j}))$. Es crucial que en cada una de estas recuperaciones complementarias se excluyan los documentos ya obtenidos tanto por Q como por las preguntas de KRAQ procesadas anteriormente, para garantizar la diversidad y evitar redundancias.
   - Todos los documentos únicos recuperados (D_Q y todos los D_{K_i}) se consolidan en un único conjunto final D_{total}.

4. **Generación de la respuesta final:** El conjunto consolidado D_{total} se concatena para formar el contexto enriquecido. Este contexto, junto con la consulta original Q, se proporciona a un LLM para generar la respuesta final al usuario, siguiendo el paradigma estándar de RAG [40].

Pseudocódigo

El procedimiento de Recuperación Combinada se formaliza en el Algoritmo 4.

Algorithm 4 Combined Retrieve RAG

Require: Consulta original Q, conjunto de preguntas generadas $\mathcal{Q}^K$, función Retrieve(q, m, D), número total de documentos M, proporción $\alpha \in (0, 1)$, número de preguntas similares n

Ensure: Respuesta generada A_{final}

$\mathcal{Q}_{\text{sim}}^K \leftarrow \text{TopNSimilar}(Q, \mathcal{Q}^K, n)$ ▷ n preguntas representativas mas similares
$m_{\text{main}} \leftarrow \lfloor \alpha \cdot M \rfloor$
$m_{\text{similar}} \leftarrow \lfloor (M - m_{\text{main}})/n \rfloor$
$D_{\text{total}} \leftarrow \text{Retrieve}(Q, m_{\text{main}}, \emptyset)$
for cada $Q_i^K \in \mathcal{Q}_{\text{sim}}^K$ **do**
    $D_i \leftarrow \text{Retrieve}(Q_i^K, m_{\text{similar}}, D_{\text{total}})$ ▷ Recupera documentos unicos
    $D_{\text{total}} \leftarrow D_{\text{total}} \cup D_{\text{i}}$
contexto $\leftarrow \text{ConcatenateDocuments}(D_{\text{total}})$
$A_{\text{final}} \leftarrow \text{GenerateAnswer}(Q, \textit{contexto})$
return A_{final}

4.2. Experimentación y resultados de Combined Retrieve RAG

En la presente sección se procede a evaluar el impacto de la integración de las preguntas representativas, generadas mediante el sistema KRAQ, dentro de un *pipeline* de RAG tradicional. Esta integración se realiza a través del enfoque de recuperación combinada,

denominado en este trabajo como Combined Retrieve RAG, cuya metodología fue detallada en la Sección 4.1.

El objetivo de este análisis experimental es determinar si la estrategia de Combined Retrieve RAG logra una mejora en la precisión de las respuestas generadas por el sistema RAG. Se hipotetiza que dicho incremento en la precisión se deriva del enriquecimiento en la diversidad del conjunto de documentos recuperados, al incorporar información complementaria obtenida a partir de preguntas similares generadas por KRAQ.

A continuación, se detallará la configuración específica de la implementación utilizada para esta evaluación y se analizarán los resultados obtenidos en términos de precisión de respuesta.

4.2.1. Diseño de evaluación

El protocolo de evaluacion de Combined Retrieve RAG se basa en la utilización de conjuntos de datos (datasets) estándar (explicados en la Sección 3.2.1)

Cada instancia de estos datasets se modela como una tripleta (Q, A, E), donde Q representa la pregunta de referencia formulada por un humano, A la respuesta considerada correcta o de referencia, y E el corpus de documentos o pasajes textuales asociados a dicha pregunta y respuesta.

Procesamiento y Generación de Respuestas

El flujo experimental para cada instancia (Q, A, E) del dataset comprende las siguientes etapas:

1. **Generación de preguntas representativas de KRAQ:** Como paso preliminar y fundamental, se aplica el pipeline completo de KRAQ sobre el corpus E correspondiente a cada dataset. Este proceso culmina con la generación de un conjunto de preguntas representativas $\mathcal{Q}^{\mathcal{K}} = \{Q_1^K, Q_2^K, \ldots, Q_k^K\}$, derivadas de la estructura semántica del corpus. Este conjunto $\mathcal{Q}^{\mathcal{K}}$ es específico para cada corpus E.

2. **Ejecución del algoritmo de recuperación combinada:** Posteriormente, para cada pregunta de referencia Q de una instancia del dataset, se ejecuta el algoritmo de Recuperación Combinada (descrito en la Sección 4.1). La pregunta Q se utiliza como la consulta original, y el conjunto $\mathcal{Q}^{\mathcal{K}}$ (generado en el paso anterior a partir del mismo corpus E) se emplea como el conjunto de preguntas representativas de KRAQ. La recuperación de documentos se realiza sobre el corpus completo E asociado al dataset. El resultado de este proceso es una respuesta generada, A_{gen}.

Baseline

Como baseline para Combined Retrieve RAG se utiliza el algoritmo de Traditional RAG explicado en la Sección 2.2

Métricas de Evaluación

La calidad de las respuestas generadas A_{gen} por los sistemas RAG se evaluó en relación con la pregunta original Q y la respuesta de referencia del dataset A. Se emplearon dos métricas.

Exact Match (EM). Exact Match (EM) es una medida estándar y rigurosa, ampliamente utilizada en tareas de pregunta-respuesta [66]. El EM verifica si la respuesta de referencia A esta exactamente en la respuesta generada A_{gen} después de un proceso de normalización textual. Este proceso de normalización es crucial para asegurar una comparación justa y típicamente incluye:

- Conversión de todo el texto a minúsculas.
- Eliminación de artículos (e.g., el, la, un, una).
- Eliminación de signos de puntuación.
- Estandarización de los espacios en blanco (e.g., múltiples espacios se reducen a uno solo, y se eliminan espacios al inicio y al final).

Una respuesta generada se considera correcta según EM solo si, tras esta normalización, la respuesta de referencia encuentra una coincidencia carácter por carácter en la respuesta de referencia normalizada. Aunque la métrica EM es considerablemente estricta y no otorga crédito parcial ni reconoce paráfrasis o reformulaciones semánticamente equivalentes, proporciona una estimación clara y precisa del grado de coincidencia literal entre la respuesta esperada y la generada. Su simplicidad y objetividad la hacen valiosa para comparar el rendimiento de los sistemas y es la metrica mas utilizada en la comunidad científica actualmente.

Evaluación con LLM como Juez. En algunos casos, la métrica de Exact Match resulta excesivamente estricta. Las respuestas de los datasets pueden admitir múltiples formulaciones correctas que, aunque semánticamente equivalentes, diferirían textualmente de la respuesta de referencia.

Por esta razón, la calidad de las respuesta generada A_{gen} tambien se evaluó mediante un enfoque de **Evaluación con LLM como Juez**. Inspirados por trabajos recientes que validan el uso de modelos de lenguaje de gran capacidad como evaluadores automáticos de la calidad de texto generado [46, 89], se empleó un LLM (Llama-3.1-8b-Instruct) para determinar si la respuesta generada A_{gen} aborda la pregunta Q de manera semánticamente equivalente y correcta en comparación con la respuesta de referencia A.

El procedimiento fue el siguiente:

Input al LLM-Juez:

- *Pregunta de Referencia (Q):* [Texto de Q del dataset]
- *Respuesta Generada por el Sistema (A_{gen}):* [Texto de A_{gen}]
- *Respuesta de Referencia (A_{gen}):* [Texto de A_{gen}]

Instrucción para el LLM-Juez:
Considerando la Pregunta de Referencia, ¿la Respuesta Generada por el Sistema responde a la pregunta de la misma manera y con la misma corrección factual que la Respuesta Ideal de Referencia? Responda únicamente con "Sí" o "No".

El prompt exacto utilizado se detalla en el Apéndice 7.7. La proporción de respuestas "Sí" se toma como la métrica de precisión. Este enfoque permite estimar la fidelidad semántica

y la corrección conceptual de la respuesta generada, trascendiendo las limitaciones de una simple comparación léxica superficial y siendo a veces más adecuado para respuestas de formato libre y explicativas.

Justificación del uso combinado. Aunque distintas en su enfoque, EM y LLM-as-Judge se complementan para ofrecer una evaluación más completa. EM aporta una medida objetiva y directa de coincidencia léxica, útil por su comparabilidad, pero limitada ante respuestas semánticamente correctas con distinta formulación. LLM-as-Judge, en cambio, permite valorar la equivalencia factual y semántica, captando mejor la comprensión y matices del modelo aunque este mas expuesto a riesgos debido a la alucinación de LLM. La combinación de ambas metricas permite reconocer tanto la precisión literal como la inteligencia semántica en las respuestas generadas.

4.2.2. Setup experimental y parámetros elegidos

Para evaluar la efectividad de la estrategia Combined Retrieve RAG, se configuró un *pipeline* de RAG cuyos componentes y parámetros se describen a continuación. El objetivo fue comparar el rendimiento de un sistema RAG tradicional con la variante enriquecida mediante las preguntas representativas generadas por KRAQ.
Componentes del Pipeline RAG:

Modelo Generador: Se empleó el modelo `llama3.1-8b-instruct` (en su versión cuantizada con AWQ), operando sobre el servidor vLLM. La configuración y justificación de esta elección se detallaron en las Seccion 3.2.2.

Retriever (Recuperador): El sistema de recuperación de información se basó en la similitud coseno entre representaciones vectoriales (*embeddings*). Específicamente, se utilizaron *embeddings* generados por el modelo `nomic-embed-text`, y la indexación y búsqueda vectorial se gestionaron mediante una instancia local de QDrant. Este componente se describió en la Sección 3.2.2.

Corpus Documental: Para la evaluación de Combined Retrieve RAG, se utilizó como base de conocimiento el mismo corpus de documentos de evidencia E (aproximadamente 5 millones de tokens por dataset, como se describe en la Sección 3.2.2) que fue procesado previamente por el sistema KRAQ para generar sus preguntas representativas. Las preguntas de referencia (Q) utilizadas para poner a prueba el sistema Combined Retrieve RAG se seleccionaron de los datasets originales, asegurándose de que los documentos de evidencia necesarios para responderlas estuvieran contenidos dentro de este corpus E. El número de preguntas de referencia de cada dataset utilizadas para la **evaluación específica de Combined Retrieve RAG** fue el siguiente:

- **TriviaQA:** 300 preguntas.
- **HotPotQA:** 300 preguntas.
- **BioASQ:** 1000 preguntas.
- **PubHealth:** 1000 preguntas (afirmaciones).

La selección de un número a veces limitado de preguntas para estos experimentos con sistemas RAG se debió, al igual que el submuestreo de los corpus, a consideraciones

pragmáticas sobre el tiempo disponible para la ejecución y análisis en el contexto de esta tesis, buscando obtener una evaluación indicativa del rendimiento.

Parámetros de recuperación para Combined Retrieve: La estrategia de Combined Retrieve opera con los siguientes parámetros clave:

- **Número total de documentos a recuperar (M):** Este valor, también conocido como *top-k* en la literatura, define la cantidad total de documentos que se pasarán al contexto del LLM generador. Para TriviaQA, HotPotQA y BioASQ se estableció $M = 15$; para PubHealth, $M = 10$. Esta diferenciación responde a las características intrínsecas de cada dataset.
- **Proporción de recuperación para la pregunta original (α):** Define la fracción de M documentos que se recuperarán utilizando la pregunta original del usuario (Q). Se utilizó $\alpha = 0{,}5$ para todos los datasets, asignando un 50 % del "presupuesto" de recuperación a la consulta original.
- **Número de preguntas representativas similares a utilizar (n):** Especifica cuántas preguntas de $\mathcal{Q}^{\mathrm{K}}$ (las más similares semánticamente a Q) se usarán para la recuperación complementaria. Se fijó $n = 2$ para todos los datasets.

En todos los casos, se implementó un mecanismo para evitar la recuperación de documentos duplicados entre la búsqueda original y las búsquedas complementarias. Para cada una de las n preguntas representativas Q_i^{G} seleccionadas, se recuperó un número de documentos únicos igual a $\lfloor (1-\alpha) \cdot M/n \rfloor$.

Prompt de generación: Para la generación final de respuestas por parte del modelo LLM (`llama3.1-8b-instruct`), se empleó un *prompt* estandarizado, cuya formulación exacta se detalla en el Apéndice 7.4. Este mismo *prompt* se utilizó consistentemente para las dos variantes comparadas:

- **Traditional RAG (RAG Base):** Recuperación basada únicamente en la pregunta original Q.
- **Combined Retrieve RAG:** Recuperación enriquecida con las preguntas similares generadas por KRAQ.

4.2.3. Resultados y analisis

La Tabla 4.1 presenta una comparación del rendimiento entre la estrategia de RAG tradicional y Combined Retrieve RAG, utilizando las métricas de EM y Evaluación con LLM como Juez. Un análisis detallado revela las siguientes tendencias:

Tab. 4.1: Comparación de la precisión de respuesta entre RAG Tradicional y Combined Retrieve RAG utilizando Exact Match (EM) y Evaluación con LLM como Juez (LLM-as-judge).

Dataset	**Exact Match**		**LLM-as-judge**	
	Traditional	**Combined**	**Traditional**	**Combined**
HotPotQA	57.0	**58.6**	76.3	**77.3**
TriviaQA	88.6	**89.0**	93.6	**94.6**
PubHealth	65.5	**66.2**	65.5	**66.2**
BioASQ	**69.6**	67.5	78.8	**79.5**

Evaluación mediante LLM como Juez. Al examinar los resultados obtenidos mediante la Evaluación con LLM como Juez, que prioriza la corrección semántica sobre la coincidencia léxica estricta, se observa una mejora consistente y generalizada con la estrategia Combined Retrieve RAG. Específicamente, en el dataset HotPotQA, la precisión se incrementó de 76.3 % a 77.3 %, lo que representa una mejora de aproximadamente el 1.3 %. De manera similar, en TriviaQA, se registró un aumento de 93.6 % a 94.6 %, (+1.1 %). Para PubHealth, el rendimiento también experimentó un ascenso, pasando de 65.5 % a 66.2 % (+1.1 %). Finalmente, en BioASQ, la puntuación mejoró de 78.8 % a 79.5 % (+0.9 %). Esta tendencia uniforme a través de los cuatro *datasets*, aunque las magnitudes de mejora sean modestas, sugiere que la diversificación del contexto de entrada para el LLM generador, lograda mediante la incorporación de documentos recuperados a partir de las preguntas representativas generadas por KRAQ, tiene un impacto positivo. El enriquecimiento con perspectivas semánticas adicionales parece dotar al LLM de una base informativa más robusta y diversa, lo que se traduce en una capacidad mejorada para generar respuestas conceptualmente más correctas y completas.

Evaluación mediante Exact Match. En lo referente a la métrica de Exact Match, que evalúa la coincidencia literal exacta tras la normalización textual, la estrategia Combined Retrieve RAG también exhibe una tendencia mayoritariamente positiva, aunque con matices. En el dataset HotPotQA, el EM se elevó de 57.0 % a 58.6 %, lo que representa un incremento de aproximadamente el 2.8 %. Para TriviaQA, la mejora fue de 88.6 % a 89.0 % (+0.5 %), mientras que en PubHealth, el EM aumentó de 65.5 % a 66.2 % (+1.1 %). Estos resultados indican que, en una proporción significativa de los casos para estos tres *datasets*, el contexto diversificado no solo no perjudica la capacidad del LLM para generar la respuesta literal esperada, sino que puede incluso mejorarla. No obstante, es importante señalar que el comportamiento en el dataset BioASQ bajo esta métrica presenta ciertas particularidades.

Consideraciones específicas para la evaluación en BioASQ. El dataset BioASQ, debido a la naturaleza de sus respuestas de referencia, requiere una consideración particular al interpretar los resultados de evaluación. Como se describió en la Sección 3.2.1, las respuestas de referencia para este dataset se presentan típicamente como una lista de entidades o términos (e.g., genes, fármacos) que contestan directamente a la pregunta.

Para la evaluación mediante **Exact Match** en este dataset, se utilizó esta lista como respuesta de referencia. Sin embargo, estas listas pueden ser problemáticas para una métrica de coincidencia literal estricta, ya que frecuentemente contienen múltiples formas de escribir la misma entidad, sinónimos, o un número extenso de ítems. Para adaptar la métrica EM a estas características, se implementó una variante para BioASQ: una respuesta generada A_{gen} se consideró correcta si contenía, de forma literal y tras la normalización, **al menos la mitad de los ítems presentes en la lista de la respuesta de referencia** A.

Dada la complejidad inherente a evaluar la cobertura de una lista mediante EM, la **Evaluación con LLM como Juez** se consideró particularmente pertinente para BioASQ, ya que permite una valoración más matizada de si la respuesta generada cubre adecuadamente los elementos clave de la lista de referencia. Para esta evaluación, la lista de respuestas de referencia se convirtió en una cadena de texto. El prompt para el LLM-Juez (detallado en el Apéndice 7.7) fue modificado para BioASQ añadiendo las siguientes directrices específicas:

The reference answers are provided as a list of true answers to the question. The generated answer should cover most of the items in the reference answers.

Esta adaptación instruye al LLM-Juez para que verifique si la respuesta generada A_{gen} incluye la mayoría de los elementos de la lista de referencia A, ofreciendo así una medida de la cobertura semántica y factual de la lista.

A pesar de la flexibilización en EM, el rendimiento para Combined Retrieve RAG en BioASQ fue de 67.5 %, una disminución en comparación con el 69.6 % del RAG tradicional (Tabla 4.1). Esta reducción en EM contrasta con la mejora observada en la misma tabla para la evaluación con LLM-as-judge (que utilizó el prompt adaptado y subió a 79.5 % frente a 78.8 % en el RAG tradicional). Este contraste podría indicar que el contexto más diverso de Combined Retrieve RAG lleva al LLM a generar respuestas que, si bien cubren semánticamente bien los elementos de la lista según LLM-as-judge, pueden no alcanzar el umbral de coincidencia literal requerido por nuestra variante de EM, quizás al parafrasear o presentar los elementos de forma diferente. Este comportamiento resalta la importancia de analizar los resultados de diferentes métricas de forma conjunta y comprender las particularidades de cada dataset y método de evaluación.

Conclusión. En conjunto, los resultados obtenidos sugieren que la estrategia Combined Retrieve RAG, que integra preguntas generadas por KRAQ, representa una optimización valiosa para sistemas RAG. La mejora consistente en la métrica de LLM-as-judge a través de todos los *datasets* indica el beneficio de la diversificación contextual en términos de calidad semántica. Las mejoras en EM para la mayoría de los *datasets* refuerzan esta observación.

La tendencia general positiva posiciona a Combined Retrieve RAG como una técnica prometedora para la optimización del rendimiento y la robustez de los sistemas RAG.

4.2.4. Estudios de ablación

Para los estudios de ablación presentados en esta sección, se utilizó la misma cantidad de preguntas de referencia de los datasets que en los experimentos principales de Combined Retrieve RAG, con el fin de asegurar la comparabilidad.

Impacto del número de preguntas similares de KRAQ

Este estudio de ablación investiga cómo varía la precisión de Combined Retrieve RAG al modificar el número de preguntas representativas de KRAQ (n) que se utilizan para la recuperación complementaria. En este análisis, se mantuvo fija la proporción de recuperación para la pregunta original en $\alpha = 0{,}5$, y el número total de documentos recuperados en $M = 15$. Los resultados obtenidos sobre el dataset HotPotQA, utilizando tanto Exact Match como la Evaluación con LLM como Juez, se muestran en la Tabla 4.2.

Tab. 4.2: Impacto del número de preguntas similares de KRAQ (n) en la precisión (EM % y LLM-as-judge %) de Combined Retrieve RAG sobre HotPotQA ($\alpha = 0{,}5, M = 15$).

Número de Preguntas Similares (n)	**EM**	**LLM-as-judge**
1	58.67	77.00
2	58.67	77.33
3	58.67	77.33
4	57.00	76.00

Análisis. Los resultados de la Tabla 4.2 indican que el rendimiento de Combined Retrieve RAG, al variar el número de preguntas similares (n) manteniendo $\alpha = 0{,}5$, exhibe una notable estabilidad para valores bajos de n, seguida de una disminución al incrementar n a 4.

En términos de **Exact Match**, la precisión se mantiene consistentemente en 58.67 % para $n = 1, 2$, y 3. Esta estabilidad sugiere que, hasta tres preguntas complementarias, el sistema logra integrar la información diversificada sin una pérdida en la capacidad de generar la respuesta literal correcta.

Una tendencia similar, aunque con un ligero óptimo, se observa con la métrica de **LLM como Juez**. La precisión comienza en 77.00 % para $n = 1$, y alcanza un máximo de 77.33 % para $n = 2$ y $n = 3$, indicando un pequeño beneficio al incorporar dos o tres perspectivas adicionales para mejorar la calidad semántica de la respuesta.

Sin embargo, para ambas métricas, se produce una clara disminución del rendimiento cuando el número de preguntas similares se incrementa a $n = 4$. El EM cae a 57.00 % y la precisión con LLM-as-judge desciende a 76.00 %. Esta caída sugiere que, si bien la diversificación del contexto mediante múltiples consultas es inicialmente beneficiosa, un número excesivo de preguntas complementarias, especialmente cuando el presupuesto total de documentos M es fijo, puede volverse contraproducente. Al distribuir los M documentos recuperados entre un mayor número de consultas, cada consulta obtiene un conjunto más reducido de documentos. Esto podría llevar a que cada perspectiva adicional no aporte suficiente información nueva o relevante, o incluso que se introduzca ruido o información menos pertinente, disminuyendo la efectividad general del contexto proporcionado al LLM.

Impacto de la proporción de recuperación (α)

Este estudio de ablación examina la influencia de la proporción α, que determina qué fracción del presupuesto total de recuperación M (fijado en 15 documentos) se asigna a la pregunta original del usuario, y qué fracción se distribuye entre las preguntas complementarias de KRAQ. Para este análisis, se mantuvo fijo el número de preguntas similares de KRAQ en $n = 2$. Se incluyó también el caso de $\alpha = 1{,}0$, que equivale al RAG tradicional donde todos los documentos se recuperan únicamente con la pregunta original. Los resultados obtenidos sobre el dataset HotPotQA, utilizando tanto Exact Match como la Evaluación con LLM como Juez, se presentan en la Tabla 4.3.

Tab. 4.3: Impacto de la proporción de recuperación para la pregunta original (α) en la precisión (EM % y LLM-as-judge %) de Combined Retrieve RAG sobre HotPotQA ($n = 2, M = 15$). El caso $\alpha = 1{,}0$ representa el RAG Tradicional.

Proporción α	**EM**	**LLM-as-judge**
0.25	56.00	75.00
0.50	58.60	77.33
0.75	**59.60**	**77.33**
1.00 (Tradicional)	57.00	76.30

Análisis. En términos de **Exact Match**, se observa un pico de rendimiento con $\alpha = 0{,}75$, alcanzando 59.60 %. Este valor es superior tanto al RAG tradicional como a configuraciones con menor peso en la pregunta original. Esto sugiere que una estrategia que combina una fuerte ponderación de la pregunta original (75 %) con una contribución menor pero significativa de las preguntas de KRAQ (25 % distribuido entre $n = 2$ preguntas) es la más efectiva para obtener la respuesta literal precisa.

Con la métrica de **LLM como Juez**, la tendencia es similar, con el rendimiento máximo de 77.33 % alcanzado tanto con $\alpha = 0{,}50$ como con $\alpha = 0{,}75$. Ambos valores superan al RAG tradicional y a la configuración con $\alpha = 0{,}25$.

En conjunto, estos resultados confirman que la pregunta original del usuario es el ancla fundamental para la relevancia, pero que la diversificación contextual introducida por las preguntas de KRAQ puede conducir a mejoras en la precisión. Una dependencia excesiva de las preguntas complementarias (como en $\alpha = 0{,}25$) o la ausencia total de ellas ($\alpha = 1{,}0$) no resulta en el rendimiento óptimo observado con $\alpha = 0{,}50$ o $\alpha = 0{,}75$.

En los resultados principales de la tabla 4.1 se utilizo $\alpha = 0{,}50$ ya que la primera hipotesis habia sido que distribuir de manera equitativa los documentos seria lo optimo. Este estudio de ablación que se realizó posteriormente a esos experimentos da indicios de que $\alpha = 0{,}75$ ofrecería aun mas beneficios en relación al RAG Tradicional. No se pudo realizar la tabla principal de experimentos con este parámetro debido a la falta de tiempo.

Impacto del modelo generador de preguntas de KRAQ

Este estudio de ablación se centró en determinar cómo la calidad y el método de generación de las preguntas representativas de KRAQ ($\mathcal{Q}^{\mathcal{K}}$) influyen en el rendimiento final del sistema Combined Retrieve RAG. Para este análisis, se mantuvo constante la configuración de recuperación de Combined Retrieve RAG (con $M = 15$ documentos en total, $n = 2$ preguntas similares de KRAQ, y $\alpha = 0{,}5$), y se varió el origen de las preguntas $\mathcal{Q}^{\mathcal{K}}$.

Las variantes del generador de preguntas de KRAQ consideradas fueron las mismas que en la evaluación de KRAQ (ver Sección 3.2.7):

1. **KRAQ (Fine-tuned):** Utiliza el conjunto de preguntas $\mathcal{Q}^{\mathcal{K}}$ generado por el modelo de KRAQ con *fine-tuning* específico (Sección 3.2.5). Esta es la configuración estándar de Combined Retrieve RAG en los resultados principales.

2. **KRAQ (Instruct):** Utiliza preguntas $\mathcal{Q}^{\mathcal{K}}$ generadas por el modelo LLaMA 3.1–8B Instruct, aplicado sobre los mismos resúmenes comunitarios de GraphRAG.

3. **Baseline-Aleatorio:** Utiliza preguntas $\mathcal{Q}^{\mathcal{K}}$ generadas por el *baseline* aleatorio (Sección 3.2.3), que formula preguntas a partir de *chunks* de texto seleccionados al azar.

La evaluación se realizó sobre el dataset BioASQ. Se presentan los resultados para ambas métricas: Exact Match y la Evaluación con LLM como Juez, ambos con la adaptación para BioASQ descrita en la Sección 4.2.3. Los resultados se muestran en la Tabla 4.4.

Tab. 4.4: Impacto del modelo generador de preguntas de KRAQ en la precisión (EM % y LLM-as-judge %) de Combined Retrieve RAG sobre BioASQ ($M = 15, n = 2, \alpha = 0{,}5$).

Modelo Generador de Preguntas $\mathcal{Q}^{\mathcal{K}}$	**EM**	**LLM-as-judge**
KRAQ (Fine-tuned)	**67.5**	**79.5**
KRAQ (Instruct)	65.1	75.7
Baseline-Aleatorio	65.3	77.7

Análisis. Los resultados de la Tabla 4.4 indican una clara dependencia del rendimiento de Combined Retrieve RAG con respecto a la calidad y el método de generación de las preguntas representativas $\mathcal{Q}^{\mathcal{K}}$ utilizadas.

El uso de preguntas generadas por el modelo KRAQ con *fine-tuning* específico consistentemente resulta en la mayor precisión en ambas métricas: 67.5 % en EM y 79.5 % en LLM-Juez. Esto sugiere que las preguntas que son semánticamente más relevantes (como las que produce el KRAQ *fine-tuned*, según se observó en la Sección 3.2.7) son más efectivas para guiar la recuperación de documentos complementarios que enriquecen el contexto del LLM generador de respuestas.

Es interesante observar el comportamiento de las otras dos variantes. El **Baseline-Aleatorio**, que genera preguntas directamente de *chunks* del corpus, alcanza un 65.3 % en EM y un 77.7 % en LLM-Juez. Estos valores superan a los obtenidos por **KRAQ (Instruct)** (65.1 % en EM y 75.7 % en LLM-Juez), que utiliza el modelo preentrenado sobre resúmenes comunitarios. Este hallazgo sugiere que, en ausencia de un *fine-tuning* específico que oriente al modelo generador de preguntas, las preguntas derivadas directamente del texto del corpus (aunque de forma aleatoria y sin análisis estructural profundo como en el Baseline-Aleatorio) pueden ser marginalmente más efectivas para la recuperación que aquellas generadas por un modelo instruct generalista a partir de resúmenes abstractos. El modelo KRAQ (Instruct), aunque opera sobre representaciones temáticas más cohesivas (los resúmenes comunitarios), podría no estar suficientemente calibrado para transformar estos resúmenes en las consultas más significativas para la recuperación sin el entrenamiento específico que recibe KRAQ (Fine-tuned).

La superioridad del KRAQ *fine-tuned* sobre ambas alternativas subraya la importancia no solo de una buena representación del contenido (resúmenes comunitarios) sino también, y de manera crucial, de un modelo generador de preguntas específicamente entrenado para producir consultas que sean semánticamente relevantes y efectivas en un contexto de recuperación. Este estudio de ablación refuerza la conclusión de que la calidad y la especificidad del entrenamiento del generador de preguntas de KRAQ son factores determinantes para el éxito de las optimizaciones propuestas en los sistemas RAG.

5. EFFICIENT SPECULATIVE RAG

Este capítulo presenta la segunda aplicación principal de las preguntas generadas por KRAQ (Capítulo 3): una optimización para el *framework* Speculative RAG [78] (Explicado en Sección 2.2.3), denominada *Efficient Speculative RAG*. El objetivo es mitigar el cuello de botella computacional asociado al cálculo en línea de embeddings instruidos en el Speculative RAG original, proponiendo un método de pre-cómputo basado en las preguntas de KRAQ. Se detallará la metodología de esta variante eficiente y se evaluará su impacto en la latencia y la calidad de las respuestas.

5.1. Metodologia de Efficient Speculative RAG

En esta sección se presenta una propuesta de mejora orientada a optimizar la eficiencia del framework Speculative RAG [78].

Antes de detallar nuestra optimización, recordemos brevemente que Speculative RAG (Explicado en profundidad en la Sección 2.2.3) opera bajo el paradigma *draft-then-verify*. En lugar de que un único LLM grande procese todo el contexto recuperado, múltiples modelos "borrador" (M_{Drafter}), más pequeños y rápidos, generan en paralelo un conjunto de respuestas candidatas, cada uno operando sobre un subconjunto diferente de los documentos recuperados. Posteriormente, un modelo "verificador" (M_{Verifier}), usualmente más potente, evalúa estos borradores y selecciona el de mayor calidad como respuesta final. Este enfoque busca reducir la latencia y explorar un espacio de respuestas más amplio.

Como se expuso en la Sección 2.2.3, uno de los principales factores que comprometen la eficiencia del pipeline Speculative RAG original es la exigencia en el calculo para la creación de los subconjuntos, alli debe calcular los embeddings para cada nueva pregunta del usuario Q, los embeddings $\mathcal{E}(d_i \mid Q)$ para cada documento recuperado $d_i \in \mathcal{D}$. Este proceso, al ser dependiente de Q, debe repetirse cada vez que la pregunta cambia, lo que imposibilita la precomputación y dificulta la escalabilidad del sistema.

Efficient Speculative RAG soluciona este problema al separar la generación de los embeddings instruidos del tiempo de consulta. En su lugar, se aprovechan embeddings precomputados para cada documento del corpus, los cuales han sido generados en función de las preguntas representativas obtenidas mediante KRAQ. Estas preguntas, al reflejar los ejes temáticos principales del corpus, actúan como proxies semánticos para las consultas mas relevantes (como fue evidenciado en la evaluación de KRAQ Sección 3.2.7).

5.1.1. Algoritmo

El procedimiento de **Efficient Speculative RAG** se articula en dos fases principales, diseñadas para optimizar la eficiencia de Speculative Rag: una fase de precomputación realizada offline y una fase de inferencia online.

Fase de precomputación (Offline)

El objetivo de esta fase es precomputar de antemano los embedding instruidos para optimizar la latencia de Speculative Rag.

1. **Generación de preguntas representativas:** A partir del corpus global de documentos E, se genera un conjunto de k preguntas representativas $\mathcal{Q}^{\mathcal{K}} = \{Q_1^K, Q_2^K, \ldots, Q_k^K\}$ mediante KRAQ.

2. **Cálculo y almacenamiento de embeddings instruidos precalculados:** Para cada documento $d_i \in E$ y para cada pregunta representativa $Q_j^K \in \mathcal{Q}^{\mathcal{K}}$, se calcula su correspondiente embedding instruido. Estos embeddings se almacenan en un índice o tabla de consulta E_{pre}. Formalmente, para cada par (d_i, Q_j^K):

$$E_{\text{pre}}(d_i, Q_j^K) \coloneqq \mathcal{E}(d_i \mid Q_j^K) \tag{5.1}$$

   donde $\mathcal{E}(d_i \mid Q_j^K)$ denota el embedding del documento d_i instruido por la pregunta representativa Q_j^K. Así, para cada documento d_i, se almacena un conjunto de k embeddings precalculados, uno por cada pregunta representativa.

Fase de inferencia (Online)

1. **Recuperación inicial de documentos:** Ante una nueva pregunta del usuario Q, se realiza una recuperación inicial de un conjunto de n documentos relevantes $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$ desde el corpus E según similitud coseno.

2. **Selección de la pregunta representativa mas similar:** Se identifica la pregunta representativa $Q_{sim}^K \in \mathcal{Q}^{\mathcal{K}}$ que exhibe la mayor similitud semántica con la pregunta del usuario Q segun la similitud coseno:

$$Q_{sim}^K = \arg\max_{Q_j^K \in \mathcal{Q}^{\mathcal{K}}} \left(\cos(\text{emb}(Q), \text{emb}(Q_j^K))\right)$$

3. **Utilización de embeddings precalculados para clustering:** Para cada uno de los n documentos $d_i \in \mathcal{D}$ recuperados en el paso 1, se accede a su embedding precalculado correspondiente a la pregunta representativa seleccionada Q_{sim}^K. Es decir, se utiliza $E_{\text{pre}}(d_i, Q_{sim}^K)$, obtenido de la tabla E_{pre} (ver Ecuación 5.1). Estos n embeddings precalculados específicos son los que se emplearán para realizar el agrupamiento temático (clustering) mediante el algoritmo K-Means.

4. **Aplicación del pipeline estándar de Speculative RAG:** Una vez definidos los clústers a partir de los embeddings $E_{\text{pre}}(d_i, Q_{sim}^K)$, el proceso continúa con el pipeline estándar de Speculative RAG, tal como se describió en la Sección **??**. Esto incluye:
   - Muestreo de subconjuntos de documentos a partir de los clústeres.
   - Generación paralela de borradores de respuesta (α_j, β_j) para cada subconjunto, utilizando el modelo especialista M_{Drafter}.
   - Evaluación de los borradores y selección de la respuesta final $\hat{A}$ mediante el modelo generalista M_{Verifier}.

Formalización de Efficient Speculative Rag

El procedimiento de Efficient Speculative RAG se formaliza en el Algoritmo 5.

Algorithm 5 Algoritmo de Efficient Speculative RAG (Inferencia Online)

Require: Pregunta del usuario Q, Corpus de documentos E, Tabla de embeddings precalculados E_{pre}, Conjunto de preguntas representativas $\mathcal{Q}^{\mathcal{K}}$, Número de documentos a recuperar n, Número de subconjuntos m, Número de clústeres k.

Ensure: Respuesta predicha $\hat{A}$ para la pregunta Q_{user}.

function EfficientSpeculativeRag($Q, E, E_{\text{pre}}, \mathcal{Q}^{\mathcal{K}}, n, m, k$)
  $\mathcal{D} \leftarrow \text{RetrieveDocs}(Q, E, n)$
  $Q_{sim}^{K} \leftarrow \text{RetrieveSimilarQuestion}(Q, \mathcal{Q}^{\mathcal{K}})$
  $E \leftarrow \emptyset$
  for cada $d_i \in \mathcal{D}$ **do**
    $E \leftarrow E \cup \{E_{\text{pre}}(d_i, Q_{sim}^{K})\}$ ▷ Embeddings precalculados de $\mathcal{D}$ para Q_{sim}^{K}
  $\{c_1, c_2, \ldots, c_k\} \leftarrow \text{K-Means}(E, k)$
  $\Delta \leftarrow \emptyset$
  for $j = 1$ **to** m **do**
    $\delta_j \leftarrow \emptyset$
    for $l = 1$ **to** k **do**
      $\delta_j \leftarrow \delta_j \cup \{\text{SampleOne}(c_l)\}$ ▷ Muestrear un documento de cada clúster c_l
    $\Delta \leftarrow \Delta \cup \{\delta_j\}$
  Drafts $\leftarrow \emptyset$
  for all $\delta_j \in \Delta$ **in parallel do**
    $(\alpha_j, \beta_j) \leftarrow M_{\text{Drafter}}.\text{generate}(Q, \delta_j)$
    $\rho_j^{\text{draft}} \leftarrow P_{\text{Drafter}}(\beta_j \mid Q, \delta_j) + P_{\text{Drafter}}(\alpha_j \mid Q, \delta_j)$
    $\rho_j^{\text{self-contain}} \leftarrow P_{\text{Drafter}}(\alpha_j, \beta_j \mid Q, \delta_j)$
    $\rho_j^{\text{self-reflect}} \leftarrow P_{M_{\text{Verifier}}}(\text{“Yes”} \mid Q, \alpha_j, \beta_j)$
    $\rho_j^{\text{final}} \leftarrow \rho_j^{\text{draft}} \cdot \rho_j^{\text{self-contain}} \cdot \rho_j^{\text{self-reflect}}$
    Drafts $\leftarrow$ Drafts $\cup \{(\alpha_j, \rho_j^{\text{final}})\}$
  $(\hat{A}, \_) \leftarrow \arg\max_{(\alpha_j, \rho_j^{\text{final}}) \in \text{Drafts}} \rho_j^{\text{final}}$ ▷ Seleccionar el mayor score final
  return $\hat{A}$

5.2. Experimentación y resultados de Efficient Speculative RAG

En la presente sección se exponen los detalles experimentales y los resultados obtenidos al evaluar la variante optimizada de Speculative Rag denominada *Efficient Speculative RAG*. El diseño experimental que se detalla a continuación tiene como objetivo principal cuantificar la ganancia en eficiencia (medida en términos de latencia) y, simultáneamente, verificar que la calidad de las respuestas generadas por *Efficient Speculative RAG* se mantiene comparable a la del método base. Para ello, se describirá el setup experimental, la implementación específica de los componentes críticos y los parámetros seleccionados, culminando con un análisis comparativo de los resultados frente al Speculative RAG original.

5.2.1. Diseño de evaluación

Con el objetivo de evaluar empíricamente el rendimiento del método propuesto, se diseñó un protocolo experimental que permite una comparación directa con el algoritmo original de Speculative RAG [78]. Esta comparación se centra en dos aspectos clave: la

latencia de inferencia y la precisión de las respuestas generadas. El propósito principal de la evaluación es verificar que la eliminación del cómputo en línea de los embeddings instruidos (elemento central de nuestra propuesta) no compromete la calidad de las respuestas, al mismo tiempo que permite una mejora significativa en la latencia del sistema

La evaluación se fundamentó en el uso de datasets estándar de QA, donde cada instancia se modela como una tripleta (Q, A, E). En esta tupla, Q representa el conjunto de preguntas del dataset, A el conjunto de respuestas consideradas correctas, y E el corpus de documentos de evidencia del dataset que sirven como base de conocimiento para las preguntas.

Protocolo de evaluación detallado

El procedimiento experimental seguido para el dataset (Q, A, E) de evaluación fue el siguiente:

1. **Preprocesamiento del corpus (fase offline para Efficient Speculative RAG):**

   - Se aplicó KRAQ sobre el corpus E para generar un conjunto de preguntas representativas $\mathcal{Q}^{\mathcal{K}} = \{Q_1^K, \ldots, Q_k^K\}$.
   - En un escenario ideal, la fase offline de Efficient Speculative RAG implicaría el pre-cómputo exhaustivo de los embeddings instruidos. Específicamente, para cada documento d_i del corpus E y para cada pregunta representativa Q_j^K del conjunto $\mathcal{Q}^{\mathcal{K}}$ generado por KRAQ, se calcularía y almacenaría el embedding $\mathcal{E}(d_i \mid Q_j^K)$ utilizando InBedder-RoBERTa, conformando así una tabla de consulta $E_{\text{pre}}(d_i, Q_j^K)$. No obstante, debido a las limitaciones temporales y de recursos computacionales inherentes al desarrollo de esta tesis, este pre-cómputo completo no fue factible. En su lugar, para poder evaluar tanto el impacto en la latencia como la precisión del sistema utilizando los embeddings conceptualmente correctos (aquellos instruidos por Q_{sim}^K), se adoptó un procedimiento de estimación, el cual se detalla en la Sección 5.2.4.

   Este paso es exclusivo de la preparación para Efficient Speculative RAG y se realiza una única vez por corpus.

2. **Ejecución comparativa de algoritmos (Fase Online):** Para cada pregunta de referencia Q de la instancia actual, y utilizando los documentos del conjunto E como base para la recuperación, se ejecutaron ambas versiones del algoritmo para generar una respuesta A_{gen}:

   - **Speculative RAG (Original):** Se recuperó un conjunto $\mathcal{D}$ de documentos de E basado en Q. Luego, se calcularon los embeddings instruidos $\text{InBedder}(d_i, Q)$ para cada $d_i \in \mathcal{D}$ en tiempo de ejecución, utilizando Q como instrucción.
   - **Efficient Speculative RAG:** Se recuperó un conjunto $\mathcal{D}$ de documentos de E basado en Q. Se seleccionó la pregunta representativa $Q_{sim}^K \in \mathcal{Q}^{\mathcal{K}}$ más similar a Q. Posteriormente, se utilizaron los embeddings precomputados $E_{\text{pre}}(d_i, Q_{sim}^K)$ para el clustering de los documentos $d_i \in \mathcal{D}$.

   Ambas variantes procedieron luego con el muestreo de subconjuntos, la generación de borradores y la verificación para producir sus respectivas A_{gen}.

3. **Aplicación de métricas de evaluación:** Para cada A_{gen} obtenida por ambos métodos, se comparó con la respuesta de referencia A utilizando las siguientes métricas:

   - *Exact Match (EM):* Se midió la coincidencia literal exacta entre A_{gen} y A, tras la normalización textual estándar.
   - *Evaluación Semántica con LLM como Juez:* Un LLM (Llama3.1-8b-instruct) evaluó si A_{gen} responde adecuadamente a Q de manera semánticamente equivalente a A, siguiendo el protocolo detallado.

   Estas dos métricas fueron explicadas de manera mas detallada en la Sección 4.2.1

4. **Medición de latencia:** Adicionalmente, se registró la latencia total de inferencia para cada método. Esta medición abarcó todos los componentes del pipeline online:

   - Recuperación inicial de documentos (común a ambos).
   - Cómputo de embeddings instruidos (aplicable solo a Speculative RAG original).
   - Selección de Q^K_{sim} y estimacion de la recuperación de embeddings precomputados (aplicable solo a Efficient Speculative RAG, estimación explicada en 5.2.4).
   - Clustering y generación de subconjuntos de documentos.
   - Generación de borradores y proceso de verificación final.

Esta evaluación comparativa busca proporcionar una base empírica sólida para validar la hipótesis central de que Efficient Speculative RAG conserva la calidad de respuesta del sistema original, al tiempo que elimina su principal cuello de botella computacional. De confirmarse, esto abriría el camino hacia implementaciones más escalables y eficientes de este avanzado paradigma de RAG en entornos de producción.

5.2.2. Cálculo de scores

La selección del borrador final en el *pipeline* de Speculative RAG, tanto en su versión original como en la variante *Efficient Speculative RAG*, se fundamenta en una puntuación combinada ρ_j para cada par de borrador de respuesta y racional (α_j, β_j). Conforme a lo explicado en la Sección 5.1.1, esta puntuación agrega tres componentes: ρ_j^{draft}, $\rho_j^{\text{self-contain}}$ y $\rho_j^{\text{self-reflect}}$. En nuestra implementación, estos elementos se derivaron de las log-probabilidades de los tokens generados, obtenidas mediante el parámetro `logprobs` de la API del cliente OpenAI en comunicación con el servidor vLLM.

Procesamiento inicial: segmentación de racional y respuesta El modelo M_{Drafter} fue instruido para producir una única secuencia textual que contiene tanto el racional β_j como la respuesta α_j (detalles del formato JSON en Sección 5.2.5). Para discernir las log-probabilidades correspondientes a cada segmento, se aplicó una heurística basada en la identificación de subcadenas de tokens, tales como "ration" o "resp". Una vez localizados dichos marcadores, se delimitaron las subsecuencias de tokens T_{β_j} (para el racional) y T_{α_j} (para la respuesta), junto con sus respectivas log-probabilidades.

Consideraciones sobre la estabilidad numérica y definición de métricas de confianza La formulación teórica de los scores en Wang et al. [78] se apoya en la probabilidad

de secuencia completa, $P(S)$. La probabilidad de una secuencia se estima como la productoria de las probabilidades de sus tokens constituyentes: $P(S) = \prod_{t \in S} P(t)$. Sin embargo, en la práctica computacional, la multiplicación directa de un gran número de probabilidades (valores inherentemente menores que 1) conduce al conocido problema del ***underflow* numérico**: el resultado se vuelve tan extremadamente pequeño que la precisión del computador no puede representarlo y colapsa a cero.

Dado que el servidor vLLM, utilizando el cliente de OpenAI, ya proporciona las log-probabilidades (`logprobs`) de cada token, el cálculo de la probabilidad de secuencia $P(S)$ se realiza mediante la siguiente formulación, que transforma la productoria de probabilidades en una suma en el espacio logarítmico:

$$P(S) = \exp\left(\sum_{t \in S} \text{logprob}(t)\right).$$

No obstante, en nuestra implementación, esta formulación resultó insuficiente para secuencias extensas como las generadas para los pares (α_j, β_j). La suma de un gran número de log-probabilidades (que son negativas) producía un valor tan pequeño que, al aplicar la exponencial, el score final colapsaba sistemáticamente a cero. Este colapso impide una diferenciación efectiva entre los borradores, volviendo en la práctica aleatoria la selección.

Para esquivar esta limitación y asegurar la estabilidad numérica, se adoptó una métrica de confianza, denotada $P_{\text{conf}}(S)$, basada en la exponencial de la log-probabilidad promedio de los tokens de la secuencia S generados por un modelo específico. Si S es una secuencia de L tokens, esta métrica se define como:

$$P_{\text{conf}}(S) = \exp\left(\frac{1}{L}\sum_{t \in S} \text{logprob}(t)\right).$$

Esta medida ofrece una cuantificación de la confianza normalizada por la longitud de la secuencia y es considerablemente más robusta frente al *underflow*. Si bien representa una desviación de la probabilidad de secuencia teórica usada en el paper original (donde no se especifica cómo se manejó este problema), se consideró una aproximación pragmática y necesaria para la viabilidad de la implementación.

Los tres componentes del score ρ_j se calcularon utilizando esta métrica P_{conf}, especificando el modelo correspondiente en cada caso:

Componentes del score

ρ_j^{draft} *(Puntuación del borrador):* Refleja la confianza combinada del modelo M_{Drafter} en la generación del racional β_j y la posterior respuesta α_j:

$$\rho_j^{\text{draft}} = P_{M_{\text{Drafter}}}(\beta_j \mid Q, \delta_j) + P_{M_{\text{Drafter}}}(\alpha_j \mid Q, \delta_j, \beta_j).$$

En nuestra implementación, estas probabilidades se aproximaron sumando las confianzas P_{conf} de cada segmento:

$$\rho_j^{\text{draft}} \approx P_{\text{conf}}(T_{\beta_j}) + P_{\text{conf}}(T_{\alpha_j}).$$

$\rho_j^{self\text{-}contain}$ *(Puntuación de auto-contención):* Mide la coherencia interna del par (α_j, β_j), representada por la probabilidad conjunta de que M_{Drafter} genere ambos elementos dado el contexto:

$$\rho_j^{\text{self-contain}} = P_{M_{\text{Drafter}}}(\alpha_j, \beta_j \mid Q, \delta_j).$$

Esta probabilidad conjunta se estimó aplicando la métrica P_{conf} a la secuencia completa de tokens concatenados $T_{\alpha_j \oplus \beta_j}$:

$$\rho_j^{\text{self-contain}} \approx P_{\text{conf}}(T_{\alpha_j \oplus \beta_j}).$$

$\rho_j^{\textit{self-reflect}}$ *(Puntuación de auto-reflexión):* Evalúa la confianza del modelo M_{Verifier} en que el racional β_j soporta adecuadamente la respuesta α_j. Teóricamente, es la probabilidad de que el verificador responda "Yes":

$$\rho_j^{\text{self-reflect}} = P_{M_{\text{Verifier}}}(\text{"Yes"} \mid Q, \alpha_j, \beta_j).$$

Esta probabilidad se obtuvo de la confianza P_{conf} en la respuesta del verificador, A_{verif}, (ver Apéndice 7.6 para ver prompt exacto utilizado):

$$\rho_j^{\text{self-reflect}} \approx \begin{cases} P_{\text{conf}}(T_{A_{\text{gen}}}) & \text{si } A_{\text{gen}} = \text{"Yes"} \\ 1 - P_{\text{conf}}(T_{A_{\text{gen}}}) & \text{si } A_{\text{gen}} = \text{"No"} \end{cases}$$

Cálculo del score final combinado ρ_j Habiendo calculado cada componente como una medida de confianza normalizada, el score final ρ_j para el borrador j se obtuvo mediante su producto. Esta estructura es consistente con la propuesta por Wang et al. [78]:

$$\rho_j = \rho_j^{\text{draft}} \cdot \rho_j^{\text{self-contain}} \cdot \rho_j^{\text{self-reflect}}.$$

Este producto asegura que cada dimensión de la calidad del borrador influya en la evaluación global. La respuesta α_j asociada al ρ_j máximo fue seleccionada como la respuesta definitiva $\hat{A}$:

$$\hat{A} = \arg\max_{\alpha_j}(\rho_j).$$

Esta implementación, con sus adaptaciones para la estabilidad numérica, busca preservar la funcionalidad esencial del mecanismo de scoring de Speculative RAG.

5.2.3. Paralelismo y estimación de latencia

Una de las ventajas conceptuales del algoritmo Speculative RAG [78], y por extensión de nuestra variante *Efficient Speculative RAG*, es la capacidad de procesar los m subconjuntos de documentos δ_j en paralelo. Este paralelismo se aplica tanto a la generación de los borradores de respuesta y racional (α_j, β_j) mediante los modelos M_{Drafter}, como a la evaluación de dichos borradores con el modelo M_{Verifier}. Esta capacidad de ejecución en paralelo es fundamental para mitigar el posible incremento en la latencia total que podría suponer la generación y evaluación de múltiples candidatos de respuesta.

Debido a las limitaciones en los recursos computacionales disponibles para esta tesis (específicamente, una única GPU NVIDIA GeForce RTX 3090), no fue factible implementar una ejecución verdaderamente paralela de las m instancias de M_{Drafter} y M_{Verifier}. En su lugar, los m subconjuntos de documentos fueron procesados de manera secuencial. Para cada consulta q_c del conjunto de prueba (donde c es el índice de la consulta) y para cada uno de sus m subconjuntos de documentos $\delta_{c,j}$:

1. Se generó el par $(\alpha_{c,j}, \beta_{c,j})$ utilizando el modelo M_{Drafter}, registrando el tiempo de ejecución $t_{c,j}^{\text{draft}}$.

2. Posteriormente, se evaluó este par con el modelo M_{Verifier} para obtener el score $\rho_{c,j}^{\text{self-reflect}}$ (y las log-probabilidades necesarias para los otros componentes del score $\rho_{c,j}$), registrando el tiempo de ejecución $t_{c,j}^{\text{verify}}$.

El tiempo total de esta fase de generación y verificación si se ejecutara de forma puramente secuencial para los m borradores de una consulta q_c, $T_{\text{secuencial-gv}}^{(c)}$, sería:

$$T_{\text{secuencial-gv}}^{(c)} = \sum_{j=1}^{m} (t_{c,j}^{\text{draft}} + t_{c,j}^{\text{verify}}).$$

Estimación de la latencia en un escenario paralelizado por consulta

Para estimar la latencia que se habría obtenido en un escenario con paralelismo ideal, donde se dispone de suficientes recursos para procesar las m operaciones de *drafting* simultáneamente (y análogamente para las m operaciones de verificación) para una consulta dada q_c, se adoptó un enfoque común en la literatura [77]. Se asume que la latencia de una etapa paralelizada está determinada por el tiempo de procesamiento de la tarea más lenta dentro de ese conjunto de operaciones paralelas.

Específicamente, para una consulta q_c, si $t_{c,j}^{\text{draft}}$ es el tiempo registrado para generar el j-ésimo borrador (de m borradores) y $t_{c,j}^{\text{verify}}$ es el tiempo para verificarlo, la latencia estimada para la fase de *drafting* en paralelo para esa consulta es:

$$T_{\text{paralelo-draft}}^{(c)} = \max_{j=1,\ldots,m} (t_{c,j}^{\text{draft}}). \tag{5.2}$$

Y la latencia estimada para la fase de verificación en paralelo para esa consulta es:

$$T_{\text{paralelo-verify}}^{(c)} = \max_{j=1,\ldots,m} (t_{c,j}^{\text{verify}}). \tag{5.3}$$

Dado que la verificación de un borrador $(\alpha_{c,j}, \beta_{c,j})$ depende de su previa generación, y asumiendo un *pipeline* donde, para una consulta q_c, todas las generaciones de borradores ocurren antes que todas las verificaciones (o que existe la capacidad de paralelizar ambas etapas en su conjunto), la latencia total estimada para la fase combinada de generación y verificación de los m borradores para esa consulta específica ($T_{\text{draft-verify}}^{(c)}$) se calcula como la suma de estas dos latencias máximas:

$$T_{\text{draft-verify}}^{(c)} = T_{\text{paralelo-draft}}^{(c)} + T_{\text{paralelo-verify}}^{(c)}. \tag{5.4}$$

Cálculo de la latencia total agregada y representativa

Para obtener una medida general y robusta de la latencia por consulta para cada uno de los algoritmos evaluados (Speculative RAG original y *Efficient Speculative RAG*), se midieron los tiempos de ejecución de las diferentes etapas del *pipeline* para cada consulta individual q_c en el conjunto de datos de prueba. Estas etapas, medidas por consulta, son:

1. $T_{\text{retrieve}}^{(c)}$: Tiempo empleado en la recuperación inicial del conjunto de $N_{\text{retrieved}}$ documentos relevantes desde el corpus para la consulta q_c.

2. $T_{\text{embed-cluster}}^{(c)}$: Tiempo necesario para la etapa de obtención de *embeddings* y clustering para la consulta q_c.

- Para el Speculative RAG original: este tiempo incluye el cálculo en línea de los *embeddings* instruidos $\mathcal{E}(d_i \mid Q_c)$ y el siguiente clustering y muestreo.
- Para *Efficient Speculative RAG*: este tiempo representa el costo de seleccionar la pregunta Q^K_{sim} más similar a Q_c, el costo (estimado) de recuperar los *embeddings* precalculados $E_{\text{pre}}(d_i, Q^K_{\text{sim}})$, y el costo del clustering y muestreo. La metodología detallada para la medición y estimación de este componente en el contexto de esta tesis, dada la ausencia de un pre-cómputo completo, se describe en la Sección 5.2.4.

Esta etapa es donde *Efficient Speculative RAG* introduce su principal optimización de latencia.

3. $T^{(c)}_{\text{draft-verify}}$: Tiempo estimado para generar y verificar los m borradores para la consulta q_c, utilizando la estimación de latencia paralela dada por la Ecuación 5.4.

4. $T^{(c)}_{\text{select}}$: Tiempo para calcular los *scores* finales $\rho_{c,j}$ y seleccionar el mejor borrador para la consulta q_c. Esta etapa es generalmente muy rápida y de tiempo despreciable.

Una vez registradas estas mediciones ($T^{(c)}_{\text{retrieve}}$, $T^{(c)}_{\text{embed-cluster}}$, $T^{(c)}_{\text{draft-verify}}$, $T^{(c)}_{\text{select}}$) para cada una de las N_q consultas del conjunto de prueba, se calcula la **mediana** de los tiempos para cada una de estas cuatro etapas a través de todas las N_q. El uso de la mediana en lugar de la media aritmética proporciona una estimación más robusta de la tendencia central, menos sensible a posibles valores atípicos (*outliers*) que podrían surgir debido a errores provenientes de los modelos LLM como la generación inusual de borradores extremadamente largos debido a bucles en la generacion. Denotemos estas medianas como $\text{med}(T_{\text{retrieve}})$, $\text{med}(T_{\text{embed-cluster}})$, $\text{med}(T_{\text{draft-verify}})$, y $\text{med}(T_{\text{select}})$.

Por lo tanto, la latencia total representativa ($L^{\text{med}}_{\text{total}}$) para procesar una consulta promedio se calcula como la suma de estas medianas:

$$L^{\text{med}}_{\text{total}} = \text{med}(T_{\text{retrieve}}) + \text{med}(T_{\text{embed-cluster}}) + \text{med}(T_{\text{draft-verify}}) + \text{med}(T_{\text{select}}). \tag{5.5}$$

Al comparar *Efficient Speculative RAG* con el Speculative RAG original, la diferencia más significativa en la latencia se espera en el componente $\text{med}(T_{\text{embed-cluster}})$. Para los demás componentes, especialmente $\text{med}(T_{\text{draft-verify}})$, se utiliza la misma estimación paralela (Ecuaciones 5.2 y 5.3) para asegurar una comparación equitativa del potencial de paralelización inherente al algoritmo. Los resultados detallados de estas mediciones y estimaciones de latencia (basadas en la mediana) se presentarán y analizarán en la Sección 5.2.8.

5.2.4. Complejidad del pre-cómputo de embeddings instruidos en Efficient Speculative RAG y su estimación

Una consideración importante para la viabilidad práctica de *Efficient Speculative RAG* es la complejidad computacional asociada a la fase de pre-cómputo *offline*, donde se generan y almacenan los *embeddings* instruidos $E_{\text{pre}}(d_i, Q^K_j)$ (ver Ecuación 5.1).

Teóricamente, para un corpus con N_D documentos (o *chunks* d_i) y un conjunto de N_Q preguntas representativas Q^K_j generadas por KRAQ, la complejidad de este pre-cómputo sería del orden de $O(N_D \times N_Q \times C_{\text{emb}})$, donde C_{emb} es el costo de generar un *embedding* instruido para un par (documento, pregunta). Si tanto N_D como N_Q son grandes, este proceso puede ser considerablemente intensivo en tiempo y almacenamiento. Por ejemplo, para

un corpus que resulta en 15,000 *chunks* y un sistema KRAQ que genera 20,000 preguntas representativas (considerando diferentes niveles jerárquicos), se requerirían 300 millones de inferencias del modelo de *embeddings* instruidos.

Hipótesis de optimización del pre-cómputo. Una hipótesis para mitigar esta complejidad es que no sería necesario precomputar los *embeddings* instruidos para *todos* los documentos del corpus con respecto a *cada* pregunta representativa Q_j^K. En su lugar, podría ser suficiente precomputar $E_{\text{pre}}(d_i, Q_j^K)$ solo para un subconjunto de los documentos d_i que son más propensos a ser relevantes para Q_j^K. Por ejemplo, para cada Q_j^K, se podría primero realizar una búsqueda semántica estándar (usando *embeddings* no instruidos, como los de `nomic-embed-text`) para encontrar los $M_s = 1000$ documentos más similares a Q_j^K. Luego, el pre-cómputo de los *embeddings* instruidos con InBedder-RoBERTa se limitaría a estos M_s documentos para esa Q_j^K particular. Bajo esta optimización, la complejidad del pre-cómputo se reduciría a aproximadamente $O(N_Q \times M_s \times C_{\text{emb}} + N_Q \times C_{\text{search}})$, donde C_{search} es el costo de la búsqueda semántica inicial. Si $M_s \ll N_D$, esto podría representar un ahorro sustancial, haciendo que la complejidad sea lineal con respecto al número de preguntas representativas. No obstante, la validación de que un valor de M_s (como 1000) es suficiente para no degradar el rendimiento del clustering posterior en *Efficient Speculative RAG* resta como un trabajo futuro y no se exploró en esta tesis.

Manejo de la estimación de latencia en esta tesis. Debido a las limitaciones de tiempo y recursos computacionales inherentes al desarrollo de esta tesis de licenciatura, no fue factible realizar el pre-cómputo exhaustivo de todos los pares (d_i, Q_j^K) para los *datasets* y el volumen de preguntas generadas por KRAQ. Para poder, no obstante, obtener una estimación fiel del rendimiento y la ganancia en latencia que *Efficient Speculative RAG* ofrecería si el pre-cómputo se hubiera realizado, se adoptó la siguiente estrategia durante la fase de experimentación *online* para esta variante:

1. Para una consulta de usuario Q_c, se recuperó la pregunta representativa más similar Q_{sim}^K (como se describe en la Sección 5.1.1). Se registró el tiempo de esta operación de búsqueda de Q_{sim}^K, denotado como $T_{\text{Qsim}}^{(c)}$.

2. Luego, en lugar de buscar en una tabla precalculada E_{pre}, se calcularon *en ese momento* los *embeddings* instruidos $\mathcal{E}(d_i \mid Q_{\text{sim}}^K)$ para los $N_{\text{retrieved}}$ documentos d_i que habían sido recuperados inicialmente para Q_c. Se registró el tiempo $T_{\text{online-emb}}^{(c)}$ que tomó esta operación específica de generación de *embeddings* instruidos.

3. Posteriormente, se realizó el clustering K-Means sobre estos *embeddings* recién calculados y el muestreo de los subconjuntos $\delta_{c,j}$. Se registró el tiempo de esta fase de clustering y muestreo como $T_{\text{cluster-sample}}^{(c)}$.

4. El resto del *pipeline* de *Efficient Speculative RAG* procedió utilizando los subconjuntos $\delta_{c,j}$ derivados.

5. Para estimar la latencia que *Efficient Speculative RAG* habría tenido con un pre-cómputo ideal, el componente $T_{\text{embed-cluster}}^{(c)}$ (referenciado en la Sección 5.2.3) se reconstruyó de la siguiente manera:

- El tiempo $T^{(c)}_{\text{online-emb}}$ (cálculo de *embeddings* instruidos en línea) se *excluyó*, ya que esta operación no ocurriría si los *embeddings* estuvieran precalculados.

- Para simular el costo de acceder y recuperar los $N_{\text{retrieved}}$ *embeddings* precalculados desde una base de datos, se utilizó el tiempo $T^{(c)}_{\text{retrieve}}$ (tiempo de recuperación inicial de los $N_{\text{retrieved}}$ documentos) como un *proxy* conservador. Esta aproximación asume que recuperar $N_{\text{retrieved}}$ vectores de *embedding* precalculados de una base de datos tendría un costo comparable o inferior al de recuperar los documentos originales.

- Por lo tanto, el tiempo $T^{(c)}_{\text{embed-cluster}}$ para *Efficient Speculative RAG* se estimó como la suma de los componentes que sí ocurrirían en un escenario con precómputo:

$$T^{(c)}_{\text{embed-cluster}} \approx T^{(c)}_{\text{Qsim}} + T^{(c)}_{\text{retrieve}} + T^{(c)}_{\text{cluster-sample}}.$$

Esta aproximación, si bien no representa un pre-cómputo real, permitió evaluar el impacto en la precisión del sistema utilizando los *embeddings* conceptualmente correctos (aquellos instruidos por Q^K_{sim}) y, simultáneamente, obtener una estimación razonable de la reducción de latencia al eliminar el costoso cálculo en línea de los *embeddings* instruidos por la consulta del usuario Q_c (como lo hace el Speculative RAG original).

5.2.5. Fine-tuning de drafter

Siguiendo la metodología propuesta por Wang et al. [78] para el entrenamiento del M_{Drafter}, se procedió a realizar un finetuning. El objetivo era instruir al modelo para que, dado un triplete (Q, D) (pregunta, documentos de contexto), generara no solo la respuesta A, sino también una justificación E (rationale) que explicara cómo A se deriva de D. El proceso de entrenamiento buscaba maximizar la verosimilitud $P_{M_{\text{Drafter}}}(A, E \mid Q, D)$.

Para este fin, se utilizaron los mismos datasets empleados en el finetuning del generador de preguntas de KRAQ (ver Sección 3.2.5): Dolly-v2 [13] y MusiQue [73]. Estos datasets se procesaron para crear tripletas de entrenamiento $(Q, D, \text{RespuestaConcatenada})$, donde RespuestaConcatenada incluía tanto la respuesta objetivo como una justificación sintetizada (siguiendo un esquema similar al descrito en Wang et al. [78], Apéndice G, para la generación de justificaciones, en este caso se uso el modelo GPT-4o para generar los racionales de entrenamiento). Se empleó el modelo LLaMA 3.1–8B Instruct como base, y el entrenamiento se realizó con QLoRA bajo una configuración de hiperparámetros similar a la detallada en la Sección 3.2.5 para el generador de KRAQ.

El entrenamiento mostró una convergencia adecuada, como se observa en la curva de pérdida sobre el conjunto de validación (Figura 5.1).

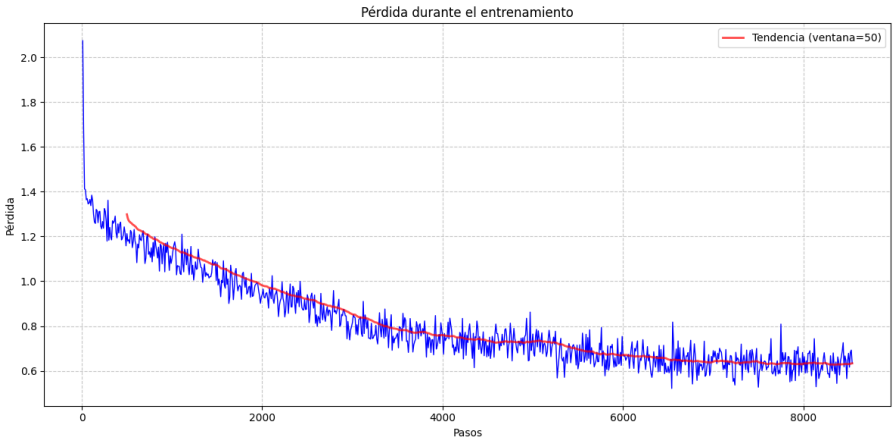

Fig. 5.1: Curva de pérdida en el conjunto de validación durante el proceso de finetuning del modelo M_{Drafter}.

A pesar de la aparente convergencia durante el entrenamiento (Figura 5.1), la evaluación del rendimiento del M_{Drafter} ajustado dentro del *pipeline* completo de Speculative RAG arrojó resultados contraintuitivos. Como se muestra en la Tabla 5.1, el Drafter *fine-tuneado* exhibió un rendimiento inferior en términos de la precisión final del sistema en comparación con el uso directo del modelo LLaMA 3.1–8B Instruct (sin este fine-tuning específico para la tarea de Drafter), tanto en la métrica de Exact Match como en la Evaluación con LLM como Juez.

Tab. 5.1: Comparación de la precisión (EM % y LLM-as-judge % en TriviaQA) del sistema Speculative RAG utilizando el modelo M_{Drafter} con y sin ajuste fino específico para la tarea de Drafter.

Variante de M_{Drafter}	**EM**	**LLM-as-judge**
LLaMA 3.1–8B Instruct (sin fine-tuning)	**77.6**	**82.0**
LLaMA 3.1–8B Instruct (con fine-tuning)	71.6	76.0

Esta disminución en el rendimiento a través de ambas métricas es significativa. Por ejemplo, en EM, la precisión cae de 77.6 % a 71.6 %, y en la evaluación con LLM como Juez, de 82.0 % a 76.0 %. Tal reducción podría atribuirse a varios factores, incluyendo un posible sobreajuste del modelo *fine-tuneado* a la forma específica de los racionales sintetizados durante la creación del *dataset* de entrenamiento, lo que podría haber limitado su capacidad de generalización o afectado negativamente su habilidad para generar la respuesta más concisa o relevante. Otra posibilidad es que el proceso de *fine-tuning* con QLoRA, si bien eficiente en recursos, no haya sido tan efectivo para esta tarea particular como lo habría sido un *fine-tuning* completo de todos los pesos del modelo, como el que se realizó en el trabajo original de Wang et al. [78].

Estrategia final: uso del modelo instruct con formato JSON

Dados los resultados de la Tabla 5.1, se tomó la decisión de no utilizar el modelo M_{Drafter} finetuneado para esta tarea. En su lugar, se optó por emplear el modelo LLaMA 3.1–8B Instruct (en su versión cuantizada AWQ, como se describe en la Sección 3.2.2) sin ninguna modificación adicional de fine-tuning para la función de Drafter.

Para asegurar que la salida del modelo Drafter contuviera de manera diferenciada la respuesta (α_j) y el racional (β_j), y facilitar su posterior procesamiento para el cálculo de scores, se instruyó al modelo para que generara su salida en un formato JSON estructurado. Esto se logró incorporando la siguiente instrucción al final del prompt enviado al M_{Drafter}:

```
Your response must be a valid JSON object with the following format:
{''response'': ''your response here'', ''rationale'': ''your rationale here''}
```

Esta aproximación permite aprovechar las robustas capacidades de seguimiento de instrucciones del modelo LLaMA 3.1–8B Instruct para obtener la información requerida en un formato parseable, sin incurrir en los costes de un fine-tuning adicional que, en este caso, no demostró ser beneficioso. La extracción de α_j y β_j se realiza entonces mediante el parseo del objeto JSON devuelto.

El prompt final para los borradores puede encontrarse en la Sección 7.5

5.2.6. Setup experimental para Efficient Speculative RAG

Para la evaluación de *Efficient Speculative RAG* y su comparación con el algoritmo Speculative RAG original, se estableció una configuración experimental base, compartiendo varios componentes y parámetros con los experimentos de Combined Retrieve RAG (descritos en la Sección 4.2.2).

Componentes del pipeline RAG:

Modelos de generación y verificación: Tanto para la generación de borradores como para la verificación final de respuestas se utilizó el modelo `llama3.1-8b-instruct` en su versión cuantizada con AWQ. En el rol de *Drafter* (M_{Drafter}), este modelo fue responsable de producir múltiples borradores y sus respectivos racionales, operando en su versión instruct, sin ajustes adicionales mediante fine-tuning. Las razones detrás de esta elección se discuten en detalle en la Sección 5.2.5. Por su parte, el mismo modelo actuó como *Verifier* (M_{Verifier}), encargado de evaluar dichos borradores. Ambos modelos corriendo sobre un servidor vLLM. Los detalles técnicos de estas elecciones se desarrollan en la Sección 3.2.2.

Modelo de embeddings: Para el clustering inicial de documentos en el Speculative RAG original (cómputo online de $\mathcal{E}(d_i \mid Q)$) y para la estimacion de la precomputación de embeddings en *Efficient Speculative RAG* (cómputo offline de $E_{\text{pre}}(d_i, Q_j^K)$), se utilizó el modelo de embeddings instruidos InBedder-RoBERTa [59], tal como proponen Wang et al. [78].

Para la recuperación inicial de documentos y la búsqueda de la pregunta de KRAQ más similar (Q_{sim}^K), se empleó el modelo `nomic-embed-text`, gestionado mediante QDrant, manteniendo la consistencia con otros módulos de la tesis (ver Sección 3.2.2.

Corpus documental: Para la evaluación de Efficient Speculative RAG (y su comparación con nuestra implementación del Speculative RAG original), se utilizó como base de conocimiento el mismo corpus de documentos de evidencia E (aproximadamente 5 millones de tokens por dataset, como se describe en la Sección 3.2.2) que fue procesado previamente por el sistema KRAQ. Las preguntas de referencia (Q) utilizadas para poner a prueba estos sistemas RAG se seleccionaron de los datasets originales,

asegurándose de que los documentos de evidencia necesarios para responderlas estuvieran contenidos dentro de este corpus E. El número de preguntas de referencia de cada dataset utilizadas para la **evaluación específica de Efficient Speculative RAG** (y su contraparte) fue el siguiente:

- **TriviaQA:** 300 preguntas.
- **HotPotQA:** 300 preguntas.
- **BioASQ:** 500 preguntas.
- **PubHealth:** 600 preguntas (afirmaciones).

Nuevamente, la selección de el número de preguntas para estos experimentos se debió a consideraciones pragmáticas sobre el tiempo disponible para la ejecución y análisis, buscando una evaluación indicativa del rendimiento.

Parámetros elegidos para la experimentación

Para la evaluación comparativa de *Efficient Speculative RAG* y el Speculative RAG original, se seleccionaron hiperparámetros específicos para cada dataset, buscando un equilibrio entre la calidad de la respuesta y la eficiencia. Estos parámetros clave, que definen el comportamiento del componente de *drafting* y verificación, son: $N_{\text{retrieved}}$ (número de documentos recuperados), k (número de clústeres, que también corresponde al número de documentos en cada subconjunto δ_j), y m (número de subconjuntos de documentos o borradores generados en paralelo). Los valores utilizados para cada dataset se detallan a continuación:

- **BioASQ:** Se configuró con $N_{\text{retrieved}} = 18$ documentos iniciales, $k = 5$ documentos por subconjunto de borrador (y, por lo tanto, 5 clústeres), y se generaron $m = 10$ borradores.
- **HotPotQA:** Se emplearon $N_{\text{retrieved}} = 10$ documentos, $k = 4$ documentos por subconjunto, y $m = 8$ borradores.
- **TriviaQA:** Se utilizaron $N_{\text{retrieved}} = 10$ documentos, $k = 2$ documentos por subconjunto, y $m = 5$ borradores.
- **PubHealth:** Similar a TriviaQA, se configuró con $N_{\text{retrieved}} = 10$ documentos, $k = 2$ documentos por subconjunto, y $m = 5$ borradores.

Estos parámetros se mantuvieron constantes para ambas variantes del algoritmo (original y modificado) en cada dataset respectivo, con el fin de asegurar una comparación justa de su rendimiento en términos de precisión y latencia. Los demás componentes, como los modelos M_{Drafter} y M_{Verifier}, se mantuvieron como se describió en la Sección 5.2.6.

5.2.7. Resultados

Los resultados obtenidos para los cuatro *datasets* se resumen en la Tabla 5.2

Tab. 5.2: Comparación de rendimiento entre nuestra implementación de Speculative RAG (denominada "Original (V.Propia)") y la variante *Efficient Speculative RAG* en términos de Precisión (Exact Match - EM y Evaluación con LLM como Juez - LLM-as-judge, en %) y Latencia Estimada (segundos).

Dataset	EM (%)		LLM-Juez (%)		Latencia (s)	
	Original (V. Propia)	Efficient	Original (V. Propia)	Efficient	Original (V. Propia)	Efficient
HotPotQA	**44.3**	44.0	48.3	**49.0**	3.01	**2.93**
TriviaQA	**77.6**	75.3	82.0	82.0	3.91	**3.51**
PubHealth	**58.3**	58.0	**58.3**	58.0	3.81	**3.36**
BioASQ	**51.4**	50.6	**56.6**	56.4	4.32	**3.97**

5.2.8. Análisis de resultados

Análisis de la precisión de respuesta. El análisis de la precisión de las respuestas generadas revela un panorama altamente competitivo para *Efficient Speculative RAG*, con un rendimiento generalmente muy cercano al de nuestra implementación del Speculative RAG Original.

Al evaluar la corrección semántica y factual mediante el **LLM como Juez**, se observa que *Efficient Speculative RAG* obtiene un ligero incremento en la puntuación para HotPotQA (49.0 % frente al 48.3 % del Original), mientras que en TriviaQA ambas variantes alcanzan un rendimiento idéntico del 82.0 %. Para PubHealth, el Original presenta una mínima ventaja (58.3 % vs. 58.0 %), y en BioASQ, esta diferencia marginal a favor del Original se mantiene (56.6 % vs. 56.4 %). Estos resultados, con diferencias mínimas entre ambas arquitecturas, sugieren que la aproximación de utilizar la pregunta de KRAQ más similar (Q^K_{sim}) como *proxy* para la pregunta del usuario (Q) en la selección de *embeddings* precalculados es, en general, capaz de preservar la calidad semántica de las respuestas a un nivel prácticamente indistinguible del método Original, que utiliza *embeddings* instruidos directamente por Q.

En cuanto a la métrica de **Exact Match**, que evalúa la coincidencia literal, nuestra implementación del Speculative RAG Original tiende a obtener puntuaciones ligeramente superiores de manera consistente. Para HotPotQA, el Original es marginalmente superior (44.3 % vs. 44.0 %). En TriviaQA, la diferencia es más notable, con el Original alcanzando un 77.6 % frente al 75.3 % de *Efficient Speculative RAG*. En PubHealth, el Original también lidera por un estrecho margen (58.3 % vs. 58.0 %). Similarmente, en BioASQ (utilizando el método de EM modificado descrito en la Sección 4.2.3), el Original obtiene un 51.4 % frente al 50.6 % de *Efficient Speculative RAG*. Esta tendencia podría indicar que el uso de *embeddings* instruidos directamente por la pregunta del usuario Q guía el *clustering* de una manera que favorece marginalmente la producción de respuestas que coinciden de forma más literal con las referencias, en comparación con el uso de Q^K_{sim} como *proxy*. No obstante, es importante destacar que las diferencias en EM son generalmente pequeñas, sugiriendo que la pérdida de precisión literal al optar por la variante *Efficient* es limitada.

Análisis de la latencia de inferencia. La principal motivación detrás de *Efficient Speculative RAG* es la reducción de la latencia de inferencia. Los resultados presentados en la Tabla 5.2 validan consistentemente esta hipótesis a través de todos los *datasets* evaluados. Específicamente, la latencia con *Efficient Speculative RAG* se reduce de 3.01s a 2.93s en HotPotQA, lo que representa una mejora del **2.66 %**. Para TriviaQA, la latencia disminuye de 3.91s a 3.51s, marcando una reducción del **10.23 %**. En PubHealth, la mejora es

aún más pronunciada, con una caída de 3.81s a 3.36s, equivalente a una disminución del **11.81 %**. Finalmente, en BioASQ, la latencia desciende de 4.32s a 3.97s, lo que se traduce en una mejora del **8.10 %**. Estas mejoras son atribuibles directamente a la eliminación del costoso cálculo en línea de los *embeddings* instruidos $\mathcal{E}(d_i \mid Q)$ para cada documento recuperado, reemplazándolo por la recuperación (estimada) de *embeddings* precalculados $E_{\text{pre}}(d_i, Q^K_{\text{sim}})$. Este hallazgo es particularmente relevante para aplicaciones que requieren respuestas rápidas y operan bajo restricciones de recursos computacionales.

Conclusión del análisis y balance precisión-eficiencia. En conjunto, *Efficient Speculative RAG* se presenta como una alternativa optimizada y eficaz a nuestra implementación de Speculative RAG. Logra reducciones consistentes y, en algunos casos, significativas en la latencia de inferencia en todos los *datasets* evaluados. Simultáneamente, mantiene un nivel de precisión de respuesta que, si bien puede ser marginalmente inferior en la métrica de Exact Match, es altamente competitivo y prácticamente idéntico en términos de calidad semántica evaluada por LLM como Juez. La ligera disminución observada en las puntuaciones de EM puede considerarse un *trade-off* aceptable y, en muchos casos, despreciable, frente a las significativas ganancias obtenidas en eficiencia. Esta reducción de latencia, que varía entre el 2.7 % y el 11.8 % según el dataset, puede ser crucial para la implementación de sistemas RAG en entornos de producción o con alta demanda. La efectividad de *Efficient Speculative RAG* subraya el potencial de utilizar preguntas representativas generadas por KRAQ para optimizar operaciones costosas en *pipelines* de RAG complejos.

5.2.9. Estudios de ablación

Para los estudios de ablación presentados en esta sección, se utilizó la misma cantidad de preguntas de referencia de los datasets que en los experimentos principales de Efficient Speculative RAG, con el fin de asegurar la comparabilidad.

Impacto del modelo generador de preguntas de KRAQ

Para investigar la sensibilidad de *Efficient Speculative RAG* a la calidad de las preguntas representativas $\mathcal{Q}^{\mathcal{K}}$ utilizadas para la selección de embeddings precalculados, se realizó el siguiente estudio de ablación. Este estudio se centró en variar el método de generación de las preguntas de KRAQ, manteniendo el resto del *pipeline* de *Efficient Speculative RAG* constante. Se evaluó el impacto en la precisión final del sistema utilizando el dataset TriviaQA, mediante las métricas de Exact Match y Evaluación con LLM como Juez.

Las variantes del generador de preguntas de KRAQ consideradas fueron las mismas que en la evaluación de KRAQ (ver Sección 3.2.7):

1. **Efficient (Fine-tuned KRAQ):** Utiliza el modelo generador de preguntas de KRAQ *fine-tuneado* específicamente para la tarea, como se describe en la Sección 3.2.5. Esta es la configuración estándar de *Efficient Speculative RAG* en los resultados principales.

2. **Efficient (Instruct KRAQ):** Utiliza el modelo LLaMA 3.1–8B Instruct (sin el *fine-tuning* de KRAQ) para generar preguntas a partir de los resúmenes comunitarios.

3. **Efficient (Random KRAQ):** Utiliza el *baseline* que genera preguntas a partir de *chunks* seleccionados aleatoriamente del corpus (Sección 3.2.3).

Los resultados de precisión en TriviaQA para estas variantes se presentan en la Tabla 5.3.

Tab. 5.3: Impacto del método de generación de preguntas de KRAQ en la precisión (EM % y LLM-as-judge %) de Efficient Speculative RAG en el dataset TriviaQA.

Variante del Generador de Preguntas de KRAQ	**EM**	**LLM-as-judge**
Efficient (Fine-tuned KRAQ)	**75.33**	**82.00**
Efficient (Random KRAQ)	73.33	81.00
Efficient (Instruct KRAQ)	73.00	77.00

Análisis. Los resultados del estudio de ablación, presentados en la Tabla 5.3, indican que la calidad y el método de generación de las preguntas de KRAQ tienen un impacto directo y significativo en el rendimiento de *Efficient Speculative RAG*.

El uso del modelo de KRAQ con ***fine-tuning*** específico (Efficient Fine-tuned KRAQ) demuestra ser consistentemente superior, alcanzando la mayor precisión tanto en EM (75.33 %) como en la evaluación con LLM como Juez (82.00 %). Esto sugiere que las preguntas más relevantes permiten una selección más adecuada de los *embeddings* precalculados $E_{\text{pre}}(d_i, Q^K_{\text{sim}})$. Una Q^K_{sim} de mayor calidad, que se asemeja más en intención y contenido a la pregunta real del usuario Q, resulta en un *clustering* de documentos más pertinente para la pregunta original. Esta mejor agrupación, a su vez, impacta positivamente en la calidad de los borradores generados y, consecuentemente, en la precisión de la respuesta final.

Cuando se utilizan preguntas generadas por el método **Random KRAQ** (Efficient Random KRAQ), que se derivan de *chunks* aleatorios, la precisión en EM disminuye a 73.33 % y en LLM-Juez a 81.00 %. Aunque estas preguntas provienen directamente del corpus, carecen del análisis estructural y la síntesis temática que KRAQ introduce.

La variante que utiliza el modelo **Instruct KRAQ** (Efficient Instruct KRAQ), es decir, el modelo LLaMA 3.1–8B Instruct aplicado a los resúmenes comunitarios de GraphRAG sin el *fine-tuning* específico de KRAQ, muestra el rendimiento más bajo: 73.00 % en EM y 77.00 % en LLM-Juez. Esto subraya que, si bien los resúmenes comunitarios proporcionan una buena base, el modelo de lenguaje necesita ser específicamente adaptado para transformar estos resúmenes en preguntas que sean efectivas para guiar la selección de *embeddings* en el contexto de *Efficient Speculative RAG*.

Impacto del número de documentos recuperados en la latencia

Para investigar más a fondo el beneficio en latencia de *Efficient Speculative RAG*, especialmente en escenarios donde el costo del cálculo de *embeddings* instruidos en línea podría ser más pronunciado, se realizó un estudio de ablación adicional. En este estudio, se varió el número de documentos recuperados inicialmente ($N_{\text{retrieved}}$) antes de la etapa de *clustering*, mientras se mantenían fijos otros parámetros del algoritmo Speculative RAG.

Específicamente, para el *dataset* HotPotQA, se fijó el número de clústeres (y, por ende, el número de documentos por subconjunto de borrador) en $k = 3$ y el número de borradores generados en $m = 8$. Luego, se comparó la latencia de inferencia estimada (según la Ecuación 5.5) entre nuestra implementación del Speculative RAG Original y la variante *Efficient Speculative RAG* para diferentes valores de $N_{\text{retrieved}} \in \{10, 15, 20\}$.

La hipótesis era que, al aumentar $N_{\text{retrieved}}$, el costo de calcular los *embeddings* instruidos en línea para el Speculative RAG Original se incrementaría de mayor manera que el costo (estimado) de recuperar un mayor número de *embeddings* precalculados en *Efficient Speculative RAG*, resultando en una mayor diferencia de tiempo ahorrado a favor de la variante eficiente. Los resultados de latencia obtenidos se presentan en la Figura 5.2.

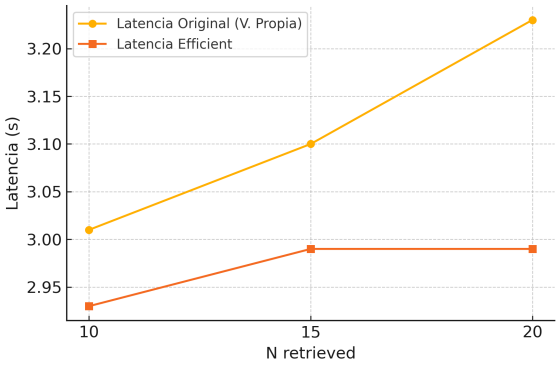

Fig. 5.2: Impacto del número de documentos recuperados inicialmente ($N_{\text{retrieved}}$) en la latencia.

Análisis de los resultados del estudio de ablación. Los resultados presentados en la figura 5.2 confirman la hipótesis planteada. A medida que aumenta el número de documentos recuperados inicialmente ($N_{\text{retrieved}}$), la ventaja en latencia de *Efficient Speculative RAG* sobre el Speculative RAG Original se vuelve más pronunciada:

- Con $N_{\text{retrieved}} = 10$, la reducción de latencia es del 2.7 %..
- Al aumentar a $N_{\text{retrieved}} = 15$, la reducción de latencia se incrementa al 3.5 %.
- Con $N_{\text{retrieved}} = 20$, la variante eficiente logra una reducción de latencia del 7.4 %. Notablemente, la latencia de *Efficient Speculative RAG* se mantiene prácticamente constante (2.93s a 2.99s) incluso al duplicar el número de documentos iniciales de 10 a 20, mientras que la latencia del Speculative RAG Original aumenta de 3.01s a 3.23s.

Este comportamiento es el esperado: el costo principal para *Efficient Speculative RAG* en la etapa de $T^{(c)}_{\text{embed-cluster}}$ (después de la selección de Q^{K}_{sim}) radica en la recuperación de los *embeddings* precalculados y el subsiguiente *clustering* y muestreo. Si bien la recuperación de más *embeddings* precalculados tiene un costo, este es presumiblemente menor y más constante que el costo de generar en línea un mayor número de *embeddings* instruidos con un modelo como InBedder-RoBERTa, como debe hacer el Speculative RAG Original.

Este estudio de ablación marca la escalabilidad de la ventaja en latencia de *Efficient Speculative RAG*, particularmente en escenarios donde se podría considerar recuperar un conjunto inicial más grande de documentos con el fin de capturar una mayor diversidad de perspectivas. La capacidad de *Efficient Speculative RAG* de mantener una latencia relativamente estable en la etapa de $T^{(c)}_{\text{embed-cluster}}$ al variar $N_{\text{retrieved}}$ es una ventaja significativa.

6. CONCLUSIONES GENERALES

6.1. Conclusiones

En la presente tesis se investigó y desarrolló una nueva metodología denominada KRAQ (*Knowledge-graph Representative Automatic Questions*), con el principal objetivo de optimizar los sistemas RAG. Este esfuerzo se originó como respuesta a las limitaciones reconocidas de los LLMs, como su conocimiento estático, su propensión a las alucinaciones y su sesgo de posición, así como a los desafíos en los sistemas RAG, que incluyen la considerable carga computacional y la frecuente recuperación de información con alta similitud superficial pero baja diversidad semántica.

La propuesta central de KRAQ se articuló en torno a la hipótesis de que un conjunto de preguntas representativas de un corpus, generado de manera informada y estructurada, podría actuar como un activo para mejorar los *pipelines* de RAG. Para ello, KRAQ se diseñó para capturar la estructura semántica de un corpus documental mediante la construcción de un grafo de conocimiento. Sobre este grafo, se aplicaron algoritmos de detección de comunidades para identificar agrupaciones temáticas cohesivas, a partir de las cuales se generaron resúmenes textuales. El componente innovador de KRAQ radica en la transformación de estos resúmenes comunitarios en un conjunto de preguntas representativas, utilizando para ello un modelo de lenguaje (*fine-tuneado*).

La evaluación experimental de KRAQ demostró su eficacia. Al comparar las preguntas generadas con aquellas de referencia en *datasets* estándar como TriviaQA, HotPotQA, BioASQ y PubHealth, se constató que KRAQ alcanzó niveles de relevancia semántica (medidos con Relevance y Relevance@τ, metricas basadas en BERTScore) significativamente superiores a los de un *baseline* que genera preguntas a partir de fragmentos aleatorios del corpus. Es crucial destacar que KRAQ, con su modelo *fine-tuned*, también superó consistentemente a una variante que utilizaba el mismo *pipeline* de resúmenes comunitarios pero empleaba un modelo de lenguaje instruct-tuneado sin ajuste específico para la generación de preguntas. Esto subraya el valor del *fine-tuning* para alinear el modelo generador con la tarea de producir preguntas temáticamente representativas a partir de los resúmenes.

Las contribuciones prácticas de esta tesis se materializaron a través de dos aplicaciones directas de las preguntas generadas por KRAQ, ambas orientadas a mitigar limitaciones específicas de los sistemas RAG:

1. **Combined Retrieve RAG:** Se propuso un algoritmo de recuperación que enriquece la consulta original del usuario con preguntas similares generadas por KRAQ, con el fin de diversificar el conjunto de documentos recuperados. Los experimentos en esta línea mostraron mejoras consistentes en la precisión de las respuestas. Específicamente, se observaron incrementos de hasta un 3 % en la métrica de Exact Match y mejoras generalizadas en la evaluación semántica mediante LLM como Juez. Estos resultados sugieren que la diversificación contextual, guiada por preguntas temáticamente relevantes y estructuralmente derivadas, beneficia la calidad de la generación final.

2. **Efficient Speculative RAG:** Se desarrolló una optimización para el *framework* Speculative RAG, donde las preguntas de KRAQ se emplearon para permitir el pre-

cómputo de los *embeddings* instruidos, un componente que representa un cuello de botella computacional en el algoritmo original. Las pruebas experimentales indicaron una reducción notable en la latencia de inferencia, alcanzando hasta un 10 % de disminución, sin una disminución significativa en la calidad de las respuestas. Este hallazgo es prometedor para la implementación de sistemas RAG avanzados en entornos con alta demanda.

Los estudios de ablación realizados reforzaron la tesis de que la calidad intrínseca de las preguntas generadas por KRAQ y la especificidad del *fine-tuning* del modelo generador son determinantes para el éxito de las optimizaciones propuestas.

En síntesis, esta tesis aporta evidencia empírica de que la integración de grafos de conocimiento como base para una generación automática de preguntas representativas constituye una estrategia efectiva y prometedora. KRAQ no solo se presenta como una herramienta para la condensación y representación del conocimiento de un corpus, sino que se establece como un componente funcional capaz de mejorar el rendimiento de los sistemas RAG, abriendo nuevas vías hacia la construcción de sistemas RAG más precisos y eficientes..

6.2. Trabajos futuros

Los resultados y la metodología desarrollada en esta tesis abren diversas líneas de investigación y desarrollo futuro que podrían expandir y refinar las contribuciones presentadas:

1. **Optimización de KRAQ:**
   - Explorar el uso de los "findings" o afirmaciones detalladas generadas por GraphRAG (además de los resúmenes comunitarios) y de otras caracteristicas del grafo de conocimiento (como las propias entidades de una comunidad) como entrada para el modelo generador de preguntas, buscando preguntas más específicas.
   - Investigar el impacto de diferentes algoritmos de detección de comunidades en la calidad y representatividad de las preguntas finales.
   - Experimentar con modelos de lenguaje de mayor capacidad para la tarea de generación de preguntas a partir de resúmenes.
2. **Mejoras en Combined Retrieve RAG:**
   - Desarrollar estrategias más sofisticadas para la selección y ponderación de las preguntas de KRAQ en la recuperación combinada, por ejemplo, adaptando dinámicamente el número de preguntas (n) o la proporción (α) en función de la complejidad o ambigüedad de la consulta del usuario.
   - Integrar mecanismos de re-ranking de los documentos recuperados que consideren tanto la similitud con la pregunta original como con las preguntas de KRAQ.
   - Investigar como podría utilizarse la jerarquía de las preguntas (en relación al nivel de la comunidad con la que fueron generadas) para optimizar el RAG tradicional.

3. **Optimización y expansión de Efficient Speculative RAG:**

   - Investigar estrategias para la optimización del pre-cómputo de *embeddings* instruidos. Esto incluye validar empíricamente la viabilidad de un pre-cómputo selectivo, donde solo se generen *embeddings* instruidos por una pregunta de KRAQ (Q_j^K) para un subconjunto de documentos del corpus que sean semánticamente más afines a Q_j^K (e.g., los M_s más relevantes). El objetivo sería reducir drásticamente el costo del pre-cómputo sin afectar negativamente la calidad del *clustering* y la respuesta final.

4. **Nuevas aplicaciones de las preguntas de KRAQ:**

   - Utilizar el conjunto de preguntas de KRAQ como base para la generación automática de datasets de entrenamiento para sistemas de QA específicos a un corpus.
   - Desarrollar herramientas de exploración de corpus donde las preguntas de KRAQ actúen como puntos de entrada o sugerencias para la navegación temática.

7. APÉNDICE

7.1. Prompt para la Síntesis de Resúmenes Comunitarios (Etapa g)

El siguiente *prompt* se utilizó con un modelo LLM (GPT-4o) para generar resúmenes temáticos R a partir de una pregunta de referencia Q y un conjunto de evidencia E. El objetivo era obtener un resumen del contenido de E sin revelar Q.

```
Given this evidence and knowing that we want to generate a question about
{target_question}, create a community-style summary that:
1. Begins with ''This community centers around...'' or ''This community focuses on...''
2. Describes the main group, organization, or topic that connects the entities
3. Lists key members, figures, or elements in the community
4. Emphasizes relationships and connections between these elements
5. IMPORTANT: Do not reference or hint at the specific question that will be asked
6. Make the summary concise, maximum 5 sentences.

Follow this style:
Example: ''This community centers around the Order of the Phoenix, a secret
organization in the Harry Potter series dedicated to combating dark forces,
particularly Voldemort and his followers. Key members include Harry Potter,
Kingsley Shacklebolt, Alastor Moody, and Nymphadora Tonks. The relationships
among these characters highlight their collaborative efforts against dark magic,
notable events such as battles against the Death Eaters, and the complexities
of their interactions, including issues of trust and loyalty.''

For the given evidence, create a similar community-focused summary that
describes the entities and their relationships without revealing the
specific question that will be asked:

Evidence:
{evidence}
```

Donde `{target_question}` era reemplazado por la pregunta Q y `{evidence}` por el texto de evidencia E.

7.2. Prompt para la Generación de Preguntas (Modelo f_θ)

Este *prompt* se utilizó como plantilla para el *fine-tuning* del modelo LLaMA 3.1–8B Instruct y para la inferencia. El objetivo es generar una pregunta natural Q_{gen} a partir de un resumen comunitario R.

```
Given this summary of a document collection, generate a natural question that a
person might ask when looking for this information. The question should be:
    - Simple and straightforward
    - Written in conversational language
```

```
    - Focused on the main topic or event
    - Something a real person would ask when searching for information

Now generate a question for this summary:
{summary}
```

Durante el *fine-tuning*, `{summary}` era reemplazado por el resumen R generado en la etapa anterior, y la respuesta esperada (del rol *assistant*) era la pregunta original Q. Para la inferencia, solo se proporciona el *prompt* con el resumen, y el modelo genera la pregunta. La estructura de datos para el entrenamiento seguía el formato:

```
{
    ''messages'': [
        {''role'': ''user'', ''content'': ''PROMPT_CON_SUMMARY_INSERTADO''},
        {''role'': ''assistant'', ''content'': ''PREGUNTA_OBJETIVO''}
    ]
}
```

7.3. Prompt para la Generación de Preguntas del Baseline de KRAQ

El siguiente *prompt* se utilizó con un LLM para generar una pregunta a partir de la concatenación de m fragmentos (*chunks*) de texto seleccionados aleatoriamente del corpus. Este proceso constituye el *baseline* para la evaluación de la calidad de las preguntas generadas por KRAQ, como se describe en la Sección 3.2.3 y el Algoritmo 3.

```
Given these random fragments, generate a natural, concise question that someone
might ask about the themes or topics present in these passages. The question
should:
- Be short and to the point
- Focus on a common theme or interesting connection between the fragments
- Be something a real person would naturally ask
- Not be too complex or academic

Fragments:
{combined_content}

Generate only ONE concise question:
```

Donde `{combined_content}` era reemplazado por el texto resultante de la concatenación de los m *chunks* aleatorios. El objetivo era que el LLM generara una única pregunta concisa basada en el contenido agregado de estos fragmentos.

7.4. Prompt para la Generación de Respuestas en Sistemas RAG

El siguiente *prompt* base se utilizó para instruir al LLM generador (Llama-3.1-8B-Instruct en esta tesis) en la etapa final de los diferentes *pipelines* RAG evaluados (e.g., RAG Tradicional, Combined Retrieve RAG, y como modelo M_{Verifier} o M_{Drafter} con adaptaciones en Speculative RAG). El objetivo era que el LLM generara una respuesta a la consulta del usuario, utilizando el contexto recuperado como evidencia.

```
Below is an instruction that describes a task. Write a response using the
evidence provided for it and state your explanation supporting your response.

### Evidence:
{context}

### Instruction:
{query}
```

Donde los *placeholders* se reemplazaban de la siguiente manera:

- `{context}`: Se insertaba el conjunto de documentos o fragmentos de texto $\mathcal{D}$ que fueron recuperados por el componente *retriever* del sistema RAG. Estos documentos se concatenaban para formar un único bloque de texto contextual.
- `{query}`: Se insertaba la pregunta original Q formulada por el usuario (o la pregunta de referencia del dataset en el contexto experimental).

Se esperaba que el LLM utilizara la "Evidence" (el contexto recuperado) para responder a la "Instruction" (la consulta), y la instrucción adicional de "state your explanation supporting your response" tenía como objetivo fomentar respuestas más fundamentadas, aunque el análisis principal de esta tesis se centró en la respuesta directa y no en la calidad de la explicación generada, salvo en el contexto específico del cálculo de *scores* para Speculative RAG donde los "racionales" juegan un papel. La respuesta directa del modelo a la consulta `{query}` se consideró como A_{gen}.

7.5. Prompt para la Generación de Borradores (M_{Drafter}) en Speculative RAG

El siguiente *prompt* se utilizó para instruir al modelo M_{Drafter} (Llama-3.1-8B-Instruct en esta tesis, como se detalla en la Sección 5.2.5) dentro del *framework* Speculative RAG. El objetivo era que el modelo, dada una instrucción (pregunta) y un conjunto de evidencia (subconjunto de documentos δ_j), generara tanto una respuesta candidata (α_j) como una justificación o racional (β_j) que la respaldara. La salida se solicitó en formato JSON para facilitar su posterior procesamiento (Explicación de la razón de esto en la Sección 5.2.5).

```
Response to the instruction. Also provide a concise rationale that justifies the
response.

### Instruction:
{instruction}

### Evidence:
{evidence}

Your response must be a valid JSON object with the following format:
{{''response'': ''your response here'', ''rationale'': ''your rationale here''}}
```

Donde los *placeholders* se reemplazaban de la siguiente manera:

- `{instruction}`: Se insertaba la pregunta original del usuario Q (o la pregunta de referencia del dataset en el contexto experimental).

- `{evidence}`: Se insertaba el subconjunto de documentos δ_j (muestreado a partir de los clústeres de documentos recuperados) que debía servir como base para la respuesta y la justificación.

El modelo M_{Drafter} debía generar un objeto JSON que contuviera dos claves: `''response''` (mapeada a α_j) y `''rationale''` (mapeada a β_j). Esta estructura permitía una extracción sencilla de ambos componentes para su uso en el cálculo de los *scores* de Speculative RAG.

7.6. Prompt para el Modelo Verificador (M_{Verifier}) en Speculative RAG

En el *framework* Speculative RAG (tanto en la versión original implementada como en *Efficient Speculative RAG*), el modelo verificador (M_{Verifier}) juega un papel crucial en la evaluación de los borradores de respuesta y sus justificaciones (racionales) generados por el modelo M_{Drafter}. Este proceso de verificación contribuye al cálculo del *score* $\rho_j^{\text{self-reflect}}$ (ver Sección 2.2.3).

El siguiente *prompt* se utilizó para instruir al modelo M_{Verifier} (Llama-3.1-8B-Instruct en esta tesis):

```
Instruction: {instruction}

Response: {response}

Rationale: {rationale}

Is the rationale good enough to support the answer?

You must respond with only a single word: ''Yes'' or ''No''.
Do not include any explanation or additional text.
```

Donde los *placeholders* se reemplazaban de la siguiente manera:

- `{instruction}`: Se insertaba la pregunta original del usuario Q (o la pregunta de referencia del dataset).

- `{response}`: Se insertaba el borrador de respuesta α_j generado por el modelo M_{Drafter}.

- `{rationale}`: Se insertaba la justificación o racional β_j generado por el modelo M_{Drafter} para acompañar a α_j.

Se esperaba que el modelo M_{Verifier} evaluara si el `{rationale}` proporcionado era un soporte adecuado y suficiente para la `{response}` en el contexto de la `{instruction}`. La respuesta del M_{Verifier} (restringida a "Yes" o "No") y las log-probabilidades asociadas a esta respuesta se utilizaban luego para calcular el componente $\rho_j^{\text{self-reflect}}$ del *score* final del borrador.

7.7. Prompt para la Evaluación con LLM como Juez

Para la evaluación de la calidad semántica y factual de las respuestas generadas por los sistemas RAG, se empleó un LLM como juez. El siguiente *prompt* fue proporcionado al LLM (Llama-3.1-8B-Instruct en esta tesis) para cada instancia de evaluación, como se describe en la Sección 4.2.1.

```
You are an expert evaluator for question answering systems. Your task is to
determine if the generated answer correctly responds to the question according
to the reference answer.

Question: {question}
Generated Answer: {generated_answer}
Reference Answer: {reference_answer}

The reference answer represents the truth. The generated answer must match the
meaning of the reference answer to be considered correct. If the generated
answer is more specific but the core meaning is the same, it is also
considered correct.

Respond with ONLY a single digit:
1 - CORRECT:
0 - INCORRECT:

Your verdict (just the digit 1 or 0):
```

Donde los *placeholders* se reemplazaban de la siguiente manera:

- `{question}`: Se insertaba el texto de la pregunta de referencia Q del dataset.
- `{generated_answer}`: Se insertaba el texto de la respuesta A_{gen} producida por el sistema RAG evaluado.
- `{reference_answer}`: Se insertaba el texto de la respuesta de referencia A del dataset.

El LLM-Juez debía responder únicamente con el dígito ”1” si consideraba que la respuesta generada era correcta y semánticamente equivalente a la respuesta de referencia, o con ”0” en caso contrario. La proporción de respuestas ”1” sobre el total de instancias evaluadas constituyó la puntuación de la métrica LLM-como-Juez.

Bibliografía

[1] Amazon (2024). Build an end-to-end rag solution using amazon bedrock knowledge bases and aws cloudformation. AWS Machine Learning Blog.

[2] Barnett, S., Kurniawan, S., Thudumu, S., Brannelly, Z., and Abdelrazek, M. (2024). Seven failure points when engineering a retrieval augmented generation system. In *Proceedings of the 3rd International Conference on AI Engineering — Software Engineering for AI (CAIN 2024)*.

[3] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

[4] Bi, S., Cheng, X., Li, Y.-F., Wang, Y., and Qi, G. (2020). Knowledge-enriched, type-constrained and grammar-guided question generation over knowledge bases. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, pages 2776–2786, Barcelona, Spain (Online). International Committee on Computational Linguistics.

[5] Blondel, V. D., Guillaume, J. L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008.

[6] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

[7] Bommasani, R. et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

[8] Brown, T., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*.

[9] Chen, X. et al. (2021). Human-like visual question generation with diverse and meaningful questions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[10] Chen, Y., Wu, L., and Zaki, M. J. (2020). Toward subgraph-guided knowledge graph question generation with graph neural networks. *arXiv preprint arXiv:2004.06015*.

[11] Chowdhery, A. et al. (2022). PaLM: Scaling language models with pathways. *arXiv preprint arXiv:2204.02311*.

[12] Christen, P. (2012). *Data Matching: Concepts and Techniques*. Springer.

[13] Conover, M., Hayes, M., Mathur, A., Xie, J., Wan, J., Shah, S., Ghodsi, A., Wendell, P., Zaharia, M., and Xin, R. (2023). Free dolly: Introducing the world's first truly open instruction-tuned llm. Accessed on .

[14] Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs. *arXiv preprint arXiv:2305.14314*.

[15] Devlin, J., Chang, M. W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

[16] Do, C. P. et al. (2023). Modeling what-to-ask and how-to-ask for answer-unaware conversational question generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*. Also arXiv:2305.03088.

[17] Dong, L. et al. (2019). Unified language model pre-training for natural language understanding and generation. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*.

[18] Du, X., Shao, J., and Cardie, C. (2017). Learning to ask: Neural question generation for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1342–1352, Vancouver, Canada. Association for Computational Linguistics.

[19] Edge, D. et al. (2024). Graphrag: A graph-based retrieval-augmented generation method / from local to global: A graphrag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

[20] Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate record detection: A survey.

[21] Fei, Z., Zhang, Q., Gui, T., Liang, D., Wang, S., Wu, W., and Huang, X. (2022). Cqg: A simple and effective controlled generation framework for multi-hop question generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6896–6906, Dublin, Ireland. Association for Computational Linguistics.

[22] Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486(3–5):75–174.

[23] Gonzalez, H., Dugan, L., Miltsakaki, E., Cui, Z., Ren, J., Li, B., Upadhyay, S., Ginsberg, E., and Callison-Burch, C. (2023). Enhancing human summaries for question-answer generation in education. In *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*, pages 108–118, Toronto, Canada. Association for Computational Linguistics.

[24] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[25] Guo, L. et al. (2024). A survey on neural question generation: Methods, applications, and prospects. In *Findings of the Association for Computational Linguistics: ACL 2024*. Also arXiv:2402.18267v2.

[26] Haviv, A. et al. (2023). Nomic embed: Open-source embedding models for text and code.

[27] Hu, E. J., Shen, Y., Wallis, P., et al. (2022). LoRA: Low-rank adaptation of large language models.

[28] Ivanov, A., Boytsov, L., and Mosichev, I. (2021). Qdrant: Vector search engine with filtering capabilities. https://qdrant.tech.

[29] Izacard, G. and Grave, E. (2021). Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282. Published at EACL 2021.*

[30] Ji, S., Pan, S., Cambria, E., Marttinen, P., and Yu, P. S. (2022). A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514.

[31] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Chen, D., Dai, W., Chan, H. S., Madotto, A., and Fung, P. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys (CSUR)*, 55(12):Article 248.

[32] Jin, D. et al. (2023). A survey of community detection approaches: From statistical modeling to deep learning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 35(2):1149–1170.

[33] Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. (2017). TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL).*

[34] Kaddour, J. et al. (2023). Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169.*

[35] Kapoor, S. (2024). Large language models must be taught to know what they don't know. Details of publication/preprint needed, add URL when available.

[36] Kotonya, N. and Toni, F. (2020). Explainable automated fact-checking for public health claims. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP).*

[37] Kuratov, Y. et al. (2024). In search of needles in a 11m haystack: Recurrent memory finds what LLMs miss. *https://arxiv.org/pdf/2402.10790.*

[38] Lampinen, A. K., Chaudhry, A., Chan, S. C., Wild, C., Wan, D., Ku, A., Bornschein, J., Pascanu, R., Shanahan, M., and McClelland, J. L. (2025). On the generalization of language models from in-context learning and finetuning: a controlled study. *arXiv preprint arXiv:2505.00661v2.*

[39] Lewis, M. et al. (2020a). BART: Denoising sequence-to-sequence pre-training for natural language generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL).*

[40] Lewis, P., Perez, E., Piktus, A., et al. (2020b). Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33 (NeurIPS).*

[41] Li, Z., Wang, J., Jiang, Z., Mao, H., Chen, Z., Du, J., Zhang, Y., Zhang, F., Zhang, D., and Liu, Y. (2024). Dmqr-rag: Diverse multi-query rewriting for retrieval-augmented generation. *arXiv preprint arXiv:2411.13154*.

[42] Liang, Y., Wang, J., Zhu, H., Wang, L., Qian, W., and Lan, Y. (2023). Prompting large language models with chain-of-thought for few-shot knowledge base question generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4329–4343, Singapore. Association for Computational Linguistics.

[43] Lin, Z., Tang, R., Mu, J., Sun, M., and Lin, X. (2023). AWQ: Activation-aware weight quantization for LLMs. *arXiv preprint arXiv:2306.00978*.

[44] Liu, J., Muennighoff, N., Varshney, L. R., et al. (2024). Benchmarking instruction-tuned models with open llm leaderboards. *arXiv preprint arXiv:2402.12345*.

[45] Liu, N. et al. (2023a). Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

[46] Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., and Zhu, C. (2023b). G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

[47] MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In Le Cam, L. M. and Neyman, J., editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to Probability Theory*, pages 281–297, Berkeley, CA. University of California Press.

[48] Madaan, A. et al. (2023). Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.

[49] Microsoft (2024). GraphRAG: Open Source Library for Knowledge Graph-based RAG. GitHub repository.

[50] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781. Published at ICLR 2013 Workshop*.

[51] Muennighoff, N. et al. (2022). MTEB: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

[52] Newman, M. E. J. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences (PNAS)*, 103(23):8577–8582.

[53] Nickel, M., Murphy, K., Tresp, V., and Gabrilovich, E. (2016). A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33.

[54] Noorbakhsh, M. et al. (2024). Savaal: Scalable, concept-oriented question generation for learning. *arXiv preprint arXiv:2402.12477v1*.

[55] OpenAI (2023). GPT-4 technical report. https://arxiv.org/abs/2303.08774. Also openai.com/research/gpt-4.

[56] Ouyang, L., Wu, J., Jiang, X., et al. (2022). Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155. Published at NeurIPS 2022.*

[57] Pan, B., Li, H., Yao, Z., Cai, D., and Sun, H. (2019). Reinforced dynamic reasoning for conversational question generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL).* Referenced as Pan et al. (2019a) in text.

[58] Pan, L., Xie, Y., Feng, Y., Chua, T.-S., and Kan, M.-Y. (2020). Semantic graphs for generating deep questions. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL).*

[59] Peng, L. et al. (2024). Answer is all you need: Instruction-following text embedding via answering the question. *arXiv preprint arXiv:2402.09642.*

[60] Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).*

[61] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT).*

[62] Qiao, X. et al. (2022). Reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10511.*

[63] Rackauckas, Z. (2024). RAG-Fusion: A new take on retrieval-augmented generation. *arXiv preprint arXiv:2402.03367.*

[64] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. OpenAI Technical Report.

[65] Raffel, C. et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research (JMLR)*, 21(140):1–67.

[66] Rajpurkar, P. et al. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP).*

[67] Rozière, B., Chen, S., Jain, S., Zeng, M., et al. (2023). vLLM: Easy, fast, and cheap LLM serving with pagedattention. *arXiv preprint arXiv:2309.06180.*

[68] Shoeybi, M. et al. (2019). Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053.*

[69] Su, H., Kumar, S. H., Mazumder, S., Chen, W., Manuvinakurike, R., Okur, E., Sahay, S., Nachman, L., Chen, S.-T., and yi Lee, H. (2023). Position matters! empirical study of order effect in knowledge-grounded dialogue. In *Proceedings of the Third DialDoc Workshop on Document-grounded Dialogue and Conversational Question Answering*, pages 28–38, Toronto, Canada. Association for Computational Linguistics.

[70] Touvron, H. et al. (2023). LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

[71] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., et al. (2024). LLaMA 3: Open foundation and instruction models. https://ai.meta.com/research/publications/llama-3-open-foundation-and-instruction-models/.

[72] Traag, V. A., Waltman, L., and van Eck, N. J. (2019). From louvain to leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233. Also arXiv:1810.08473.

[73] Trivedi, H., Balasubramanian, N., Khot, T., and Sabharwal, A. (2022). Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics (TACL)*, 10:539–556.

[74] Tsatsaronis, G. et al. (2015). An overview of the BioASQ large-scale biomedical semantic indexing and question answering competition. *BMC Bioinformatics*, 16(Suppl 1):S138.

[75] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NeurIPS)*. Also arXiv preprint arXiv:1706.03762.

[76] Wang, C. et al. (2020). Neural question generation with answer pivot. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. Referenced as Wang et al. (2020a) in text.

[77] Wang, D., Joshi, G., and Wornell, G. W. (2019). Efficient straggler replication in large-scale parallel computing. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 4(2):1–24.

[78] Wang, Z. et al. (2025). Speculative RAG: Enhancing retrieval augmented generation through drafting.

[79] Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442.

[80] Wei, J., Tay, Y., Barham, P., et al. (2022). Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682. Published at TMLR.*

[81] Xia, A. et al. (2023). Improving question generation with multi-level content planning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[82] Xia, H., Yang, Z., Dong, Q., Wang, P., Li, Y., Ge, T., Liu, T., Li, W., and Sui, Z. (2024). Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding. *arXiv preprint arXiv:2401.07851*.

[83] Xie, T. et al. (2021). Visual question generation for class acquisition of unknown objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[84] Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. (2018). HotpotQA: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600. Published at EMNLP 2018.*

[85] Yasunaga, M., Ren, H., Leskovec, J., and Liang, P. (2021). Qa-gnn: Reasoning with language models and knowledge graphs for question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 535–546.

[86] Yasunaga, M., Wu, Z., Liu, S., Moradshahi, M., and Liang, P. (2023). Linking language models to knowledge graphs for open-domain question answering. In *Advances in Neural Information Processing Systems 36 (NeurIPS).*

[87] Zhang, Q., Qiu, L., Wang, Y., and Ren, X. (2023). Task-aware knowledge graph construction for language model reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL).*

[88] Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2020). BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations (ICLR).*

[89] Zheng, X. et al. (2023). Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *arXiv preprint arXiv:2306.05685.*

[90] Zhu, Y. et al. (2024). LLMs for knowledge graph construction and reasoning: Recent capabilities and future opportunities. *arXiv preprint arXiv:2305.13168.*