Desarrollo de un RAG (Retrieval Augmented Generation) para desarrolladores de MuleSoft

Daniel Montoya García

27 de junio de 2026

Resumen – El objetivo de este proyecto es diseñar e implementar un sistema de Retrieval-Augmented Generation (RAG) orientado a desarrolladores de MuleSoft. El sistema permite a los desarrolladores realizar consultas en lenguaje natural sobre la tecnología y obtener respuestas precisas basadas en la documentación oficial, almacenada en una base de datos vectorial. El trabajo se abordó en dos fases. Primero se construyó un RAG tradicional que recuperaba fragmentos relevantes de la documentación y los entregaba a un modelo de lenguaje para generar la respuesta. Después se desarrolló un RAG agéntico, capaz de razonar sobre las consultas, decidir qué herramientas utilizar (documentación indexada, búsqueda web o catálogo de Anypoint Exchange) e iterar hasta producir una respuesta satisfactoria. Finalmente, ambos sistemas se compararon mediante una evaluación que mide la calidad de las respuestas y el coste computacional de cada enfoque.

Palabras clave – Agente de IA, Base de datos vectorial, Chunk, Embedding, LangChain, LLM, MuleSoft, RAG

Abstract – The goal of this project is to design and implement a Retrieval-Augmented Generation (RAG) system aimed at MuleSoft developers. The system allows developers to ask questions in natural language about the technology and obtain accurate answers based on the official documentation, stored in a vector database. The work was approached in two phases. First, a traditional RAG was built that retrieved relevant fragments of the documentation and fed them to a language model to generate the answer. Then, an agentic RAG was developed, capable of reasoning about queries, deciding which tools to use (indexed documentation, web search, or Anypoint Exchange catalog) and iterating until producing a satisfactory answer. Finally, both systems were compared through an evaluation that measures the quality of the answers and the computational cost of each approach.

Keywords – AI Agent, Chunk, Embedding, LangChain, LLM, MuleSoft, RAG, Vectorial database

1 Introducción

1.1. Puesta en contexto

La integración y conexión entre aplicaciones, servicios, bases de datos y dispositivos se ha convertido en el pilar fundamental que permite a organizaciones ofrecer los servicios modernos y eficientes que utilizamos día a día. MuleSoft es una de las principales plataformas de integración, en la cual cada vez se está aumentando más la cantidad y complejidad de documentación técnica que ofrece. Los profesionales que utilizan MuleSoft deben consultar frecuentemente esta documentación para resolver dudas o informarse sobre distintos componentes. Sin embargo, la búsqueda en estos manuales puede ser compleja y engorrosa, ya que hay información desorganizada o dividida en distintos lugares.

Por otro lado, en los últimos años, los avances de la inteligencia artificial han hecho que grandes modelos de lenguaje (LLM) entrenen y nos proporcionen respuestas a nuestras preguntas. Sin embargo, si buscamos respuestas concretas en un contexto determinado habrá una alta probabilidad de que se produzcan alucinaciones y se inventen la respues-

• E-mail de contacto: dmontoyagarcia6@gmail.com
• Mención: Ingeniería de Computadores
• Trabajo tutorizado por: Ramon Grau Sala
• Curso 2025/26

ta. Es aquí donde nace RAG (Retrieval Augmented Generation). Esta arquitectura combina 2 enfoques, la recuperación de información en base a datos e información organizada que tenemos sobre un tema y la generación de texto a partir de la extracción de ese contexto.

Este trabajo consiste en el diseño e implementación de un sistema RAG especializado para profesionales de MuleSoft, que permita realizar consultas en lenguaje natural y obtener respuestas contextualizadas basadas en su documentación. El sistema integrará bases de datos vectoriales, técnicas de embeddings y modelos de lenguaje.

1.2. Objetivos

1.2.1. Objetivo general

Diseñar, implementar y evaluar un sistema RAG que facilite el acceso a conocimiento técnico especializado de MuleSoft mediante consultas en lenguaje natural, facilitando el acceso a información relevante de forma eficiente y precisa.

1.2.2. Objetivos específicos

- Analizar y entender los fundamentos teóricos de los sistemas RAG.
- Evaluar diferentes tecnologías para la construcción de sistemas RAG (técnicas de embeddings, bases de datos vectoriales y modelos de lenguaje).
- Diseñar la arquitectura RAG adecuada para el acceso eficiente a documentación técnica especializada.
- Construir una base de conocimiento a partir de documentación relevante de MuleSoft, definiendo metadatos que mejoren la recuperación semántica de la información.
- Generar representaciones vectoriales (embeddings) de la documentación para permitir su recuperación semántica.
- Implementar el sistema de recuperación de información mediante una base de datos vectorial.
- Integrar un modelo de lenguaje para la generación de respuestas.
- Evaluar el sistema desarrollado en términos de calidad de respuesta, rendimiento y limitaciones.

1.3. Metodología y planificación

El trabajo sigue una metodología de trabajo de cascada. Cada fase deberá completarse antes de pasar a la siguiente, de forma secuencial.

Estas son las diferentes fases en las que se ha dividido el proyecto:

1.3.1. Fase 1: Estudio del estado del arte

Estudio de los conceptos teóricos relacionados con los sistemas RAG. Se ha estudiado sobre los LLM, los métodos de generación de embeddings, las bases de datos vectoriales y las técnicas de recuperación de información.

1.3.2. Fase 2: Análisis y selección de tecnologías

Se han evaluado las diferentes tecnologías estudiadas en la fase anterior. Se expusieron las distintas alternativas y se justificó la mejor elección en cada caso.

1.3.3. Fase 3: Diseño de la arquitectura del sistema

Se ha diseñado la arquitectura del sistema RAG, especificando los diferentes componentes y cómo se conectarán entre ellos. Se ha diseñado el flujo completo del sistema, desde la consulta del usuario hasta la generación de la respuesta final.

1.3.4. Fase 4: Construcción de la base de conocimiento

Se ha construido la base de conocimiento que será utilizada por el sistema. Se ha recopilado documentación sobre MuleSoft y se ha procesado dividiendo en fragmentos pequeños (chunks) para facilitar el procesamiento y mejorando la precisión.

1.3.5. Fase 5: Implementación del sistema RAG

Se ha implementado el sistema que permite realizar consultas en lenguaje natural. La consulta será convertida en vectores y se buscará en la base de datos vectorial para recuperar la información relevante. Los fragmentos recuperados se utilizarán como contexto para la consulta final.

1.3.6. Fase 6: Evaluación del sistema

Se ha evaluado el rendimiento del sistema. Se han realizado diferentes pruebas para analizar la precisión y fiabilidad de las respuestas.

2 BASE TEÓRICA

2.1. Base teórica de MuleSoft

Tradicionalmente, las integraciones entre sistemas se han diseñado mediante conexiones point-to-point (P2P), donde cada sistema se conecta directamente a otro con código personalizado. Esto deriva en una arquitectura conocida como ”espagueti”, difícil de escalar, reutilizar y mantener.

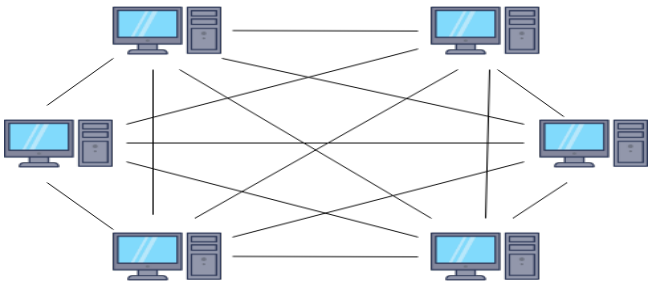

Fig. 1: Arquitectura ”espagueti” resultante de integraciones point-to-point.

MuleSoft, la compañía líder en software de integración, propone una alternativa basada en el principio ”API-Led Connectivity”, donde todas las integraciones se centralizan mediante APIs reutilizables organizadas en tres capas:

- Experience APIs: presentación de la información.
- Process APIs: procesamiento de los datos.
- System APIs: acceso a datos.

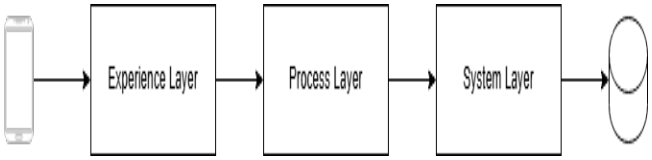

Fig. 2: Modelo API-Led Connectivity.

Su producto principal, Anypoint Platform, es una solución que implementa este principio en la nube. Se compone de cuatro elementos:

- **Design Center**: diseño y documentación de especificaciones API en lenguaje RAML.
- **Exchange**: repositorio de assets reutilizables (APIs, conectores, plantillas).
- **Anypoint Studio**: IDE con interfaz drag and drop para implementar la lógica de las APIs.
- **Management Center**: monitorización y gestión del despliegue y de las APIs publicadas.

En resumen, MuleSoft permite orquestar la comunicación entre múltiples sistemas desde un único punto, sustituyendo la arquitectura espagueti por una red ordenada y mantenible de microservicios.

2.2. Base teórica de las tecnologías

2.2.1. Modelos de lenguaje (LLM)

Conceptos generales

Los LLM son modelos de aprendizaje profundo basados en la arquitectura Transformer, pre-entrenados con grandes cantidades de datos. Representan las palabras como vectores multidimensionales en los que las palabras con significados similares quedan próximas entre sí en el espacio vectorial, lo que les permite comprender el contexto y generar respuestas coherentes en tareas como responder preguntas, resumir textos o traducir.

Sin embargo, presentan varias limitaciones. Pueden producir respuestas inexactas o inventadas (alucinaciones), su conocimiento es estático y queda limitado a la fecha de su último entrenamiento, requieren grandes recursos computacionales para entrenarse y ejecutarse, y su adaptación a dominios específicos es costosa.

Selección de modelo de lenguaje

La selección del modelo se ha basado en tres criterios del comparador ”Artificial Analysis”. El Agentic Index (capacidad para planificar tareas, usar herramientas y razonar en múltiples pasos), la velocidad de respuesta y el tamaño de la ventana de contexto. **GPT-5.4 (xhigh)** obtiene la mejor puntuación en Agentic Index, una de las mayores ventanas de contexto y una velocidad intermedia, lo que lo convierte en la opción más equilibrada para el sistema.

2.2.2. Embeddings

Conceptos generales

Un embedding es una representación vectorial de un dato de entrada, obtenida tras el entrenamiento de una red neuronal. En el espacio vectorial resultante, los elementos con significados similares quedan próximos entre sí.

En lenguaje natural, las palabras no se encuentran aisladas, sino que dependen del contexto. Los embeddings a nivel de palabra no son capaces de capturar esta información contextual. Para superar esta limitación, los LLM actuales utilizan redes Transformer. Al generar el embedding de una palabra o frase, analiza su relación con el resto de elementos del texto, obteniendo así una representación que sí captura el contexto.

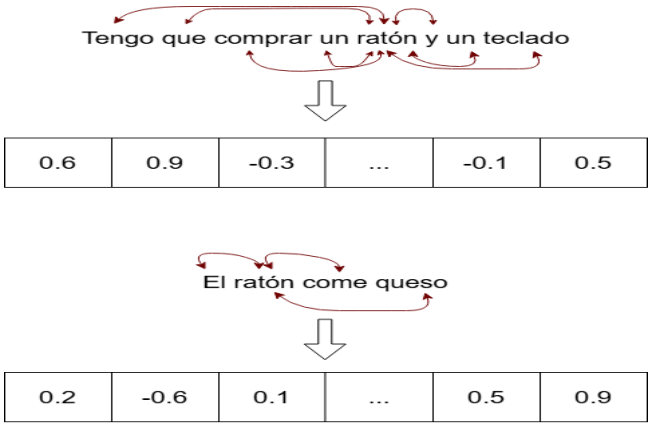

Fig. 3: Embedding contextual generado por una red Transformer.

Selección de modelo de embedding

Dado que la documentación técnica de MuleSoft contiene texto e imágenes, se requiere un modelo de embeddings multimodal. Consultando el ”MMEB leaderboard” y los rankings de ”Hugging Face”, se identifican como candidatos los modelos Qwen3-VL-Embedding-8B y Qwen3-VL-Embedding-2B. Se ha seleccionado el **Qwen3-VL-Embedding-2B**, que ofrece un equilibrio adecuado entre calidad y coste computacional, además de soporte multilingual.

2.2.3. Bases de datos vectoriales

Conceptos generales

Los LLM están entrenados con grandes cantidades de datos genéricos, pero no disponen de información específica de un dominio concreto. Para resolver esto, se utiliza un LLM como intermediario entre el usuario y una base de conocimiento. El modelo recibe la pregunta, consulta la información relevante y genera una respuesta en lenguaje natural.

Una base de datos vectorial es una base de datos que almacena la información en forma de embeddings. Tanto la documentación indexada como las consultas del usuario se representan como vectores en un mismo espacio vectorial.

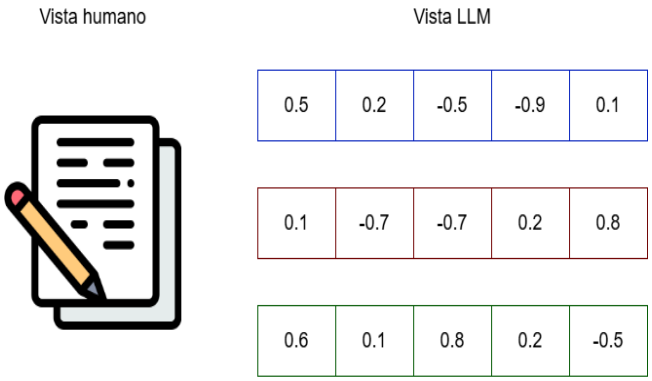

Fig. 4: Representación vectorial de la base de conocimiento y la consulta.

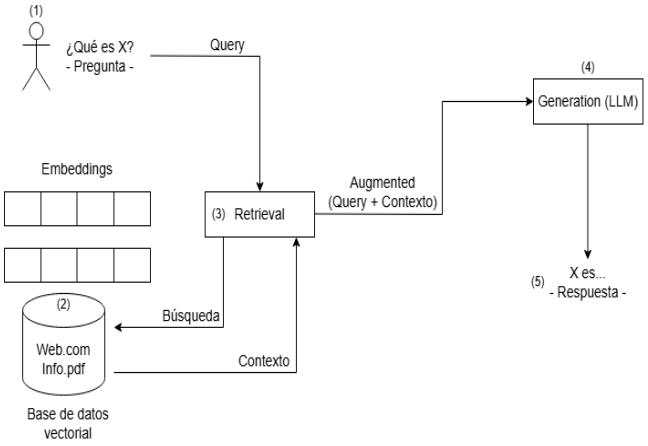

Fig. 5: Arquitectura de un sistema RAG.

La recuperación de información se realiza mediante la similitud del coseno, una operación matemática que compara la dirección de dos vectores. Cuanto más similares son los significados, más cercana a 1 es su similitud. Cuanto más distintos, más cercana a 0.

Selección de base de datos vectorial

Para seleccionar la base de datos vectorial se aplican varios criterios de filtrado. Primero, se requiere soporte explícito para embeddings de texto e imágenes, dado que la documentación de MuleSoft contiene ambos tipos de contenido. Segundo, se descartan las soluciones que no son open source. Tercero, consultando la web Vector DB Comparison, se descartan también las opciones con soporte parcial para embeddings de imágenes o sin integración con los frameworks LangChain y LlamaIndex. Tras aplicar estos filtros, las alternativas finales son Qdrant, Weaviate y Chroma.

Cada una presenta un perfil distinto. Weaviate destaca en búsquedas complejas, pero tiene una curva de aprendizaje pronunciada. Qdrant ofrece un filtrado potente por metadatos, pero está orientado a sistemas con grandes volúmenes de datos y actualizaciones frecuentes. Chroma es la más simple y rinde adecuadamente para volúmenes moderados (hasta unos 100.000 vectores), suficientes para el alcance de este proyecto.

Por su simplicidad y por ajustarse a las necesidades reales del sistema (documentación técnica, guías y documentos de buenas prácticas), se ha seleccionado **ChromaDB** como base de datos vectorial.

2.2.4. RAG

Conceptos generales

Los LLM no son capaces de responder con precisión a preguntas sobre temas muy específicos o muy recientes para los que no han sido entrenados, lo que puede provocar alucinaciones. Los sistemas RAG resuelven este problema dando al modelo acceso a una base de datos externa con información relevante.

El flujo es el siguiente. El usuario formula una pregunta, que junto con los documentos previamente fragmentados (chunks) y almacenados como embeddings en una base de datos vectorial, pasan a un sistema de recuperación (retrieval). Este sistema, mediante similitud vectorial, identifica los fragmentos más relevantes para la consulta. Finalmente, esos fragmentos se entregan al LLM junto con la pregunta original, lo que le permite generar una respuesta precisa apoyada en información concreta en lugar de en su conocimiento general.

Selección de tipo de RAG

Existen distintas variantes de RAG. El RAG clásico trata la consulta como una única petición estática y no verifica la calidad de los fragmentos recuperados. El Iterative RAG sí permite reiterar la búsqueda hasta obtener resultados suficientes, y el Tree RAG descompone consultas complejas en subconsultas más simples. Sin embargo, ninguno de los tres resuelve la principal limitación del enfoque tradicional, la incapacidad de acceder a fuentes externas de información cuando la base de datos vectorial no cubre el dominio de la consulta.

El RAG agéntico aborda este problema. En este enfoque, un agente interpreta la intención del usuario, descompone o reformula la consulta si es necesario y decide qué herramientas invocar para obtener la información: búsqueda vectorial en la base de datos, búsqueda web o acceso a APIs externas. Posteriormente, un LLM genera la respuesta final a partir de toda la información recopilada.

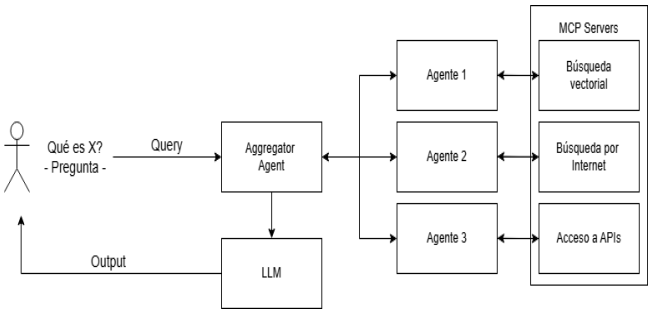

Fig. 6: Arquitectura de un sistema RAG agéntico.

Sus tres capacidades clave son la memoria a corto y lar-

go plazo (que permite planificar tareas complejas), el enrutamiento de consultas con toma de decisiones, y la invocación dinámica de herramientas mediante APIs. Por todas estas razones, y especialmente por la capacidad de combinar múltiples fuentes de información, se ha seleccionado el **RAG agéntico** como tipo de sistema para este proyecto.

3 Diseño de la arquitectura del sistema

El sistema se basa en una arquitectura **RAG agéntica** compuesta por los siguientes componentes:

- Un modelo de lenguaje (**GPT-5.4 xhigh**) que actúa como agente orquestador, interpretando la consulta y generando la respuesta final.
- Un modelo de embeddings (**Qwen3-VL-Embedding-2B**) que transforma tanto la consulta como la documentación en vectores.
- Una base de datos vectorial (**ChromaDB**) para almacenar y recuperar los fragmentos mediante similitud.
- Un conjunto de *herramientas* (búsqueda vectorial, búsqueda web, APIs externas).

El usuario formula una consulta en lenguaje natural, el agente analiza la intención y decide qué herramientas invocar. Con la información recuperada como contexto, el LLM genera la respuesta final.

Para mejorar la precisión del retrieval, cada fragmento almacenado en la base de datos vectorial incluye metadatos adicionales: identificador, nombre del documento, fecha, tipo de contenido (texto, imagen, código), tecnología asociada (MuleSoft, DataWeave, etc.) y fuente. Estos metadatos permiten filtrar y priorizar los resultados durante la recuperación.

4 Construcción de la base de conocimiento

4.1. Recopilación de información

La documentación oficial de MuleSoft contiene cientos de páginas, por lo que una indexación exhaustiva no es viable. Para descubrir las URLs candidatas se desarrolló un crawler en Python basado en una búsqueda BFS desde la URL raíz, que devolvió aproximadamente 7.000 URLs únicas.

Tras filtrar contenido no relevante (notas de versión, traducciones a otros idiomas y versiones obsoletas), el conjunto se redujo a unas 3.000 URLs. Dado el coste computacional del proceso de indexación (aproximadamente 15 minutos por URL), para la fase de validación se seleccionaron 50 URLs según un criterio de diversidad temática, cubriendo áreas como diseño de APIs, DataWeave, despliegue, gestión de errores y conectores.

4.2. Limpieza y preprocesamiento

Cada URL se descarga y parsea, eliminando elementos <script> y <style>. Posteriormente se identifica la estructura jerárquica del documento mediante los `<div class="sect1">`, donde cada uno corresponde a una sección y se trata como candidato a chunk.

Durante el desarrollo se detectaron varios casos especiales que requirieron un tratamiento específico:

- **Secciones de navegación** (*What You Learned*, *What's Next*, *See Also*, *Developer Deep Dive*): se descartan porque compiten injustamente con contenido útil al tener alta similitud léxica pero poco valor informativo.
- **Bloques de pestañas (tabs)**: se generan sub-chunks independientes por cada opción para evitar mezclar contenidos de alternativas distintas.
- **Bloques de código**: se procesan como chunks independientes, ya que mezclar código con texto produce embeddings de baja calidad y un bloque de código puede responder por sí solo a una consulta.
- **Imágenes**: en lugar de procesarlas con el codificador visual del modelo, se construye un texto enriquecido con el atributo *alt*, una clasificación automática y la sección donde aparecen. Se vectoriza ese texto en lugar de la imagen.

A continuación, los chunks se dividen en fragmentos más pequeños mediante el componente *RecursiveCharacterTextSplitter* de LangChain, que divide recursivamente por párrafos, frases, palabras o caracteres. Se configuran dos divisores, texto en prosa (1.000 caracteres, 15 % de solapamiento) y bloques de código (2.000 caracteres, 10 % de solapamiento). Cada chunk se identifica con un hash único calculado a partir de la URL, título, tipo, índice y contenido.

4.3. Generación de embeddings

Cada chunk se convierte en una representación vectorial mediante el modelo Qwen3-VL-Embedding-2B. El proceso consiste en tokenizar el texto, obtener la representación interna del modelo para cada token, calcular su media para obtener un único vector y normalizarlo a magnitud 1. El resultado es un vector de 2.048 dimensiones.

Una limitación encontrada durante el desarrollo es que la capacidad multimodal del modelo no se aprovecha finalmente, ya que los embeddings visuales y textuales producidos por Qwen3-VL ocupan regiones lejanas del espacio vectorial, lo que impide recuperar imágenes mediante consultas textuales.

4.4. Almacenamiento en base de datos

Los embeddings se almacenan en ChromaDB mediante una operación upsert, que combina insert y update. Cada registro contiene el identificador único, el texto original del chunk, su vector, los chunks cercanos y los metadatos asociados (tipo de contenido, subtipo, URL, título de la sección e índice). Estos metadatos resultan claves durante la fase de retrieval, ya que permiten realizar consultas filtradas e implementar mecanismos como la recuperación de chunks hermanos.

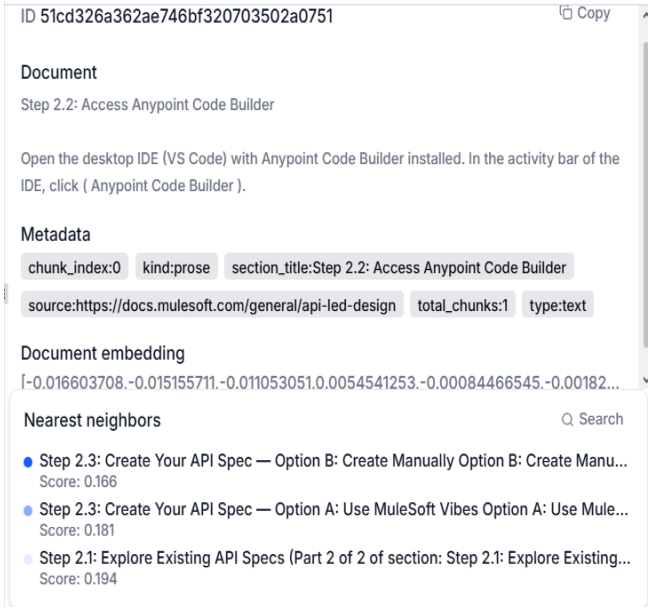

Fig. 7: Estructura de un registro en la base de datos vectorial.

El proceso completo (extracción, fragmentación, embeddings e inserción) tarda unos 15 minutos por URL. Para el corpus experimental de 50 URLs, esto supone aproximadamente 12 horas de procesamiento, lo que hace inviable indexar el conjunto completo de 3.000 URLs dentro de los tiempos del TFG.

5 Implementación del sistema RAG

A diferencia del RAG tradicional, donde el flujo es lineal (pregunta → recuperación → generación), el sistema agéntico introduce capacidad de razonamiento autónomo. El modelo de lenguaje decide qué herramientas utilizar, en qué orden, e itera hasta considerar que tiene información suficiente para responder.

5.1. Modelo de consulta y arquitectura del agente

El agente se implementa con LangChain y su extensión LangGraph, siguiendo el patrón ReAct. El modelo (GPT-5.4) razona sobre la pregunta, elige una herramienta, lee el resultado y repite el ciclo hasta construir una respuesta. Su comportamiento se define mediante un prompt que establece las reglas principales. Debe fundamentar las respuestas en evidencia recuperada, responder "I don't know" si no hay información disponible, citar las fuentes mediante notación `[Source N]` con URL exacta, priorizar la búsqueda en la documentación indexada, y rechazar las preguntas ajenas al dominio sin invocar herramientas.

5.2. Herramientas del agente

El agente dispone de cuatro herramientas que puede invocar dinámicamente:

- `search_documentation`: búsqueda semántica en ChromaDB sobre la documentación indexada, con expansión a chunks hermanos. Es la herramienta principal.
- `search_web`: búsqueda web a través de Tavily Search, reservada para casos no cubiertos por la documentación indexada.
- `search_anypoint_exchange`: consulta al catálogo público de Anypoint Exchange para localizar conectores, API specs o templates.
- `get_anypoint_asset_detail`: obtiene la información técnica completa de un asset específico.

LangChain facilita esta integración. Cada herramienta incluye una descripción que el modelo lee internamente para entender cuándo conviene utilizarla.

5.3. Mecanismo de autoevaluación

Para garantizar la calidad de las respuestas se incorpora un segundo modelo de lenguaje que actúa como evaluador. Tras cada respuesta del agente, este recibe la pregunta y la respuesta candidata, y emite un veredicto en formato JSON: **OK** (respuesta aceptable) o **INSUFFICIENT** (respuesta con carencias). En este último caso, el sistema reintenta la consulta incorporando el motivo del rechazo como contexto adicional, hasta un máximo de 3 iteraciones.

5.4. Arquitectura final

A diferencia de algunas arquitecturas que distribuyen el trabajo entre múltiples agentes especializados, la solución implementada utiliza un único agente con acceso directo a las cuatro herramientas. Esta decisión se justifica porque un agente con visión global puede combinar libremente las fuentes sin necesidad de un orquestador intermedio, lo que reduce la latencia y el coste por consulta. El evaluador es el único componente adicional, y su función es complementaria al agente.

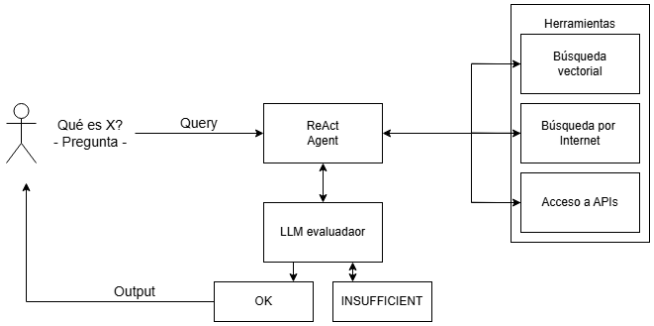

Fig. 8: Arquitectura final del sistema RAG agéntico.

El sistema se expone como una única función que recibe una pregunta y devuelve la respuesta junto con datos adicionales (número de iteraciones, herramientas utilizadas, veredicto del evaluador y detalle de cada iteración), información necesaria para analizar posteriormente el comportamiento del sistema durante la fase de evaluación.

6 Evaluación del sistema y resultados

Una vez implementados los dos sistemas, esta fase los somete a una evaluación comparativa para medir su rendi-

miento e identificar en qué situaciones cada enfoque resulta más adecuado.

6.1. Demostración del funcionamiento

Se ha desarrollado una interfaz web con Streamlit que permite interactuar con el sistema en lenguaje natural. Además de la respuesta, la interfaz muestra las fuentes citadas y un panel desplegable con los detalles internos del proceso (número de iteraciones, herramientas invocadas y veredicto del evaluador), lo que convierte el funcionamiento del agente en información inspeccionable.

Se puede ver su funcionamiento en el apéndice.

6.2. Comparación con un LLM general

Para mostrar la superioridad del sistema agéntico frente a un LLM general, se planteó a ambos sistemas la siguiente pregunta:

> *In the Anypoint Connector for Database in Mule 4, what are all the supported connection types I can configure?*

ChatGPT respondió con omisiones (olvidó uno de los seis tipos de conexión) e invenciones (atribuyó una lista detallada de motores a la *Generic Connection* que no aparece en la documentación oficial). El RAG agéntico, en cambio, identificó correctamente los seis tipos (Derby, SQL Server, MySQL, Oracle, Generic y Data Source Reference), los describió con sus parámetros y los acompañó de tres URLs verificables.

Este caso ilustra el problema central que justifica el enfoque RAG. Un LLM general puede generar respuestas que combinan fragmentos correctos con omisiones e invenciones, sin que el usuario pueda distinguirlos. El RAG agéntico evita este fallo respaldando cada afirmación con una fuente verificable.

Esta comparación de puede ver en el apéndice.

6.3. Metodología de evaluación

Se han diseñado dos conjuntos de preguntas adaptados a las capacidades de cada sistema. El conjunto base, para el RAG tradicional, contiene 29 preguntas en seis categorías (literales, conceptuales, parafraseadas, multi-chunk, de detalle y fuera de contexto). El conjunto extendido, para el RAG agéntico, añade 14 preguntas adicionales que evalúan capacidades exclusivas del agente: búsqueda web, consulta a Anypoint Exchange, multi-salto, premisa falsa y fuera de dominio. La construcción inicial del *eval set* contó con la ayuda de un modelo generativo, aunque las decisiones de diseño y la validación final son propias.

Cada sistema requiere métricas distintas. El RAG tradicional se evalúa con métricas de retrieval (Recall@K, MRR) y de calidad de la respuesta (keyword recall). El RAG agéntico se evalúa con el *verdict OK* de un LLM evaluador externo, complementado con métricas específicas de su comportamiento (tool routing, aclaración de premisa falsa, rechazo de dominio, eficiencia). El keyword recall se mantiene como única métrica directamente comparable entre ambos sistemas.

6.4. Resultados del RAG tradicional

El sistema se evaluó sobre 29 preguntas y tardó 9 minutos. La tabla 1 resume los resultados sobre las 25 preguntas in-scope.

Métrica	Valor
Recall@1	56,0 %
Recall@3	72,0 %
Recall@5	88,0 %
MRR	0,671
Keyword recall medio	84,0 %

TABLA 1: RESULTADOS GLOBALES DEL RAG TRADICIONAL.

El sistema localiza el fragmento correcto entre los cinco primeros resultados en el 88 % de los casos, aunque no siempre lo prioriza correctamente (Recall@1 = 56 %). El *keyword recall* del 84 % muestra que la mayoría de respuestas cubren los términos esperados. Por tipos, el punto fuerte son las preguntas conceptuales (100 %) y el más débil las parafraseadas (Recall@1 del 20 %, keyword recall del 70 %), ya que el cambio de vocabulario aleja el embedding de la consulta del fragmento correcto. Las 4 preguntas fuera de contexto se resolvieron correctamente con una abstención ("I don't know").

Tres preguntas concentran los fallos del sistema, cada una con un patrón distinto: **lit_01** (versiones de OAS) recupera la URL correcta pero no la sección concreta, y el LLM se abstiene; **par_05** (compartir documentación en Exchange) no recupera ni la URL ni la sección correctas, y construye una respuesta plausible pero incompleta; **det_02** (fecha de obligatoriedad de MFA) recupera la URL correcta pero una sección equivocada del mismo documento, devolviendo una fecha incorrecta. Estos tres casos ilustran las limitaciones que el RAG agéntico aborda mediante la reformulación interna de consultas.

6.5. Resultados del RAG agéntico

El sistema se evaluó sobre 39 preguntas y tardó 24 minutos. La tabla 2 resume los resultados globales.

Métrica	Valor
Verdict OK	100,0 % (39/39)
Keyword recall medio	95,8 %
Tool routing correcto	97,3 % (36/37)
Aclaración de premisa falsa	100,0 % (2/2)
Rechazo de dominio	100,0 % (3/3)

TABLA 2: RESULTADOS GLOBALES DEL RAG AGÉNTICO.

El evaluador externo aprobó el 100 % de las respuestas y la cobertura de palabras clave alcanzó el 95,8 %. Las categorías que el RAG tradicional manejaba bien (conceptual, literal, multi-chunk) se mantienen al 100 %. Las preguntas parafraseadas mejoran (80 % frente al 70 % del tradicional) y las de detalle pasan del 75 % al 100 % gracias a la combinación de varias búsquedas. Las categorías exclusivas del

agente (búsqueda web, Exchange, multi-salto, premisa falsa, fuera de dominio) se resolvieron todas correctamente.

En cuanto a la eficiencia, el 95 % de las preguntas se resuelven en una sola iteración, con una media de 1,69 llamadas a herramientas. Las multi-salto son las más costosas (hasta 12 llamadas en algún caso), mientras que las conceptuales o literales se resuelven con una sola llamada. La distribución por herramientas muestra el predominio de `search_documentation` (66,7 %), seguida de `search_web` (15,2 %), `search_anypoint_exchange` (12,1 %) y `get_anypoint_asset_detail` (6,1 %).

6.6. Comparativa entre sistemas

La única métrica directamente comparable es el *keyword recall* sobre las 25 preguntas in-scope compartidas.

Métrica	Tradicional	Agéntico
Keyword recall	84,0 %	95,8 %
Preguntas resueltas con éxito	22/25	25/25
Tiempo por pregunta	18,6 s	37,4 s

TABLA 3: COMPARATIVA CUANTITATIVA ENTRE AMBOS SISTEMAS.

El RAG agéntico mejora la cobertura de palabras clave en 12 puntos porcentuales y resuelve las tres preguntas que el tradicional no podía. La mejora más marcada está en las preguntas de detalle (75 % $\rightarrow$ 100 %) y las literales (83,3 % $\rightarrow$ 100 %). El agente compensa la rigidez del retrieval semántico mediante tres mecanismos: reformulación interna de consultas (caso lit_01, donde expande "OAS" a "OpenAPI Specification"), múltiples búsquedas con queries progresivamente más específicas (caso par_05), y razonamiento sobre los resultados para distinguir secciones próximas (caso det_02).

Adicionalmente, el agente atiende 14 preguntas exclusivas (búsqueda web, Exchange, multi-salto, premisa falsa, fuera de dominio) con un acierto del 100 %, ampliando el alcance del sistema a casos que el tradicional no puede manejar.

La contrapartida es un coste computacional aproximadamente 2 veces superior por pregunta, justificado cuando el sistema debe operar sobre consultas en lenguaje natural, requerir información externa o comportarse de forma robusta ante preguntas ambiguas o fuera de contexto.

6.7. Limitaciones del estudio

La evaluación presenta limitaciones. El tamaño del *eval set* (29 y 39 preguntas) es suficiente para detectar tendencias pero limita el alcance estadístico. El keyword recall basado en matching literal produce tanto falsos negativos (ante sinónimos) como falsos positivos, aunque el verdict del evaluador externo compensa parcialmente esta limitación. Los LLM no son deterministas, por lo que los resultados son una instantánea de una ejecución concreta. El evaluador puede aprobar respuestas plausibles pero incorrectas. Finalmente, el corpus indexado (50 URLs) es una fracción reducida de la documentación completa de MuleSoft.

6.8. Conclusiones de la evaluación

La evaluación permite extraer cuatro conclusiones. Primero, el RAG agéntico mejora la cobertura de palabras clave en 12 puntos sobre el tradicional (95,8 % frente a 84,0 %) y resuelve correctamente las tres preguntas en las que el tradicional fallaba. Segundo, las métricas tradicionales de retrieval (Recall@K, MRR) no son directamente aplicables al sistema agéntico, lo que justifica el uso de métricas comunes (keyword recall). Tercero, el agente extiende el alcance del sistema a categorías que el tradicional no puede atender por diseño, con un acierto del 100 % en las 14 preguntas exclusivas. Cuarto, esta mejora tiene un coste computacional cuantificable (2 veces más lento por pregunta), justificable en escenarios donde la calidad y la robustez priman sobre la latencia.

En conjunto, los resultados confirman empíricamente la hipótesis principal del trabajo: un sistema RAG agéntico no solo iguala al tradicional, sino que añade capacidades distintas.

7 CONCLUSIONES

7.1. Conclusiones generales

El objetivo del proyecto consistía en diseñar, implementar y evaluar un sistema RAG orientado a desarrolladores de MuleSoft. Este objetivo se ha cumplido satisfactoriamente, construyendo dos sistemas funcionales (RAG tradicional y RAG agéntico) y realizando una evaluación comparativa entre ambos.

Los resultados empíricos confirman la hipótesis principal del trabajo. El RAG agéntico mejora la cobertura de palabras clave sobre el tradicional, resuelve correctamente las preguntas en las que este último fallaba y extiende el alcance del sistema a categorías que el enfoque clásico no puede atender por diseño (búsqueda web, Anypoint Exchange, multi-salto, premisas falsas y rechazo de dominio), con un acierto del 100 %.

Además, la comparación con un LLM general evidenció que el sistema desarrollado no solo iguala al modelo general, sino que aporta una ventaja clave, la trazabilidad. Cada afirmación está respaldada por URLs verificables de la documentación oficial.

La principal contrapartida del enfoque agéntico es su coste computacional, aproximadamente 2 veces superior al tradicional por pregunta. Este coste es asumible en escenarios donde la calidad de la respuesta está por encima de la latencia.

7.2. Líneas de mejora futuras

A lo largo del desarrollo se han identificado varias líneas de mejora que podrían abordarse en trabajos posteriores:

- **Ampliación del corpus indexado** al conjunto completo de la documentación de MuleSoft (unas 3.000 URLs).
- **Vectorización multimodal nativa** de las imágenes, descartada en este trabajo por la falta de alineación entre los espacios visual y textual del modelo.

- **Eval set ampliado** con varios cientos de preguntas para obtener estadísticas más precisas.
- **Tratamiento estructurado de las tablas HTML** mediante chunks por fila o columna con metadatos enriquecidos.
- **Despliegue en producción**, teniendo en cuenta aspectos como escalabilidad, gestión de credenciales y actualización automática del corpus.

USO DE LA IA GENERATIVA

Se utilizaron herramientas de inteligencia artificial, concretamente ChatGPT, como apoyo para la corrección gramatical y reformulación de textos. También se utilizó Claude para la creación del conjunto de preguntas de evaluación (*eval set*), proporcionándole como contexto la documentación indexada y las especificaciones de cada categoría de pregunta. El contenido generado fue revisado y validado.

REFERENCIAS

[1] P. Fernández, "MuleSoft: 'Vivimos la transformación más importante que ha sufrido la humanidad en su historia'," *Silicon*, 23 de abril de 2025. [En línea]. Disponible en: https://www.silicon.es/mulesoft-vivimos-la-transformacion-mas-importante-que-ha-sufrido-la-humanidad-en-su-historia-2566902

[2] Dirección General del Dato, "RAG - Retrieval Augmented Generation: La llave que abre la puerta de la precisión a los modelos del lenguaje," 24 de enero de 2024. [En línea]. Disponible en: https://datos.gob.es/es/blog/rag-retrieval-augmented-generation-la-llave-que-abre-la-puerta-de-la-precision-los-modelos-del

[3] IBM, "¿Qué es una plataforma de integración? ¿Necesito una?" [En línea]. Disponible en: https://www.ibm.com/es-es/think/insights/integration-platform

[4] Seidor, "¿Qué es MuleSoft? ¿Componentes de Anypoint Platform?," 13 de febrero de 2023. [En línea]. Disponible en: https://www.seidor.com/es-es/blog/salesforce-mulesoft-que-es

[5] Amazon Web Services, "¿Qué es un LLM (modelo de lenguaje de gran tamaño)?" [En línea]. Disponible en: https://aws.amazon.com/es/what-is/large-language-model/

[6] D. Soler, "Limitaciones y desafíos de los LLM." [En línea]. Disponible en: https://keepcoding.io/blog/limitaciones-y-desafios-de-los-llm/

[7] A. Kargwal, "Los 10 mejores modelos de lenguaje grande (LLM) de 2026," *Botpress*, 10 de enero de 2026. [En línea]. Disponible en: https://botpress.com/es/blog/best-large-language-models

[8] Artificial Analysis, "Comparison of Models: Intelligence, Performance and Price Analysis." [En línea]. Disponible en: https://artificialanalysis.ai/models

[9] Codificando Bits, *¿Qué son los EMBEDDINGS? Grandes Modelos de Lenguaje* (28 de agosto de 2023). [Archivo de vídeo]. Consultado el 12 de junio de 2026. [En línea]. Disponible en: https://youtu.be/h4GNDHC-s50?si=FHLBxFp0KNve0fdu

[10] P. Thangavel, "Embeddings," 16 de febrero de 2026. [En línea]. Disponible en: https://prasanth.io/notes/Embeddings

[11] Hugging Face, "Models." [En línea]. Disponible en: https://huggingface.co/models

[12] Hugging Face, "MMEB Leaderboard (VLM2Vec)." [En línea]. Disponible en: https://huggingface.co/spaces/TIGER-Lab/MMEB-Leaderboard

[13] Sam Witteveen, *Qwen3 Multimodal Embeddings: Finally, RAG That Sees* (15 de enero de 2026). [En línea]. Disponible en: https://youtu.be/fY3-YeveBgA?si=m0G8XWhOXXz3ar1i

[14] Codificando Bits, *¿Qué son las BASES DE DATOS VECTORIALES? Grandes Modelos de Lenguaje* (21 de octubre de 2024). [En línea]. Disponible en: https://youtu.be/MCqViT-yHpY?si=9SU6D8ts534HTTMP

[15] R. Fuentes, "Tabla comparativa de bases de datos vectoriales," *LinkedIn*, noviembre de 2025. [En línea]. Disponible en: https://www.linkedin.com/posts/rfmit_te-comparto-esta-tabla-comparativa-de-bbdd-activity-7393338962678820864-Y-dZ/

[16] M. Ali, "Las 7 mejores bases de datos vectoriales en 2026," *DataCamp*, 11 de diciembre de 2025. [En línea]. Disponible en: https://www.datacamp.com/es/blog/the-top-5-vector-databases

[17] Superlinked, "Vector DB Comparison." [En línea]. Disponible en: https://superlinked.com/vector-db-comparison

[18] Ziru, "Vector Databases para RAG: Qdrant vs Pinecone vs Weaviate vs ChromaDB (Guía Completa 2025)," *El Diario IA*, 16 de diciembre de 2025. [En línea]. Disponible en: https://www.eldiarioia.es/2025/12/16/vector-databases-rag/

[19] Codificando Bits, *RAG ¡EXPLICADO! Grandes Modelos de Lenguaje* (7 de noviembre de 2024). [En línea]. Disponible en: https://youtu.be/esQ4LMVdbaA?si=evPZDyuKZBelWxgi

[20] R. Herrera, "Tipos de RAG," *LinkedIn*, 28 de enero de 2025. [En línea]. Disponible en: https://www.linkedin.com/pulse/tipos-de-rag-rogger-herrera-canonico-faw3f/

[21] I. Belcic y C. Stryker, "¿Qué es la RAG agentiva?," *IBM*. [En línea]. Disponible en: https://www.ibm.com/es-es/think/topics/agentic-rag

[22] M. Basantini, "Agentic RAG vs Corrective RAG: The Battle for Smarter AI Retrieval Systems," *Medium*, 21 de junio de 2025. [En línea]. Disponible en: https://medium.com/@mohitbasantani1987/agentic-rag-vs-corrective-rag-the-battle-for-smarter-ai-retrieval-systems-7d9076309fb0

[23] OpenAI, "ChatGPT." [En línea]. Disponible en: https://chat.openai.com

[24] Anthropic, "Claude." [En línea]. Disponible en: https://claude.ai/new

[25] Chroma, "ChromaDB." [En línea]. Disponible en: https://www.trychroma.com/home

[26] Project Jupyter, "Jupyter Notebook." [En línea]. Disponible en: https://jupyter.org

[27] Hugging Face, "Qwen/Qwen3-VL-Embedding-2B." [En línea]. Disponible en: https://huggingface.co/Qwen/Qwen3-VL-Embedding-2B

[28] LangChain, "RecursiveCharacterTextSplitter." [En línea]. Disponible en: https://reference.langchain.com/python/langchain-text-splitters/character/RecursiveCharacterTextSplitter

[29] LangChain, "LangGraph." [En línea]. Disponible en: https://www.langchain.com/langgraph

[30] MuleSoft, "MuleSoft Documentation." [En línea]. Disponible en: https://docs.mulesoft.com/general/

[31] Tavily, "Tavily Search API." [En línea]. Disponible en: https://www.tavily.com

[32] MuleSoft, "Anypoint Exchange." [En línea]. Disponible en: https://anypoint.mulesoft.com/exchange

APÉNDICE

A continuación se presentan capturas representativas del sistema RAG agéntico en funcionamiento sobre la interfaz desarrollada con Streamlit. Estas capturas ilustran los distintos comportamientos del sistema descritos en la sección de demostración y comparación.

Consulta respondida desde la documentación indexada

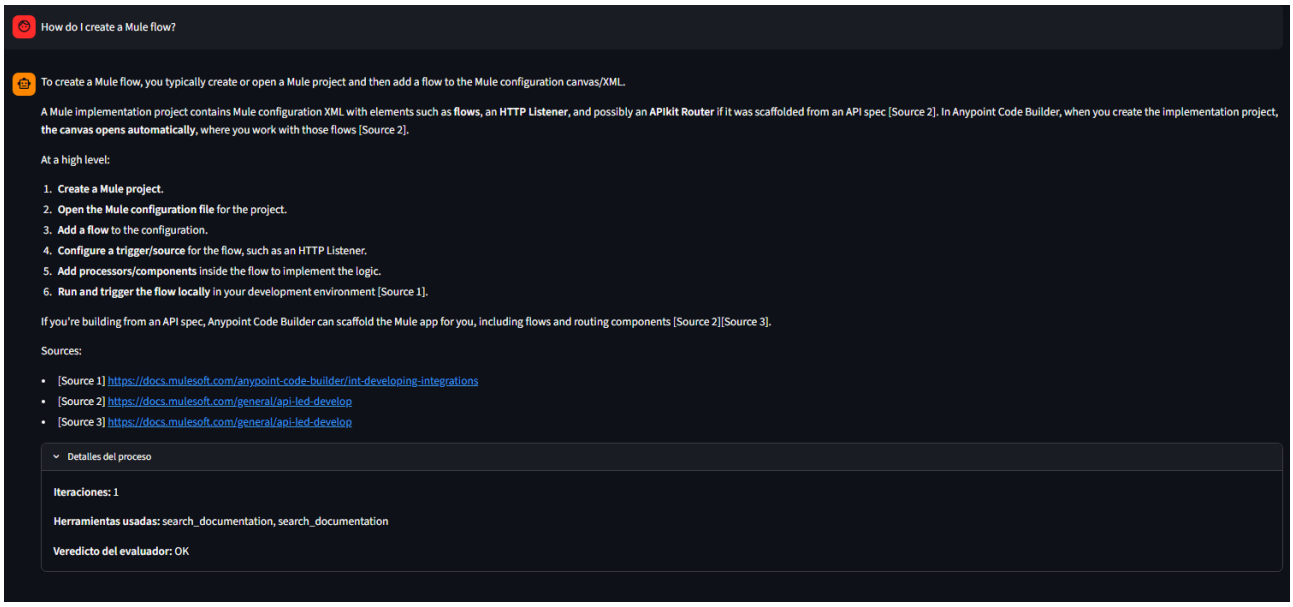

Fig. 9: Ejemplo de consulta respondida directamente a partir de la documentación indexada en la base de datos vectorial.

Consulta resuelta mediante una fuente externa

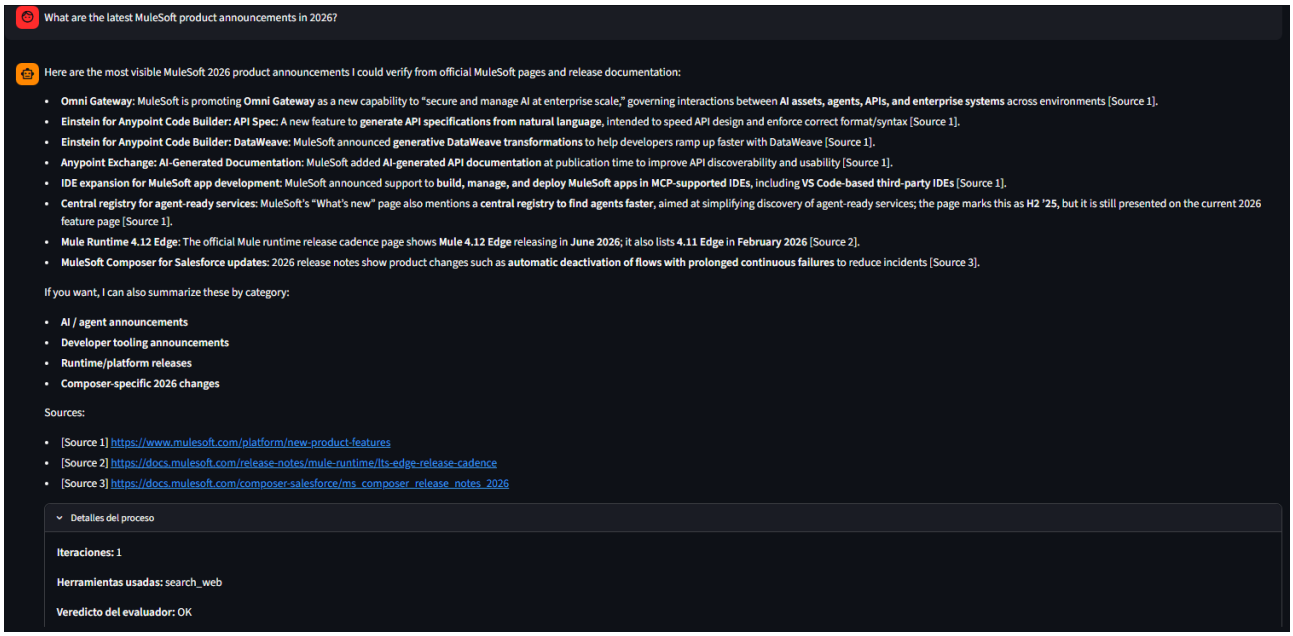

Fig. 10: Ejemplo de consulta resuelta mediante una fuente externa (búsqueda web).

Consulta multi-salto

Las siguientes figuras muestran una consulta multi-salto en la que el agente combina varias herramientas para construir la respuesta final.

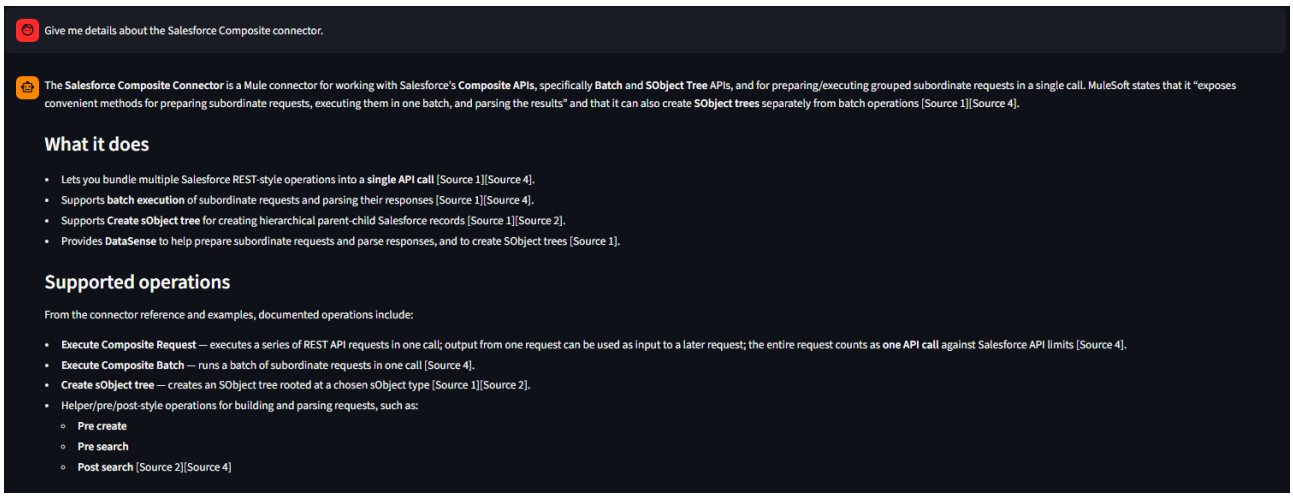

Fig. 11: Consulta multi-salto que combina varias herramientas (1/4).

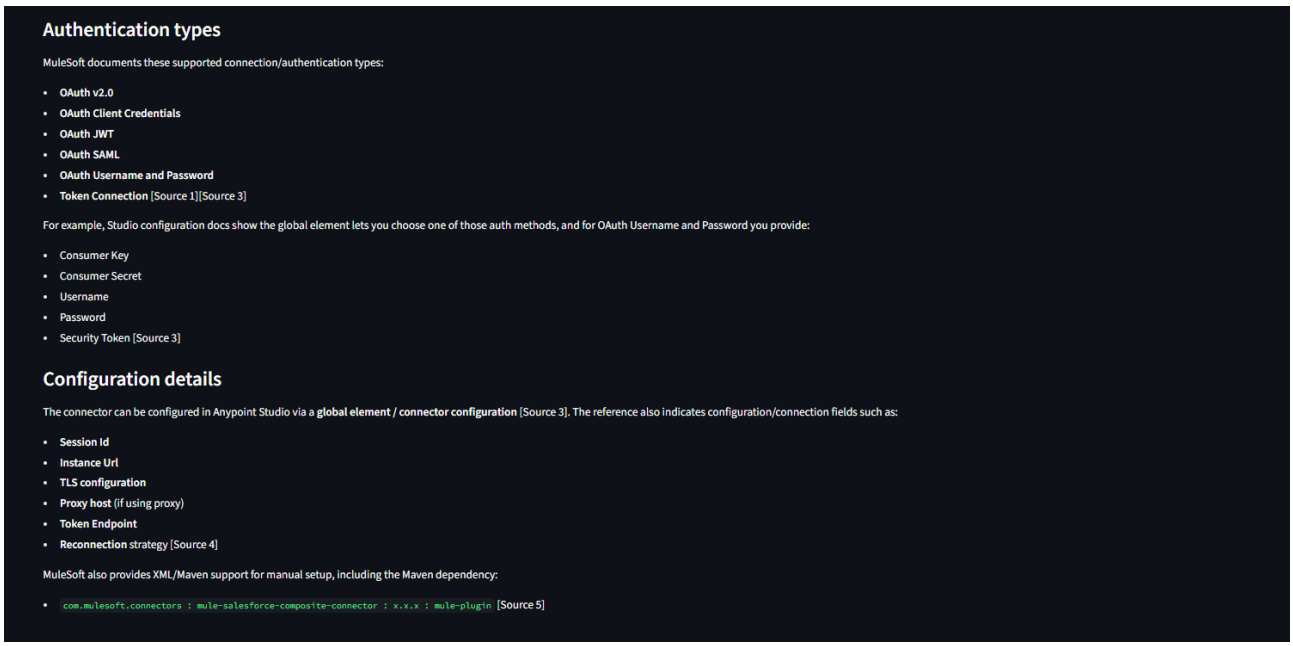

Fig. 12: Consulta multi-salto que combina varias herramientas (2/4).

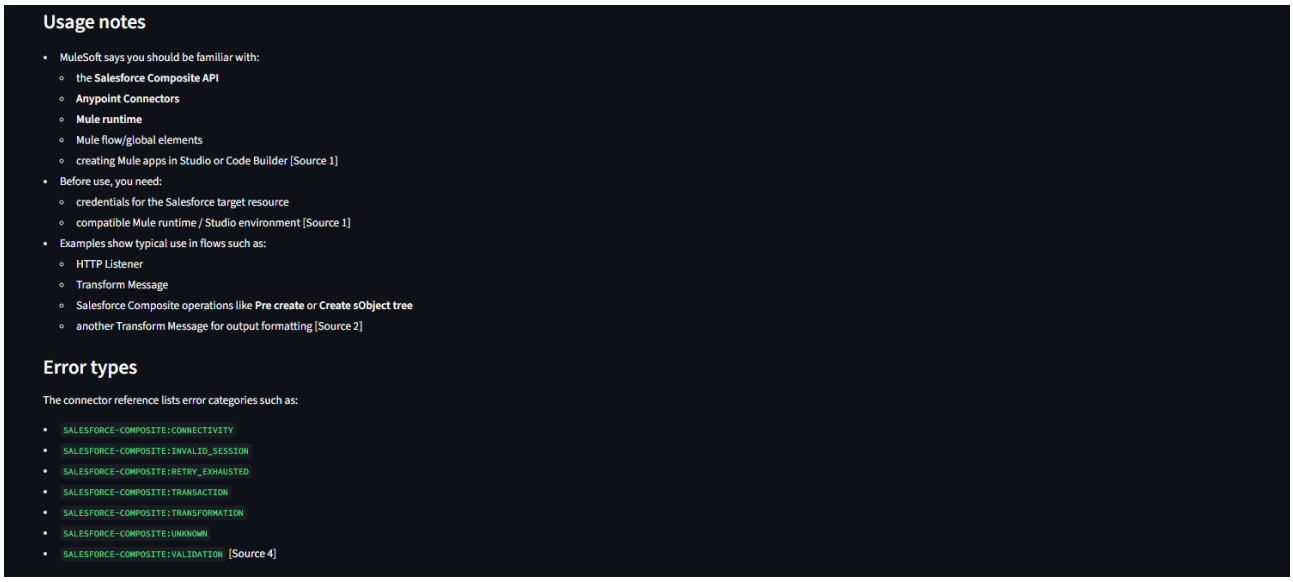

Fig. 13: Consulta multi-salto que combina varias herramientas (3/4).

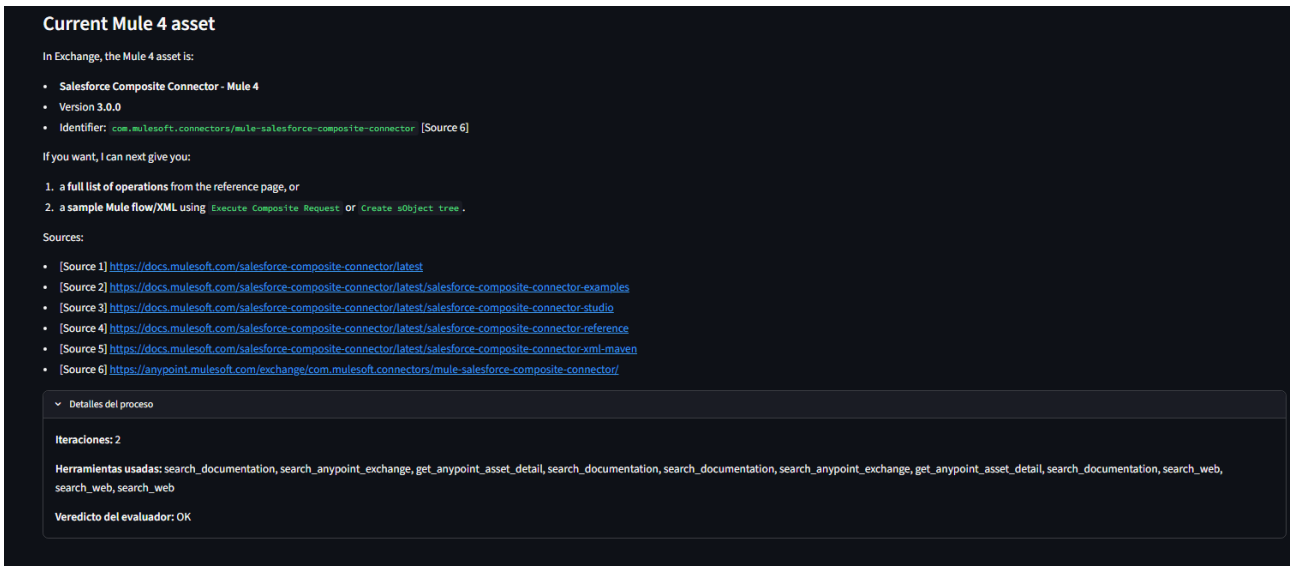

Fig. 14: Consulta multi-salto que combina varias herramientas (4/4).

Consulta fuera de dominio

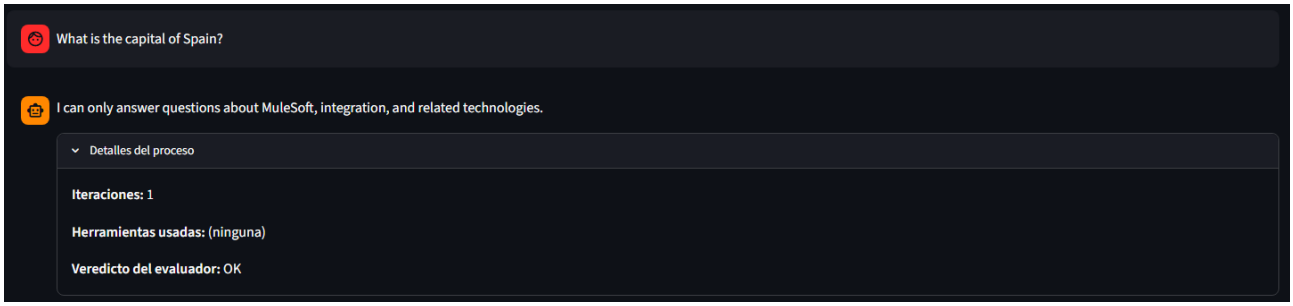

Fig. 15: Ejemplo de consulta fuera de dominio rechazada por el sistema sin invocar herramientas.

Comparación con un LLM general (Database Connector)

Las siguientes figuras muestran la comparación entre la respuesta de ChatGPT y la del RAG agéntico ante una misma pregunta técnica sobre los tipos de conexión soportados por el Database Connector de MuleSoft.

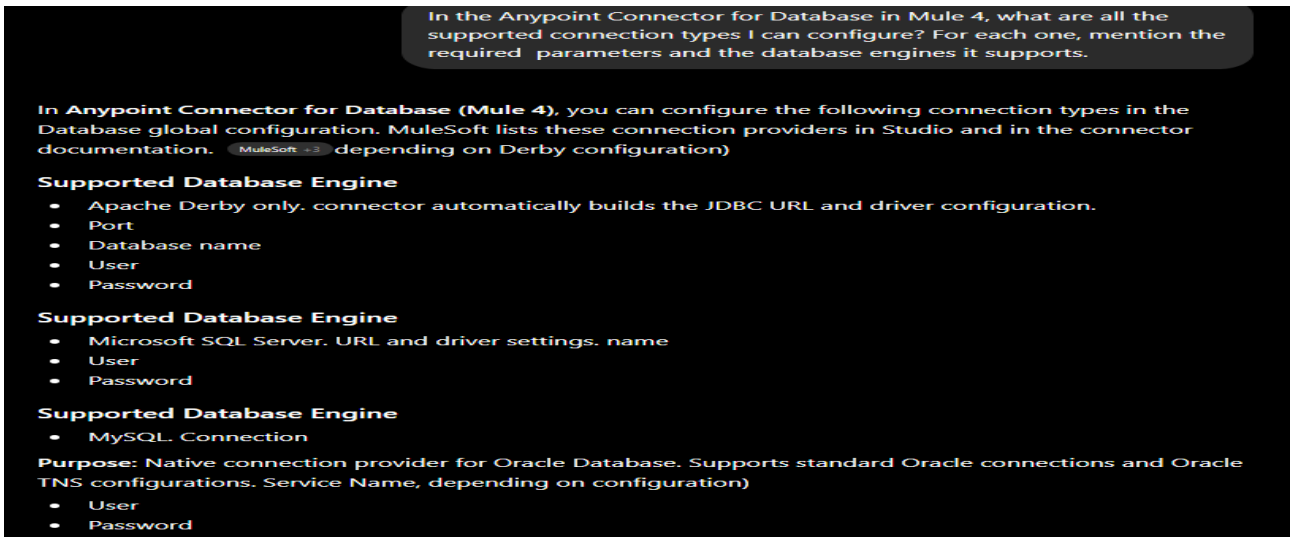

Fig. 16: Respuesta de ChatGPT a la pregunta sobre los tipos de conexión del Database Connector (1/2).

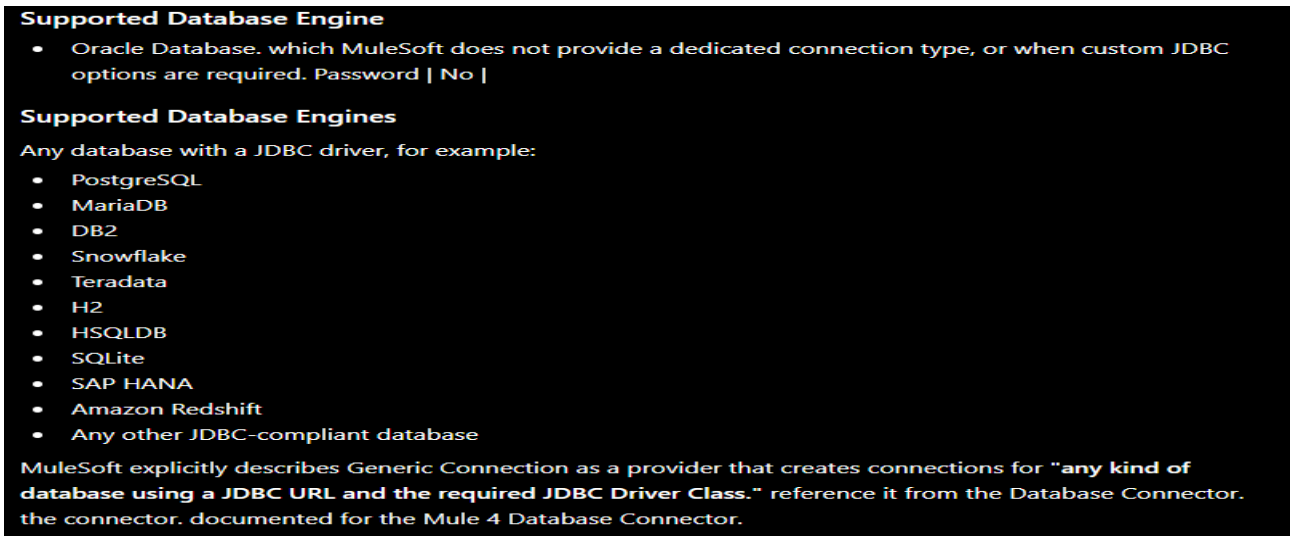

Fig. 17: Respuesta de ChatGPT a la pregunta sobre los tipos de conexión del Database Connector (2/2).

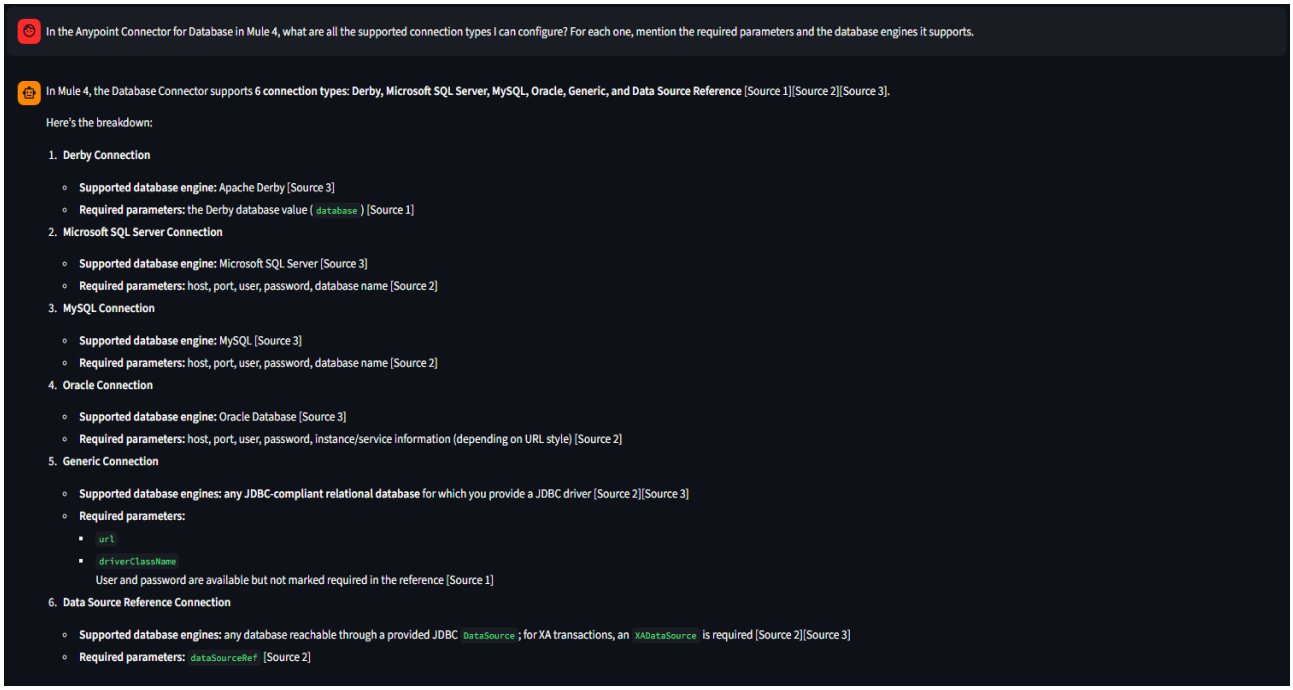

Fig. 18: Respuesta del RAG agéntico a la misma pregunta (1/2).

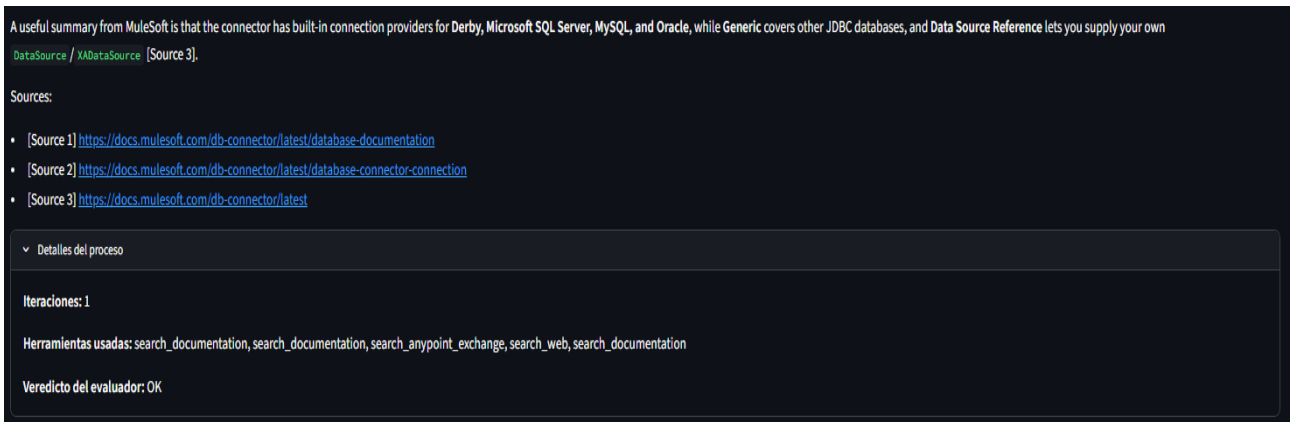

Fig. 19: Respuesta del RAG agéntico a la misma pregunta (2/2).