RAG por *dentro*

COMPONENTES, HERRAMIENTAS
Y CÓMO MEDIRLO · NIVEL INTERMEDIO

www.intelifica.com

Contenido generado con inteligencia artificial

Los textos de este libro se han redactado con modelos de inteligencia artificial y describen el estado de la tecnología en agosto de 2026.

Como cualquier contenido generado así, puede contener imprecisiones, simplificaciones o afirmaciones que hayan quedado desactualizadas. Contrasta cualquier dato del que dependa una decisión técnica o de inversión.

Intelifica · www.intelifica.com

Introducción

Este libro recorre los mismos siete temas que «RAG para tu negocio», pero por dentro: el pipeline pieza a pieza, las técnicas de recuperación, las herramientas que hay de verdad en el mercado y qué medir para saber si el sistema responde bien.

Da por sabido lo que se cuenta en el primer libro: qué es RAG, por qué un modelo se inventa cosas y en qué se diferencia de un chatbot corriente. Si eso no te suena, empieza por allí.

Los capítulos se pueden leer sueltos. Los términos van en un glosario único al final. Cuando algo exige entrar en arquitectura, se remite a «RAG en producción», que es donde se trata.

Índice

El pipeline de RAG por dentro: componentes y decisiones

FUNDAMENTOS

Una guía técnica accesible sobre cómo se construye un sistema RAG real, para responsables de proyecto y gestores de sistemas que necesitan entender las piezas del engranaje antes de decidir.

1. De la IA generativa a un asistente anclado a tus datos

Si has llegado hasta Este capítulo, probablemente ya tienes una idea general de lo que es la IA generativa y de por qué un modelo de lenguaje (LLM, del inglés Large Language Model) puede dar respuestas incorrectas o desactualizadas cuando se le pregunta sobre información específica de una empresa. En Este capítulo vamos a abrir la caja: cómo se construye realmente un sistema RAG, qué piezas lo componen y qué decisiones técnicas hay que tomar al diseñarlo.

La idea central sigue siendo la misma: un sistema RAG combina dos capacidades independientes —un **motor de búsqueda** capaz de encontrar la información relevante dentro de un conjunto de documentos, y un **modelo de lenguaje** capaz de redactar una respuesta natural apoyándose en esa información recuperada—. Lo interesante, y lo que conviene entender bien antes de evaluar cualquier propuesta técnica o comercial, es cómo se conectan ambas piezas y qué decisiones de diseño afectan a la calidad final.

> CONCEPTO CLAVE
>
> Una idea que conviene interiorizar desde el principio: **la calidad de un sistema RAG está limitada, sobre todo, por la calidad de la recuperación de información**, no tanto por lo potente que sea el modelo de lenguaje elegido. Un modelo excelente con una recuperación mediocre da respuestas mediocres; un modelo modesto con una recuperación excelente puede dar muy buenos resultados. Esto tiene implicaciones directas a la hora de invertir el presupuesto de un proyecto.

2. El pipeline técnico completo, paso a paso

Un sistema RAG se organiza en dos fases: una fase de **indexación** (la preparación de la «biblioteca», que se ejecuta una vez y se repite cada vez que cambian los documentos) y una fase de **consulta** (lo que ocurre en tiempo real cuando alguien hace una pregunta). Vamos a recorrer cada etapa.

2.1 Troceado o «chunking»

Los documentos de una empresa —manuales, PDF, páginas web, hojas de cálculo— suelen ser demasiado largos para tratarlos como una sola unidad de búsqueda. Por eso, el primer paso consiste en dividirlos en fragmentos más pequeños y manejables, llamados chunks (en español, «trozos» o «fragmentos»). A este proceso se le llama **chunking** o troceado.

Existen distintas estrategias de troceado, y la elección afecta directamente a la calidad de las respuestas. Las más habituales en proyectos reales son:

Tamaño fijo: se corta cada cierto número de caracteres o palabras, con o sin solapamiento entre fragmentos consecutivos. Es rápido de implementar, pero puede cortar una frase o una idea por la mitad.

División recursiva por estructura: respeta las fronteras naturales del texto —primero intenta cortar por párrafos, luego por frases, luego por palabras si es necesario—. Es la opción por defecto más habitual en proyectos en producción, por su buen equilibrio entre sencillez y calidad.

Troceado jerárquico (padre-hijo): mantiene fragmentos pequeños para buscar con precisión, pero recupera el fragmento «padre» más amplio que los contiene para dárselo al modelo, de forma que la respuesta tenga suficiente contexto y no se quede corta.

Troceado semántico: corta el texto donde detecta un cambio real de tema, en lugar de por un número fijo de caracteres, generando fragmentos más coherentes temáticamente.

> **A TENER EN CUENTA**
>
> No existe un tamaño de fragmento «perfecto» universal. Fragmentos demasiado pequeños pierden contexto (la respuesta llega incompleta); fragmentos demasiado grandes diluyen la relevancia (se mezcla información útil con ruido irrelevante, y el sistema tiene más dificultad para identificar qué es lo importante). El tamaño óptimo depende del tipo de documento y conviene ajustarlo probando con casos reales de tu propio negocio.

2.2 Embeddings: convertir texto en «coordenadas» de significado

Una vez trocados los documentos, cada fragmento se convierte en una lista de números —un vector— mediante un modelo especializado llamado **modelo de embeddings**. La idea, explicada de forma intuitiva, es que ese vector representa el «significado» del fragmento como si fueran unas coordenadas en un mapa: los fragmentos que hablan de temas parecidos quedan ubicados cerca entre sí en ese mapa, aunque usen palabras completamente distintas.

> **EJEMPLO PRÁCTICO**
>
> Los fragmentos «plazo de devolución de un pedido» y «cuántos días tengo para reembolsar una compra» usan palabras casi totalmente distintas, pero un buen modelo de embeddings los coloca muy cerca en ese «mapa de significado», porque tratan del mismo concepto. Esto es lo que permite que la búsqueda funcione aunque el cliente no use las palabras exactas de tu documentación.

2.3 Base de datos vectorial: el almacén organizado

Todos esos vectores se guardan en una **base de datos vectorial**, un tipo de almacén diseñado específicamente para encontrar, entre millones de vectores, cuáles son los más «cercanos» (más parecidos en significado) a otro vector dado, en fracciones de segundo. Es la pieza de infraestructura que hace viable la búsqueda semántica a gran escala.

2.4 Recuperación: la búsqueda en el momento de la pregunta

Cuando un usuario escribe una pregunta, esta se convierte también en un vector con el mismo modelo de embeddings, y la base de datos vectorial devuelve los fragmentos cuyo vector está más cerca —es decir, los más relevantes para esa pregunta—. Este paso se llama **recuperación** (retrieval), y es el que da nombre a la técnica.

Conviene saber que la búsqueda puramente semántica no siempre es suficiente: falla, por ejemplo, con referencias exactas como códigos de error, números de producto o nombres propios poco frecuentes, donde una búsqueda tradicional por palabras clave (conocida técnicamente como búsqueda léxica o BM25) puede funcionar mejor. Por eso muchos sistemas en producción combinan ambos enfoques —búsqueda semántica y búsqueda por palabras clave— en lo que se conoce como **búsqueda híbrida**, fusionando los resultados de las dos vías para obtener lo mejor de cada una.

2.5 Aumento del prompt y generación de la respuesta

Por último, los fragmentos recuperados se insertan junto con la pregunta original en la instrucción que recibe el modelo de lenguaje —a esto se le llama **aumento del prompt**, porque se «aumenta»

la pregunta original con contexto adicional—. El modelo genera entonces la respuesta final, apoyándose en ese contexto en lugar de improvisar solo con lo que aprendió en su entrenamiento general. Muchos sistemas piden además al modelo que indique de qué fragmento o documento ha sacado cada dato, lo que permite mostrar citas y fuentes verificables junto a la respuesta.

Etapa	Cuándo ocurre	Qué hace
Troceado (chunking)	Al preparar los documentos	Divide los documentos en fragmentos manejables
Embeddings	Al preparar los documentos y en cada consulta	Convierte texto en vectores de significado
Base de datos vectorial	Almacenamiento permanente	Guarda los vectores para búsquedas rápidas
Recuperación	En cada consulta del usuario	Encuentra los fragmentos más relevantes
Aumento del prompt	En cada consulta del usuario	Combina la pregunta con los fragmentos recuperados
Generación	En cada consulta del usuario	El modelo redacta la respuesta final

3. RAG frente a fine-tuning y frente a contexto largo

Una pregunta habitual en la fase de decisión de un proyecto es si conviene usar RAG, entrenar un modelo a medida (fine-tuning) o simplemente meter todo el documento dentro del prompt aprovechando que los modelos actuales admiten instrucciones muy largas (lo que se llama trabajar con «contexto largo»). No son alternativas excluyentes, pero cada una resuelve un problema distinto.

3.1 RAG

Es la opción por defecto para la mayoría de aplicaciones empresariales: conocimiento que cambia con frecuencia (precios, stock, políticas), necesidad de citar de dónde sale cada respuesta, y actualización de la información sin tener que reentrenar nada. Coste y tiempo de puesta en marcha razonables para una pyme.

3.2 Fine-tuning (afinado del modelo)

Consiste en reentrenar un modelo con ejemplos específicos para que adopte un estilo, un formato de salida o un tono de marca determinados, o para optimizar coste y velocidad usando un modelo más pequeño especializado. No es la herramienta adecuada para «enseñarle hechos nuevos» que cambian con el tiempo —un modelo afinado con tu catálogo de hoy seguirá dando esa misma información aunque el catálogo cambie mañana, a menos que se vuelva a reentrenar—.

3.3 Contexto largo (meter todo el documento en el prompt)

Los modelos de lenguaje actuales pueden aceptar instrucciones muy extensas, lo que en teoría permitiría «pegar» un documento entero en cada pregunta en lugar de construir un sistema de recuperación. Funciona razonablemente bien con corpus pequeños y estáticos, pero escala mal: cuantos más documentos tenga tu empresa, más caro y más lento se vuelve cada consulta, y se pierde la trazabilidad clara de qué fuente exacta respalda cada afirmación, algo que RAG resuelve de forma más natural.

Enfoque	Mejor para	Limitación principal
RAG	Conocimiento cambiante, trazabilidad de fuentes, coste moderado	Depende de la calidad de la recuperación

Fine-tuning	Tono de marca, formato fijo, optimizar coste/velocidad	Caro y lento de actualizar cuando cambian los hechos

Enfoque	Mejor para	Limitación principal
Contexto largo	Corpus pequeños y estables	Coste y latencia crecientes al escalar; peor trazabilidad

> **A TENER EN CUENTA**
>
> En proyectos maduros es habitual combinar enfoques: un sistema RAG como base, con un pequeño ajuste (fine-tuning) del modelo para fijar el tono de marca o un formato de respuesta específico. No es «uno u otro» de forma excluyente, sino una decisión de diseño según las necesidades de cada empresa.

4. Los componentes de un sistema RAG real

Cuando un proveedor te presenta una propuesta de proyecto RAG, es útil poder identificar qué pieza de infraestructura corresponde a cada función. Estos son los componentes habituales:

4.1 Fuente de datos y proceso de ingesta

El conjunto de documentos originales (PDF, páginas web, hojas de cálculo, bases de datos internas) y el proceso automatizado que los recoge, limpia y prepara para el troceado. Este paso, a menudo infravalorado, suele ser el que más tiempo consume en un proyecto real, especialmente si los documentos están en formatos heterogéneos o mal estructurados.

4.2 Modelo de embeddings

El motor encargado de convertir texto en vectores. Existen distintas familias, con diferencias en calidad, coste y capacidad para procesar textos largos de una sola vez.

4.3 Base de datos vectorial

El almacén donde viven los vectores. Las opciones más relevantes hoy incluyen soluciones gestionadas en la nube y soluciones de código abierto que se pueden alojar en infraestructura propia. Para el volumen de datos habitual de una pyme (documentación interna, catálogo, histórico de conversaciones), no suele hacer falta la infraestructura de máxima escala pensada para miles de millones de registros: hay opciones mucho más ligeras y económicas que cubren de sobra esas necesidades.

4.4 Motor de recuperación

La lógica que decide qué fragmentos traer para cada pregunta: cuántos fragmentos recuperar, si se combina búsqueda semántica con búsqueda por palabras clave, y si se aplica un paso adicional de reordenamiento (un modelo especializado, llamado reranker, que revisa los candidatos recuperados y los reordena por relevancia real antes de pasárselos al modelo de lenguaje, mejorando notablemente la precisión final).

4.5 Modelo de lenguaje (LLM) generador

El modelo que redacta la respuesta final a partir de la pregunta y los fragmentos recuperados. Puede ser un modelo de un proveedor externo al que se accede mediante una conexión programática (API), o un modelo alojado dentro de la infraestructura de la propia empresa por motivos de privacidad o cumplimiento normativo.

4.6 Capa de orquestación e interfaz

El software que conecta todas las piezas anteriores entre sí y las expone al usuario final —el chat en la web, la integración con WhatsApp, el panel interno para empleados—, además de gestionar aspectos como el

registro de conversaciones, los permisos de acceso a cada documento y el monitoreo del funcionamiento del sistema.

> **CONCEPTO CLAVE**
>
> Ningún componente funciona de forma aislada: la calidad final depende de cómo encajan entre sí. Un modelo de embeddings excelente con un troceado mal hecho da resultados mediocres; una base de datos vectorial de última generación no compensa una fuente de datos desactualizada. Evaluar un proyecto RAG exige mirar la cadena completa, no una sola pieza.

5. Qué preguntas hacerle a un proveedor de este tipo de soluciones

Si tu empresa está valorando contratar un proyecto de este tipo, estas preguntas ayudan a distinguir una propuesta seria y bien planteada de una superficial:

¿Cómo se actualiza la información? ¿Cada cuánto se vuelve a indexar el contenido cuando cambian los documentos, y qué proceso hay que seguir para actualizarlo?

¿Dónde vivirán mis datos? ¿En qué región se alojan, quién tiene acceso, se usan mis datos para entrenar modelos de terceros o quedan completamente aislados?

¿Cómo se controla el acceso por usuario o por documento? Si distintos empleados o clientes deben ver solo cierta información, ¿cómo se garantiza que el sistema no mezcle contenidos que no deberían cruzarse?

¿El sistema cita sus fuentes? ¿Puede el usuario final comprobar de qué documento sale cada respuesta?

¿Qué pasa cuando no encuentra información relevante? Un buen sistema debe poder reconocer que no tiene suficiente base para responder, en lugar de inventar algo.

¿Cómo se mide la calidad antes de lanzar el proyecto? ¿Existe un conjunto de preguntas de prueba con respuestas correctas conocidas contra el que se valida el sistema antes de ponerlo en producción?

¿Qué mantenimiento hace falta después del lanzamiento? ¿Quién revisa el rendimiento del sistema en el tiempo y quién decide cuándo hay que ajustar algo?

¿Cuál es el coste real, incluyendo el uso continuado? Más allá del coste inicial de puesta en marcha, conviene entender el coste recurrente asociado al volumen de consultas y al almacenamiento de datos.

> **EJEMPLO PRÁCTICO**
>
> Una empresa de distribución evalúa dos propuestas para un asistente de atención al cliente. La primera solo menciona «usamos inteligencia artificial avanzada» sin más detalle. La segunda explica claramente cómo se actualizará el catálogo cada noche, dónde se alojan los datos, cómo se probará la calidad de las respuestas antes de lanzar el sistema y qué pasa cuando un cliente pregunta algo fuera de la documentación disponible. Esta segunda propuesta, aunque no sea la más barata, ofrece muchas más garantías de que el proyecto funcionará bien a largo plazo.

6. Cómo se mide si un sistema RAG funciona bien

No basta con lanzar un asistente y confiar en que funcione bien; conviene establecer, desde el principio, una forma de medir su rendimiento. Existen tres grandes bloques de indicadores que conviene conocer, aunque sea a nivel conceptual, para poder hacer un seguimiento con tu proveedor.

6.1 Calidad de la recuperación

Mide si el sistema encuentra los fragmentos correctos para cada pregunta, independientemente de si la respuesta final está bien redactada. Se evalúa comparando, para un conjunto de preguntas de prueba, qué fragmentos debería haber encontrado el sistema frente a los que realmente encontró.

6.2 Calidad de la respuesta generada

Mide si la respuesta final está realmente fundamentada en los fragmentos recuperados (y no inventada por el modelo), si responde de verdad a la pregunta, y si cita correctamente sus fuentes.

6.3 Rendimiento de negocio

Incluye aspectos más tangibles para una empresa: tiempo de respuesta (latencia), coste por consulta, tasa de preguntas que el sistema resuelve sin intervención humana, y satisfacción de los usuarios finales.

> **A TENER EN CUENTA**
>
> Medir la calidad de un sistema RAG no es un ejercicio de una sola vez antes de lanzarlo: la documentación cambia, el volumen de preguntas crece y el comportamiento del sistema puede degradarse con el tiempo si no se revisa. Un proyecto serio incluye un proceso de seguimiento continuo, no solo una prueba inicial.

Técnicas de RAG: cómo funcionan por dentro los sistemas inteligentes de recuperación

TÉCNICAS

Una introducción con nombres técnicos explicados —búsqueda híbrida, reranking, HyDE, Corrective RAG, Self-RAG, GraphRAG y chunking— para entender qué hace cada técnica, qué problema resuelve y cuándo conviene aplicarla.

1. De la búsqueda simple al RAG inteligente

El punto de partida de cualquier sistema RAG es lo que en la literatura técnica se conoce como «Naive RAG» (RAG ingenuo o básico): se trocean los documentos en fragmentos, se convierte cada fragmento en un vector numérico mediante un modelo de **embeddings**, se guardan esos vectores en una base de datos vectorial y, cuando llega una pregunta, se busca qué fragmentos tienen vectores más parecidos al de la pregunta. Esos fragmentos se insertan en el mensaje (**prompt**) que recibe el modelo de lenguaje, junto con la pregunta original, y el modelo genera la respuesta.

> CONCEPTO CLAVE
>
> Un **embedding** es una representación numérica del significado de un texto: dos frases con un significado parecido tienen vectores «cercanos» entre sí, aunque usen palabras distintas. Es la pieza que permite buscar por significado y no solo por coincidencia literal de palabras.

El Naive RAG funciona razonablemente bien para preguntas factuales sencillas y para FAQs bien estructuradas. Sus límites aparecen en tres frentes recurrentes: recupera fragmentos poco relevantes cuando la pregunta usa vocabulario muy distinto al del documento original o incluye términos exactos (códigos, referencias, nombres propios) que el enfoque puramente semántico maneja mal; genera respuestas mal fundamentadas cuando lo recuperado no es realmente útil, porque no hay ningún paso que compruebe la calidad de lo recuperado antes de responder; y no sabe combinar información dispersa en varios documentos cuando la pregunta exige una visión de conjunto.

Las técnicas que veremos en Este capítulo son, cada una, una respuesta a uno de esos tres límites. No son alternativas excluyentes entre sí: en un sistema de producción real es habitual combinar varias de ellas —por ejemplo, búsqueda híbrida más reranking es prácticamente el estándar actual en cualquier implementación seria—, ajustando el nivel de sofisticación al riesgo y al presupuesto de cada caso de uso.

Limitación del RAG básico	Técnica que la aborda
Recuperación poco precisa con términos exactos o vocabulario distinto	Búsqueda híbrida + reranking
Preguntas vagas, ambiguas o con varias partes	Transformación de consultas (reescritura, HyDE)

Limitación del RAG básico	Técnica que la aborda
Respuestas mal fundamentadas o directamente inventadas	Corrective RAG / Self-RAG
Preguntas que exigen conectar información de muchos documentos	GraphRAG

2. Búsqueda híbrida: semántica y palabras clave a la vez

La búsqueda semántica (por embeddings) es excelente entendiendo intención y sinónimos, pero tiene un punto débil: los términos muy concretos —códigos de error, números de referencia, nombres de producto poco frecuentes, siglas— no siempre quedan bien representados en el espacio de vectores, porque el modelo de embeddings los trata como «palabras raras» sin demasiado contexto semántico distintivo. Ahí es donde la búsqueda léxica tradicional, basada en coincidencia de palabras (el algoritmo de referencia se llama **BM25**), sigue siendo insustituible: encuentra exactamente el fragmento que contiene «ERR-4471» aunque ese código no tenga ningún significado especial para un modelo de embeddings.

2.1 Cómo se combinan los dos resultados

La búsqueda híbrida lanza ambas búsquedas en paralelo —la semántica y la léxica— y después fusiona las dos listas de resultados en una sola, dando más peso a los fragmentos que aparecen bien posicionados en ambas listas. La técnica de fusión más habitual se llama **Reciprocal Rank Fusion** (fusión recíproca de rangos): en lugar de comparar directamente las puntuaciones de cada búsqueda (que no son comparables entre sí, porque usan escalas distintas), combina la posición de cada resultado en cada lista, dando más peso a lo que aparece arriba en cualquiera de las dos.

CONCEPTO CLAVE

La búsqueda híbrida es, con diferencia, la mejora con mejor relación entre esfuerzo de implementación y beneficio obtenido. La mayoría de las bases de datos vectoriales modernas ya la ofrecen de forma nativa o casi lista para usar, por lo que suele ser el primer paso recomendable al mejorar un RAG básico.

EJEMPLO PRÁCTICO

Un servicio técnico que da soporte a clientes de una plataforma de software recibe la pregunta «tengo el error E-2091 al exportar el informe». La parte «E-2091» se resuelve perfectamente con búsqueda léxica; la parte «al exportar el informe» aporta contexto semántico útil si el código de error no está documentado exactamente con ese formato en el manual. La búsqueda híbrida cubre ambos frentes en una sola consulta.

El coste de implementar búsqueda híbrida es moderado: la mayoría de bases de datos vectoriales de uso habitual (Qdrant, Weaviate, pgvector con extensiones adicionales) permiten combinar ambos tipos de índice. El coste de latencia añadido suele ser pequeño, porque las dos búsquedas pueden ejecutarse en paralelo.

3. Reranking: reordenar los resultados antes de responder

Cuando se recupera un primer conjunto de fragmentos candidatos —digamos, los veinte o treinta más parecidos a la pregunta—, no todos son igual de útiles, y el orden en el que aparecen importa: el modelo de lenguaje presta más atención a lo que está al principio del contexto que se le entrega. El **reranking** (reordenamiento) añade un segundo filtro, más lento pero más preciso, que evalúa la relevancia real de cada candidato frente a la pregunta y reordena la lista antes de pasarla al modelo de lenguaje.

3.1 Por qué hace falta un segundo paso

La búsqueda inicial (semántica, léxica o híbrida) tiene que ser rápida, porque compara la pregunta contra millones de fragmentos potenciales; por eso usa métodos aproximados y relativamente ligeros. El reranking, en cambio, solo tiene que evaluar un conjunto pequeño de candidatos —los que ya pasaron el primer filtro—, así que puede permitirse usar un modelo más lento y más preciso, normalmente un tipo de modelo llamado **cross-encoder**, que analiza la pregunta y cada fragmento juntos (en lugar de comparar vectores calculados por separado), lo que le da mucha más capacidad de juicio sobre la relevancia real.

CONCEPTO CLAVE

Un servicio de reranking muy usado en la industria es Cohere Rerank, aunque también existen modelos cross-encoder de código abierto. La idea general vale para cualquiera de ellos: recuperar «amplio y rápido» primero, y luego «afinar y preciso» con un número menor de candidatos.

EJEMPLO PRÁCTICO

Una gestoría recupera 25 fragmentos candidatos para la pregunta «requisitos para darse de alta como autónomo societario». Varios de esos fragmentos mencionan «autónomo» de forma genérica sin ser realmente relevantes. El reranker analiza cada uno frente a la pregunta completa y coloca en primer lugar los tres o cuatro fragmentos que hablan específicamente del caso «societario», mejorando notablemente la calidad de la respuesta final sin cambiar nada en la búsqueda inicial.

El coste de latencia que añade el reranking suele ser bajo si se aplica solo sobre un conjunto reducido de candidatos (veinte o treinta, no miles), lo que lo convierte en una de las mejoras con mejor relación coste-beneficio después de la búsqueda híbrida.

4. Transformación de consultas: reescritura y HyDE

No todas las preguntas que hacen los usuarios están bien formuladas para una búsqueda eficaz. Preguntas vagas («¿esto funciona bien?»), preguntas con varias partes a la vez («¿qué garantía tiene y puedo cambiarlo por otro modelo?») o preguntas que usan un lenguaje muy distinto al de los documentos originales, reducen la calidad de la recuperación aunque el sistema de búsqueda en sí sea excelente. La transformación de consultas actúa antes de buscar, reescribiendo o descomponiendo la pregunta original para mejorar las probabilidades de encontrar lo relevante.

4.1 Reescritura y descomposición

La forma más sencilla es la reescritura: el propio modelo de lenguaje recibe la pregunta original y genera una versión más clara, más explícita o con sinónimos añadidos, antes de lanzarla contra el sistema de búsqueda. Cuando la pregunta combina varias cuestiones distintas, se puede aplicar descomposición: dividirla en subpreguntas independientes, buscar la respuesta a cada una por separado, y combinar los resultados al final.

4.2 HyDE: buscar con una respuesta hipotética

Una técnica más sofisticada y menos intuitiva es **HyDE** (Hypothetical Document Embeddings, o «embeddings de documento hipotético»). En lugar de buscar directamente con la pregunta del usuario, el sistema le pide primero al modelo de lenguaje que redacte una respuesta hipotética plausible a esa pregunta —aunque el modelo no tenga garantía de que sea correcta— y luego convierte esa respuesta hipotética en un vector, y busca con ese vector en lugar de con el de la pregunta original.

CONCEPTO CLAVE

La lógica detrás de HyDE es que una respuesta (aunque sea inventada) suele parecerse más, en estilo y vocabulario, a los fragmentos de un documento real que a la pregunta corta y a veces informal que escribe un usuario. Buscar con «algo que se parece a la respuesta» puede encontrar mejores candidatos que buscar con la pregunta en sí.

EJEMPLO PRÁCTICO

Ante la pregunta «¿por qué me cobran de más este mes?», HyDE generaría primero una respuesta hipotética del tipo «el cargo adicional puede deberse a un cambio de tarifa, a un servicio adicional contratado o a un ajuste de facturación de meses anteriores», y buscaría con ese texto. Es mucho más probable que ese texto hipotético coincida semánticamente con los apartados reales de la política de facturación que la pregunta corta original.

A TENER EN CUENTA

La transformación de consultas añade una llamada extra al modelo de lenguaje antes de buscar, lo que incrementa la latencia y el coste por consulta. Conviene reservarla para los casos en los que la pregunta original es realmente ambigua o compleja, no aplicarla de forma sistemática a cualquier consulta sencilla.

5. Verificar antes de responder: Corrective RAG, Self-RAG y GraphRAG

5.1 Corrective RAG: corregir la recuperación sobre la marcha

Corrective RAG (RAG correctivo) añade un paso de evaluación justo después de recuperar los fragmentos: un componente (que puede ser el propio modelo de lenguaje con un prompt específico) valora si los fragmentos recuperados son realmente relevantes para la pregunta. Si la evaluación es negativa o dudosa, el sistema no se conforma con lo que tiene: puede lanzar una nueva búsqueda con parámetros distintos, ampliar la consulta, o incluso recurrir a una fuente externa (por ejemplo, una búsqueda web) antes de generar la respuesta final.

5.2 Self-RAG: el modelo evalúa su propia respuesta

Self-RAG (RAG autorreflexivo) va un paso más allá: no solo evalúa lo recuperado, sino que también evalúa la respuesta ya generada, comprobando si está bien fundamentada en los fragmentos usados o si el modelo se ha desviado de ellos. Si detecta que la respuesta no está suficientemente respaldada, el sistema puede reformularla, buscar información adicional, o directamente abstenerse de responder con seguridad y avisar de que no dispone de información suficiente.

CONCEPTO CLAVE

Tanto Corrective RAG como Self-RAG introducen un «bucle» en el proceso: en lugar de un flujo lineal (buscar → responder), el sistema puede volver atrás una o varias veces antes de dar la respuesta definitiva. Esto mejora la fiabilidad, pero incrementa la latencia y el coste, por lo que suele reservarse para dominios de alto riesgo: salud, ámbito legal, finanzas, o cualquier caso en el que una respuesta incorrecta tenga consecuencias serias.

EJEMPLO PRÁCTICO

Un asistente de una correduría de seguros usa Self-RAG para responder preguntas sobre coberturas de pólizas. Si un cliente pregunta por una cobertura muy específica y el sistema no encuentra un fragmento claramente aplicable, en lugar de generar una respuesta aproximada y arriesgada, el propio sistema detecta la falta de fundamento y responde: «no encuentro esa cobertura especificada en tu póliza, te recomiendo confirmarlo con tu agente», evitando así una posible alucinación con consecuencias reales para el cliente.

5.3 GraphRAG: conectar información con un grafo de conocimiento

El tercer patrón de esta familia aborda un problema distinto: preguntas que requieren combinar información repartida en muchos documentos («¿qué relación han tenido estos dos proveedores a lo largo del tiempo?»). **GraphRAG** construye, de forma previa (durante la fase de indexación, no en el momento de la pregunta), un grafo de conocimiento: una red de entidades (personas, empresas, productos, eventos) y las relaciones que hay entre ellas, extraídas automáticamente de los documentos por el propio modelo de lenguaje. Después, se agrupan las entidades relacionadas entre sí en «comunidades» temáticas, lo que permite responder tanto preguntas puntuales como preguntas de visión de conjunto sobre un tema entero.

> **A TENER EN CUENTA**
>
> GraphRAG tiene un coste de indexación considerablemente más alto que un RAG estándar, porque implica analizar todo el corpus en profundidad para extraer entidades y relaciones antes de poder responder ninguna pregunta. Para catálogos pequeños o casos de uso con preguntas mayoritariamente puntuales, suele ser una complejidad innecesaria; brilla especialmente en corpus grandes con preguntas que exigen conectar hechos dispersos.

6. Estrategias de chunking: fijo, recursivo, semántico y jerárquico

Todas las técnicas anteriores dependen de que, antes, los documentos se hayan troceado (chunking) de forma adecuada. Un mismo documento puede convertirse en fragmentos muy distintos según la estrategia usada, y esa decisión afecta directamente a la calidad de todo lo que viene después. Veamos las cuatro estrategias más habituales, de la más simple a la más sofisticada.

6.1 Troceado de tamaño fijo

Corta el texto cada cierto número de caracteres o de tokens (unidades de texto que usa el modelo), normalmente con un pequeño solapamiento entre fragmentos consecutivos para no perder contexto en los cortes. Es la opción más simple y rápida de implementar, útil para prototipos o pruebas iniciales, pero tiene el inconveniente de que puede cortar frases o ideas por la mitad sin ningún criterio, generando fragmentos con poco sentido propio.

6.2 División recursiva

En lugar de cortar a ciegas cada N caracteres, la división recursiva usa una jerarquía de separadores naturales del texto: primero intenta cortar por párrafos, si un fragmento sigue siendo demasiado grande corta por líneas, luego por frases, y como último recurso por palabras. El resultado son fragmentos que respetan mucho mejor las fronteras naturales del contenido. Es, hoy en día, la opción por defecto en la mayoría de implementaciones de producción, por su buena relación entre sencillez y calidad.

6.3 Troceado semántico

Va un paso más allá: mide la similitud de significado entre frases consecutivas y corta el fragmento justo donde esa similitud cae por debajo de un umbral, es decir, donde el tema cambia de verdad. El resultado son fragmentos temáticamente coherentes de principio a fin, lo que suele mejorar la calidad de la recuperación posterior. A cambio, requiere más cómputo durante la fase de indexación, porque hay que analizar el texto frase a frase antes de decidir dónde cortar.

6.4 Troceado jerárquico (padre-hijo)

Resuelve un problema muy concreto: a veces el fragmento más preciso para encontrar un dato es demasiado corto para que el modelo entienda bien el contexto completo alrededor de ese dato. La solución es mantener dos niveles: fragmentos «hijo» pequeños y muy específicos, que son los que se usan para la búsqueda

(porque localizan con precisión el dato relevante), y fragmentos «padre»

más grandes que los engloban, que son los que finalmente se entregan al modelo de lenguaje para que tenga contexto suficiente al redactar la respuesta.

Estrategia	Cuándo conviene	Principal limitación
Tamaño fijo	Prototipos rápidos, pruebas iniciales	Puede cortar ideas por la mitad
Recursiva	Producción general, opción por defecto	Menos precisión temática que la semántica
Semántica	Documentos con temas mezclados, alta exigencia de precisión	Más coste de cómputo al indexar
Jerárquica (padre- hijo)	Cuando el dato preciso necesita contexto amplio para entenderse	Mayor complejidad de gestión del índice

EJEMPLO PRÁCTICO

Un fabricante de maquinaria industrial trocea sus manuales técnicos con la estrategia jerárquica: el fragmento «hijo» que se recupera puede ser una sola instrucción de mantenimiento («comprobar el nivel de aceite cada 200 horas»), pero al modelo se le entrega el apartado «padre» completo, que incluye el contexto de a qué modelo de máquina aplica esa instrucción y qué precauciones tomar antes de hacerlo.

Comparando herramientas reales: bases de datos vectoriales, orquestadores y embeddings

TECNOLOGÍAS

Un mapa práctico del mercado actual de tecnologías RAG —qué producto conviene según el tamaño de tu pyme, qué hace cada framework de orquestación y cómo elegir un modelo de embeddings— sin necesidad de saber programar.

1. Del concepto a la compra: qué hay que decidir realmente

El capítulo equivalente de «RAG para tu negocio», el primer libro de esta colección explicaba, con analogías y sin jerga, las cinco piezas que forman cualquier sistema RAG: la base de datos vectorial, los embeddings, el motor de búsqueda semántica, el orquestador y el modelo de lenguaje. Ese nivel de comprensión es suficiente para entender qué hace un sistema de este tipo, pero no todavía para tomar decisiones concretas sobre cómo construirlo, porque cada una de esas piezas conceptuales se resuelve en la práctica con productos y herramientas específicas que compiten entre sí, tienen precios distintos y encajan mejor o peor según el tamaño y las necesidades de cada negocio.

Este capítulo entra en ese terreno: en lugar de hablar de «una base de datos vectorial» en abstracto, hablaremos de Pinecone, Qdrant, Weaviate, pgvector, Milvus, Chroma, MongoDB Atlas Vector Search y LanceDB, con sus ventajas y limitaciones reales. En lugar de hablar de «un orquestador» en abstracto, hablaremos de LangChain, LangGraph y LlamaIndex. Y en lugar de hablar de «embeddings» en abstracto, hablaremos de los proveedores concretos que los generan —OpenAI, Cohere, Jina, y modelos de código abierto como BGE o E5— y de una técnica que suele acompañarlos, el reranking, que mejora la precisión final de lo que se recupera.

> **CONCEPTO CLAVE**
>
> No existe una única «mejor» base de datos vectorial ni un único «mejor» orquestador: existe la herramienta más adecuada para el tamaño de tu catálogo documental, tu presupuesto, tu equipo técnico y el tipo de preguntas que tus usuarios harán. Elegir bien no consiste en perseguir la opción más potente del mercado, sino en evitar tanto quedarse corto como sobredimensionar el sistema.

1.1 Un aviso antes de empezar: esto no sustituye una decisión técnica

Este capítulo está pensado para que una persona responsable de un negocio —sin formación en programación— pueda leer una propuesta técnica de un proveedor de IA, entender de qué está hablando, y hacer las preguntas adecuadas. No pretende convertir al lector en alguien capaz de instalar y configurar estas herramientas por su cuenta: esa es la labor de un equipo técnico como el de Intelifica, que además debe valorar factores adicionales —seguridad, cumplimiento normativo, integración con los sistemas ya existentes— que van más allá de lo que cubre esta introducción. Por ejemplo, cuando una gestoría con 40 empleados recibe una propuesta que menciona «usaremos Milvus con aceleración por GPU para escalar a cientos de millones de vectores», el contenido de Este capítulo le permite reconocer que su archivo de unos pocos miles de documentos no necesita ni de lejos esa capacidad, y que una alternativa como pgvector sería probablemente más sensata y más barata de mantener.

2. Bases de datos vectoriales: el mercado, producto a producto

Como vimos en el documento básico, la base de datos vectorial es el «almacén de significados» del sistema. En el mercado actual conviven productos con filosofías muy distintas: algunos son servicios totalmente gestionados en la nube (no hay que instalar ni mantener nada), otros son software de código abierto que una empresa puede alojar donde prefiera, y otros son, sencillamente, una función añadida a una base de datos que la empresa ya utiliza para otras cosas. Repasamos los más relevantes para una pyme española con stack habitual de Laravel, FastAPI o Python.

2.1 Servicios totalmente gestionados

Pinecone es probablemente el nombre más conocido del sector: un servicio en la nube, «llave en mano», que se ocupa de toda la infraestructura por detrás. Su gran ventaja es la velocidad de puesta en marcha —no hay servidores que configurar ni mantener— y un buen soporte para arquitecturas de inteligencia artificial agéntica (sistemas donde el modelo toma varias decisiones encadenadas). Su contrapartida es el modelo de precio: el coste crece con el volumen de datos y de consultas, lo que puede convertirse en una partida presupuestaria relevante a medida que el proyecto crece.

MongoDB Atlas Vector Search sigue una filosofía parecida —gestión totalmente en la nube—, pero con una particularidad interesante: permite guardar en un mismo sitio los vectores, los documentos en formato JSON y sus metadatos. Para una empresa que ya usa MongoDB Atlas para otras partes de su aplicación, añadir búsqueda vectorial no supone incorporar una pieza tecnológica completamente nueva, sino activar una función adicional sobre una infraestructura que el equipo ya conoce.

2.2 Código abierto, listo para producción

Qdrant, escrito en el lenguaje de programación Rust (conocido por su eficiencia), destaca por una relación muy favorable entre precio y rendimiento, y por un sistema de filtrado avanzado que permite combinar la búsqueda por significado con condiciones muy precisas (por ejemplo, «solo documentos de la categoría factura, publicados después de una fecha concreta»). Es una opción sólida para pymes con presupuesto ajustado que aun así necesitan buen rendimiento en producción.

Weaviate aporta como diferencial la búsqueda híbrida nativa: combina en una sola consulta la búsqueda semántica (por significado), la búsqueda léxica clásica (por coincidencia de palabras) y el filtrado por metadatos, sin que haya que montar esa combinación «a mano» encima. También tiene soporte multimodal, es decir, capacidad de trabajar no solo con texto sino con imágenes u otros formatos dentro del mismo almacén. Su punto débil habitual es una curva de aprendizaje algo más pronunciada, en parte porque su forma de consultarlo se basa en GraphQL, un lenguaje de consulta que exige cierta familiaridad técnica.

Milvus (y su versión gestionada en la nube, Zilliz Cloud) está diseñado para una escala verdaderamente masiva —cientos de millones de vectores— con aceleración por tarjeta gráfica (GPU) para acelerar las búsquedas. Es la herramienta adecuada para plataformas con volúmenes de datos muy grandes, pero su complejidad operativa lo convierte en una opción sobredimensionada —y por tanto poco recomendable— para la inmensa mayoría de pymes, cuyos catálogos documentales rara vez superan unos pocos millones de fragmentos.

2.3 pgvector: la opción natural si ya usas PostgreSQL

pgvector no es una base de datos independiente, sino una extensión que añade capacidad de búsqueda vectorial a PostgreSQL, uno de los gestores de bases de datos relacionales más usados del mundo y, en particular, muy habitual en el stack tecnológico de Intelifica junto con Laravel y FastAPI. Su ventaja principal es evidente: si una empresa ya tiene su información de negocio en PostgreSQL, no necesita levantar ni mantener ninguna infraestructura nueva para incorporar búsqueda semántica, sino simplemente activar una extensión sobre la base de datos que ya opera, factura y respalda. Esto simplifica enormemente el mantenimiento, la seguridad y las copias de respaldo, porque todo vive en un único sistema en lugar de en dos separados.

Como referencia orientativa, pgvector funciona bien hasta volúmenes de en torno a diez millones de vectores —recordemos que cada documento troceado genera varios fragmentos, así que ese número no equivale al número de documentos originales—; por encima de esa cifra conviene valorar alternativas pensadas específicamente para escalas mayores. Para el perfil típico de pyme que atiende esta colección —con catálogos que van de unos cientos a unos pocos millones de fragmentos— esa cifra rara vez supone una limitación real.

2.4 Opciones para prototipar y arquitecturas sin servidor

Chroma es, en la práctica, la forma más rápida de pasar de cero a tener una búsqueda vectorial funcionando en una prueba de concepto: se instala en minutos y sirve muy bien para validar una idea o hacer una demostración interna. No está pensado, sin embargo, para sostener el tráfico de un sistema en producción con muchos usuarios simultáneos, así que conviene verlo como un banco de pruebas, no como destino final.

LanceDB propone un enfoque distinto: en lugar de depender de un servidor que esté siempre encendido, almacena los datos sobre almacenamiento de objetos en la nube (del tipo Amazon S3 o Google Cloud Storage), lo que encaja bien con arquitecturas «sin servidor» y con proyectos que combinan texto con otros formatos, como imágenes. Conviene mencionar también, para evitar confusiones, a **Faiss**: una librería —no una base de datos completa— muy usada en investigación para experimentar con búsqueda por similitud, pero sin las funciones de persistencia y consulta que necesita un sistema en producción real. Si un proveedor propone Faiss como base del sistema definitivo de una pyme, vale la pena preguntar por qué no se ha optado por una alternativa pensada para producción.

2.5 Tabla comparativa y orientación por perfil de pyme

Herramienta	Punto fuerte	Encaja mejor con...
Pinecone	Gestión total, puesta en marcha rápida	Equipos sin infraestructura propia que priorizan velocidad
Qdrant	Precio/rendimiento, filtrado avanzado	Pymes con presupuesto ajustado y algo de capacidad técnica
Weaviate	Búsqueda híbrida nativa, soporte multimodal	Negocios donde la palabra clave importa tanto como el significado
pgvector	Reutiliza la base de datos PostgreSQL existente	Empresas con stack Laravel/FastAPI y PostgreSQL ya en marcha
Milvus / Zilliz Cloud	Escala masiva, aceleración por GPU	Plataformas con decenas o cientos de millones de vectores
MongoDB Atlas Vector Search	Vectores y JSON en un mismo sitio	Empresas que ya operan sobre MongoDB Atlas
Chroma	Arranque casi inmediato	Pruebas de concepto y demostraciones internas
LanceDB	Sin servidor, sobre almacenamiento de objetos	Arquitecturas serverless y contenido multimodal

> **A TENER EN CUENTA**
>
> Para el caso típico de una pyme —de cientos de miles a unos pocos millones de fragmentos, con volumen de consultas moderado— tanto pgvector como Qdrant suelen cubrir sobradamente las necesidades reales, con una fracción del coste operativo de las opciones de gran escala. Weaviate solo merece la pena si la búsqueda híbrida es, desde el primer día, un requisito central del proyecto. Desconfía de cualquier propuesta que hable de «escalar a cientos de millones de vectores» para un catálogo que hoy cabría en unas pocas carpetas.

3. Los orquestadores: LangChain, LangGraph y LlamaIndex

El documento básico presentó el orquestador con la analogía del director de una orquesta sinfónica: la pieza que no toca ningún instrumento, pero que decide en qué orden entra cada uno. En la práctica, ese papel lo asumen «cajas de herramientas» de software —frameworks, en la terminología del sector— que un equipo técnico usa para no tener que programar desde cero cada uno de esos pasos. Los tres nombres que aparecen con más frecuencia en este terreno son LangChain, LangGraph y LlamaIndex, y aunque están relacionados entre sí, resuelven necesidades ligeramente distintas.

3.1 LangChain: el «kit de piezas» generalista

LangChain es, con diferencia, el framework más extendido para construir aplicaciones que combinan modelos de lenguaje con fuentes de información externas y con otras herramientas (consultar un calendario, enviar un correo, buscar en internet). Su gran valor está en el ecosistema: cuenta con conectores ya hechos para prácticamente cualquier base de datos vectorial, cualquier proveedor de modelos de lenguaje y decenas de fuentes de datos habituales, de modo que un equipo técnico no tiene que «reinventar la rueda» para conectar cada pieza. Pensemos en LangChain como una gran caja de piezas de construcción estandarizadas que encajan entre sí según unas reglas conocidas, acelerando el montaje de un sistema RAG razonablemente estándar.

3.2 LangGraph: cuando el proceso deja de ser una línea recta

El documento básico mencionó que, en los sistemas RAG más avanzados, el orquestador puede tomar pequeñas decisiones sobre la marcha: repetir una búsqueda con otras palabras si la primera no dio buenos resultados, o consultar dos fuentes distintas antes de responder. Este tipo de comportamiento —con bucles, condiciones y posibilidad de repetir pasos— es precisamente lo que **LangGraph** está pensado para modelar. Es una extensión de LangChain que representa el proceso completo como un grafo: en lugar de una secuencia fija de pasos, define un mapa de posibles caminos («si ocurre esto, ve a este paso; si ocurre aquello, repite el anterior»), con memoria de lo ocurrido en cada punto del recorrido. Esta capacidad de «ir y volver» es justamente lo que hace posible los patrones de RAG más sofisticados —verificar si lo recuperado es realmente relevante antes de redactar la respuesta, o encadenar varias búsquedas para resolver una pregunta compleja— y es, no por casualidad, el enfoque que usa Intelifica en su propio stack para construir asistentes con este tipo de comportamiento más inteligente.

3.3 LlamaIndex: especialista en la parte de datos

Mientras que LangChain y LangGraph tienen una vocación generalista —sirven para construir todo tipo de aplicaciones con modelos de lenguaje, no solo sistemas RAG—, **LlamaIndex** nació con un enfoque más concreto: especializarse en la indexación y recuperación de datos, es decir, en trocear documentos, organizarlos de forma jerárquica (con resúmenes de alto nivel que apuntan a secciones más detalladas) y consultar varias fuentes a la vez. Para un proyecto centrado exclusivamente en «hacer preguntas sobre un conjunto de documentos» —por ejemplo, un buscador sobre una hemeroteca de veinte años de artículos— LlamaIndex ofrece atajos ya pensados para ese caso, mientras que un proceso con varias decisiones encadenadas y conexión a otros sistemas (como consultar el estado de una suscripción y escalar una queja a una persona) encaja mejor con LangGraph, apoyado en el ecosistema de conectores de LangChain.

Conviene aclarar un malentendido frecuente: estos tres frameworks no son alternativas a Pinecone, Qdrant o pgvector, sino que se usan junto a ellas. La base de datos vectorial guarda el significado de los documentos; el orquestador decide cuándo y cómo consultarla, y qué hacer con el resultado. Al leer una propuesta técnica, es normal —y buena señal— ver mencionadas ambas cosas a la vez, por ejemplo «LangGraph sobre pgvector» o «LlamaIndex con Qdrant».

4. Modelos de embeddings y reranking: afinando la calidad de las respuestas

El documento básico explicó los embeddings con la analogía de una huella dactilar del significado de un texto. Toca ahora dar un paso más: ¿quién genera exactamente esa huella, y qué diferencia a un proveedor de otro? Y, como complemento, presentaremos una técnica adicional —el **reranking**, o reordenamiento— que muchos sistemas RAG bien construidos incorporan para mejorar todavía más la precisión de lo que se recupera antes de dárselo al modelo de lenguaje.

4.1 Los proveedores de modelos de embeddings

Existen, a grandes rasgos, dos tipos de proveedores de modelos de embeddings: los servicios comerciales a los que se accede mediante una llamada a internet (pagando por uso), y los modelos de código abierto que una empresa puede ejecutar en su propia infraestructura.

Entre los servicios comerciales, **OpenAI** ofrece su familia de modelos «text-embedding-3», ampliamente usada por su buen equilibrio entre calidad y coste, y por integrarse de forma natural en proyectos que ya usan otros productos de OpenAI. **Cohere** ofrece su familia «Embed», con una versión reciente (Embed v4) que destaca por ser multimodal: puede generar huellas numéricas no solo de texto, sino también de imágenes, situándolas en el mismo «mapa de significados», lo que resulta valioso para negocios cuyo conocimiento vive también en fotografías, capturas de pantalla o tablas escaneadas. **Jina**, por su parte, se ha especializado en modelos de «contexto largo», capaces de procesar fragmentos de texto mucho más extensos —hasta unos ocho mil fragmentos de palabra o «tokens» de una sola vez— antes de generar su huella numérica, lo que resulta útil para documentos con referencias que se extienden a lo largo de muchas páginas.

Entre los modelos de código abierto, destacan familias como **BGE**, **E5** y **GTE**, disponibles para que cualquier empresa las ejecute en su propia infraestructura, sin depender de un servicio externo ni pagar por cada consulta. La ventaja de esta vía es el control total sobre dónde vive el dato —relevante con requisitos estrictos de privacidad o de residencia de datos en la Unión Europea— y la ausencia de coste por uso; la contrapartida es que alguien debe alojar y mantener ese modelo en marcha, complejidad que un servicio comercial gestionado se ahorra. En cualquier caso, elegir un modelo de embeddings no es solo una decisión técnica: influye en el idioma con el que mejor funciona el sistema (algunos rinden notablemente mejor en español que otros), en el coste recurrente del asistente, y en dónde viajan los datos de la empresa al generar cada huella numérica —relevante si se manejan datos de clientes sujetos al Reglamento General de Protección de Datos—.

4.2 El reranking: una segunda criba antes de responder

El motor de búsqueda semántica, presentado en el documento básico, hace un buen trabajo encontrando candidatos relevantes con rapidez entre miles o millones de fragmentos, pero esa velocidad tiene una contrapartida: el criterio que usa para medir «cercanía de significado» es relativamente sencillo y, en ocasiones, no distingue bien entre un fragmento que es realmente la mejor respuesta y otro que solo está relacionado de forma superficial con la pregunta.

El **reranking** (reordenamiento) añade un segundo paso, más lento pero mucho más preciso, que solo se aplica sobre ese grupo reducido de candidatos —no sobre todo el archivo—: un modelo especializado, llamado reranker, examina la pregunta original junto con cada uno de los candidatos, uno por uno, y les asigna una puntuación de relevancia real mucho más fina que la del primer filtro. Es como si, tras un primer filtro rápido que selecciona veinte currículums de una pila de mil, un segundo revisor —más lento pero más criterioso— leyera esos veinte con calma y decidiera cuáles cinco son realmente los más adecuados. Un despacho de abogados que pregunta «¿qué plazo tengo para recurrir una sanción de tráfico?» recibe así, gracias a este segundo filtro, el fragmento específico sobre plazos de recurso en primer lugar, en vez de mezclado entre otros fragmentos solo relacionados de forma superficial con sanciones de tráfico en general.

El proveedor más citado en este terreno es **Cohere Rerank**, un servicio comercial especializado exclusivamente en esta tarea, aunque también existen modelos de reranking de código abierto — de la familia técnica llamada cross-encoder— que una empresa puede alojar por su cuenta, siguiendo la misma

lógica de control y coste que se explicó para los embeddings de código abierto.

A TENER EN CUENTA

El reranking añade calidad, pero también un paso más al proceso, con su correspondiente coste y una pequeña demora adicional. Para un asistente sencillo de preguntas frecuentes, puede que no compense. Para un sistema donde la precisión importa especialmente —una asesoría, un servicio sanitario informativo, un departamento legal— suele merecer la pena, porque el coste de una respuesta imprecisa es mayor que el de una fracción de segundo de espera.

Casos de uso de RAG por sector: qué alimenta el sistema y cómo medir el éxito

CASOS DE USO

Un recorrido por cinco sectores —pymes en general, comercio electrónico, medios digitales, ámbito legal y formación interna— con el documento fuente, las preguntas típicas y las métricas de éxito de cada caso de uso.

1. Un marco común: documentos, preguntas y métricas

Antes de entrar sector por sector, conviene fijar una forma ordenada de analizar cualquier caso de uso de RAG. En Intelifica, cuando evaluamos si un caso de uso tiene sentido para un cliente, respondemos siempre a tres preguntas, en este orden.

1.1 ¿Qué documentos alimentan el sistema?

Un asistente RAG es tan bueno como la documentación de la que dispone. Antes de pensar en la tecnología, hay que hacer inventario: catálogos de producto, políticas internas, manuales, contratos tipo, archivo histórico de artículos, actas, normativa aplicable. Un error habitual es lanzar un proyecto con documentación desactualizada, incompleta o contradictoria («basura entra, basura sale»); la calidad de las respuestas nunca supera la calidad de esos documentos de origen.

1.2 ¿Qué preguntas resuelve realmente?

Las preguntas factuales y acotadas —«¿cuál es el plazo de devolución?», «¿qué dice la cláusula 4 del contrato modelo?»— son las que mejor resuelve un sistema RAG bien construido. Las que exigen combinar información de muchas fuentes o razonar sobre datos numéricos exactos necesitan patrones más sofisticados, que se desarrollan en «RAG en producción», el tercer libro de esta colección.

1.3 ¿Cómo se mide el éxito?

Un proyecto de este tipo no se juzga por «si funciona la demo», sino por indicadores de negocio sostenidos en el tiempo. Los más habituales:

Tiempo de respuesta: cuánto tarda el usuario en obtener una respuesta útil, comparado con el proceso anterior.

Tasa de resolución: qué porcentaje de consultas se resuelven sin intervención humana adicional.

Tasa de escalado: con qué frecuencia el sistema deriva correctamente a una persona cuando no tiene una respuesta fiable, en lugar de inventar una.

Satisfacción: valoración directa del usuario (clientes o empleados) tras la interacción.

Impacto operativo: horas de trabajo humano ahorradas, reducción de devoluciones, reducción de tickets de soporte repetidos.

> CONCEPTO CLAVE
>
> La **tasa de resolución** mide qué proporción de las consultas que recibe el asistente terminan resueltas sin que un empleado tenga que intervenir. Es, junto con la satisfacción del usuario, el indicador más citado a la hora de justificar la inversión en un proyecto de RAG ante la dirección de una pyme.

Con este marco en la cabeza, los siguientes cinco capítulos aplican la misma estructura — documentos, preguntas, métricas— a cinco sectores distintos, con especial atención al de medios y publicaciones digitales, uno de los verticales de mayor interés para Intelifica.

2. Pymes en general: atención al cliente y soporte interno

La mayoría de las pymes, independientemente de su sector, comparten dos frentes donde un asistente RAG aporta valor casi de inmediato: la atención al cliente de cara al público y el soporte interno de cara al propio equipo.

2.1 Atención al cliente

Aquí el sistema se alimenta de condiciones generales de venta, catálogo de productos o servicios, políticas de garantía y devolución, y el histórico de preguntas frecuentes ya resueltas por el equipo humano —una fuente de información a menudo infravalorada—. Las preguntas típicas giran en torno a plazos, precios, cobertura de garantía, formas de pago y estado de un pedido. El éxito se mide con el tiempo medio de primera respuesta, la tasa de resolución sin intervención humana y la satisfacción recogida tras la conversación.

2.2 Soporte interno

El segundo frente alimenta el sistema con manuales de procedimiento, protocolos de seguridad, políticas de recursos humanos y documentación técnica de equipos o maquinaria. Las preguntas suelen referirse a «cómo se hace X» o «qué procedimiento sigo en la situación Y». Aquí el indicador más relevante suele ser el tiempo ahorrado por empleado y semana, y la reducción de interrupciones a compañeros con más experiencia —un coste real, aunque poco visible en la contabilidad tradicional.

> **EJEMPLO PRÁCTICO**
>
> Una gestoría administrativa de doce personas implementa un asistente interno alimentado con sus propios procedimientos de tramitación (altas de autónomos, modelos fiscales, plazos habituales). En dos meses, las consultas internas sobre «cómo se tramita X» que antes recaían siempre en la misma persona sénior caen a la mitad, y esa persona recupera varias horas semanales para tareas de mayor valor.

3. Comercio electrónico: catálogo, fichas técnicas y devoluciones

El comercio electrónico es uno de los sectores donde la inversión en un asistente RAG suele amortizarse más rápido, porque cada compra dudosa que se aclara antes del pago es una devolución —con su coste logístico— que no llega a producirse.

3.1 Documentos fuente

El sistema se alimenta principalmente de las fichas técnicas de cada producto, el catálogo completo con variantes (tallas, colores, compatibilidades), la política de envíos y devoluciones, y a menudo también de reseñas y preguntas ya respondidas por el servicio de atención al cliente en el pasado. Cuanto más completas y estructuradas estén las fichas de producto, mejor funciona el asistente.

3.2 Preguntas típicas

Compatibilidad entre productos, medidas exactas, materiales, plazos de entrega a una zona concreta, condiciones para devolver un artículo ya abierto o usado, disponibilidad de tallas o colores. Muchas de estas preguntas hoy se pierden en el limbo de un formulario de contacto que tarda horas o días en responderse, tiempo en el que el cliente puede simplemente abandonar la compra.

EJEMPLO PRÁCTICO

Una tienda online de electrodomésticos de cocina conecta su asistente al catálogo completo con fichas técnicas de dimensiones y potencia. Un cliente pregunta si un horno concreto «cabe en un hueco de encastre de 60 por 60 centímetros»; el asistente compara las medidas exactas de la ficha con la pregunta y responde con precisión, evitando una compra —y probable devolución— basada en una suposición.

3.3 Cómo se mide el éxito

Además de la tasa de resolución y la satisfacción, en e-commerce cobran especial peso dos indicadores propios del negocio: la **tasa de devoluciones** por incompatibilidad o error de compra, que debería reducirse tras la implementación, y la **tasa de conversión** de las conversaciones con el asistente, es decir, qué porcentaje de clientes que interactúan con él terminan completando la compra. Si el catálogo cambia con frecuencia, conviene además que la sincronización entre el sistema de gestión de la tienda y la base documental del asistente sea automática, para evitar respuestas basadas en stock o precios desactualizados.

4. Medios y publicaciones digitales: hemeroteca y asistente editorial

Este es, para Intelifica, uno de los verticales más interesantes: medios que nacieron o crecieron en papel y hoy publican en digital, y que acumulan un archivo —la hemeroteca— con un valor informativo enorme y muy poco aprovechado en la práctica diaria.

4.1 Hemeroteca consultable para lectores

El sistema se alimenta de todo el archivo histórico de artículos, entrevistas, crónicas y reportajes del medio, idealmente con metadatos de fecha, autor y sección. Las preguntas de los lectores suelen ser de tipo «¿qué se ha publicado sobre X a lo largo del tiempo?» o «¿cómo ha evolucionado este asunto desde que empezó?». La diferencia frente a un buscador convencional es que el asistente no devuelve una lista de titulares sueltos, sino una respuesta redactada que sintetiza varios artículos y cita cada uno con su fecha, dejando al lector decidir si quiere abrir la pieza original.

EJEMPLO PRÁCTICO

Un lector de un diario comarcal pregunta: «¿Qué ha pasado con el proyecto de la nueva depuradora desde que se anunció?». El asistente recorre el archivo, localiza ocho artículos publicados a lo largo de tres años —el anuncio inicial, retrasos, cambios de presupuesto, la inauguración— y devuelve una cronología breve con enlaces a cada pieza. El lector obtiene en segundos lo que antes exigía buscar manualmente en el archivo, si es que sabía que ese archivo existía.

4.2 Recomendación de artículos relacionados

El mismo archivo, usado de forma distinta, permite sugerir automáticamente artículos relacionados con lo que un lector está consultando en ese momento, aumentando el tiempo de permanencia en la web y el número de páginas vistas por visita —dos indicadores muy relevantes para la publicidad digital y, en algunos casos, para las suscripciones.

4.3 Asistente editorial para la redacción

De cara al equipo de redacción, el sistema alimentado con el archivo propio del medio permite responder preguntas como «¿qué hemos publicado antes sobre esta persona o esta empresa?» o «¿tenemos ya una entrevista con esta fuente?», ayudando a evitar contradicciones con coberturas anteriores y a dar contexto rápido a un periodista que se incorpora a una noticia en marcha.

A TENER EN CUENTA

Cuando el archivo abarca muchos años y miles de piezas, y las preguntas exigen conectar información dispersa entre ellas —no solo recuperar un artículo aislado—, un sistema RAG «clásico» empieza a quedarse corto. «RAG en producción», el tercer libro de esta colección explica cuándo conviene reforzar la arquitectura con un patrón conocido como GraphRAG, pensado precisamente para este tipo de archivo documental extenso.

5. Sector legal y administrativo: normativa y contratos con cita exacta

En despachos de abogados, asesorías, gestorías y departamentos administrativos, el valor de un asistente RAG está directamente ligado a su capacidad de citar la fuente exacta —artículo, cláusula, párrafo— de cada afirmación, porque en este ámbito una respuesta sin respaldo verificable tiene poco valor práctico.

5.1 Documentos fuente

Normativa aplicable al sector del cliente, contratos tipo y sus cláusulas habituales, convenios colectivos, procedimientos internos de tramitación y, en algunos casos, jurisprudencia o resoluciones anteriores relevantes. La actualización constante de estos documentos —la normativa cambia con frecuencia— es aquí un requisito, no un extra.

5.2 Preguntas típicas y forma de trabajar

«¿Qué dice la cláusula de rescisión de nuestro contrato tipo de arrendamiento?», «¿qué plazo establece la normativa vigente para este trámite?», «¿qué cambió en la última actualización de este reglamento respecto a la anterior?». En todos los casos, la respuesta útil no es solo el contenido, sino la referencia exacta que permite a un profesional verificarlo antes de aplicarlo.

EJEMPLO PRÁCTICO

Una asesoría laboral conecta su asistente a los convenios colectivos que gestiona para sus clientes. Un cliente pregunta: «¿Cuántos días de permiso retribuido corresponden por matrimonio según nuestro convenio?». El asistente localiza el artículo exacto del convenio aplicable y responde citándolo literalmente, en lugar de dar una cifra genérica que podría no corresponder a ese convenio en particular.

5.3 Cómo se mide el éxito

Aquí la métrica de **fidelidad de la cita** —que la referencia indicada exista realmente y diga lo que el asistente afirma que dice— importa más que la velocidad. Se complementa con el tiempo que un profesional dedica a verificar manualmente una norma o cláusula antes y después de contar con el asistente.

A TENER EN CUENTA

En este sector, la supervisión humana de las respuestas sigue siendo imprescindible antes de tomar decisiones con consecuencias legales. El asistente acelera la búsqueda y la primera lectura, pero no sustituye el criterio profesional en casos con matices.

6. Formación y onboarding interno

El último caso de uso de Este capítulo afecta a cualquier empresa con empleados nuevos: la incorporación de una persona a un puesto de trabajo genera, casi siempre, una avalancha de preguntas parecidas que ya tienen respuesta en algún manual, solo que disperso y difícil de encontrar para alguien recién llegado.

6.1 Documentos fuente

Manual de bienvenida, procedimientos internos por departamento, políticas de empresa (horarios, vacaciones, herramientas), documentación de los sistemas y aplicaciones que se usan a diario, y grabaciones o transcripciones de formaciones anteriores si existen.

6.2 Preguntas típicas

«¿Cómo solicito el material de trabajo?», «¿a quién reporto si tengo un problema con X herramienta?», «¿cuál es el procedimiento para pedir vacaciones?», «¿dónde encuentro la plantilla para este informe?». Son preguntas de bajo riesgo pero alto volumen, ideales para un asistente RAG bien alimentado.

EJEMPLO PRÁCTICO

Una cadena de tiendas de electrónica con alta rotación de personal de tienda implementa un asistente de onboarding alimentado con el manual de procedimientos de caja, atención al público y gestión de incidencias. Los nuevos empleados resuelven la mayoría de sus dudas de la primera semana sin necesidad de interrumpir al encargado de turno, que antes dedicaba varias horas semanales solo a formación básica repetitiva.

6.3 Cómo se mide el éxito

Tiempo medio hasta que un empleado nuevo es autónomo, número de consultas resueltas por el asistente frente a las que requieren a un responsable, y valoración del proceso de incorporación en encuestas internas.

RAG frente a las alternativas: ventajas técnicas y de negocio

VENTAJAS

Cómo se compara RAG con el fine-tuning, el prompting simple y el contexto largo, y por qué suele ser la opción con mejor equilibrio entre coste, fiabilidad y capacidad de mantenimiento.

1. Tres formas de dar conocimiento propio a un modelo de IA

Cuando una empresa decide que quiere que un modelo de lenguaje (LLM) responda con precisión sobre su negocio —sus productos, sus procesos, su normativa interna— se enfrenta, en esencia, a tres estrategias distintas. No son mutuamente excluyentes del todo, pero sí parten de planteamientos técnicos muy diferentes, con implicaciones de coste, mantenimiento y fiabilidad que conviene entender antes de elegir.

La primera es el **fine-tuning** (ajuste fino): reentrenar los parámetros internos del modelo con ejemplos específicos de la empresa, de forma que el conocimiento quede «incorporado» dentro del propio modelo. La segunda es el **prompting simple**, es decir, confiar en las instrucciones que se dan al modelo en el momento de la consulta, sin ningún mecanismo adicional de búsqueda de información. La tercera es meter documentos completos dentro de la ventana de contexto del modelo —lo que se conoce como enfoque de **contexto largo**— aprovechando que los modelos actuales admiten entradas de texto cada vez más extensas.

RAG (Retrieval-Augmented Generation, o Generación Aumentada por Recuperación) introduce una cuarta vía que, en la práctica, se ha convertido en la opción por defecto para la mayoría de aplicaciones empresariales: en lugar de incorporar el conocimiento al modelo o depender de meterlo todo en cada consulta, RAG añade un paso previo de **recuperación**: busca automáticamente, en una base de conocimiento externa, solo los fragmentos de información relevantes para la pregunta concreta, y se los entrega al modelo junto con la instrucción de responder basándose en ellos.

> CONCEPTO CLAVE
>
> El pipeline habitual de RAG es: los documentos se dividen en fragmentos (**chunking** o troceado), cada fragmento se convierte en un vector numérico (**embedding**) que representa su significado, y esos vectores se guardan en una **base de datos vectorial**. Cuando llega una pregunta, se busca qué fragmentos son más parecidos semánticamente a esa pregunta, y esos fragmentos —no el documento entero— son los que se envían al modelo junto con la pregunta original.

Este capítulo introduce el marco de comparación; los siguientes tres desarrollan, uno por uno, cómo se posiciona RAG frente a cada una de las otras estrategias.

2. RAG frente al fine-tuning: actualizar hechos sin reentrenar

2.1 Dónde vive el conocimiento

La diferencia más importante entre RAG y fine-tuning es dónde reside la información. En el fine- tuning, el conocimiento queda codificado dentro de los propios parámetros (los «pesos») del modelo, mezclado de forma indistinguible con todo lo demás que el modelo ya sabía. En RAG, el conocimiento permanece fuera del modelo, en documentos que se pueden leer, editar y auditar de forma independiente. Esta separación es la que explica casi todas las ventajas prácticas de RAG en el día a día de una empresa.

2.2 Coste y tiempo de implementación

Poner en marcha un sistema de RAG razonablemente bueno —con una base de datos vectorial, un proceso de troceado y una integración con un modelo de lenguaje ya existente— suele poder hacerse en días o pocas semanas, reutilizando modelos disponibles sin necesidad de entrenar nada desde cero. El fine-tuning, en cambio, requiere preparar un conjunto de datos de entrenamiento de calidad (a menudo cientos o miles de ejemplos bien etiquetados), disponer de capacidad de cómputo para el entrenamiento, y evaluar cuidadosamente que el modelo ajustado no haya empeorado en otras capacidades que no se querían tocar (un efecto conocido como «olvido catastrófico» cuando se hace mal). El coste y el tiempo de un proyecto de fine-tuning suelen ser sensiblemente mayores que los de un proyecto de RAG equivalente.

2.3 Actualización de hechos que cambian con el tiempo

Aquí está la ventaja más citada de RAG frente al fine-tuning: cuando un hecho cambia —un precio, una fecha límite, una condición contractual, la disponibilidad de un producto— actualizar un documento en la base de conocimiento es prácticamente inmediato y no requiere ninguna intervención técnica especializada. Actualizar un modelo afinado por fine-tuning para que «olvide» el dato antiguo y «aprenda» el nuevo implica, casi siempre, repetir (al menos parcialmente) el proceso de entrenamiento. Por eso las notas de investigación del sector son claras en este punto: el fine-tuning tiene sentido para fijar un formato de salida, un tono de marca o un comportamiento consistente, pero no es la herramienta adecuada para «enseñar hechos nuevos» que van a seguir cambiando con el tiempo.

Factor	Fine-tuning	RAG
Dónde vive el conocimiento	Dentro de los parámetros del modelo	En documentos externos, fuera del modelo

Factor	Fine-tuning	RAG
Actualizar un hecho puntual	Requiere repetir (parte del) entrenamiento	Se edita el documento fuente
Coste y tiempo de puesta en marcha	Altos: datos de entrenamiento, cómputo, validación	Moderados: indexar documentos existentes
Mejor uso recomendado	Tono, formato, comportamiento consistente, reducir tamaño/coste del modelo	Conocimiento factual que cambia con frecuencia

EJEMPLO PRÁCTICO

Una cadena de tiendas de moda con precios y stock que cambian semanalmente evaluó ambas opciones. Con fine-tuning, cada actualización de catálogo habría implicado un nuevo ciclo de entrenamiento y validación, con el consiguiente retraso y coste recurrente. Con RAG, el equipo de e-commerce simplemente sincroniza el feed de productos con la base de conocimiento cada noche, y el asistente responde siempre con el precio y el stock del día siguiente por la mañana.

3. RAG frente al prompting simple: de la improvisación a la trazabilidad

3.1 Qué es el prompting simple sin recuperación

Muchas implementaciones improvisadas de «chatbot con IA» consisten en dar al modelo unas instrucciones generales en el mensaje de sistema —por ejemplo, «eres el asistente de la empresa X, responde de forma amable»— y confiar en que el modelo, con su conocimiento general más esas instrucciones, sea capaz de responder bien. No hay ningún paso de búsqueda de información específica de la empresa: todo depende de lo que el modelo ya sabía de su entrenamiento o de lo poco que se le haya podido incluir directamente en el

propio prompt (mensaje de instrucción).

3.2 El coste de no recuperar información

El problema de este enfoque es doble. Primero, el riesgo de alucinación es alto en cuanto se pregunta algo específico de la empresa que el modelo nunca vio durante su entrenamiento: en ausencia de datos reales, el modelo generará la respuesta estadísticamente más probable, que puede sonar bien y ser completamente incorrecta. Segundo, no hay ninguna forma de **trazabilidad**: si el modelo dice algo, no hay un documento al que apuntar como fuente, porque no ha consultado ninguno. Esto es especialmente problemático en sectores donde la respuesta puede tener implicaciones legales, contractuales o de seguridad, donde no basta con que la respuesta «suene bien»: hay que poder demostrar en qué se basa.

> **A TENER EN CUENTA**
>
> El prompting simple no es «malo» en sí mismo: es perfectamente válido para tareas generales de redacción, resumen o conversación donde no se necesita conocimiento específico y verificable de la empresa. El problema aparece cuando se usa para responder preguntas factuales sobre datos propios sin ningún mecanismo de recuperación detrás.

3.3 Qué añade RAG

RAG resuelve ambos problemas con el mismo mecanismo: al forzar al modelo a basar su respuesta en fragmentos concretos recuperados de la base de conocimiento, reduce drásticamente la probabilidad de que invente información (porque tiene delante el dato real en el momento de responder) y, además, permite mostrar exactamente qué fragmento y qué documento sustentan la respuesta. Esa citación de fuentes no es un añadido cosmético: es la diferencia entre un sistema que se puede auditar y uno que no.

4. RAG frente al contexto largo: escalar sin disparar el coste

4.1 Por qué el contexto largo parece una alternativa tentadora

Los modelos de lenguaje actuales admiten ventanas de contexto cada vez más grandes: es técnicamente posible pegar decenas o cientos de páginas de documentación directamente en el mensaje que se envía al modelo, sin ningún paso previo de búsqueda. Para un corpus de conocimiento pequeño y que apenas cambia —por ejemplo, un único manual de producto de veinte páginas— este enfoque puede funcionar razonablemente bien y evita la complejidad de montar una base de datos vectorial.

4.2 Dónde deja de escalar

El problema aparece en cuanto la base de conocimiento crece más allá de ese caso pequeño y estático. Cada consulta al modelo con contexto largo implica procesar (y pagar) todo ese texto de nuevo, aunque solo una pequeña parte sea relevante para la pregunta concreta. Esto tiene dos efectos directos: el **coste por consulta** aumenta de forma proporcional al tamaño del contexto que se envía, y la **latencia** (el tiempo que tarda el modelo en responder) también crece, porque hay más texto que procesar antes de empezar a generar la respuesta. Si la base de conocimiento de la empresa tiene cientos de documentos —catálogos completos, manuales de varias líneas de producto, años de políticas internas—, meterlo todo en cada consulta deja de ser viable tanto en coste como en tiempo de respuesta.

Factor	Contexto largo (todo en el prompt)	RAG (recuperación selectiva)
Coste por consulta	Crece con el tamaño total del corpus	Depende solo de los fragmentos recuperados (unos pocos)

Latencia	Aumenta con más texto que procesar	Se mantiene más estable al procesar solo lo relevante
Escala con corpus grandes	Limitada; se vuelve poco práctico	Diseñado para crecer con la base de conocimiento
Trazabilidad de fuentes	Difícil de precisar qué parte del texto se usó	Se puede señalar el fragmento exacto recuperado

Además, incluso cuando el coste es asumible, un modelo con una ventana de contexto muy llena no siempre «presta atención» de forma uniforme a todo el texto: la evidencia práctica en el sector muestra que la información situada en según qué posiciones del contexto puede recibir menos peso que la más cercana al principio o al final. RAG evita este problema seleccionando de antemano solo la información relevante, en vez de confiar en que el modelo sepa encontrarla dentro de un texto muy largo.

EJEMPLO PRÁCTICO

Una empresa de formación online con un catálogo de 400 cursos evaluó meter toda la documentación de cursos en el contexto de cada consulta. El coste por pregunta resultó desproporcionado y las respuestas tardaban varios segundos de más. Al pasar a RAG, cada pregunta recupera únicamente las fichas de los 3 o 4 cursos más relevantes, y tanto el coste como el tiempo de respuesta bajaron de forma notable, sin perder precisión.

5. Actualizar el conocimiento cambiando solo los documentos fuente

Un beneficio transversal a las tres comparaciones anteriores merece un capítulo propio, porque es probablemente la ventaja operativa más apreciada por los equipos que ya usan RAG en producción: el ciclo de actualización del conocimiento se convierte en un proceso de **gestión documental**, no en un proyecto técnico.

En la práctica, esto significa que el flujo de trabajo para mantener el asistente al día se parece más a gestionar una carpeta compartida que a un proyecto de ingeniería de datos: se sustituye un PDF de condiciones comerciales por su versión revisada, se añade la ficha de un producto nuevo, se retira la documentación de un servicio descontinuado. El sistema de RAG detecta el cambio en la siguiente sincronización, vuelve a trocear y a indexar el documento actualizado, y a partir de ese momento las respuestas reflejan la versión más reciente.

CONCEPTO CLAVE

Este proceso de volver a trocear e indexar un documento cuando cambia se llama habitualmente **reindexación**. Cuanto más automatizado esté este proceso —por ejemplo, conectado directamente a la carpeta o al sistema donde ya se gestionan los documentos de la empresa—, menos fricción operativa tiene mantener el asistente actualizado.

Esta ventaja tiene un efecto organizativo interesante: descentraliza el mantenimiento del conocimiento de la IA. No hace falta que un equipo técnico intervenga cada vez que cambia un dato del negocio; puede hacerlo la misma persona que ya gestiona esa documentación — marketing actualizando fichas de producto, atención al cliente revisando las preguntas frecuentes, el departamento legal sustituyendo una plantilla de contrato—. Esto reduce cuellos de botella y acorta el tiempo entre que algo cambia en el negocio y ese cambio se refleja en las respuestas del asistente.

6. Cuándo RAG no basta por sí solo

Sería poco honesto presentar RAG como la solución universal sin matices. Existen situaciones donde conviene combinarlo con otras técnicas o donde otra alternativa puede ser más adecuada, y conocerlas ayuda a tomar mejores decisiones de negocio.

Formato o tono de marca muy específico: si lo que se necesita no es tanto «saber datos» sino «hablar siempre con un estilo determinado» de forma muy consistente, un ajuste fino ligero puede complementar bien a RAG.

Corpus muy pequeño y prácticamente estático: si la base de conocimiento es de pocas páginas y apenas cambia, un enfoque de contexto largo puede ser suficiente y evitar la complejidad de montar infraestructura de recuperación.

Documentación de baja calidad: RAG no corrige documentos desactualizados, contradictorios o mal redactados; simplemente los recupera tal cual. Ninguna técnica de IA sustituye la necesidad de mantener una base documental fiable.

Necesidad de razonamiento complejo entre muchos documentos a la vez: preguntas que exigen combinar información dispersa en decenas de fuentes distintas pueden requerir arquitecturas de RAG más avanzadas (tratadas en «RAG en producción», el tercer libro de esta colección) en lugar del enfoque más básico.

> **A TENER EN CUENTA**
>
> Elegir RAG no es una decisión de «todo o nada» frente a fine-tuning o contexto largo. En proyectos maduros es habitual encontrar combinaciones: RAG como columna vertebral para el conocimiento factual, con algún ajuste fino puntual para el estilo de respuesta, y contexto largo reservado para documentos concretos que conviene analizar de forma completa en una consulta específica.

Problemas, límites y retos de RAG

PROBLEMAS

Un recorrido por las causas concretas que hacen fallar a un sistema RAG en producción —troceado, calidad del dato, recuperación, mantenimiento y seguridad— y las prácticas contrastadas para mitigar cada una.

1. Chunking mal hecho: cuando el troceado corta la información por la mitad

Un sistema RAG combina varios eslabones —troceado de documentos, generación de vectores, búsqueda, ensamblado del contexto y redacción de la respuesta— y un fallo en cualquiera de ellos degrada el resultado final, por bueno que sea el modelo de lenguaje empleado. De hecho, la calidad de un sistema RAG está limitada, casi siempre, por la calidad de la recuperación, no por la capacidad del modelo que redacta la respuesta. Este capítulo recorre las causas técnicas más habituales de los fallos en producción —empezando por el troceado— y las buenas prácticas que las mitigan.

> CONCEPTO CLAVE
>
> El «pipeline» de un RAG clásico: los documentos se trocean en fragmentos (chunks), cada fragmento se convierte en un vector numérico (embedding) que representa su significado, esos vectores se guardan en una base de datos vectorial y, al llegar una pregunta, se buscan los fragmentos más parecidos para insertarlos junto a ella en el prompt del modelo de lenguaje.

Los documentos largos no se cargan «enteros»: se dividen en fragmentos llamados **chunks**, porque la búsqueda compara fragmentos concretos con la pregunta del usuario, y porque el modelo solo puede recibir una cantidad limitada de texto por respuesta. Esta división, aparentemente mecánica, es una de las decisiones que más influye en la calidad final, y hacerla mal es una causa muy frecuente de respuestas incompletas o descontextualizadas.

1.1 Qué puede salir mal

Fragmentos demasiado pequeños: pierden el contexto necesario para entenderse por sí solos. Un párrafo que empieza con «esto también aplica en los siguientes casos» pierde todo su sentido si se separa del párrafo anterior que explica a qué se refiere «esto».

Fragmentos demasiado grandes: mezclan varios temas en un mismo chunk y diluyen su relevancia en la búsqueda, porque un fragmento con demasiados temas no se «parece» mucho a ninguna pregunta concreta.

Cortes en el lugar equivocado: una división por tamaño fijo (por ejemplo, cada 1.000 caracteres, sin mirar el contenido) puede partir una tabla o un procedimiento paso a paso justo por la mitad.

> EJEMPLO PRÁCTICO
>
> Un despacho de gestoría documenta el alta de autónomos en cinco pasos numerados. Si el troceado por tamaño fijo corta el documento entre el paso 3 y el 4, el fragmento recuperado para «cómo darme de alta» solo contendrá los tres primeros pasos, y el asistente responderá de forma incompleta sin que nadie lo note de inmediato.

1.2 Buenas prácticas de mitigación

División recursiva por estructura: en lugar de cortar por un número fijo de caracteres, se respeta la jerarquía natural del documento —párrafos, luego frases— para evitar cortes a mitad de una idea. Es la opción de referencia en la mayoría de proyectos.

Troceado jerárquico (parent-child): se generan fragmentos pequeños y precisos para la búsqueda, pero al resultar relevante uno de ellos se entrega al modelo el fragmento «padre» más amplio que lo contiene, para no perder contexto.

Troceado semántico: se mide dónde cambia realmente de tema el texto y se corta ahí, produciendo fragmentos temáticamente coherentes.

Para documentos con estructura visual clara (contratos paginados, catálogos), el **troceado por página** suele ser la opción más simple y efectiva.

> **A TENER EN CUENTA**
>
> No existe una estrategia de troceado universalmente óptima: depende del tipo de documento. Un buen proveedor prueba varias estrategias con una muestra de preguntas reales antes de fijar la definitiva.

2. Datos de origen: calidad, contradicciones y actualidad

Incluso con un troceado perfecto, un sistema RAG no puede dar buenas respuestas si los documentos de partida son de mala calidad. Este es el problema conocido en el sector como «basura entra, basura sale» (garbage in, garbage out): el sistema no tiene forma de distinguir un documento fiable de uno obsoleto, ni de detectar por sí mismo una contradicción entre dos fuentes, salvo que se le añadan mecanismos específicos para ello.

2.1 Los tres problemas de origen más comunes

Problema	Efecto en las respuestas
Documentos desactualizados que conviven con los vigentes	El sistema puede citar precios, plazos o condiciones que ya no aplican.
Información contradictoria entre documentos	Respuestas inconsistentes según qué fragmento recupere cada vez, incluso para la misma pregunta repetida.
Documentos mal estructurados (sin títulos, escaneados sin texto legible, formato inconsistente)	Peor calidad de troceado y de búsqueda, porque el sistema no puede apoyarse en la estructura para entender el contenido.

2.2 Buenas prácticas de mitigación

Realizar una **auditoría inicial de la documentación** antes de cargarla al sistema: identificar duplicados, versiones obsoletas y contradicciones evidentes.

Establecer una **única fuente de verdad** por tema (por ejemplo, un solo documento «vivo» de política de devoluciones, en lugar de varias versiones dispersas en distintas carpetas).

Etiquetar los documentos con metadatos de fecha y estado (vigente / obsoleto), de modo que se puedan filtrar o dar prioridad a los más recientes.

Convertir documentos escaneados o en formatos poco estructurados a texto legible antes de indexarlos (mediante OCR y limpieza de formato), en lugar de cargarlos «tal cual».

> **A TENER EN CUENTA**

La preparación de los datos de origen suele ser la fase que más tiempo consume en un proyecto RAG bien ejecutado. Conviene presupuestarla como tal desde el inicio.

3. Recuperación pobre: cuando el sistema no encuentra el fragmento correcto

Aunque los documentos sean buenos y el troceado esté bien hecho, puede fallar el paso intermedio: la **recuperación**, es decir, el mecanismo que decide qué fragmentos son relevantes para una pregunta concreta. La mayoría de sistemas RAG «naive» (básicos) recuperan información únicamente por similitud semántica —comparando el significado general de la pregunta con el significado general de cada fragmento—, y esa aproximación tiene puntos ciegos conocidos.

3.1 Cuándo falla la búsqueda semántica pura

La búsqueda semántica es muy buena entendiendo la intención general de una pregunta, pero pierde precisión con elementos muy específicos: códigos de referencia, números de producto, nombres propios poco frecuentes o términos técnicos exactos. Un vector que representa el «significado» de una pregunta como «¿qué cubre la garantía del modelo XR-450?» puede acabar más cerca de un fragmento sobre garantías en general que de la ficha exacta del modelo XR-450, porque el modelo de embeddings no le da tanto peso a ese código concreto como le daría una búsqueda por palabras clave.

EJEMPLO PRÁCTICO

Una empresa de electrodomésticos recibe con frecuencia preguntas sobre códigos de error concretos («error E4 en el modelo X»). Con una búsqueda puramente semántica, el sistema recupera a veces el manual general de la marca en lugar de la tabla específica de códigos de error, porque el vector de la pregunta se «parece» más al contenido general que al fragmento exacto con el código E4.

3.2 Buenas prácticas de mitigación

Búsqueda híbrida: combinar la búsqueda semántica (por significado) con la búsqueda léxica tradicional (por coincidencia de palabras clave, tipo BM25), fusionando ambos resultados. Es la mejora con mejor relación esfuerzo/beneficio para resolver justamente el problema de códigos, referencias y nombres propios.

Reordenación de resultados (reranking): recuperar un conjunto más amplio de candidatos y usar un segundo modelo especializado en evaluar la relevancia real de cada uno antes de pasarlos al modelo de lenguaje final, mejorando notablemente la precisión con un coste de latencia moderado.

Reescritura de la consulta: cuando la pregunta del usuario es ambigua o está mal formulada, un paso previo puede reformularla o descomponerla en subpreguntas más precisas antes de lanzar la búsqueda.

Medir de forma continua qué proporción de preguntas recupera el fragmento correcto (lo que en el nivel Avanzado de esta colección se detalla como métricas de recuperación), para detectar este problema antes de que lo note el cliente final.

A TENER EN CUENTA

La recuperación pobre es traicionera: el sistema da una respuesta bien redactada, pero basada en el documento equivocado. Conviene revisar de vez en cuando no solo si la respuesta «suena bien», sino si cita el fragmento correcto.

4. Mantenimiento continuo de la base de conocimiento

Un sistema RAG no es un proyecto que se entrega y queda terminado, sino un servicio vivo que necesita mantenimiento igual que cualquier otro sistema de información de la empresa. Sin este mantenimiento, incluso el sistema mejor diseñado en su lanzamiento se degrada con el tiempo, porque la base de conocimiento deja de reflejar la realidad del negocio.

4.1 Qué implica el mantenimiento en la práctica

Actualización: incorporar nuevos documentos y nuevas versiones cuando cambia algo relevante (precios, políticas, catálogo, normativa aplicable).

Depuración (curation): retirar documentos obsoletos o duplicados en lugar de dejarlos acumulados junto a los vigentes, lo que evita que el sistema los recupere por error.

Reindexación: tras cambios importantes en la documentación, es necesario volver a procesar (trocear y vectorizar) los documentos afectados para que la búsqueda refleje el estado actual.

Revisión del comportamiento real: analizar periódicamente una muestra de preguntas y respuestas reales para detectar temas mal cubiertos, respuestas pobres recurrentes o documentación que falta.

CONCEPTO CLAVE

El «ciclo de vida de la base de conocimiento» es el proceso continuo de añadir, actualizar, depurar y reindexar los documentos que alimentan al sistema. No es una tarea puntual del proyecto de implantación, sino una responsabilidad operativa permanente, normalmente compartida entre el proveedor tecnológico y el equipo interno de la empresa.

A TENER EN CUENTA

Un error de planificación habitual es no asignar de antemano quién es responsable de mantener la base de conocimiento actualizada dentro de la empresa cliente. Sin un responsable claro, el mantenimiento simplemente no ocurre, y el sistema se degrada de forma silenciosa hasta que los usuarios empiezan a notar respuestas desactualizadas.

5. Seguridad básica: quién puede ver qué documentos

Un aspecto que se pasa por alto con frecuencia en proyectos pequeños es el control de quién puede acceder a qué información a través del asistente. Si toda la documentación se carga en un único espacio sin diferenciación de permisos, cualquier usuario que hable con el asistente puede llegar a recibir, sin quererlo, información que no debería ver —por ejemplo, condiciones comerciales especiales para grandes clientes, datos internos de nómina o documentos legales confidenciales.

5.1 El riesgo en términos sencillos

Un sistema RAG básico no distingue, salvo que se le configure explícitamente, entre un documento público (como una política de devoluciones para clientes) y uno interno (como un acuerdo de condiciones especiales con un proveedor). Si ambos están en la misma base de conocimiento sin ningún control adicional, el asistente puede citar el segundo al responder a una pregunta aparentemente inocente, filtrando información que debería quedar restringida a determinadas personas.

EJEMPLO PRÁCTICO

Una empresa de servicios B2B usa el mismo asistente tanto para atender a clientes externos como para el soporte interno de su equipo comercial, y carga en la misma base de conocimiento tanto el catálogo público como las tarifas negociadas confidencialmente con cada cliente grande. Sin una separación de permisos, un cliente que pregunte por precios podría recibir, por error de recuperación, una referencia a la tarifa especial de otro cliente.

5.2 Buenas prácticas básicas de mitigación

Separar la documentación en distintos espacios o «colecciones» según su nivel de sensibilidad (pública, interna, confidencial), en lugar de una sola bolsa común.

Configurar el sistema para que cada usuario (o cada tipo de usuario) solo pueda recuperar información de las colecciones a las que tiene permiso, antes de generar cualquier respuesta.

Evitar cargar en la base de conocimiento del asistente de cara al cliente documentos que contienen datos personales sensibles o información estrictamente interna, salvo que exista un control de acceso robusto.

Revisar periódicamente qué documentos están cargados y con qué nivel de visibilidad, especialmente después de campañas de carga masiva de documentación.

A TENER EN CUENTA

El control de acceso granular a nivel de documento —quién puede ver exactamente qué fragmento, no solo qué colección general— es un tema con más profundidad técnica, que se trata en detalle en «RAG en producción», el tercer libro de esta colección. Para la mayoría de pymes, el primer paso razonable es simplemente separar con claridad lo público de lo interno.

Tendencias en RAG: agentic RAG, multimodalidad y GraphRAG

FUTURO

Cómo están evolucionando los sistemas RAG en 2026 y qué preguntas conviene hacerle a un proveedor para saber si su tecnología está realmente al día.

1. De la búsqueda de un paso a los sistemas que deciden

El pipeline clásico de RAG —trocear documentos, convertirlos en vectores, buscar los fragmentos más parecidos a la pregunta y pasárselos al modelo— sigue siendo la base de casi todo, pero cada vez con más frecuencia deja de ser un proceso «de un solo paso». La tendencia más consolidada de 2026 es lo que en el sector se llama **agentic RAG**, o RAG agéntico: el sistema ya no se limita a ejecutar siempre la misma secuencia fija, sino que decide, consulta por consulta, qué necesita buscar y cuándo dar por buena la información recogida.

> CONCEPTO CLAVE
>
> En RAG agéntico, el modelo actúa como un pequeño «planificador»: descompone una pregunta compleja en subpreguntas, decide qué fuente consultar para cada una —a veces en paralelo—, evalúa si la respuesta que puede construir está bien fundamentada y, si no lo está, repite el ciclo antes de responder.

La diferencia con el RAG clásico no es solo de matiz. Un sistema de un solo paso recupera un conjunto de fragmentos y genera la respuesta con lo que tenga, aunque sea insuficiente. Un sistema agéntico puede, por ejemplo, notar que la pregunta mezcla una condición contractual con un dato de catálogo, buscar cada pieza por separado en índices distintos, y solo entonces construir una respuesta conjunta y verificada.

> EJEMPLO PRÁCTICO
>
> Un despacho de gestoría recibe: «¿puedo aplazar el pago del IVA del tercer trimestre si ya tengo un aplazamiento en curso de otro impuesto?». Un RAG agéntico separa la pregunta en dos búsquedas —normativa sobre aplazamientos y compatibilidad entre aplazamientos simultáneos—, contrasta ambos resultados y compone una respuesta que cubre el matiz real de la pregunta, algo que un sistema de un solo paso probablemente pasaría por alto.

Un benchmark citado con frecuencia en 2026 sitúa en torno a un 62% la reducción de alucinaciones cuando se combina RAG agéntico con una base de conocimiento estructurada en forma de grafo (lo que veremos en el capítulo 5). La cifra concreta varía según el estudio y el dominio, pero la dirección es consistente: descomponer y verificar reduce errores frente a responder de un tirón.

2. RAG multimodal: texto, imagen, tabla y vídeo en un mismo asistente

Durante años, «recuperación de información» significó casi exclusivamente «recuperación de texto». Los documentos con imágenes se indexaban solo por su texto circundante, y las tablas complejas se aplanaban a un formato que perdía buena parte de su estructura. El **RAG multimodal** rompe esa limitación: recupera y razona sobre imágenes, tablas y, cada vez más, fragmentos de vídeo y audio, dentro del mismo flujo de trabajo que el texto.

Lo que lo hace posible son los **modelos de embeddings de espacio compartido**: en lugar de tener un modelo para vectorizar texto y otro completamente distinto para vectorizar imágenes, estos modelos —como Cohere Embed v4, referencia habitual en este terreno— generan vectores para texto e imágenes dentro del mismo espacio matemático, de forma que se pueden comparar directamente una pregunta en texto con una imagen del corpus, o una tabla escaneada con un párrafo explicativo.

CONCEPTO CLAVE

«Espacio compartido» significa que el sistema puede calcular qué tan parecidos son, por ejemplo, la frase «avería en el motor» y una fotografía real de esa avería, aunque uno sea texto y el otro una imagen: ambos se representan como vectores en el mismo espacio, comparables entre sí.

EJEMPLO PRÁCTICO

Un fabricante de maquinaria agrícola indexa sus manuales técnicos junto con fotografías de piezas y despieces en PDF. Un cliente pregunta «¿cómo se llama esta pieza?» adjuntando una foto: el sistema compara la imagen directamente con el catálogo visual indexado y devuelve la ficha técnica correspondiente, sin que nadie haya tenido que describir esa pieza en texto de antemano.

Las tablas merecen mención aparte porque son uno de los formatos donde más se nota la mejora: los sistemas actuales son bastante más fiables preservando la relación entre filas y columnas al indexar tarifas, cuadros de compatibilidad o fichas técnicas con muchos parámetros, un punto débil histórico del RAG basado solo en texto plano.

3. Evaluación continua: medir la calidad en producción

Durante mucho tiempo, «evaluar» un sistema RAG significaba hacerle una tanda de pruebas antes de lanzarlo y, si los resultados eran razonables, darlo por bueno. Esa práctica está quedando obsoleta. La tendencia consolidada es tratar la evaluación como un proceso continuo, integrado en la operación diaria del sistema, no como un examen puntual de aprobado o suspenso.

El motivo es sencillo: un sistema RAG puede degradarse silenciosamente con el tiempo, por ejemplo cuando el corpus crece de forma desordenada, cuando cambian los patrones de preguntas de los usuarios, o cuando una actualización del modelo subyacente altera sutilmente su comportamiento. Sin monitorización continua, esa degradación puede pasar desapercibida durante semanas.

Capa de evaluación	Qué mide	Ejemplo de métrica
Recuperación	¿Encuentra los fragmentos correctos?	Precision@k, Recall@k, MRR
Generación	¿La respuesta se apoya en lo recuperado?	Fidelidad (faithfulness), tasa de alucinación
Extremo a extremo	¿Funciona bien en su conjunto?	Latencia, coste, tasa de rechazo, cumplimiento

CONCEPTO CLAVE

Un conjunto de prueba «dorado» es una colección fija de preguntas y respuestas correctas conocidas, usada para comparar el rendimiento del sistema a lo largo del tiempo, igual que un examen tipo que se repite cada cierto tiempo para ver si el nivel sube o baja.

Herramientas como **Ragas** se han convertido en una referencia habitual para automatizar buena parte de esta evaluación, combinando métricas de recuperación y de generación y permitiendo generar preguntas de prueba sintéticas. Cada vez es más frecuente que estas evaluaciones se ejecuten de forma recurrente sobre tráfico real, no solo sobre un conjunto de prueba fijo en un laboratorio.

4. Chunking más inteligente: troceado tardío y jerárquico

El troceado de documentos (chunking) —decidir cómo se cortan los textos largos en fragmentos manejables antes de convertirlos en vectores— parece un detalle técnico menor, pero en la práctica es uno de los factores que más influye en la calidad final de un sistema RAG. Cortar mal un documento puede dejar un fragmento sin contexto suficiente para ser útil, o mezclar dos ideas que deberían tratarse por separado.

4.1 Troceado jerárquico: precisión y contexto a la vez

Una de las mejoras más extendidas es el **troceado jerárquico**: se mantienen fragmentos «hijo» pequeños, muy precisos, para la búsqueda inicial, y fragmentos «padre» más grandes que aportan el contexto necesario para que el modelo entienda bien lo que ha encontrado. Así se resuelve un problema muy habitual: el fragmento recuperado es el correcto, pero es demasiado corto para interpretarse sin más contexto.

4.2 Troceado tardío (late chunking)

Otra técnica que está ganando terreno es el **troceado tardío**: en lugar de trocear el documento primero y vectorizar cada trozo por separado —perdiendo las referencias cruzadas entre secciones lejanas—, se procesa primero el documento completo con un modelo de contexto largo y solo después se extraen los vectores de cada fragmento a partir de esa representación global. El resultado conserva mejor relaciones como «esto se explicó en la introducción y se retoma tres páginas después».

> **A TENER EN CUENTA**
>
> Estas técnicas más avanzadas exigen modelos de embeddings preparados para contextos largos y añaden cierto coste de procesamiento en la fase de indexación. No siempre compensan: para un catálogo de fichas cortas y bien estructuradas, un troceado recursivo sencillo suele seguir siendo suficiente.

5. GraphRAG: conectar información dispersa en grandes corpus

El RAG clásico, incluso con búsqueda híbrida y reranking, tiene un punto débil: le cuesta responder preguntas que exigen «unir puntos» repartidos por decenas de documentos distintos. Preguntas del tipo «¿qué proveedores compartimos con el cliente X y en qué proyectos coincidieron?» no tienen una respuesta en un único fragmento; requieren conectar piezas sueltas de información.

GraphRAG aborda justamente ese problema: en lugar de (o además de) indexar fragmentos de texto, construye un grafo de conocimiento formado por entidades (personas, empresas, productos, conceptos) y las relaciones entre ellas, extraídas del propio corpus mediante un modelo de lenguaje. Sobre ese grafo se pueden detectar «comunidades» de información relacionada, lo que permite responder preguntas de visión de conjunto que el RAG basado solo en similitud de texto no resuelve bien.

> **CONCEPTO CLAVE**
>
> Piensa en GraphRAG como en pasar de una biblioteca donde solo puedes buscar libros por palabras del título, a una biblioteca donde además existe un mapa que conecta unos libros con otros por autor, tema o época, permitiendo preguntas de conjunto que un simple buscador de palabras no puede responder.

EJEMPLO PRÁCTICO

Un medio digital con veinte años de hemeroteca quiere responder preguntas como «¿qué políticos aparecen mencionados junto a esta empresa a lo largo de los años?». Un RAG clásico devolvería artículos sueltos que mencionen ambos términos; un sistema con GraphRAG puede recorrer la red de menciones y relaciones extraídas de todo el archivo y ofrecer una respuesta de conjunto, con las fuentes concretas que la sustentan.

A TENER EN CUENTA

GraphRAG tiene un coste de indexación notablemente más alto que el RAG clásico, porque construir el grafo exige procesar todo el corpus con un modelo de lenguaje. Para preguntas simples y puntuales suele ser una complejidad innecesaria; su valor aparece con corpus grandes y preguntas que requieren conectar información dispersa.

6. Preguntas que hacerle a tu proveedor de RAG

Con el panorama anterior ya tienes elementos suficientes para valorar si un proveedor de soluciones RAG está realmente al día o si te está ofreciendo la tecnología de hace tres años bajo un nombre comercial nuevo. No hace falta que entiendas los detalles técnicos de cada respuesta: basta con que la persona que te atiende sepa explicártelos con naturalidad.

¿El sistema puede descomponer una pregunta en varias búsquedas, o siempre hace una única búsqueda por consulta? Es la pregunta clave para saber si hay algo de RAG agéntico detrás.

¿Puede trabajar con imágenes y tablas, o solo con texto? Si tu negocio depende de catálogos visuales o tarifas complejas, esto es determinante.

¿Cómo se mide la calidad del sistema una vez está en producción, y con qué frecuencia? Una respuesta vaga («lo probamos antes de lanzarlo y ya») es una señal de alarma.

¿Qué pasa cuando el sistema no tiene información suficiente para responder con seguridad? Debería poder reconocerlo, no inventar una respuesta plausible.

¿Cómo se protege el corpus frente a documentos manipulados o desactualizados? Pregunta especialmente relevante si varias personas pueden subir contenido a la base de conocimiento.

¿El proveedor puede explicarte, con ejemplos, un caso en el que el sistema decidió no responder o pidió más información en vez de arriesgarse? Es una buena señal de que existe algún mecanismo de autorrevisión real.

A TENER EN CUENTA

No todas estas capacidades son necesarias para todos los negocios. Un pequeño comercio con un catálogo sencillo puede no necesitar GraphRAG ni multimodalidad avanzada. Lo importante es que el proveedor sepa explicarte con honestidad qué tiene y qué no, y por qué eso es o no relevante para tu caso concreto.

Glosario

TODOS LOS TÉRMINOS DEL LIBRO

Agentic RAG (RAG agéntico) Variante de RAG en la que el sistema descompone la consulta, decide qué y cuándo recuperar, evalúa lo obtenido y repite el ciclo si es necesario, en vez de seguir siempre la misma secuencia fija.

Aumento del prompt Proceso de incorporar los fragmentos recuperados junto con la pregunta original en la instrucción que recibe el modelo de lenguaje, para que genere una respuesta basada en esa información.

Base de datos vectorial Sistema de almacenamiento especializado en guardar vectores (embeddings) y encontrar rápidamente los más similares a uno dado, aunque el conjunto de datos sea muy grande.

Base documental / base de conocimiento Conjunto de documentos que un sistema RAG utiliza como fuente de información para responder: catálogos, manuales, contratos, archivo histórico, normativa, etc.

BM25 Algoritmo clásico de búsqueda por palabras clave (léxica), usado como componente de la búsqueda híbrida junto con la búsqueda semántica.

Búsqueda híbrida Técnica de recuperación que combina la búsqueda semántica (por significado) con la búsqueda léxica clásica (por coincidencia de palabras), fusionando ambos resultados para obtener lo mejor de los dos enfoques.

Chunking (troceado) Proceso de dividir los documentos en fragmentos más pequeños (chunks) antes de indexarlos, para que el sistema de búsqueda pueda trabajar con unidades manejables de información.

Contexto largo Capacidad de un modelo de lenguaje para aceptar instrucciones muy extensas, lo que permite incluir documentos completos directamente en el prompt en lugar de recuperar fragmentos.

Contexto largo / ventana de contexto Cantidad de texto que un modelo puede procesar en una sola consulta. El enfoque de «contexto largo» consiste en incluir documentos completos directamente en esa ventana en lugar de recuperar solo fragmentos relevantes.

Control de acceso Conjunto de reglas que determinan qué usuarios pueden ver qué documentos o fragmentos concretos a través del asistente.

Corrective RAG Patrón que evalúa la relevancia de los fragmentos recuperados y, si no son adecuados, relanza la búsqueda o recurre a fuentes adicionales antes de responder.

Cross-encoder Tipo de modelo usado habitualmente para reranking, que analiza la pregunta y cada candidato de forma conjunta (no por separado), lo que le permite emitir un juicio de relevancia mucho más fino que un simple cálculo de cercanía en el mapa de significados.

Embedding Representación numérica (un vector) del significado de un fragmento de texto, que permite comparar automáticamente qué tan parecidos son dos textos en cuanto a su contenido.

Embeddings de espacio compartido Modelos que representan contenidos de distinto tipo (texto, imagen) como vectores comparables entre sí dentro de un mismo espacio matemático.

Evaluación continua Práctica de medir de forma recurrente, sobre tráfico real en producción, la calidad de un sistema RAG, en lugar de evaluarlo solo antes del lanzamiento.

Fidelidad de la cita Grado en que la referencia a un documento que ofrece el asistente (artículo, cláusula, fecha) existe realmente y respalda con exactitud lo que se afirma.

Fine-tuning (ajuste fino) Proceso de reentrenar los parámetros de un modelo de lenguaje con datos específicos, incorporando el conocimiento dentro del propio modelo en vez de mantenerlo externo.

Framework de orquestación Conjunto de herramientas de software (como LangChain, LangGraph o LlamaIndex) que facilita construir el «director de orquesta» de un sistema RAG, sin tener que programar cada conexión y cada paso desde cero.

Fuente de verdad (single source of truth) Documento único y actualizado que se considera la referencia oficial sobre un tema concreto, para evitar contradicciones entre varias versiones dispersas.

Grafo de agentes Forma de representar un proceso con posibles bucles, condiciones y repeticiones —en lugar de una secuencia fija de pasos—, propia de sistemas RAG que necesitan tomar pequeñas decisiones sobre la marcha, como LangGraph.

GraphRAG Patrón que construye un grafo de conocimiento (entidades y relaciones extraídas de los documentos) para responder preguntas que exigen conectar información dispersa en muchos documentos.

Hemeroteca Archivo histórico de publicaciones de un medio de comunicación.

HyDE (Hypothetical Document Embeddings) Técnica de transformación de consulta que genera una respuesta hipotética a la pregunta y busca con el vector de esa respuesta, en lugar de con el vector de la pregunta original.

Latencia Tiempo que transcurre entre que un usuario hace una pregunta y recibe la respuesta completa del sistema.

Modelo de embeddings Sistema de inteligencia artificial especializado en convertir un texto (o una imagen) en su huella numérica de significado; distintos proveedores (OpenAI, Cohere, Jina, o modelos de código abierto como BGE o E5) ofrecen resultados de distinta calidad, coste e idoneidad según el idioma y el tipo de contenido.

Onboarding Proceso de incorporación de un nuevo empleado a la empresa, incluyendo la formación inicial y la resolución de sus primeras dudas.

pgvector Extensión de la base de datos PostgreSQL que añade capacidad de búsqueda vectorial, permitiendo guardar datos relacionales y vectores de significado en un mismo sistema.

Prompting simple Enfoque en el que el modelo responde solo con instrucciones dadas en el mensaje de consulta, sin ningún mecanismo de búsqueda de información adicional.

RAG (Generación Aumentada por Recuperación) Arquitectura que añade un paso de búsqueda de información en una base de conocimiento externa antes de que el modelo de lenguaje genere su respuesta.

RAG multimodal Capacidad de recuperar y razonar sobre información en varios formatos —texto, imagen, tabla, vídeo, audio— dentro de un mismo sistema, gracias a modelos de embeddings de espacio compartido.

Reciprocal Rank Fusion Técnica para combinar los resultados de dos o más búsquedas distintas en una sola lista ordenada, basada en la posición de cada resultado en cada lista original.

Recuperación (retrieval) Etapa del proceso RAG en la que el sistema busca y selecciona los fragmentos de información más relevantes para responder a una pregunta concreta.

Reindexación Proceso de volver a trocear y actualizar en la base de datos vectorial un documento que ha cambiado, para que las siguientes consultas reflejen la versión más reciente.

Reranking (reordenamiento) Segundo paso de filtrado, más lento pero más preciso que la búsqueda inicial, que examina un grupo reducido de candidatos recuperados y los reordena según su relevancia real frente a la pregunta original.

Self-RAG Patrón que, además de evaluar la recuperación, evalúa si la respuesta generada está bien fundamentada en los fragmentos usados, y puede abstenerse de responder si no lo está.

Tasa de escalado Porcentaje de consultas que el asistente deriva correctamente a un humano por no disponer de una respuesta fiable.

Tasa de resolución Porcentaje de consultas que el asistente resuelve sin necesidad de que intervenga una persona.

Trazabilidad Capacidad de identificar y verificar el documento y fragmento concretos en los que se apoya una respuesta generada por IA.

Troceado jerárquico Estrategia de chunking que combina fragmentos pequeños para la búsqueda precisa con fragmentos más grandes que aportan contexto al modelo.

Troceado tardío (late chunking) Técnica de chunking que procesa el documento completo antes de generar los vectores de cada fragmento, preservando mejor las referencias entre partes distantes del texto.

Para seguir

Con lo visto aquí ya se puede evaluar una propuesta técnica: sabes qué preguntas hacer sobre el troceado, qué implica elegir una base vectorial u otra, y qué métricas pedir para comprobar que el sistema responde como dicen que responde.

El paso siguiente suele ser medir sobre contenido propio, no sobre demostraciones. Un sistema que funciona muy bien con documentación de ejemplo puede comportarse de otra forma con la tuya.

«RAG en producción» entra en la arquitectura y en lo que cuesta sostenerla.