Facsímil 04

La caja de herramientas

Capítulo 11

Agentic RAG y GraphRAG: cuándo complicar

686f6c61 · 2026

Capítulo 11: Agentic RAG y GraphRAG: cuándo complicar

Cuando un RAG fijo se queda corto

En el capítulo 09 construimos el RAG básico: partir una colección, buscar fragmentos, pasarlos al modelo y responder con citas. En el capítulo 10 aprendimos a medir si la evidencia aparece, si el contexto sostiene la respuesta y si el sistema sabe abstenerse.

Ese patrón funciona muy bien para preguntas directas: “¿qué dice esta normativa sobre X?”, “¿cuál es el plazo?”, “¿dónde se configura este parámetro?”. Pero en proyectos reales aparecen preguntas menos limpias:

- “Compara los requisitos de matrícula con los requisitos de beca y dime dónde se contradicen.”
- “¿Qué temas se repiten en todas las quejas del alumnado este curso?”
- “¿Qué documentos explican por qué cambió esta política?”
- “Busca primero en la normativa; si no basta, mira FAQ, calendario y expediente.”
- “Esta respuesta cita una fuente floja; revisa si hay otra fuente mejor.”

Aquí ya no basta con una única búsqueda top-k. El sistema necesita planificar, reescribir consultas, dividir la pregunta, escoger fuentes, comprobar si lo recuperado basta, volver a buscar si no basta y quizá consultar un grafo de relaciones. Eso es lo que solemos llamar **Agentic RAG** y **GraphRAG**. La pregunta importante no es “¿puedo hacerlo?”, sino “¿merece la pena pagar la complejidad?”.

Estado del arte con fecha de corte

Fecha de corte: 26 de mayo de 2026.
Fuentes consultadas ese día: documentación oficial de LangChain sobre arquitecturas RAG, documentación de LlamaIndex sobre estrategias agentic, documentación de Microsoft GraphRAG, papers de ReAct, Toolformer, HyDE, Self-RAG, Corrective RAG, RAPTOR y GraphRAG, y el capítulo anterior de evaluación del propio facsímil.

El patrón base sigue siendo RAG: recuperar información externa en tiempo de consulta para responder con contexto específico.[1] Lo que cambia es quién decide los pasos. LangChain distingue entre RAG de dos pasos, Agentic RAG e híbridos con validación; en su documentación, el RAG de dos pasos es más predecible y el agentic gana flexibilidad a costa

de latencia variable.[2] LlamaIndex describe estrategias agentic como routing, query transformations, sub-question query engines y agentes de datos sobre motores RAG existentes.[3]

La parte “agentic” bebe de trabajos como ReAct, que intercalan razonamiento y acciones para consultar fuentes externas durante la resolución.[4] Toolformer exploró cómo modelos de lenguaje pueden aprender a usar herramientas externas mediante ejemplos auto-supervisados.[5] En retrieval avanzado aparecen técnicas como HyDE, que genera un documento hipotético para buscar textos reales cercanos; Self-RAG, que introduce recuperación y crítica/reflexión; Corrective RAG, que evalúa la calidad de documentos recuperados y activa acciones correctivas si la evidencia es débil; y Adaptive RAG, que clasifica la complejidad de cada pregunta para decidir cuánta recuperación aplicar, desde ninguna hasta varios pasos.[6]

GraphRAG se volvió relevante porque muchas preguntas no piden “el chunk más parecido”, sino entender relaciones o patrones globales del corpus. El paper de Microsoft GraphRAG plantea un índice basado en grafo de entidades y resúmenes de comunidades para responder preguntas de comprensión global sobre colecciones privadas.[7] La documentación de GraphRAG separa local search, global search, DRIFT search, basic search y question generation.[8]

Qué no es complicar bien

Complicar un RAG no significa meter un agente delante de todo. Si el sistema siempre hace una pregunta directa sobre un documento vigente, un RAG de dos pasos puede ser mejor: menos latencia, menos coste, menos puntos de fallo y evaluación más sencilla.

Tampoco significa que el modelo “piense libremente” hasta encontrar la respuesta. En ingeniería, un Agentic RAG serio tiene herramientas permitidas, límites de pasos, trazas, umbrales de evidencia y reglas de salida. Si no puedes reconstruir qué buscó, qué encontró, qué descartó y por qué respondió, no has ganado inteligencia: has perdido depuración.

GraphRAG tampoco es “usar una base de grafos porque suena potente”. Un grafo merece la pena cuando las relaciones importan: entidades, dependencias, comunidades, jerarquías, trazabilidad entre documentos, patrones globales o preguntas que no se resuelven con un párrafo aislado. Si tu corpus son veinte FAQs cortas, GraphRAG puede ser una mudanza para cruzar la calle.

Qué sí es Agentic RAG

Agentic RAG es un RAG donde el sistema puede decidir pasos intermedios antes de responder. La palabra clave no es “autonomía”; la palabra clave es **control del flujo**.

Un RAG fijo hace esto:

```
pregunta -> retrieval -> contexto -> respuesta
```

Un Agentic RAG puede hacer esto:

```
pregunta -> diagnosticar
         -> elegir herramienta
         -> buscar
         -> evaluar evidencia
         -> responder o abstenerse
```

Técnicamente, un agente es un bucle de estado, acción y transición: el modelo observa un estado, elige una acción (buscar, leer, validar, responder) y produce un nuevo estado. Es la estructura del proceso de decisión secuencial, y la base del patrón ReAct, que intercala razonamiento y acciones de búsqueda.[9]

$$s_t = (x,\ H_t,\ E_t,\ B_t)$$

$$a_t = \pi_\theta(s_t)$$

$$s_{t+1} = \mathrm{step}(s_t,\ a_t)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x	Pregunta original.	“Compara normativa y FAQ sobre pagos pendientes”.
H_t	Historial de pasos hasta el momento.	Búsqueda en normativa, lectura de FAQ, validación.
E_t	Evidencia acumulada.	Chunks, citas, resultados SQL o relaciones de grafo.
B_t	Presupuesto restante.	Máximo 4 pasos, 2 búsquedas y 1 consulta externa.
s_t	Estado del flujo en el paso t.	Lo que el sistema sabe y puede hacer ahora.
a_t	Acción elegida.	`buscar_normativa`, `consultar_grafo`, `responder`.
π_θ	Política de decisión del modelo o del router.	Decide el siguiente paso.
step	Ejecución controlada de una acción.	Llama a una herramienta y actualiza la traza.

La lista de acciones no debería ser infinita. En un sistema real se define algo así:

ACCIÓN	QUÉ HACE	CUÁNDO TIENE SENTIDO	QUÉ REGISTRA
`buscar_texto`	Busca chunks por consulta.	Pregunta directa o evidencia textual.	Query, filtros, top-k, scores y chunks.
`buscar_hibrido`	Combina vector, BM25 y filtros.	Hay términos exactos y significado aproximado.	Rankings de cada señal y fusión.
`descomponer`	Divide una pregunta en subpreguntas.	La respuesta depende de varias fuentes.	Subpreguntas y razón de cada una.
`router`	Elige corpus, índice o herramienta.	Hay normativa, FAQ, SQL, tickets o grafo.	Opción elegida y alternativas descartadas.
`evaluar_evidencia`	Mide si lo recuperado basta.	Antes de redactar o cuando hay duda.	Soporte, citas candidatas y huecos.
`consultar_grafo`	Busca relaciones entre entidades.	Importan dependencias, comunidades o vínculos.	Nodos, aristas, fuente y camino usado.
`consultar_tabla`	Consulta datos estructurados.	Fechas, importes, estados o conteos exactos.	Consulta, resultado y validación.
`responder`	Redacta con citas.	Evidencia suficiente.	Claims, citas y decisión final.
`abstenerse`	No responde por falta de soporte.	Evidencia insuficiente o permisos insuficientes.	Qué faltó y qué fuente sería necesaria.

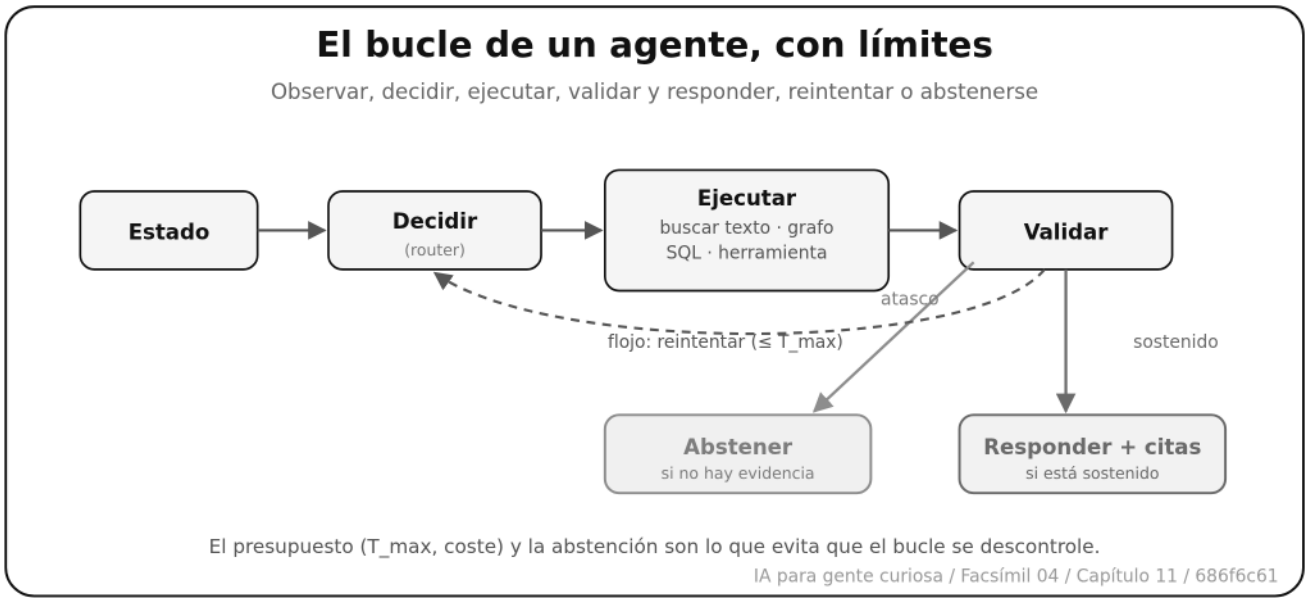
El coste de un agente es la suma del coste de cada paso, y por eso un agente sin tope de pasos es un agente sin tope de factura:

$$C_{\text{total}} = \sum_{t=1}^{T} c(a_t)$$

$$T \leq T_{\max}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
C_{total}	Coste total de la ejecución.	Tokens, llamadas, latencia o euros.
$c(a_t)$	Coste de la acción a_t.	Una búsqueda cuesta poco; una llamada LLM cuesta más.
T	Número de pasos ejecutados.	3 pasos.
$T_{\max}$	Límite máximo permitido.	5 pasos por pregunta.

La frase que debería quedarse en la cabeza: **Agentic RAG no es “hacer más cosas”; es decidir si hace falta hacerlas y dejar rastro de cada decisión**.



El porqué de los bucles: planificación y reflexión

¿Por qué un bucle mejora a una sola pasada? Porque permite **planificar y corregir**. ReAct fue el patrón base (razonar y actuar intercalados), pero la investigación añadió dos ideas que conviene conocer. **Reflexion** hace que el agente, tras un fallo, escriba en lenguaje natural qué salió mal y lo use como memoria para el siguiente intento, en vez de repetir el error.[10] **Tree-of-Thoughts** explora varias ramas de razonamiento y se queda con la mejor, en lugar de comprometerse con la primera.[11]

La lección de ingeniería no es «usa siempre el más sofisticado». Es la contraria: estas técnicas pagan en tareas de varios pasos con verificación clara (resolver, comprobar, reintentar), y son **puro derroche** en la mayoría de preguntas de un RAG, que se resuelven en una pasada. Conocerlas sirve para saber qué tienes disponible cuando una tarea de verdad lo necesita, no para meterlas por defecto.

Tipos de RAG avanzado y casos de uso

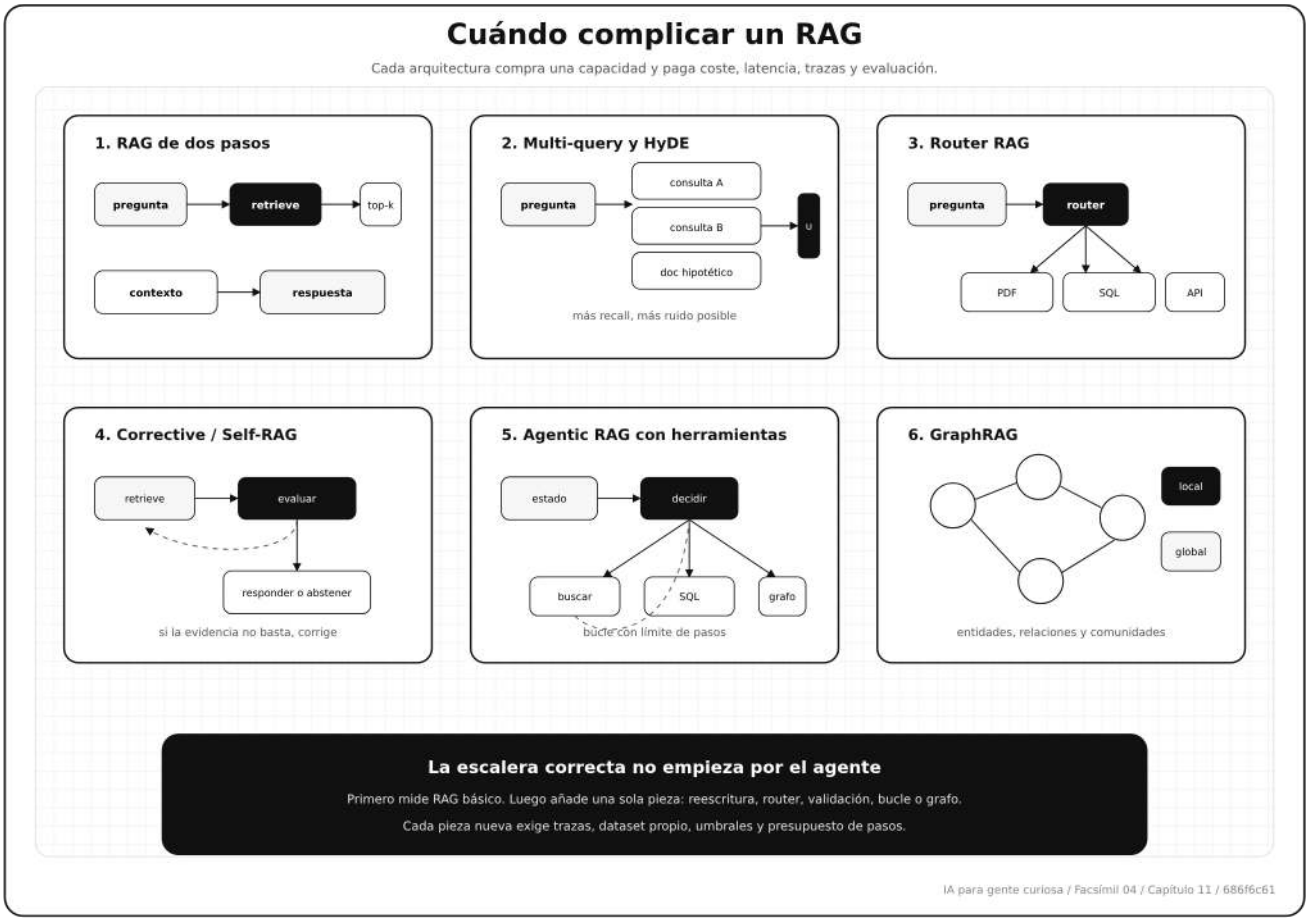
Antes de elegir una arquitectura, conviene ponerle nombre a cada patrón. Muchos problemas se arreglan con una pieza pequeña: reescritura, router o validación. No todos necesitan un bucle completo.

TIPO	QUÉ SIGNIFICA	CASO DONDE ENCAJA	NO SIRVE PARA
RAG de dos pasos	Siempre recupera primero y genera después.	FAQ, documentación técnica, normativa directa.	Preguntas que requieren decidir varias rutas.
Multi-query retrieval	Genera varias consultas para una misma pregunta y une resultados.	“doble factor”, “2FA”, “MFA” y “autenticación” pueden aparecer en documentos distintos.	Fuentes con permisos complejos si no filtras antes.
HyDE	Genera un texto hipotético que parece responder y busca documentos reales similares.	Consulta vaga sin etiquetas de relevancia ni ejemplos de entrenamiento.	Dominios donde el texto hipotético puede desviar hacia detalles falsos.
Query decomposition	Divide una pregunta compuesta en subpreguntas.	Comparar beca, matrícula y pagos pendientes.	Preguntas simples donde añade latencia sin necesidad.
Router RAG	Elige corpus, índice, herramienta o flujo.	Normativa en PDFs, estado vivo en SQL y manuales en Markdown.	Sistemas sin fuentes diferenciadas.
Hybrid RAG con validación	Recupera, evalúa evidencia, corrige o regenera si hace falta.	Dominios donde citar mal rompe confianza.	Prototipos donde aún no hay dataset de evaluación.
Corrective RAG	Evalúa la calidad de los documentos recuperados y activa otra búsqueda si no basta.	Corpus incompleto, preguntas ambiguas o retrieval frágil.	Sistemas sin fuente alternativa ni umbral definido.
Self-RAG	Decide cuándo recuperar y critica pasajes/respuestas mediante señales internas o entrenadas.	Respuestas largas donde no siempre hace falta recuperar.	Integraciones donde necesitas flujo totalmente determinista.
Adaptive RAG	Clasifica la complejidad de la pregunta y elige la estrategia, desde no recuperar hasta varios pasos.	Sistemas con mezcla de preguntas simples y complejas, donde aplicar siempre lo más caro sale mal.	Cargas homogéneas donde una sola estrategia ya basta.
RAPTOR	Construye árboles de resúmenes para recuperar a varios niveles de abstracción.	Manuales largos, informes extensos, libros internos.	Corpus pequeño con respuestas puntuales.
GraphRAG local	Usa grafo para preguntas sobre entidades y relaciones concretas.	“¿Qué políticas dependen de este requisito?”	Preguntas puramente textuales sin relaciones útiles.

TIPO	QUÉ SIGNIFICA	CASO DONDE ENCAJA	NO SIRVE PARA
GraphRAG global	Usa resúmenes de comunidades para preguntas sobre el corpus completo.	“¿Qué patrones aparecen en todas las incidencias?”	Preguntas que solo piden una fecha exacta.
RAG con herramientas	El modelo consulta búsqueda, SQL, APIs o grafo según necesidad.	“Busca la política y comprueba si mi expediente cumple.”	Saltarse permisos o validar por intuición.

La clave práctica: cada fila añade una nueva promesa y una nueva deuda. Multi-query promete más cobertura y paga más búsquedas. Router promete elegir mejor la fuente y paga clasificación. GraphRAG promete visión relacional y paga extracción, grafo, resúmenes y evaluación nueva.

Arquitecturas, una por una



El diagrama enseña la escalera de complejidad. Si una pregunta se resuelve con un RAG de dos pasos, no necesitas un bucle. Si falla por vocabulario, quizá basta multi-query. Si falla por elegir mal la fuente, quizá basta router. Si falla porque el corpus completo tiene patrones, GraphRAG puede tener sentido.

Recuperar bajo demanda y corregir: Self-RAG y CRAG

Dos arquitecturas merecen mirarse de cerca porque atacan dos errores opuestos del RAG fijo: recuperar siempre (aunque no haga falta) y confiar en lo recuperado (aunque sea malo).

Self-RAG entrena al modelo para que **decida cuándo recuperar** y para que critique su propio trabajo con tokens de reflexión: emite señales que indican si un pasaje es relevante, si la respuesta está sostenida por él y si responde a la pregunta.[12] El valor para ingeniería: no recupera para preguntas que el modelo ya sabe (ahorra coste y ruido) y se autoevalúa antes de responder.

Corrective RAG (CRAG) añade un **evaluador de recuperación**: puntúa la calidad de los chunks recuperados y, si es baja, descompone, reescribe o busca en una fuente alternativa (por ejemplo, la web) en lugar de responder con evidencia floja.[13] El valor: convierte un retrieval frágil en una respuesta robusta sin esperar a que falle en producción.

La idea común, y la lección del capítulo: **recuperar no es siempre, y confiar no es gratis**. Ambas técnicas son baratas conceptualmente (una decisión y una validación) comparadas con saltar directamente a un agente con bucle completo.

GraphRAG: qué cambia cuando aparece un grafo

Un grafo representa entidades y relaciones. En vez de tratar el corpus solo como chunks sueltos, intentamos extraer una estructura:

$$G = (V,\ E)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
G	Grafo de conocimiento extraído o curado.	Grafo de normativa, trámites y requisitos.
V	Conjunto de nodos o entidades.	“Ampliación de matrícula”, “pagos pendientes”, “beca general”.
E	Conjunto de aristas o relaciones.	“requiere”, “contradice”, “se aplica a”, “depende de”.

Un grafo de conocimiento no es solo nodos y aristas: cada arista (una relación entre dos entidades) debería guardar su procedencia para ser auditable, como insiste el trabajo de GraphRAG.[14] La estructura de una arista es:

$$e = (v_i,\ r,\ v_j,\ fuente,\ confianza)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
v_i	Entidad origen.	“Ampliación de matrícula”.
r	Relación.	“requiere”.
v_j	Entidad destino.	“no tener pagos pendientes vencidos”.
`fuente`	Documento o chunk que sostiene la relación.	`norm-2026#art-14`.
`confianza`	Señal de extracción o validación.	0,82.

La diferencia con RAG básico es que GraphRAG puede contestar usando caminos, vecindarios o comunidades:

MODO	QUÉ BUSCA	PREGUNTA TÍPICA
Local search	Entidades y relaciones cercanas a una entidad.	“¿Qué requisitos dependen de pagos pendientes?”
Global search	Resúmenes de comunidades del grafo.	“¿Qué patrones aparecen en las incidencias del curso?”
DRIFT search	Combina señal comunitaria con seguimiento local más amplio.	“Explora este tema y saca líneas de investigación.”
Question generation	Propone preguntas siguientes para investigar el corpus.	“¿Qué debería revisar ahora?”

La documentación de Microsoft GraphRAG lo expresa así: local search combina datos del grafo con chunks originales; global search busca sobre community reports en estilo map-reduce; DRIFT usa información de comunidades para ampliar el punto de partida local.[15]

GraphRAG encaja cuando la pregunta no vive en un solo fragmento:

CASO CERCANO	POR QUÉ GRAPHRAG AYUDA
Analizar miles de tickets de soporte.	Las relaciones entre producto, síntoma, versión y solución revelan comunidades.
Revisar normativa dispersa.	Las dependencias entre requisitos importan tanto como cada párrafo.
Explorar literatura académica.	Autores, métodos, datasets y resultados forman un grafo natural.
Mapear incidencias de producto.	Puede mostrar temas recurrentes y relaciones entre módulos.
Entender una organización documental.	Las entidades conectan documentos que no comparten las mismas palabras.

Pero también tiene costes:

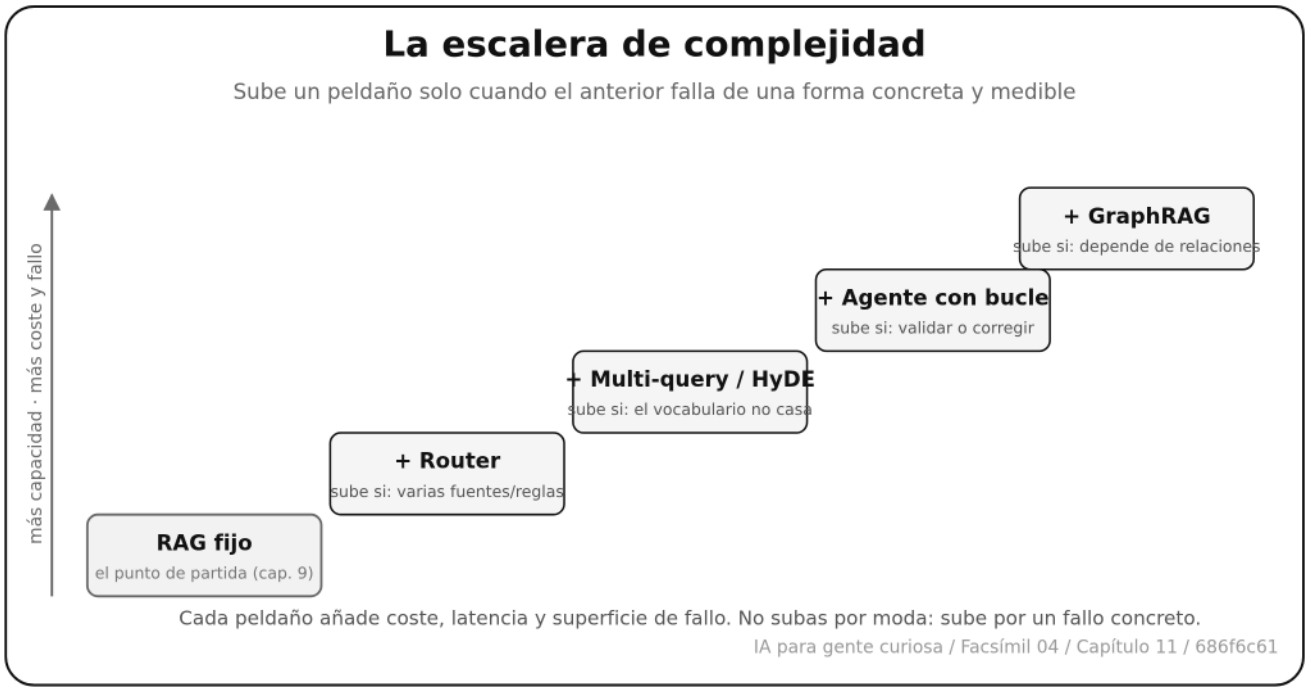
COSTE	QUÉ IMPLICA
Extracción	El modelo debe detectar entidades y relaciones con calidad suficiente.
Normalización	“doble factor”, “2FA” y “MFA” quizá son la misma entidad.
Actualización	Si cambian documentos, hay que actualizar grafo, resúmenes e índices.
Evaluación	Ya no basta medir top-k; hay que medir caminos, relaciones y resúmenes.
Explicabilidad	Una respuesta global debe enseñar qué comunidades o fuentes la sostienen.

Cómo elegir sin montar una catedral

La pregunta de arquitectura se puede formular como diagnóstico:

SÍNTOMA	PRIMERA MEJORA RAZONABLE	SI NO BASTA
No aparece el documento correcto por vocabulario.	Multi-query o HyDE.	Entrenar o cambiar embeddings; añadir BM25/fusión.
La pregunta mezcla varias cosas.	Query decomposition.	Agentic RAG con subpreguntas y validación.
Hay varias fuentes con reglas distintas.	Router.	Herramientas especializadas por fuente.
El sistema trae contexto flojo.	Retrieval validation y reranker.	Corrective RAG o búsqueda alternativa.
El corpus es largo y jerárquico.	Resúmenes por sección.	RAPTOR o índice jerárquico.
La respuesta depende de relaciones.	Grafo de entidades.	GraphRAG local.
La pregunta pide patrones del corpus entero.	Resúmenes agregados.	GraphRAG global.
Hace falta estado exacto.	Tool o SQL.	Capítulo 12: Text-to-SQL y herramientas de datos.

Regla práctica: añade una sola pieza por experimento. Si incorporas router, multi-query, GraphRAG y validación a la vez, cuando mejore o empeore no sabrás por qué.



Antes de complicar: la lista de comprobación

La escalera solo sirve si te resistes a subirla por moda. Antes de añadir un peldaño (router, agente, GraphRAG), pasa esta lista. Si falta una sola casilla, todavía no toca:

- [ ] **El RAG simple falla, y sé exactamente en qué consulta.** Tengo el caso reproducido, no una sensación.
- [ ] **El fallo es de la pieza que voy a añadir.** Si falla por retrieval básico, un grafo no lo arregla.
- [ ] **Tengo presupuesto.** Hay un tope de pasos (`T_max`), de coste y de latencia, y una condición de parada.
- [ ] **Guardo traza por paso.** Si la nueva pieza falla, podré ver qué decidió y por qué.
- [ ] **Puedo medir la mejora.** Tengo un baseline y un dataset para hacer un *ablation* (añadir solo esta pieza y comparar).
- [ ] **La mejora compensa la factura.** El ejemplo de coste y latencia de este capítulo dice que sí.

La regla detrás de la lista: complicar es una decisión de ingeniería con coste, no un gesto de sofisticación. Un sistema simple que funciona y se puede medir vale más que uno complejo que nadie sabe por qué responde lo que responde.

Evaluar RAG avanzado

Todo lo que complicas debe medirse. Un Agentic RAG no se evalúa solo por respuesta final: se evalúa por ruta.

CAPA	MÉTRICA O REVISIÓN	PREGUNTA
Router	Accuracy de ruta o matriz de confusión.	¿Eligió la fuente correcta?
Multi-query	Recall@k por consulta y unión final.	¿Las variantes trajeron evidencia nueva o solo ruido?
Decomposition	Subpreguntas necesarias y suficientes.	¿Dividió bien el problema?
Corrective RAG	Tasa de corrección útil.	¿Volvió a buscar cuando debía?
Agentic loop	Pasos, coste, latencia y salida.	¿Gastó pasos con sentido?
Graph local	Nodos/aristas correctos y fuentes.	¿El camino usado está sostenido?
Graph global	Cobertura, diversidad y trazabilidad.	¿El resumen global representa el corpus?
Respuesta	Groundedness, citas y abstención.	¿Lo dicho está sostenido?

La decisión de adoptar (o no) una arquitectura más compleja se puede escribir como un gate: solo complicas si ganas calidad por encima de un umbral sin romper los presupuestos de coste y tiempo. Un gate mínimo:

$$G \geq \tau_g$$

$$C \leq C_{\max}$$

$$T \leq T_{\max}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
G	Groundedness o soporte mínimo.	0,92.
τ_g	Umbral de soporte.	0,90.
C	Coste total de la ejecución.	0,012 euros o 8.000 tokens.
$C_{\max}$	Coste máximo aceptable.	0,02 euros.
T	Pasos realizados.	4.
$T_{\max}$	Máximo de pasos.	5.

Para GraphRAG hay una evaluación adicional: no basta con que la respuesta suene bien. Hay que revisar si las entidades, relaciones y comunidades usadas son correctas. Un resumen global puede ser fluido y aun así esconder que una comunidad importante no entró en el mapa.

Casos cercanos

Secretaría académica. Preguntan: “¿Puedo ampliar matrícula si tengo una beca pendiente y pagos vencidos?”. Un RAG básico quizá encuentra solo la normativa de ampliación. Query decomposition separa “ampliación”, “beca pendiente” y “pagos vencidos”. El router decide si mirar normativa, becas y calendario. La respuesta final debe citar cada pieza.

Equipo de soporte técnico. Preguntan: “¿Qué problemas se repiten desde la última versión?”. No quieres una respuesta sobre un ticket concreto. Quieres agrupar incidencias por producto, síntoma, versión y solución. Aquí GraphRAG global o resúmenes jerárquicos pueden mostrar patrones que un top-k no ve.

Documentación de ingeniería. Preguntan: “¿Cómo configuro el conector si uso PostgreSQL y despliegue local?”. El sistema puede necesitar buscar en documentación, consultar ejemplos de configuración y revisar límites de versión. Agentic RAG con herramientas de búsqueda acotadas puede ser útil, siempre que cite y limite dominios.

Compliance documental. Preguntan: “¿Qué políticas dependen de este requisito?”. GraphRAG local encaja porque lo importante es el vecindario de una entidad: requisito, políticas, controles, evidencias y documentos fuente.

Producto con datos vivos. Preguntan: “¿Qué clientes están afectados y qué documento explica la política?”. Aquí no basta RAG. Necesitas tool o SQL para estado vivo y RAG para explicar la política. Este puente prepara el capítulo 12.

Para entenderlo sin perderse

Una forma sencilla de explicarlo: un RAG básico se parece a preguntar a alguien que tiene una estantería y te trae tres páginas parecidas a tu pregunta. Si esas páginas contienen la respuesta, perfecto. Si la pregunta exige comparar, comprobar vigencia, mirar otra fuente o seguir una relación entre documentos, la persona necesita una libreta de trabajo: primero mira dónde buscar, luego consulta, después comprueba si lo encontrado basta y solo entonces responde.

En esa metáfora:

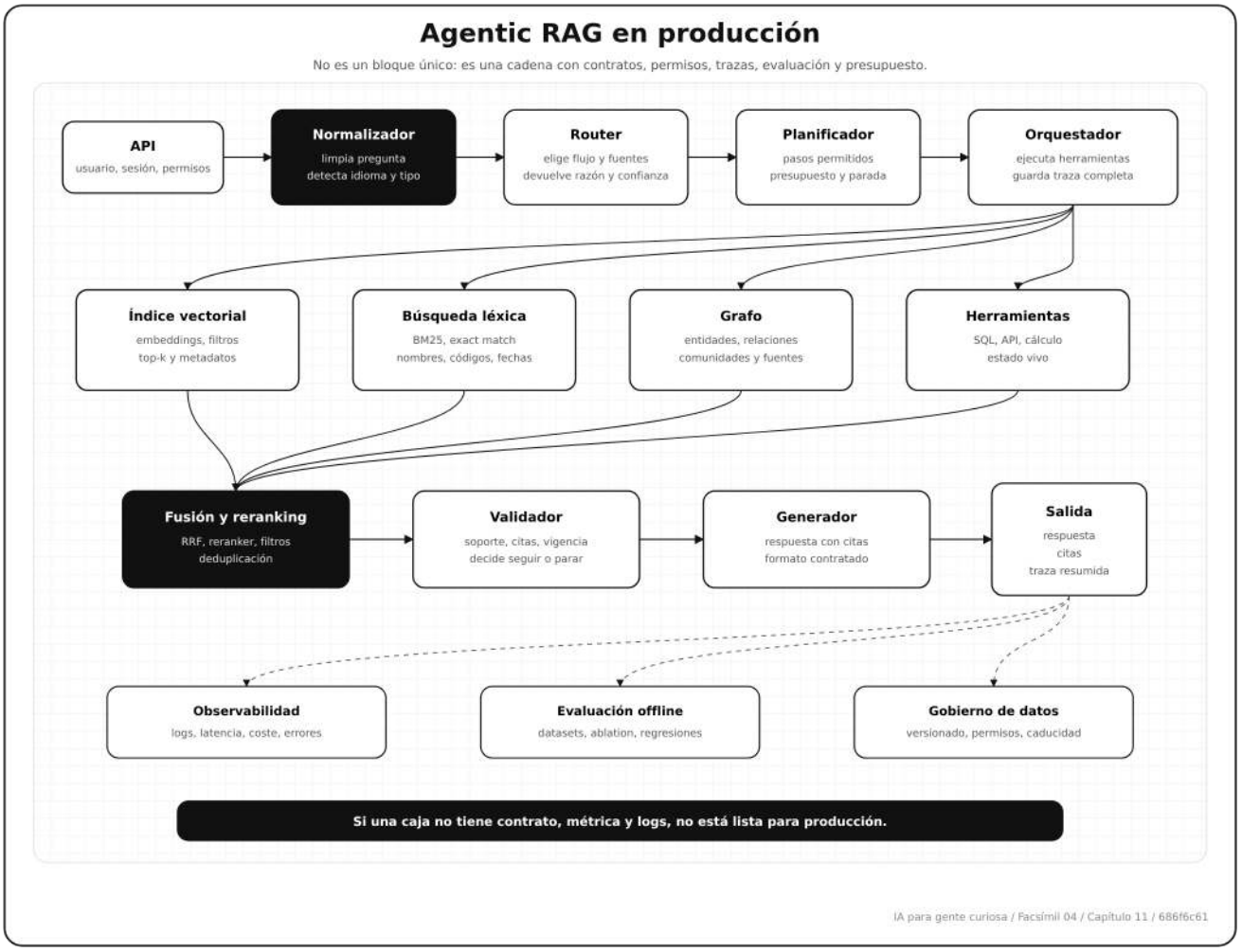
PIEZA	TRADUCCIÓN MENTAL	QUÉ DEBERÍA QUEDAR CLARO
RAG básico	Traer páginas parecidas.	Sirve si la pregunta vive en uno o varios fragmentos cercanos.
Router	Decidir en qué estantería mirar.	No es una caja negra: clasifica la pregunta y elige fuente.
Multi-query	Preguntar lo mismo con varias palabras.	Mejora cobertura cuando el vocabulario cambia.
Descomposición	Separar una pregunta grande en preguntas pequeñas.	Ayuda si hay que comparar o juntar varias condiciones.
Validador	Comprobar si las páginas sostienen la respuesta.	Reduce respuestas bonitas pero poco justificadas.
Traza	Libreta de lo que hizo el sistema.	Permite depurar, auditar y evaluar.
Grafo	Mapa de cosas conectadas.	Ayuda cuando la relación importa tanto como el texto.
Comunidad	Barrio dentro del grafo.	Grupo de entidades que aparecen conectadas muchas veces.

El error habitual es imaginar que Agentic RAG “razona más”. En producción me interesa una definición menos romántica: **Agentic RAG toma decisiones explícitas entre pasos permitidos y deja evidencia de esas decisiones**. Si no hay pasos permitidos, presupuesto y traza, no tienes una arquitectura: tienes una conversación difícil de depurar.

Cómo lo montaría en un sistema real

Si tuviera que construir esto en una empresa, no empezaría por GraphRAG ni por un agente completo. Empezaría por una pregunta incómoda: “¿qué fallo real quiero corregir?”. Si el fallo es vocabulario, multi-query. Si el fallo es escoger mal la fuente, router. Si el fallo es no saber si la evidencia basta, validador. Si el fallo es entender relaciones entre documentos, grafo.

La arquitectura mínima seria tendría estas capas:



Cada caja de la figura debería poder probarse por separado. Un ingeniero no debería preguntar “¿funciona el agente?”, sino cosas más concretas:

- ¿El router elige bien entre normativa, FAQ, SQL, tickets y grafo?
- ¿El retriever trae evidencia suficiente con filtros de permisos?
- ¿La fusión elimina duplicados y mantiene fuentes distintas?
- ¿El validador detecta falta de soporte, citas flojas o documentos caducados?
- ¿El generador respeta el formato de salida y no inventa campos?
- ¿La traza permite reproducir la respuesta cinco días después?

Herramientas y orquestadores que ya existen

No tienes que escribir el bucle del agente ni el pipeline del grafo a mano. Hay orquestadores maduros, y conocerlos evita reinventar lo difícil (el control de flujo, las trazas, los reintentos). La elección depende de cuánto control quieras y de si tu problema es agentic, de grafo o de pipeline.

HERRAMIENTA	PARA QUÉ	CUÁNDO ELEGIRLA
LangGraph	Modela el agente como un grafo de estados explícito (nodos, aristas, ciclos con condición).	Quieres control fino del bucle, los reintentos y la parada, con trazas.
LlamaIndex (Workflows / agents)	Workflows por eventos y agentes de RAG sobre tus índices.	Ya indexas con LlamaIndex y quieres pasos agentic encima.
Microsoft GraphRAG	Pipeline completo de GraphRAG: extracción de entidades, comunidades (Leiden) y resúmenes.	Vas a montar GraphRAG y no quieres reimplementar la indexación del grafo.
DSPy	Programa el sistema como módulos y **optimiza** sus prompts contra una métrica.	Quieres dejar de ajustar prompts a mano y optimizar contra tu eval.
Haystack	Pipelines de RAG componibles (retrievers, rankers, generadores) listos para producción.	Quieres un pipeline modular y estable más que un agente libre.

La regla, otra vez: la herramienta da el andamiaje, pero el contrato de cada pieza (entrada, salida, errores, timeout, permisos, trazas) y el presupuesto los pones tú. Un orquestador no te exime de saber qué hace cada paso y por qué.

Contratos de herramientas

Una herramienta en un sistema agentic no debería ser “una función que el modelo puede llamar”. Debe ser un contrato. El contrato dice qué recibe, qué devuelve, qué permisos exige, cuánto tarda, cómo falla y cómo se audita.

Ejemplo de contrato de una herramienta de búsqueda documental:

```
{
  "name": "buscar_normativa",
  "input": {
    "query": "string",
    "filters": {
      "curso": "2026",
      "estado": "vigente"
    },
    "top_k": 8
  },
  "output": {
    "results": [
      {
        "chunk_id": "norm-2026#art-14",
        "score": 0.84,
        "source": "Normativa 2026",
        "valid_from": "2026-01-01",
        "text": "fragmento recuperado"
      }
    ]
  },
  "errors": [
    "permission_denied",
    "source_unavailable",
    "low_recall"
  ],
  "timeout_ms": 1200,
  "audit": true
}
```

Lo importante no es el JSON, sino la disciplina:

CAMPO	QUÉ APORTA	QUÉ SE ROMPE SI FALTA
`name`	Identidad estable de la herramienta.	No puedes comparar ejecuciones ni versionar cambios.
`input`	Qué puede pedir el sistema.	El modelo puede mandar consultas vagas o imposibles.
`filters`	Permisos, vigencia, idioma, cliente o corpus.	Puedes mezclar documentos que no deberían mezclarse.
`top_k`	Límite de resultados.	Coste y contexto crecen sin control.
`score`	Señal de recuperación.	No sabes por qué entró un fragmento.
`source`	Procedencia legible.	No puedes citar ni revisar.
`valid_from`	Vigencia temporal.	Respuestas correctas pueden quedar obsoletas.
`errors`	Fallos esperados.	El sistema trata un fallo como si fuera ausencia de información.
`timeout_ms`	Límite de latencia.	Una herramienta lenta secuestra toda la respuesta.
`audit`	Obligación de guardar traza.	No puedes reproducir ni explicar decisiones.

Un contrato de herramienta también debe declarar qué hacer cuando no basta:

RESULTADO	DECISIÓN CORRECTA
`permission_denied`	No buscar atajos; responder que no hay permiso suficiente.
`source_unavailable`	Degradar con aviso o pedir reintento, según criticidad.
`low_recall`	Reescribir consulta, usar búsqueda híbrida o abstenerse.
`empty_result`	Distinguir “no existe” de “no he podido encontrarlo”.
`conflicting_sources`	Priorizar vigencia, autoridad y citar el conflicto.

GraphRAG por dentro

GraphRAG no empieza en la query; empieza en la ingesta. Antes de responder hay que convertir documentos en entidades, relaciones, comunidades y resúmenes. Ese proceso tiene mucha ingeniería escondida.

Una tubería razonable:

PASO	QUÉ HACE	RIESGO PRINCIPAL	CONTROL
Ingesta	Lee PDFs, HTML, Markdown, tickets, tablas o transcripciones.	Perder estructura del documento.	Guardar fuente, página, sección y fecha.
Chunking	Parte documentos en unidades recuperables.	Cortar relaciones importantes.	Chunks con solape y metadatos ricos.
Extracción	Detecta entidades y relaciones.	Extraer nombres distintos para lo mismo.	Diccionario, revisión y normalización.
Canonicalización	Une variantes de una entidad.	Mezclar entidades parecidas pero distintas.	Alias, reglas y confianza.
Aristas	Crea relaciones con fuente.	Relación sin prueba documental.	Toda arista guarda chunk, frase y score.
Comunidades	Agrupa zonas densas del grafo.	Comunidades demasiado grandes o pequeñas.	Medir modularidad y revisar muestras.
Resúmenes	Resume comunidades.	Perder excepciones importantes.	Citar nodos y documentos representativos.
Índices	Indexa texto, nodos, aristas y resúmenes.	Recuperar solo una vista parcial.	Búsqueda híbrida y evaluación por tipo.
Actualización	Reprocesa cambios del corpus.	Grafo viejo con documentos nuevos.	Versionado, diffs y caducidad.

Se puede escribir de forma compacta:

$$V = \text{canon}(\text{entidades}(D))$$

$$E = \{(v_i,\ r,\ v_j,\ fuente,\ confianza)\}$$

$$R_c = \text{resumen}(C_c,\ fuentes_c)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
D	Documentos de entrada.	Normativa, FAQ, tickets y manuales.
V	Entidades normalizadas.	“2FA” y “doble factor” como la misma entidad.
E	Relaciones con fuente.	“ampliación requiere no tener pagos vencidos”.
C_c	Comunidad del grafo.	Trámites de matrícula y pagos.
R_c	Resumen de comunidad.	“Los bloqueos se concentran en pagos y plazos”.

La parte difícil no es dibujar nodos. La parte difícil es saber si el nodo es correcto, si dos nodos son la misma cosa, si la relación tiene fuente, si el resumen no borra excepciones y si el grafo se actualiza cuando cambia el corpus.

Comunidades y consulta: local frente a global

¿De dónde salen esos «resúmenes de comunidad»? Una vez construido el grafo, GraphRAG detecta **comunidades**: grupos de entidades más conectadas entre sí que con el resto, usando un algoritmo de detección de comunidades. El estándar es **Leiden**, que mejora al clásico Louvain garantizando que las comunidades queden bien conectadas.[16] Luego un LLM resume cada comunidad. Eso permite responder preguntas que ningún chunk suelto contiene, porque la respuesta vive en la *estructura*, no en el texto.

Esa construcción habilita dos modos de consulta que conviene no confundir:

MODO	QUÉ HACE	PREGUNTA TÍPICA
Local	Parte de una entidad concreta y recorre sus vecinos en el grafo.	«¿Qué requisitos dependen de la beca?»
Global	Agrega los resúmenes de comunidad para responder sobre el corpus entero.	«¿Qué patrones de bloqueo se repiten en toda la normativa?»

La regla de ingeniería: el modo **local** es para preguntas sobre relaciones de una entidad; el **global**, para preguntas de síntesis sobre todo el corpus, que es justo donde un RAG por chunks fracasa porque la respuesta no está escrita en ningún sitio, hay que componerla.

Coste, latencia y presupuesto

Un RAG avanzado puede mejorar respuestas y empeorar producto si duplica latencia sin medirlo. Por eso hay que convertir la arquitectura en números.

Si generas q consultas, traes k fragmentos por consulta y cada fragmento tiene L tokens de media:

$$N_{\text{ctx}} \approx q \cdot k \cdot L$$

Si además usas reranker y un generador:

$$T_{\text{total}} = T_{\text{router}} + \max_i(T_{\text{retrieval},i}) + T_{\text{rerank}} + T_{\text{validación}} + T_{\text{generación}}$$

Y el coste de una respuesta puede aproximarse así:

$$C_{\text{respuesta}} = C_{\text{emb}} + C_{\text{retrieval}} + C_{\text{rerank}} + C_{\text{LLM}}$$

NÚMERO	QUÉ SIGNIFICA	QUÉ MIRAR EN PRODUCCIÓN
q	Número de consultas generadas.	Si sube, suben recall, coste y ruido.
k	Fragmentos recuperados por consulta.	Un top-k alto puede tapar la evidencia buena.
L	Tokens por fragmento.	Chunks largos llenan contexto rápido.
T_{total}	Latencia total.	Medir P50, P95 y timeouts, no solo media.
$C_{\text{respuesta}}$	Coste por respuesta.	Dividir por respuesta útil, no por llamada.
C_{index}	Coste de indexar.	En GraphRAG puede ser alto antes de la primera query.

Para GraphRAG hay otro coste:

$$C_{\text{graph-index}} = C_{\text{extracción}} + C_{\text{normalización}} + C_{\text{comunidades}} + C_{\text{resúmenes}}$$

Donde $C_{\text{extracción}}$ es el coste de que un LLM extraiga entidades y relaciones de cada documento, $C_{\text{normalización}}$ el de unificar entidades duplicadas, $C_{\text{comunidades}}$ el de detectarlas (Leiden) y $C_{\text{resúmenes}}$ el de resumir cada una con un LLM. Este coste se paga al construir o actualizar el índice, no solo al responder. Por eso GraphRAG puede ser brillante en colecciones estables y caro en corpus que cambian cada hora.

Pongámoslo en números para que las cuentas dejen de ser abstractas. Una consulta agentic que genera $q = 3$ subconsultas, trae $k = 5$ fragmentos por consulta de $L = 300$ tokens cada uno, con reranker, validación y un generador:

COMPONENTE	CUENTA	RESULTADO
Contexto	$N_{\text{ctx}} = 3 \cdot 5 \cdot 300$	4 500 tokens
Latencia	$0{,}3 + 0{,}4 + 0{,}5 + 0{,}3 + 1{,}5$ (router, retrieval, rerank, validación, generación)	3,0 s
Coste	embeddings + 4 700 tokens de entrada + 400 de salida, a tarifas ilustrativas	≈ 0,013 por respuesta

Compáralo con un RAG fijo del capítulo 9 (una consulta, sin rerank ni validación): la versión agentic multiplica por tres el contexto y la latencia, y dobla el coste. Esa es la factura de complicar, y solo se justifica si la calidad sube lo suficiente como para pagarla. La cuenta, no la intuición, es lo que te dice si el peldaño merece la pena.

Los fallos propios de un agente

Dar autonomía a un sistema añade modos de fallo que un RAG fijo no tiene, y que un ingeniero debe acotar **por diseño**, no esperar a que aparezcan en producción:

FALLO	QUÉ OCURRE	CÓMO SE ACOTA
Bucle infinito	El agente repite pasos sin converger.	Tope de pasos $T_{\max}$ y condición de parada explícita.
Deriva del objetivo	A los pocos pasos responde a una pregunta distinta de la original.	Re-anclar la pregunta original x en cada paso del estado.
Sobre-recuperación	Trae más y más contexto «por si acaso».	Presupuesto de contexto y de coste por respuesta.
Atasco sin evidencia	No encuentra nada útil y aun así insiste.	Abstención cuando el soporte no supera el umbral.
Coste descontrolado	Cada reintento multiplica llamadas al LLM.	*Circuit breaker*: corta y degrada a una ruta simple.

La regla que resume todo el capítulo: **un agente sin límites es una factura sin límites**. La autonomía es útil solo cuando va acompañada de un presupuesto (`T_max`, coste, tiempo) y de una salida de emergencia que devuelva el control a una ruta simple cuando algo se

descontrola.

Evaluación para ingeniería

La evaluación de RAG avanzado tiene que separar piezas. Si solo miras la respuesta final, no sabes si mejoró el retriever, el router, el grafo o simplemente hubo suerte en una muestra.

Un plan de evaluación serio:

PRUEBA	QUÉ COMPARA	PREGUNTA QUE RESPONDE
Baseline	RAG básico contra sistema nuevo.	¿Complicar mejora algo medible?
Ablation	Quitar una pieza cada vez.	¿Qué aporta router, multi-query, grafo o validador?
Router accuracy	Ruta esperada contra ruta elegida.	¿Va a la fuente correcta?
Recall@k	Evidencia esperada dentro del top-k.	¿Recupera lo necesario?
MRR / nDCG	Orden de documentos relevantes.	¿Lo bueno aparece arriba?
Node precision	Nodos correctos del grafo.	¿Las entidades recuperadas son válidas?
Edge precision	Relaciones correctas del grafo.	¿Las aristas están sostenidas por fuentes?
Citation support	Claims con cita suficiente.	¿Cada afirmación importante está soportada?
Abstention rate	Casos donde no responde.	¿Se abstiene cuando falta evidencia?
Latencia P95	Tiempo para el 95% de consultas.	¿El producto aguanta en uso real?
Coste por respuesta útil	Coste dividido por respuestas aceptables.	¿La mejora compensa?

El ablation test es especialmente sano:

VARIANTE	QUÉ ACTIVA	QUÉ ESPERAS APRENDER
A	RAG básico.	Línea base.
B	A + búsqueda híbrida.	Si el problema era vocabulario exacto.
C	B + multi-query.	Si faltaba cobertura semántica.
D	C + router.	Si elegir fuente aporta mejora.
E	D + validador.	Si reduce respuestas sin soporte.
F	E + GraphRAG.	Si las relaciones añaden valor real.

Si F gana solo un 1% pero dobla latencia y coste, quizá no merece producción. Si F gana mucho en preguntas relacionales y pierde en preguntas simples, el router debe activar GraphRAG solo cuando toque.

Caso completo de diseño

Supongamos un sistema con 50.000 documentos internos: normativa, manuales, tickets, FAQs, actas y algunas tablas con estado vivo. El objetivo es responder a equipos internos con citas, permisos y trazas.

Yo lo diseñaría por fases:

FASE	QUÉ HARÍA	CRITERIO PARA AVANZAR
1. Corpus	Inventario de fuentes, permisos, vigencia y formatos.	Saber qué puede ver cada usuario y qué fuente manda.
2. Baseline	RAG básico con chunking, embeddings y búsqueda híbrida.	Dataset de 100-300 preguntas reales con respuestas esperadas.
3. Evaluación	Medir recall@k, soporte de citas, abstención y latencia.	Detectar el fallo dominante.
4. Router	Separar normativa, FAQ, tickets, manuales, SQL y grafo.	Accuracy de ruta suficiente y trazas legibles.
5. Validación	Añadir groundedness, vigencia y conflicto entre fuentes.	Menos respuestas sin soporte.
6. Grafo local	Entidades y relaciones para normativa y dependencias.	Mejora clara en preguntas relacionales.
7. GraphRAG global	Comunidades para tickets, actas e incidencias.	Mejora clara en preguntas de patrones.
8. Producción	Observabilidad, costes, permisos, caché y regresiones.	Alertas y evaluación continua antes de ampliar uso.

La decisión final no sería “usar GraphRAG sí o no”. Sería una política:

TIPO DE PREGUNTA	FLUJO RECOMENDADO
“¿Dónde dice X?”	RAG básico con búsqueda híbrida y citas.
“¿Cuál es el estado actual de X?”	Tool o SQL, después explicación con RAG.
“Compara X e Y”	Descomposición, router y validación.
“¿Qué depende de este requisito?”	GraphRAG local.
“¿Qué patrones aparecen en todo el corpus?”	GraphRAG global.
“No hay evidencia suficiente”	Abstención con fuente necesaria.

Este diseño también ayuda a una persona curiosa: no hay una herramienta universal. Hay una escalera. Cada peldaño se sube cuando el anterior falla de una forma concreta.

En el día a día

En el día a día, lo agentic y GraphRAG aparecen cuando una pregunta deja de tener una sola respuesta en un solo fragmento. «¿Puedo ampliar matrícula si tengo una beca pendiente y un pago vencido?» no se resuelve con un chunk: hay que recuperar dos normas, ver cómo se relacionan y componer. Ahí un router que descompone la pregunta, o un grafo que conoce qué requisito bloquea a cuál, gana al RAG fijo. Pero la mayoría de las preguntas de un día normal («¿cuándo abre la ampliación?») se resuelven con el RAG básico del capítulo 9, y meterles un agente solo añade latencia, coste y formas nuevas de fallar.

La parte delicada, y el corazón de este capítulo, es saber **cuándo no complicar**. Cada peldaño de la escalera (router, multi-query, agente con bucle, GraphRAG) se sube cuando el anterior falla de una forma concreta y medible, no porque suene moderno. Un equipo que monta GraphRAG sin tener preguntas que dependan de relaciones acaba manteniendo una catedral para resolver lo que un RAG simple ya hacía.

Por qué debería importarte

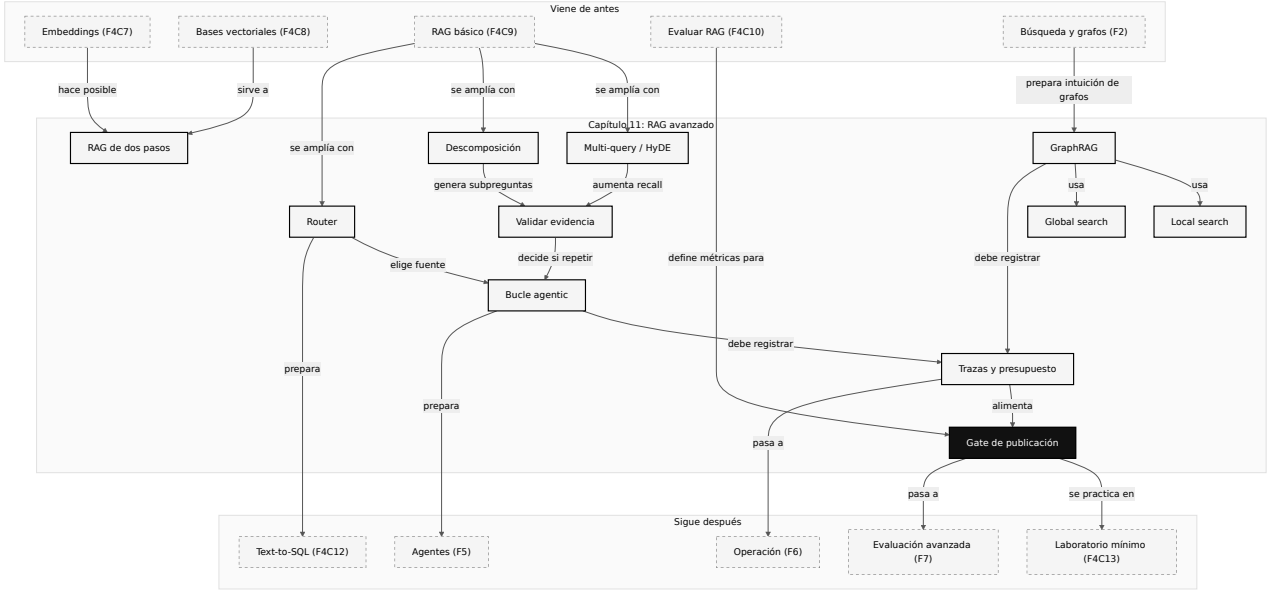
Porque la complejidad mal puesta es la forma más cara de empeorar un sistema. Un agente con bucles sin presupuesto es una factura impredecible; un grafo sin relaciones útiles es mantenimiento permanente sin mejora; una arquitectura con cinco piezas nuevas a la vez es imposible de depurar cuando falla. Saber cuándo escalar, y cuándo resistirse, es lo que separa a un ingeniero de alguien que colecciona arquitecturas.

Y porque cada peldaño añade superficie de fallo que hay que medir. Lo agentic no se evalúa solo por la respuesta final: hay que evaluar el router, las subpreguntas, los nodos y aristas del grafo, los pasos del bucle y el coste por respuesta útil. Si añades complejidad sin añadir medición, no has mejorado el sistema: solo has perdido la capacidad de saber si funciona.

Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Llamar agente a cualquier if	Un router determinista puede ser suficiente; no todo flujo condicional es un agente.	Nombrar la pieza exacta: router, validador, descomposición o bucle.
Añadir bucles sin presupuesto	La latencia y el coste se vuelven impredecibles.	Definir $T_{\max}$, coste máximo y condición de parada.
Usar GraphRAG sin relaciones útiles	Si no hay entidades ni vínculos relevantes, el grafo añade mantenimiento sin mejorar respuestas.	Probar primero si las preguntas fallan por relaciones, no por retrieval básico.
No guardar la ruta de decisión	Si falla, no sabes qué fuente eligió ni por qué.	Guardar traza: ruta, consultas, resultados, scores, citas y validaciones.
Medir solo groundedness final	Puede responder bien por casualidad aunque haya elegido mal el camino.	Evaluar router, subpreguntas, nodos, aristas, pasos y respuesta.
Mezclar todas las mejoras a la vez	No puedes atribuir la mejora ni depurar el fallo.	Añadir una pieza por experimento y comparar contra baseline.
No definir contratos de herramientas	Cada integración acaba devolviendo lo que quiere y fallando de forma distinta.	Declarar input, output, errores, timeout, permisos y trazas.
Ignorar el coste de indexar GraphRAG	El grafo también cuesta antes de contestar la primera pregunta.	Separar coste de indexación y coste por query.
Mirar solo la media de latencia	La media puede ocultar consultas lentas que rompen la experiencia.	Medir P50, P95, timeouts y coste por respuesta útil.

Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Agentic RAG	RAG donde un modelo o router decide pasos de búsqueda, herramientas y validación antes de responder.
RAG de dos pasos	Flujo fijo: recuperar contexto y generar respuesta.
Multi-query retrieval	Varias consultas para cubrir vocabularios distintos de una misma necesidad.
HyDE	Generar un documento hipotético para buscar documentos reales parecidos.
Query decomposition	Dividir una pregunta grande en subpreguntas recuperables.
Router	Pieza que decide qué corpus, índice, herramienta o flujo usar.
Corrective RAG	RAG que evalúa si lo recuperado basta y corrige si no basta.
Self-RAG	Enfoque donde la recuperación y la crítica forman parte del comportamiento del modelo.
RAPTOR	Recuperación con árbol de resúmenes a distintos niveles de abstracción.
GraphRAG	RAG basado en grafo de entidades, relaciones y resúmenes.
Local search	Búsqueda centrada en entidades o relaciones concretas del grafo.
Global search	Búsqueda sobre resúmenes de comunidades para preguntas del corpus completo.
Community summary	Resumen de un grupo de nodos relacionados dentro del grafo.
Presupuesto de pasos	Límite de acciones, llamadas, coste o latencia permitido.
Contrato de herramienta	Especificación de entradas, salidas, errores, permisos, timeout y auditoría de una herramienta.
Traza	Registro reproducible de ruta, consultas, resultados, decisiones, costes y citas usadas.
Canonicalización	Unión controlada de variantes que representan la misma entidad.
Ablation test	Prueba donde se quita una pieza del sistema para medir qué aporta realmente.

TÉRMINO	DEFINICIÓN
Latencia P95	Tiempo por debajo del cual responde el 95% de las consultas.
Edge precision	Proporción de relaciones del grafo que son correctas y están sostenidas por fuentes.

Antes de pasar página

- [ ] ¿Puedo explicar cuándo basta un RAG de dos pasos? (Si no, vuelve a «Qué sí es Agentic RAG».)
- [ ] ¿Sé distinguir multi-query, HyDE y query decomposition? (Si no, vuelve a «Arquitecturas, una por una».)
- [ ] ¿Puedo explicar qué hace un router y qué debe registrar? (Si no, vuelve a «Arquitecturas, una por una».)
- [ ] ¿Sé por qué Corrective RAG y Self-RAG no son “más complejidad por presumir”, sino validación y decisión de recuperación? (Si no, vuelve a «Arquitecturas, una por una».)
- [ ] ¿Puedo escribir qué es un grafo $G = (V, E)$ y qué significa una arista con fuente? (Si no, vuelve a «GraphRAG: qué cambia cuando aparece un grafo».)
- [ ] ¿Sé cuándo GraphRAG local encaja mejor que GraphRAG global? (Si no, vuelve a «GraphRAG: qué cambia cuando aparece un grafo».)
- [ ] ¿Puedo explicar cómo se construye GraphRAG: entidades, relaciones, comunidades y resúmenes? (Si no, vuelve a «GraphRAG por dentro».)
- [ ] ¿Sé diseñar un contrato mínimo para una herramienta de búsqueda? (Si no, vuelve a «Contratos de herramientas».)
- [ ] ¿Puedo definir un presupuesto de pasos, coste y latencia para un Agentic RAG? (Si no, vuelve a «Coste, latencia y presupuesto».)
- [ ] ¿Sé calcular por qué multi-query aumenta contexto, coste y ruido? (Si no, vuelve a «Coste, latencia y presupuesto».)
- [ ] ¿Sé qué métricas miraría además de la respuesta final? (Si no, vuelve a «Evaluación para ingeniería».)
- [ ] ¿Puedo plantear un ablation test para saber qué pieza aporta mejora? (Si no, vuelve a «Evaluación para ingeniería».)
- [ ] ¿Sé qué mirar en la traza de cada pregunta para saber si el sistema acertó? (Si no, repasa la sección correspondiente.)

En resumen

IDEA FUERZA	DETALLE
No empieces por Agentic RAG.	Empieza por RAG básico evaluado; complica solo cuando sabes qué falla.
Agentic RAG decide pasos.	Puede reescribir, dividir, enrutar, consultar herramientas, validar y volver a buscar.
GraphRAG usa relaciones.	Sirve cuando importan entidades, dependencias, comunidades o preguntas globales del corpus.
Producción exige contratos.	Cada herramienta necesita entradas, salidas, errores, permisos, timeout y trazas.
GraphRAG cuesta antes de responder.	Extraer entidades, normalizar, crear comunidades y resumir también forman parte del presupuesto.
Cada pieza nueva exige evaluación propia.	Router, descomposición, grafo, bucle y respuesta final se miden por separado.
El ablation test evita autoengaños.	Compara RAG básico contra mejoras incrementales para saber qué aporta cada pieza.
El coste no es solo dinero.	También pagas latencia, trazabilidad, mantenimiento, permisos y complejidad de depuración.

Para saber más

Asai, A., Wu, Z., Wang, Y., Sil, A. y Hajishirzi, H. (2023). *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*. arXiv

Cormack, G. V., Clarke, C. L. A. y Buettcher, S. (2009). Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. *Proceedings of SIGIR*, 758-759. DOI

Edge, D. y otros (2024). *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*. arXiv

Gao, L., Ma, X., Lin, J. y Callan, J. (2022). *Precise Zero-Shot Dense Retrieval without Relevance Labels*. arXiv

LangChain. (2026). *Retrieval*. Documentación oficial

Lewis, P. y otros (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems 33*, 9459-9474. NeurIPS

LlamaIndex. (2026). *Agentic strategies*. Documentación oficial

Microsoft. (2026). *GraphRAG Query Engine overview*. Documentación oficial

OpenAI. (2026). *Graders*. Documentación oficial

Sarthi, P. y otros (2024). *RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval*. arXiv

Schick, T. y otros (2023). *Toolformer: Language Models Can Teach Themselves to Use Tools*. DOI

Yan, S.-Q., Gu, J.-C., Zhu, Y. y Ling, Z.-H. (2024). *Corrective Retrieval Augmented Generation*. arXiv

Yao, S. y otros (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*. *International Conference on Learning Representations*. arXiv

---

Notas

1. Lewis, P. y otros (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. *Advances in Neural Information Processing Systems 33*, 9459-9474. NeurIPS. ↩

2. LangChain. (2026). *Retrieval*. Documentación oficial. Consultado el 26 de mayo de 2026. La página compara 2-step RAG, Agentic RAG e Hybrid RAG, y explica que un agente puede decidir cuándo y cómo recuperar mediante herramientas. ↩

3. LlamaIndex. (2026). *Agentic strategies*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

4. Yao, S. y otros (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*. *International Conference on Learning Representations*. arXiv. ↩

5. Schick, T. y otros (2023). *Toolformer: Language Models Can Teach Themselves to Use Tools*. DOI. ↩

6. Gao, L., Ma, X., Lin, J. y Callan, J. (2022). *Precise Zero-Shot Dense Retrieval without Relevance Labels*. arXiv. Asai, A., Wu, Z., Wang, Y., Sil, A. y Hajishirzi, H. (2023). *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*. arXiv. Yan, S.-Q., Gu, J.-C., Zhu, Y. y Ling, Z.-H. (2024). *Corrective Retrieval Augmented Generation*. arXiv. Jeong, S., Baek, J., Cho, S., Hwang, S. J. y Park, J. C. (2024). *Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity*. arXiv. ↩

7. Edge, D. y otros (2024). *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*. arXiv. ↩

8. Microsoft. (2026). *GraphRAG Query Engine overview*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

9. Shunyu Yao y otros (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*. ICLR. https://arxiv.org/abs/2210.03629. La formulación de estado-acción-transición es la de los procesos de decisión secuencial: Richard S. Sutton y Andrew G. Barto. (2018). *Reinforcement Learning: An Introduction* (2.ª ed.). MIT Press. ↩

.0. Noah Shinn y otros (2023). *Reflexion: Language Agents with Verbal Reinforcement Learning*. NeurIPS. https://arxiv.org/abs/2303.11366. ↩

.1. Shunyu Yao y otros (2023). *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. NeurIPS. https://arxiv.org/abs/2305.10601. ↩

.2. Akari Asai y otros (2023). *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*. ICLR 2024. https://arxiv.org/abs/2310.11511. ↩

.3. Shi-Qi Yan y otros (2024). *Corrective Retrieval Augmented Generation*. https://arxiv.org/abs/2401.15884. ↩

.4. Darren Edge y otros (2024). *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*. https://arxiv.org/abs/2404.16130. ↩

.5. Microsoft, 2026. ↩

.6. Vincent A. Traag, Ludo Waltman y Nees Jan van Eck. (2019). *From Louvain to Leiden: guaranteeing well-connected communities*. Scientific Reports, 9, 5233. https://doi.org/10.1038/s41598-019-41695-z. ↩