Facsímil 04

La caja de herramientas

Capítulo 13

Lo que deberías saber: la caja de herramientas

686f6c61 · 2026

Capítulo 13: Lo que deberías saber: la caja de herramientas

Entrando en el cierre

Este facsímil empezó con una pregunta muy práctica: cuando un sistema de IA no hace lo que necesitas, ¿qué cambias exactamente?

Podrías cambiar el prompt. Podrías exigir una salida estructurada. Podrías añadir documentos con RAG. Podrías conectar una herramienta. Podrías elegir otro modelo. Podrías servirlo en local. Podrías usar embeddings, bases vectoriales, búsqueda híbrida, evaluación, GraphRAG o Text-to-SQL. La caja ya no está vacía.

Pero tener herramientas no significa tener criterio. Un equipo puede complicar una solución hasta que nadie la entiende. También puede quedarse corto y llamar “IA” a una demo que no soporta coste, permisos, datos reales ni evaluación. Este capítulo es una revisión activa: no busca que recuerdes nombres, sino que puedas justificar decisiones.

Si has entendido el facsímil, deberías poder mirar un caso y decir con calma: qué problema hay, qué intervención toca, qué contrato hace falta, qué métrica lo prueba, qué coste introduce y qué pasaría si mañana cambia el dato, el modelo o el usuario.

Este capítulo es la **brújula** del cierre: una revisión activa de todo lo construido, capítulo a capítulo, para poder justificar decisiones en vez de recordar nombres. Si el recap es una conversación técnica, su valor está en sostener cada afirmación con criterio: qué intervención toca, qué contrato hace falta y qué métrica lo prueba.

Fecha de corte y alcance

Fecha de corte: 26 de mayo de 2026.
Alcance: este cierre resume mecanismos estables del facsímil y referencias ya trabajadas: function calling, salidas estructuradas, prompt caching, model cards, LoRA, QLoRA, RAG, búsqueda vectorial, búsqueda híbrida, evaluación RAG, GraphRAG y Text-to-SQL.

Las herramientas concretas cambiarán. Los nombres de APIs, modelos, precios, runtimes y proveedores se moverán. El criterio que queremos conservar es más estable: separar entrada, contexto, herramienta, pesos, despliegue, evaluación y trazas.

Qué debería llevarse cada perfil

Este facsímil está escrito para gente curiosa con o sin perfil técnico. Eso no significa que todo el mundo tenga que llevarse lo mismo. Significa que cada persona debería poder salir con una forma más precisa de mirar un sistema de IA.

PERFIL	DEBERÍA PODER HACER AL TERMINAR ESTE FACSÍMIL	PREGUNTA QUE YA NO DEBERÍA ACEPTAR SIN MÁS
Ingeniería	Diseñar una arquitectura mínima con contratos, permisos, trazas y evaluación.	“¿Y si le ponemos un agente?”
Datos	Separar pregunta documental, búsqueda semántica, métrica, SQL y validación de resultados.	“¿El modelo ya sabe los datos?”
Producto	Decidir si una función merece prompt, RAG, tool, ajuste, modelo local o nada de IA.	“¿Podemos meter IA aquí?”
Docencia	Explicar cada herramienta con un ejemplo pequeño, una limitación y una prueba.	“¿Esto funciona porque lo he visto en una demo?”
Persona curiosa	Preguntar qué entra, qué sale, qué se mide, qué cuesta y qué pasa cuando no hay evidencia.	“¿La IA lo ha dicho, entonces será verdad?”

El punto común es el criterio. La persona técnica lo aplicará escribiendo código, definiendo contratos o levantando infraestructura. La persona no técnica lo aplicará haciendo mejores preguntas, detectando promesas débiles y pidiendo pruebas antes de confiar.

La brújula: qué quieres cambiar

La pregunta más importante del facsímil no es “¿qué herramienta está de moda?”. Es esta:

> ¿Dónde está el cuello del problema?

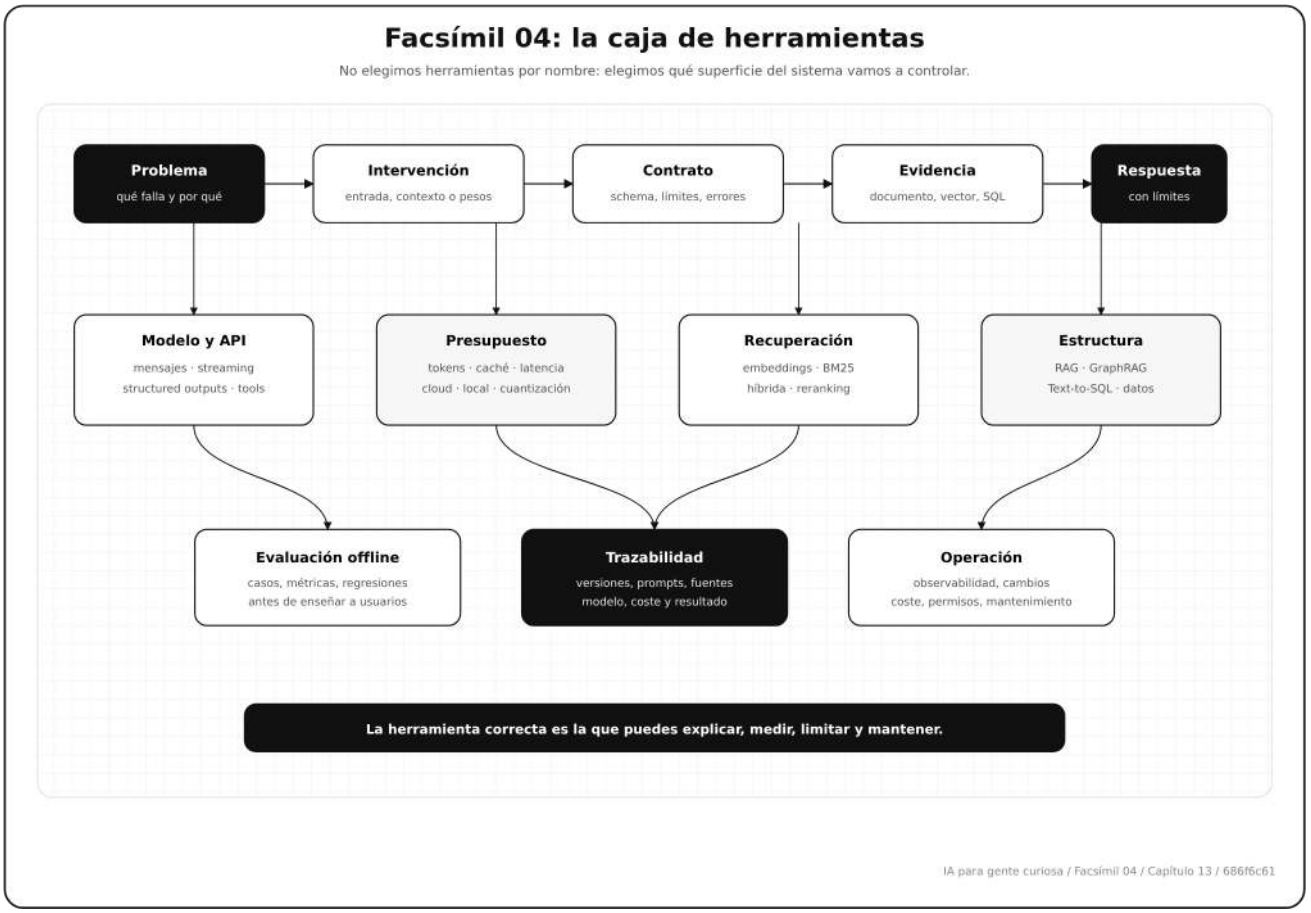
Cada intervención cambia una parte distinta del sistema. Si confundes la parte, puedes trabajar mucho y mejorar poco.

SI EL PROBLEMA ESTÁ EN...	INTERVENCIÓN NATURAL	QUÉ CAMBIA	QUÉ NO ARREGLA
Instrucción confusa	Prompt, ejemplos, formato	La entrada	Datos que el modelo no tiene.
Salida frágil	JSON schema, parser, validador	El contrato de salida	La calidad del contenido.
Conocimiento cambiante	RAG o herramienta de consulta	El contexto	El razonamiento sobre datos mal definidos.
Estado real	Tool/API/base de datos	La capacidad de actuar o consultar	Permisos, auditoría o validación.
Estilo repetido	Fine-tuning o LoRA	Parte del comportamiento	Información viva.
Coste o latencia	Modelo menor, caché, local, cuantización	Infraestructura y presupuesto	Preguntas mal formuladas.
Recuperación pobre	Embeddings, BM25, híbrida, reranking	Evidencia encontrada	Respuestas sin verificación.
Preguntas relacionales	GraphRAG o Text-to-SQL	Estructura consultable	Ambigüedad de negocio.

OpenAI documenta function calling como forma de describir herramientas con esquemas y recibir argumentos estructurados.[1] Las salidas estructuradas permiten exigir una forma de respuesta compatible con un esquema.[2] Esos mecanismos no hacen que el modelo “sepa más”; hacen que el sistema sea más gobernable.

La primera habilidad del facsímil es elegir la intervención pequeña que toca. La segunda es saber cuándo esa intervención ya no basta.

El mapa completo de la caja



El mapa tiene una idea central: las herramientas no están alineadas por glamour, sino por responsabilidad. Si una pieza no tiene contrato, métrica ni traza, todavía no es una pieza de ingeniería; es una promesa.

1. Elegir la intervención correcta

En el capítulo 01 aprendimos a no empezar por la herramienta. Antes de decir RAG, fine-tuning, modelo local o agente, hay que diagnosticar el tipo de fallo.

Para recordar. Prompt y ejemplos cambian la entrada. Structured outputs cambian el contrato. RAG cambia el contexto. Una tool cambia la capacidad de consultar o actuar. Fine-tuning y LoRA cambian parte del comportamiento aprendido.[3]

Caso cercano. Una universidad quiere un asistente que responda dudas de matrícula. Si falla el tono, quizá basta con ejemplos. Si falla porque desconoce normas nuevas, RAG. Si debe consultar expediente, tool. Si debe devolver JSON para abrir un ticket, structured outputs. Si siempre corrige con una rúbrica propia, puede aparecer un ajuste.

Vuelve al capítulo 01 si: no puedes rellenar una tabla de “cambia / sirve cuando / no sirve para” sin mirar apuntes.

---

2. APIs: mensajes, eventos y contratos

En el capítulo 02 dejamos de tratar la API como una caja negra. Una llamada moderna tiene mensajes, roles, instrucciones, entradas multimodales, parámetros, streaming, tools, esquemas de salida y manejo de errores.

Para recordar. Una API buena no es solo “envío prompt y recibo texto”. Es un contrato entre aplicación y modelo. La aplicación decide qué datos manda, qué formato exige, qué herramientas declara, cómo valida y qué hace si la respuesta no cumple.

PIEZA	PREGUNTA QUE RESPONDE	ERROR TÍPICO
Mensajes	¿Quién dice qué y con qué prioridad?	Mezclar instrucciones de sistema con texto de usuario.
Parámetros	¿Cuánta variación permito?	Tocar temperatura sin evaluación.
Streaming	¿Cómo llega la respuesta?	No diseñar cancelación ni estados parciales.
Tools	¿Qué puede consultar el sistema?	Dejar que el modelo decida permisos.
Structured outputs	¿Qué forma debe tener la salida?	Parsear texto libre con expresiones frágiles.

Caso cercano. Un asistente administrativo clasifica correos y crea tareas. No debería “redactar algo parecido a JSON”. Debería devolver campos obligatorios, categorías permitidas, confianza y razón breve. La aplicación valida antes de crear nada.

Vuelve al capítulo 02 si: no puedes diseñar una llamada completa con entrada, salida estructurada, streaming opcional, tool declarada y errores previstos.

---

3. Tokens, contexto y caché

En el capítulo 03 aprendimos que el contexto no es gratis. Cada tabla, documento, ejemplo, historial y política consume tokens. El coste no vive solo en la salida.

Para recordar. El contexto es memoria temporal, no memoria perfecta. La caché puede reducir coste si reutilizas prefijos estables, pero no arregla un prompt lleno de ruido. La documentación de prompt caching trata precisamente esta idea: reutilizar segmentos repetidos para optimizar coste y latencia cuando el patrón lo permite.[4]

Identidad de coste (no es un teorema). El coste de un turno es, sencillamente, la suma de sus partes. No predice un número exacto: sirve para ver dónde se va el dinero y qué recortar. Es la misma lógica del coste total de propiedad (TCO): un coste real no es solo el precio visible, sino la suma de todos los componentes que hay que sostener.[5]

$$C_{\text{turno}} = C_{\text{input}} + C_{\text{output}} + C_{\text{herramientas}} + C_{\text{observabilidad}}$$

SÍMBOLO	QUÉ SIGNIFICA	QUÉ REVISAR
C_{input}	Coste de contexto enviado.	Esquema, docs, ejemplos, historial.
C_{output}	Coste de tokens generados.	Respuestas largas, razonamiento, formato.
$C_{\text{herramientas}}$	Consultas externas.	Bases, vector stores, APIs.
$C_{\text{observabilidad}}$	Trazas y evaluación.	Logs, métricas, almacenamiento.

Caso cercano. Meter todo el manual de 300 páginas en cada llamada puede funcionar en una demo y fracasar en coste. Un sistema serio recupera lo relevante, cachea instrucciones estables y mide si el contexto añadido mejora la respuesta.

Vuelve al capítulo 03 si: no puedes explicar por qué más contexto puede empeorar una respuesta.

---

4. Model cards y elección de modelos

En el capítulo 04 aprendimos a leer una ficha de modelo como un expediente técnico. Nombre, licencia, parámetros, contexto, precisión, benchmarks, proveedor, plantilla de chat y formato de pesos no son adornos.

Para recordar. Una model card sirve para comparar promesas con restricciones. El trabajo original sobre model cards propuso documentar uso previsto, factores relevantes, métricas, datos de evaluación y consideraciones éticas de los modelos.[6]

TÉRMINO	QUÉ APORTA REALMENTE	PREGUNTA SANA
Parámetros totales	Capacidad almacenada aproximada.	¿Cuántos se activan por token?
Context length	Ventana máxima declarada.	¿Con qué coste y calidad real?
Tensor type	Precisión y formatos presentes.	¿Qué usa inferencia de verdad?
License	Condiciones de uso.	¿Permite mi caso concreto?
Benchmarks	Señales comparativas.	¿Se parecen a mi tarea?
Chat template	Contrato de entrada.	¿Estoy enviando mensajes como toca?

Caso cercano. Un modelo con ventana de contexto enorme puede parecer perfecto para contratos. Pero si recupera mal detalles de la mitad del documento, tarda demasiado o cuesta diez veces más, el número de contexto no resuelve el producto.

Vuelve al capítulo 04 si: no puedes explicar una model card sin convertirla en un catálogo de siglas.

---

5. Local, cuantización y dependencia operativa

En los capítulos 05 y 06 bajamos del modelo abstracto a la máquina real: memoria, VRAM, cuantización, runtimes, cloud, local, alquiler de GPUs, latencia, privacidad, coste y mantenimiento.

Para recordar. Local no significa gratis. Cloud no significa simple. Un modelo local necesita formato, runtime, memoria, CPU/GPU, plantilla, servidor, límites, observabilidad y actualización. Un modelo cloud necesita contrato de datos, precios, latencia, región, cuotas, cambios de versión y fallback.

DECISIÓN	GANAS	PAGAS
API cloud	Facilidad, escalado, modelos potentes.	Coste variable, dependencia, políticas de datos.
Local en portátil	Control y aprendizaje.	Menos potencia, setup, rendimiento limitado.
Servidor local	Control operativo.	GPU, mantenimiento, scheduling, monitorización.
GPU alquilada	Capacidad temporal.	Gestión de imágenes, datos, apagado y coste/hora.
Cuantización	Menos memoria y coste.	Posible pérdida de calidad y estabilidad.

Caso cercano. Si una herramienta interna procesa expedientes sensibles y tiene pocos usuarios, local puede ser razonable. Si atiende miles de solicitudes variables y necesita el mejor modelo disponible, cloud puede ganar. Si la carga es estable y grande, quizá compensa un servidor propio. No hay respuesta universal: hay presupuesto y restricciones.

Vuelve a los capítulos 05 y 06 si: no puedes estimar memoria, latencia y coste antes de instalar nada.

---

6. Embeddings y búsqueda semántica

En el capítulo 07 convertimos texto en vectores. Un embedding no es una etiqueta. Es una posición en un espacio de dimensiones donde podemos medir cercanía.

Para recordar. La dimensión de un embedding no es “una idea humana”, sino un eje numérico aprendido. El significado aparece por la combinación de muchos ejes. Comparar embeddings permite buscar por parecido semántico, no solo por palabras exactas.

La similitud coseno sigue siendo la fórmula mental básica. No es un invento reciente: es la medida del modelo de espacio vectorial, propuesto para recuperación de información en los años setenta, donde documentos y consultas se representan como vectores y su relevancia se estima por el ángulo entre ellos.[7]

$$\mathrm{sim}(q, d) = \frac{q \cdot d}{\|q\|\|d\|}$$

SÍMBOLO	QUÉ SIGNIFICA	LECTURA
q	Vector de la consulta.	Lo que pregunta la persona en números.
d	Vector del documento.	Un fragmento convertido en números.
$q \cdot d$	Producto escalar.	Cuánto apuntan en dirección parecida.
$\|q\|$, $\|d\|$	Normas.	Tamaño de cada vector.

El coseno mide dirección, no magnitud: dos textos sobre el mismo tema apuntan al mismo sitio aunque uno sea más largo. Por eso, cuando los vectores están normalizados a norma 1, ordenar por coseno equivale a ordenar por producto escalar, y eso es justo lo que explotan los índices vectoriales para buscar rápido.

FAISS mostró cómo buscar de forma eficiente entre miles de millones de vectores en GPU.[8] HNSW es una estructura de grafos aproximados muy usada para búsqueda vectorial eficiente.[9]

Caso cercano. “Cómo pedir vacaciones” puede recuperar “procedimiento de ausencia laboral” aunque no comparta palabras. Eso es buenísimo. Pero “parecido” no significa “suficiente”: todavía hay que validar fecha, versión, permisos y respuesta.

Vuelve al capítulo 07 si: no puedes explicar qué es una dimensión y por qué dos textos cercanos pueden no responder la misma pregunta.

---

7. Bases vectoriales y búsqueda híbrida

En el capítulo 08 vimos que buscar no es solo guardar embeddings. Hay índices, filtros, metadatos, fragmentos, reranking y búsqueda híbrida.

Para recordar. BM25 sigue siendo fuerte para coincidencia léxica y términos exactos.[10] Reciprocal Rank Fusion permite combinar rankings de distintas búsquedas sin convertir todo a una misma escala.[11]

PIEZA	QUÉ APORTA	QUÉ PUEDE FALLAR
Chunking	Divide documentos recuperables.	Cortar contexto necesario.
Embeddings	Busca por significado.	Perder números, códigos o nombres exactos.
BM25	Busca por términos.	No captar paráfrasis.
Filtros	Acotan por permisos, fecha, tipo.	Filtrar demasiado o tarde.
Reranking	Reordena candidatos.	Añadir latencia sin mejorar.
Metadatos	Dan contexto operativo.	Estar incompletos o desactualizados.

Caso cercano. Si buscas “artículo 17.3” necesitas texto exacto. Si buscas “cómo se solicita una revisión”, necesitas semántica. Si necesitas ambas cosas, híbrida.

Vuelve al capítulo 08 si: no puedes diseñar un índice con texto, metadatos, filtros y estrategia de recuperación.

---

8. RAG básico: evidencia antes que respuesta

En el capítulo 09 construimos RAG: recuperar información externa, pasarla al modelo y responder con evidencia. RAG no es “hacer que el modelo sepa más”. Es darle contexto verificable en el momento de responder.

Para recordar. Un RAG mínimo necesita colección, limpieza, chunking, embeddings o búsqueda léxica, recuperación, contexto, generación, citas y abstención. El trabajo original de RAG combinó recuperación con generación para tareas intensivas en conocimiento.[12]

Caso cercano. Un asistente de normativa responde mejor si cita el documento vigente. Pero si recupera un párrafo antiguo, la redacción puede ser impecable y la respuesta falsa. RAG no elimina el problema: lo hace medible.

Vuelve al capítulo 09 si: no puedes explicar la diferencia entre entrenar conocimiento y recuperar evidencia.

---

9. Evaluar RAG y no fiarse de la demo

En el capítulo 10 dejamos claro que una demo que responde bien tres veces no es una evaluación. Hay que medir retrieval, contexto, groundedness, abstención, latencia, coste y regresiones.

Para recordar. Evaluar RAG tiene varias capas:

CAPA	PREGUNTA
Recuperación	¿Aparece la fuente correcta entre los candidatos?
Contexto	¿Lo recuperado contiene evidencia suficiente?
Generación	¿La respuesta se apoya en esa evidencia?
Abstención	¿Sabe parar cuando no hay base?
Operación	¿Cuánto cuesta y tarda en casos reales?

RAGAS popularizó métricas orientadas a RAG como faithfulness, answer relevancy, context precision y context recall.[13]

Caso cercano. Si tu sistema responde “no lo sé” ante una pregunta que sí tiene respuesta, falta recall. Si responde con seguridad sin evidencia, falta abstención o groundedness. Si acierta pero tarda doce segundos, falta operación.

Vuelve al capítulo 10 si: no puedes proponer un dataset de evaluación con preguntas, fuentes esperadas, respuestas aceptables y casos donde debe abstenerse.

---

10. Agentic RAG y GraphRAG: complicar con permiso

En el capítulo 11 vimos cuándo un RAG fijo se queda corto: preguntas compuestas, rutas de búsqueda, fuentes múltiples, relaciones globales o necesidad de comprobar si la evidencia basta.

Para recordar. Agentic RAG añade decisiones de flujo: descomponer, elegir fuente, recuperar de nuevo, revisar evidencia, usar herramientas. GraphRAG añade estructura de entidades y relaciones para responder preguntas locales o globales sobre corpus complejos. Microsoft GraphRAG propuso construir grafos de entidades y resúmenes de comunidades para preguntas de comprensión global.[14]

SI NECESITAS...	PUEDE TENER SENTIDO
Comparar varias fuentes	Query decomposition o multi-query.
Buscar con distintas estrategias	Router controlado.
Entender relaciones entre entidades	GraphRAG o grafo de conocimiento.
Revisar evidencia insuficiente	Corrective RAG o validación previa.
Limitar coste	Presupuesto de pasos y trazas.

Caso cercano. “Qué departamentos aparecen conectados a quejas sobre becas y retrasos de pago” no se resuelve igual que “cuál es el plazo de becas”. La primera pide patrones y relaciones. La segunda pide recuperar una fuente concreta.

Vuelve al capítulo 11 si: no puedes explicar qué complejidad pagas cuando añades pasos agentic.

---

11. Text-to-SQL y herramientas de datos

En el capítulo 12 cruzamos otra frontera: preguntas que no se contestan con documentos, sino con datos. Ahí no basta con recuperar párrafos. Hay que consultar tablas, esquemas, permisos y métricas.

Para recordar. Text-to-SQL traduce una pregunta humana a SQL controlado. No es acceso libre a la base de datos. Es una cadena: intención, esquema, semántica, SQL candidato, validación, plan, ejecución limitada, resultado y traza. Spider ayudó a medir Text-to-SQL con bases nuevas y consultas complejas.[15]

PREGUNTA	RIESGO SI LO SIMPLIFICAS
“Alumnos con pagos pendientes”	Contar pagos en vez de alumnos.
“Ingresos de marzo”	Usar fecha de creación en vez de fecha de pago.
“Top campus”	No definir si top es por alumnos, importe o incidencias.
“Datos por titulación”	Exponer columnas innecesarias.

Caso cercano. Un `JOIN` mal hecho puede duplicar filas y devolver una cifra convincente pero falsa. En Text-to-SQL, la sintaxis correcta es solo el comienzo. La semántica y la cardinalidad mandan.

Vuelve al capítulo 12 si: no puedes explicar por qué `COUNT(*)` y `COUNT(DISTINCT alumno_id)` responden preguntas distintas.

La matriz de decisión final

Esta matriz es una forma de hacer arquitectura sin postureo. Primero identifica el tipo de necesidad; después elige la herramienta más pequeña que cubre el caso.

NECESIDAD REAL	PRIMER INTENTO RAZONABLE	ESCALARÍA A...	SEÑAL DE QUE TE ESTÁS PASANDO
Mejor tono o formato	Prompt + ejemplos	Structured outputs	Cambiar pesos sin evaluar prompts.
JSON fiable	Schema + validador	Tool contract	Parsear texto libre.
Responder sobre documentos	RAG básico	Agentic RAG	Usar agente para una FAQ simple.
Encontrar documentos	Híbrida + filtros	Reranking	Embeddings sin metadatos.
Consultar datos vivos	Tool o Text-to-SQL	Semantic layer	Dejar SQL libre sin permisos.
Reducir coste	Modelo menor + caché	Cuantización/local	Recortar contexto sin medir calidad.
Controlar despliegue	API cloud con contratos	Servidor propio	Montar GPUs sin carga estable.
Dominio repetido	Plantillas + evals	LoRA/fine-tuning	Ajustar pesos para conocimiento cambiante.
Relaciones globales	RAG + metadatos	GraphRAG	Construir grafo sin preguntas de competencia.

La matriz no decide por ti. Te obliga a decir qué estás comprando con cada capa nueva. Esa frase, “qué estoy comprando”, es una de las mejores defensas contra arquitecturas hinchadas.

Mini casos con solución

Una buena recapitulación no solo pregunta “¿lo entendiste?”. Te pone delante situaciones pequeñas y obliga a decidir. Estas son decisiones de bolsillo, pero condensan casi todo el facsímil.

CASO	DECISIÓN RAZONABLE	POR QUÉ
Una FAQ interna cambia cada mes y el asistente responde con información antigua.	RAG básico con fuentes versionadas, citas y evaluación.	El problema es conocimiento cambiante, no estilo del modelo.
Un clasificador de tickets devuelve texto libre y rompe el flujo de soporte.	Structured outputs con catálogo cerrado y validador.	El fallo está en el contrato de salida, no en saber más contenido.
Una herramienta debe decir cuántos pagos pendientes hay por campus.	Tool de datos o Text-to-SQL con permisos y métricas definidas.	La respuesta vive en tablas, no en documentos parecidos.
Un RAG recupera fragmentos relacionados pero no la norma exacta.	Búsqueda híbrida, filtros por fecha/tipo y quizá reranking.	El problema está en retrieval, no en generación.
El sistema acierta pero tarda demasiado y cuesta demasiado por consulta.	Reducir contexto, cachear prefijos, probar modelo menor o cuantización.	El cuello está en presupuesto operativo.
El equipo quiere fine-tuning para añadir una política que cambia cada semana.	No ajustar pesos; usar RAG o tool.	El conocimiento vivo debe vivir fuera del modelo.
La pregunta exige comparar documentos, entidades y patrones de varias fuentes.	Agentic RAG controlado o GraphRAG si las relaciones importan.	La complejidad se justifica si la pregunta necesita pasos o estructura.

Ahora lo mismo, pero con tres decisiones desarrolladas.

Caso 1: “Tenemos una FAQ interna que cambia cada mes”.
No empezaría por fine-tuning. Cambiar pesos para información viva suele crear deuda: cada cambio exige nuevo ajuste, evaluación y despliegue. Empezaría por RAG básico: documentos versionados, chunking razonable, búsqueda híbrida si hay códigos o artículos, citas visibles y casos donde el sistema debe abstenerse. La métrica mínima sería: ¿recupera la fuente vigente?, ¿cita bien?, ¿responde solo con esa evidencia?

Caso 2: “Queremos consultar pagos pendientes”.
No usaría RAG como primera opción. Un documento puede explicar qué es un pago pendiente, pero el número está en una base de datos. Haría una tool o Text-to-SQL

controlado: rol de usuario, tablas permitidas, métrica definida, SQL validado, límite de filas, query plan y traza. La pregunta clave sería: ¿estoy contando pagos, alumnos o expedientes?

Caso 3: “El RAG responde bonito pero cita mal”.
No tocaría temperatura ni modelo todavía. Mediría recuperación antes: recall de fuentes esperadas, precisión de contexto, frescura de documentos, metadatos, filtros y reranking. Si la evidencia correcta no entra en el contexto, el generador no puede arreglarlo de forma fiable. Si la evidencia entra pero la respuesta no se apoya en ella, entonces revisaría prompt, formato de citas y evaluación de groundedness.

La idea de estos casos no es memorizar respuestas. Es aprender el gesto mental: diagnosticar la pieza que falla antes de cambiar la herramienta.

Cuándo no usar IA

Una caja de herramientas seria también incluye la opción de no sacar ninguna herramienta de IA. Hay problemas que se resuelven mejor con una regla, una consulta, un formulario, una validación clásica o una interfaz más clara.

SITUACIÓN	MEJOR PRIMERA OPCIÓN	POR QUÉ
Categorías cerradas y reglas simples.	`if`, reglas o tabla de decisión.	Más barato, explicable y determinista.
Cálculo exacto.	Código o SQL.	Un modelo generativo no debería inventar aritmética.
Flujo con permisos estrictos.	API con roles y validación.	El permiso no debe depender de texto.
Información ya estructurada.	Consulta directa o dashboard.	No hace falta traducir a lenguaje natural si el informe basta.
Tarea muy sensible y sin evaluación.	Esperar, medir o rediseñar.	No se despliega lo que no se puede comprobar.
Problema mal definido.	Taller de requisitos.	La IA no arregla una pregunta que nadie entiende.

Esto no va contra la IA. Va a favor de usarla bien. A veces la decisión más profesional es decir: aquí no hace falta un modelo; hace falta una regla clara, una tabla limpia o una conversación de producto.

El cálculo que deberías hacer antes de construir

Antes de escribir código, puedes estimar si una solución tiene sentido.

Identidad de decisión (cualitativa, no es un teorema). La utilidad neta es lo que queda al restar los costes al valor que aporta. Es una disciplina mental para discutir, no una fórmula para sacar decimales. Reproduce la estructura del análisis coste-beneficio: una intervención merece la pena cuando el beneficio neto (beneficios menos costes, incluido el riesgo) es positivo frente a la alternativa.[16]

$$U = Q - (C_{\text{tokens}} + C_{\text{latencia}} + C_{\text{mantenimiento}} + C_{\text{riesgo}})$$

SÍMBOLO	QUÉ SIGNIFICA	CÓMO SE OBSERVA
U	Utilidad neta de la solución.	Valor práctico después de costes.
Q	Calidad útil para la tarea.	Aciertos, groundedness, satisfacción, ahorro real.
C_{tokens}	Coste de entrada y salida.	Tokens, caché, modelo elegido.
C_{latencia}	Tiempo que tarda.	TTFT, P95, tiempo total.
$C_{\text{mantenimiento}}$	Trabajo de sostenerlo.	Datos, índices, evals, versiones.
C_{riesgo}	Impacto de fallos.	Permisos, datos sensibles, decisiones críticas.

No es una fórmula para sacar decimales. Es una disciplina mental. Si una herramienta sube Q un poco pero dispara mantenimiento y latencia, quizá no compensa. Si baja coste sin destruir calidad, merece atención. Si reduce riesgo aunque cueste algo más, puede ser la decisión correcta.

El lugar donde una demo se vuelve discutible

Una demo sirve para ver una posibilidad. Un laboratorio sirve para decidir si esa posibilidad aguanta un poco de realidad.

En este facsímil hemos hablado de APIs, modelos locales, tokens, costes, embeddings, RAG, evaluación, GraphRAG y Text-to-SQL. Todo eso puede quedarse en palabras si no lo llevamos a una mesa de trabajo mínima: datos pequeños, código ejecutable, métricas claras, trazas legibles y una decisión final.

Esta segunda parte del cierre es ese puente. No vamos a montar una plataforma industrial. Vamos a construir lo justo para que otra persona pueda ejecutar, revisar, criticar y mejorar lo que hicimos. Ese es el gesto profesional: no pedir confianza; dejar evidencia.

Qué no es un laboratorio

Un laboratorio no es un notebook que funciona una vez en tu máquina y queda abandonado. Tampoco es una captura bonita de una respuesta acertada. Y no es una colección de librerías instaladas sin una pregunta clara.

Un laboratorio tampoco sustituye a producción. En producción aparecen permisos reales, usuarios, colas, coste variable, cambios de datos y mantenimiento. El laboratorio no pretende resolver todo eso. Pretende descubrir antes qué merece pasar a una fase más seria.

Si no sabes qué pregunta estás probando, qué métrica mirarás y qué harás si sale mal, todavía no tienes laboratorio. Tienes una exploración. Puede ser útil, pero no permite decidir.

Qué sí debería dejar

Un laboratorio mínimo debe dejar cinco artefactos:

ARTEFACTO	QUÉ CONTIENE	POR QUÉ IMPORTA
Dataset	Casos de prueba con respuesta o fuente esperada.	Permite repetir la evaluación.
Runner	Código que ejecuta el sistema sobre esos casos.	Evita evaluar a mano caso por caso.
Métricas	Números que resumen el comportamiento.	Permite comparar versiones.
Trazas	Pasos internos de cada ejecución.	Permite depurar por qué falló.
Decisión	Pasar, parar, cambiar o medir más.	Evita terminar con “parece que va bien”.

Los notebooks son útiles porque mezclan explicación, código y resultados. El formato Jupyter se basa en documentos JSON con celdas, salidas y metadatos, lo que permite guardar no solo código, sino también el contexto de ejecución.[17] Esa flexibilidad es estupenda para aprender, pero también exige disciplina: fijar datos, ordenar celdas, limpiar salidas innecesarias y convertir lo aprendido en scripts o tests cuando el experimento empieza a importar.

En observabilidad, OpenTelemetry describe una traza como una operación formada por spans, donde cada span representa una unidad de trabajo con contexto y atributos.[18] Nosotros haremos una versión casera: una lista de pasos con nombre, entrada, salida y metadatos. No será una herramienta de producción, pero enseñará la forma mental correcta.

Estado de herramientas con fecha de corte

Fecha de corte: 26 de mayo de 2026.
Fuentes consultadas ese día: documentación de Jupyter nbformat, OpenTelemetry Tracing API, OpenAI Graders, Ragas metrics, LangSmith RAG evaluation y Arize Phoenix evaluation.

OpenAI documenta graders como evaluadores usados en evals y fine-tuning, incluyendo validación de graders y ejemplos de evaluadores basados en modelos.[19] Ragas organiza métricas para aplicaciones RAG, entre ellas context precision y faithfulness.[20] LangSmith estructura la evaluación de RAG alrededor de corrección, relevancia, groundedness y relevancia de documentos.[21]

Phoenix separa evaluación de retrieval y evaluación de respuesta, y permite trabajar con trazas para analizar qué documentos se recuperaron y cómo se generó la respuesta.[22] Su documentación de evaluación distingue evaluadores deterministas y evaluadores con modelo, y los presenta como forma de detectar regresiones y comparar cambios.[23]

La lección estable no depende de una marca concreta: una evaluación útil separa dataset, ejecución, métrica, trazas y decisión.

Cómo medir de verdad: métricas, jueces y comparación

Métricas como Hit@1 y MRR bastan para empezar, pero un ingeniero de IA necesita un repertorio más amplio y, sobre todo, saber cuándo cada una miente. Esta sección reúne lo que de verdad usa la evaluación seria de sistemas con recuperación y generación.

Cuando hay varios documentos relevantes: nDCG

Hit@1 y MRR suponen una sola fuente correcta. En cuanto una pregunta admite varios documentos relevantes con distinta importancia, la métrica estándar es el nDCG (*normalized discounted cumulative gain*), que premia colocar lo más relevante arriba y aplica un descuento logarítmico a las posiciones siguientes.[24]

$$\mathrm{DCG@k} = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)} \qquad \mathrm{nDCG@k} = \frac{\mathrm{DCG@k}}{\mathrm{IDCG@k}}$$

SÍMBOLO	QUÉ SIGNIFICA	LECTURA
rel_i	Relevancia del documento en la posición i	0 = irrelevante, 2 = muy relevante.
$\log_2(i+1)$	Descuento por posición	Cuanto más abajo, menos vale.
$\mathrm{IDCG@k}$	DCG del ranking ideal	El mejor orden posible.
$\mathrm{nDCG@k}$	DCG normalizado al rango $[0, 1]$	1.0 = orden perfecto.

La figura muestra el mismo ranking puntuado de tres formas. Hit@1 y MRR se quedan en 1.0 porque solo miran el primer acierto; nDCG baja a 0.94 porque dos documentos relevantes quedaron por debajo de documentos irrelevantes. Elegir métrica es elegir qué fallos te importan.

Qué es «groundedness» exactamente

Usamos «groundedness» (o «faithfulness») como si fuera obvio, pero tiene una definición operativa: una respuesta está fundamentada si **cada afirmación** que contiene puede deducirse de la evidencia recuperada. RAGAS la calcula descomponiendo la respuesta en afirmaciones atómicas y midiendo qué fracción está respaldada por el contexto.[25]

$$\text{faithfulness} = \frac{\text{afirmaciones respaldadas por el contexto}}{\text{afirmaciones totales de la respuesta}}$$

Una respuesta puede ser correcta y aun así poco fundamentada: acertó por conocimiento del modelo, no por la evidencia. En un sistema serio eso es un fallo, porque si mañana cambia el documento la respuesta deja de ser fiable sin avisar. Por eso groundedness se mide aparte de la corrección.

Cuando no hay métrica exacta: el modelo como juez

Muchas cualidades (claridad, tono, si una respuesta «responde de verdad») no tienen fórmula. La práctica habitual es usar otro modelo como evaluador: se le dan la respuesta, el contexto y una rúbrica, y devuelve una puntuación con su razón. Es la técnica de *LLM-as-judge*, que correlaciona razonablemente con el juicio humano cuando la rúbrica es clara.[26] El método G-Eval formalizó pedir al modelo una puntuación guiada por criterios y mostró mejor alineación con humanos que las métricas automáticas clásicas.[27]

Pero un juez-modelo tiene sesgos conocidos que hay que controlar:

SESGO	QUÉ HACE	MITIGACIÓN
Posición	Prefiere la primera (o la segunda) opción en comparaciones.	Alternar el orden y promediar.
Verbosidad	Premia respuestas largas aunque no sean mejores.	Penalizar longitud; pedir concisión en la rúbrica.
Autopreferencia	Favorece textos de su propia familia de modelos.	Usar un juez distinto del modelo evaluado.
Formato	Se deja llevar por listas, negritas o seguridad aparente.	Rúbrica que separa fondo de forma.

La regla: el juez-modelo es útil para escalar la evaluación, no para tener la última palabra. Calíbralo contra un pequeño set anotado a mano y vuelve a revisarlo cuando cambies de modelo.

¿Mi cambio ayudó de verdad? Comparar dos variantes

Medir una variante no basta; casi siempre quieres saber si B es mejor que A. Con datasets pequeños, una diferencia de «0.80 frente a 0.85» puede ser ruido. Para datos emparejados (los mismos casos en A y B), la prueba estándar es la de McNemar, que mira solo los casos donde una variante acierta y la otra falla.[28] Cuando la métrica no es binaria, el bootstrap (remuestrear los casos y mirar la distribución de la diferencia) da un intervalo de confianza sin suponer una forma concreta.[29]

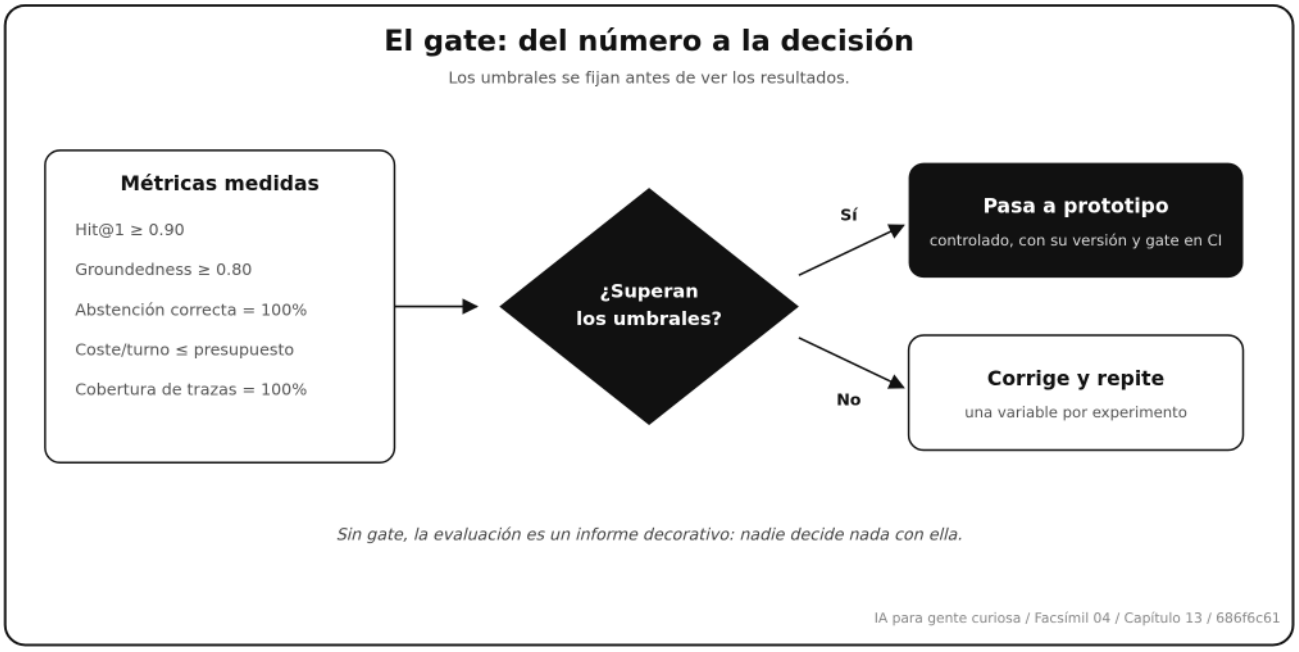
La idea práctica: no celebres una mejora sin preguntarte si sobreviviría a repetir el experimento. Una diferencia pequeña sobre pocos casos casi nunca sobrevive.

El enemigo silencioso: contaminación del set

Una evaluación solo vale si sus casos son nuevos para el sistema. Si las preguntas (o sus respuestas) estaban en los datos de entrenamiento o ya venían en el contexto, la métrica mide memoria, no capacidad. Es un problema real y creciente con modelos entrenados sobre casi toda la web.[30] En tu laboratorio: usa casos propios, fecha el dataset y desconfía de un 100% que llega demasiado fácil.

Del número a la decisión: el gate

Nada de lo anterior sirve si no se traduce en una decisión. Un *gate* convierte métricas en un sí o no automático: define umbrales mínimos **antes** de mirar los resultados y deja que la evaluación decida si una variante avanza o vuelve a la mesa de trabajo. Definir el umbral después de ver el número es engañarse.



Presupuesto con números: latencia y capacidad

Una última pieza que separa demo de sistema: estimar coste y latencia antes de prometer nada. La latencia de una respuesta generativa se descompone en el tiempo hasta el primer token (TTFT) y el tiempo entre tokens (TPOT):

$$L \approx \mathrm{TTFT} + (N_{\mathrm{out}} - 1) \cdot \mathrm{TPOT}$$

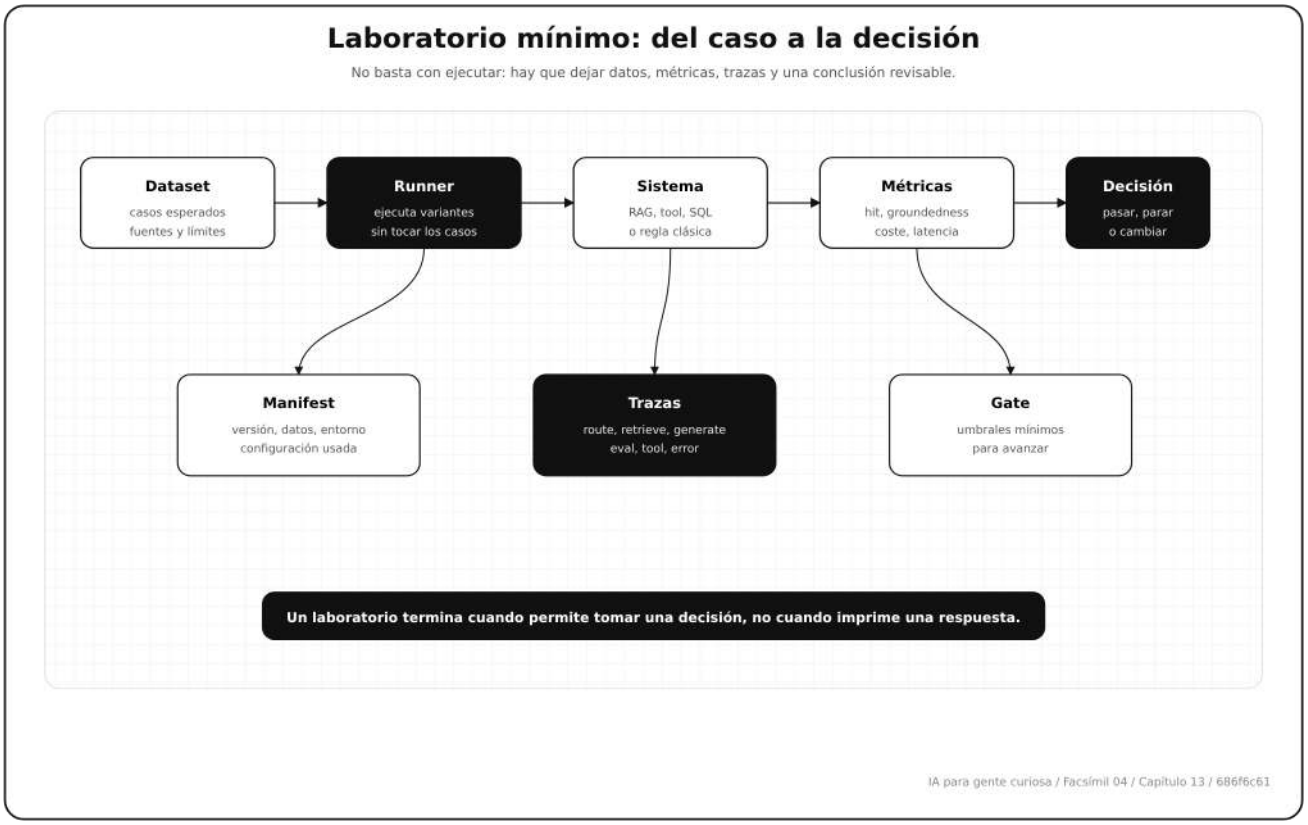
Para dimensionar capacidad sirve la ley de Little, un resultado clásico de teoría de colas: el número medio de peticiones dentro del sistema es la tasa de llegada por el tiempo medio que pasa cada una.[31]

$$L = \lambda \cdot W$$

SÍMBOLO	QUÉ SIGNIFICA	EJEMPLO
L	Peticiones simultáneas en el sistema	Concurrencia media.
λ	Tasa de llegada	5 peticiones por segundo.
W	Tiempo medio en el sistema	2 s por petición.

Con $\lambda = 5$ y $W = 2$, el sistema sostiene de media $L = 10$ peticiones a la vez. Si tu servidor no aguanta diez en vuelo, o bajas la latencia W o se forma cola y la latencia percibida sube. Estimar esto antes evita la sorpresa de «funcionaba en la demo y se cae con diez usuarios».

Anatomía de un laboratorio mínimo



La figura puede leerse de izquierda a derecha: parto de casos, ejecuto variantes, observo resultados y decido. Debajo aparecen las tres piezas que suelen faltar en las demos: manifest, trazas y gate.

En el día a día

En el trabajo real, este cierre no aparece como un examen, sino como una conversación. Alguien propone «metamos un agente», «conectemos RAG» o «esto lo resuelve un modelo». La caja de herramientas que has recorrido sirve para responder con calma: qué problema hay, qué intervención cambia algo, qué contrato hace falta, qué métrica lo prueba y qué pasa si falla.

El laboratorio es el otro lado de esa conversación. Cuando alguien afirma que «funciona», la pregunta profesional no es «¿lo has visto?», sino «¿puedo ejecutarlo, medirlo y leer la traza?». En el día a día eso se traduce en gestos pequeños: un dataset de diez casos, un runner que no toca los casos, una métrica por capa y una decisión escrita. No es burocracia; es lo que separa una demo de algo que otra persona puede sostener.

Verás el patrón en revisiones de arquitectura, en estimaciones de coste antes de construir, en la decisión de no usar IA cuando una regla basta y en la defensa de un sistema delante de alguien que pregunta «¿y si mañana cambia el dato?».

Por qué debería importarte

Porque la diferencia entre un equipo que improvisa y uno que decide está casi siempre aquí. El primero acumula demos que nadie puede reproducir; el segundo deja evidencia: contratos, trazas, evaluaciones y una conclusión defendible. Con el tiempo, el segundo avanza más rápido, no más despacio, porque no rehace lo que no entiende.

También importa para ti como persona curiosa. Saber leer un sistema (ver dónde falla, qué intervención toca, qué cuesta) es una forma de autonomía: la IA deja de ser magia y pasa a ser algo que puedes interrogar. Y cuando el siguiente facsímil hable de agentes y orquestación, esa costumbre de exigir límites, permisos y trazas será justo lo que evite que un sistema más capaz se vuelva, simplemente, más difícil de depurar.

Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Empezar por la herramienta	«Metamos RAG» no dice qué fallo estás resolviendo.	Escribir primero qué cambia: entrada, contexto, herramienta, pesos o despliegue.
Confundir demo con sistema	Una respuesta buena no prueba permisos, coste, latencia ni regresiones.	Exigir dataset mínimo, trazas y casos donde debe abstenerse.
Confundir notebook con laboratorio	Un notebook puede ejecutar una vez y no dejar decisión ni reproducibilidad.	Añadir dataset, métricas, manifest y conclusión.
Medir solo la respuesta final	No sabes si falló retrieval, routing, herramienta o generación.	Medir por capas y guardar spans.
Meter más contexto sin medir	Puede subir coste y ruido sin mejorar groundedness.	Medir recall, precisión de contexto y calidad de respuesta por separado.
Cambiar prompt y dataset a la vez	Si mejora, no sabes qué lo causó.	Fijar dataset y cambiar una variable por experimento.
Complicar antes de cerrar lo básico	Agentic RAG o GraphRAG pueden esconder problemas de chunking, filtros o definición.	Probar primero el flujo más simple que pueda evaluarse.
No poner gate	La evaluación queda como un informe decorativo.	Definir umbrales mínimos antes de mirar resultados.
Olvidar mantenimiento	Índices, prompts, modelos, esquemas y datos cambian.	Nombrar propietario, versión y prueba de regresión de cada pieza.
No diseñar salida de fallo	El sistema acaba respondiendo incluso cuando no sabe.	Definir abstención, aclaración y escalado antes de producción.
No contemplar la opción sin IA	Algunas tareas se resuelven mejor con reglas, SQL o una interfaz más clara.	Preguntar siempre qué gana el modelo frente a una solución clásica.

Cuadernos para practicar

A lo largo del facsímil has montado y medido sistemas con herramientas; estos cuadernos te dejan tocar las piezas sueltas. Son *notebooks* que se abren en Google Colab —gratis, en el navegador— o te puedes descargar. Cada uno, explicado paso a paso, con salidas reales, y

anclado al capítulo del que sale.

Tokens, contexto y coste: la factura de un LLM

Qué practicas: contar tokens reales y estimar lo que cuesta procesar un volumen de trabajo. **Dónde encaja:** capítulo 3 (tokens, coste, contexto y caché). **Qué necesitas:** un navegador. Corre en CPU; sin claves.

Mides en tokens reales lo que procesas y pagas. Primero ves que un token no es una palabra ni una letra: «antidisestablishmentarianism» son 6 tokens, tres emojis son 7, y el español gasta más tokens por palabra que el inglés (y por tanto más dinero). Luego calculas el caso del día a día: clasificar diez mil tickets de soporte sale por unos **0,75 dólares** con precios de ejemplo, y ves cuánto ahorra cachear las instrucciones fijas. Por último traduces la ventana de contexto a algo tangible: 128.000 tokens son unas **176 páginas**.

La primera sorpresa de poner un LLM en producción es la factura. Aquí aprendes a estimarla antes de gastar.

► Abrir en Google Colab **↓ Descargar el cuaderno (.ipynb)**

Salidas estructuradas: que el modelo rellene tu formulario

Qué practicas: exigir al modelo un formato fijo y validarlo automáticamente. **Dónde encaja:** capítulo 2 (APIs de modelos: mensajes, *streaming* y salidas estructuradas). **Qué necesitas:** un navegador. Corre en CPU; sin claves (se simulan las respuestas).

Dejas de tratar la respuesta del modelo como un texto del que sacar datos a mano y le exiges un esquema: una factura con proveedor, importe positivo, fecha y concepto. Una respuesta buena se valida y se convierte en un dato con tipos —la fecha es una fecha de verdad, con la que ya puedes operar—. Una mala (el importe como texto «cuarenta», un importe negativo, un campo que falta) salta en el sitio, con un mensaje claro de qué falló, en vez de colarse en tu base de datos y reventar tres pasos después. Y montas el bucle de reintento que usan los sistemas serios: devolverle el error al modelo para que corrija.

Si vas a meter lo que dice el modelo en una base de datos o en una decisión, no puedes permitirte un «más o menos».

► Abrir en Google Colab **↓ Descargar el cuaderno (.ipynb)**

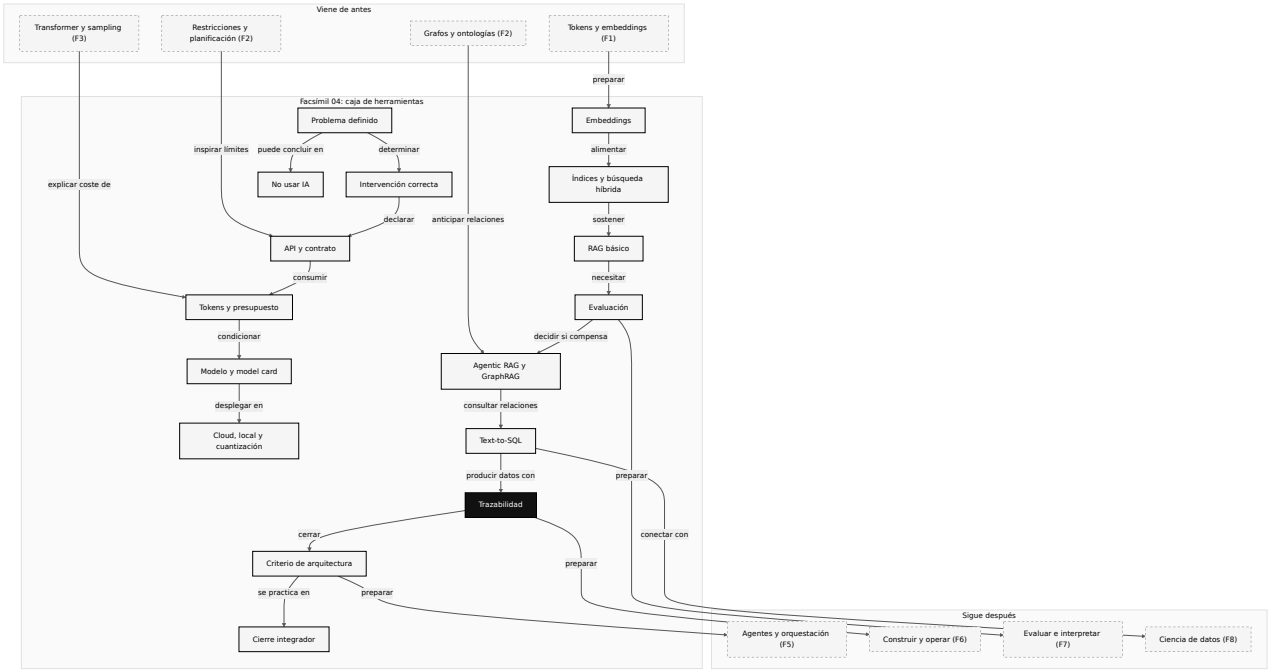
Del cierre al siguiente facsímil

El cierre natural de este facsímil no es “ya sabemos usar herramientas”. Es más preciso: ya sabemos **poner herramientas bajo control**.

El criterio de este cierre debería poder demostrarse con algo concreto: una evaluación reproducible, una traza que explique qué ocurrió, una decisión de arquitectura defendible y una salida que no dependa de buena suerte. Si una solución no deja rastro, no tiene contrato y no se puede evaluar, todavía pertenece al terreno de la demo.

El facsímil 05 dará el siguiente paso: agentes y orquestación. Ahí las herramientas dejan de ser piezas aisladas y empiezan a coordinarse. Por eso este cierre insiste tanto en límites, permisos, trazas, costes y abstención. Un agente sin esas piezas no es más capaz: solo es más difícil de depurar.

Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Intervención	Cambio concreto sobre entrada, contexto, herramienta, pesos o infraestructura.
Contrato	Especificación de entrada, salida, límites, errores y trazas.
Superficie de control	Lugar del sistema donde puedes limitar, validar o medir.
Presupuesto operativo	Límite de coste, tokens, latencia, memoria, pasos o consultas.
Evidencia	Información recuperada o calculada que sostiene una respuesta.
Abstención	Decisión de no responder cuando no hay base suficiente.
Trazabilidad	Capacidad de reconstruir cómo se produjo una respuesta.
Traza	Registro estructurado de los pasos que produjeron una respuesta.
Span	Paso concreto dentro de una traza.
Laboratorio	Espacio reproducible para probar una idea con casos, métricas y trazas.
Notebook	Documento ejecutable que mezcla explicación, código, salidas y metadatos.
Eval	Prueba sistemática para medir si un sistema cumple un comportamiento esperado.
Dataset de evaluación	Conjunto de casos con entradas, resultados esperados y criterios de aceptación.
Evaluación offline	Pruebas con casos preparados antes de uso real.
Evaluación online	Medición con uso real, tráfico y cambios.
Hit@k	Indica si el resultado esperado aparece entre los k primeros.
MRR	Media del inverso de la posición del primer resultado correcto.
Gate	Umbral o regla que decide si una variante puede avanzar.
Manifest	Registro de versión, datos, entorno y configuración del experimento.
Arquitectura mínima suficiente	Diseño más sencillo que resuelve el problema medido.

TÉRMINO	DEFINICIÓN
Decisión sin IA	Elección de resolver con reglas, SQL, interfaz o proceso cuando un modelo no aporta valor suficiente.

Antes de pasar página

Responde sin consultar el facsímil. Si fallas una pregunta, la pista te dice dónde volver.

- [ ] **1.** ¿Puedo elegir entre prompt, RAG, tool y ajuste explicando qué cambia cada uno? (Si no, vuelve al capítulo 01.)
- [ ] **2.** ¿Sé diseñar una llamada API con mensajes, parámetros, streaming, tool y salida estructurada? (Si no, vuelve al capítulo 02.)
- [ ] **3.** ¿Puedo estimar coste de tokens, contexto y caché antes de construir? (Si no, vuelve al capítulo 03.)
- [ ] **4.** ¿Sé leer una model card y separar marketing, métrica y restricción operativa? (Si no, vuelve al capítulo 04.)
- [ ] **5.** ¿Puedo explicar cuándo usar local, cloud, GPU alquilada o cuantización? (Si no, vuelve al capítulo 05.)
- [ ] **6.** ¿Sé qué es una dimensión de embedding y qué mide la similitud coseno? (Si no, vuelve al capítulo 06.)
- [ ] **7.** ¿Puedo diseñar búsqueda híbrida con filtros y metadatos? (Si no, vuelve al capítulo 08.)
- [ ] **8.** ¿Sé explicar por qué RAG no es memoria ni entrenamiento? (Si no, vuelve al capítulo 09.)
- [ ] **9.** ¿Puedo proponer métricas de evaluación RAG y casos donde el sistema debe abstenerse? (Si no, vuelve al capítulo 10.)
- [ ] **10.** ¿Sé cuándo Agentic RAG o GraphRAG compensan su complejidad? (Si no, vuelve al capítulo 11.)
- [ ] **11.** ¿Puedo diseñar una herramienta Text-to-SQL con permisos, validación y traza? (Si no, vuelve al capítulo 12.)
- [ ] **12.** ¿Puedo revisar una arquitectura y decir qué falta para llevarla a prototipo controlado? (Si no, vuelve a «La matriz de decisión final».)
- [ ] **13.** ¿Puedo defender cuándo no usar IA? (Si no, vuelve a «Cuándo no usar IA».)
- [ ] **14.** ¿Puedo explicar la diferencia entre demo, notebook y laboratorio? (Si no, vuelve a «Qué no es un laboratorio».)

- [ ] **15.** ¿Sé qué debe contener un dataset de evaluación mínimo y por qué importa el manifest? (Si no, vuelve a «Qué sí debería dejar».)

- [ ] **16.** ¿Puedo calcular Hit@1 y MRR con un ejemplo pequeño y leer un span? (Si no, vuelve a «Cómo medir de verdad: métricas, jueces y comparación».)

- [ ] **17.** ¿Puedo definir un gate con umbrales antes de ejecutar el experimento? (Si no, vuelve a «Anatomía de un laboratorio mínimo».)

- [ ] **18.** ¿Sé qué artefactos deja un laboratorio mínimo y por qué una traza permite auditar una decisión? (Si no, vuelve a «Anatomía de un laboratorio mínimo».)

En resumen

IDEA FUERZA	DETALLE
La caja de herramientas empieza con diagnóstico.	Antes de elegir técnica, hay que saber qué parte del sistema falla.
Un modelo útil necesita contratos alrededor.	APIs, schemas, herramientas, validadores y trazas hacen gobernable la respuesta.
La evidencia se recupera, se mide y se cita.	RAG, búsqueda híbrida, GraphRAG y Text-to-SQL son formas distintas de traer base verificable.
El coste también es arquitectura.	Tokens, contexto, caché, latencia, GPU, local y cloud cambian el diseño.
La evaluación separa demo de sistema.	Sin casos, métricas, abstención y regresiones, no sabes si mejoraste.
La complejidad tiene que ganarse el sitio.	Agentic RAG, GraphRAG o fine-tuning solo compensan cuando resuelven un fallo medido.
No usar IA también es una decisión técnica.	Si una regla, SQL o una interfaz clara resuelve mejor, esa es la herramienta correcta.
El siguiente paso es coordinar herramientas.	El facsímil 05 parte de esta base para hablar de agentes y orquestación.

Recursos para seguir: leer, construir y experimentar

Esta es la caja de herramientas: APIs, tokens, modelos locales, embeddings, bases vectoriales y RAG. Es el facsículo más práctico, así que el mejor recurso es ponerse.

Para experimentar sin apenas código. Lo has hecho en las cajas «Pruébalo en 5 minutos»: el tokenizador de OpenAI (`platform.openai.com/tokenizer`) para ver tokens y coste; los playgrounds (`platform.openai.com` , `aistudio.google.com` , `console.anthropic.com`) para salidas estructuradas y SQL; Hugging Face (`huggingface.co`) para leer fichas de modelos; LM Studio (`lmstudio.ai`) y Ollama (`ollama.com`) para correr modelos locales; y el Embedding Projector (`projector.tensorflow.org`) para la búsqueda semántica.

Para construir. Las piezas reales por tema: los SDK oficiales de OpenAI, Anthropic y Google para llamar a las APIs; LangChain o LlamaIndex para orquestar RAG; bases vectoriales como Qdrant, Chroma o pgvector para la búsqueda; y Ollama o llama.cpp para lo local. Empieza con un runner pequeño y propio antes de montar una plataforma.

Para leer. Aquí lo importante no son los papers, sino la documentación: las guías oficiales de cada proveedor, que cada capítulo enlaza en su «Para saber más». Leerlas con un caso propio en mente es lo que convierte entender una herramienta en usarla en serio.

Para saber más

Arize Phoenix. (2026). *Evaluate RAG*. Documentación oficial

Arize Phoenix. (2026). *Evaluation concepts*. Documentación oficial

Boardman, A. E., Greenberg, D. H., Vining, A. R. y Weimer, D. L. (2018). *Cost-Benefit Analysis: Concepts and Practice* (5.ª ed.). Cambridge University Press. DOI

Brown, L. D., Cai, T. T. y DasGupta, A. (2001). *Interval Estimation for a Binomial Proportion*. *Statistical Science*, 16(2), 101-133. DOI

Chen, M. y otros (2021). *Evaluating Large Language Models Trained on Code*. arXiv

Cormack, G. V., Clarke, C. L. A. y Büttcher, 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

Efron, B. y Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman & Hall/CRC. DOI

Ellram, L. M. (1995). *Total cost of ownership: an analysis approach for purchasing*. *International Journal of Physical Distribution & Logistics Management*, 25(8), 4-23. DOI

Es, S. y otros (2023). *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. arXiv

Hu, E. J. y otros (2022). *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv

Järvelin, K. y Kekäläinen, J. (2002). *Cumulated gain-based evaluation of IR techniques*. *ACM Transactions on Information Systems*, 20(4), 422-446. DOI

Johnson, J., Douze, M. y Jégou, H. (2019). *Billion-scale similarity search with GPUs*. *IEEE Transactions on Big Data*, 7(3), 535-547. DOI

Jupyter. (2026). *The Notebook file format*. Documentación oficial

LangChain. (2026). *Evaluate a RAG application*. 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

Little, J. D. C. (1961). *A Proof for the Queuing Formula: $L = \lambda W$*. *Operations Research*, 9(3), 383-387. DOI

Liu, Y. y otros (2023). *G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment*. *Proceedings of EMNLP 2023*. arXiv

Malkov, Y. A. y Yashunin, D. A. (2020). *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs*. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4), 824-836. DOI

Manning, C. D., Raghavan, P. y Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. Disponible en línea

McNemar, Q. (1947). *Note on the sampling error of the difference between correlated proportions or percentages*. *Psychometrika*, 12(2), 153-157. DOI

Mitchell, M. y otros (2019). *Model Cards for Model Reporting*. *Proceedings of FAT* 2019, 220-229. DOI

OpenAI. (2026). *Function calling*. Documentación oficial

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

OpenAI. (2026). *Prompt caching*. Documentación oficial

OpenAI. (2026). *Structured model outputs*. Documentación oficial

OpenTelemetry. (2026). *Tracing API*. Documentación oficial

Ragas. (2026). *List of available metrics*. Documentación oficial

Robertson, S. y Zaragoza, H. (2009). *The probabilistic relevance framework: BM25 and beyond*. *Foundations and Trends in Information Retrieval*, 3(4), 333-389. DOI

Sainz, O. y otros (2023). *NLP Evaluation in Trouble: On the Need to Measure LLM Data Contamination for each Benchmark*. *Findings of EMNLP 2023*, 10776-10787. arXiv

Salton, G., Wong, A. y Yang, C. S. (1975). *A vector space model for automatic indexing*. *Communications of the ACM*, 18(11), 613-620. DOI

Voorhees, E. M. (1999). *The TREC-8 Question Answering Track Report*. *Proceedings of the 8th Text REtrieval Conference (TREC-8)*, 77-82. NIST

Yu, T. y otros (2018). *Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task*. *Proceedings of EMNLP*, 3911-3921. ACL Anthology

Zheng, L. y otros (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. *Advances in Neural Information Processing Systems 36*. arXiv

---

Notas

1. OpenAI. (2026). *Function calling*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

2. OpenAI. (2026). *Structured model outputs*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

3. Hu, E. J. y otros (2022). *LoRA: Low-Rank Adaptation of Large Language Models*. *International Conference on Learning Representations*. arXiv. Dettmers, T. y otros (2023). *QLoRA: Efficient Finetuning of Quantized LLMs*. *Advances in Neural Information Processing Systems 36*. arXiv. ↩

4. OpenAI. (2026). *Prompt caching*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

5. Ellram, L. M. (1995). *Total cost of ownership: an analysis approach for purchasing*. *International Journal of Physical Distribution & Logistics Management*, 25(8), 4-23. DOI. ↩

6. Mitchell, M. y otros (2019). *Model Cards for Model Reporting*. *Proceedings of FAT* 2019, 220-229. DOI. ↩

7. Salton, G., Wong, A. y Yang, C. S. (1975). *A vector space model for automatic indexing*. *Communications of the ACM*, 18(11), 613-620. DOI. ↩

8. Johnson, J., Douze, M. y Jégou, H. (2019). *Billion-scale similarity search with GPUs*. *IEEE Transactions on Big Data*, 7(3), 535-547. DOI. ↩

9. Malkov, Y. A. y Yashunin, D. A. (2020). *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs*. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4), 824-836. DOI. ↩

10. Robertson, S. y Zaragoza, H. (2009). *The probabilistic relevance framework: BM25 and beyond*. *Foundations and Trends in Information Retrieval*, 3(4), 333-389. DOI. ↩

.1. Cormack, G. V., Clarke, C. L. A. y Büttcher, S. (2009). *Reciprocal rank fusion outperforms Condorcet and individual rank learning methods*. *Proceedings of SIGIR*, 758-759. DOI. ↩

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

.3. Es, S. y otros (2023). *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. arXiv. ↩

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

.5. Yu, T. y otros (2018). *Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task*. *Proceedings of EMNLP*, 3911-3921. ACL Anthology. ↩

.6. Boardman, A. E., Greenberg, D. H., Vining, A. R. y Weimer, D. L. (2018). *Cost-Benefit Analysis: Concepts and Practice* (5.ª ed.). Cambridge University Press. Enlace. ↩

.7. Jupyter. (2026). *The Notebook file format*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

.8. OpenTelemetry. (2026). *Tracing API*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

.9. OpenAI. (2026). *Graders*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

!0. Ragas. (2026). *List of available metrics*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

!1. LangChain. (2026). *Evaluate a RAG application*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

!2. Arize Phoenix. (2026). *Evaluate RAG*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

!3. Arize Phoenix. (2026). *Evaluation concepts*. Documentación oficial. Consultado el 26 de mayo de 2026. ↩

!4. Järvelin, K. y Kekäläinen, J. (2002). *Cumulated gain-based evaluation of IR techniques*. *ACM Transactions on Information Systems*, 20(4), 422-446. DOI. ↩

!5. Es, S. y otros (2023). *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. arXiv. ↩

!6. Zheng, L. y otros (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. *Advances in Neural Information Processing Systems 36*. arXiv. ↩

27. Liu, Y. y otros (2023). *G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment*. *Proceedings of EMNLP 2023*. arXiv. ↩

28. McNemar, Q. (1947). *Note on the sampling error of the difference between correlated proportions or percentages*. *Psychometrika*, 12(2), 153-157. DOI. ↩

29. Efron, B. y Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman & Hall/CRC. DOI. ↩

30. Sainz, O. y otros (2023). *NLP Evaluation in Trouble: On the Need to Measure LLM Data Contamination for each Benchmark*. *Findings of the Association for Computational Linguistics: EMNLP 2023*, 10776-10787. arXiv. ↩

31. Little, J. D. C. (1961). *A Proof for the Queuing Formula: $L = \lambda W$*. *Operations Research*, 9(3), 383-387. DOI. ↩

32. Manning, C. D., Raghavan, P. y Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, cap. 8. Disponible en línea. ↩

33. Voorhees, E. M. (1999). *The TREC-8 Question Answering Track Report*. *Proceedings of the 8th Text REtrieval Conference (TREC-8)*, 77-82. NIST. ↩

34. Manning, C. D., Raghavan, P. y Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, cap. 8 (accuracy, precision y recall). Disponible en línea. ↩

35. Chen, M. y otros (2021). *Evaluating Large Language Models Trained on Code*. arXiv. ↩

36. Brown, L. D., Cai, T. T. y DasGupta, A. (2001). *Interval Estimation for a Binomial Proportion*. *Statistical Science*, 16(2), 101-133. DOI. ↩