Facsímil 07

Evaluar, calibrar e interpretar

Coleccionable completo · 6 capítulos

686f6c61 · 2026

Índice

Capítulo 01: Qué es una eval y qué decisión permite tomar

Entrando en el tema

Este facsímil empieza con una idea que parece menos brillante que hablar de interpretabilidad, calibración o modelos evaluadores, pero es la que sostiene todo lo demás: **una evaluación buena no existe para decorar una presentación; existe para tomar una decisión**.

Al terminar este capítulo deberías poder hacer esto:

RESULTADO DE APRENDIZAJE	EVIDENCIA DE QUE LO SABES HACER
Distinguir una demo de una eval.	Puedes explicar por qué una respuesta bonita no prueba que el sistema funcione.
Diseñar una eval mínima.	Defines hipótesis, casos, salida esperada, rúbrica, baseline, candidate, métricas y umbrales.
Convertir métricas en decisión.	Dices si una variante se acepta, se rechaza, se revisa o se limita.
Separar promedio y fallo crítico.	No dejas que una media alta esconda un caso que no debería pasar.
Crear una scorecard ejecutable.	Produces un JSON con métricas, coste, regresiones, incertidumbre y decisión.
Conectar evaluación con operación.	Ves cómo una eval alimenta CI, release gates, runbooks y mejora continua.

La frase que nos va a acompañar durante todo el facsímil es esta:

> No preguntes primero “¿qué métrica saco?”. Pregunta “¿qué decisión quiero poder defender?”.

La escena: cambiar algo sin saber si empeora

Imagina que tienes un asistente interno para alumnado. Responde dudas sobre matrícula, becas, horarios y trámites. El equipo quiere cambiar el prompt, usar otro modelo y añadir una capa de recuperación documental. En una demo, la versión nueva suena más fluida. Parece mejor.

Pero una demo no responde preguntas incómodas:

PREGUNTA	POR QUÉ IMPORTA
¿Mejora en los casos frecuentes o solo en el ejemplo que hemos mirado?	Una mejora local puede esconder regresiones.
¿Sigue absteniéndose cuando no hay evidencia?	Una respuesta inventada puede ser peor que no responder.
¿Cuánto cuesta cada caso aceptado?	El modelo barato por token puede ser caro por tarea real.
¿Qué ocurre con los casos frontera?	Ahí aparecen los errores que una demo limpia no enseña.
¿Qué evidencia dejamos para revisar después?	Sin trazas ni scorecard, no hay aprendizaje reproducible.

Una eval nace justo ahí: cuando dejamos de mirar una anécdota y empezamos a construir evidencia repetible.

Qué no es una eval

Una eval no es “he probado tres preguntas y me gusta más”. Eso puede servir para exploración inicial, pero no para decidir una release.

Tampoco es un benchmark público usado como respuesta automática. Un benchmark puede orientar, comparar familias de modelos y detectar capacidades generales. Pero tu sistema vive en tus datos, tus contratos, tus usuarios, tus costes y tus límites operativos. HELM propuso una evaluación amplia de modelos de lenguaje precisamente para mirar varios escenarios, métricas y dimensiones de forma sistemática, no para reducir todo a un número único.[1]

Y una eval tampoco es solo un evaluador LLM. Un evaluador puede ser útil cuando hay que valorar calidad semántica, groundedness o completitud. Pero si puedes validar JSON, una llamada de herramienta, un diff, una cita o un cálculo con código determinista, suele ser

mejor empezar por ahí. OpenAI describe graders como mecanismos que comparan respuestas de referencia y salidas del modelo para devolver puntuaciones, con tipos como comprobaciones de texto, similitud, modelos evaluadores y ejecución de código.[2]

Qué sí es una eval

En este facsímil llamaremos **eval** a un diseño reproducible que compara una versión candidata contra casos, criterios, métricas y umbrales para tomar una decisión.

No hace falta inventar una fórmula para entender su anatomía. Lo útil es escribir el contrato con piezas observables:

PIEZA	PREGUNTA QUE RESPONDE	EJEMPLO
Dataset de casos	¿Con qué entradas medimos el comportamiento?	80 preguntas reales y 20 casos sin evidencia suficiente.
Tarea evaluada	¿Qué debe hacer el sistema en cada caso?	Responder con cita o abstenerse.
Graders o evaluadores	¿Quién convierte una salida en señal medible?	Validador JSON, comprobador de cita, evaluador de rúbrica, revisión humana.
Métricas agregadas	¿Qué señales resumimos sin perder trazabilidad?	Exactitud, groundedness, tasa de abstención correcta, coste por aceptada.
Umbrales de decisión	¿Qué condiciones separan aceptar, revisar o bloquear?	`quality >= 0.85`, `critical_failures == 0`.
Acción	¿Qué haremos con el resultado?	Aceptar, rechazar, limitar, revisar o volver a baseline.

Lo importante es que la acción forma parte de la evaluación. Si una eval no termina en una acción posible, es un informe interesante, pero todavía no es una puerta de ingeniería.

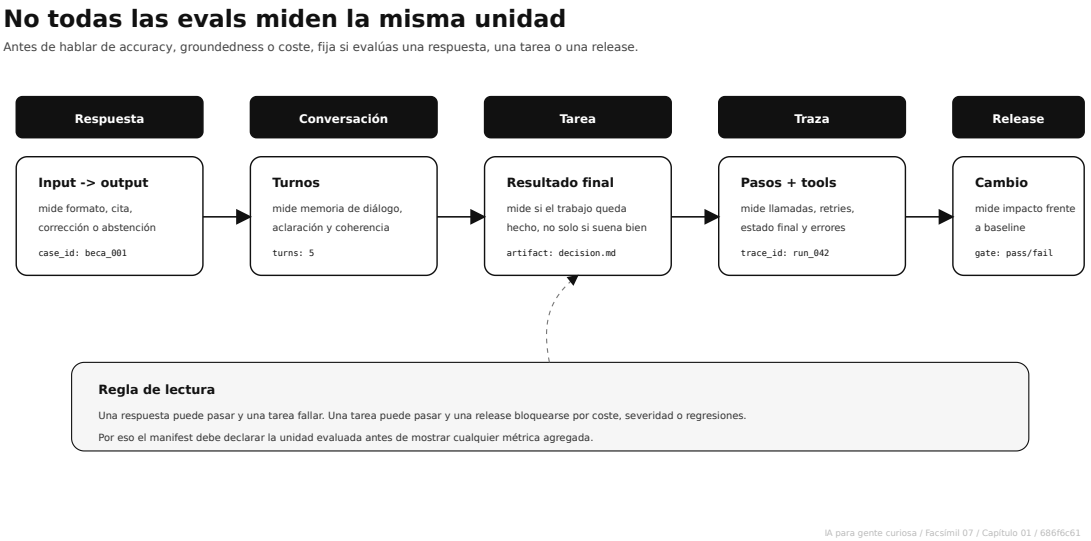
Unidad de evaluación: qué estás midiendo exactamente

Antes de elegir métrica hay una pregunta más básica: **cuál es la unidad que vas a medir**. Parece una sutileza, pero en proyectos reales explica muchas discusiones absurdas. Un equipo dice “nuestro sistema acierta el 90%”, otro responde “pero falla tareas completas”, y los dos pueden tener razón porque no están midiendo la misma cosa.

No es igual evaluar una respuesta aislada que una conversación de cinco turnos, una tarea con herramientas, una traza de agente o una release. La respuesta aislada puede ser correcta y la tarea completa fallar porque se llamó mal a una API. Una conversación puede sonar bien turno a turno y acabar perdiendo el objetivo. Un agente puede producir una salida final aceptable, pero con una trayectoria cara, insegura o imposible de auditar. Por eso la unidad de evaluación debe escribirse antes del dataset.

UNIDAD	QUÉ MIDE	QUÉ PUEDE ESCONDER	EJEMPLO DE DECISIÓN
Respuesta	Una salida concreta ante un input.	Pérdida de contexto entre turnos o uso incorrecto de herramientas.	Cambiar prompt de respuesta.
Conversación	Varios turnos con memoria de diálogo.	Que una respuesta individual parezca bien, pero el flujo no resuelva.	Ajustar política de aclaración.
Tarea	Resultado final con pasos intermedios.	Trayectorias caras o frágiles que acaban acertando por casualidad.	Limitar herramientas o exigir confirmación.
Traza de agente	Secuencia de razonamiento externo: tools, estados, errores y retry.	Que el resultado final oculte una acción incorrecta.	Bloquear una arquitectura de agente.
Release	Cambio completo de sistema frente a baseline.	Mejoras locales con regresiones por slice.	Publicar, revisar, hacer canary o bloquear.

La lectura de ingeniería es directa: una métrica solo tiene sentido si sabes sobre qué objeto se calcula. Si mezclas unidades, el score deja de ser evidencia y se convierte en ruido con decimales.



> Unidad de evaluación: medir una respuesta, una tarea o una release no responde la misma pregunta.

Fecha de corte del estado del arte

Fecha de corte: 28 de mayo de 2026.
Fuentes consultadas: documentación de OpenAI Evals y graders; documentación de LangSmith Evaluation; guías de Braintrust sobre evaluación sistemática; documentación de Promptfoo sobre assertions y métricas; Hugging Face Evaluate; EleutherAI Language Model Evaluation Harness; HELM; model cards; datasheets for datasets; TFX; acuerdo entre revisores; bootstrap; comparación pareada; y literatura de ingeniería de software para ML.

OpenAI presenta Evals como una forma de crear, gestionar y ejecutar evaluaciones sobre modelos con data sources y graders.[3] Braintrust describe una evaluación como la combinación de datos, tarea y funciones de scoring, con comparación de experimentos y seguimiento de regresiones.[4] LangSmith sitúa la evaluación en datasets, evaluadores y experimentos trazables dentro del ciclo de desarrollo de aplicaciones LLM.[5]

Promptfoo permite expresar expectativas como assertions sobre outputs, incluyendo igualdad, JSON, similitud, funciones de Python o JavaScript, pesos y umbrales.[6] Hugging Face Evaluate insiste en elegir métricas según la tarea, porque no hay una métrica universal que sirva para todo.[7] EleutherAI `lm-evaluation-harness` ofrece un marco unificado para evaluar modelos de lenguaje en tareas académicas y backends distintos.[8]

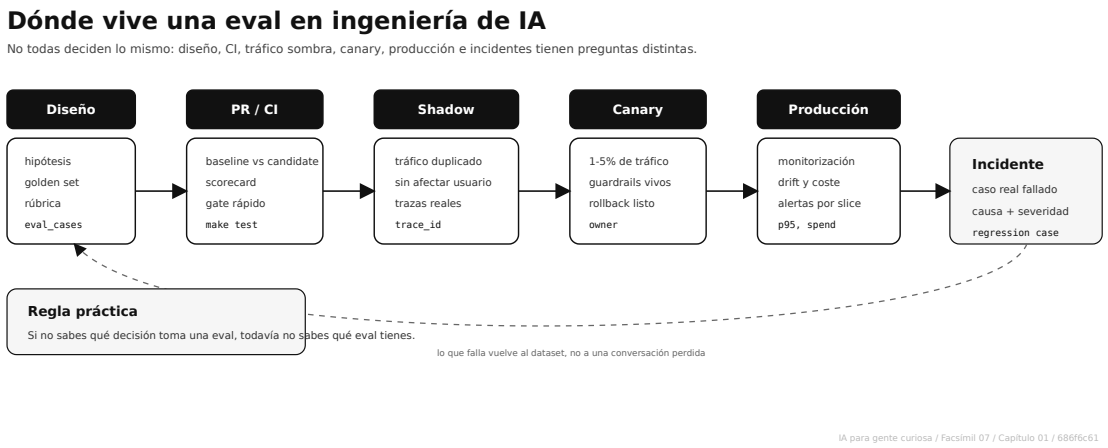
La conclusión práctica es sencilla: el mercado tiene herramientas distintas, pero la anatomía se repite. Necesitas casos, tarea, evaluadores, métricas, trazas, comparación y decisión.

Tipos de eval según el momento

Una palabra puede esconder varios trabajos distintos. No es lo mismo una eval que ejecutas en local antes de abrir un Pull Request que una eval que corre sobre tráfico real en modo sombra. Tampoco es lo mismo medir exactitud en un conjunto estable que buscar abusos, fugas de privacidad o fallos raros. Para un ingeniero, distinguir el momento de la eval evita pedirle a una herramienta lo que pertenece a otra.

TIPO DE EVAL	CUÁNDO APARECE	QUÉ DECIDE	RIESGO SI LA USAS MAL
Eval exploratoria	Al principio, cuando todavía estás entendiendo la tarea.	Si merece la pena construir un dataset serio.	Confundir intuición inicial con evidencia de release.
Golden set u offline regression eval	Antes de mezclar cambios en prompt, modelo, RAG o herramientas.	Si una candidata empeora casos conocidos.	Sobreajustar al conjunto si se toca demasiado.
Eval de Pull Request o CI	En cada cambio relevante.	Si el cambio puede pasar revisión técnica.	Convertirla en un test lento y ruidoso que el equipo empieza a ignorar.
Shadow eval	Con tráfico real duplicado, sin afectar al usuario.	Si la nueva versión se comporta bien en casos vivos.	Medir sin guardar contexto suficiente para explicar fallos.
Canary o A/B con guardrails	Con un porcentaje pequeño de usuarios o tareas.	Si la candidata aguanta producción limitada.	Mirar solo conversión o satisfacción y olvidar fallos críticos.
Monitorización continua	Después de publicar.	Si hay drift, regresiones o cambios de coste.	Detectar tarde porque no hay alertas por slice o severidad.
Red-team eval	Antes o después de publicar, según riesgo.	Si existen caminos de abuso, privacidad o seguridad.	Tratarla como checklist de cumplimiento en vez de como búsqueda activa de fallos.

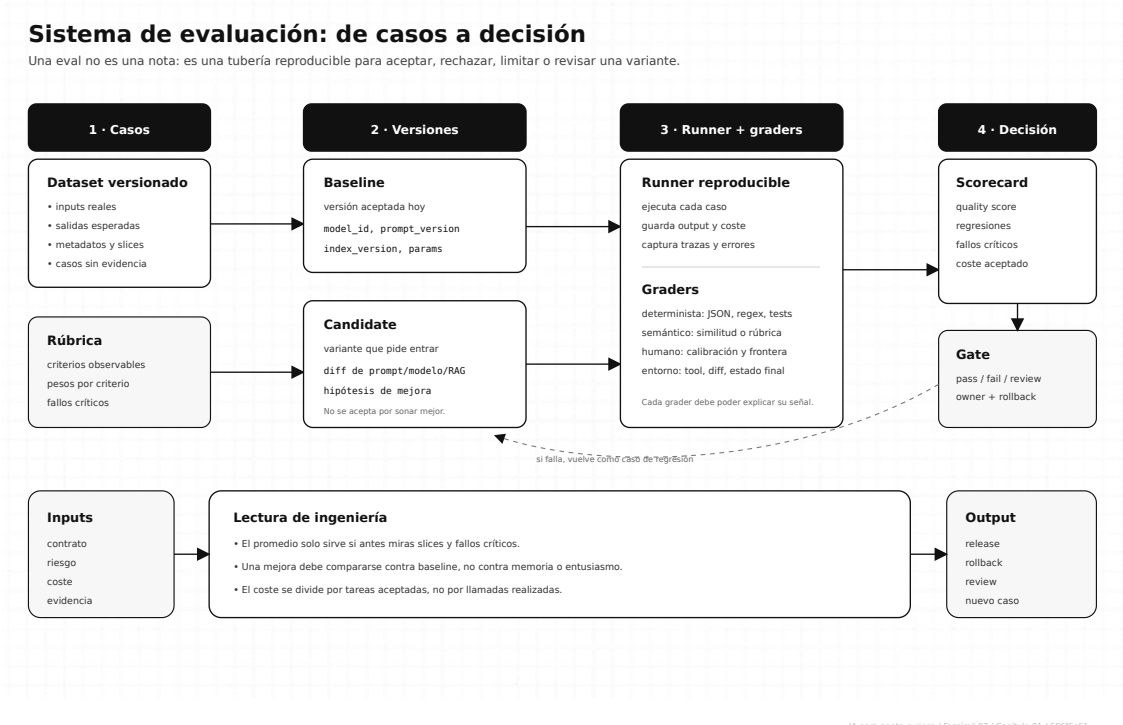
Este capítulo se centra sobre todo en la eval offline con gate de release, porque es la primera que necesitas para trabajar con orden. Pero no la veas como una isla: el mismo dataset puede crecer con incidentes de producción, el mismo manifest puede alimentar auditoría, y el mismo gate puede acabar en CI o en un runbook de operación.



Ciclo de vida de las evals: del diseño al incidente que vuelve como caso de regresión.

La anatomía técnica de una evaluación

Una evaluación profesional separa piezas. Si las mezclamos, no sabemos qué arreglar.



Anatomía de una eval como sistema: casos, versiones, runner, graders, métricas, gate y bucle de mejora.

La imagen tiene una intención muy concreta. Una eval no vive solo en la columna de métricas. Vive en todo el circuito:

1. Casos suficientemente representativos.
2. Dos versiones comparables.
3. Ejecución reproducible.
4. Evaluadores separados por tipo de señal.
5. Scorecard trazable.
6. Gate que decide.
7. Regresiones que vuelven al dataset.

Esa última flecha es vital. Un fallo que se descubre en producción y no entra en la eval queda condenado a repetirse.

Métricas: el número no decide solo

Una métrica resume una parte del comportamiento. Una decisión combina varias señales.

Podemos construir una puntuación ponderada como una rúbrica: cada criterio recibe un peso, cada respuesta recibe una puntuación y el resultado final resume cuánto de importante se ha cumplido. No la voy a presentar como fórmula porque en este facsímil la norma es clara: si una expresión matemática no es una fórmula académica reconocible y referenciable, mejor escribirla como procedimiento, tabla o ejemplo.

La idea práctica sí importa. No todas las señales pesan igual: un formato JSON inválido puede romper integración, pero una respuesta inventada en un caso sin evidencia puede ser directamente bloqueante. El peligro es creer que, por estar todo en un número, ya has decidido bien. Una puntuación ponderada resume; no sustituye al análisis de fallos críticos, slices y regresiones.

Ejemplo:

CRITERIO	PESO	RESULTADO	APORTE PONDERADO
Formato válido	1	1,00	1,00
Respuesta correcta	3	0,90	2,70
Cita verificable	3	0,70	2,10
Abstención cuando toca	3	0,50	1,50
Total	**10**		**7,30**

La puntuación agregada sería 0,73 porque el aporte ponderado total es 7,30 sobre 10. Parece decente, pero hay una alarma: abstención correcta vale 0,50. Si el sistema responde cuando no tiene evidencia, quizá no puede publicarse aunque la media no sea catastrófica.

Por eso un gate real suele tener dos capas. No lo escribimos como fórmula académica, sino como contrato operativo: una regla que podría vivir en un JSON, en un workflow de CI o en una revisión de release.

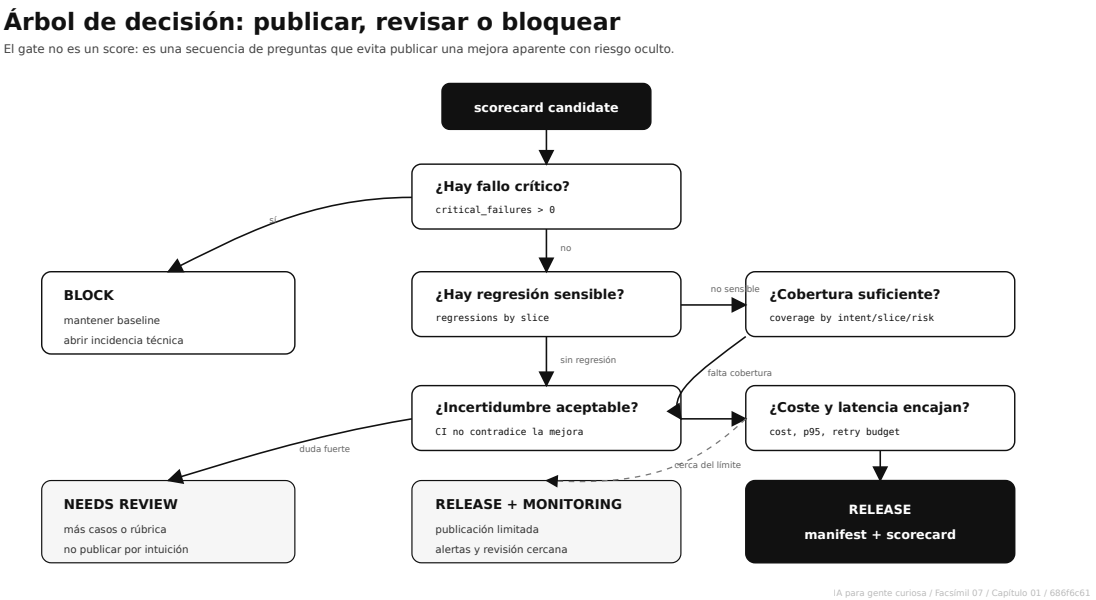
```
aceptar si:
  quality_score >= quality_threshold
  critical_failures == 0
  cost_per_accepted <= cost_budget
```

CAMPO DEL CONTRATO	SIGNIFICADO	EJEMPLO
`quality_score`	Puntuación agregada.	0,87.
`quality_threshold`	Umbral mínimo de calidad.	0,85.
`critical_failures`	Número de fallos críticos.	1 respuesta sin evidencia.
`cost_per_accepted`	Coste por tarea aceptada.	0,031 €.
`cost_budget`	Presupuesto máximo por aceptada.	0,040 €.

La media puede pasar y el gate puede fallar. Eso no es una contradicción: es ingeniería.

Una política de decisión completa suele separar cuatro salidas, no solo “pasa” o “falla”:

SALIDA	CUÁNDO USARLA	QUÉ DEBERÍA OCURRIR DESPUÉS
`release`	Pasa calidad mínima, no hay fallos críticos, el coste entra en presupuesto y la incertidumbre no contradice la mejora.	Publicar con manifest, scorecard y seguimiento.
`release_with_monitoring`	Pasa lo esencial, pero hay muestra pequeña, coste cercano al límite o alguna duda no bloqueante.	Publicar limitado, activar alertas y revisar pronto.
`needs_review`	El intervalo cruza cero, hay desacuerdo de etiquetado o falta cobertura de slices relevantes.	Pedir más casos, revisar rúbrica o etiquetar muestra adicional.
`block`	Hay fallo crítico, regresión en slice sensible, coste fuera de presupuesto o ruptura de contrato.	Mantener baseline, abrir tarea técnica y añadir caso de regresión.



Árbol de decisión de release: la media no decide sola; se mira fallo crítico, regresión, cobertura, incertidumbre, coste y latencia.

Tipos de evaluadores que conviene combinar

Ningún grader lo ve todo. Una buena eval combina evaluadores según la naturaleza de la tarea.

EVALUADOR	QUÉ MIDE BIEN	QUÉ NO VE BIEN	EJEMPLO
Determinista	Formato, exact match, regex, JSON, rangos, campos obligatorios.	Calidad semántica rica.	“El JSON tiene `categoria`, `prioridad` y `siguiente_paso`”.
Código o entorno	Tests, diffs, estado final, herramientas llamadas, cálculos.	Intención, estilo o utilidad percibida.	“La función pasa unit tests y no modifica archivos fuera de ruta”.
Similaridad	Cercanía semántica entre salida y referencia.	Errores pequeños pero graves.	“La respuesta se parece al resumen esperado”.
Rúbrica humana	Juicio experto, contexto, ambigüedad.	Escalabilidad y consistencia sin guía.	“Un docente revisa 30 casos frontera”.
LLM como evaluador	Groundedness, completitud, estilo, comparación A/B.	Variabilidad, sesgo de longitud, coste y cambios de versión.	“Puntúa si la respuesta está apoyada por la cita”.
Métrica operativa	Latencia, coste, reintentos, trazas, tasa de aceptación.	Calidad de contenido por sí sola.	“p95 menor que 4 s y coste por aceptada menor que 0,04 €”.

La regla práctica es simple: **usa lo determinista para lo verificable, usa rúbrica para lo semántico y reserva revisión humana para calibrar o decidir casos delicados**.

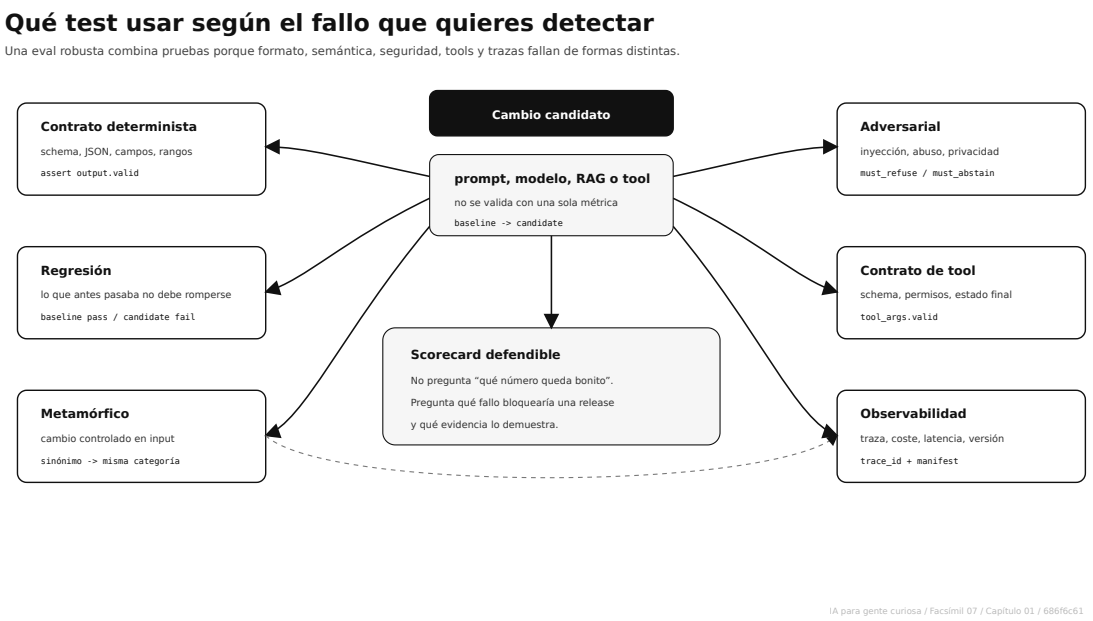
Familias de tests de ingeniería para evals

En un proyecto serio no deberíamos pedirle a una sola métrica que lo vea todo. Lo mismo que en software no confundes un test unitario con una prueba de carga, en IA no conviene confundir una comprobación de JSON con una evaluación semántica o con una prueba de seguridad. Cada familia de test responde una pregunta diferente.

La palabra *test* aquí no significa solo “assert en CI”. Significa una forma controlada de producir evidencia. A veces será una función determinista. A veces será una revisión humana. A veces será una batería de casos adversariales. A veces será una comparación entre dos versiones. Lo importante es que el test esté conectado con un fallo posible y con una acción si falla.

FAMILIA	PREGUNTA QUE RESPONDE	SEÑAL TÍPICA	CUÁNDO USARLA
Test determinista	¿La salida cumple un contrato verificable?	JSON válido, campos obligatorios, regex, rango numérico, código de estado.	Integraciones, APIs, extracción estructurada, herramientas.
Test de regresión	¿Hemos roto algo que antes funcionaba?	Caso que baseline pasaba y candidate falla.	Cambios de prompt, modelo, índice, parser o tool.
Test metamórfico	¿Se mantiene una relación esperada al cambiar la entrada?	Si añado ruido irrelevante, la decisión no debería cambiar.	Sistemas donde no hay respuesta única, pero sí invariantes. Chen y coautores formalizaron esta idea para probar programas cuando el oráculo exacto es difícil.[9]
Test adversarial	¿Qué ocurre cuando alguien fuerza el límite?	Prompt injection, datos sensibles, instrucciones conflictivas, entradas largas.	Asistentes públicos, RAG, agentes con tools, productos regulados.
Test de contrato de herramienta	¿La herramienta fue llamada con argumentos correctos y permiso adecuado?	Tool name, schema, argumentos, estado final, error controlado.	Agentes, SDKs, acciones sobre sistemas externos.
Test de observabilidad	¿Podremos depurar el fallo después?	`trace_id`, versión, latencia, coste, contexto, grader y error taxonómico.	Cualquier sistema que vaya a producción.

Un test metamórfico merece una pausa. En muchos sistemas de IA no existe una única respuesta exacta. Pero sí podemos definir relaciones esperadas. Si una pregunta se reescribe con sinónimos, la categoría debería mantenerse. Si se añade un párrafo irrelevante, la respuesta no debería inventar otra fuente. Si se permutan dos documentos equivalentes, el ranking no debería hundir el documento correcto sin motivo. No es magia estadística: es ingeniería para construir oráculos parciales cuando el oráculo perfecto no existe.



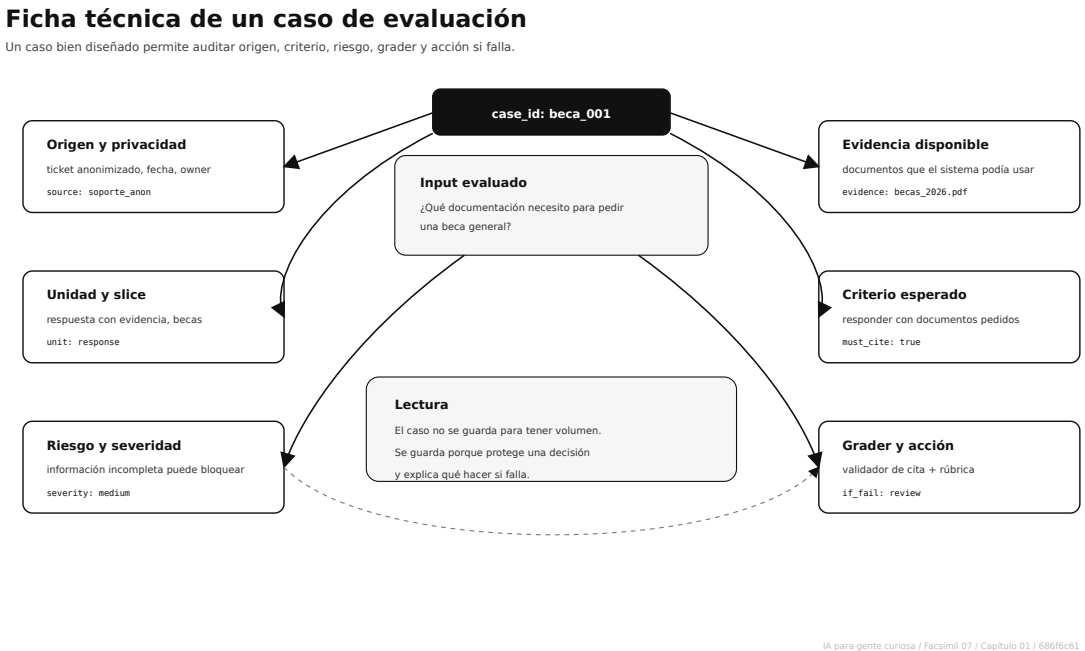
Familias de tests: cada prueba existe para detectar un tipo de fallo y sostener una decisión concreta.

Anatomía de un caso de evaluación

Después de decidir la unidad, toca diseñar los casos. Un caso de eval no es una frase metida en un fichero. Es una unidad de evidencia. Debe permitir que alguien entienda de dónde sale, qué riesgo cubre, qué respuesta sería aceptable, qué grader lo evalúa y qué hacer si falla.

Esta disciplina evita dos problemas. El primero es el dataset opaco: una lista de prompts sin fuente ni intención. El segundo es el dataset decorativo: casos que existen porque "parecían interesantes", pero no están conectados con un riesgo, una métrica o una decisión. En ingeniería, un caso debería poder defenderse como se defiende un test de regresión: existe porque protege algo.

CAMPO	POR QUÉ ESTÁ	EJEMPLO
`case_id`	Identificador estable para comparar corridas.	`beca_001`.
Fuente	Permite auditar si viene de producción, experto, incidente o síntesis controlada.	Ticket de soporte anonimizado.
Unidad	Aclara si evalúa respuesta, conversación, tarea, traza o release.	Respuesta con evidencia.
Slice	Evita que la media tape subgrupos.	Becas, matrícula, soporte, sin evidencia.
Severidad	Prioriza fallos que bloquean aunque sean raros.	Alta si inventa una norma.
Evidencia disponible	Marca qué documentos o contexto podía usar el sistema.	Reglamento de matrícula 2026.
Criterio esperado	Define qué debe ocurrir.	Citar fuente o abstenerse.
Oracle o grader	Explica quién decide si pasó.	Validador de cita y rúbrica humana.
Política si falla	Conecta el caso con una acción.	Bloquear release y añadir regresión.



Anatomía de un caso de eval: cada campo existe para sostener trazabilidad, criterio y acción.

Dataset: dónde se decide la calidad de la eval

El dataset de evaluación no es un CSV cualquiera. Es el instrumento de medida.

Gebru y coautoras propusieron *Datasheets for Datasets* para documentar motivación, composición, recogida, procesamiento, usos recomendados y mantenimiento de datasets.[10] Esa idea encaja directamente con evals: si no sabes de dónde salen tus casos, qué cubren y qué dejan fuera, la métrica puede sonar seria y medir mal.

Una eval mínima debería incluir:

PARTE DEL DATASET	QUÉ DEBE CONTENER	POR QUÉ IMPORTA
Casos frecuentes	Preguntas o tareas que aparecen cada semana.	Miden utilidad cotidiana.
Casos frontera	Entradas ambiguas, incompletas o con varias interpretaciones.	Enseñan si el sistema pide aclaración.
Casos sin evidencia	Preguntas que el corpus no permite responder.	Miden abstención.
Casos de formato	Salidas que deben cumplir contrato.	Evitan romper integraciones.
Casos por segmento	Idioma, canal, tipo de usuario, producto o país.	Detectan media buena con subgrupo malo.
Casos de regresión	Fallos reales convertidos en test permanente.	Evitan repetir errores ya vistos.

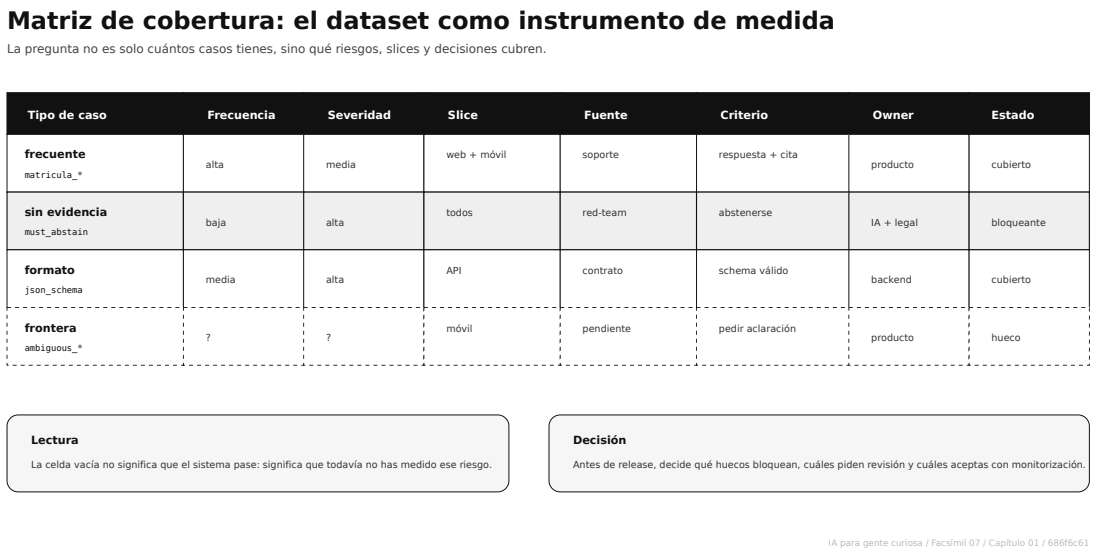
La documentación de modelos también importa. Las model cards nacen para registrar detalles de uso previsto, factores, métricas, datos de evaluación y consideraciones de despliegue.[11] En un sistema aplicado, la scorecard de eval cumple una función parecida para una release concreta: dice qué hemos probado, con qué límites y con qué resultado.

Matriz de cobertura: lo que el promedio no enseña

Antes de celebrar una métrica, conviene mirar qué cubre el dataset. Una matriz de cobertura cruza los tipos de caso con dimensiones que sí importan en producción: intención, slice, frecuencia, severidad, fuente, riesgo y owner. Si una celda está vacía, la eval no está diciendo “todo va bien”; está diciendo “aquí no he mirado”.

DIMENSIÓN	PREGUNTA DE INGENIERÍA	EJEMPLO
Intención	¿Qué quiere hacer el usuario o sistema?	Matrícula, becas, soporte, reclamación.
Slice	¿Qué subgrupo puede comportarse distinto?	Idioma, canal, país, perfil, producto.
Severidad	¿Qué daño produce el fallo?	Molestia, bloqueo, privacidad, coste, seguridad.
Frecuencia	¿Cuánto aparece en uso real?	Diario, semanal, raro pero crítico.
Fuente	¿De dónde sale el caso?	Producción, soporte, experto, red-team, incidente.
Criterio de aceptación	¿Cómo sabremos si pasó?	Cita válida, abstención, JSON válido, tool correcta.
Owner	¿Quién responde si falla?	Equipo de RAG, producto, legal, operación.

El truco está en no llenar la matriz por estética. Si un caso es raro pero severo, merece sitio aunque aparezca poco. Si un caso es muy frecuente pero de bajo riesgo, puede tener más muestras para estimar estabilidad. Esta es una decisión de ingeniería, no una decoración de dataset.



Matriz de cobertura: no basta con contar casos; hay que saber qué riesgos y decisiones cubren.

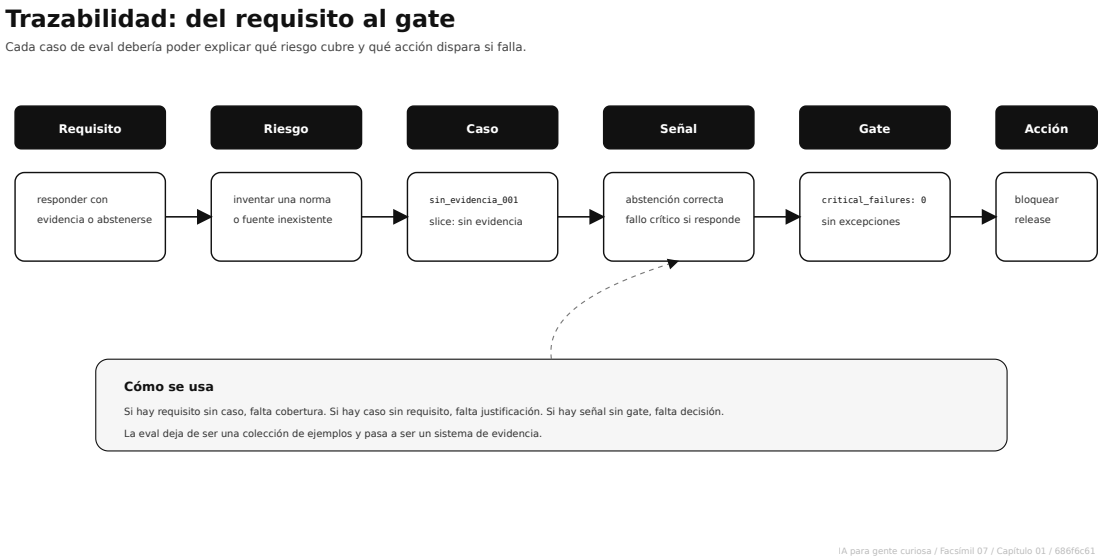
Matriz de trazabilidad: por qué existe cada caso

La matriz de cobertura responde “qué zonas mira el dataset”. La matriz de trazabilidad responde otra pregunta: **por qué existe cada caso y qué decisión protege**. En proyectos de IA esto es muy útil porque evita datasets llenos de ejemplos bonitos pero difíciles de defender. Si un caso no conecta con un requisito, un riesgo o una decisión, quizá no sobra, pero todavía no sabemos qué papel cumple.

Una matriz de trazabilidad no tiene que ser burocrática. Puede vivir como tabla en un README, como JSON junto al dataset o como columnas en una hoja de revisión. Lo importante es que permita seguir la cadena completa: requisito de producto, riesgo si falla, caso que lo cubre, métrica que lo mide, gate que decide y acción técnica si se rompe.

REQUISITO	RIESGO	CASO	SEÑAL	GATE	ACCIÓN SI FALLA
Responder solo con evidencia.	Inventar una norma interna.	`sin_evidencia_001`.	Abstención correcta y fallo crítico.	Cero fallos críticos.	Bloquear release y añadir regresión.
Mantener contrato JSON.	Romper integración downstream.	`json_schema_004`.	Validador determinista.	JSON válido en todos los casos críticos.	Rechazar PR y corregir parser.
No subir coste por tarea aceptada.	Hacer inviable el sistema.	`cost_slice_soporte`.	Coste por aceptada y p95.	Presupuesto por caso y por release.	Revisar modelo, routing o longitud.

Esta matriz también ayuda a detectar huecos. Si tienes requisitos sin casos, estás confiando en la suerte. Si tienes casos sin requisito, quizá estás midiendo algo que no cambia ninguna decisión. Y si tienes métricas sin acción, probablemente estás produciendo un dashboard, no una eval.



Matriz de trazabilidad: requisito, riesgo, caso, señal, gate y acción tienen que poder seguirse de punta a punta.

Hipótesis evaluable antes de tocar nada

Una eval seria no empieza ejecutando un script. Empieza escribiendo una hipótesis que pueda salir bien o mal. Si no puedes escribirla, probablemente todavía no sabes qué estás intentando mejorar.

Una forma mínima de escribirla no es una ecuación, sino una ficha de decisión. Sirve para que un Pull Request o una release no cambie algo sin declarar efecto, métrica, riesgo y acción.

CAMPO	QUÉ OBLIGA A DECIDIR	EJEMPLO
Cambio propuesto	Qué tocamos exactamente.	Añadir instrucción de citar fuente y fecha.
Efecto esperado	Qué mejora esperamos observar.	Sube groundedness y baja respuesta sin evidencia.
Métrica que lo comprueba	Cómo sabremos si pasó.	Groundedness, abstención correcta, coste por aceptada.
Riesgo vigilado	Qué puede empeorar aunque la media suba.	Respuestas más largas, más coste, más latencia.
Acción si falla	Qué haremos si la evidencia no acompaña.	Mantener baseline y añadir casos de regresión.

Ejemplo escrito como lo pondríamos en un Pull Request:

CAMPO	CONTENIDO
Cambio	Sustituimos el prompt de respuesta libre por uno que exige cita o abstención.
Efecto esperado	La tasa de respuestas con evidencia verificable sube de 0,78 a 0,86.
Riesgo	El modelo puede sobre-abstenerse o subir coste por respuestas más largas.
Métrica primaria	Groundedness ponderado.
Métricas de guardia	Abstención correcta, coste por aceptada, p95 de latencia y regresiones por slice.
Gate	Aceptar solo si `groundedness >= 0.86`, `critical_failures == 0` y `cost_per_accepted <= 0.04`.

La hipótesis evita dos males muy comunes: cambiar varias cosas a la vez sin saber cuál ayudó, y declarar “mejor” algo que solo cambió el estilo.

Etiquetado y acuerdo entre revisores

Si una eval necesita etiquetas humanas, entonces también necesita una guía de etiquetado. No basta con decir “que alguien lo revise”. Hay que definir qué significa correcto, parcialmente correcto, incorrecto, no respondible, cita válida, salida útil y fallo crítico.

Una guía mínima de etiquetado debería responder:

PREGUNTA	DECISIÓN PRÁCTICA
¿Quién etiqueta?	Dos personas para una muestra inicial y una persona para el resto si el acuerdo es suficiente.
¿Qué ve quien etiqueta?	Input, output, evidencia recuperada, referencia esperada y rúbrica.
¿Qué no debería ver?	Nombre del modelo si queremos reducir sesgo de marca.
¿Qué etiquetas existen?	`pass`, `partial`, `fail`, `must_abstain`, `critical_failure`.
¿Cómo se resuelve desacuerdo?	Tercera revisión o reunión corta para cambiar la guía, no para forzar unanimidad silenciosa.
¿Qué se guarda?	Revisor, fecha, versión de guía, etiqueta y comentario breve.

Antes de aplicar una medida académica, lo mínimo es mirar el acuerdo observado: cuántas veces coinciden dos revisores sobre los mismos casos. Si revisan 100 respuestas y coinciden en 82, el acuerdo observado es 0,82. Sirve para empezar, pero tiene una trampa: si casi todo pertenece a una clase fácil, dos personas podrían coincidir mucho por azar o por distribución de etiquetas. Por eso el acuerdo observado es una señal inicial, no el final de la historia.

Pero el acuerdo simple no corrige coincidencias por azar. Cohen propuso kappa para medir acuerdo entre dos codificadores en categorías nominales teniendo en cuenta el acuerdo esperado por azar.[1][2]

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
κ	Acuerdo corregido por azar.	0,71.
p_o	Acuerdo observado.	0,82.
p_e	Acuerdo esperado por azar según las distribuciones de etiquetas.	0,38.

Jacob Cohen fue un psicólogo y estadístico estadounidense muy influyente en medición psicológica y tamaño del efecto. Kappa aparece aquí porque una eval de IA muchas veces depende de etiquetas humanas: correcto, parcial, incorrecto, debe abstenerse, fallo crítico. Si las etiquetas no son estables entre revisores, la evaluación mide una mezcla de calidad del modelo y ambigüedad de la rúbrica. En ingeniería, kappa no se usa para presumir de estadística; se usa para saber si el instrumento de medida merece confianza.

No hace falta convertir kappa en una religión. Lo importante para ingeniería es más sencillo: si dos personas no se ponen de acuerdo siguiendo la misma guía, el problema no está en el modelo; está en la definición de calidad.

Calibración de revisores humanos

Cuando una eval usa personas para etiquetar, el trabajo no consiste en repartir casos y sumar votos. Primero hay que calibrar a quienes revisan. Calibrar significa que dos personas leen la misma guía, revisan una muestra común, comparan discrepancias y ajustan la guía hasta que el criterio sea suficientemente estable. Si saltas este paso, puedes acabar midiendo preferencias personales con apariencia de métrica.

Un protocolo razonable empieza con una muestra piloto pequeña y variada. No buscamos "ganar kappa" a cualquier precio; buscamos descubrir ambigüedades. Si una persona marca `partial` y otra marca `fail`, quizá no hay un problema de atención, sino una definición incompleta de "respuesta útil". La guía se mejora con ejemplos frontera, contraejemplos y reglas de desempate. Después se vuelve a etiquetar una muestra, se mide acuerdo y solo entonces se escala al resto del dataset.

PASO	QUÉ SE HACE	QUÉ EVIDENCIA DEJA
Muestra piloto	Dos revisores etiquetan los mismos casos variados.	Tabla con etiquetas, desacuerdos y comentarios.
Revisión de discrepancias	Se discute la causa, no quién "tenía razón".	Cambios concretos en la guía.
Congelación de guía	Se versiona la guía antes de etiquetar en serio.	`labeling_guide.md@v1`.
Medición de acuerdo	Se calcula acuerdo observado y, si aplica, kappa.	Señal de estabilidad del instrumento.
Muestreo de control	Se reetiqueta una parte periódicamente.	Detección de drift del revisor o de la tarea.

Este punto es muy importante en sistemas generativos porque muchas etiquetas no son obvias. "Respuesta correcta" puede ser demasiado pobre. A veces necesitas separar factualidad, completitud, utilidad, tono, cita, formato, abstención y severidad. Si todo eso vive en una sola etiqueta, el desacuerdo humano se vuelve inevitable y la eval pierde fuerza.

Baseline, candidate y regresión

Evaluar una versión aislada dice poco. Lo que necesitamos casi siempre es comparación: candidate frente a baseline, caso por caso, con el mismo dataset y los mismos graders. La pregunta no es solo si candidate tiene una media mayor; la pregunta importante es qué casos mejora y qué casos rompe.

Para eso guardamos una lista explícita de regresiones: casos que baseline pasaba y candidate falla. No hace falta vestirlo como notación matemática. En una scorecard real puede aparecer como una lista de identificadores, severidades, slices y causas técnicas.

LECTURA	QUÉ SIGNIFICA	DECISIÓN TÍPICA
Candidate mejora varios casos y no rompe ninguno conocido.	Hay señal positiva, todavía pendiente de mirar incertidumbre y cobertura.	Puede pasar a revisión de release.
Candidate mejora la media, pero rompe un caso crítico.	La mejora agregada es engañosa.	Bloquear y añadir el caso a regresión.
Candidate mejora solo casos fáciles y empeora un slice sensible.	La cobertura del dataset está avisando de riesgo.	Revisar dataset, prompt, retrieval o política.
Candidate empata casi todo y cuesta más.	No hay razón técnica clara para cambiar.	Mantener baseline o buscar routing selectivo.

Si aparece una regresión crítica, no basta con decir “pero el promedio sube”. Esa frase es exactamente el tipo de pensamiento que una eval debe impedir.

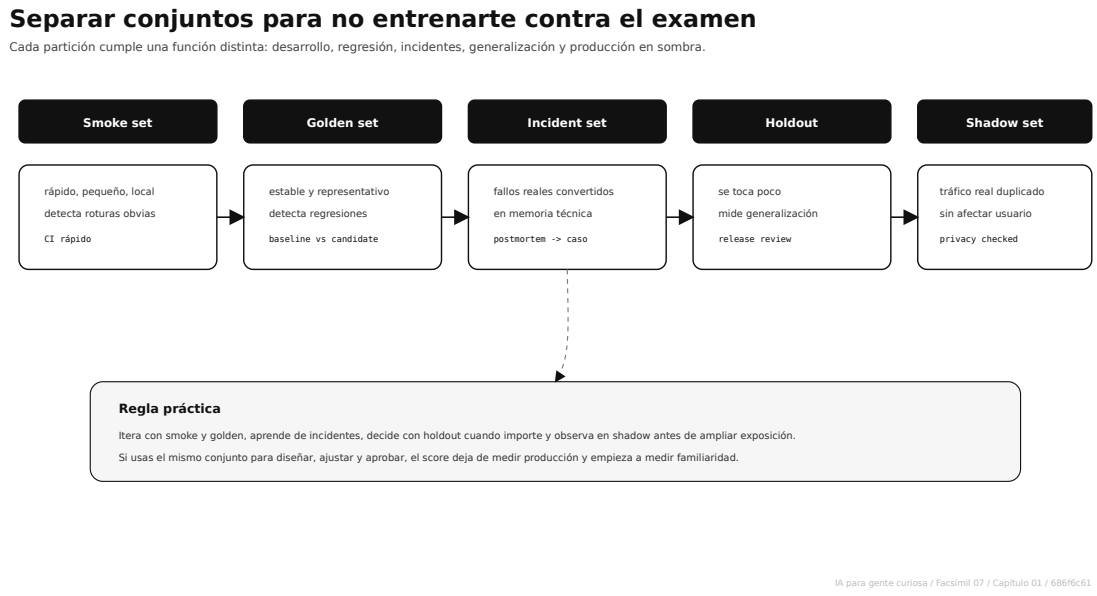
Cómo no sobreajustar contra tu propia eval

Una eval puede morir de éxito. Al principio descubre fallos. El equipo corrige prompt, retrieval, parsers o herramientas. Vuelve a ejecutar. Corrige otra vez. Al cabo de unas cuantas iteraciones, el sistema ya no está mejorando necesariamente en la tarea real: puede estar aprendiendo a pasar ese conjunto concreto. En aprendizaje automático esto se parece al sobreajuste; en ingeniería de producto se ve como “nuestro dashboard sube, pero producción sigue dando sustos”.

La forma práctica de evitarlo no es esconder el dataset a todo el mundo. Es separar conjuntos con funciones distintas. Un conjunto rápido sirve para desarrollo local. Un golden set estable sirve para regresión. Un conjunto de incidentes conserva memoria de fallos reales. Y un holdout, que se toca menos, ayuda a comprobar si la mejora sale del entorno donde se iteró.

CONJUNTO	PARA QUÉ SIRVE	QUÉ NO DEBES HACER
Smoke set	Comprobación rápida en local o CI.	Usarlo como prueba de calidad final.
Golden set	Detectar regresiones conocidas antes de publicar.	Ajustar cada cambio mirando solo ese score.
Incident set	Convertir fallos reales en pruebas permanentes.	Dejar incidentes en conversaciones perdidas.
Holdout	Medir generalización con menos contaminación.	Abrirlo cada vez que una variante no gusta.
Shadow set	Observar tráfico real sin afectar al usuario.	Mezclarlo sin anonimizar ni revisar privacidad.

Sculley y coautores avisaron de la deuda técnica oculta en sistemas de ML: las dependencias de datos, configuraciones, feedback loops y cambios de mundo pueden volver frágil un sistema que parecía correcto en laboratorio.[13] En evals de IA ocurre algo parecido. Si no versionas datasets, graders y casos de regresión, no sabes si el sistema mejoró o si el examen cambió debajo de tus pies.



Partición de datasets de eval: desarrollo rápido, regresión estable, memoria de incidentes, holdout y tráfico sombra no cumplen la misma función.

Incertidumbre: no creas demasiado en un decimal

Una eval de 20 casos no tiene la misma fuerza que una eval de 2.000. Si candidate obtiene 0,87 y baseline 0,85, quizá hay mejora real. O quizá hemos visto ruido de muestra. La estadística no está para adornar: está para impedir que tomemos decisiones caras sobre diferencias frágiles.

Para comparar dos versiones sobre los mismos casos, lo primero es mirar comparación pareada:

SITUACIÓN DEL CASO	QUÉ SIGNIFICA
Baseline pasa y candidate pasa.	No informa sobre diferencia entre versiones.
Baseline falla y candidate falla.	Tampoco informa sobre diferencia.
Baseline falla y candidate pasa.	Mejora directa.
Baseline pasa y candidate falla.	Regresión directa.

McNemar propuso una prueba para diferencias entre proporciones correlacionadas, justo el tipo de situación que aparece cuando dos clasificadores o dos versiones se evalúan sobre los mismos casos.[14] En lectura de ingeniería, la idea práctica es que solo importan los casos discordantes:

$$\chi^2 = \frac{(|b - c| - 1)^2}{b + c}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
b	Casos donde baseline falla y candidate pasa.	12 mejoras.
c	Casos donde baseline pasa y candidate falla.	3 regresiones.
χ^2	Estadístico aproximado de McNemar con corrección de continuidad.	7,11.

Quinn McNemar fue un psicólogo y estadístico asociado a métodos de medida para datos emparejados. Su prueba es útil aquí porque baseline y candidate se evalúan sobre los mismos casos, no sobre muestras independientes. Eso cambia la pregunta: no queremos saber solo cuántos aciertos tiene cada versión, sino en qué casos discrepan. Los casos donde ambas pasan o ambas fallan no explican la diferencia entre versiones; los discordantes sí.

No vamos a convertir este capítulo en un curso de inferencia, pero sí debemos llevarnos la intuición: **si mejoras 12 casos y rompes 3, la lectura es distinta que si mejoras 4 y rompes 3**.

También podemos estimar incertidumbre con bootstrap: re-muestrear los casos con reemplazo muchas veces, recalcular la diferencia de score y mirar el rango donde cae la mayoría de diferencias. Efron introdujo el bootstrap moderno como método de remuestreo para estimar la variabilidad de estadísticos sin depender de una fórmula cerrada para cada caso.[15]

Bradley Efron, estadístico de Stanford, formalizó el bootstrap moderno a finales de los setenta. La idea es muy útil en evals pequeñas: si no puedes repetir el mundo real 2.000 veces, re-muestreas tus casos con reemplazo para estimar cómo variaría la métrica. No convierte cinco casos en mil casos reales, pero te recuerda que una diferencia puntual puede ser frágil. Si el intervalo es enorme, el mensaje de ingeniería es humilde: falta evidencia, no entusiasmo.

Lectura práctica:

RESULTADO	QUÉ HARÍA
Intervalo claramente por encima de 0 y sin fallos críticos.	Candidate parece mejorar de forma consistente.
Intervalo cruza 0.	No hay evidencia fuerte de mejora; pediría más casos o más análisis.
Intervalo mejora, pero hay regresión crítica.	No publicaría; arreglaría esa clase de fallo primero.
Intervalo mejora, pero coste se dispara.	Miraría coste por aceptada y routing antes de decidir.

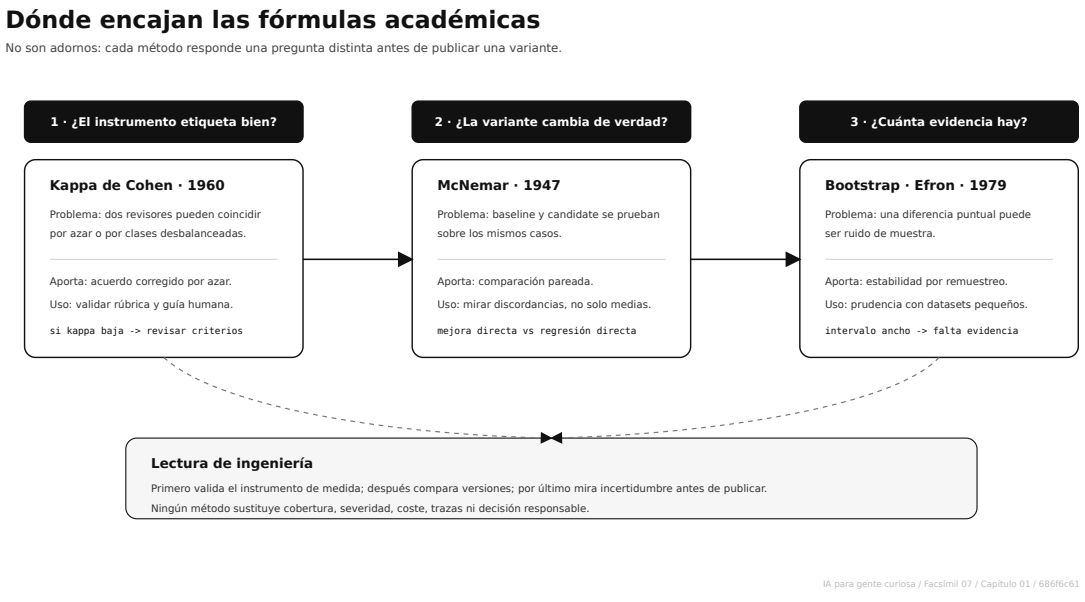
Fórmulas académicas que sí aparecen

En este capítulo solo dejamos fórmulas matemáticas cuando son reconocibles en la literatura y están citadas. El resto queda escrito como procedimiento, tabla o ejemplo numérico. La razón es pedagógica: una expresión operativa con aspecto matemático puede parecer más científica de lo que realmente es.

FÓRMULA	DE DÓNDE VIENE	QUÉ APORTA EN UNA EVAL	CUIDADO PRÁCTICO
$\kappa = \frac{p_o - p_e}{1 - p_e}$	Kappa de Cohen, propuesta por Jacob Cohen en 1960 para acuerdo entre codificadores.	Corrige el acuerdo observado por el acuerdo esperado al azar.	Ayuda a validar la rúbrica, pero no arregla una guía mal escrita. Si kappa sale bajo, se revisa el instrumento.
$\chi^2 = \frac{(\lvert b-c\rvert - 1)^2}{b+c}$	Prueba de McNemar con corrección de continuidad, publicada por Quinn McNemar en 1947.	Compara dos versiones sobre los mismos casos mirando solo discordancias.	En muestras pequeñas debe leerse con prudencia y junto a la severidad de errores.

Bootstrap también es un método académico, pero aquí no lo reduzco a una fórmula porque su expresión depende del estadístico estimado y del intervalo elegido. Lo importante en este capítulo es saber qué significa: re-muestrear con reemplazo para estimar la estabilidad de una diferencia observada. Si más adelante necesitamos una formulación formal, debe entrar con la referencia completa y el contexto estadístico correspondiente.

La lectura universitaria sería esta: una fórmula no merece estar en el capítulo porque parezca técnica, sino porque pertenece a un método reconocido, ayuda a tomar una decisión mejor y sabemos explicar sus límites. Si no podemos sostener origen, uso y riesgo, mejor usar prosa, pseudocódigo o un ejemplo numérico real.



Mapa de métodos académicos: kappa valida el etiquetado, McNemar compara versiones pareadas y bootstrap recuerda cuánta incertidumbre queda.

Riesgos de una eval que parece seria

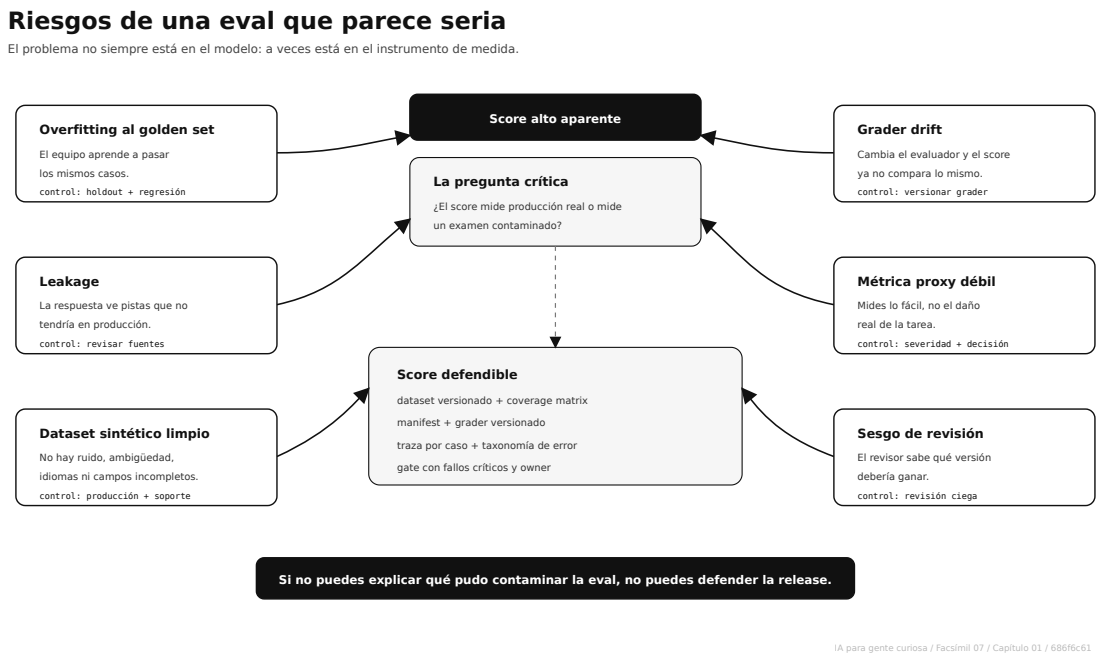
Una eval puede tener JSON, dashboards y fórmulas y aun así estar mal diseñada. El problema no es que falte tecnología; es que el instrumento de medida se haya contaminado. En ingeniería de IA esto pasa mucho porque los equipos iteran rápido: miran los fallos, ajustan prompt, vuelven a mirar los mismos casos, cambian el evaluador, vuelven a ejecutar, y al cabo de unos días la eval ya no mide generalización; mide cuánto hemos aprendido a pasar ese examen.

En investigación experimental se habla de amenazas a la validez para separar varios problemas: si medimos el concepto correcto, si el diseño permite atribuir el efecto al cambio, si la conclusión estadística es razonable y si el resultado generaliza fuera del experimento. Campbell y Stanley popularizaron esta forma de pensar en diseños experimentales, y Messick amplió la idea de validez como interpretación defendible de una medición, no como una propiedad mágica del test.[16][17]

Traducido a nuestro terreno: una eval de IA no es válida porque tenga muchas filas. Es más válida cuando sus casos representan la tarea real, sus graders miden el comportamiento que dicen medir, sus comparaciones son justas y sus conclusiones no prometen más de lo que los datos permiten.

TIPO DE VALIDEZ	PREGUNTA EN UNA EVAL DE IA	FALLO TÍPICO
Validez de constructo	¿La métrica mide lo que llamamos calidad, groundedness, seguridad o utilidad?	Usar longitud de respuesta como proxy de completitud.
Validez interna	¿La mejora se debe al cambio probado o a otra cosa que también cambió?	Cambiar modelo, prompt y dataset a la vez.
Validez externa	¿Lo medido se parece a producción?	Dataset limpio, corto y sin ruido real.
Validez de conclusión	¿La evidencia es suficiente para sostener la decisión?	Celebrar una diferencia pequeña en 20 casos sin mirar incertidumbre.

RIESGO	QUÉ SIGNIFICA	SEÑAL DE ALERTA	ANTÍDOTO
Overfitting al golden set	Ajustas prompt, retrieval o modelo hasta pasar los mismos casos.	El score sube en eval fija, pero fallan casos nuevos parecidos.	Separar smoke set, regression set y holdout; añadir casos por análisis de error.
Leakage	El modelo o sistema ve información que no tendría en producción.	El caso parece resuelto por memoria o por pista escondida en el prompt.	Revisar fuentes, contexto recuperado, datos de entrenamiento y plantillas.
Grader drift	Cambia el evaluador, rúbrica o modelo evaluador.	Comparas scores de fechas distintas como si midieran lo mismo.	Versionar graders, prompts de evaluación y criterios humanos.
Dataset sintético demasiado limpio	Los casos no parecen producción real.	Inputs perfectos, sin ambigüedad, sin ruido, sin idiomas raros, sin campos incompletos.	Mezclar producción, soporte, expertos, red-team e incidentes.
Métrica proxy mal elegida	Mides algo fácil que no representa el daño real.	Mejora exact match, pero empeora utilidad o seguridad.	Conectar cada métrica con una decisión y una consecuencia.
Sesgo de revisión	El revisor sabe qué versión generó la salida.	Candidate recibe más indulgencia porque “debería ser mejor”.	Etiquetado ciego cuando el juicio humano sea importante.



Mapa de riesgos de medición: un score alto no basta si el dataset, el grader o la revisión están contaminados.

La solución no es desconfiar de todo, sino documentar. Un manifest honesto debe decir qué dataset se usó, qué versión del grader puntuó, qué casos se añadieron, qué quedó fuera y qué decisión permite tomar. Si esa explicación no existe, el número puede ser correcto y aun así no ser defendible.

Coste por tarea aceptada

En IA aplicada, el coste por llamada no suele ser la métrica que decide. Decide el coste por salida aceptada.

Este cálculo mezcla inferencia, tools, reintentos y revisión humana porque son las partidas que suelen aparecer en una aplicación de IA. En tu sistema quizá faltará almacenamiento, anotación, observabilidad o coste de oportunidad. Lo importante no es memorizar una ecuación, sino no comparar solo precio por llamada.

Ejemplo numérico:

PARTIDA	COSTE
Inferencia	0,70 €
Herramientas externas	0,18 €
Reintentos	0,22 €
Revisión humana	1,70 €
Coste total	**2,80 €**
Tareas aceptadas	90
Coste por tarea aceptada	**0,031 €**

Si una versión nueva cuesta el doble pero reduce mucho revisión humana, quizá es más barata por tarea aceptada. Si una versión barata genera más rechazos, quizá sale cara aunque el precio por token parezca atractivo.

En el día a día

En un equipo de ingeniería, una eval debería vivir como un artefacto versionado:

ARCHIVO	QUÉ CONTIENE	QUIÉN LO USA
`eval_cases.jsonl`	Casos de evaluación con input, criterios y metadatos.	Ingeniería, producto, datos.
`eval_hypothesis.json`	Cambio, efecto esperado, métricas primarias, métricas de guardia y acción si falla.	Autor del cambio y reviewer.
`eval_policy.json`	Umbrales, pesos, fallo crítico y presupuesto.	Tech lead, operación, producto.
`labeling_guide.md`	Guía para etiquetar casos y resolver desacuerdos.	Revisores humanos y docentes.
`error_taxonomy.json`	Catálogo de errores que convierte fallos en acciones técnicas.	Ingeniería y análisis de errores.
`eval_run_manifest.json`	Versiones, hashes, parámetros, fecha y owner de la corrida.	Auditoría técnica y reproducibilidad.
`eval_runner.py`	Script reproducible que ejecuta y puntúa.	CI, desarrollo local.
`scorecard.json`	Resultado de una corrida concreta.	Pull request, release, runbook.
`decision.md`	Lectura humana de aceptar, rechazar o revisar.	Equipo y responsables de decisión.

Amershi y coautores describen cómo los sistemas de ML introducen necesidades especiales en ingeniería de software: datos, evaluación, monitorización, experimentación y evolución del comportamiento.[18] TFX también se diseñó alrededor de pipelines reproducibles que integran datos, entrenamiento, validación y serving.[19]

La eval es una pieza de ese mismo mundo. No es un notebook olvidado; es parte del sistema.

Una buena regla de trabajo: **si dentro de dos semanas no puedes repetir la eval y explicar por qué salió lo que salió, no tenías una evaluación; tenías una medición suelta**.

Cómo entra en CI/CD

En un proyecto real, una eval debería aparecer en el Pull Request como aparece un test de integración: no para sustituir la revisión humana, sino para dejar evidencia. El cambio puede ser un prompt, un modelo, un retriever, un ranking, una herramienta o una política de abstención. El pipeline ejecuta baseline y candidate sobre el conjunto acordado, genera artefactos y deja una decisión legible.

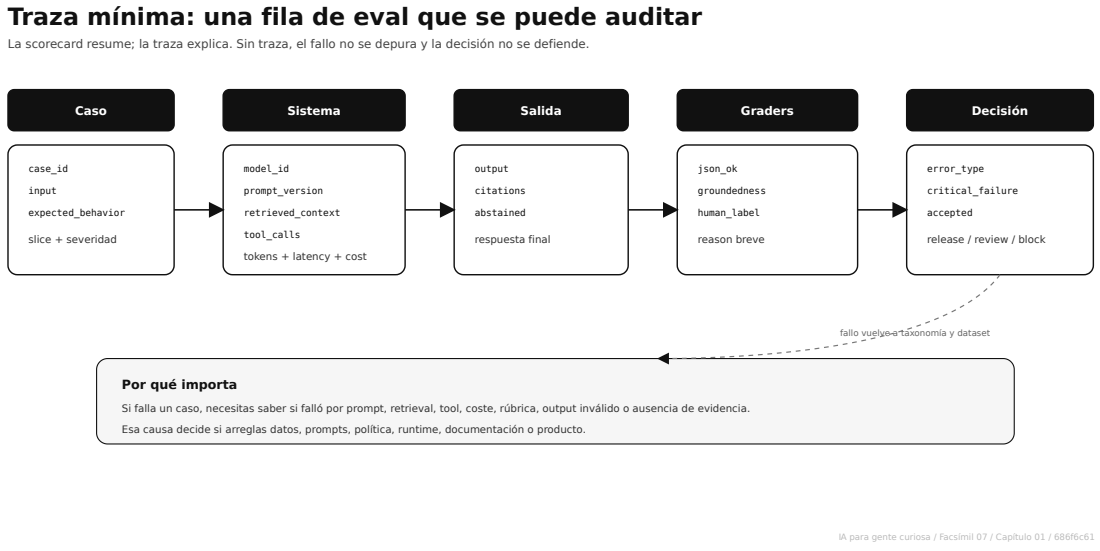
PASO	QUÉ OCURRE	ARTEFACTO ESPERADO
Cambio	Alguien modifica prompt, modelo, RAG, tool o política.	Diff revisable.
Ejecución	El runner evalúa baseline y candidate con el mismo dataset.	`eval_scorecard.json`.
Gate	Se aplican umbrales, fallos críticos, coste y regresiones.	Decisión `release`, `needs_review` o `block`.
Evidencia	El PR adjunta scorecard, manifest, hashes y resumen humano.	`decision.md`.
Aprendizaje	Cada fallo relevante vuelve como caso de regresión.	Nuevo caso versionado.

La señal importante no es “CI verde” sin contexto. Es poder abrir la scorecard y leer: qué se probó, qué cambió, qué falló, cuánto cuesta, qué incertidumbre hay y quién toma la decisión. Si el pipeline solo dice `passed`, obliga al equipo a confiar en una caja negra. Una eval bien hecha hace justo lo contrario: reduce magia.

Traza mínima por caso evaluado

Una scorecard agregada sirve para decidir, pero la depuración vive en la traza por caso. Cada fila debería poder explicar por qué un caso pasó, falló o quedó en revisión. En IA moderna, esa traza suele incluir más que input y output: contexto recuperado, llamadas a herramientas, parámetros del modelo, tokens, coste, latencia, grader, tipo de error y decisión.

CAMPO DE TRAZA	PARA QUÉ SIRVE
`case_id`	Permite volver al caso exacto y convertirlo en regresión.
`input`	Entrada evaluada, sin depender de memoria del equipo.
`expected_behavior`	Qué debía ocurrir: responder, citar, llamar herramienta, abstenerse.
`retrieved_context`	Evidencia que recibió el sistema si hay RAG.
`tool_calls`	Acciones externas ejecutadas o simuladas.
`model_id` y `prompt_version`	Reproducibilidad y comparación entre versiones.
`tokens`, `latency_ms`, `cost_eur`	Coste operativo real, no solo calidad.
`grader_result`	Señal de evaluación y explicación breve.
`error_type`	Taxonomía que convierte fallo en acción técnica.
`decision`	Qué hace el gate con ese caso o con la corrida.



Traza mínima por caso: lo que permite depurar una eval y defender una decisión técnica.

Por qué debería importarte

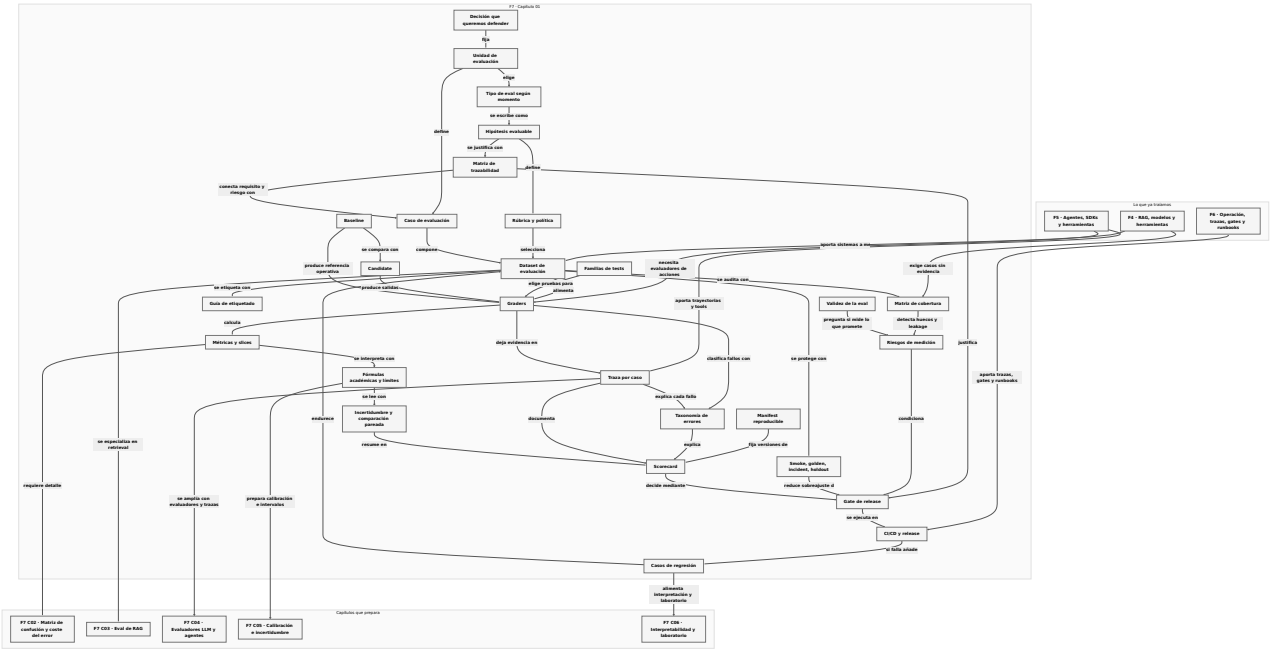
Porque una eval te protege de publicar por sensación. En sistemas de IA, una demo buena suele enseñar lo que el sistema puede hacer; una eval útil enseña lo que todavía puede romper. Esa diferencia cambia conversaciones enteras: en vez de discutir si “parece mejor”, el equipo mira casos, regresiones, coste, incertidumbre y fallos críticos.

También importa porque convierte calidad en algo revisable. Producto puede discutir si el umbral tiene sentido, ingeniería puede revisar el runner y los hashes, datos puede mirar cobertura de slices, y operación puede decidir si el gate bloquea, avisa o deja pasar con seguimiento. No es burocracia: es la forma de no depender de memoria, entusiasmo o autoridad.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Confundir una demo con una eval	Una demo sirve para explorar y una eval sirve para decidir.	Escribir por adelantado qué acción tomarás si el resultado sale bien, mal o dudoso.
Mirar solo la media	Una media puede ocultar que un segmento empeora o que un caso crítico falla.	Mirar slices, regresiones y fallos críticos antes de celebrar el score global.
Cambiar el evaluador y comparar como si nada	Si cambias prompt, modelo o rúbrica del evaluador, cambiaste el instrumento de medida.	Versionar graders igual que versionas código.
No guardar casos de producción	Un fallo real que no vuelve al dataset es una oportunidad perdida.	Convertir cada incidente relevante en caso de regresión.
No conectar coste con aceptación	El precio por llamada puede engañar.	Medir coste por tarea aceptada y separar inferencia, tools, reintentos y revisión.

Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN BREVE
Eval	Diseño reproducible para medir una versión y tomar una decisión.
Unidad de evaluación	Objeto mínimo que se mide: respuesta, conversación, tarea, traza o release.
Caso de evaluación	Entrada versionada con criterio, fuente, slice, riesgo, grader y acción si falla.
Dataset de evaluación	Casos reservados para medir comportamiento de forma comparable.
Golden set	Conjunto estable de casos usado para detectar regresiones antes de publicar.
Holdout	Parte reservada que no se usa para iterar y que ayuda a comprobar generalización.
Matriz de cobertura	Tabla que cruza caso, slice, severidad, frecuencia, fuente, criterio y owner.
Matriz de trazabilidad	Relación entre requisito, riesgo, caso, señal, gate y acción técnica.
Rúbrica	Criterios escritos que explican qué significa hacerlo bien.
Baseline	Versión actual o de referencia.
Candidate	Variante nueva que se compara contra baseline.
Shadow eval	Evaluación con tráfico real duplicado sin afectar la respuesta que recibe el usuario.
Canary	Publicación limitada con guardrails, métricas vivas y rollback preparado.
Grader	Evaluador que transforma una salida en puntuación o veredicto.
Grader drift	Cambio del evaluador que vuelve incomparables dos scores de fechas distintas.
Gate	Regla de decisión que acepta, bloquea o manda a revisión.
Scorecard	Resultado resumido de una corrida de evaluación.
Traza de evaluación	Registro por caso que explica input, sistema, salida, grader, coste, error y decisión.
Leakage	Contaminación que hace que la eval mida información que el sistema no tendría en producción.
Hipótesis evaluable	Cambio esperado expresado como efecto medible y riesgo vigilado.

TÉRMINO	DEFINICIÓN BREVE
Validez de constructo	Grado en que una eval mide realmente el concepto que dice medir.
Test metamórfico	Prueba basada en relaciones esperadas entre entradas transformadas y salidas.
Manifest de evaluación	Registro de versiones, hashes, parámetros y contexto de una corrida.
Acuerdo entre revisores	Medida de coincidencia entre personas que etiquetan los mismos casos.
Kappa de Cohen	Acuerdo entre dos revisores corregido por coincidencias esperadas por azar.
Bootstrap	Remuestreo con reemplazo para estimar incertidumbre de una métrica.
Intervalo de confianza	Rango que expresa cuánta incertidumbre tiene una estimación.
McNemar	Comparación pareada para ver si dos versiones difieren en sus aciertos.
Taxonomía de errores	Catálogo que convierte fallos en causas y acciones técnicas.
Slice	Subconjunto del dataset con una característica común.
Regresión	Caso que antes pasaba y ahora falla.
Fallo crítico	Error que bloquea aunque la media sea buena.
Coste por aceptada	Coste total dividido por salidas que realmente pasan criterios.

Antes de pasar página

Antes de avanzar al siguiente capítulo, deberías poder responder:

1. ¿Qué diferencia hay entre una demo, un benchmark público y una eval propia?
2. ¿Por qué una eval debe empezar por la decisión que permite tomar?
3. ¿Qué cambia si la unidad evaluada es una respuesta, una tarea, una traza o una release?
4. ¿Qué piezas mínimas necesita una eval para pasar de opinión a decisión defendible?
5. ¿Qué campos debería tener un caso de evaluación para poder auditarse?
6. ¿Por qué una matriz de trazabilidad evita datasets decorativos?
7. ¿Por qué un gate puede fallar aunque la puntuación media sea alta?
8. ¿Qué debería contener una hipótesis evaluable antes de cambiar modelo o prompt?

9. ¿Por qué una guía de etiquetado puede ser más importante que añadir otro modelo que evalúe?
10. ¿Qué te dice un intervalo bootstrap que no te dice una media sola?
11. ¿Por qué McNemar mira solo los casos discordantes entre baseline y candidate?
12. ¿Qué fórmulas académicas aparecen en el capítulo, quién las propuso y qué límite práctico tienen?
13. ¿Qué diferencia hay entre smoke set, golden set, incident set, holdout, shadow eval y canary?
14. ¿Qué tipos de grader y familias de test conviene combinar y cuándo usarías cada uno?
15. ¿Por qué los casos sin evidencia son importantes en sistemas RAG y asistentes?
16. ¿Qué significa validez de constructo en una eval de IA?
17. ¿Qué campos mínimos debería guardar una traza de evaluación por caso?
18. ¿Qué significa leakage en una eval y por qué puede invalidar una métrica aparentemente buena?
19. ¿En qué pieza de la política de evaluación deberían vivir los umbrales de decisión?
20. ¿Qué entregarías para demostrar que tu eval es reproducible?

En resumen

IDEA	QUÉ TE LLEVAS
Una eval existe para decidir.	Si no termina en aceptar, rechazar, limitar o revisar, le falta la parte más importante.
La unidad de evaluación cambia la lectura.	No es lo mismo medir una respuesta que una tarea, una traza de agente o una release.
Un caso de eval es una unidad de evidencia.	Debe tener fuente, criterio, slice, riesgo, grader y acción si falla.
El dataset es el instrumento de medida.	Casos pobres producen métricas pobres, aunque el gráfico sea bonito.
La trazabilidad protege la decisión.	Requisito, riesgo, caso, métrica, gate y acción deberían poder seguirse de punta a punta.
Hay familias de tests.	Determinista, regresión, metamórfico, adversarial, contrato de tool y observabilidad no detectan lo mismo.
Un gate combina media, fallos críticos, coste, incertidumbre y regresiones.	No todo se resuelve con un score global.
Una hipótesis y un manifest evitan decisiones irreproducibles.	Antes de correr, dices qué esperas; después, dejas versiones y hashes para repetir.
El etiquetado también se evalúa.	Si los revisores no coinciden, la métrica no tiene una base estable.
Las fórmulas académicas necesitan contexto.	Kappa de Cohen y McNemar aportan señales concretas; ninguna decide sola.
La eval también puede sobreajustarse.	Smoke, golden, incident, holdout y shadow set cumplen funciones distintas.
La cobertura se diseña.	Hay que mirar frecuencia, severidad, slices, fuente y owner, no solo número total de casos.
La traza explica el fallo.	Sin input, contexto, versión, grader, coste, error y decisión, una scorecard no se puede depurar bien.
El riesgo de medición existe.	Overfitting al golden set, leakage y grader drift pueden hacer que una eval parezca seria y mida mal.
La validez importa.	Una eval debe medir el concepto que dice medir, con diseño justo y conclusión proporcionada.

IDEA	QUÉ TE LLEVAS
La práctica debe ser reproducible.	Una scorecard ejecutable vale más que una opinión bien escrita.
Evaluar es operar antes de publicar.	Las evals conectan desarrollo, CI, release, runbooks e incidencias.

Para saber más

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B. y Zimmermann, T. (2019). Software engineering for machine learning: A case study. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*, 291-300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., Koo, C. Y., Lew, L., Mewald, C., Modi, A. N., Polyzotis, N., Ramesh, S., Roy, S., Whang, S. E. y Wicke, M. (2017). TFX: A TensorFlow-based production-scale machine learning platform. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1387-1395. https://doi.org/10.1145/3097983.3098021

Braintrust. (2026). *Evaluate Systematically*. https://www.braintrust.dev/docs/evaluate

Campbell, D. T. y Stanley, J. C. (1963). *Experimental and quasi-experimental designs for research*. Houghton Mifflin.

Chen, T. Y., Cheung, S. C. y Yiu, S. M. (1998). *Metamorphic testing: A new approach for generating next test cases* (Technical Report HKUST-CS98-01). Hong Kong University of Science and Technology.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1), 37-46. https://doi.org/10.1177/001316446002000104

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1), 1-26. https://doi.org/10.1214/aos/1176344552

EleutherAI. (2026). *Language Model Evaluation Harness*. https://github.com/EleutherAI/lm-evaluation-harness

Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Daumé III, H. y Crawford, K. (2021). Datasheets for datasets. *Communications of the ACM*, 64(12), 86-92. https://doi.org/10.1145/3458723

Hugging Face. (2026). *Evaluate*. https://huggingface.co/docs/evaluate/index

LangChain. (2026). *LangSmith Evaluation*. https://docs.langchain.com/langsmith/evaluation

Liang, P. y otros (2022). *Holistic Evaluation of Language Models*. arXiv. https://arxiv.org/abs/2211.09110

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2), 153-157. https://doi.org/10.1007/BF02295996

Messick, S. (1995). Validity of psychological assessment: Validation of inferences from persons' responses and performances as scientific inquiry into score meaning. *American Psychologist*, 50(9), 741-749. https://doi.org/10.1037/0003-066X.50.9.741

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D. y Gebru, T. (2019). Model cards for model reporting. *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 220-229. https://doi.org/10.1145/3287560.3287596

OpenAI. (2026). *Graders*. https://developers.openai.com/api/docs/guides/graders

OpenAI. (2026). *Working with Evals*. https://developers.openai.com/api/docs/guides/evals

Promptfoo. (2026). *Assertions & metrics*. https://www.promptfoo.dev/docs/configuration/expected-outputs/

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F. y Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems*, 28.

Notas

1. Liang, P. y otros (2022). *Holistic Evaluation of Language Models*. arXiv. https://arxiv.org/abs/2211.09110. Consultado el 28 de mayo de 2026. ↩

2. OpenAI. (2026). *Graders*. https://developers.openai.com/api/docs/guides/graders. Consultado el 28 de mayo de 2026. ↩

3. OpenAI. (2026). *Working with Evals*. https://developers.openai.com/api/docs/guides/evals. Consultado el 28 de mayo de 2026. ↩

4. Braintrust. (2026). *Evaluate Systematically*. https://www.braintrust.dev/docs/evaluate. Consultado el 28 de mayo de 2026. ↩

5. LangChain. (2026). *LangSmith Evaluation*. https://docs.langchain.com/langsmith/evaluation. Consultado el 28 de mayo de 2026. ↩

6. Promptfoo. (2026). *Assertions & metrics*. https://www.promptfoo.dev/docs/configuration/expected-outputs/. Consultado el 28 de mayo de 2026. ↩

7. Hugging Face. (2026). *Evaluate*. https://huggingface.co/docs/evaluate/index. Consultado el 28 de mayo de 2026. ↩

8. EleutherAI. (2026). *Language Model Evaluation Harness*. https://github.com/EleutherAI/lm-evaluation-harness. Consultado el 28 de mayo de 2026. ↩

9. Chen, T. Y., Cheung, S. C. y Yiu, S. M. (1998). Metamorphic testing: A new approach for generating next test cases. *Technical Report HKUST-CS98-01*. Consultado el 28 de mayo de 2026. ↩

.0. Gebru, T. y otros (2021). Datasheets for datasets. *Communications of the ACM*, 64(12), 86-92. https://doi.org/10.1145/3458723 ↩

.1. Mitchell, M. y otros (2019). Model cards for model reporting. *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 220-229. https://doi.org/10.1145/3287560.3287596 ↩

.2. Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1), 37-46. https://doi.org/10.1177/001316446002000104 ↩

.3. Sculley, D. y otros (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems*, 28. Consultado el 28 de mayo de 2026. ↩

.4. McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2), 153-157. https://doi.org/10.1007/BF02295996 ↩

.5. Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1), 1-26. https://doi.org/10.1214/aos/1176344552 ↩

.6. Campbell, D. T. y Stanley, J. C. (1963). *Experimental and quasi-experimental designs for research*. Houghton Mifflin. Consultado el 28 de mayo de 2026. ↩

.7. Messick, S. (1995). Validity of psychological assessment: Validation of inferences from persons' responses and performances as scientific inquiry into score meaning. *American Psychologist*, 50(9), 741-749. https://doi.org/10.1037/0003-066X.50.9.741 ↩

.8. Amershi, S. y otros (2019). Software engineering for machine learning: A case study. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*, 291-300. https://doi.org/10.1109/ICSE-SEIP.2019.00042 ↩

.9. Baylor, D. y otros (2017). TFX: A TensorFlow-based production-scale machine learning platform. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1387-1395. https://doi.org/10.1145/3097983.3098021 ↩

Capítulo 02: Métricas clásicas: matriz de confusión y coste del error

Entrando en el tema

En el capítulo anterior construimos una eval como expediente: hipótesis, casos, unidad de evaluación, scorecard y decisión. Ahora necesitamos una herramienta mucho más humilde y, precisamente por eso, imprescindible: la matriz de confusión.

La matriz de confusión no es una tabla para rellenar por costumbre. Es la contabilidad mínima de una decisión automática. Si tu sistema marca un ticket como urgente, acepta un documento, bloquea una operación, recomienda revisión humana o deja pasar una alerta, tienes cuatro preguntas antes de presumir de métrica:

PREGUNTA	QUÉ TE OBLIGA A MIRAR
¿Cuántas veces acertó cuando debía actuar?	Verdaderos positivos.
¿Cuántas veces actuó cuando no debía?	Falsos positivos.
¿Cuántas veces dejó pasar algo importante?	Falsos negativos.
¿Cuántas veces descartó bien lo que no importaba?	Verdaderos negativos.

Al terminar este capítulo deberías poder hacer esto:

RESULTADO DE APRENDIZAJE	EVIDENCIA DE QUE LO SABES HACER
Leer una matriz de confusión.	Distingues TP, FP, FN y TN sin memorizar siglas vacías.
Calcular métricas básicas.	Obtienes accuracy, precision, recall, specificity, F1, F-beta y balanced accuracy.
Elegir métrica según decisión.	No optimizas accuracy si el error importante vive en una clase minoritaria.
Traducir métricas a coste.	Asignas coste a FP, FN y revisión antes de elegir umbral.
Comparar umbrales.	Escaneas umbrales y eliges por coste, cobertura y restricciones.
Defender una política.	Produces una scorecard con matriz, coste, slices, zona de revisión y decisión escrita.

La idea central es esta: **una métrica no es una medalla; es una forma comprimida de hablar de consecuencias**.

Conviene insistir en esto porque, en proyectos reales, la métrica suele llegar demasiado pronto a la conversación. Alguien pregunta “¿qué F1 tenemos?” antes de haber fijado qué clase es positiva, qué casos entran en evaluación, qué ocurre con los ambiguos, quién revisa los errores y qué decisión se tomará si el número baja. Esa prisa produce evaluaciones aparentemente rigurosas, pero débiles: llenas de decimales, vacías de criterio.

En este capítulo vamos a usar métricas clásicas, sí, pero no como una lista de definiciones. Las vamos a tratar como instrumentos de lectura. Accuracy te dice una cosa, precision otra, recall otra, specificity otra, F1 otra. Ninguna de ellas sabe por sí sola si debes publicar, bloquear, revisar o cambiar el diseño. Esa decisión aparece cuando conectas la métrica con el coste del error, la capacidad de revisión y el contexto donde vive el sistema.

El problema: una accuracy alta puede ser una mala noticia

Imagina un clasificador que decide si un ticket de soporte debe tratarse como urgente. De cada 100 tickets, solo 10 son realmente urgentes. Un sistema perezoso podría decir “ninguno es urgente” y acertar 90 veces.

Eso le da 90 % de accuracy.

Y aun así sería un desastre operativo, porque no detecta ni un ticket urgente. La cifra global queda bonita, pero la decisión es inútil. Este ejemplo aparece una y otra vez en ingeniería de IA: fraude raro, incidentes raros, documentos peligrosos raros, bugs críticos raros, abuso raro. Lo que importa suele ser minoritario.

Por eso este capítulo no va de coleccionar métricas. Va de aprender a hacer una pregunta mejor:

> ¿Qué error puedo tolerar, cuál no, y qué decisión cambia cuando miro la matriz?

Soporte, prevalencia y clase positiva

Antes de calcular nada hay que fijar tres piezas que parecen pequeñas y no lo son: **qué clase llamas positiva**, cuántos casos tiene cada clase y cuál es la frecuencia base del fenómeno que buscas. En documentación de métricas, `support` suele significar cuántas muestras reales hay de cada clase. Si tienes 1.000 tickets y solo 30 urgentes, el soporte de la clase positiva es 30. Esa cifra condiciona toda la lectura.

La **prevalencia** o frecuencia base es la proporción habitual de positivos en el conjunto que evalúas o en producción. En un clasificador de urgencias, si el 3 % de los tickets son urgentes, un sistema que marque el 40 % como urgente quizá tenga recall alto, pero puede ser inviable para soporte. En detección de fraude, si el positivo real es rarísimo, una tasa pequeña de falsos positivos puede generar miles de revisiones. En moderación, si cambian las campañas de abuso, la prevalencia puede moverse de una semana a otra y dejar obsoleto un umbral que ayer parecía sensato.

También hay que decidir qué significa “positivo”. Parece obvio, pero no siempre lo es. En un filtro de seguridad, positivo puede ser “riesgoso”; en un sistema médico, positivo puede ser “sospecha”; en soporte, positivo puede ser “urgente”; en recuperación de documentos, positivo puede ser “relevante”. Cambiar esa convención cambia la matriz, cambia precision y recall, y cambia la conversación con el equipo. Si no lo escribes, dos personas pueden mirar los mismos resultados y discutir porque están imaginando positivos distintos.

PIEZA	PREGUNTA CONCRETA	ERROR TÍPICO
Clase positiva	¿Qué evento quiero detectar o activar?	Llamar positivo a lo cómodo, no a lo importante.
Soporte	¿Cuántos casos reales tengo de cada clase?	Concluir demasiado con 8 positivos.
Prevalencia	¿Con qué frecuencia aparece el positivo en producción?	Evaluar con una muestra artificial y esperar el mismo comportamiento en tráfico real.
Unidad	¿Evalúo ticket, documento, respuesta, sesión o release?	Mezclar casos que no deberían compartir métrica.

Esta sección parece previa a las métricas, pero en realidad es parte de la métrica. Una precision del 80 % no significa lo mismo si has evaluado 10 positivos que si has evaluado 10.000. Un recall del 95 % no significa lo mismo si el 5 % que pierdes son tickets de baja prioridad que si son incidentes de producción. La matriz de confusión empieza antes de dibujar la tabla.

Qué sí es una matriz de confusión

Una matriz de confusión cruza dos cosas:

1. La realidad revisada.
2. La decisión del sistema.

Para clasificación binaria:

	PREDICE POSITIVO	PREDICE NEGATIVO
Real positivo	TP	FN
Real negativo	FP	TN

En un sistema de tickets urgentes:

SÍMBOLO	SIGNIFICADO	EJEMPLO
TP	Verdadero positivo.	Ticket urgente marcado como urgente.
FP	Falso positivo.	Ticket normal marcado como urgente.
FN	Falso negativo.	Ticket urgente marcado como normal.
TN	Verdadero negativo.	Ticket normal marcado como normal.

La matriz obliga a hacer una pregunta adulta: **qué error duele más**. En spam quizá un falso positivo duele mucho porque pierdes un correo importante. En incidentes de producción quizá duele más un falso negativo porque se te escapa una caída. En moderación, salud, pagos o educación, la respuesta depende del contexto, no de la métrica favorita del equipo.

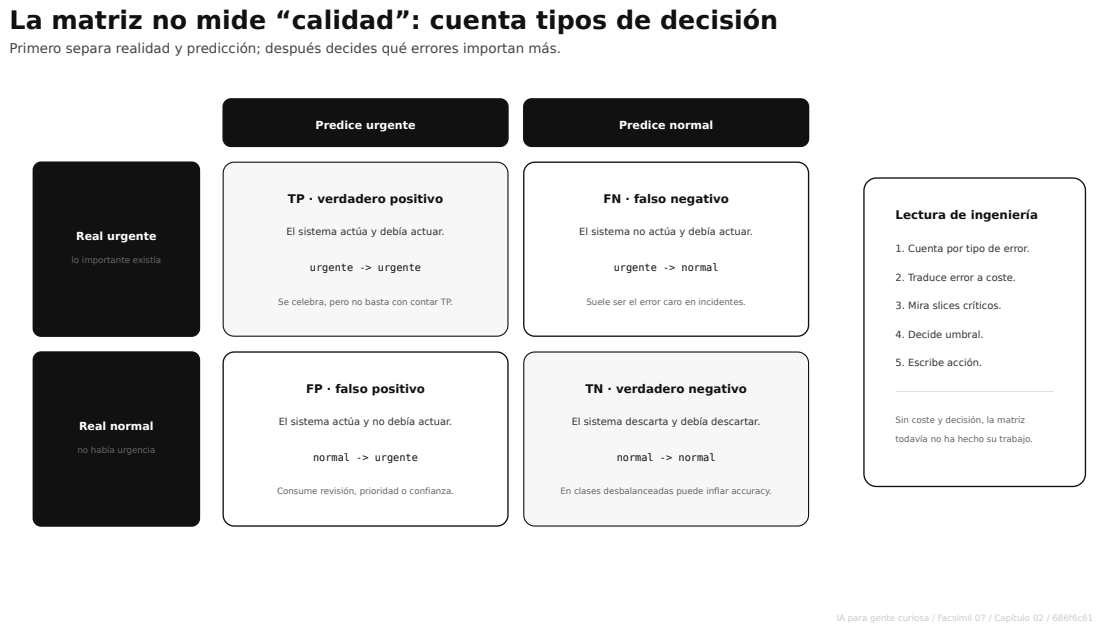
Para leerla con algo de oficio, no mires primero el número grande. Mira la diagonal y luego mira los errores. La diagonal, TP y TN, son aciertos. Los otros dos cuadrantes, FP y FN, son decisiones incorrectas. Pero el peso técnico no está repartido por igual. Si un FN deja pasar

una incidencia grave, ese cuadrante puede mandar más que todos los TN juntos. Si un FP bloquea una transferencia legítima o manda a revisión a demasiadas personas, ese otro cuadrante puede convertirse en coste operativo.

Con números, una matriz podría leerse así:

	PREDICE URGENTE	PREDICE NORMAL
Real urgente	18	4
Real normal	7	71

Una lectura floja diría: “hay 89 aciertos sobre 100”. Una lectura de ingeniería diría algo más parecido a esto: “detectamos 18 de 22 urgentes, perdemos 4 urgentes, generamos 7 falsas urgencias y descartamos bien 71 normales. Ahora hay que saber si perder 4 urgentes es aceptable, cuánto cuesta revisar 7 falsos positivos y si esos 4 falsos negativos pertenecen a un slice crítico”. La segunda lectura es menos cómoda, pero es la que sirve.



> La matriz de confusión convierte una clasificación en una contabilidad de errores. A partir de ahí empieza la decisión de ingeniería.

Fecha de corte del estado del arte

Fecha de corte: 28 de mayo de 2026.

La parte estable de este capítulo viene de métricas clásicas de clasificación y evaluación de clasificadores. scikit-learn documenta métricas como accuracy, balanced accuracy, precision, recall, F-measure, ROC AUC, matriz de confusión y reportes de clasificación con APIs reproducibles.[1] La función `confusion_matrix` cuenta observaciones reales frente a predichas por clase.[2] `classification_report` resume precision, recall, F1 y soporte por clase.[3]

Fawcett presentó ROC como herramienta para visualizar el equilibrio entre tasa de verdaderos positivos y tasa de falsos positivos al mover el umbral.[4] Davis y Goadrich explicaron la relación entre ROC y precision-recall, y por qué ambas vistas no son intercambiables sin más.[5] Saito y Rehmsmeier mostraron que, en datasets desbalanceados, la curva precision-recall suele ser más informativa que ROC para evaluar clasificadores binarios.[6]

Para coste sensible, la idea no es inventar una ecuación decorativa: viene de aprendizaje con costes y teoría de decisión aplicada a clasificación. Elkan formuló los fundamentos de *cost-sensitive learning* y Domingos propuso MetaCost como método general para hacer clasificadores sensibles al coste.[7][8]

La parte que cambia por proyecto es el coste real del error, la capacidad de revisión, el umbral y la decisión que quieres automatizar. Ahí no hay tabla universal: hay que medir, discutir con negocio/operación y dejar evidencia.

Las métricas básicas, con símbolos claros

Partimos de esta identidad básica:

$$N = TP + FP + FN + TN$$

En palabras: el total de casos evaluados es la suma de los cuatro cuadrantes de la matriz.

SÍMBOLO	SIGNIFICADO	EJEMPLO
N	Número total de casos evaluados.	100 tickets.
TP	Positivos reales predichos como positivos.	18 urgentes detectados.
FP	Negativos reales predichos como positivos.	7 normales marcados urgentes.
FN	Positivos reales predichos como negativos.	4 urgentes perdidos.
TN	Negativos reales predichos como negativos.	71 normales bien clasificados.

La **accuracy** mide proporción total de aciertos:

$$accuracy = \frac{TP + TN}{N}$$

En palabras: cuenta todas las decisiones correctas y las divide por todos los casos.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$accuracy$	Aciertos totales sobre casos totales.	$(18 + 71)/100 = 0,89$.
$TP + TN$	Decisiones correctas.	89.
N	Total de casos.	100.

Accuracy es cómoda porque resume todo en una proporción fácil de comunicar. Por eso se usa tanto, y por eso también engaña tanto. Si las clases están equilibradas y FP y FN cuestan parecido, puede ser una primera lectura razonable. Si una clase domina el dataset, accuracy se deja arrastrar por esa clase. En el ejemplo de tickets, 71 TN pesan muchísimo; pueden hacer que el sistema parezca bueno aunque los 4 FN sean justo lo que más te importaba.

Una forma práctica de usar accuracy sin caer en la trampa es leerla siempre junto al soporte de cada clase. No digas “89 % de accuracy” sin decir también cuántos positivos reales había. En revisión técnica, una frase más honesta sería: “89 % de accuracy sobre 100 casos, con 22 urgentes reales y 4 urgentes perdidos”. Esa segunda frase ya no permite esconder el problema detrás del porcentaje.

La **precision** responde: de lo que marqué como positivo, ¿cuánto lo era de verdad?

$$precision = \frac{TP}{TP + FP}$$

En palabras: mide la pureza de las alarmas positivas.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$precision$	Proporción de positivos predichos que eran positivos reales.	$18/(18 + 7) = 0,72$.
TP	Positivos correctos.	18.
FP	Positivos que no lo eran.	7.

Precision es la métrica que mira el ruido de tus positivos predichos. Si el sistema dice “esto es urgente” 25 veces y 18 lo eran, precision es 0,72. En lenguaje operativo: de cada 100 casos que mando a la cola urgente, 72 deberían estar ahí y 28 son trabajo extra. Esto importa

mucho cuando cada positivo predicho dispara una acción: revisión humana, bloqueo, alerta, correo, llamada, escalado, retención de pago o intervención manual.

Precision no te dice cuántos positivos reales se te escaparon. Puedes tener precision perfecta si solo marcas como urgente los casos obvios y dejas pasar todos los difíciles. Por eso una precision alta puede ser buena o puede ser cobarde: depende de si también estás encontrando lo importante.

El **recall** responde: de los positivos reales, ¿cuántos encontré?

$$recall = \frac{TP}{TP + FN}$$

En palabras: mide cobertura de lo importante.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$recall$	Proporción de positivos reales detectados.	$18/(18+4) = 0,82$.
TP	Positivos encontrados.	18.
FN	Positivos perdidos.	4.

Recall es la métrica que mira cobertura. En un sistema de urgencias, incidentes, fraude, seguridad o salud, suele ser la primera métrica que pone nervioso al equipo porque pregunta: “de lo que de verdad importaba, ¿cuánto he encontrado?”. Si el recall es 0,82, encontraste 18 de 22 urgentes y perdiste 4. El número no dice todavía si 4 es tolerable; eso lo decide el coste, la severidad y el slice.

Recall tampoco es gratis. Puedes subirlo bajando el umbral y marcando más casos como positivos. Eso detectará más urgentes, pero probablemente aumentará falsos positivos y revisión. En un sistema real, perseguir recall máximo sin mirar coste puede convertir una automatización en una fábrica de interrupciones.

La **specificity** responde: de los negativos reales, ¿cuántos dejé como negativos?

$$specificity = \frac{TN}{TN + FP}$$

En palabras: mide cobertura de negativos reales, algo muy útil cuando quieres controlar falsas alarmas.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$specificity$	Proporción de negativos reales descartados correctamente.	$71/(71+7) = 0,91$.
TN	Negativos correctos.	71.
FP	Negativos marcados como positivos.	7.

Specificity suele recibir menos cariño que recall, pero en sistemas con muchas falsas alarmas es decisiva. Si el equipo se queja de que “el sistema avisa demasiado”, muchas veces está hablando de specificity sin nombrarla. Una specificity baja significa que muchos negativos reales se convierten en positivos predichos. En soporte, eso llena colas; en seguridad, produce fatiga de alertas; en producto, rompe confianza.

La tensión clásica aparece aquí: subir recall puede bajar specificity, y subir specificity puede bajar recall. No hay una métrica “buena” en abstracto. Hay una frontera de decisión que mueve trabajo entre positivos perdidos, falsas alarmas y revisión.

F1 resume precision y recall con media armónica:

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

En palabras: F1 cae si una de las dos piezas, precision o recall, cae mucho. Por eso se usa como resumen cuando quieres equilibrarlas.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$F1$	Equilibrio entre precision y recall.	$2 \cdot 0,72 \cdot 0,82/(0,72+0,82) = 0,77$.
$precision$	Pureza de positivos predichos.	0,72.
$recall$	Cobertura de positivos reales.	0,82.

F1 es útil porque penaliza desequilibrios fuertes entre precision y recall. Si una de las dos cae mucho, F1 cae. Eso evita que alguien enseñe solo la métrica que le favorece. Pero F1 sigue siendo un resumen estadístico, no una política. No sabe si un FN cuesta 12 veces más que un FP. No sabe si tienes personas suficientes para revisar la zona gris. No sabe si el slice de pagos pesa más que el de consultas generales.

Por eso F1 sirve bien como indicador de lectura comparativa, especialmente cuando quieres comparar variantes bajo un mismo protocolo. Sirve peor como criterio único de publicación. Si una release se aprueba solo porque F1 sube, sin mirar matriz, coste y slices, la evaluación está incompleta.

F-beta permite dar más peso a recall o a precision:

$$F_\beta = \frac{(1+\beta^2)\cdot precision \cdot recall}{\beta^2 \cdot precision + recall}$$

En palabras: si $\beta > 1$, recall pesa más; si $\beta < 1$, precision pesa más.

SÍMBOLO	SIGNIFICADO	EJEMPLO
F_β	F-score con peso ajustable.	F_2 prioriza recall.
β	Peso relativo de recall frente a precision.	$\beta = 2$.
$precision$	Pureza de positivos predichos.	0,72.
$recall$	Cobertura de positivos reales.	0,82.

F-beta es una forma más honesta de reconocer una preferencia. Si el problema castiga mucho perder positivos, puedes usar F_2 para dar más peso a recall. Si el problema castiga mucho molestar con falsas alarmas, puedes usar un beta menor que 1 para dar más peso a precision. La clave es no elegir beta porque “queda avanzado”, sino porque expresa una decisión de dominio.

Powers revisa precision, recall, F-measure y medidas relacionadas como informedness, markedness y correlación, útiles para no reducir la evaluación a una sola cifra sin contexto.[9]

Qué aporta y qué no aporta cada métrica

Una forma práctica de no perderse es separar cada métrica por pregunta, utilidad y límite. Esta tabla no sustituye las fórmulas; ayuda a decidir cuándo una fórmula te está contando algo útil y cuándo te está distrayendo.

MÉTRICA	PREGUNTA QUE RESPONDE	ÚTIL CUANDO	NO TE CUENTA
Accuracy	¿Qué proporción total acerté?	Clases equilibradas y errores parecidos.	Si una clase minoritaria crítica está fallando.
Precision	¿Cuánto ruido hay entre mis positivos predichos?	Cada positivo dispara trabajo, bloqueo o alerta.	Cuántos positivos reales se escaparon.
Recall	¿Cuánto de lo importante encontré?	Perder positivos es caro o peligroso.	Cuántas falsas alarmas generaste.
Specificity	¿Cuánto de lo normal descarté bien?	Te preocupa saturar colas o producir fatiga.	Si cubres bien la clase positiva.
F1	¿Cómo equilibran precision y recall?	Comparas variantes bajo el mismo protocolo.	Coste, capacidad, severidad y slices.
F-beta	¿Quiero inclinar el resumen hacia precision o recall?	Hay una preferencia explícita de dominio.	La justificación de esa preferencia.

Si tienes que elegir una sola frase para una revisión técnica, que sea esta: “la métrica que enseño responde a esta pregunta y no responde a estas otras”. Esa frase ahorra muchos malentendidos.

Accuracy, balanced accuracy y clases desbalanceadas

Cuando hay muchas más clases negativas que positivas, `accuracy` puede ser complaciente. Si detectas spam, fraude, tickets urgentes o documentos peligrosos, la clase que te importa puede ser pequeña. En esos casos, un sistema que no hace nada puede parecer bueno.

Una alternativa sencilla es balanced accuracy:

$$balanced\ accuracy = \frac{recall + specificity}{2}$$

En palabras: da el mismo peso a la capacidad de encontrar positivos y a la capacidad de descartar negativos.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$balanced\ accuracy$	Media entre cobertura positiva y negativa.	$(0,82 + 0,91)/2 = 0,865$.
$recall$	Tasa de positivos encontrados.	0,82.
$specificity$	Tasa de negativos bien descartados.	0,91.

Balanced accuracy evita que una clase mayoritaria tape la lectura de la minoritaria, pero sigue sin saber cuánto cuesta cada error. Por eso no cierra el problema: solo lo lee con algo más de justicia estadística.

En ingeniería conviene leer balanced accuracy como una alarma contra el autoengaño. Si accuracy es alta y balanced accuracy cae, probablemente estás beneficiándote de la clase mayoritaria. Si ambas son altas, todavía no has terminado: falta mirar coste, intervalos de confianza, slices y estabilidad temporal. Un buen capítulo de evaluación no termina en “sube la métrica”, sino en “esta métrica sube, bajo estas condiciones, con estos límites, y por eso puedo o no puedo tomar esta decisión”.

Coste sensible: cuando FP y FN no cuestan lo mismo

El paso de métrica a ingeniería empieza cuando escribes una matriz de costes. No porque los costes didácticos sean perfectos, sino porque obligan al equipo a verbalizar sus prioridades.

Un coste no tiene por qué ser dinero exacto. Puede representar minutos de revisión, puntos de riesgo, impacto en SLA, fricción de usuario, carga de soporte o severidad operacional. Lo importante es que sea explícito y discutible. Si el equipo dice “un falso negativo es grave”, todavía no hay política. Si dice “un falso negativo pesa seis veces más que un falso positivo y la revisión cuesta menos que ambos errores”, ya puedes comparar umbrales de una forma auditable.

Esta conversación suele ser incómoda porque saca a la luz desacuerdos reales. Producto quizá quiere automatizar más. Operaciones quizá quiere menos falsas alarmas. Riesgo quizá prefiere revisar cualquier caso dudoso. Datos quizá avisa de que la muestra es pequeña. La matriz de costes no elimina el desacuerdo, pero lo convierte en algo que se puede probar en una scorecard.

En clasificación sensible al coste, una forma estándar de expresar el riesgo empírico de un clasificador es:

$$R(h) = \frac{1}{n} \sum_{i=1}^{n} C(y_i, h(x_i))$$

En palabras: calculas el coste que produce el clasificador h en cada caso evaluado y promedias sobre la muestra. Esta idea pertenece al marco de aprendizaje sensible al coste, no a una ocurrencia del capítulo.[1011]

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\widehat{R}(h)$	Riesgo empírico sensible al coste del clasificador.	Coste medio por ticket evaluado.
n	Número de casos evaluados.	100 tickets.
x_i	Entrada del caso i.	Texto y metadatos de un ticket.
y_i	Etiqueta real revisada del caso i.	Urgente o normal.
$h(x_i)$	Predicción del clasificador.	Urgente o normal.
$C(y_i, h(x_i))$	Coste de predecir $h(x_i)$ cuando la realidad es y_i.	12 si pierdes un urgente; 2 si generas falsa urgencia.

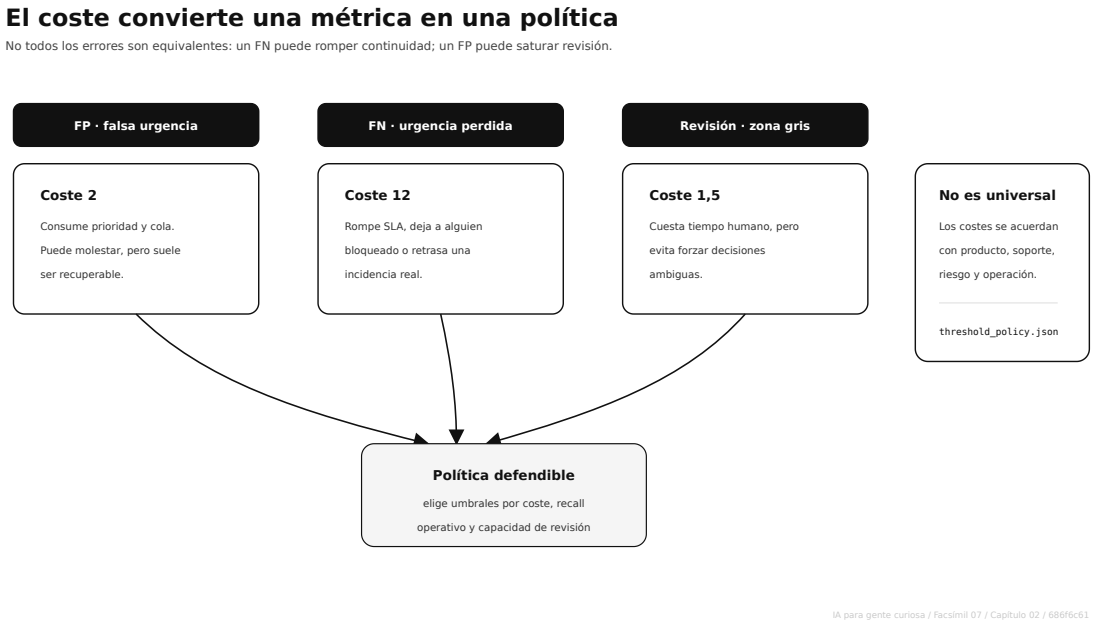
En el caso binario, si solo contamos falsos positivos y falsos negativos, esta lectura se traduce de forma práctica así:

ERROR	PREGUNTA DE NEGOCIO	COSTE DIDÁCTICO
Falso positivo	¿Qué pasa si trato un ticket normal como urgente?	2
Falso negativo	¿Qué pasa si trato un ticket urgente como normal?	12
Revisión	¿Qué cuesta que una persona mire la zona gris?	1,5

No voy a convertir la revisión humana en una fórmula nueva del facsímil. Es una política operativa: cuentas cuántos casos mandas a revisar, multiplicas por el coste acordado y comparas políticas. Lo importante es que esa decisión se pueda auditar.

En el cuaderno del facsímil, los costes son didácticos: falso positivo 2, falso negativo 12 y revisión 1,5. No significan euros reales. Significan relación de prioridades: perder un urgente pesa mucho más que crear una falsa urgencia, y revisar un caso ambiguo es más barato que equivocarse automáticamente. En un proyecto real, esos valores se deberían estimar con datos históricos y conversación de dominio. ¿Cuántos minutos cuesta revisar? ¿Cuánto cuesta incumplir un SLA? ¿Qué pasa si un falso positivo bloquea una acción legítima? ¿Qué coste reputacional o legal tiene cada error?

También conviene separar coste de severidad. Dos errores pueden tener el mismo tipo, pero no la misma gravedad. Un falso negativo en una consulta menor no pesa igual que un falso negativo en acceso durante una entrega. Por eso el dataset de práctica incluye `slice` y `severity`: no para adornar el JSON, sino para recordarte que una matriz global puede mezclar daños muy distintos.



> Coste sensible no significa poner números al azar: significa explicitar qué consecuencias acepta o no acepta tu sistema.

Umbrales: mover la frontera cambia el sistema

Un clasificador suele devolver un score. El umbral convierte ese score en una acción. Para no meter una fórmula innecesaria, pensemos en pseudocódigo:

```
si score >= umbral:
    decidir "urgente"
si no:
    decidir "normal"
```

Si subes el umbral, normalmente marcas menos casos como positivos. Eso puede subir precision, pero baja recall. Si bajas el umbral, detectas más positivos, pero generas más falsos positivos. No hay magia: estás moviendo trabajo entre errores, automatización y revisión.

El umbral no debería elegirse en el test final. Lo normal es usar un conjunto de validación para explorar umbrales y reservar un test final para estimar cómo se comporta la política ya elegida. Si tanteas umbrales mirando el test una y otra vez, conviertes el test en parte del entrenamiento de la decisión. El número final parecerá más sólido de lo que es.

También hay que evitar otro malentendido: un score alto no siempre es una probabilidad calibrada. Un score de 0,80 puede significar “este caso está muy arriba en el ranking”, pero no necesariamente “hay un 80 % de probabilidad real”. Para elegir umbrales puede bastar con que el score ordene bien; para interpretar riesgo como probabilidad necesitas calibración, que veremos en el capítulo 05. Mezclar esas dos cosas es una fuente clásica de errores en sistemas de IA.

En sistemas reales, muchas veces conviene usar dos umbrales:

```
si score <= umbral_bajo:
    decidir "normal"
si score >= umbral_alto:
    decidir "urgente"
si no:
    enviar a revisión
```

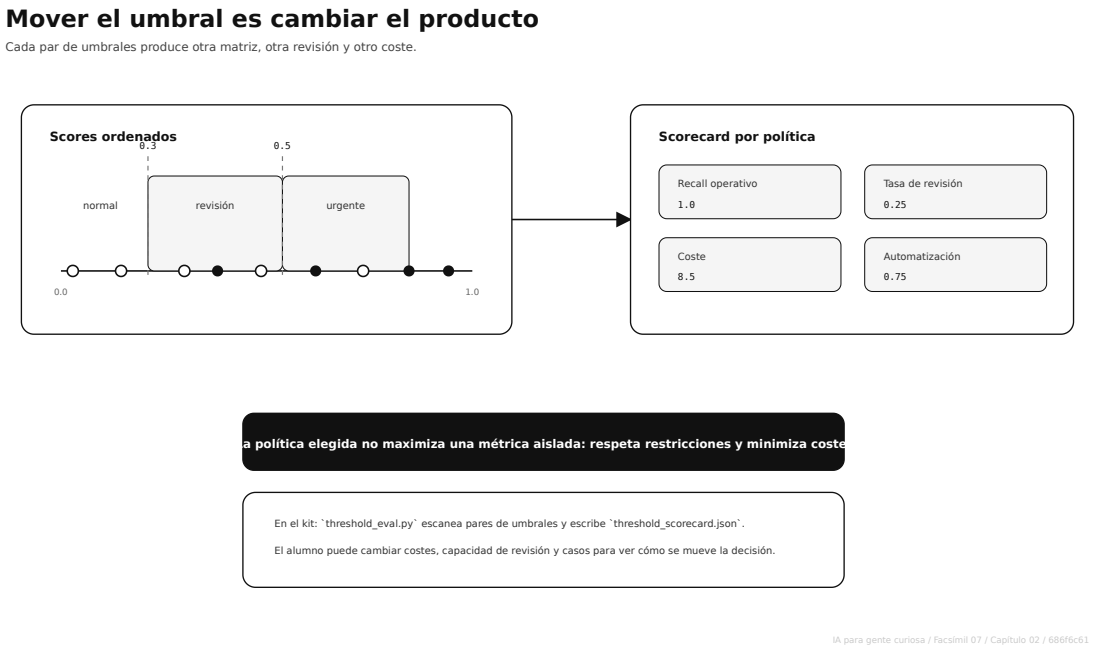
La zona gris no es una derrota. Es una herramienta de ingeniería cuando el coste de equivocarte supera el coste de revisar. El problema es que la revisión también tiene capacidad finita. Si todo cae en la zona gris, no has automatizado: has creado una cola.

La zona gris se diseña con dos restricciones a la vez. La primera es de calidad: quiero que los casos claramente normales salgan como normales y los claramente urgentes salgan como urgentes. La segunda es de capacidad: solo puedo revisar una parte del volumen. Si el equipo puede revisar un 35 % de los casos, una política que manda el 60 % a revisión no es “prudente”; es inoperable.

En el runner de la práctica, cada par de umbrales produce cuatro lecturas: coste, recall operativo, tasa de revisión y tasa de automatización. Esa combinación es más útil que discutir un único número. Si una política tiene gran recall pero revisa demasiado, no escala. Si automatiza mucho pero pierde positivos, no es segura. Si minimiza coste en la muestra pero falla en el slice de acceso, no debería publicarse sin más.

DECISIÓN DE UMBRAL	QUÉ SUELE PASAR	RIESGO QUE VIGILAS
Umbral alto muy exigente	Pocos positivos automáticos, precision alta.	Pierdes positivos reales.
Umbral bajo muy permisivo	Muchos positivos automáticos, recall alto.	Generas demasiadas falsas alarmas.
Dos umbrales con zona gris	Automatizas extremos y revisas ambiguos.	Saturas revisión si la zona gris es grande.
Umbral por slice	Ajustas decisión por segmento.	Puedes introducir reglas difíciles de gobernar.

Esta última fila merece cuidado. A veces tiene sentido usar umbrales distintos por canal, producto o idioma, pero eso aumenta complejidad y exige trazabilidad. Si una persona recibe una decisión distinta por pertenecer a un slice, necesitas justificarlo técnicamente y vigilar impacto. No es solo una mejora de métrica; es una política.



El umbral no se elige por costumbre: se barre, se mide y se defiende con una política de coste y revisión.

ROC, PR y qué mirar primero

ROC compara tasa de verdaderos positivos contra tasa de falsos positivos mientras movemos el umbral:

$$TPR = \frac{TP}{TP + FN}$$

En palabras: TPR es lo mismo que recall; mide qué proporción de positivos reales detectas.

$$FPR = \frac{FP}{FP + TN}$$

En palabras: FPR mide qué proporción de negativos reales conviertes en falsas alarmas.

SÍMBOLO	SIGNIFICADO	EJEMPLO
TPR	True Positive Rate; equivalente a recall.	0,82.
FPR	False Positive Rate.	$7/(7+71) = 0,09$.

Precision-recall mira precision contra recall. En clases muy desbalanceadas, PR suele enseñar mejor si los positivos predichos están llenos de ruido. Por eso Saito y Rehmsmeier recomiendan mirar precision-recall en datasets desbalanceados.[1][2]

La diferencia práctica es importante. Una curva ROC puede verse razonablemente buena aunque el positivo sea rarísimo, porque el número de negativos reales es enorme y la tasa de falsos positivos puede parecer pequeña. Pero una tasa pequeña aplicada a millones de negativos puede producir miles de falsas alarmas. La curva precision-recall te obliga a mirar la pureza de los positivos que realmente vas a tocar.

En un sistema de tickets, ROC responde bien a la pregunta “¿cómo se mueve la tasa de detección frente a la tasa de falsas alarmas?”. PR responde mejor a “cuando digo urgente, ¿cuántas veces estoy metiendo ruido en la cola?”. Si el equipo humano trabaja sobre los positivos predichos, PR suele estar más cerca del dolor diario.

Lectura práctica:

SITUACIÓN	MÉTRICA O CURVA QUE MIRARÍA PRIMERO
Clases equilibradas y costes parecidos.	Accuracy, matriz, F1 y ROC.
Positivo raro y caro de perder.	Recall, PR curve, F-beta con $\beta > 1$, coste de FN.
Positivo marcado genera trabajo humano.	Precision, coste de FP, tasa de revisión.
Decisión con score usado como probabilidad.	Calibración, Brier/log loss y umbrales; lo veremos en el capítulo 05.
Sistema con slices críticos.	Métricas por slice antes que media global.

En el día a día

Supón que un equipo quiere automatizar priorización de tickets. No necesita solo “un modelo que clasifique”. Necesita una política que pueda defender ante soporte, producto y operaciones.

En una demo, el modelo puede parecer útil porque marca como urgente los casos obvios: “no puedo entrar”, “servicio bloqueado”, “pago duplicado”. Pero la decisión real vive en los casos intermedios: mensajes ambiguos, clientes con distinto contexto, incidencias que suenan normales pero ocurren antes de un cierre, o tickets de acceso que parecen rutina hasta que bloquean una entrega.

Ahí las métricas se vuelven operativas:

PREGUNTA	DECISIÓN CONCRETA
¿Qué clase positiva importa?	`urgente`.
¿Qué coste tiene perder un urgente?	Alto: afecta tiempos de respuesta y continuidad.
¿Qué coste tiene marcar normal como urgente?	Medio: consume revisión y altera prioridad.
¿Cuántos casos puede revisar el equipo?	Por ejemplo, 35 % del volumen de la muestra.
¿Qué umbral acepta producto?	El que minimice coste respetando capacidad y recall mínimo.
¿Qué pasa si el volumen cambia?	Se monitoriza base rate, matriz por día y cola de revisión.

El capítulo 01 nos enseñó a crear el expediente de evaluación. Este capítulo añade el motor numérico para defender ese expediente: matriz, métricas, coste, umbral y decisión.

En una reunión real, una buena lectura no sería “F1 sale 0,80”. Sería algo más parecido a esto: “con umbral bajo 0,3 y alto 0,5 automatizamos el 75 % de la muestra, revisamos el 25 %, no perdemos urgentes en esta validación, generamos dos falsas urgencias en consultas y mantenemos el coste en 8,5 unidades didácticas. El slice de acceso queda cubierto por automatización o revisión; el slice de consulta concentra las falsas urgencias. Si el volumen de consultas crece, tendremos que revisar coste y umbral”.

Esa frase es más larga, pero es muchísimo más útil. Da decisión, evidencia, límites y siguiente vigilancia. Ese es el tipo de densidad que buscamos en una evaluación: no más jerga, sino más información accionable por frase.

Por qué debería importarte

Porque estas métricas son el punto donde un sistema de IA deja de ser una respuesta bonita y se convierte en una decisión que afecta trabajo real.

Si eliges mal la métrica, puedes publicar un sistema que parece mejorar y aun así empeora lo importante. Si eliges mal el umbral, puedes saturar una cola humana o dejar pasar casos críticos. Si no miras slices, puedes aprobar globalmente una release que falla en pagos, acceso, idioma, región o tipo de usuario. Y si no escribes el coste, cada reunión se convierte en una discusión de gustos.

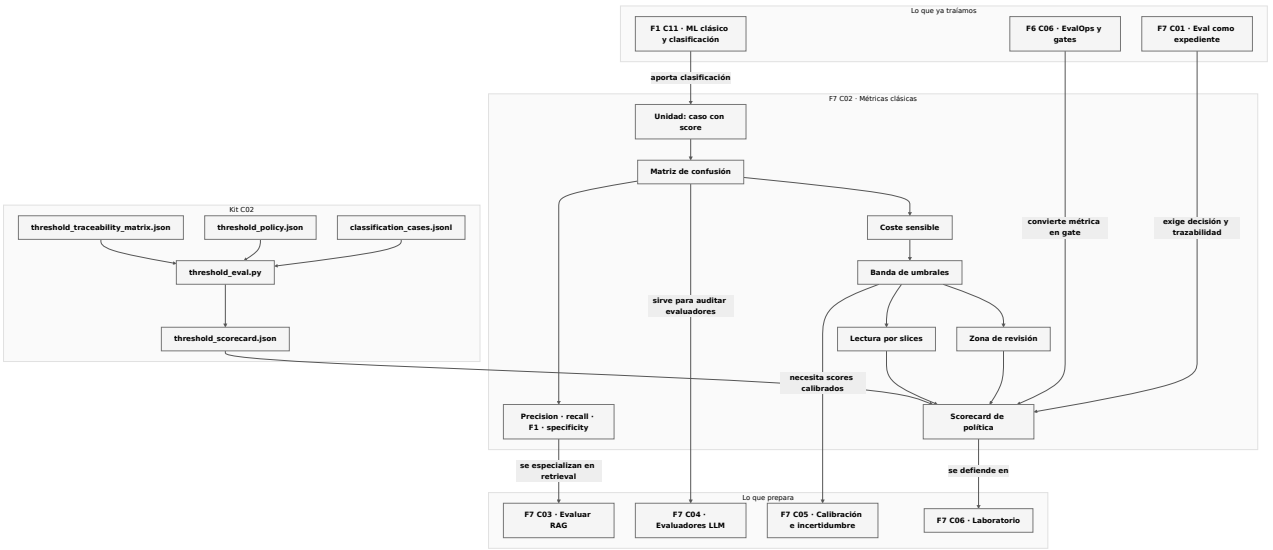
La matriz de confusión te da lenguaje común. Precision y recall te dan lectura. El coste sensible te obliga a priorizar. La zona gris te permite no fingir certeza cuando el sistema no la tiene. Esa combinación es ingeniería básica para evaluar clasificadores, routers, filtros, moderadores, detectores de riesgo, evaluadores de respuestas y gates de release.

También importa porque muchos sistemas modernos disfrazan decisiones binarias bajo interfaces más amables. Un router decide si una consulta va a RAG o a respuesta directa. Un guardrail decide si deja pasar una salida. Un evaluador decide si una respuesta cumple una rúbrica. Un detector decide si manda una conversación a revisión. Aunque por fuera hables de agentes, LLMs o pipelines multimodales, por dentro siguen apareciendo decisiones de clasificación. Si no sabes leer matriz, coste y umbral, acabarás aceptando decisiones automáticas sin entender qué errores compras.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Celebrar accuracy sin mirar la clase positiva.	Si el positivo es raro, accuracy puede ser alta aunque el sistema no encuentre lo importante.	Empezar por matriz de confusión, recall y coste del FN.
Creer que F1 elige por mí.	F1 no conoce capacidad de revisión, coste operativo ni daño de cada error.	Usar F1 como resumen, no como jefe.
Mover el umbral en el test final.	Si eliges umbral mirando el test final, contaminas la estimación.	Separar validación para elegir umbral y test para estimar rendimiento final.
No contar la revisión como salida.	Enviar a revisión consume tiempo y cambia el coste.	Medir `review_rate`, `automation_rate` y coste de revisión.
No mirar slices.	Una política puede funcionar globalmente y fallar en un segmento pequeño.	Reportar matriz y métricas por slice antes de publicar.
Tratar un score como probabilidad calibrada.	Muchos scores ordenan casos, pero no expresan probabilidad real.	Dejar la calibración para el capítulo 05 y no prometer más de lo medido.

Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN BREVE
Matriz de confusión	Tabla que cruza realidad y predicción por tipos de acierto y error.
TP	Positivo real que el sistema marca como positivo.
FP	Negativo real que el sistema marca como positivo.
FN	Positivo real que el sistema marca como negativo.
TN	Negativo real que el sistema marca como negativo.
Accuracy	Aciertos totales entre casos totales.
Precision	Proporción de positivos predichos que eran positivos reales.
Recall	Proporción de positivos reales que el sistema detecta.
Specificity	Proporción de negativos reales que el sistema deja como negativos.
F1	Media armónica entre precision y recall.
F-beta	Variante de F1 que da más peso a recall o precision.
Balanced accuracy	Media entre recall y specificity.
Coste sensible	Evaluación que pondera errores distintos con consecuencias distintas.
Umbral	Corte que convierte score en acción.
Zona de revisión	Rango de score que se manda a revisión.
Recall operativo	Positivos cubiertos por automatización positiva o revisión.
Tasa de revisión	Proporción de casos enviados a revisión.
Tasa de automatización	Proporción de casos decididos sin revisión.
Slice	Subgrupo donde miramos métricas separadas.
Soporte	Número de casos reales por clase o por slice.
Prevalencia	Frecuencia base de la clase positiva en evaluación o producción.

TÉRMINO	DEFINICIÓN BREVE
Matriz de costes	Tabla que asigna consecuencias a errores y decisiones.

Antes de pasar página

Antes de avanzar al siguiente capítulo, deberías poder responder:

1. ¿Por qué accuracy puede ser engañosa en clases desbalanceadas?
2. ¿Qué diferencia hay entre FP y FN en un sistema de tickets urgentes?
3. ¿Qué pregunta responde precision?
4. ¿Qué pregunta responde recall?
5. ¿Por qué F1 no sustituye una matriz de costes?
6. ¿Qué cambia cuando usamos dos umbrales y zona de revisión?
7. ¿Por qué PR curve suele ser más útil que ROC cuando el positivo es raro?
8. ¿Qué significa elegir umbral por coste sensible?
9. ¿Qué archivos produce la práctica del capítulo?
10. ¿Qué entregarías para defender una política de automatización?

En resumen

IDEA	QUÉ TE LLEVAS
La matriz de confusión es la contabilidad básica de la clasificación.	Sin ella, no sabes qué tipo de error estás cometiendo ni qué cuadrante debería preocupar al equipo.
Soporte y prevalencia condicionan todo.	Un 90 % de accuracy no significa lo mismo con clases equilibradas que con un positivo raro.
Precision y recall responden preguntas distintas.	Precision mide ruido en positivos predichos; recall mide positivos reales encontrados. Ambas necesitan contexto.
F1 resume, pero no decide.	Si FP y FN cuestan distinto, necesitas coste sensible, restricciones y política operativa.
El umbral es una decisión de producto e ingeniería.	Moverlo cambia automatización, revisión, errores y coste; no debería elegirse tanteando el test final.
La zona gris es útil si cabe en operación.	Revisar casos ambiguos puede ser mejor que forzar automatización, pero una cola imposible no es una solución.
Los slices importan.	Una métrica global puede esconder el fallo que más te importa, especialmente en segmentos pequeños o críticos.
El cuaderno no es un adorno.	Practicas con datos, política, runner, tests y salida para tomar una decisión real y defenderla.

Para saber más

Davis, J. y Goadrich, M. (2006). The relationship between precision-recall and ROC curves. *Proceedings of the 23rd International Conference on Machine Learning*, 233-240. https://doi.org/10.1145/1143844.1143874

Domingos, P. (1999). MetaCost: A general method for making classifiers cost-sensitive. *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 155-164. https://doi.org/10.1145/312129.312220

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1), 1-26. https://doi.org/10.1214/aos/1176344552

Elkan, C. (2001). The foundations of cost-sensitive learning. *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 973-978.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), 861-874. https://doi.org/10.1016/j.patrec.2005.10.010

Hand, D. J. (2009). Measuring classifier performance: A coherent alternative to the area under the ROC curve. *Machine Learning*, 77(1), 103-123. https://doi.org/10.1007/s10994-009-5119-5

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2), 153-157. https://doi.org/10.1007/BF02295996

Powers, D. M. W. (2011). Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation. *Journal of Machine Learning Technologies*, 2(1), 37-63.

Saito, T. y Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3), e0118432. https://doi.org/10.1371/journal.pone.0118432

scikit-learn. (2026). *Classification Metrics*. https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics

scikit-learn. (2026). *classification_report*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

scikit-learn. (2026). *confusion_matrix*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

Notas

1. scikit-learn. (2026). *Classification Metrics*. https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics. Consultado el 28 de mayo de 2026. ↩

2. scikit-learn. (2026). *confusion_matrix*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. Consultado el 28 de mayo de 2026. ↩

3. scikit-learn. (2026). *classification_report*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html. Consultado el 28 de mayo de 2026. ↩

4. Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), 861-874. https://doi.org/10.1016/j.patrec.2005.10.010 ↩

5. Davis, J. y Goadrich, M. (2006). The relationship between precision-recall and ROC curves. *Proceedings of the 23rd International Conference on Machine Learning*, 233-240. https://doi.org/10.1145/1143844.1143874 ↩

6. Saito, T. y Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3), e0118432. https://doi.org/10.1371/journal.pone.0118432 ↩

7. Elkan, C. (2001). The foundations of cost-sensitive learning. *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 973-978. ↩

8. Domingos, P. (1999). MetaCost: A general method for making classifiers cost-sensitive. *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 155-164. https://doi.org/10.1145/312129.312220 ↩

9. Powers, D. M. W. (2011). Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation. *Journal of Machine Learning Technologies*, 2(1), 37-63. ↩

.0. Elkan, 2001. ↩

.1. Domingos, 1999. ↩

.2. Saito y Rehmsmeier, 2015. ↩

Cuadernos para practicar

Escanea el código con el móvil para abrir el cuaderno en Google Colab y ejecutarlo sin instalar nada.

El coste del error: dónde poner el umbral

https://colab.research.google.com/github/686f6c61/iaparagentecuriosa-notebooks/blob/main/Facs%C3%ADmil%2007/02-coste-error-umbral.ipynb

Capítulo 03: Evaluar RAG: retrieval, groundedness y abstención

Entrando en el tema

En el facsímil 4, capítulo 09 montamos un RAG básico: documentos, chunks, embeddings, búsqueda, contexto y respuesta. En el facsímil 4, capítulo 10 vimos una primera idea de evaluación: no mirar solo si la respuesta "suena bien". Ahora toca hacerlo con mentalidad de ingeniería.

Un RAG no es un modelo que responde con magia documental. Es una cadena. Entra una pregunta, se busca evidencia, se ordenan fragmentos, se empaqueta contexto, el modelo genera una respuesta, se citan fuentes y se decide si había base suficiente para responder. Si una pieza falla, la respuesta final puede seguir pareciendo bonita. Ese es el peligro.

La idea central del capítulo es esta: **evaluar RAG es seguir la cadena de custodia de la evidencia**. No basta con preguntar "¿la respuesta está bien?". Hay que poder decir dónde se rompió el sistema: corpus, chunking, retrieval, ranking, contexto, generación, citas, abstención, coste o latencia.

Al terminar deberías poder hacer algo concreto: coger trazas de un RAG, calcular métricas de recuperación, revisar si las afirmaciones están sostenidas, medir si las citas cubren lo importante, comprobar si el sistema se abstiene cuando toca y producir una decisión de release. Esto conecta directamente con el capítulo 01: una eval no existe para decorar una demo, existe para permitir una decisión.

Fecha de corte del estado del arte

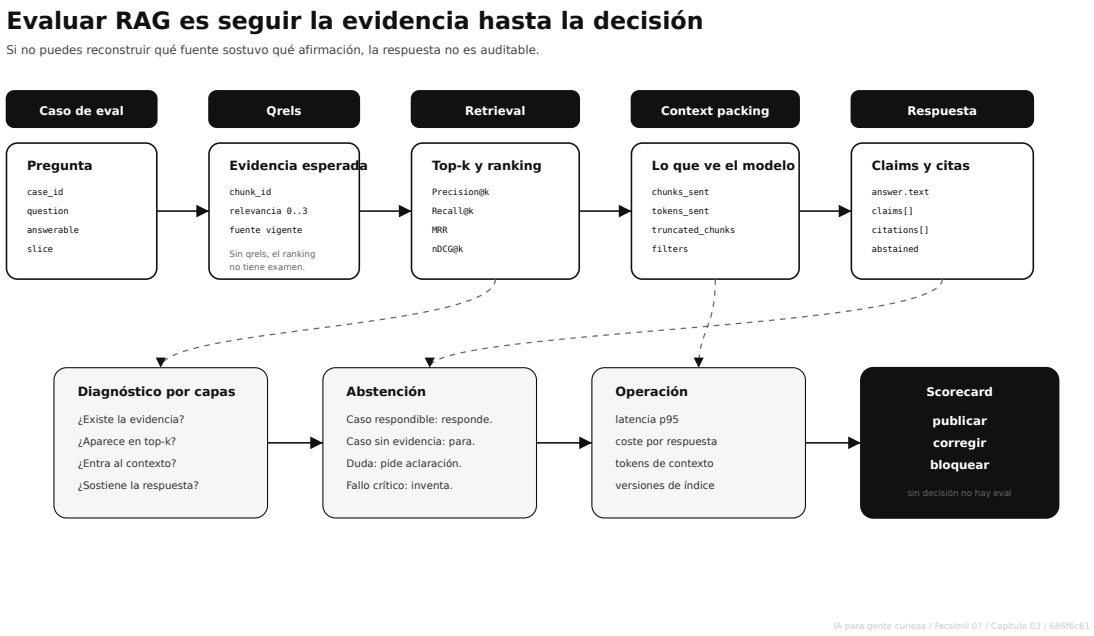
Fecha de corte: 31 de mayo de 2026.

RAG aparece formalizado en el trabajo de Lewis y otros como una forma de combinar recuperación y generación para tareas intensivas en conocimiento.[1] La evaluación de retrieval se apoya en una tradición anterior de recuperación de información, rankings y relevancia graduada. BEIR y MTEB son referencias modernas para comparar modelos y tareas de retrieval/embeddings en dominios variados.[2]

En aplicaciones RAG actuales, herramientas como RAGAS, TruLens, LangSmith, LlamaIndex y Phoenix separan señales como relevancia del contexto, faithfulness/groundedness, respuesta, citas y trazas.[3]

Conviene distinguir dos cosas. Las métricas de ranking como Precision@k, Recall@k, MRR o nDCG vienen de recuperación de información y tienen una formulación académica estable. En cambio, “groundedness”, “faithfulness”, “citation quality” o “abstention quality” suelen ser definiciones operativas de una herramienta o de un equipo. Son útiles, pero no deben presentarse como si fueran una ley universal. En este capítulo las uso como señales de ingeniería: se definen, se auditan y se conectan con una decisión.

Anatomía de una evaluación RAG



> La evaluación RAG útil no empieza en la respuesta final. Empieza en el caso, los qrels y la evidencia que debería aparecer.

El diagrama resume la disciplina que vamos a seguir. Primero se define el caso de evaluación. Después se declara qué evidencia sería válida. Luego se mira si el retriever la encuentra, si el ranking la coloca arriba, si el contexto que entra al modelo la conserva, si la respuesta la usa y si las citas permiten auditar lo dicho.

Esto tiene una consecuencia muy práctica: cuando una respuesta falla, no dices “el RAG es malo”. Dices algo más útil: “el chunk correcto no existía”, “existía pero no fue recuperado”, “fue recuperado pero quedó fuera del contexto”, “entró al contexto pero el modelo añadió una afirmación sin soporte”, o “no había evidencia y aun así respondió”.

Qué se mide antes de llamar al modelo

La primera tentación es evaluar la respuesta final. Al usuario, al fin y al cabo, le importa la respuesta. Pero para ingeniería es demasiado tarde. Si la evidencia no llega al contexto, el generador no puede resolverlo de forma fiable. Por eso el primer bloque de evaluación se hace solo con ranking.

Definimos estos símbolos:

SÍMBOLO	SIGNIFICADO	EJEMPLO
q	Pregunta evaluada.	"¿Puedo ampliar matrícula si tengo un pago pendiente?"
Q	Conjunto de preguntas de evaluación.	Dataset de 200 preguntas reales o revisadas.
G_q	Chunks relevantes esperados para q.	`{normativa#plazos, normativa#pagos}`
$R_k(q)$	Lista de los k primeros chunks recuperados.	Top 3 devuelto por el retriever.
rel_i	Relevancia graduada del resultado en la posición i.	0, 1, 2 o 3.
$rank_q$	Posición de la primera evidencia relevante.	1 si aparece arriba del todo.

Con eso podemos usar métricas académicas de recuperación de información:

$$\text{Precision@k}(q) = \frac{|R_k(q) \cap G_q|}{k}$$

En palabras: de los k documentos que has traído, cuántos eran realmente evidencia esperada. Si top-3 trae un chunk bueno y dos irrelevantes, Precision@3 es $1/3$.

$$\text{Recall@k}(q) = \frac{|R_k(q) \cap G_q|}{|G_q|}$$

En palabras: de toda la evidencia que hacía falta, cuánta apareció entre los k primeros. Si la pregunta necesitaba dos chunks y solo aparece uno, Recall@3 es $1/2$. Esta métrica es muy importante en preguntas que necesitan varias piezas.

$$\text{Hit@k}(q) = \begin{cases} 1 & \text{si } R_k(q) \cap G_q \neq \emptyset \\ 0 & \text{si } R_k(q) \cap G_q = \emptyset \end{cases}$$

En palabras: basta con que aparezca una evidencia válida. Es una métrica rápida, pero puede engañar: una sola fuente útil no basta si la respuesta necesitaba dos.

$$\mathrm{RR}(q) = \frac{1}{rank_q}$$

$$\mathrm{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \mathrm{RR}(q)$$

En palabras: premia que la primera evidencia útil aparezca pronto. Si la primera evidencia buena aparece en posición 1, RR vale 1. Si aparece en posición 4, vale 0,25.

Cuando la relevancia no es solo “sí/no”, usamos ganancia acumulada. Järvelin y Kekäläinen formalizaron DCG y nDCG para rankings con relevancia graduada.[4]

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

$$\mathrm{nDCG@k}(q) = \frac{\mathrm{DCG@k}(q)}{\mathrm{IDCG@k}(q)}$$

En palabras: un chunk de relevancia 3 en la posición 1 pesa mucho más que un chunk de relevancia 1 en la posición 5. nDCG divide por el ranking ideal para que el resultado quede normalizado.

MÉTRICA	QUÉ DETECTA	QUÉ NO DETECTA
Precision@k	Ruido en los primeros resultados.	Si falta una segunda fuente necesaria.
Recall@k	Cobertura de evidencia esperada.	Si la evidencia aparece demasiado abajo para tu top-k real.
Hit@k	Si aparece al menos una fuente útil.	Si la respuesta necesita varias fuentes.
MRR	Si lo primero útil aparece pronto.	Si el resto del contexto está lleno de ruido.
nDCG@k	Orden con relevancia graduada.	Si la respuesta final cita bien.

Estas fórmulas no son decoración: sirven para separar retrieval de generación. Si Recall@k ya es bajo antes de llamar al LLM, no tiene sentido gastar diez horas ajustando el prompt. Primero debes mirar corpus, chunking, filtros, embeddings, búsqueda híbrida, reranking o reescritura de consulta.

Qrels: el examen del retrieval

Un qrel es un juicio de relevancia. En lenguaje sencillo: para una pregunta concreta, alguien declara qué chunks deberían aparecer y con qué fuerza. No es glamuroso, pero sostiene toda la evaluación.

Un qrel mínimo debería guardar:

CAMPO	QUÉ GUARDA	POR QUÉ IMPORTA
`case_id`	Identificador estable.	Permite comparar versiones sin perder trazabilidad.
`question`	Pregunta evaluada.	Debe parecerse a uso real, no a una frase perfecta de laboratorio.
`answerable`	Si el corpus permite responder.	Activa métricas de abstención.
`gold_chunks`	Chunks esperados y relevancia.	Permite Recall@k, MRR y nDCG@k.
`slice`	Segmento de análisis.	Evita que la media esconda fallos por fuente, idioma o producto.
`why_it_exists`	Motivo de incluir el caso.	Evita datasets decorativos.

La relevancia graduada suele bastar con cuatro niveles:

VALOR	LECTURA PRÁCTICA	EJEMPLO
0	No aporta evidencia.	Documento parecido, pero de otro curso.
1	Relacionado, insuficiente.	FAQ general sin la condición clave.
2	Útil para parte de la respuesta.	Fragmento con el plazo, pero no con excepciones.
3	Evidencia central.	Fragmento que sostiene la afirmación principal.

En un proyecto serio, los qrels se revisan como código: diff, propietario, fecha, fuente, motivo y vínculo al documento. Si cambian los qrels, puede cambiar la métrica aunque el sistema no haya cambiado. Por eso el dataset de evaluación también debe versionarse.

Groundedness, citas y abstención

Groundedness no significa que una respuesta “suene bien”. Significa que las afirmaciones importantes de la respuesta están sostenidas por evidencia recuperada. RAGAS habla de faithfulness y otras métricas de contexto/respuesta; TruLens resume su evaluación RAG con una tríada: relevancia de contexto, groundedness y relevancia de respuesta. La idea común es separar “me gusta la respuesta” de “puedo defender cada afirmación con el contexto”.

Para trabajar con groundedness de forma auditable, no empieces por una nota global. Empieza por claims:

PASO	QUÉ HACES	EVIDENCIA QUE PRODUCES
1	Separas la respuesta en afirmaciones verificables.	`claims[]`
2	Para cada afirmación, apuntas qué chunk la sostiene.	`supporting_chunks[]`
3	Marcas afirmaciones sin soporte.	diagnóstico por caso
4	Revisas si las citas cubren lo importante.	`citation_precision`, `citation_recall` en la práctica
5	Bloqueas si hay una respuesta sin evidencia en un caso no respondible.	fallo crítico

Aquí no voy a disfrazar una definición operativa de fórmula universal. En el cuaderno del facsímil, `groundedness` se calcula como proporción de claims con soporte explícito. Es una decisión práctica para enseñar el mecanismo. En una herramienta real, ese cálculo puede hacerse con reglas, revisión humana, evaluadores LLM, embeddings o combinaciones. Lo importante es que la traza permita auditar por qué una afirmación se considera sostenida.

Las citas también tienen dos lecturas distintas:

SEÑAL	PREGUNTA QUE RESPONDE	ERROR TÍPICO
Precisión de cita	De las citas usadas, ¿cuántas apuntan a evidencia válida?	Citar un documento real que no sostiene la frase.
Cobertura de cita	De la evidencia esperada, ¿cuánta aparece citada?	Responder con una fuente parcial y olvidar la condición importante.

La abstención es la tercera pata. Un RAG serio no solo responde. También sabe decir “con la evidencia disponible no puedo afirmarlo”. Esto no es un gesto de humildad estética: es una política de producto. En un asistente de normativa, soporte técnico, salud, finanzas, compliance o documentación interna, responder sin evidencia puede ser peor que no responder.

CASO	QUÉ DEBERÍA PASAR	QUÉ MIDES
Pregunta respondible y evidencia recuperada.	Responder con citas.	Ranking, groundedness, citas, coste y latencia.
Pregunta respondible, pero evidencia fuera de top-k.	Abstenerse o pedir más contexto.	Retrieval y política de abstención.
Pregunta no respondible por el corpus.	Abstenerse con explicación breve.	Abstención correcta y fallos críticos.
Pregunta ambigua.	Pedir aclaración.	Contrato conversacional y trazas.

Diagnóstico por capas

Cuando un RAG falla, no cambies todo a la vez

El síntoma te dice qué capa mirar primero y qué acción tiene sentido.

Síntoma	Capa probable	Señal	Acción técnica	No hagas esto primero
Recall bajo falta evidencia esperada	corpus, chunking, filtros embedding o búsqueda híbrida	Recall@k, Hit@k qrels por slice	revisar índice y fuentes probar BM25 + vector	cambiar el modelo generador sin mirar retrieval
nDCG bajo la evidencia aparece tarde	ranking o reranking orden de contexto	MRR, nDCG@k posición de gold chunks	añadir reranker o RRF medir latencia p95	subir top-k sin mirar ruido ni coste de contexto
Claims sin soporte la respuesta añade de más	prompt, contrato de salida o contexto incompleto	groundedness operativo claims con soporte	cita por afirmación validar salida estructurada	aceptar una cita al final como si cubriera todo
No se abstiene responde sin evidencia	política de respuesta y casos no respondibles	abstention accuracy critical failures	bloquear release añadir regresiones	premiar respuestas largas sin evidencia verificable
P95 o coste sube la mejora no cabe en operación	top-k, reranker, tokens y modelo generador	p95, tokens, coste por respuesta aceptada	comparar variantes con restricciones	elegir la métrica más alta ignorando el SLO

Una evaluación RAG debe producir diagnóstico accionable. Si la salida solo dice “calidad 0,72”, todavía no te ayuda a arreglar nada.

Cuando un RAG falla, el orden de trabajo importa. Si Recall@k es bajo, no empieces retocando el prompt. Si Recall@k es alto pero groundedness baja, mira el contexto final y el contrato de salida. Si la respuesta tiene citas, pero no sostienen las afirmaciones, el problema no es “falta citar”, sino “falta trazabilidad claim-cita”. Si todo mejora pero p95 y coste se disparan, el candidato quizá sirve para un flujo de revisión, pero no para producción.

SÍNTOMA	QUÉ MIRAR PRIMERO	CAMBIO RAZONABLE
Recall@k bajo.	Corpus, filtros, embeddings, búsqueda híbrida, query rewriting.	Mejorar qrels, chunking, índice o estrategia de búsqueda.
Recall alto, nDCG bajo.	Orden de resultados.	Añadir reranker, RRF o señales de metadata.
Retrieval correcto, groundedness baja.	Prompt, contrato de citas, claims y contexto final.	Exigir respuesta con soporte y validar afirmaciones.
Citas presentes pero débiles.	Asociación claim-cita.	Citas por afirmación, no bibliografía decorativa.
Buena calidad, coste alto.	Top-k, reranker, tamaño de chunks, cache, modelo.	Reducir contexto, cachear retrieval o usar ruta escalonada.
Responde sin evidencia.	Casos no respondibles, umbral y contrato de abstención.	Endurecer política y añadir regresiones.
Funciona en media, falla en un grupo.	Slices.	Métricas por idioma, producto, fuente, fecha o perfil.

Trazas de depuración

Una evaluación sin traza solo dice “falló”. Una evaluación con traza te dice dónde mirar. Una traza útil no guarda solo la respuesta final; guarda versiones, filtros, scores, tokens, latencia y coste.

```
{
  "run_id": "rag-run-2026-05-31-00042",
  "case_id": "rag_002",
  "pipeline_version": "rag-pipeline-v0.7.3",
  "corpus_version": "normativa-campus-2026-05-30",
  "index_version": "hnsw-embeddings-2026-05-30",
  "retriever": {
    "type": "hybrid_rrf",
    "top_k_sparse": 20,
    "top_k_dense": 20,
    "top_k_final": 5,
    "filters": {"course": "2026", "document_status": "vigente"}
  },
  "retrieved_chunks": [
    {
      "rank": 1,
      "chunk_id": "normativa-2026#plazos-ampliacion",
      "score_dense": 0.88,
      "score_sparse": 12.4,
      "tokens": 170
    }
  ],
  "context": {
    "chunks_sent": ["normativa-2026#plazos-ampliacion"],
    "tokens_sent": 170,
    "truncated_chunks": []
  },
  "answer": {
    "abstained": false,
    "citations": ["normativa-2026#plazos-ampliacion"],
    "claims_count": 2,
    "output_tokens": 78
  },
  "latency_ms": {"retrieval": 42, "generation": 920, "total": 1115},
  "estimated_cost": {"generation": 0.0038, "total": 0.004}
}
```

Con una traza así puedes contestar preguntas reales:

PREGUNTA	CAMPO QUE LA RESPONDE
¿Cambiamos el índice sin darnos cuenta?	`index_version`
¿El filtro de curso estaba activo?	`retriever.filters`
¿La evidencia se recuperó pero quedó fuera del contexto?	`retrieved_chunks` y `context.chunks_sent`
¿El contexto se recortó?	`truncated_chunks`
¿La respuesta se encareció por salida larga?	`answer.output_tokens`
¿El fallo viene de generación o retrieval?	comparación entre `retrieved_chunks`, `context` y `answer`

Comparar variantes sin hacerse trampas

Un RAG no mejora por intuición. Mejora comparando variantes controladas. La palabra importante es **controladas**: si cambias embeddings, chunking, reranker, prompt y modelo a la vez, quizá suba una métrica, pero no sabrás qué pieza produjo la mejora.

Una matriz mínima de experimentos:

VARIANTE	QUÉ CAMBIA	QUÉ QUEDA FIJO	MÉTRICAS QUE MIRAS
`bm25_base`	Búsqueda léxica.	Corpus, chunks, prompt, modelo.	Recall@k, nDCG, latencia.
`dense_base`	Embedding denso.	Corpus, chunks, top-k, prompt, modelo.	Recall@k, MRR, coste de índice.
`hybrid_rrf`	Fusión BM25 + vector.	Corpus, chunks, prompt, modelo.	Recall@k, precision@k, nDCG.
`hybrid_rerank`	Añade reranker.	Corpus, chunks, generador.	nDCG, groundedness, latencia p95.
`chunk_300`	Chunks más pequeños.	Índice, prompt, modelo.	Recall@k, citation recall, tokens.
`chunk_900`	Chunks más grandes.	Índice, prompt, modelo.	Groundedness, ruido, coste.
`topk_8`	Más contexto.	Retriever, chunks, prompt, modelo.	Recall, precisión de contexto, tokens.
`strict_abstain`	Umbral más exigente.	Retriever, corpus, respuesta.	Abstención correcta, cobertura y satisfacción.

RRF, Reciprocal Rank Fusion, es una técnica sencilla para fusionar rankings de varios sistemas y suele usarse al combinar búsqueda léxica y vectorial.[5] No necesitas venderla como solución mágica. Necesitas medir si, para tu corpus, sube cobertura sin disparar latencia, coste o ruido.

También debes protegerte de la fuga de evaluación. Si usas los mismos casos para ajustar prompts, retocar chunks, cambiar filtros y declarar victoria, has entrenado contra el examen. Separa al menos tres conjuntos:

CONJUNTO	USO CORRECTO	QUÉ NO DEBERÍAS HACER
`dev`	Ajustar prompts, top-k, chunking y umbrales.	Presentarlo como resultado final.
`regression`	Vigilar errores conocidos que no deben volver.	Usarlo como único dataset de calidad.
`holdout`	Estimar rendimiento antes de publicar.	Mirarlo cada vez que retocas el sistema.

Si el dataset es pequeño, una mejora de 2 puntos puede ser ruido. Para comparaciones pareadas puedes usar bootstrap para estimar incertidumbre o McNemar cuando comparas dos sistemas en aciertos/errores emparejados.[6]

En el día a día

Imagina un asistente de normativa académica. La pregunta es: “¿Puedo ampliar matrícula si tengo un pago pendiente?”. La respuesta correcta necesita dos evidencias: el plazo de ampliación y la regla sobre pagos pendientes. Si el RAG recupera solo el plazo, Hit@3 puede ser 1, pero Recall@3 será 0,5. Si responde “sí, puedes” citando solo el plazo, la respuesta parece útil, pero no está completa.

Ahora imagina soporte interno. Una persona pregunta cómo rotar una clave de producción. El corpus tiene una guía antigua y una nueva. Si el retriever trae la guía antigua porque tiene más coincidencias léxicas, la respuesta puede ser peligrosa aunque esté “citada”. La cita no basta: tienes que comprobar vigencia, metadata, filtros y prioridad.

Otro caso muy común aparece en documentación de producto. El usuario pregunta por una funcionalidad que todavía no existe. El RAG encuentra documentos parecidos, el modelo extrapola y da instrucciones inventadas. Aquí la métrica importante no es solo groundedness: es abstención. Si el sistema no sabe decir “no encuentro evidencia suficiente”, no está listo para producción.

Por qué debería importarte

Porque RAG suele entrar en sistemas donde la fuente importa: políticas internas, normativa, contratos, documentación técnica, manuales, tickets, soporte, compliance o conocimiento cambiante. Si no evalúas la cadena de evidencia, no sabes si tu sistema responde por documentos o por memoria estadística del modelo.

Para ingeniería, esto cambia decisiones concretas:

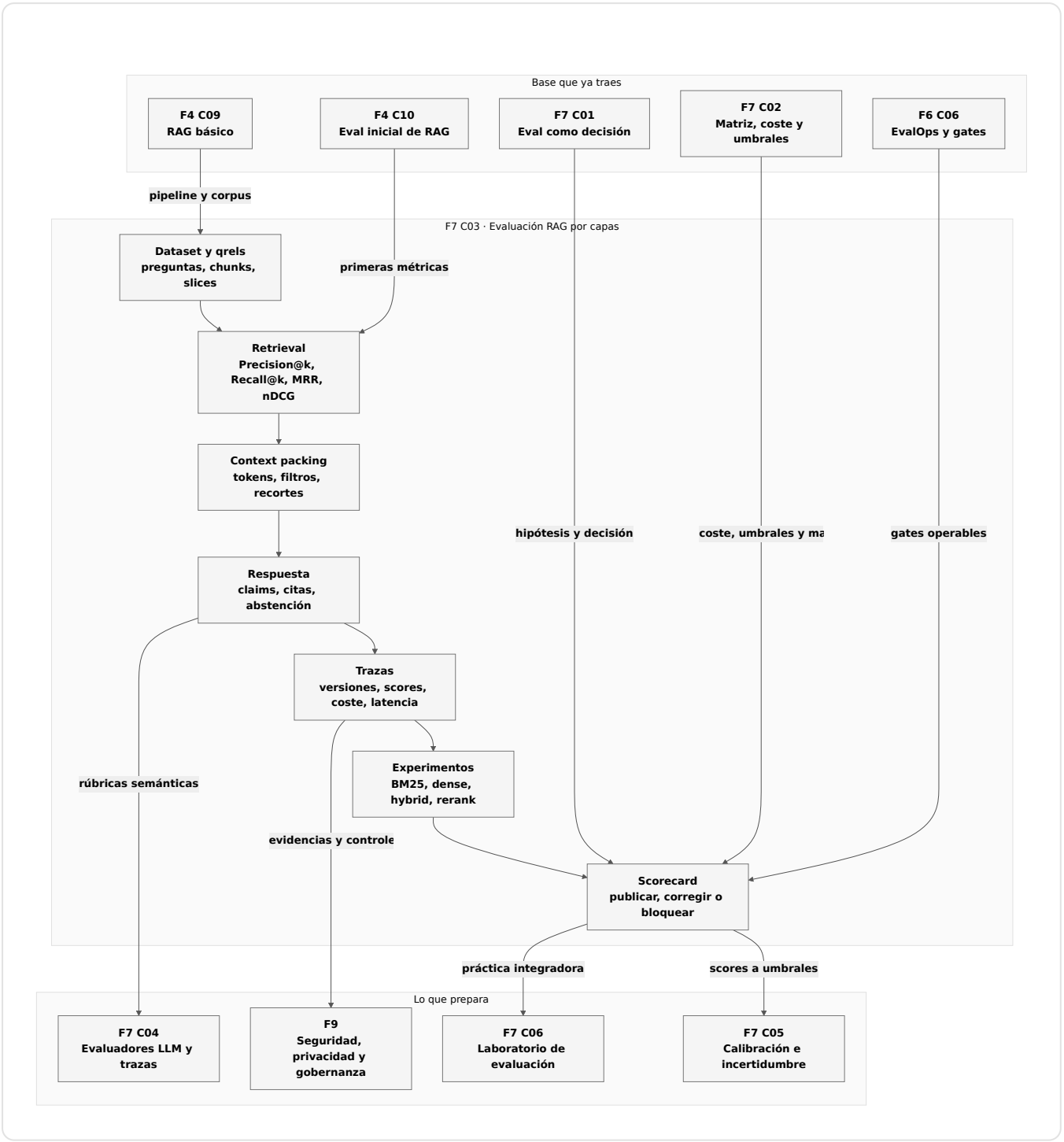
DECISIÓN	SIN EVALUACIÓN POR CAPAS	CON EVALUACIÓN POR CAPAS
Cambiar embeddings	Lo haces porque “parece mejor”.	Lo haces si mejora Recall@k/nDCG por slice sin romper coste.
Subir top-k	Metes más contexto y cruzas dedos.	Mides recall, ruido, tokens y p95.
Añadir reranker	Lo vendes como mejora automática.	Lo publicas solo si sube ranking y cabe en SLO.
Ajustar prompt	Lo usas para tapar fallos de retrieval.	Lo tocas cuando ya sabes que el contexto llega.
Publicar una versión	Mirar ejemplos bonitos.	Scorecard con umbrales y fallos críticos.

Esta forma de trabajar también evita discusiones vagas. Si el equipo de producto dice “responde bien”, ingeniería puede preguntar “¿en qué slices, con qué qrels, con qué tasa de abstención y con qué p95?”. No es pedantería: es proteger la decisión.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Medir solo la respuesta final	Si la respuesta falla, no sabes si arreglar corpus, chunking, retrieval, reranking, prompt o citas.	Medir por capas antes de tocar el sistema.
Celebrar Hit@k	Hit@k puede ser 1 aunque falte la segunda fuente necesaria.	Mirar Recall@k y casos multi-evidencia.
Confundir cita con evidencia	Una cita puede apuntar a un documento real y aun así no sostener la frase.	Validar claim por claim.
Subir top-k sin mirar coste	Más contexto puede traer evidencia, pero también ruido, tokens y latencia.	Medir recall, precision, nDCG y coste juntos.
No tener preguntas no respondibles	Si todas las preguntas tienen respuesta, nunca mides abstención.	Incluir casos plausibles sin evidencia.
Cambiar umbrales después de ver el resultado	Convierte la eval en ajuste manual.	Escribir la política antes de ejecutar.
Comparar variantes cambiándolo todo a la vez	Si sube la métrica, no sabes si fue por embeddings, reranker, prompt, chunks o corpus.	Usar una matriz de experimentos con una variable principal por variante.
No versionar el índice	Puedes creer que comparas dos pipelines cuando en realidad cambió el índice.	Guardar `corpus_version`, `chunker_version`, `embedding_model` e `index_version`.
Usar el holdout como zona de pruebas	Si miras el examen final cada vez que ajustas, deja de ser final.	Separar `dev`, `regression` y `holdout`.

Cómo encaja todo



Este capítulo es el puente entre construir un RAG y operar un RAG. El facsímil 4 te da la arquitectura; el facsímil 6 te recuerda que una métrica debe convertirse en gate; C1 y C2 de este facsímil te dan unidad de evaluación, coste y umbrales; C3 aplica todo eso a evidencia documental. C4 usará evaluadores LLM cuando haga falta criterio semántico, pero ahora ya tienes una base medible sin delegarlo todo en otro modelo.

Vocabulario aprendido

TÉRMINO	DEFINICIÓN BREVE
RAG	Arquitectura que combina recuperación de evidencia con generación.
Qrel	Juicio de relevancia entre pregunta y chunk.
Relevancia graduada	Puntuación que diferencia evidencia central, parcial o irrelevante.
Precision@k	Proporción de resultados útiles entre los k primeros.
Recall@k	Proporción de evidencias esperadas que aparecen en top-k.
Hit@k	Si al menos una evidencia esperada aparece en top-k.
MRR	Media del recíproco de la posición de la primera evidencia útil.
nDCG@k	Métrica de ranking que premia relevancia alta en posiciones altas.
Context packing	Selección y ordenación del contexto que entra al modelo.
Groundedness	Comprobación de si las afirmaciones están sostenidas por evidencia recuperada.
Cita válida	Cita que apunta a evidencia que sostiene lo afirmado.
Abstención	No responder cuando falta evidencia suficiente.
Fallo crítico	Respuesta sin evidencia en un caso que debía abstenerse.
Scorecard	Resumen de métricas y restricciones para decidir.
RRF	Técnica para fusionar rankings de varios retrievers.
Traza	Registro de una run: versiones, recuperación, contexto, respuesta, latencia y coste.
Holdout	Conjunto reservado para estimar rendimiento sin usarlo para ajustar.
Variante dominada	Configuración peor o igual en calidad y peor o igual en coste frente a otra.
Shadow	Ejecución paralela de una versión nueva sin responder todavía al usuario.

Antes de pasar página

Antes de avanzar, deberías poder responder:

1. ¿Por qué una respuesta correcta puede seguir siendo mala señal en un RAG?
2. ¿Qué diferencia hay entre Hit@k y Recall@k?
3. ¿Cuándo usarías nDCG@k en vez de solo Recall@k?
4. ¿Qué contiene un qrel útil?
5. ¿Por qué groundedness debe mirar afirmaciones y no impresiones?
6. ¿Qué diferencia práctica hay entre precisión de cita y cobertura de cita?
7. ¿Por qué necesitas preguntas no respondibles en el dataset?
8. ¿Qué métrica o check bloquearía una versión que responde sin evidencia?
9. ¿Qué cambiarías si Recall@k es alto pero groundedness es bajo?
10. ¿Qué debería guardar una traza profesional de RAG?
11. ¿Por qué una matriz de experimentos debe cambiar una pieza cada vez?
12. ¿Qué diferencia hay entre `dev`, `regression` y `holdout`?
13. ¿Qué significa que una variante esté dominada?
14. ¿Qué archivos entrega la práctica del capítulo?

En resumen

IDEA	QUÉ TE LLEVAS
RAG se evalúa por capas.	Corpus, retrieval, contexto, respuesta, citas y abstención tienen señales distintas.
Los qrels sostienen la evaluación.	Sin evidencia esperada no puedes medir retrieval de forma seria.
Retrieval se mide antes de generación.	Ahorras coste y localizas el fallo antes de culpar al modelo.
Las fórmulas académicas están en ranking.	Precision@k, Recall@k, MRR y nDCG@k tienen tradición de IR y fuentes reconocibles.
Groundedness es una señal operativa.	En la práctica se audita claim por claim, no como impresión general.
Abstención es parte de calidad.	Responder sin evidencia puede bloquear una versión.
La scorecard debe decidir.	Métricas sin gate no cambian el sistema.
Las variantes se comparan con diseño.	Una matriz de experimentos evita mejorar a ciegas.
La traza es parte de la eval.	Sin versiones, scores, contexto, coste y latencia no hay depuración seria.
El holdout se protege.	Si optimizas contra el examen final, la mejora deja de ser fiable.

Para saber más

Cormack, G. V., Clarke, C. L. A. y Buettcher, S. (2009). Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. *SIGIR*, 758-759. https://doi.org/10.1145/1571941.1572114

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1), 1-26. https://doi.org/10.1214/aos/1176344552

Es, S., James, J., Espinosa-Anke, L. y Schockaert, S. (2023). *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. https://arxiv.org/abs/2309.15217

Järvelin, K. y Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4), 422-446. https://doi.org/10.1145/582415.582418

LangChain. (2026). *Evaluate a RAG application*. https://docs.langchain.com/langsmith/evaluate-rag-tutorial

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

LlamaIndex. (2026). *Evaluation modules*. https://developers.llamaindex.ai/python/framework/module_guides/evaluating/modules/

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2), 153-157. https://doi.org/10.1007/BF02295996

Muennighoff, N. y otros (2023). *MTEB: Massive Text Embedding Benchmark*. https://arxiv.org/abs/2210.07316

Phoenix. (2026). *Evaluate RAG*. https://arize.com/docs/phoenix/cookbook/evaluation/evaluate-rag

Ragas. (2026). *List of available metrics*. https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/

Thakur, N. y otros (2021). *BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models*. https://arxiv.org/abs/2104.08663

TruLens. (2026). *RAG Triad*. https://www.trulens.org/getting_started/core_concepts/rag_triad/

Notas

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

2. Thakur, N. y otros (2021). *BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models*. arXiv:2104.08663. Muennighoff, N. y otros (2023). *MTEB: Massive Text Embedding Benchmark*. arXiv:2210.07316. ↩

3. Es, S., James, J., Espinosa-Anke, L. y Schockaert, S. (2023). *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. arXiv:2309.15217. Ragas. (2026). *List of available metrics*. https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/. TruLens. (2026). *RAG Triad*. https://www.trulens.org/getting_started/core_concepts/rag_triad/. LangChain. (2026). *Evaluate a RAG application*. https://docs.langchain.com/langsmith/evaluate-rag-tutorial. Arize Phoenix. (2026). *Evaluate RAG*. https://arize.com/docs/phoenix/cookbook/evaluation/evaluate-rag. ↩

4. Järvelin, K. y Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4), 422-446. https://doi.org/10.1145/582415.582418 ↩

5. Cormack, G. V., Clarke, C. L. A. y Buettcher, S. (2009). Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. *SIGIR*, 758-759. https://doi.org/10.1145/1571941.1572114 ↩

6. Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1), 1-26. McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2), 153-157. ↩

Capítulo 04: Evaluadores LLM y agentes: rúbricas, trazas y coste

Entrando en el tema

En el capítulo anterior evaluamos RAG por capas. Ahora entra una pieza incómoda: muchas respuestas de IA no se pueden corregir solo con `exact match`, JSON Schema o una fórmula. Hay que valorar utilidad, suficiencia, claridad, groundedness, orden de razonamiento operativo o trayectoria de un agente. Ahí aparece el evaluador LLM.

Un evaluador LLM puede ser útil. También puede ser una fuente nueva de error. Por eso este capítulo no va de "pon otro modelo a corregir". Va de **evaluar al evaluador**.

Al terminar deberías poder hacer esto:

RESULTADO DE APRENDIZAJE	EVIDENCIA DE QUE LO SABES HACER
Decidir cuándo usar un evaluador LLM.	Primero separas validadores deterministas, métricas y revisión humana.
Escribir una rúbrica evaluable.	Defines criterios observables, escala, ejemplos y condiciones de bloqueo.
Calibrar un evaluador.	Comparas sus veredictos contra un conjunto revisado por personas.
Medir acuerdo.	Calculas accuracy, kappa, pases indebidos y errores por criterio.
Evaluar trazas de agentes.	Puntúas resultado final, tools, orden, argumentos, permisos, coste y latencia.
Controlar coste de evaluación.	Calculas coste por evaluación útil y presupuesto de eval.
Diseñar un gate con evaluador.	Un evaluador ayuda, pero no decide solo si el sistema se publica.

La idea central: **un evaluador LLM no es una fuente de verdad; es un instrumento que se calibra, se monitoriza y se limita**.

El problema: corregir lenguaje abierto no es como validar JSON

Hay tareas donde una máquina puede validar casi todo:

TAREA	VALIDACIÓN SUFICIENTE
Salida JSON	Schema, campos obligatorios, tipos y catálogos.
Cálculo	Resultado numérico y tolerancia.
Tool call	Nombre de tool, argumentos, permisos y error esperado.
Código	Tests, lint, tipos, diff y cobertura.
Cita RAG	Chunk citado existe y sostiene una afirmación.

Pero hay otras tareas más abiertas:

TAREA	POR QUÉ CUESTA VALIDARLA
Resumir un informe.	Puede haber varias respuestas correctas.
Explicar un concepto.	Importan claridad, completitud y nivel del público.
Revisar una respuesta de soporte.	Importan tono, precisión y siguiente paso.
Evaluar un agente.	Importa la trayectoria, no solo la frase final.
Comparar dos variantes.	A veces hay que decidir cuál ayuda más con el mismo dato.

Aquí un evaluador puede ayudar a escalar revisión. Pero si el evaluador no tiene rúbrica, no tiene ejemplos, no se compara contra personas y no conserva trazas, solo cambia una opinión por otra opinión con apariencia de número.

Fecha de corte del estado del arte

Fecha de corte: 31 de mayo de 2026.
Fuentes consultadas: documentación de OpenAI Graders, OpenAI agent evals y trace grading; LangSmith LLM-as-judge y evaluación; Ragas rubrics; Phoenix LLM evals; Google ADK Evaluate; OpenTelemetry; y trabajos sobre LLM-as-a-judge, G-Eval, AgentBench y WebArena.

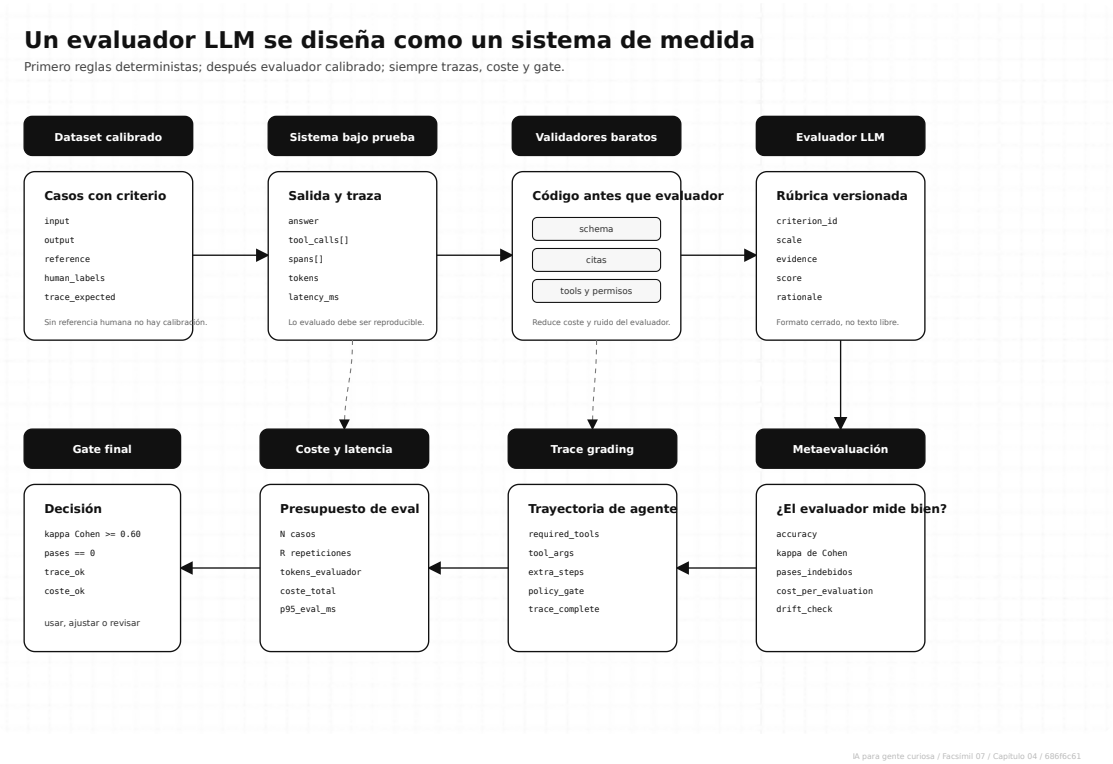
En castellano usaré **evaluador LLM**. En documentación y papers aparece a menudo como `LLM-as-a-judge`; lo citaremos así cuando sea el nombre técnico de la fuente, pero en el cuerpo del capítulo hablaremos de evaluadores.

OpenAI documenta graders para evals y fine-tuning, incluyendo model graders, validación del grader y ejecución con muestras de prueba.[1] LangSmith permite definir evaluadores LLM, usando la denominación `LLM-as-a-judge`, para evaluación offline y online sobre trazas.[2] Ragas ofrece métricas basadas en rúbricas y criterios definidos por el usuario.[3]

Zheng y otros estudiaron LLM-as-a-judge en MT-Bench y Chatbot Arena, señalando acuerdo alto con preferencias humanas en ciertos entornos, pero también sesgos de posición, verbosidad y preferencia por respuestas propias.[4] G-Eval propuso usar LLMs con instrucciones y formularios de evaluación para tareas de generación, con mejor correlación con valoraciones humanas que métricas automáticas clásicas en los experimentos reportados.[5] Para agentes, AgentBench y WebArena muestran que evaluar acción en entornos interactivos exige mirar trayectorias, no solo respuestas finales.[6]

La conclusión útil no es “los evaluadores funcionan” ni “los evaluadores no funcionan”. La conclusión adulta es: **funcionan bajo diseño, calibración, trazas y límites**.

Anatomía de un sistema de evaluadores



Un evaluador LLM entra en un sistema de medición: validadores baratos, rúbrica, trazas, calibración, coste y gate.

Primero código, luego evaluador

La regla más barata y más sana:

> Si puedes evaluarlo con código, no lo mandes primero a un evaluador LLM.

CRITERIO	MEJOR PRIMERA OPCIÓN	CUÁNDO ENTRA EL EVALUADOR
JSON válido	Parser y schema.	Casi nunca.
Campos obligatorios	Validación estructurada.	Casi nunca.
Cita existe	Lookup contra chunks recuperados.	Si hay que valorar si sostiene una frase.
Tool correcta	Comparación de trayectoria.	Si hay varias trayectorias aceptables.
Argumentos de tool	Schema, rangos, catálogos.	Si el argumento es semántico.
Cálculo	Recalcular.	Casi nunca.
Resumen útil	Rúbrica y evaluador calibrado.	Cuando no hay respuesta única.
Explicación didáctica	Rúbrica y ejemplos.	Cuando importa nivel, claridad y completitud.

Esto no es una manía. Es coste, reproducibilidad y depuración. Un validador determinista suele ser más barato, más estable y más fácil de explicar que un evaluador.

Qué es una rúbrica evaluable

Una rúbrica no es “califica del 1 al 5”. Eso es una invitación al ruido. Una rúbrica evaluable tiene criterios observables, escala concreta, ejemplos y condiciones de bloqueo.

PIEZA	PREGUNTA	EJEMPLO
Criterio	¿Qué se evalúa?	`groundedness`, `completitud`, `tono`, `trayectoria`.
Evidencia	¿Qué debe mirar el evaluador?	Respuesta, referencia, contexto, trazas, tools.
Escala	¿Qué significa cada valor?	0, 1, 2 o 3 con descripciones cerradas.
Bloqueo	¿Qué caso no puede aprobar?	Respuesta sin evidencia cuando la tarea exige fuente.
Ejemplos	¿Cómo se ve cada nota?	Casos calibrados con explicación humana.
Salida	¿Qué formato devuelve?	JSON con `score`, `label`, `rationale`, `evidence`.

Ejemplo de criterio:

```
{
  "criterion_id": "groundedness",
  "description": "La respuesta debe apoyarse en la evidencia proporcionada.",
  "scale": {
    "0": "Afirmaciones centrales sin soporte.",
    "1": "Parte de la respuesta tiene soporte, pero falta una condición importante.",
    "2": "La respuesta está mayoritariamente soportada, con detalle menor discutible.",
    "3": "Todas las afirmaciones relevantes están soportadas por evidencia citada."
  },
  "blocking_rule": "Si una conclusión importante no tiene soporte, el caso no puede
aprobar.",
  "required_evidence": ["answer", "reference", "retrieved_context", "citations"]
}
```

La escala debe evitar adjetivos vagos. “Bueno” o “malo” no bastan. El evaluador necesita saber qué evidencia convierte un 1 en un 2 y un 2 en un 3.

Contrato de salida del evaluador

La rúbrica dice qué debe mirar el evaluador. El contrato de salida dice qué debe devolver para que un sistema pueda auditarlo. Si la salida es texto libre, cada integración acaba escribiendo parsers frágiles, excepciones raras y revisiones manuales. Si la salida es estructurada, el evaluador se convierte en una pieza de ingeniería: se valida, se versiona y se compara.

Un contrato mínimo debería incluir identidad del caso, versión de rúbrica, decisión, puntuaciones por criterio, evidencia usada, razón breve, trazabilidad y metadatos de coste. No hace falta que todos los equipos usen los mismos nombres, pero sí que los campos sean estables.

```
{
  "case_id": "evaluator_002",
  "rubric_version": "academic_agent_evaluator_v1",
  "evaluator_version": "evaluator_v2_rubric",
  "pass": false,
  "scores": {
    "answer_quality": 0,
    "evidence": 0,
    "trace": 0,
    "policy": 1
  },
  "blocking_reasons": ["evidence"],
  "evidence": [
    {
      "source": "retrieved_context",
      "quote_or_span": "La fuente no sostiene la cifra exacta",
      "supports_decision": true
    }
  ],
  "rationale": "La respuesta cita una fuente real, pero la fuente no sostiene el dato
numérico que se presenta como probado.",
  "trace_span_id": "span_9f2c",
  "parse_ok": true,
  "input_tokens": 1120,
  "output_tokens": 170
}
```


La diferencia entre una salida así y una nota del 1 al 5 es enorme. Con una nota global sabes que “algo” fue mal. Con un contrato sabes si falló evidencia, traza, política, parseo o coste. Además puedes escribir tests: si `parse_ok` es falso, si falta `rubric_version`, si `blocking_reasons` está vacío aunque `evidence` sea 0, el caso no debería poder pasar.

En el cuaderno del facsímil trabajamos con `schemas/evaluator_output.schema.json`, `templates/evaluator_prompt.md` y `templates/eval_run_card.md` para que esto no se quede en una recomendación abstracta. Puedes abrir esos archivos, adaptarlos y usarlos como punto de partida en una práctica real.

Tipos de evaluadores

No todos los evaluadores son iguales. Conviene elegir el tipo más simple que responda la pregunta.

TIPO	QUÉ DEVUELVE	SIRVE PARA	RIESGO TÉCNICO
Clasificador binario	`pass/fail`	Gates, contratos semánticos, revisión rápida.	Puede ocultar matices.
Escala ordinal	0-3, 1-5	Calidad, completitud, claridad.	Los números pueden no estar calibrados.
Comparador pareado	Gana A, gana B, empate.	Elegir entre dos variantes.	Puede depender del orden de presentación.
Evaluador por criterio	Varias notas separadas.	Diagnóstico útil.	Más coste y más superficie de inconsistencia.
Evaluador de traza	Puntúa pasos, tools y argumentos.	Agentes y workflows.	Requiere trazas limpias y schema estable.
Panel de evaluadores	Varios modelos o prompts.	Casos de alta variabilidad.	Multiplica coste y puede dar falsa seguridad.
Humano asistido	Persona con ayuda de tooling.	Casos de impacto alto o ambigüedad real.	Coste, variabilidad y tiempo.

La elección profesional suele ser híbrida: reglas deterministas para lo verificable, evaluador para lo semántico, revisión humana para casos límite y scorecard para decidir.

Sesgos y fallos frecuentes de un evaluador

Los papers y la práctica coinciden en algo: los evaluadores LLM son útiles, pero tienen patrones de error.

PATRÓN	QUÉ SIGNIFICA	CÓMO MITIGARLO
Sesgo de posición	Prefiere A o B según el orden.	Aleatorizar orden y medir `order_flip_rate`.
Sesgo de verbosidad	Premia respuestas más largas aunque no aporten más.	Rúbrica con penalización de relleno y coste.
Preferencia por estilo	Confunde fluidez con calidad factual.	Separar claridad, evidencia y completitud.
Inconsistencia	Cambia veredicto entre repeticiones.	Temperatura baja, formato cerrado y repetición en casos críticos.
Arrastre de referencia	Copia la respuesta de referencia sin evaluar equivalencia.	Pedir evidencia de decisión, no solo nota.
Atajos de puntuación	Aprende señales superficiales.	Calibrar con casos difíciles y revisar errores.
Falta de sensibilidad al coste	Aprueba respuestas que exigen demasiados pasos.	Incluir tokens, tools y latencia en la rúbrica.

Zheng y otros ya mostraban que hay que vigilar posición, verbosidad y sesgos de preferencia. OpenAI también advierte que un sistema entrenado contra un grader puede aprender a explotar debilidades del propio grader, de modo que conviene contrastarlo con evaluación experta.[7]

Metaevaluación: evaluar al evaluador

Si el evaluador se usa para bloquear releases, necesita su propio expediente.

La primera fórmula no es de Cohen ni tiene un autor único que debamos citar como si fuera un resultado original. Es la definición estándar de exactitud o acuerdo exacto: contar cuántas etiquetas coinciden y dividir por el número total de casos. En aprendizaje automático solemos llamarlo `accuracy`; en un trabajo de anotación también puede leerse como proporción de acuerdo observado.

Sea h_i la etiqueta humana para el caso i y j_i la etiqueta del evaluador:

$$\text{accuracy}_{eval} = \frac{1}{N}\sum_{i=1}^{N} 1[h_i = j_i]$$

Aquí $1[h_i = j_i]$ vale 1 si la etiqueta humana y la etiqueta del evaluador coinciden, y vale 0 si no coinciden. Si tienes 100 casos y el evaluador coincide con la referencia humana en 82, la exactitud es $82/100 = 0{,}82$. Esto es útil, pero puede engañar cuando una clase domina el dataset: un evaluador que siempre diga "aceptable" puede parecer bueno si casi todo el conjunto era aceptable.

La fórmula con autor propio en este bloque es el kappa de Cohen, propuesto por Jacob Cohen en 1960 para medir acuerdo entre dos codificadores en escalas nominales corrigiendo el acuerdo que esperaríamos por azar:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

SÍMBOLO	SIGNIFICADO
p_o	Acuerdo observado entre evaluador y referencia humana.
p_e	Acuerdo esperado por azar según las distribuciones marginales.
κ	Acuerdo corregido por azar.

Si $p_o = 0{,}82$ y, por la distribución de etiquetas, el acuerdo esperado por azar es $p_e = 0{,}50$, entonces $\kappa = (0{,}82 - 0{,}50)/(1 - 0{,}50) = 0{,}64$. La lectura práctica es más honesta que la accuracy sola: no dice solo "coincidimos mucho", sino "coincidimos más de lo que cabría esperar por la frecuencia de las clases". Por eso encaja mejor cuando el evaluador va a decidir gates de release, donde los falsos pases importan más que quedar bonito en una media.

FÓRMULA	DE DÓNDE VIENE	POR QUÉ APARECE AQUÍ
accuracy_{eval}	No es una fórmula con autor propio. Es la definición estándar de exactitud en clasificación: proporción de predicciones correctas sobre el total. Sokolova y Lapalme la tratan dentro del análisis clásico de medidas de evaluación para clasificación.	Sirve como primera lectura de acuerdo entre referencia humana y evaluador, pero no debe decidir sola un gate.
$\kappa = (p_o - p_e)/(1 - p_e)$	Kappa de Cohen, formulado por Jacob Cohen en 1960 para medir acuerdo entre dos codificadores en variables nominales corrigiendo el azar.	Sirve para detectar si el evaluador coincide con humanos más allá de lo que explicaría la distribución de clases.

Y para ingeniería añadimos métricas más incómodas:

MÉTRICA	POR QUÉ IMPORTA
`false_pass_rate`	Casos que el evaluador aprueba y la referencia humana rechaza.
`false_block_rate`	Casos que el evaluador bloquea aunque eran aceptables.
`critical_false_passes`	Pases indebidos en criterios bloqueantes.
`score_mae`	Error absoluto medio si hay escala numérica.
`order_flip_rate`	Cambios de preferencia al invertir A/B.
`rubric_parse_error_rate`	Veces que el evaluador no devuelve formato válido.
`cost_per_useful_evaluation`	Coste dividido entre evaluaciones que pasan control de calidad.

Un evaluador puede tener accuracy alta y aun así no servir para gate si deja pasar justo los casos que más importan.

Antes de calibrar: diseña bien el conjunto humano

Un evaluador no se calibra contra “la verdad”, sino contra una referencia construida. Si esa referencia humana está mal diseñada, el evaluador puede parecer malo por razones injustas o parecer bueno porque el examen era demasiado fácil. Para ingeniería de IA, el `calibration set` no es un apéndice: es el instrumento de medida.

Un buen conjunto de calibración debería mezclar casos fáciles, casos frontera y negativos duros. Los casos fáciles comprueban que el evaluador no está roto. Los casos frontera enseñan qué diferencia un 1 de un 2 o un `pass` de un `fail`. Los negativos duros son los más valiosos: respuestas fluidas pero sin evidencia, citas que existen pero no sostienen la afirmación, herramientas correctas usadas fuera de política, resúmenes que omiten una excepción legal o trayectorias que llegan al resultado por un camino que no puedes publicar.

PIEZA DEL CALIBRATION SET	QUÉ DEBE CONTENER	ERROR SI FALTA
Casos positivos claros	Respuestas que deberían pasar sin discusión.	El evaluador parece demasiado duro.
Casos negativos claros	Respuestas que deberían bloquearse.	El evaluador parece mejor de lo que es.
Casos frontera	Salidas parcialmente correctas, evidencia incompleta o traza discutible.	No aprendes dónde está el límite operativo.
Slices	Idioma, tipo de tarea, severidad, canal, longitud, usuario o dominio.	La media esconde fallos concentrados.
Doble revisión humana	Al menos una muestra revisada por dos personas.	No sabes si el problema es el evaluador o la etiqueta humana.
Guía de etiquetado	Criterios, ejemplos y resolución de desacuerdos.	Cada persona usa una rúbrica distinta.

Aquí entra una idea importante: si dos personas expertas no se ponen de acuerdo, pedir al evaluador automático que acierte una única etiqueta puede ser una trampa. Cohen propuso kappa para dos codificadores; Fleiss generalizó la idea para varios evaluadores nominales; Krippendorff estudió la fiabilidad entre observadores con especial cuidado en análisis de contenido y datos incompletos.[8] La lección práctica no es meter más símbolos, sino no saltarse la pregunta: **¿cuánto acuerdo humano hay antes de pedirle acuerdo al evaluador LLM?**

En un proyecto real, yo guardaría un expediente de calibración con cuatro archivos mínimos: `calibration_cases.jsonl`, `labeling_guide.md`, `human_disagreements.csv` y `eval_run_card.md`. Si el equipo no puede explicar por qué un caso es `fail`, el evaluador tampoco debería usar ese caso como verdad absoluta.

Evaluadores para agentes: mirar trayectoria

En agentes no basta con evaluar la salida final. Necesitamos juzgar la trayectoria.

ELEMENTO DE TRAZA	PREGUNTA DE EVALUACIÓN
`model_call`	¿El prompt y el modelo eran los esperados?
`tool_call`	¿La tool era necesaria y estaba permitida?
`tool_args`	¿Los argumentos eran completos, mínimos y válidos?
`observation`	¿La tool devolvió evidencia útil?
`handoff`	¿Se transfirió la tarea al actor correcto?
`approval`	¿Pidió aprobación cuando la política lo exigía?
`retry`	¿El reintento tenía motivo y presupuesto?
`final_answer`	¿La respuesta final refleja la evidencia y el estado real?

Podemos separar la evaluación de salida y trayectoria sin inventar una fórmula. Lo correcto es escribir una scorecard con componentes y pesos decididos antes de ejecutar la eval.

COMPONENTE	QUÉ MIDE	DECISIÓN PRÁCTICA
Salida final	Si la respuesta final es correcta, útil y apoyada en evidencia.	No basta si la trayectoria incumple política.
Trayectoria	Tools usadas, orden, argumentos, observaciones y handoffs.	Penaliza pasos innecesarios o herramientas incorrectas.
Política	Permisos, aprobaciones, límites, datos sensibles y acciones externas.	Puede bloquear aunque la salida suene bien.
Operación	Trazas completas, reintentos, finalización limpia, coste y latencia.	Decide si el agente se puede mantener en producción.

La scorecard no convierte la valoración en verdad automática. Obliga a escribir qué pesa más. Un agente que ahorra tiempo pero usa tools de más puede ser ineficiente. Un agente que da buena respuesta pero omite una aprobación no debe pasar. Un agente que necesita tres reintentos por caso puede salir caro aunque responda bien.

Coste de evaluar con evaluadores

Evaluar también cuesta. Si un evaluador automático corre sobre 10.000 casos con respuestas largas y trazas completas, puedes descubrir el problema en la factura.

Un presupuesto mínimo debe desglosar las partidas en vez de esconderlas en una fórmula propia. Ajusta la tabla si tu proveedor cobra por llamada, por token, por batch, por tool o por revisión humana.

PARTIDA	QUÉ DEBES ESTIMAR
Número de casos	Tamaño del dataset que quieres evaluar.
Repeticiones	Repeticiones por caso, panel de evaluadores o inversión de orden A/B.
Tokens de entrada	Prompt, rúbrica, respuesta, referencia y traza.
Tokens de salida	JSON final, explicación breve y campos obligatorios.
Revisión humana	Calibración inicial, desacuerdos y casos límite.
Evaluaciones válidas	Casos que devuelven formato correcto y señal usable.

El coste que me interesa de verdad es el coste por evaluación útil: el coste total dividido entre evaluaciones válidas y aceptadas. Un evaluador que falla formato, cambia criterio entre repeticiones o exige revisión humana constante encarece el sistema aunque parezca barato por llamada.

Si un evaluador devuelve JSON inválido, cambia criterio entre repeticiones o exige revisión humana constante, su coste útil sube.

Checklist de publicación de un evaluador

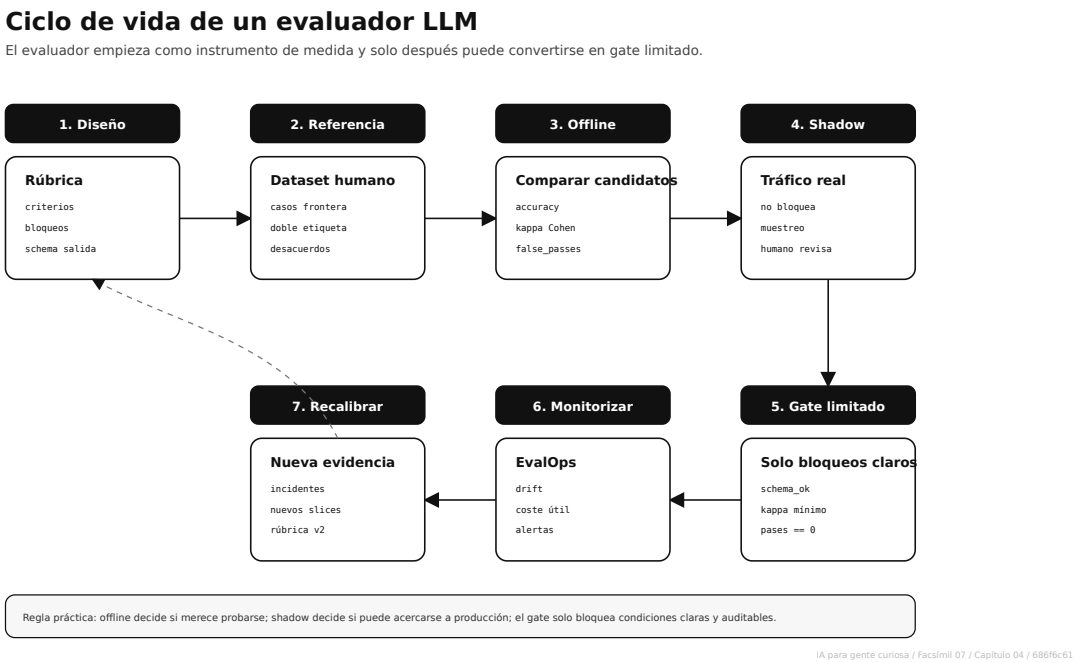
Antes de usar un evaluador como gate, pediría esto:

CONTROL	PREGUNTA
Rúbrica versionada	¿Sabemos qué criterio se usó?
Calibration set	¿Hay casos con referencia humana?
Kappa mínimo	¿El acuerdo corrige azar?
Pases indebidos	¿Cuántos casos malos aprueba?
Parseo estricto	¿Siempre devuelve JSON válido?
Estabilidad	¿Repite veredicto en casos frontera?
Coste	¿Sabemos cuánto cuesta por evaluación útil?
Trazas	¿Podemos reconstruir qué vio el evaluador?
Drift	¿Reevaluamos cuando cambia modelo, prompt o rúbrica?
Revisión humana	¿Qué casos llegan a persona?

Un evaluador sin checklist puede ser útil en exploración. Un evaluador con checklist puede formar parte de ingeniería.

Del experimento al gate: ciclo de vida

El evaluador no debería saltar directamente de una prueba local a bloquear releases. La secuencia sana se parece más a un despliegue progresivo: primero diseño, después calibración, luego ejecución en sombra, después gate limitado y, solo cuando hay evidencia, uso operativo.



El evaluador se despliega como cualquier pieza sensible: diseño, referencia humana, prueba offline, sombra, gate limitado y recalibración.

MODO	QUÉ EVALÚA	QUÉ DECISIÓN PERMITE	QUÉ NO DEBERÍAS HACER
Offline	Dataset versionado, casos calibrados, outputs guardados.	Comparar prompts, modelos o evaluadores antes de publicar.	Concluir que producción irá igual que el dataset.
Shadow mode	Tráfico real o preproducción sin bloquear al usuario.	Ver coste, parseo, drift, falsos pases y carga de revisión humana.	Usarlo como autoridad si todavía no tiene expediente.
Gate limitado	Condiciones claras y auditables.	Bloquear releases o mandar casos a revisión.	Bloquear por una nota global no explicada.
Monitorización online	Muestras, trazas, alertas, drift y errores humanos confirmados.	Recalibrar o retirar el evaluador.	Dejar el evaluador fijo mientras cambia el producto.

Goodhart formuló el problema de las métricas que se convierten en objetivo en economía, pero la intuición se aplica muy bien aquí: si el equipo optimiza para complacer al evaluador, el evaluador deja de medir bien.[9] En IA generativa esto se ve rápido: respuestas más largas porque el evaluador premia completitud, citas decorativas porque el evaluador busca URLs, o razonamientos teatrales porque el evaluador confunde explicación con evidencia.

La defensa no es una métrica mágica. Es una combinación de contrato de salida, casos frontera, revisión humana de muestra, trazas completas y recalibración cuando cambian modelo, prompt, política, canal o distribución de usuarios.

Herramientas reales que verás en equipos

No hace falta casarse con una herramienta para entender la arquitectura. Lo útil es saber qué capa cubre cada una y qué hueco deja.

HERRAMIENTA O PLATAFORMA	CAPA DONDE ENCAJA	QUÉ APORTA	CUIDADO PRÁCTICO
OpenAI Graders / Evals	Evaluación y graders programables.	Permite definir evaluadores para workflows de eval y fine-tuning.	En junio de 2026 OpenAI anunció deprecación de la plataforma Evals clásica; hay que revisar la ruta actual antes de montar algo nuevo.[10]
LangSmith	Evaluación offline, online y trazas.	Evaluadores LLM, evaluadores de código, datasets y ejecución sobre traces.	Si no etiquetas bien los datasets, solo tendrás dashboards bonitos.
Phoenix / Arize	Observabilidad y evals sobre trazas.	Evals LLM, datasets, anotaciones humanas e integración con OpenInference.	Es potente cuando instrumentas trazas; sin trazas, pierde mucha señal.
Ragas	Métricas para RAG y rúbricas.	Métricas de RAG, factualidad, rúbricas y evaluadores basados en LLM.	Sus métricas también necesitan datos, contexto y configuración de dominio.
Promptfoo	CLI, CI/CD, comparación y red teaming.	Tests de prompts, modelos, RAGs, assertions, matrices y seguridad.	Muy útil para CI, pero exige escribir casos específicos, no solo ejecutar una plantilla.
DeepEval	Framework tipo tests para apps LLM.	Casos de prueba, métricas, G-Eval y flujo parecido a testing.	Puede dar sensación de cobertura si los casos no representan fallos reales.
Giskard	Testing, escaneo y red teaming.	Detección de vulnerabilidades, sesgos y problemas de calidad en sistemas IA.	Lo generado por la herramienta debe entrar en tu golden dataset, no quedarse en informe.
Google ADK Evaluate	Evaluación de agentes ADK.	Golden datasets, configuración de criterios y detección de regresiones en agentes.	Encaja mejor si tu agente ya vive en el ecosistema ADK o puedes adaptar sus trazas.
Braintrust	EvalOps y gestión de experimentos.	Datasets, experimentos, comparación y monitorización de cambios.	El valor aparece cuando el equipo versiona prompts, datasets y decisiones, no solo runs.

En el día a día

Imagina un agente académico que revisa una cita:

1. El usuario pega un párrafo con una afirmación.
2. El agente busca la fuente.
3. Comprueba si la fuente sostiene la afirmación.
4. Propone una cita en APA.
5. Indica si falta dato o si conviene revisar.

Salida final posible:

> “La fuente localizada sostiene la idea general, pero no la cifra exacta. Te propongo citarla solo para la parte conceptual y revisar la cifra antes de publicarla.”

Un evaluador de salida puede puntuar utilidad y claridad. Pero un evaluador de traza debe mirar más:

PASO	QUÉ SE EVALÚA
Búsqueda	¿Buscó fuente antes de validar?
Fuente	¿Usó una fuente recuperable y no solo memoria del modelo?
Verificación	¿Separó idea general de cifra exacta?
APA	¿Generó formato correcto con datos disponibles?
Límite	¿Pidió revisión donde falta evidencia?
Coste	¿Usó una ruta razonable para una tarea corta?

Ese es el salto: **evaluar agentes es evaluar una ejecución, no una frase**.

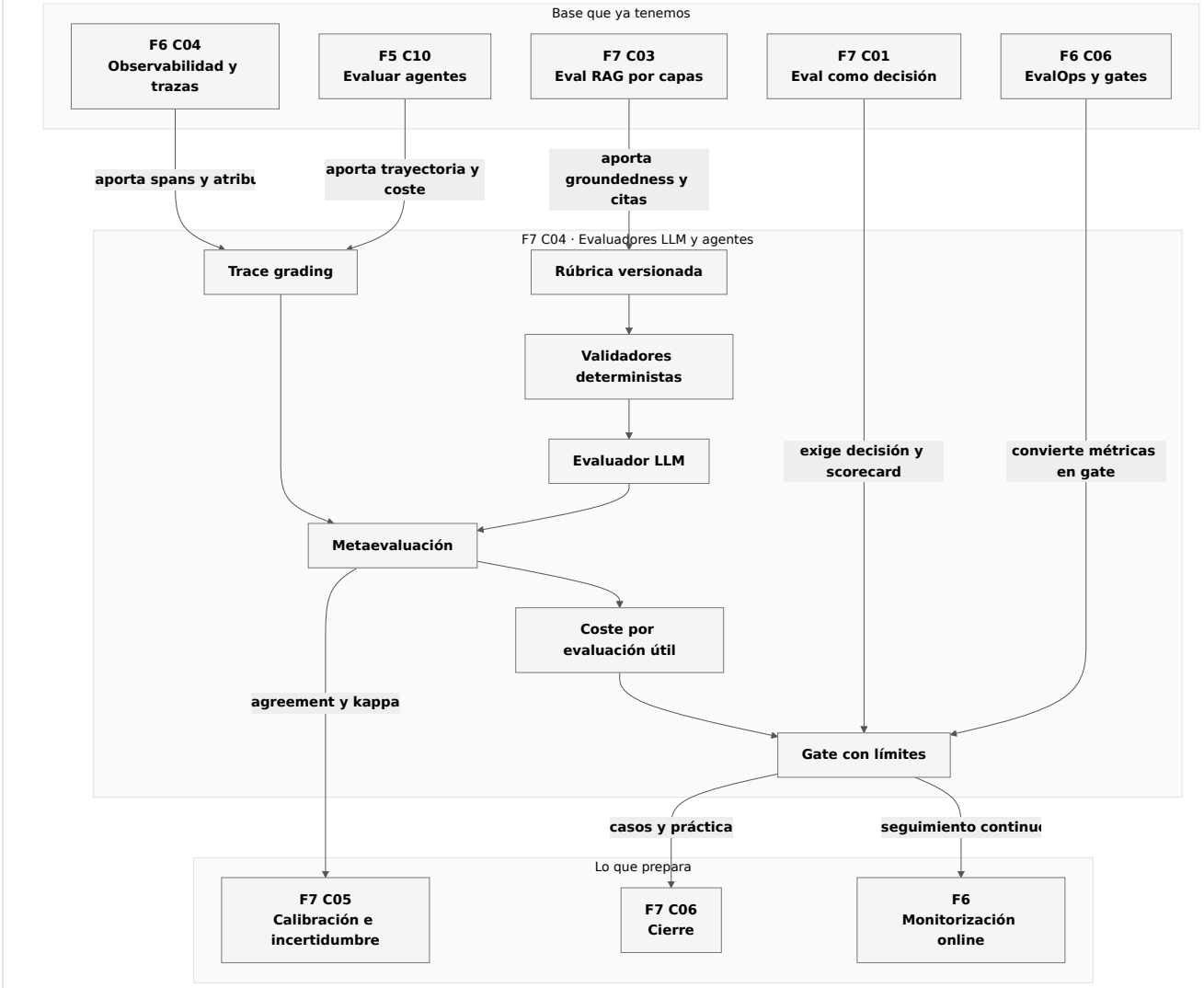
Por qué debería importarte

Porque un evaluador LLM suele aparecer justo cuando el equipo está cansado de revisar a mano. En ese momento es muy tentador convertirlo en autoridad: si el evaluador dice `pass`, pasa. Ese atajo es peligroso. El evaluador puede preferir respuestas largas, dejarse convencer por un estilo fluido, no detectar una cita débil o aprobar una trayectoria que incumple una política.

Para ingeniería, el cambio mental es claro: el evaluador es otra pieza del sistema, no una voz externa. Tiene versión, entrada, salida, coste, errores, sesgos, regresiones y dueño. Si va a formar parte de CI, de un gate de release o de una revisión preproducción, debe tener expediente propio.

DECISIÓN REAL	QUÉ APORTA ESTE CAPÍTULO
Automatizar parte de la revisión.	Te obliga a separar lo verificable por código de lo semántico.
Comparar prompts o modelos.	Te da comparadores, rúbricas y calibration set.
Evaluar agentes.	Te recuerda mirar tools, argumentos, observaciones, aprobaciones y coste.
Bajar coste de revisión humana.	Te enseña a calcular coste por evaluación útil, no coste por llamada.
Usar un evaluador como gate.	Te exige metaevaluación: kappa, falsos pases, parseo y trazas.

Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN BREVE
Evaluador LLM	Modelo que evalúa una salida o traza según una rúbrica.
Rúbrica	Criterios observables, escala, ejemplos y reglas de bloqueo.
Metaevaluación	Evaluación del propio evaluador.
Calibration set	Casos con referencia humana para calibrar el evaluador.
Kappa de Cohen	Acuerdo corregido por azar entre dos evaluadores.
Pases indebidos	Casos que el evaluador aprueba aunque la referencia humana rechaza.
Trace grading	Evaluación de la trayectoria completa de un agente.
Coste por evaluación útil	Coste de evaluar dividido entre evaluaciones válidas y aceptadas.
Criterio bloqueante	Criterio que impide aprobar aunque la media sea buena.
Drift del evaluador	Cambio de comportamiento del evaluador al cambiar modelo, prompt, tarea o rúbrica.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Usar un evaluador sin calibration set	Si no lo comparas contra referencia humana, no sabes si mide lo que necesitas.	Empezar con pocos casos, pero revisados con cuidado.
Pedir una nota global	Una nota única no dice si falló evidencia, claridad, formato, trayectoria o política.	Puntuar por criterio.
Mandarlo todo al evaluador	Validar JSON, tool calls o cálculos con un LLM es caro y frágil.	Usar código para lo verificable.
No contar pases indebidos	Accuracy alta puede esconder que el evaluador aprueba casos que no debería.	Mirar `false_passes` y criterios bloqueantes.
Olvidar el coste de evaluar	Un sistema de evaluación también puede romper presupuesto.	Medir coste por evaluación útil.
Evaluar agentes como si fueran respuestas sueltas	La frase final puede sonar bien aunque la trayectoria sea mala.	Usar trace grading.

Antes de pasar página

Antes de avanzar, deberías poder responder:

1. ¿Por qué un evaluador LLM no sustituye una referencia humana?
2. ¿Qué validarías con código antes de usar un evaluador?
3. ¿Qué debe incluir una rúbrica evaluable?
4. ¿Qué diferencia hay entre evaluador binario, ordinal y pareado?
5. ¿Qué es un pase indebido y por qué importa?
6. ¿Por qué kappa aporta más que accuracy en algunos casos?
7. ¿Qué elementos de una traza de agente debe mirar un evaluador?
8. ¿Cómo calculas coste por evaluación útil?
9. ¿Cuándo mandarías un caso a revisión humana?
10. ¿Qué archivos entrega la práctica del capítulo?

En resumen

IDEA	QUÉ TE LLEVAS
Un evaluador LLM es un instrumento, no una verdad.	Debe calibrarse contra referencias humanas o reglas fiables.
La rúbrica manda.	Sin criterios observables, el número no significa gran cosa.
Código antes que evaluador.	Lo verificable se valida con parsers, schemas, tests o cálculos.
La metaevaluación es obligatoria.	Accuracy, kappa, pases indebidos, parseo y coste dicen si el evaluador sirve.
Agentes exigen trace grading.	La trayectoria puede fallar aunque la salida final suene bien.
El coste de evaluar también se diseña.	Un evaluador útil debe caber en presupuesto y aportar señal accionable.

Para saber más

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1), 37-46. https://doi.org/10.1177/001316446002000104

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1), 1-26. https://doi.org/10.1214/aos/1176344552

Fleiss, J. L. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5), 378-382. https://doi.org/10.1037/h0031619

Goodhart, C. A. E. (1975). Problems of monetary management: The U.K. experience. *Papers in Monetary Economics*. Reserve Bank of Australia.

Krippendorff, K. (2004). Reliability in content analysis: Some common misconceptions and recommendations. *Human Communication Research*, 30(3), 411-433. https://doi.org/10.1111/j.1468-2958.2004.tb00738.x

Sokolova, M. y Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427-437. https://doi.org/10.1016/j.ipm.2009.03.002

LangChain. (2026). *How to define an LLM-as-a-judge evaluator*. https://docs.langchain.com/langsmith/llm-as-judge

Liu, X. y otros (2024). AgentBench: Evaluating LLMs as Agents. *International Conference on Learning Representations*. https://arxiv.org/abs/2308.03688

Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R. y Zhu, C. (2023). G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. *EMNLP*. https://arxiv.org/abs/2303.16634

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2), 153-157. https://doi.org/10.1007/BF02295996

OpenAI. (2026). *Graders*. https://developers.openai.com/api/docs/guides/graders

OpenAI. (2026). *Trace grading*. https://developers.openai.com/api/docs/guides/trace-grading

Ragas. (2026). *General Purpose Metrics*. https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/general_purpose/

Zheng, L. y otros (2023). Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *NeurIPS Datasets and Benchmarks*. https://arxiv.org/abs/2306.05685

Zhou, S. y otros (2023). WebArena: A Realistic Web Environment for Building Autonomous Agents. https://arxiv.org/abs/2307.13854

Notas

1. OpenAI. (2026). *Graders*. https://developers.openai.com/api/docs/guides/graders. Consultado el 31 de mayo de 2026. ↩

2. LangChain. (2026). *How to define an LLM-as-a-judge evaluator*. https://docs.langchain.com/langsmith/llm-as-judge. Consultado el 31 de mayo de 2026. ↩

3. Ragas. (2026). *General Purpose Metrics*. https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/general_purpose/. Consultado el 31 de mayo de 2026. ↩

4. Zheng, L. y otros (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. NeurIPS Datasets and Benchmarks. ↩

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

6. Liu, X. y otros (2024). *AgentBench: Evaluating LLMs as Agents*. ICLR. Zhou, S. y otros (2023). *WebArena: A Realistic Web Environment for Building Autonomous Agents*. arXiv. ↩

7. OpenAI. (2026). *Graders*. https://developers.openai.com/api/docs/guides/graders. Consultado el 31 de mayo de 2026. ↩

8. Fleiss, J. L. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5), 378-382. https://doi.org/10.1037/h0031619. Krippendorff, K. (2004). Reliability in content analysis: Some common misconceptions and recommendations. *Human Communication Research*, 30(3), 411-433. https://doi.org/10.1111/j.1468-2958.2004.tb00738.x. ↩

9. Goodhart, C. A. E. (1975). Problems of Monetary Management: The U.K. Experience. *Papers in Monetary Economics*. ↩

.0. OpenAI. (2026). *Evals deprecation*. https://developers.openai.com/api/docs/deprecations. Consultado el 21 de junio de 2026. ↩

Capítulo 05: Calibración e incertidumbre: de scores a decisiones

Entrando en el tema

En el capítulo 02 vimos que un score se convierte en acción cuando le ponemos umbrales. En el capítulo 04 vimos que un evaluador LLM también debe medirse antes de confiar en sus veredictos. Ahora falta una pregunta incómoda: **¿ese 0,82 que aparece en una métrica, un clasificador, un recuperador o un evaluador significa algo parecido a “82 %”?**

Muchas veces no. Un sistema puede ordenar bien los casos y estar mal calibrado. También puede decir “alta confianza” sin que esa confianza corresponda a una frecuencia real de acierto. Para ingeniería, esa diferencia importa muchísimo: no es lo mismo una puntuación útil para ordenar que una probabilidad útil para automatizar.

Qué deberías poder hacer al terminar

Al terminar deberías poder hacer esto:

RESULTADO DE APRENDIZAJE	EVIDENCIA DE QUE LO SABES HACER
Separar score, probabilidad y decisión.	Puedes explicar por qué un 0,9 puede ordenar bien y aun así no ser una probabilidad fiable.
Medir calibración.	Calculas Brier score, log loss, ECE y lees un reliability diagram.
Elegir una técnica de calibración.	Distingues Platt scaling, isotonic regression, histogram/binning y temperature scaling.
Separar datos sin contaminar evaluación.	Distingues train, calibration, test final y producción revisada.
Leer calibración multiclase.	Sabes cuándo mirar top-label, por clase o distribución completa.
Usar incertidumbre sin teatralizarla.	Diseñas zona de revisión, abstención o salida con intervalo cuando el sistema no tiene suficiente evidencia.
Entender conformal prediction.	Construyes conjuntos o intervalos con cobertura objetivo y sabes qué supuesto los sostiene.
Convertir calibración en política operativa.	Escribes umbrales, costes, revisión y criterios de publicación.
Conectar umbrales con capacidad humana.	Usas tasa de revisión, cola y SLO para no diseñar una política imposible.
Medir incertidumbre estadística.	Añades intervalos, bootstrap y lectura por slice antes de sacar conclusiones.
Vigilar deriva.	Identificas dataset shift, slices nuevos y triggers de recalibración.
Entregar una práctica reproducible.	Produces datos, script, reporte JSON, manifest y decisión Markdown que un equipo puede revisar.

La idea central del capítulo es sencilla: **un score no merece mandar hasta que sabes qué significa cuando se equivoca**.

El 0,92 que no significa 92 %

Imagina un sistema que prioriza tickets de soporte. Un caso llega con score 0,92 de “urgente”. Producto quiere automatizar: si supera 0,90, se marca como urgente y se salta la cola.

Antes de hacerlo, necesitamos saber qué significa ese 0,92.

LECTURA INGENUA	LECTURA DE INGENIERÍA
“El sistema está seguro al 92 %”.	“En casos parecidos con score cercano a 0,92, ¿cuántos eran realmente urgentes?”
“El score es alto, automatizamos”.	“¿Qué coste tiene equivocarnos aquí y cuántos casos de esta banda hemos medido?”
“Si ordena bien, ya sirve”.	“Ordenar bien no basta si el score alimenta umbrales, revisión o SLA.”

Esto aparece por todas partes en IA aplicada:

LUGAR DONDE APARECE UN SCORE	ERROR TÍPICO
Clasificador de tickets	Leer `0.87` como probabilidad sin medir calibración.
RAG	Confundir similitud de embedding con probabilidad de que la respuesta esté soportada.
Evaluador automático	Tratar una nota `4/5` como verdad operacional sin calibrarla contra casos revisados.
Agente con tool calls	Usar “confidence” textual del modelo como si fuera una métrica medida.
Modelo local o API	Comparar scores de proveedores distintos como si vivieran en la misma escala.

Un score puede servir para ordenar. Una probabilidad calibrada sirve para decidir bajo coste. No son la misma promesa.

Qué no es calibrar

Calibrar no es subir la accuracy. Un sistema puede tener la misma accuracy antes y después de calibrar, pero producir probabilidades más honestas.

Tampoco es “hacer que el modelo dude”. A veces calibrar baja scores exagerados; otras veces sube scores demasiado conservadores. La dirección no importa. Importa que el número signifique algo empírico.

Y calibrar tampoco sustituye a una buena evaluación. Si tu dataset no representa el uso real, si mezclas datos de ajuste con datos de evaluación o si cambias el umbral después de mirar el resultado, tendrás una apariencia de rigor, no una política fiable.

Podemos resumirlo así:

CONCEPTO	PREGUNTA QUE RESPONDE	QUÉ NO RESPONDE
Discriminación	¿Ordena positivos por encima de negativos?	Si el 0,8 significa 80 %.
Accuracy	¿Cuántos acierta con un umbral dado?	Si sus probabilidades son honestas.
Calibración	¿El score coincide con frecuencia real?	Si el modelo entiende el dominio.
Incertidumbre	¿Cuánto margen de duda queda?	Qué decisión de producto conviene tomar.
Política	¿Qué hacemos con esa duda?	Si los datos de partida eran buenos.

Qué sí es una probabilidad calibrada

Una predicción probabilística está calibrada cuando, entre los casos a los que asigna probabilidad p, la frecuencia real del evento también es p.

La expresión siguiente no es una fórmula inventada para el capítulo: es la definición estándar de calibración probabilística o fiabilidad de probabilidades. La verás escrita con esta forma, o con bandas aproximadas, en literatura de predicción probabilística y calibración de clasificadores. En el capítulo la usamos como definición ideal; en datos finitos la aproximamos con reliability diagrams y métricas por bandas.

$$\mathbb{P}(Y = 1 \mid \hat{p}(X) = p) = p$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
X	Entrada del sistema.	Texto de un ticket.
Y	Etiqueta real.	1 si el ticket era urgente.
$\hat{p}(X)$	Probabilidad predicha por el sistema.	0,80.
p	Banda o valor de confianza.	Casos alrededor de 0,80.
$\mathbb{P}(Y = 1 \mid \hat{p}(X) = p)$	Frecuencia real de positivos dentro de esa banda.	0,78 en la muestra.

En palabras: si el sistema asigna 0,80 a muchos casos parecidos, alrededor del 80 % de esos casos deberían ser positivos. Si no ocurre, el número puede ordenar, pero no se puede leer como probabilidad honesta.

En la práctica no tenemos infinitos casos con exactamente $p = 0,80$. Agrupamos predicciones en bandas: 0,0-0,1; 0,1-0,2; 0,2-0,3; y así sucesivamente. Si en la banda 0,8-0,9 el score medio es 0,84 y el 62 % de los casos son realmente positivos, el modelo está sobreconfiado en esa banda.

La calibración es local. Un promedio bonito puede esconder bandas malas. Por eso miramos la curva completa.

Fecha de corte del estado del arte

Fecha de corte: 1 de junio de 2026.
Fuentes consultadas: trabajos clásicos sobre probabilidades calibradas, Brier score, reliability diagrams, calibración supervisada, calibración de redes neuronales modernas y conformal prediction; además de documentación de scikit-learn, trabajos sobre incertidumbre en LLMs, documentación técnica de modelos/datos y guías de producción de sistemas ML.

Brier propuso en 1950 una puntuación para predicciones probabilísticas que mide el error cuadrático entre probabilidad y resultado observado.[1] Murphy descompuso después esa puntuación en componentes relacionados con fiabilidad, resolución e incertidumbre.[2]

Niculescu-Mizil y Caruana mostraron que clasificadores distintos producen probabilidades con comportamientos de calibración muy distintos, aunque su capacidad de ranking sea buena.[3] Platt scaling popularizó una calibración sigmoidal para salidas de SVM.[4] Guo y otros mostraron que redes neuronales modernas pueden estar muy bien en accuracy y mal calibradas, y que temperature scaling puede corregir parte de esa sobreconfianza con un ajuste simple.[5]

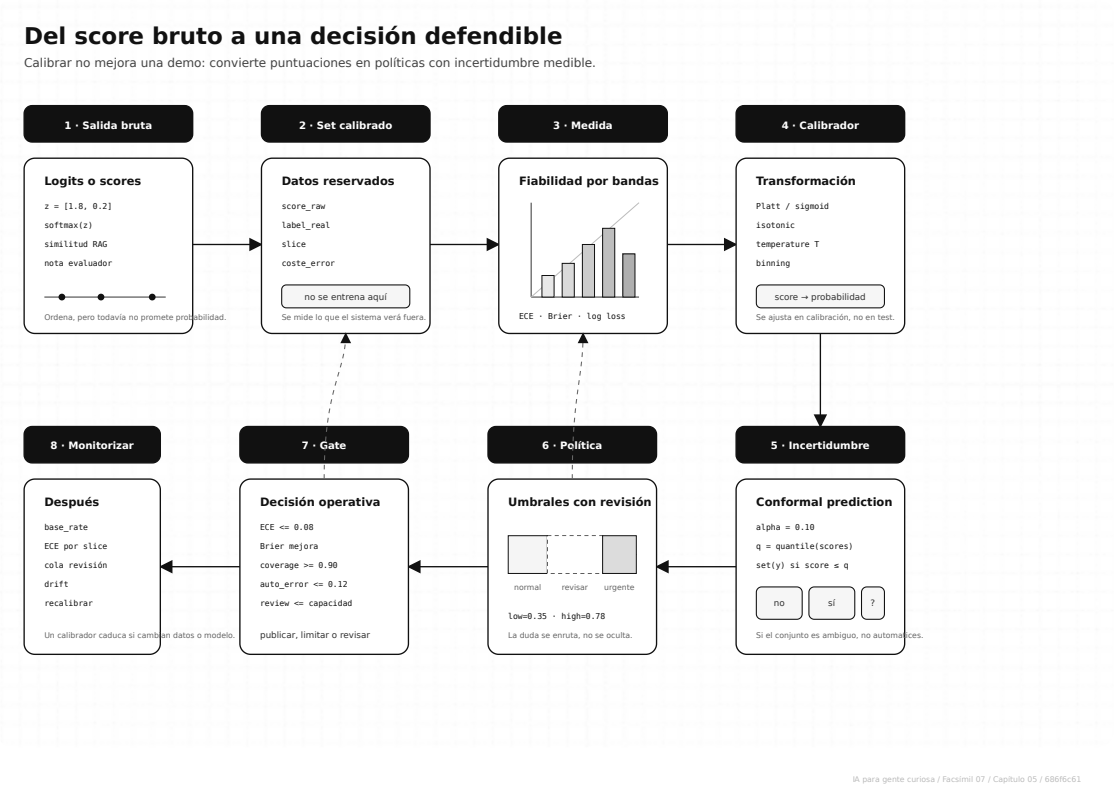
Para cuantificar incertidumbre con garantías finitas, la familia de conformal prediction viene de los trabajos de Vovk, Gammerman y Shafer.[6] Shafer y Vovk ofrecen una introducción tutorial al enfoque.[7] Angelopoulos y Bates escribieron una introducción moderna a conformal prediction y cuantificación de incertidumbre libre de distribución.[8]

La documentación de scikit-learn resume la diferencia práctica entre calibración, curvas de fiabilidad y métodos como sigmoid e isotonic calibration.[9]

Para llevar esto a ingeniería de IA moderna, también nos apoyamos en trabajos sobre incertidumbre en modelos de lenguaje, documentación de modelos y datos, y preparación para producción.

PIEZA DE INGENIERÍA	FUENTE USADA
Incertidumbre en modelos de lenguaje	Kadavath y otros estudian cuándo los modelos de lenguaje pueden reconocer límites de conocimiento bajo determinados protocolos.[10]
Incertidumbre semántica	Kuhn, Gal y Farquhar proponen agrupar respuestas que dicen lo mismo aunque usen palabras distintas.[11]
Documentación de modelos	Model Cards propone reportar usos previstos, límites y resultados por segmentos.[12]
Documentación de datos	Data Cards estructura información sobre origen, composición, límites y uso previsto de datasets.[13]
Preparación para producción	Sculley y otros describen deuda técnica propia de sistemas ML.[14]
Readiness de ML	El ML Test Score propone pruebas y necesidades de monitorización para sistemas ML en producción.[15]
SLI/SLO	La guía SRE de Google separa indicador, objetivo y presupuesto de error para decidir con datos.[16]

Anatomía de una política calibrada



Una política calibrada conecta score bruto, datos reservados, medición, calibrador, incertidumbre, umbrales, gate y monitorización.

Medir calibración: Brier, log loss y ECE

Hay tres medidas que conviene tener cerca. No dicen exactamente lo mismo, y esa diferencia es útil.

Brier score

Brier score mide el error cuadrático entre la probabilidad predicha y la etiqueta real:

La fórmula viene del Brier score de Brier (1950), una regla de puntuación probabilística para predicciones expresadas como probabilidades. En binario se escribe como error cuadrático entre probabilidad y resultado observado.

$$BS = \frac{1}{N}\sum_{i=1}^{N}(\hat{p}_i - y_i)^2$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
BS	Brier score; cuanto menor, mejor.	0,142.
N	Número de casos.	200 tickets.
$\hat{p}_i$	Probabilidad predicha para el caso i.	0,80.
y_i	Etiqueta real del caso i: 0 o 1.	1 si era urgente.

En palabras: Brier mide cuánto se separa cada probabilidad de lo que ocurrió. Si das mucha probabilidad a algo que no pasa, pagas mucho; si das una probabilidad cercana al resultado real, pagas poco.

Si predices 0,80 y el caso era positivo, aportas $(0,80-1)^2 = 0,04$. Si predices 0,80 y el caso era negativo, aportas $(0,80-0)^2 = 0,64$. El Brier castiga tanto mala calibración como mala discriminación, pero es fácil de explicar.

Log loss

Log loss penaliza de forma dura estar muy seguro y equivocarte:

Esta fórmula es la pérdida logarítmica binaria, equivalente a la negative log-likelihood de una variable Bernoulli. En la literatura de scoring rules aparece dentro de las reglas de puntuación propias: si quieres incentivar probabilidades honestas, no basta con contar aciertos; debes puntuar la distribución probabilística. Gneiting y Raftery (2007) revisan esta familia de reglas de puntuación propias.

$$LL = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)\right]$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
LL	Log loss; cuanto menor, mejor.	0,41.
$\log$	Logaritmo natural.	$\log(0,9)$.
$\hat{p}_i$	Probabilidad predicha, recortada para evitar 0 o 1 exactos.	0,97.
y_i	Etiqueta real.	0.

En palabras: log loss castiga la mala sorpresa. Equivocarse con duda moderada duele; equivocarse diciendo casi 1 o casi 0 duele muchísimo más.

Si un sistema dice 0,99 y falla, log loss lo deja en evidencia. Eso es sano cuando una decisión automática usa el score como confianza.

ECE

Expected Calibration Error agrupa predicciones por bandas y compara confianza media con accuracy real:

La ECE no tiene la misma solidez histórica que Brier o log loss: es una métrica práctica por bandas usada mucho en calibración moderna. La asociamos a trabajos como Naeini, Cooper y Hauskrecht (2015), y aparece después en Guo y otros (2017) para analizar calibración de redes neuronales modernas. Por eso la tratamos como señal útil, no como veredicto suficiente.

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{N} \left|\mathrm{acc}(B_m) - \mathrm{conf}(B_m)\right|$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
M	Número de bandas.	10 bandas.
B_m	Casos que caen en la banda m.	Scores entre 0,8 y 0,9.
$	B_m	$
$\mathrm{acc}(B_m)$	Frecuencia real de acierto en esa banda.	0,65.
$\mathrm{conf}(B_m)$	Confianza media predicha en esa banda.	0,84.

En palabras: ECE mira banda a banda cuánto se separa lo que el sistema decía de lo que ocurrió. Una banda con muchos casos pesa más que una banda pequeña.

ECE es intuitivo, pero depende de cómo elijas bandas. Por eso no lo usaría solo. Lo pondría junto a Brier, log loss, reliability diagram y análisis por slice.

Reliability diagram: mirar la curva, no solo el número

Un reliability diagram pone en el eje X la confianza media de cada banda y en el eje Y la frecuencia real de acierto. La diagonal perfecta significa calibración ideal.

PATRÓN VISUAL	LECTURA
Curva por debajo de la diagonal	El sistema está sobreconfiado: promete más de lo que cumple.
Curva por encima de la diagonal	El sistema es conservador: acierta más de lo que su score dice.
Bien en scores bajos, mal en scores altos	Peligroso si automatizas por umbral alto.
Bien en promedio, mal en un slice	No publiques sin política específica para ese slice.

Para proyectos de IA, el reliability diagram debe mirarse por familias de caso: idioma, dominio, fuente documental, tipo de usuario, longitud de entrada, modelo usado, recuperador usado, herramienta invocada o evaluador aplicado.

Una calibración global puede esconder que el sistema es honesto en tickets simples y sobreconfiado en documentos largos.

Métodos de calibración que sí se usan

Calibrar significa aprender una transformación que convierte scores brutos en probabilidades más fiables. Esa transformación debe ajustarse en un conjunto de calibración reservado, no en el test final.

MÉTODO	IDEA	CUÁNDO ENCAJA	CUIDADO
Platt scaling	Ajusta una sigmoide sobre el score bruto.	Clasificadores binarios con forma de error suave.	Puede quedarse corto si la curva real no es sigmoidal.
Isotonic regression	Aprende una función monótona por tramos.	Tienes suficientes datos de calibración.	Puede sobreajustar con pocos casos.
Histogram/ binning	Agrupa scores y sustituye por frecuencia observada.	Necesitas algo explicable y auditable.	Bandas con pocos casos son ruidosas.
Temperatur e scaling	Divide logits por una temperatura T antes de softmax.	Redes neuronales y clasificación multicategoría.	No cambia el ranking; solo suaviza o endurece confianza.
Vector/matr ix scaling	Ajustes más flexibles sobre logits multicategoría.	Multiclase con datos suficientes.	Más parámetros, más riesgo de ajuste a calibración.
Calibración por slice	Calibradores separados o correcciones por segmento.	Los segmentos se comportan de forma distinta.	Necesita volumen y control de deriva.

Temperature scaling se escribe así:

Esta fórmula es la softmax aplicada a logits después de dividirlos por una temperatura T. En este capítulo aparece por Guo y otros (2017), que muestran temperature scaling como calibración posentrenamiento sencilla para redes neuronales: ajusta confianza sin cambiar el ranking de clases.

$$\hat{p}_k = \frac{\exp(z_k / T)}{\sum_{j=1}^{K} \exp(z_j / T)}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
z_k	Logit bruto de la clase k.	3,2 para `urgente`.
T	Temperatura aprendida en calibración.	1,7.
K	Número de clases.	3: normal, revisar, urgente.
$\hat{p}_k$	Probabilidad calibrada de la clase k.	0,74.

En palabras: la temperatura no cambia cuál era la clase más alta; cambia cuánto se concentra o se reparte la confianza entre clases.

Si $T > 1$, la distribución se suaviza: menos seguridad extrema. Si $T < 1$, se endurece: más masa en la clase dominante. Guo y otros mostraron que, para muchas redes modernas, una temperatura aprendida podía mejorar mucho la calibración sin cambiar la clase predicha.[17]

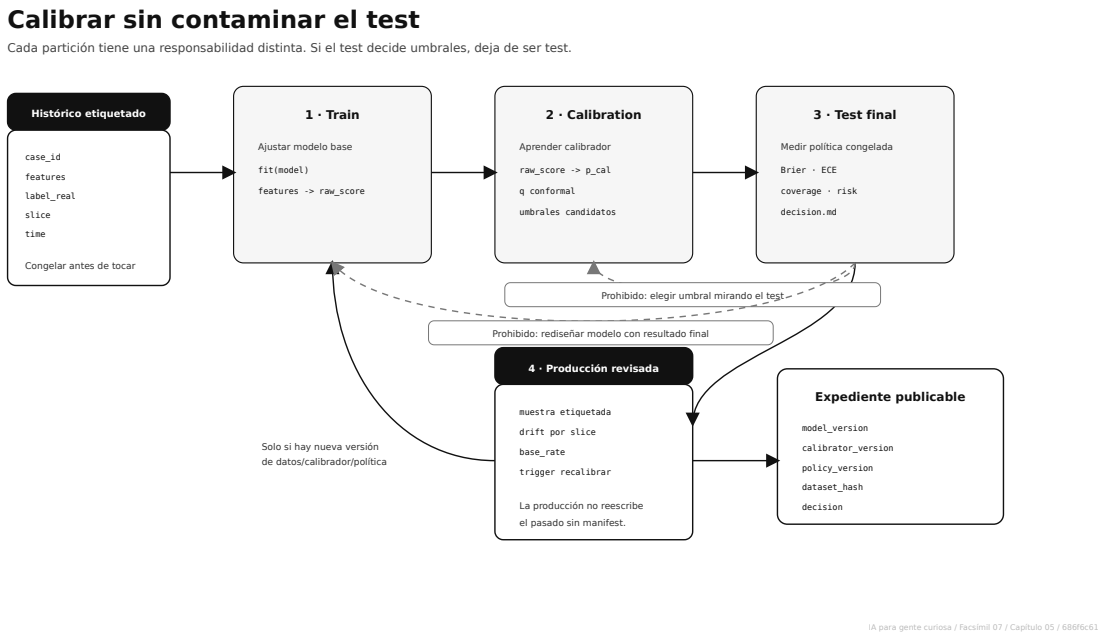
Partición de datos: calibrar no es mirar el examen final

El error más peligroso en calibración no es elegir Platt, isotonic o temperature scaling. Es usar mal los datos. Si entrenas el modelo, ajustas el calibrador, eliges umbrales y después dices que has validado todo con el mismo conjunto, no has medido una política: la has ido moldeando hasta que encaja con el examen.

La partición mínima debería separar cuatro papeles:

PARTE	QUÉ SE HACE AHÍ	QUÉ NO DEBERÍA HACERSE
Entrenamiento	Ajustar pesos, árbol, prompt entrenado, clasificador o modelo base.	Medir el resultado final de publicación.
Calibración	Aprender el calibrador y, si procede, estimar cuantiles conformales.	Cambiar la arquitectura principal mirando el test.
Evaluación final	Medir la política completa ya congelada.	Elegir umbrales después de ver el resultado.
Producción revisada	Vigilar deriva, base rate y errores reales.	Recalibrar en caliente sin control de versiones.

En scikit-learn, `CalibratedClassifierCV` existe precisamente porque calibrar con predicciones sesgadas por el entrenamiento puede engañarte: la documentación separa el ajuste del estimador base y el ajuste del calibrador mediante particiones de validación cruzada.[18] MAPIE usa el término `conformalization` para el conjunto donde se estiman scores de conformidad antes de producir intervalos o conjuntos predictivos.[19]



El split separa entrenamiento, calibración, evaluación final y producción revisada. El test no se usa para elegir umbrales.

En proyectos pequeños no siempre hay datos para particiones perfectas. Aun así, la intención debe quedar escrita. Si hay pocos casos, prefiero una política conservadora con revisión humana y una muestra nueva cada semana antes que una calibración que presume de precisión con doce ejemplos.

Calibración multiclase: cuando no basta mirar la clase ganadora

Hasta aquí el ejemplo ha sido binario porque ayuda a pensar. En producción aparecen tareas multicategoría: tipo de ticket, intención del usuario, ruta del agente, clase documental, nivel de severidad o decisión de negocio. Ahí la calibración se vuelve más delicada.

Una trampa común es mirar solo la confianza de la clase ganadora. Si el modelo dice `beca` con 0,82, parece razonable preguntar si los casos con top-label 0,82 aciertan alrededor del 82 %. Eso es útil, pero no agota el problema: una clase minoritaria puede estar muy mal calibrada aunque el promedio global parezca decente. Gupta y Ramdas estudian precisamente calibración top-label y reducciones multiclase-a-binario para razonar sobre esta diferencia.[20]

Para un sistema de IA, yo miraría tres niveles:

NIVEL	QUÉ PREGUNTA RESPONDE	EJEMPLO DE FALLO
Confianza top-label	“Cuando el modelo dice 0,8 en la clase elegida, ¿acierta cerca del 80 %?”	Parece bien en global, pero falla una intención rara.
Calibración por clase	“Para `becas`, `matrícula` o `producción`, ¿el score significa lo mismo?”	`becas` está sobreconfiado porque hay pocos ejemplos recientes.
Distribución completa	“¿La probabilidad repartida entre clases alternativas tiene sentido?”	El modelo pone 0,78 en `normal` y 0,20 en `urgente`, pero ambas entran en un conjunto conformal.

Esto cambia cómo decides. Si solo necesitas enrutar tickets y siempre habrá revisión, top-label puede bastar al principio. Si una clase dispara una acción cara o irreversible, necesitas medir esa clase como producto propio. Un “buen ECE global” no compensa automatizar mal el 4 % de casos que más daño hacen.

En LLMs y agentes, esta misma idea aparece con otros nombres. Un router de modelos puede elegir `modelo barato`, `modelo grande` o `revisión humana`. Un agente puede elegir `buscar`, `calcular`, `escribir` o `pedir permiso`. La calibración que importa no es una media estética: es si cada ruta tiene una probabilidad defendible de salir bien.

Incertidumbre: no todo tiene que salir como sí o no

En ingeniería, la incertidumbre no es una disculpa. Es una señal de control.

SEÑAL DE INCERTIDUMBRE	ACCIÓN RAZONABLE
Score cerca del umbral.	Mandar a revisión o pedir más evidencia.
Reliability diagram malo en esa banda.	No automatizar esa banda.
Conformal set con dos clases posibles.	No elegir una sola clase en automático.
RAG con citas débiles.	Abstenerse o responder con alcance limitado.
Evaluador automático con desacuerdo alto.	Revisar muestra humana y recalibrar.
Drift de base rate.	Recalcular calibración antes de mantener umbrales.

La salida profesional no siempre es “sí” o “no”. A veces es:

```
{
  "decision": "revisar",
  "reason": "score calibrado dentro de zona gris",
  "calibrated_probability": 0.61,
  "conformal_set": ["normal", "urgente"],
  "next_step": "enviar a cola de soporte nivel 2"
}
```

Esto no es menos inteligente. Es más honesto.

Conformal prediction: garantías con supuestos claros

Conformal prediction no intenta adivinar si el modelo “está seguro” en abstracto. Construye conjuntos o intervalos que contienen la respuesta correcta con una cobertura objetivo, bajo un supuesto clave: los datos de calibración y los datos futuros son intercambiables, es decir, vienen del mismo mecanismo de generación en el sentido estadístico que necesitamos para el problema.

En clasificación, una forma sencilla es usar como score de no conformidad:

Las fórmulas de esta sección vienen de conformal prediction. La familia clásica está desarrollada por Vovk, Gammerman y Shafer, y la exposición moderna de Angelopoulos y Bates ayuda a leerla como cuantificación de incertidumbre con cobertura finita bajo intercambiabilidad. Aquí usamos una versión sencilla para clasificación: score de no conformidad, cuantil y conjunto predictivo.

$$a_i = 1 - \hat{p}_{y_i}(x_i)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
a_i	Score de no conformidad del caso i.	0,18.
$\hat{p}_{y_i}(x_i)$	Probabilidad asignada a la clase correcta del caso i.	0,82.
x_i	Entrada del caso i.	Ticket de soporte.
y_i	Clase real del caso i.	`urgente` .

En palabras: un caso es poco conforme si el modelo dio poca probabilidad a la clase que luego resultó ser la correcta.

Después elegimos un cuantil de esos scores en el conjunto de calibración:

Este cuantil es la pieza central del split conformal: se calcula sobre los scores de no conformidad de calibración y se elige de forma conservadora para sostener la cobertura finita descrita por Vovk, Gammerman y Shafer, y por la formulación moderna de Angelopoulos y Bates.

$$q = \text{Quantile}_{\lceil (n+1)(1-\alpha) \rceil / n} \left(a_1, \ldots, a_n \right)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
q	Umbral conformal de no conformidad.	0,42.
n	Número de casos de calibración.	100.
α	Tasa de error permitida.	0,10 para cobertura 90 %.
$\lceil \cdot \rceil$	Redondeo hacia arriba.	Garantiza una elección conservadora.

En palabras: elegimos un umbral suficientemente alto para cubrir la fracción prometida de casos bajo el supuesto de intercambiabilidad.

Para un caso nuevo, incluimos cada clase cuyo score de no conformidad no supera q:

El conjunto predictivo $\Gamma_\alpha(x)$ es la salida propia de la predicción conformal en clasificación: no fuerza una única clase si varias siguen siendo plausibles bajo el umbral de no conformidad aprendido.

$$\Gamma_\alpha(x) = \{ y : 1 - \hat{p}_y(x) \leq q \}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\Gamma_\alpha(x)$	Conjunto de clases plausibles para x.	`{normal, urgente}`.
y	Clase candidata.	`urgente`.
$\hat{p}_y(x)$	Probabilidad de esa clase para el caso nuevo.	0,63.
q	Umbral aprendido en calibración.	0,42.

En palabras: una clase entra en el conjunto si no parece demasiado rara comparada con los errores vistos en calibración.

Si el conjunto tiene una sola clase, quizá podemos automatizar. Si tiene dos o más, el sistema está diciendo: “con la cobertura que me pediste, no puedo elegir solo una sin perder garantía”. Esa frase vale oro en un producto real.

En regresión, la versión más sencilla usa residuos absolutos:

La forma de regresión conformal más básica usa residuos absolutos en calibración. No inventa una incertidumbre nueva: convierte los errores observados del modelo en un margen empírico.

$$a_i = |y_i - \hat{f}(x_i)|$$

Y construye un intervalo:

El intervalo conformal simétrico alrededor de $\hat{f}(x)$ es la traducción directa de ese margen q a una predicción numérica. Es sencillo, auditable y útil como punto de partida antes de variantes adaptativas.

$$C_\alpha(x) = \left[\hat{f}(x) - q, \hat{f}(x) + q\right]$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\hat{f}(x)$	Predicción numérica del modelo.	18 minutos de espera.
q	Cuantil de residuos en calibración.	6 minutos.
$C_\alpha(x)$	Intervalo conformal.	[12, 24] minutos.

En palabras: en regresión, miramos cuánto se equivocó el modelo en calibración y usamos ese margen para construir un intervalo alrededor de la nueva predicción.

La parte honesta: conformal prediction no arregla un dataset que ya no representa producción. Si cambia el canal de entrada, el idioma, el modelo base, la política de producto o el tipo de caso, recalibramos.

Conformal en proyectos reales: cobertura no es utilidad

La promesa de conformal prediction es potente: bajo supuestos claros, puedes construir conjuntos o intervalos con cobertura. Pero un sistema de ingeniería no vive solo de cobertura. Vive de utilidad. Un conjunto que contiene siempre cinco clases puede cubrir mucho y decidir poco.

Por eso hay que mirar dos cosas a la vez:

MEDIDA	QUÉ TE DICE	CUÁNDO PREOCUPA
Cobertura empírica	Si el conjunto contiene la respuesta real con la frecuencia prometida.	Si cae por debajo de la cobertura objetivo en evaluación o producción revisada.
Tamaño del conjunto	Cuántas respuestas plausibles deja abiertas.	Si casi todo acaba en conjuntos grandes y el sistema no decide nada.
Cobertura por slice	Si la garantía se sostiene por segmento.	Si `general` cubre bien y `becas` no.
Tasa de abstención	Cuánto trabajo manda a revisión.	Si supera la capacidad humana o hace inútil el producto.

En la práctica, hay variantes que conviene conocer:

VARIANTE	QUÉ INTENTA RESOLVER	CUÁNDO LA MIRARÍA
Split conformal	Separar entrenamiento y calibración para obtener un umbral sencillo.	Primer sistema defendible, fácil de auditar.
Cross-conformal	Usar particiones cruzadas para aprovechar mejor pocos datos.	Dataset pequeño donde separar demasiado duele.
Mondrian o por grupos	Buscar cobertura separada por categorías o slices.	Segmentos con comportamiento distinto: idioma, canal, producto, severidad.
Conformal para regresión	Crear intervalos alrededor de predicciones numéricas.	Tiempos, costes, demanda, riesgo continuo.
Conformal para clasificación	Crear conjuntos de clases plausibles.	Intenciones, rutas, etiquetas, severidades.

MAPIE es una librería Python orientada a cuantificación de incertidumbre y control de riesgo con métodos conformal para regresión, clasificación y series temporales.[21] No hace magia. Te ayuda a implementar patrones conocidos, pero sigues teniendo que decidir qué datos representan producción, qué cobertura necesitas y qué haces cuando el conjunto sale grande.

Un criterio útil para un equipo:

SI OCURRE ESTO	DECISIÓN SENSATA
Cobertura buena y conjuntos pequeños.	Automatizar con monitorización.
Cobertura buena y conjuntos grandes.	El modelo sabe cubrirse, pero no separa suficiente; revisar features, RAG, modelo o tarea.
Cobertura mala en un slice.	No automatizar ese slice y reunir datos.
Cobertura buena en test y mala en producción.	Sospechar dataset shift o cambio de proceso.

La frase que quiero que te lleves: **conformal prediction no sustituye al producto; le da una forma honesta de decir “aquí no sé decidir solo”**.

De probabilidad a decisión: coste, revisión y cobertura

Una probabilidad calibrada no decide sola. Necesita una política.

Para un caso binario, podemos comparar coste esperado de automatizar frente a revisar.

No lo escribo como fórmula porque en este contexto es una política operativa. La idea sí es de decisión bajo coste: si automatizar mal cuesta mucho, el umbral de automatización debe ser más exigente.

PREGUNTA	QUÉ ESTIMAS	EJEMPLO
¿Qué probabilidad calibrada tiene el caso?	Confianza de que la acción automática sea correcta.	0,93.
¿Qué cuesta automatizar mal?	Daño económico, legal, operativo o reputacional.	6 unidades.
¿Qué cuesta revisar?	Tiempo humano, cola, espera y coste de oportunidad.	1,20 unidades.
¿Qué restricciones no son negociables?	Cumplimiento, capacidad, severidad y calidad por slice.	Casos de privacidad no se automatizan.

Automatizar tiene sentido cuando el coste esperado de automatizar mal queda por debajo del coste de revisar, siempre que no viole restricciones de producto, cumplimiento, capacidad o calidad por slice.

En sistemas con zona gris, usamos dos umbrales.

Esta regla convierte una probabilidad calibrada en tres acciones: automatizar negativo, revisar o automatizar positivo. En producción habría que añadir restricciones por slice, capacidad de cola y severidad del caso.

ZONA	CONDICIÓN PRÁCTICA	ACCIÓN
Baja probabilidad	La probabilidad calibrada queda por debajo del umbral bajo.	Automatizar como caso normal si la política lo permite.
Zona gris	La probabilidad queda entre umbral bajo y umbral alto.	Revisar; el sistema no decide solo.
Alta probabilidad	La probabilidad queda por encima del umbral alto.	Automatizar como urgente si no hay restricción de severidad o cumplimiento.

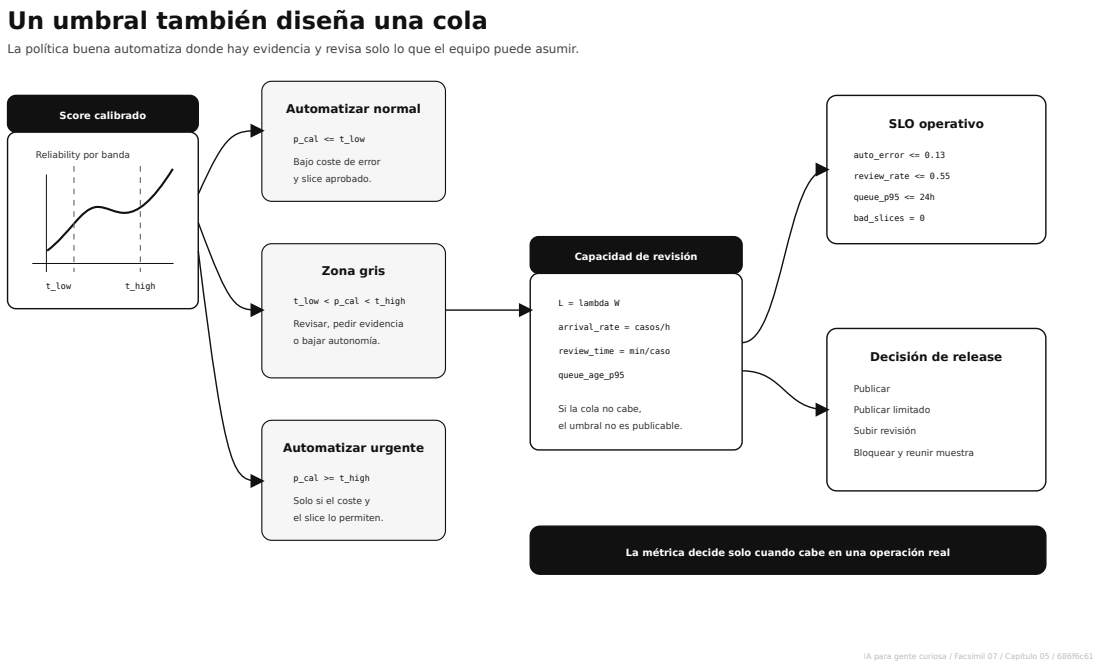
Pero falta una pregunta muy de producción: **¿quién revisa la zona gris?** Si un umbral técnicamente bonito manda 3.000 casos diarios a revisión y el equipo puede revisar 400, ese umbral no es una política. Es una deuda operativa.

Para razonar sobre colas, una ley clásica es la ley de Little. Little demostró en 1961 la relación $L = \lambda W$ para sistemas de colas en régimen estable.[22] No es una fórmula de IA, pero sí es muy útil para no diseñar una zona de revisión imposible.

$$L = \lambda W$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
L	Número medio de casos en el sistema o cola.	180 tickets esperando revisión.
λ	Tasa media de llegada efectiva.	60 tickets por hora entran a revisión.
W	Tiempo medio que un caso pasa en el sistema.	3 horas hasta resolverse.

En palabras: si entran muchos casos a revisión y cada uno tarda mucho, la cola crece. Por eso calibrar umbrales también exige mirar capacidad humana, no solo métricas del modelo.



> La zona gris conecta score calibrado, umbrales, revisión humana, SLO y decisión de release. Si la cola no cabe, la política no está lista.

La política correcta no es la que más automatiza. Es la que automatiza donde tiene evidencia suficiente y deja trazas para mejorar.

En el día a día

Si trabajas con sistemas de IA, calibración aparece de maneras menos limpias que en el ejemplo de manual.

SISTEMA	QUÉ CALIBRARÍA	QUÉ MEDIRÍA
Clasificador de soporte	Score de prioridad.	Brier, ECE, error automático, cola de revisión.
RAG documental	Probabilidad de que la respuesta esté soportada.	Groundedness por banda, abstención correcta, evidencia faltante.
Evaluador automático	Veredicto `pass/fail` o nota por rúbrica.	Acuerdo con referencia humana, ECE por criterio, pases indebidos.
Router de modelos	Probabilidad de que un modelo barato baste.	Calidad por ruta, coste por aceptada, fallback rate.
Agente con tools	Probabilidad de éxito sin revisión.	Error por trayectoria, permisos, latencia, coste y cobertura.

Un patrón útil es guardar siempre estos campos por caso:

```
{
  "case_id": "ticket_1842",
  "raw_score": 0.91,
  "calibrated_probability": 0.76,
  "decision": "revisar",
  "conformal_set": ["normal", "urgente"],
  "slice": "becas",
  "model_version": "support-prioritizer-2026-06-01",
  "calibrator_version": "histogram-v1",
  "policy_version": "support-thresholds-v3"
}
```

Sin esos campos, luego nadie sabe si falló el modelo, el calibrador, el umbral, el slice, el evaluador o la política.

Por qué debería importarte

Hasta aquí parece que calibrar consiste en tomar un score y ajustarlo. En sistemas de IA reales, el problema suele ser más amplio: primero hay que decidir **qué score** merece ser calibrado.

CASO	SCORE TENTADOR	POR QUÉ NO BASTA	QUÉ CALIBRARÍA DE VERDAD
LLM con logprobs	Probabilidad media de tokens.	Una respuesta larga acumula logprob distinto a una corta; token probable no implica respuesta correcta.	Resultado de tarea: respuesta aceptada, cita correcta, formato válido, acción correcta.
RAG	Similitud de embedding o score del reranker.	Similaridad no es soporte factual.	Probabilidad de groundedness o de respuesta soportada por evidencia.
Router de modelos	Score de “modelo barato basta”.	El coste bajo puede esconder más fallback o más revisión.	Probabilidad de salida aceptada sin fallback y coste por aceptada.
Evaluador automático	Nota de rúbrica.	La nota puede estar desplazada por estilo, longitud o versión del modelo.	Acuerdo con referencia humana por criterio y tasa de pases indebidos.
Agente con herramientas	“Éxito” declarado por el agente.	La salida final puede sonar bien aunque la trayectoria falle.	Éxito de run: herramienta correcta, permisos, evidencia, coste y finalización limpia.

Para LLMs, conviene escribir una regla brutalmente clara:

> No calibres la sensación verbal de confianza. Calibra un evento observable.

Ejemplos de eventos observables:

EVENTO CALIBRABLE	ETIQUETA REAL
“La respuesta contiene JSON válido y completo”.	1 si el parser y el contrato pasan.
“La respuesta está soportada por las citas”.	1 si una revisión o validador de evidencia lo confirma.
“El modelo barato basta para este caso”.	1 si pasa la misma eval que el modelo de referencia.
“El agente puede actuar sin revisión”.	1 si la run cumple trayectoria, permisos y resultado.

Esto evita una trampa muy común: convertir una frase como “estoy bastante seguro” en una métrica. Esa frase puede servir para UX, pero no para gates.

LLMs reales: calibrar respuestas, no frases bonitas

En clasificación clásica, el modelo suele devolver un score por clase. En LLMs generativos, la cosa se complica: el modelo produce texto token a token, puede dar varias respuestas distintas que significan lo mismo y puede sonar seguro aunque la evidencia sea débil.

Por eso, para ingeniería, hay que separar tres niveles:

NIVEL	QUÉ MIDE	POR QUÉ IMPORTA
Token	Probabilidad del siguiente token.	Sirve para entender generación, pero no prueba que la respuesta completa sea correcta.
Secuencia	Probabilidad agregada de una salida concreta.	Penaliza longitud y redacción; dos respuestas equivalentes pueden tener probabilidades diferentes.
Evento de tarea	Si la respuesta cumple el contrato.	Es lo que realmente decide producto, operación o evaluación.

Un ejemplo: ante una pregunta documental, el modelo puede responder:

RESPUESTA	TOKENS DISTINTOS	MISMO SIGNIFICADO
“La matrícula cierra el 15 de julio.”	Sí	Sí
“El plazo termina el 15/07.”	Sí	Sí
“La fecha límite es el quince de julio.”	Sí	Sí

Si miramos solo tokens, tratamos esas salidas como objetos diferentes. Si miramos la tarea, las tres dicen lo mismo. Ahí entra la incertidumbre semántica: no preguntamos solo “¿qué tan probable era esta frase exacta?”, sino “¿cuánta dispersión hay entre significados posibles?”.

Una práctica seria con LLMs suele medir al menos esto:

SEÑAL	CÓMO SE MIDE	QUÉ DECISIÓN PERMITE
Validez de formato	Parser, JSON Schema, Pydantic o contrato equivalente.	Reintentar, reparar o rechazar salida.
Soporte documental	Comparación con citas, spans o evidencia recuperada.	Responder, pedir más contexto o revisar.
Consistencia semántica	Varias muestras agrupadas por significado.	Detectar preguntas ambiguas o información insuficiente.
Acuerdo con referencia	Evaluación humana o dataset revisado.	Calibrar score de aceptabilidad.
Coste de fallback	Tokens, latencia y llamadas adicionales.	Decidir cuándo usar modelo grande o revisión.

El punto de ingeniería es incómodo pero liberador: **la incertidumbre útil no vive en una frase de confianza; vive en una variable que puedes medir contra realidad**.

Rigor estadístico mínimo: no publiques un ECE desnudo

ECE es útil, pero no es garantía suficiente. Depende del número de bandas, del tamaño de muestra y de cómo se distribuyen los casos. Con veinte ejemplos puedes fabricar una tabla que parece precisa y, en realidad, solo tiene ruido.

Para no engañarnos, cada política de calibración debería traer tres capas de incertidumbre:

CAPA	QUÉ AÑADE	QUÉ EVITA
Conteo por banda	Cuántos casos sostienen cada punto del reliability diagram.	Concluir demasiado con una banda de 2 casos.
Intervalo de proporción	Rango plausible de accuracy real por banda o slice.	Leer 0,75 como si fuera exacto.
Bootstrap de métricas	Variabilidad aproximada de Brier o ECE al remuestrear.	Comparar mejoras microscópicas como si fueran sólidas.

Para una proporción, un intervalo Wilson es más informativo que enseñar solo el punto medio. Viene del trabajo de Wilson sobre inferencia de proporciones y evita parte de los problemas de intervalos ingenuos con muestras pequeñas.[23] Si en una banda hay 6 aciertos de 8 casos, la accuracy observada es 0,75, pero el intervalo es amplio. No es lo mismo decir “esta banda acierta el 75 %” que decir “he observado 6 de 8; todavía necesito más muestra”.

El bootstrap usa una idea práctica: tomar muchas muestras con reemplazo del conjunto de evaluación, recalcular la métrica y mirar su distribución.[24] No convierte un dataset malo en bueno, pero obliga a ver si una mejora tiene cuerpo o es una fluctuación.

Una revisión profesional debería bloquear o limitar despliegue cuando vea cualquiera de estas señales:

SEÑAL	LECTURA
Bandas con muy pocos casos	El reliability diagram no sostiene decisiones finas.
ECE global baja y slice malo	La media esconde un segmento problemático.
Mejora de Brier menor que su variabilidad bootstrap	No hay evidencia fuerte de mejora.
Base rate distinto entre calibración y evaluación	El calibrador ya nace con deriva posible.
Umbral elegido después de mirar demasiadas variantes	Estás ajustando a la evaluación, no validando.

Riesgo-cobertura: automatizar menos, fallar mejor

Cuando un sistema puede abstenerse, revisar o escalar, no miramos solo accuracy. Miramos la curva riesgo-cobertura: qué error queda cuando automatizamos cierto porcentaje de casos.

Las expresiones siguientes son una forma operativa de la idea de clasificación selectiva: aceptar solo una parte de los casos y medir el error en ese subconjunto. Geifman y El-Yaniv (2017) estudian esta lógica para redes profundas; aquí la traducimos a una política de automatización, revisión y coste.

Definimos una función de aceptación $A_i(c)$, que vale 1 si el caso i se automatiza bajo una política con cobertura objetivo c, y 0 si se revisa:

$$Coverage(c) = \frac{1}{N}\sum_{i=1}^{N} A_i(c)$$

$$Risk(c) = \frac{\sum_{i=1}^{N} \ell_i A_i(c)}{\sum_{i=1}^{N} A_i(c)}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$Coverage(c)$	Proporción de casos automatizados.	0,62.
$Risk(c)$	Error medio dentro de los casos automatizados.	0,08.
$A_i(c)$	Indicador de automatización del caso i.	1 si sale de la zona gris.
ℓ_i	Pérdida o coste del caso i.	0 si acierta, 8 si pierde un urgente.
N	Número total de casos evaluados.	500.

En palabras: cobertura dice cuánto automatizas; riesgo dice cuánto error o coste queda dentro de lo que has decidido automatizar.

La lectura profesional es esta:

RESULTADO	DECISIÓN
Alta cobertura y bajo riesgo	Buen candidato para automatización.
Alta cobertura y alto riesgo	El sistema automatiza demasiado.
Baja cobertura y bajo riesgo	Puede servir como primera fase conservadora.
Baja cobertura y alto riesgo	El score no separa bien; no basta con calibrar.

La literatura de clasificación selectiva trabaja precisamente con esta idea: permitir que el modelo responda solo cuando su confianza supera una condición y medir el error en el subconjunto aceptado.[25]

En un producto de IA, esta curva te ayuda a defender frases como:

> “Automatizamos el 40 % de los casos con error automático menor del 13 %, y el resto pasa a revisión porque el conjunto conformal sigue ambiguo”.

Eso es mucho más útil que “el modelo tiene 86 % de accuracy”.

Contrato de calibración en producción

Un calibrador no debería vivir como una función suelta escondida en código. Debería tener un contrato operativo.

CAMPO	PREGUNTA QUE RESPONDE
`model_version`	¿Qué modelo produjo el score bruto?
`score_name`	¿Qué número estamos calibrando exactamente?
`score_semantics`	¿Qué evento observable intenta predecir?
`calibration_dataset_hash`	¿Con qué datos se ajustó?
`evaluation_dataset_hash`	¿Con qué datos se validó?
`calibrator_type`	¿Qué transformación se usó?
`calibrator_version`	¿Qué versión del calibrador está desplegada?
`policy_version`	¿Qué umbrales y costes deciden?
`valid_slices`	¿En qué segmentos se ha medido?
`known_bad_slices`	¿Dónde no debe automatizar?
`recalibration_triggers`	¿Qué cambios obligan a recalibrar?
`owner`	¿Quién responde por esta política?

Un manifest mínimo podría verse así:

```
{
  "model_version": "support-prioritizer-2026-06-01",
  "prompt_version": "ticket-routing-prompt-v4",
  "retrieval_version": "support-docs-index-2026-05-30",
  "score_name": "raw_score",
  "score_semantics": "probabilidad de que el ticket requiera prioridad urgente",
  "calibrator_type": "histogram_laplace",
  "calibrator_version": "cal-ticket-v1",
  "policy_version": "support-thresholds-v3",
  "valid_slices": ["general", "becas", "matricula"],
  "known_bad_slices": [],
  "recalibration_triggers": {
    "model_changed": true,
    "prompt_changed": true,
    "base_rate_relative_change": 0.20,
    "ece_regression": 0.03,
    "new_slice_without_50_cases": true
  },
  "owner": "equipo-ia"
}
```

Este manifest no es burocracia. Es lo que permite revisar una incidencia tres semanas después y saber qué número mandaba, con qué datos se calibró y cuándo dejó de ser fiable.

Documentación profesional: model card, data card y SLO

Una política calibrada no debería quedarse encerrada en un notebook. Si va a afectar a un sistema real, necesita tres documentos vivos:

DOCUMENTO	QUÉ CONTIENE	PREGUNTA QUE RESUELVE
Model card	Modelo, uso previsto, límites, métricas, resultados por slice y cambios relevantes.	“¿Para qué sirve este modelo y dónde no deberíamos usarlo?”
Data card	Origen de datos, composición, etiquetas, cobertura, huecos y transformaciones.	“¿Qué mundo representa este dataset y cuál deja fuera?”
SLO de IA	Objetivos medibles de calidad, revisión, latencia, coste y disponibilidad.	“¿Cuándo decimos que el sistema está suficientemente sano?”

En calibración, esos documentos deberían conectarse así:

PIEZA	CAMPO MÍNIMO
Model card	`model_version`, `score_name`, `score_semantics`, métricas por slice, límites conocidos.
Data card	`dataset_hash`, split, fecha, criterio de etiquetado, distribución por slice.
SLO	`max_auto_error_rate`, `max_review_rate`, `min_auto_coverage`, latencia y coste máximo.
Manifest	Qué combinación exacta de modelo, datos, calibrador y política está aprobada.

Esto no es papeleo académico. Sculley y otros muestran que los sistemas ML acumulan deuda técnica por dependencias ocultas, cambios silenciosos y límites difusos entre componentes. En calibración, una dependencia oculta puede ser un prompt, un índice RAG, un proveedor, una plantilla de salida, un criterio de etiquetado o una cola de revisión.

Una frase útil para equipos:

> Si no puedes decir qué cambió entre dos runs, no puedes decir si el calibrador sigue siendo válido.

El SLO de IA tampoco debería sonar genérico. Debe ser medible:

SLI	SLO POSIBLE
Error automático en casos aceptados	Menor o igual que 18 % en evaluación revisada.
Tasa de revisión	Menor o igual que 60 % con capacidad operativa disponible.
Cobertura automática	Mayor o igual que 40 % sin romper el error automático.
ECE calibrado	Menor o igual que 0,16 en evaluación y revisado por slice.
Latencia de decisión	p95 menor de 1,5 s si no hay revisión.

Si el SLO se rompe, la acción debe estar escrita: limitar automatización, volver a una política anterior, aumentar revisión, recalibrar o bloquear despliegue hasta reunir muestra suficiente.

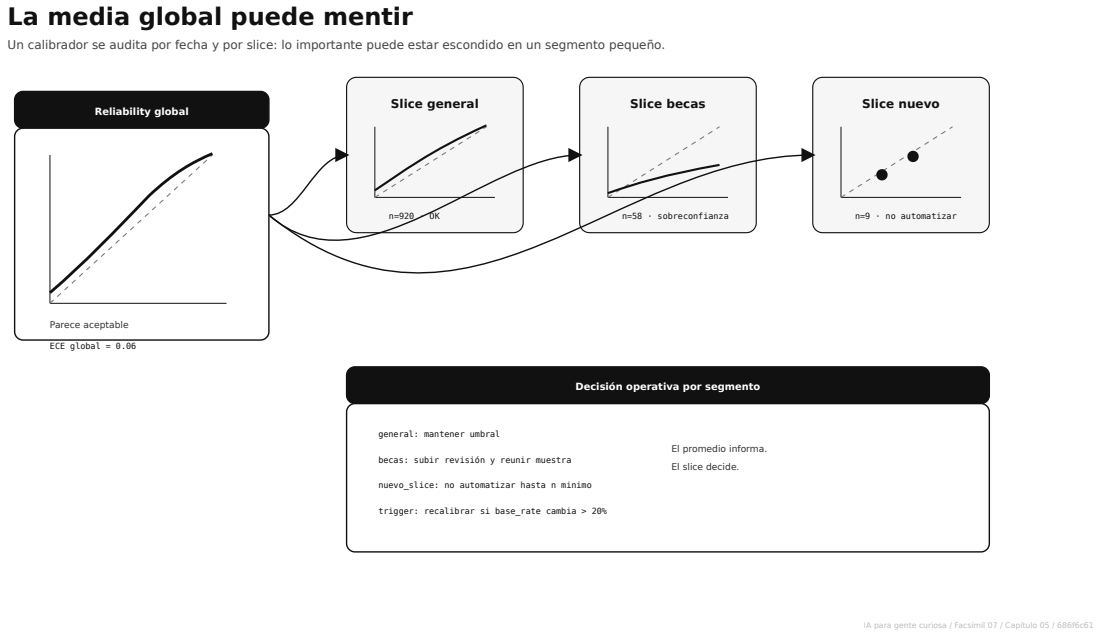
Para ordenar responsabilidades, el NIST AI RMF también es útil porque separa gobernar, mapear, medir y gestionar riesgos de sistemas de IA.[26] En este capítulo no lo usamos como marco legal, sino como recordatorio técnico: medir sin gestionar no cambia el sistema.

Deriva: cuando el calibrador envejece

Un calibrador aprende una relación entre score y frecuencia real en un contexto. Si ese contexto cambia, el calibrador puede seguir ejecutándose y estar equivocado. Eso es especialmente peligroso porque no falla con una excepción: falla con una falsa sensación de control.

La literatura sobre dataset shift describe precisamente el problema de entrenar o evaluar bajo una distribución y usar el sistema bajo otra.[27] En calibración, esa diferencia suele aparecer en tres formas:

CAMBIO	QUÉ CAMBIA	EFECTO SOBRE CALIBRACIÓN
Covariate shift	Cambian las entradas: canal, idioma, longitud, tipo documental.	Las bandas de score pueden mezclar casos que antes no existían.
Label shift o base rate shift	Cambia la proporción real de positivos.	Un umbral que antes revisaba poco puede empezar a aceptar demasiado.
Concept drift	Cambia la relación entre entrada y etiqueta.	El score deja de significar lo mismo aunque su distribución parezca estable.



La calibración se revisa por segmentos. Un promedio global aceptable puede esconder un slice sobreconfiado o sin muestra suficiente.

La decisión técnica no debería ser “recalibrar todo” cada vez que algo se mueve. A veces basta con bloquear un slice nuevo, subir revisión en un canal, recoger más etiquetas o volver temporalmente a un umbral conservador. Lo importante es que la política diga qué señal dispara qué acción.

Monitorización: cuándo recalibrar

La calibración no es una ceremonia de una vez. Se vigila.

SEÑAL ONLINE	QUÉ INDICA	ACCIÓN
Cambia el base rate	La proporción real de positivos ya no se parece a calibración.	Recalcular reliability diagram por fecha y slice.
Sube ECE en muestra revisada	El score ya no corresponde a frecuencia real.	Recalibrar o limitar automatización.
Crece la cola de revisión	La zona gris está absorbiendo demasiados casos.	Revisar umbrales, capacidad o calidad del modelo.
Aumentan errores automáticos	La política acepta casos que antes separaba bien.	Bajar cobertura automática y abrir análisis de regresión.
Aparece un slice nuevo	No hay evidencia para automatizar ese segmento.	Revisar hasta reunir muestra mínima.
Cambia modelo, prompt, RAG o herramienta	Cambió el sistema que produce el score.	Invalidar calibrador anterior salvo prueba contraria.

Para ingenieros de IA, el patrón operativo sano es:

1. Versionar modelo, prompt, datos, calibrador y política.
2. Mantener una muestra revisada de producción.
3. Medir Brier, ECE, error automático y revisión por slice.
4. Tener una acción automática si el calibrador caduca: limitar automatización, subir revisión o volver a política anterior.

Si no hay acción asociada, la métrica es decoración.

Herramientas que sí usaría con criterio

No hace falta empezar con una plataforma enorme. Hace falta saber qué problema tienes. Las herramientas son útiles cuando acompañan una decisión concreta: calibrar probabilidades, construir intervalos, vigilar drift o dejar evidencias.

HERRAMIENTA	PARA QUÉ SIRVE AQUÍ	CUÁNDO LA USARÍA	LÍMITE
scikit-learn	`CalibratedClassifierCV`, curvas de calibración y calibradores clásicos.	Clasificación tabular o pipeline Python clásico.	No sustituye una política de producto ni monitoriza producción sola.
netcal	Medir y mitigar miscalibration en estimaciones de confianza.	Redes neuronales o experimentos donde quieres más métodos de calibración.	Exige entender qué score estás calibrando.
MAPIE	Prediction intervals, prediction sets y conformal prediction.	Cuando necesitas cuantificar incertidumbre con conjuntos o intervalos.	La garantía depende de datos representativos y supuestos de intercambio.
Evidently	Evaluación, tests y monitorización de drift/datos/modelos.	Para vigilar producción, comparar datasets y generar reportes operativos.	Detectar drift no decide automáticamente qué acción tomar.

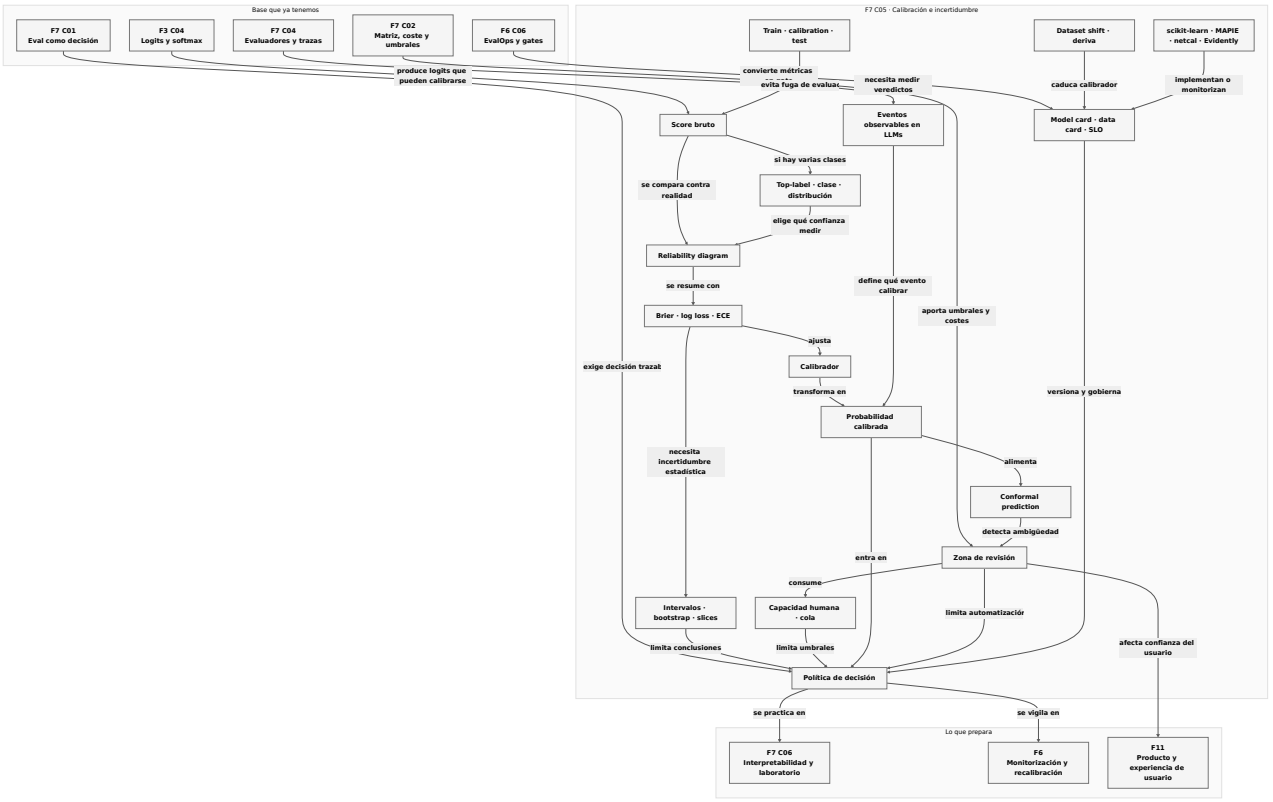
netcal se presenta como framework para medir y mitigar miscalibration de estimaciones de confianza.[28] Evidently documenta métricas y presets para detectar cambios en distribución de datos, además de monitorización de sistemas de ML y LLM.[29]

El orden sensato para un alumno o equipo pequeño sería:

1. Empezar con CSV, script y manifest como en el cuaderno del facsímil.
2. Si el modelo es clásico, probar calibración con scikit-learn y comparar con la implementación propia.
3. Si necesitas intervalos o conjuntos, estudiar MAPIE y contrastarlo con el split conformal explicado aquí.
4. Si ya tienes producción, añadir monitorización de drift y reportes con una herramienta como Evidently.
5. Si la calibración es central en el producto, documentar versión de calibrador, dataset y política como artefactos de release.

La herramienta buena no es la más completa. Es la que te permite contestar: qué score calibré, con qué datos, qué cambió, qué decisión toma y cuándo deja de ser fiable.

Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN BREVE
Score bruto	Puntuación salida del modelo antes de calibrar.
Probabilidad calibrada	Score interpretable como frecuencia esperada de acierto.
Discriminación	Capacidad de ordenar casos positivos por encima de negativos.
Calibración	Correspondencia entre confianza predicha y frecuencia real.
Brier score	Error cuadrático medio de probabilidades.
Log loss	Pérdida que castiga mucho equivocarse con confianza alta.
ECE	Error de calibración esperado por bandas.
Reliability diagram	Gráfico de confianza media frente a accuracy por banda.
Platt scaling	Calibración sigmoidal de un score.
Isotonic regression	Calibración monótona por tramos.
Conjunto de calibración	Partición reservada para ajustar calibradores o cuantiles sin tocar el test final.
Top-label calibration	Calibración de la confianza de la clase predicha.
Temperature scaling	Ajuste de logits con una temperatura aprendida.
Conformal prediction	Construcción de conjuntos o intervalos con cobertura objetivo.
Cobertura	Proporción de casos donde el conjunto contiene la respuesta correcta.
Score de no conformidad	Medida de rareza que decide si una clase o predicción entra en el conjunto conformal.
Calibración por slice	Medición o ajuste de calibración por segmento operativo.
Zona de revisión	Banda donde el sistema no automatiza por incertidumbre.
Riesgo-cobertura	Curva que compara porcentaje automatizado y error en lo automatizado.
Deriva de calibración	Cambio de la relación entre score y frecuencia real.
Dataset shift	Cambio entre distribución de entrenamiento, evaluación o producción.

TÉRMINO	DEFINICIÓN BREVE
Capacidad de revisión	Volumen de casos que el equipo humano puede revisar sin romper SLO.
Manifest de calibración	Contrato versionado de score, calibrador, política, datos y triggers.
Incertidumbre semántica	Duda sobre el significado de una respuesta, no solo sobre su texto exacto.
Intervalo Wilson	Intervalo para una proporción observada, útil con muestras pequeñas.
Bootstrap	Remuestreo con reemplazo para estimar variabilidad de métricas.
Model card	Documento de modelo con uso previsto, límites y métricas por segmento.
Data card	Documento de datos con origen, composición, huecos y uso previsto.
SLO de IA	Objetivo medible de calidad, coste, revisión, latencia o disponibilidad.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Leer cualquier score como probabilidad	El número parece probabilístico aunque solo ordene.	Medir calibración antes de automatizar con umbrales.
Mirar solo accuracy	Puedes acertar mucho y estar sobreconfiado.	Añadir Brier, log loss, ECE y reliability diagram.
Calibrar con el test final	El resultado queda contaminado por decisiones de ajuste.	Separar train, calibration y evaluation.
Usar ECE como número absoluto	Depende de bandas y puede esconder slices malos.	Mirar curva, slices y casos frontera.
Automatizar la zona gris	La presión por reducir revisión empuja a decidir donde falta señal.	Diseñar revisión como parte del sistema, no como fracaso.
Olvidar que la calibración caduca	Cambian datos, modelo, prompt, retrieval o usuarios.	Versionar calibrador y recalibrar con monitorización.
Confundir logprob con verdad	Un token probable no garantiza una respuesta correcta.	Calibrar eventos observables de tarea.
Publicar sin intervalos	Una métrica puntual puede parecer más estable de lo que es.	Añadir Wilson, bootstrap y mínimos por slice.

Antes de pasar página

Antes de avanzar, deberías poder responder:

1. ¿Por qué un score alto no implica probabilidad calibrada?
2. ¿Qué diferencia hay entre discriminación y calibración?
3. ¿Cómo se calcula Brier score y qué penaliza?
4. ¿Por qué log loss castiga tanto una predicción confiada que falla?
5. ¿Qué mide ECE y por qué depende de las bandas?
6. ¿Cómo leerías un reliability diagram por debajo de la diagonal?
7. ¿Cuándo usarías temperature scaling frente a isotonic regression?
8. ¿Por qué no se ajusta el calibrador con el test final?
9. ¿Qué diferencia hay entre calibrar top-label y calibrar por clase?

10. ¿Qué supuesto sostiene conformal prediction?
11. ¿Qué significa que un conjunto conformal tenga dos clases?
12. ¿Por qué cobertura alta no implica utilidad alta?
13. ¿Por qué la curva riesgo-cobertura es más útil que una accuracy global para decidir automatización?
14. ¿Cómo afecta la capacidad de revisión a los umbrales?
15. ¿Qué debería contener un manifest de calibración?
16. ¿Por qué no basta con mirar logprobs para calibrar una respuesta de LLM?
17. ¿Qué aporta un intervalo Wilson en una banda pequeña?
18. ¿Qué diferencia hay entre model card, data card y manifest de calibración?
19. ¿Qué señales de dataset shift obligarían a limitar o recalibrar?
20. ¿Qué herramienta usarías para calibrar, conformalizar o monitorizar y por qué?
21. ¿Qué SLO de IA escribirías para decidir si esta política puede desplegarse?
22. ¿Qué archivos entrega la práctica del capítulo?

En resumen

IDEA	QUÉ TE LLEVAS
Un score no es automáticamente una probabilidad.	Primero mide si su confianza coincide con frecuencias reales.
Calibrar no es subir accuracy.	Es hacer que el número sea útil para decidir bajo coste.
El split importa tanto como la métrica.	Train, calibration, test y producción revisada no cumplen el mismo papel.
Multiclase exige mirar más que la clase ganadora.	Top-label puede orientar, pero los slices y clases críticas deciden.
Brier, log loss, ECE y reliability diagram se complementan.	Ninguna métrica aislada basta para publicar una política.
Conformal prediction convierte incertidumbre en conjuntos o intervalos.	Si el conjunto es ambiguo, el sistema debe revisar o abstenerse.
Cobertura no es utilidad.	Un conjunto enorme puede cubrir mucho y decidir poco.
La zona gris consume operación.	Si revisión humana no cabe, el umbral no es viable.
En LLMs se calibran eventos, no frases de confianza.	El evento debe ser observable: formato válido, respuesta soportada, acción correcta o salida aceptada.
La calibración es operativa.	Versiona calibrador, umbrales, política, datos, SLOs, herramientas y monitorización porque todo eso caduca.

Para saber más

Angelopoulos, A. N. y Bates, S. (2021). A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification. *arXiv*. https://arxiv.org/abs/2107.07511

Brier, G. W. (1950). Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, 78(1), 1-3. https://doi.org/10.1175/1520-0493(1950)078<0001:VOEPIO>2.0.CO;2

Breck, E., Cai, S., Nielsen, E., Salib, M. y Sculley, D. (2017). The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. *IEEE Big Data*, 1123-1132. https://research.google/pubs/pub46555/

Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1), 1-26. https://doi.org/10.1214/aos/1176344552

Evidently AI. (2026). *Data Drift Documentation*. https://docs.evidentlyai.com/metrics/explainer_drift

Geifman, Y. y El-Yaniv, R. (2017). Selective Classification for Deep Neural Networks. *Advances in Neural Information Processing Systems*. https://proceedings.neurips.cc/paper/2017/hash/4a5cfa9281924139db466a8a19291aff-Abstract.html

Gneiting, T. y Raftery, A. E. (2007). Strictly Proper Scoring Rules, Prediction, and Estimation. *Journal of the American Statistical Association*, 102(477), 359-378. https://doi.org/10.1198/016214506000001437

Guo, C., Pleiss, G., Sun, Y. y Weinberger, K. Q. (2017). On Calibration of Modern Neural Networks. *Proceedings of the 34th International Conference on Machine Learning*, 70, 1321-1330. https://proceedings.mlr.press/v70/guo17a.html

Gupta, C. y Ramdas, A. (2021). *Top-label Calibration and Multiclass-to-Binary Reductions*. Workshop on Uncertainty and Robustness in Deep Learning. https://www.gatsby.ucl.ac.uk/~balaji/udl2021/accepted-papers/UDL2021-paper-060.pdf

Jones, C., Wilkes, J., Murphy, N. y Smith, C. (2016). Service Level Objectives. En *Site Reliability Engineering*. https://sre.google/sre-book/service-level-objectives/

Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., Johnston, S. y otros. (2022). *Language Models (Mostly) Know What They Know*. arXiv. https://arxiv.org/abs/2207.05221

Kuhn, L., Gal, Y. y Farquhar, S. (2023). Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation. *International Conference on Learning Representations*. https://arxiv.org/abs/2302.09664

Little, J. D. C. (1961). A Proof for the Queuing Formula: L = lambda W. *Operations Research*, 9(3), 383-387. https://doi.org/10.1287/opre.9.3.383

MAPIE. (2026). *MAPIE: Model Agnostic Prediction Interval Estimator*. https://mapie.readthedocs.io/

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D. y Gebru, T. (2019). Model Cards for Model Reporting. *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 220-229. https://doi.org/10.1145/3287560.3287596

Murphy, A. H. (1973). A New Vector Partition of the Probability Score. *Journal of Applied Meteorology*, 12(4), 595-600. https://doi.org/10.1175/1520-0450(1973)012<0595:ANVPOT>2.0.CO;2

Naeini, M. P., Cooper, G. F. y Hauskrecht, M. (2015). Obtaining Well Calibrated Probabilities Using Bayesian Binning. *AAAI*. https://ojs.aaai.org/index.php/AAAI/article/view/9602

netcal. (2026). *API Reference of netcal*. https://efs-opensource.github.io/calibration-framework/build/html/index.html

Niculescu-Mizil, A. y Caruana, R. (2005). Predicting Good Probabilities with Supervised Learning. *Proceedings of the 22nd International Conference on Machine Learning*, 625-632. https://doi.org/10.1145/1102351.1102430

Platt, J. C. (1999). Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. En *Advances in Large Margin Classifiers*. MIT Press.

Pushkarna, M., Zaldivar, A. y Kjartansson, O. (2022). *Data Cards: Purposeful and Transparent Dataset Documentation for Responsible AI*. arXiv. https://arxiv.org/abs/2204.01075

Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A. y Lawrence, N. D. (Eds.). (2009). *Dataset Shift in Machine Learning*. MIT Press. https://mitpress.mit.edu/9780262170055/dataset-shift-in-machine-learning/

scikit-learn. (2026). *Probability Calibration*. https://scikit-learn.org/stable/modules/calibration.html

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J. F. y Dennison, D. (2015). Hidden Technical Debt in Machine Learning Systems. *Advances in Neural Information Processing Systems*. https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems

Shafer, G. y Vovk, V. (2008). A Tutorial on Conformal Prediction. *Journal of Machine Learning Research*, 9, 371-421. https://www.jmlr.org/papers/v9/shafer08a.html

Tabassi, E. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.100-1

Vovk, V., Gammerman, A. y Shafer, G. (2005). *Algorithmic Learning in a Random World*. Springer. https://doi.org/10.1007/b106715

Wilson, E. B. (1927). Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, 22(158), 209-212. https://doi.org/10.1080/01621459.1927.10502953

Notas

1. Brier, G. W. (1950). Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, 78(1), 1-3. https://doi.org/10.1175/1520-0493(1950)078<0001:VOEPIO>2.0.CO;2 ↩

2. Murphy, A. H. (1973). A New Vector Partition of the Probability Score. *Journal of Applied Meteorology*, 12(4), 595-600. https://doi.org/10.1175/1520-0450(1973)012<0595:ANVPOT>2.0.CO;2 ↩

3. Niculescu-Mizil, A. y Caruana, R. (2005). Predicting Good Probabilities with Supervised Learning. *ICML*. https://doi.org/10.1145/1102351.1102430 ↩

4. Platt, J. C. (1999). Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. En *Advances in Large Margin Classifiers*. MIT Press. ↩

5. Guo, C., Pleiss, G., Sun, Y. y Weinberger, K. Q. (2017). On Calibration of Modern Neural Networks. *ICML*. https://proceedings.mlr.press/v70/guo17a.html ↩

6. Vovk, V., Gammerman, A. y Shafer, G. (2005). *Algorithmic Learning in a Random World*. Springer. https://doi.org/10.1007/b106715 ↩

7. Shafer, G. y Vovk, V. (2008). A Tutorial on Conformal Prediction. *Journal of Machine Learning Research*, 9, 371-421. https://www.jmlr.org/papers/v9/shafer08a.html ↩

8. Angelopoulos, A. N. y Bates, S. (2021). A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification. arXiv. https://arxiv.org/abs/2107.07511 ↩

9. scikit-learn. (2026). *Probability Calibration*. https://scikit-learn.org/stable/modules/calibration.html. Consultado el 1 de junio de 2026. ↩

10. Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., Johnston, S. y otros. (2022). *Language Models (Mostly) Know What They Know*. arXiv. https://arxiv.org/abs/2207.05221 ↩

11. Kuhn, L., Gal, Y. y Farquhar, S. (2023). Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation. *ICLR*. https://arxiv.org/abs/2302.09664 ↩

12. Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D. y Gebru, T. (2019). Model Cards for Model Reporting. *FAT*, 220-229. https://doi.org/10.1145/3287560.3287596 ↩

13. Pushkarna, M., Zaldivar, A. y Kjartansson, O. (2022). *Data Cards: Purposeful and Transparent Dataset Documentation for Responsible AI*. arXiv. https://arxiv.org/abs/2204.01075 ↩

14. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J. F. y Dennison, D. (2015). Hidden Technical Debt in Machine Learning Systems. *NeurIPS*. https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems ↩

15. Breck, E., Cai, S., Nielsen, E., Salib, M. y Sculley, D. (2017). The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. *IEEE Big Data*. https://research.google/pubs/pub46555/ ↩

16. Jones, C., Wilkes, J., Murphy, N. y Smith, C. (2016). Service Level Objectives. En *Site Reliability Engineering*. https://sre.google/sre-book/service-level-objectives/ ↩

17. Guo y otros, 2017. ↩

18. scikit-learn. (2026). Probability calibration. https://scikit-learn.org/stable/modules/calibration.html ↩

19. MAPIE. (2026). The conformalization calibration set. https://mapie.readthedocs.io/en/stable/split_cross_conformal.html ↩

20. Gupta, C. y Ramdas, A. (2021). Top-label Calibration and Multiclass-to-Binary Reductions. https://www.gatsby.ucl.ac.uk/~balaji/udl2021/accepted-papers/UDL2021-paper-060.pdf ↩

21. MAPIE. (2026). MAPIE: Model Agnostic Prediction Interval Estimator. https://mapie.readthedocs.io/ ↩

22. Little, J. D. C. (1961). A Proof for the Queuing Formula: L = lambda W. *Operations Research*, 9(3), 383-387. https://doi.org/10.1287/opre.9.3.383 ↩

23. Wilson, E. B. (1927). Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, 22(158), 209-212. https://doi.org/10.1080/01621459.1927.10502953 ↩

24. Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1), 1-26. https://doi.org/10.1214/aos/1176344552 ↩

25. Geifman, Y. y El-Yaniv, R. (2017). Selective Classification for Deep Neural Networks. *NeurIPS*. https://proceedings.neurips.cc/paper/2017/hash/4a5cfa9281924139db466a8a19291aff-Abstract.html ↩

26. Tabassi, E. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.100-1 ↩

27. Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A. y Lawrence, N. D. (eds.). (2009). *Dataset Shift in Machine Learning*. MIT Press. https://mitpress.mit.edu/9780262170055/dataset-shift-in-machine-learning/ ↩

28. netcal. (2026). API Reference of netcal. https://efs-opensource.github.io/calibration-framework/build/html/index.html ↩

29. Evidently AI. (2026). Data Drift Documentation. https://docs.evidentlyai.com/metrics/explainer_drift ↩

Cuadernos para practicar

Escanea el código con el móvil para abrir el cuaderno en Google Colab y ejecutarlo sin instalar nada.

Calibración: cuando el modelo dice «90%», ¿es un 90%?

https://colab.research.google.com/github/686f6c61/iaparagentecuriosa-notebooks/blob/main/Facs%C3%ADmil%2007/05-calibracion-probabilidades.ipynb

Capítulo 06: Interpretabilidad práctica y evaluación

Entrando en el tema

Este facsímil empezó con una idea sobria: una eval existe para tomar una decisión. Después medimos errores, evaluamos RAG, diseñamos evaluadores, calibramos scores y convertimos incertidumbre en política. Ahora queda una pregunta que aparece siempre en ingeniería de IA:

> ¿Podemos explicar lo suficiente para depurar, publicar, limitar o rechazar este sistema?

Interpretabilidad no es una palabra para tranquilizar a alguien en una reunión. Tampoco es una imagen bonita, una tabla de pesos o una respuesta del modelo diciendo “he decidido esto por...”. Interpretabilidad, en ingeniería, es una herramienta para hacer mejores preguntas: qué feature pesa, qué caso cambia, qué slice se rompe, qué explicación es estable, qué parte del sistema no entendemos todavía y qué decisión podemos defender.

La idea central del capítulo es esta: **una explicación no se acepta porque suene bien; se acepta porque ayuda a diagnosticar y resiste comprobaciones**.

Qué deberías poder hacer al terminar

Este capítulo no pretende que salgas repitiendo nombres de librerías. Pretende que puedas mirar una explicación y decidir si sirve para depurar un sistema, para documentar una release, para pedir revisión humana o para decir “todavía no podemos publicar esto”.

Al terminar deberías poder hacer esto:

RESULTADO DE APRENDIZAJE	EVIDENCIA DE QUE LO SABES HACER
Separar interpretabilidad, explicabilidad y transparencia.	No usas esas palabras como sinónimos automáticos.
Elegir método según pregunta.	Distingues explicación local, global, contrafactual, conceptual y mecánica.
Evaluar una explicación.	Mides fidelidad, estabilidad, sensibilidad y utilidad operativa.
Leer atribuciones con cautela.	Sabes por qué una atribución plausible puede ser infiel.
Diseñar contrafactuales útiles.	Separas cambios accionables de cambios imposibles o injustos.
Conectar interpretabilidad con EvalOps.	Conviertes explicaciones en checks, model card y casos de regresión.
Cerrar el facsímil con criterio.	Sabes integrar métricas, RAG, evaluadores, calibración e interpretación en una decisión de release defendible.

El problema: una explicación puede sonar perfecta y no explicar nada

Imagina un sistema que prioriza tickets académicos. Para un caso concreto devuelve:

```
{
  "score": 0.86,
  "decision": "urgente",
  "explanation": "El caso parece urgente por su tono y por la fecha cercana."
}
```

La frase suena humana. Pero un ingeniero debería preguntar:

PREGUNTA	POR QUÉ IMPORTA
¿El modelo tenía una feature llamada `tono`?	Si no existe, la explicación puede ser una racionalización.
¿Qué pasa si eliminamos la feature principal?	Si la salida no cambia, la explicación no sostiene la decisión.
¿La explicación cambia ante pequeñas perturbaciones?	Si cambia demasiado, no es estable.
¿El caso era parte de un slice problemático?	Una explicación local puede esconder un patrón global malo.
¿El cambio recomendado es accionable?	No todo contrafactual sirve para una persona o un equipo.

Esto es especialmente delicado en LLMs. Una respuesta puede construir una narración convincente después de producir la salida. En ese caso, la explicación puede ser plausible para nosotros, pero no fiel al proceso que produjo el resultado.

Por eso el capítulo no va de “hacer explicable la IA” en abstracto. Va de crear un expediente técnico: qué método usamos, qué pregunta responde, qué prueba lo contradice y qué decisión permite tomar.

Qué no es interpretabilidad

Interpretabilidad no es necesariamente simplicidad. Un modelo lineal puede ser fácil de leer y aun así estar usando features mal definidas, proxies pobres o datos incompletos. Un árbol pequeño puede ser comprensible y seguir aprendiendo una regla poco útil.

Tampoco es una promesa de verdad. Una explicación post hoc puede aproximar el comportamiento de un modelo complejo, pero no convertirse en el modelo real. LIME aproxima localmente; SHAP reparte contribuciones bajo supuestos; saliency maps señalan sensibilidad; contrafactuales proponen cambios; ninguna de esas piezas sustituye una evaluación.

Y, sobre todo, interpretabilidad no es decoración de compliance. Si nadie puede decir qué decisión cambia gracias a la explicación, quizá solo hemos añadido otra pantalla.

CONFUSIÓN	LECTURA DE INGENIERÍA
“Tenemos explicación, así que el sistema es fiable”.	La explicación también se evalúa.
“El mapa de calor marca la zona importante”.	Hay que probar sensibilidad, estabilidad y sanity checks.
“El modelo dice por qué decidió”.	Una explicación textual puede no ser fiel.
“SHAP lo arregla”.	SHAP responde una familia concreta de preguntas bajo supuestos concretos.
“Contrafactual significa recomendación”.	Solo algunos cambios son accionables y aceptables.

Lipton advertía que “interpretabilidad” suele mezclar propiedades distintas: transparencia, simulabilidad, decomponibilidad, post hoc explanations y confianza humana.[1] Doshi-Velez y Kim proponían tratar la interpretabilidad como una cuestión evaluable, no como una etiqueta estética.[2]

Qué sí es interpretar un sistema de IA

Interpretar es responder una pregunta situada. No “explícame el modelo”, sino:

PREGUNTA	MÉTODO RAZONABLE
¿Por qué este caso se marcó como urgente?	Explicación local y prueba de borrado.
¿Qué features pesan globalmente?	Importancia por permutación, SHAP agregado o modelo interpretable.
¿Qué tendría que cambiar para otra decisión?	Contrafactual accionable.
¿Qué concepto humano usa el modelo?	TCAV o análisis por conceptos.
¿Qué región de una imagen activó una clase?	Grad-CAM u otro método visual con sanity checks.
¿Qué parte interna participa en una asociación factual?	Intervenciones causales o análisis mecanicista.
¿Qué explicación puedo mostrar a usuario final?	Una explicación validada por utilidad, no solo por fidelidad técnica.

Hay dos ejes que conviene escribir siempre:

EJE	PREGUNTA
Local frente a global	¿Explicamos un caso concreto o el comportamiento general?
Fidelidad frente a plausibilidad	¿Refleja el modelo o solo convence a la persona?
Diagnóstico frente a comunicación	¿Sirve para depurar o para informar una decisión?
Accionable frente a descriptivo	¿Permite hacer algo distinto?

Jacovi y Goldberg separan explícitamente fidelidad y plausibilidad en NLP: una explicación puede parecer buena a una persona y no reflejar el mecanismo que produjo la salida.[3] Esa distinción es clave para LLMs.

Fecha de corte del estado del arte

Fecha de corte: 6 de junio de 2026.
Fuentes consultadas: trabajos clásicos sobre ciencia de la interpretabilidad, LIME, SHAP, Integrated Gradients, Grad-CAM, sanity checks, contrafactuales, modelos interpretables para decisiones sensibles, TCAV, interpretabilidad fiel en NLP y localización causal de asociaciones factuales en GPT.

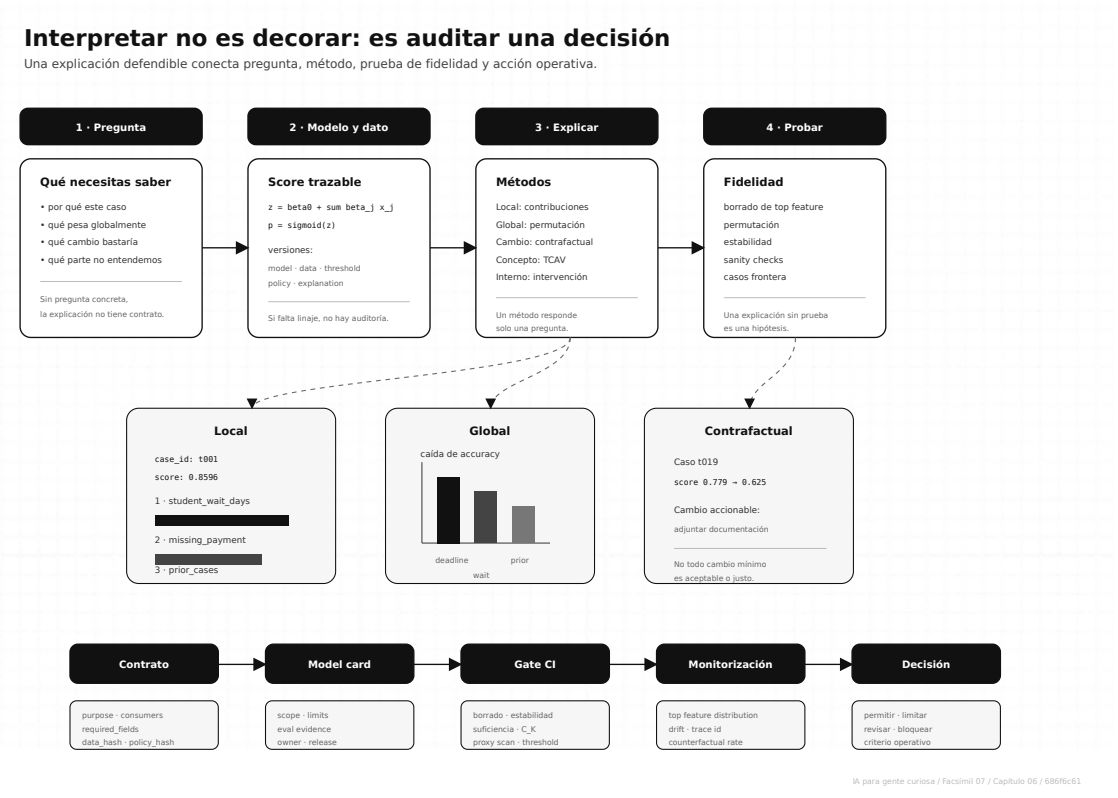
LIME propone aproximar localmente un modelo complejo con un modelo interpretable alrededor de una predicción concreta.[4] SHAP conecta atribuciones aditivas con valores de Shapley y un marco común para asignar importancia a features.[5] Integrated Gradients plantea axiomas como sensibilidad e invariancia de implementación para atribuciones en redes profundas.[6]

Grad-CAM localiza regiones relevantes para modelos visuales usando gradientes hacia capas convolucionales.[7] Adebayo y otros mostraron que algunos mapas de saliencia pueden fallar sanity checks: si la explicación apenas cambia al aleatorizar parámetros o etiquetas, hay que desconfiar.[8]

Los contrafactuales explican decisiones indicando cambios mínimos que producirían otra salida.[9] Rudin defiende que en decisiones de alto impacto conviene preferir modelos interpretables cuando sea posible, en lugar de explicar una caja negra después.[10]

Para conceptos humanos, TCAV cuantifica sensibilidad a direcciones conceptuales aprendidas.[11] En modelos de lenguaje, trabajos como ROME usan intervenciones causales para localizar asociaciones factuales en GPT.[12]

Anatomía de una auditoría de interpretabilidad



> Una auditoría de interpretabilidad empieza con una pregunta y termina en una decisión. Entre medias hay método, prueba y documentación.

La explicación empieza en el contrato de datos

Antes de hablar de SHAP, LIME o contrafactuales hay una pregunta más aburrida y más útil: ¿qué significa cada feature? Si el dataset no tiene contrato, la explicación hereda ambigüedad. Una feature llamada `prior_cases` puede significar casos previos reales, tickets duplicados, incidencias abiertas, expedientes rechazados o simples contactos con soporte. La explicación “prior_cases empuja la decisión” no vale lo mismo en cada caso.

En ingeniería de IA conviene documentar cada feature con cuatro piezas:

PIEZA	PREGUNTA	EJEMPLO DEL CUADERNO
Semántica	¿Qué representa exactamente?	`student_wait_days` mide días de espera acumulada.
Procedencia	¿De qué sistema sale y cuándo se actualiza?	CRM académico o sistema de tickets.
Rango válido	¿Qué valores son posibles y cuáles son errores?	0 a 21 en `student_wait_days`.
Uso permitido	¿Puede usarse para decidir, revisar o solo diagnosticar?	`missing_payment` puede activar revisión, no comunicación automática.

Esto importa porque una explicación puede ser fiel al modelo y aun así mala para producto. Si `prior_cases` está muy correlacionada con `student_wait_days`, quizá ambas features cuentan la misma historia con dos nombres distintos. Si `contains_policy_keyword` se dispara por palabras mal tokenizadas, la explicación puede parecer jurídica sin serlo. Si `docs_attached` se usa como recomendación, hay que confirmar que adjuntar documentos sea realmente una acción disponible para la persona o para el equipo.

La interpretabilidad útil, por tanto, no termina en “la feature pesa”. Termina en una decisión sobre datos: mantener, renombrar, separar, eliminar, auditar por slice o convertir en una regla de revisión humana. Esa es la diferencia entre una explicación de demo y una explicación que un equipo puede defender.

Las fórmulas que sí conviene saber

En el cuaderno del facsímil usamos un modelo logístico lineal porque permite ver las tripas sin depender de librerías. No porque todos los sistemas reales sean lineales, sino porque es el punto más honesto para aprender a auditar explicaciones. La formulación viene de la familia de modelos lineales generalizados y aparece en textos estándar de aprendizaje estadístico como Hastie, Tibshirani y Friedman.[13]

El modelo calcula un logit:

$$z(x) = \beta_0 + \sum_{j=1}^{d} \beta_j x_j$$

Y lo convierte en probabilidad con una sigmoide:

$$\hat{p}(x) = \sigma(z) = \frac{1}{1 + e^{-z(x)}}$$

En palabras: primero sumamos señales ponderadas en escala logit y después aplicamos la sigmoide para obtener un valor entre 0 y 1. Ese valor puede leerse como probabilidad estimada solo si el modelo y la calibración lo sostienen; por eso el capítulo anterior fue tan importante.

SÍMBOLO	SIGNIFICADO	EJEMPLO
x	Caso que evaluamos.	Ticket `t001`.
x_j	Valor de la feature j.	`student_wait_days = 12`.
β_j	Peso de la feature j.	0,13.
β_0	Intercepto.	-2,35.
$z(x)$	Suma lineal antes de probabilidad.	1,812.
$\hat{p}(x)$	Probabilidad estimada.	0,8596.

En un modelo lineal aditivo, la contribución local de una feature puede leerse como el término que aporta al logit. Esto no es una métrica nueva inventada para el capítulo: es la descomposición algebraica del predictor lineal $z(x)$. La ventaja pedagógica es que permite comprobar si la explicación que mostramos coincide con los términos que el propio modelo suma.

$$c_j(x) = \beta_j x_j$$

En palabras: si una feature tiene peso positivo y valor alto, empuja el logit hacia arriba; si tiene peso negativo, lo empuja hacia abajo; si vale cero, no aporta en ese caso aunque el peso exista.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$c_j(x)$	Contribución de la feature j al logit.	1,56.
β_j	Peso aprendido o fijado.	0,13.
x_j	Valor del caso.	12 días de espera.

Para `t001`, el cuaderno obtiene:

FEATURE	VALOR	PESO	CONTRIBUCIÓN
`student_wait_days`	12	0,13	1,56
`missing_payment`	1	1,25	1,25
`prior_cases`	3	0,28	0,84

La explicación es clara: el modelo sube prioridad por días de espera, pago pendiente y casos previos. Pero no nos basta con leer esa tabla. Probamos si al quitar la feature superior el score cae de forma relevante.

La prueba de borrado no necesita una ecuación adicional en el texto. Es un procedimiento: calculas el score original, neutralizas una feature, vuelves a calcular el score y miras cuánto cae. En el cuaderno del facsímil, por ejemplo, neutralizar `student_wait_days` en el caso más sensible produce una caída de 0,44. Esa caída no demuestra causalidad por sí sola, pero sí dice: “esta feature sostiene buena parte de esta decisión”.

PASO	QUÉ HACES	QUÉ APRENDES
Score original	Ejecutas el caso tal como llega.	Punto de partida.
Neutralización	Sustituyes una feature por un valor base razonable.	Qué pasa si esa señal desaparece.
Comparación	Miras la caída de score.	Sensibilidad local de la decisión.
Revisión	Compruebas si la feature es legítima y accionable.	Si la explicación se puede defender.

Para importancia global por permutación usamos la idea de romper la asociación entre una feature y la salida, medir cuánto cae el rendimiento y leer esa caída como dependencia del modelo. La importancia por permutación aparece en la tradición de Random Forests de Breiman y se generaliza como *model reliance* en Fisher, Rudin y Dominici.[14][15]

$$I_j = M(D) - M(D_{\pi(j)})$$

En palabras: si al permutar una feature la métrica baja mucho, el modelo dependía de esa feature para rendir. Ojo: esto no prueba causalidad y puede crear combinaciones de datos poco realistas si las features están muy correlacionadas. En ingeniería se usa como señal de dependencia, no como sentencia causal.

SÍMBOLO	SIGNIFICADO	EJEMPLO
I_j	Importancia global de la feature j.	0,25 para `deadline_hours`.
$M(D)$	Métrica original en dataset.	Accuracy 0,75.
$D_{\pi(j)}$	Dataset con la feature j permutada.	`deadline_hours` desordenada.

En el cuaderno, las tres features con mayor caída son:

FEATURE	ACCURACY ORIGINAL	ACCURACY PERMUTADA	CAÍDA
`deadline_hours`	0,75	0,50	0,25
`student_wait_days`	0,75	0,55	0,20
`prior_cases`	0,75	0,60	0,15

Para contrafactuales seguimos la formulación de Wachter, Mittelstadt y Russell: buscar un caso cercano que cambie la decisión.[16]

$$x^{\prime *} = \arg\min_{x'} d(x, x') \quad \text{sujeto a} \quad f(x') \neq f(x)$$

En palabras: queremos el cambio más pequeño que lleve a otra decisión. En producto añadimos una restricción que la fórmula pura no resuelve sola: el cambio tiene que ser accionable, aceptable y compatible con el dominio.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$x^{\prime *}$	Caso contrafactual elegido.	Ticket con documentación adjunta.
$d(x, x')$	Distancia o coste de cambiar de x a x'.	Un cambio accionable.
$f(x')$	Decisión del modelo para el caso modificado.	Pasa de urgente a normal.

Esta fórmula parece limpia, pero en producto tiene una condición escondida: el cambio debe ser accionable y aceptable. No sirve decir “si fueras otra persona” o “si tu historial no existiera”. Sirve decir “si falta documentación, pide la documentación” o “si el pago está pendiente, compruébalo”.

Método no es garantía: cómo elegir bien

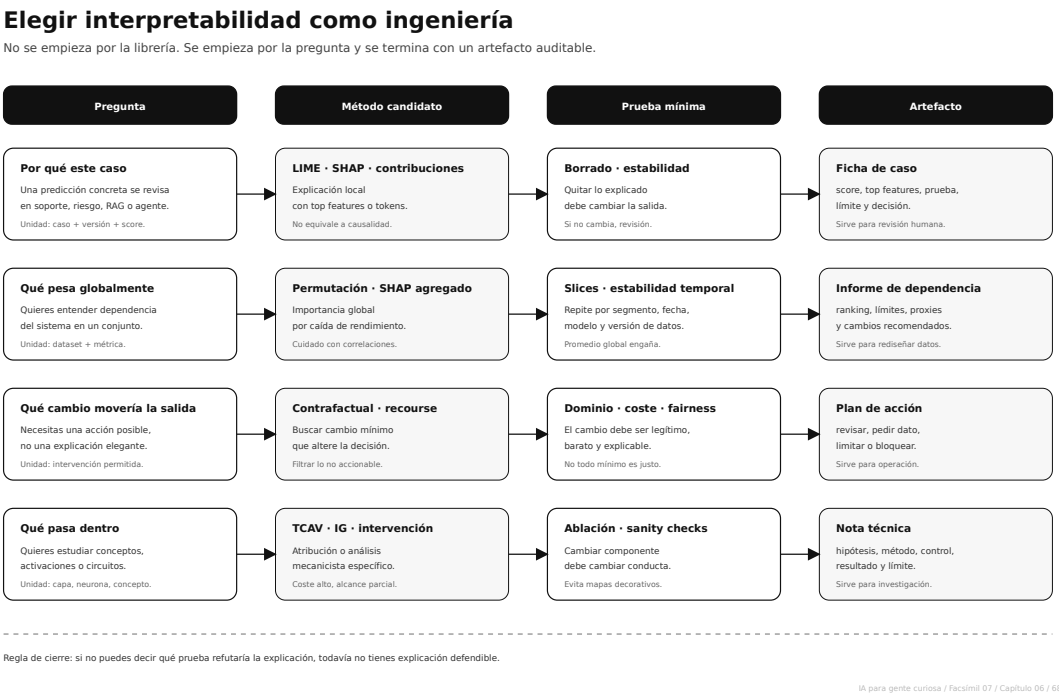
Los métodos de interpretación responden preguntas distintas:

MÉTODO	PREGUNTA QUE RESPONDE	RIESGO
Modelo interpretable	¿Puedo entender el mecanismo completo?	Puede ser demasiado simple para el problema.
LIME	¿Qué modelo simple aproxima esta predicción localmente?	Depende de perturbaciones y vecindario.
SHAP	¿Cómo se reparten contribuciones entre features?	Depende del fondo, correlaciones y coste computacional.
Integrated Gradients	¿Qué entrada aporta al cambio desde una línea base?	La línea base puede cambiar la historia.
Grad-CAM	¿Qué región visual pesa para una clase?	Mapa grueso y sensible a sanity checks.
Contrafactuales	¿Qué cambio produciría otra decisión?	Puede proponer cambios no accionables.
TCAV	¿Qué concepto humano afecta al modelo?	Requiere buenos ejemplos del concepto.
Intervenciones internas	¿Qué componente participa causalmente?	Es costoso, específico y fácil de sobreinterpretar.

La regla práctica:

> No elijas el método por popularidad. Elige el método por la decisión que necesitas tomar.

Si el equipo quiere depurar un clasificador tabular, importancia por permutación y contrafactuales pueden bastar. Si quiere revisar un modelo de visión, Grad-CAM puede ayudar, pero con sanity checks. Si quiere saber si un LLM recupera un hecho por una zona concreta, hacen falta intervenciones más cercanas a interpretabilidad mecanicista. Si la decisión tiene alto impacto, Rudin nos recuerda que quizá el primer debate no sea “cómo explico la caja negra”, sino “por qué no uso un modelo interpretable desde el principio”.



Mapa de decisión para no elegir SHAP, LIME, Grad-CAM o contrafactuales por moda. La pregunta manda; el artefacto final se audita.

Herramientas que verás en equipos

Las herramientas ayudan, pero no sustituyen el contrato. En un equipo real conviene separar tres preguntas: qué tipo de modelo tienes, qué tipo de entrada explicas y qué evidencia necesitas guardar. Un paquete puede generar una visualización excelente y aun así no responder a la pregunta de release.

HERRAMIENTA	ENCAJA MEJOR CUANDO	CUIDADO DE INGENIERÍA
SHAP	Necesitas atribuciones locales o globales y puedes definir bien el fondo de comparación.[17]	El resultado depende del explainer, del background y del coste computacional.
LIME	Quieres aproximar localmente una predicción con un modelo interpretable.[18]	El vecindario y las perturbaciones cambian la explicación.
Captum	Trabajas con PyTorch y quieres Integrated Gradients, saliency, DeepLIFT u otras atribuciones.[19]	Hay que fijar línea base, modalidad y controles.
InterpretML / interpret-community	Trabajas con tabular, glassbox models o explicaciones comparables en notebooks.[20]	Útil para análisis, pero hay que versionar datos y configuración.
Alibi Explain	Necesitas explicaciones locales, contrafactuales o inspección de modelos en flujos Python.[21]	No conviertas un contrafactual sintético en recomendación sin filtro de dominio.
scikit-learn inspection	Quieres importancia por permutación reproducible en modelos clásicos.[22]	La permutación puede crear filas irreales si las variables están correlacionadas.

La herramienta que elegiría para empezar no siempre es la más famosa. Si el problema es tabular y la decisión es sensible, empezaría por un modelo interpretable o por una auditoría de features. Si el sistema es RAG, guardaría trazas y evidencia recuperada antes de pedir una explicación textual. Si el sistema es de visión, usaría mapas visuales solo con sanity checks. Si es un LLM, separaría explicación narrada, traza operativa y evidencia externa.

Cómo evaluar una explicación

Una explicación debe pasar pruebas. No todas son matemáticas sofisticadas; algunas son puro criterio de ingeniería:

PRUEBA	QUÉ COMPRUEBA	SEÑAL MALA
Borrado	Quitar la parte explicada cambia la salida.	La salida no cambia.
Inserción	Añadir features importantes recupera la salida.	Features supuestamente clave no aportan.
Permutación	Desordenar una feature global baja métrica.	Importancia alta sin caída real.
Estabilidad	Perturbaciones pequeñas conservan explicación.	Explicación cambia por ruido menor.
Sanity check	Explicación responde a modelo/datos reales.	Mapa igual con pesos aleatorios.
Revisión de slice	Explicación se sostiene por segmento.	Global bien, segmento mal.
Utilidad humana	La persona decide mejor con la explicación.	Más confianza sin mejor decisión.

Hay una trampa clásica: una explicación muy bonita puede aumentar confianza sin aumentar acierto. Eso es peligroso. En entornos de producto, una explicación debería medirse por decisión: reduce errores, mejora revisión, acelera diagnóstico o permite detectar un problema antes.

En el día a día

En un proyecto de IA, interpretabilidad aparece en cinco momentos:

MOMENTO	PREGUNTA
Diseño	¿Necesitamos modelo interpretable por defecto?
Desarrollo	¿Qué features dominan y cuáles son proxies problemáticos?
Evaluación	¿Las explicaciones se sostienen en fallos y slices?
Producción	¿Podemos explicar una incidencia o una decisión revisada?
Mejora	¿Qué casos se convierten en regresión o cambio de datos?

Un ejemplo cercano: en un asistente RAG, la explicación no debería ser “respondí esto porque el modelo lo consideró relevante”. Debería mostrar:

CAPA	EVIDENCIA
Retrieval	Documentos recuperados, scores, reranker y citas usadas.
Respuesta	Afirmaciones principales y soporte por chunk.
Evaluación	Groundedness, cobertura de cita, abstención y errores.
Calibración	Probabilidad de respuesta aceptada o zona de revisión.
Operación	Modelo, prompt, índice, release y trace id.

En agentes, una explicación útil no es solo el resumen final. Es la trayectoria: qué herramienta eligió, con qué argumentos, qué observó, qué descartó, cuánto costó y dónde pidió revisión.

Por qué debería importarte

La interpretabilidad cambia decisiones concretas de ingeniería. Si una explicación se usa solo para adornar una pantalla, añade ruido. Si se usa bien, permite decidir si una release se publica, si un caso pasa a revisión humana, si una feature debe salir del dataset, si un contrato de salida necesita campos nuevos o si una incidencia de producción se puede depurar sin adivinar.

Esto importa especialmente cuando el sistema empieza a afectar a otras personas. Un score mal explicado puede crear confianza falsa. Un contrafactual mal diseñado puede recomendar una acción imposible. Una explicación global sin slices puede ocultar que el modelo funciona bien en promedio y mal en un grupo concreto de casos. Y una explicación textual de un LLM puede sonar convincente sin ser evidencia del mecanismo que produjo la respuesta.

En un equipo serio, la explicación no vive sola. Vive junto al contrato de datos, la model card, las trazas, los umbrales de calibración, los checks de CI y la política de revisión. Esa es la razón por la que este capítulo no se queda en “qué es SHAP” o “qué es LIME”: el objetivo es que sepas convertir una explicación en un artefacto que alguien pueda revisar, discutir y defender.

Contrato de explicación: quién puede usarla y para qué

Una explicación profesional debería tener contrato. No basta con producir un gráfico o una frase. Hay que declarar para qué sirve, quién puede verla, qué campos son obligatorios, qué versión del modelo la produjo y qué usos no están permitidos.

En el cuaderno del facsímil generamos `output/explanation_contract.json`. Un ejemplo resumido:

```
{
  "model_version": "support-prioritizer-linear-v1",
  "explanation_policy_version": "interp-audit-v1",
  "owner": "equipo-ia",
  "purpose": "diagnostico interno y revision operativa de tickets priorizados",
  "allowed_consumers": ["ingenieria", "soporte_n2", "producto"],
  "not_for": ["decision final sin revision", "comunicacion automatica a usuario"],
  "required_fields": [
    "case_id",
    "model_version",
    "score",
    "prediction",
    "top_features",
    "deletion_test",
    "counterfactual",
    "data_hash_sha256",
    "policy_hash_sha256"
  ]
}
```

La parte importante no es el JSON bonito. Es la disciplina:

CAMPO	QUÉ EVITA
`purpose`	Que una explicación de diagnóstico acabe vendida como verdad final.
`allowed_consumers`	Que cualquier equipo use la explicación sin entender sus límites.
`not_for`	Que se automatice una decisión que exige revisión.
`required_fields`	Que una explicación llegue sin score, versión, prueba o linaje.
`data_hash_sha256`	Que no sepamos con qué datos se generó.
`policy_hash_sha256`	Que cambie el umbral o la política y nadie lo vea.

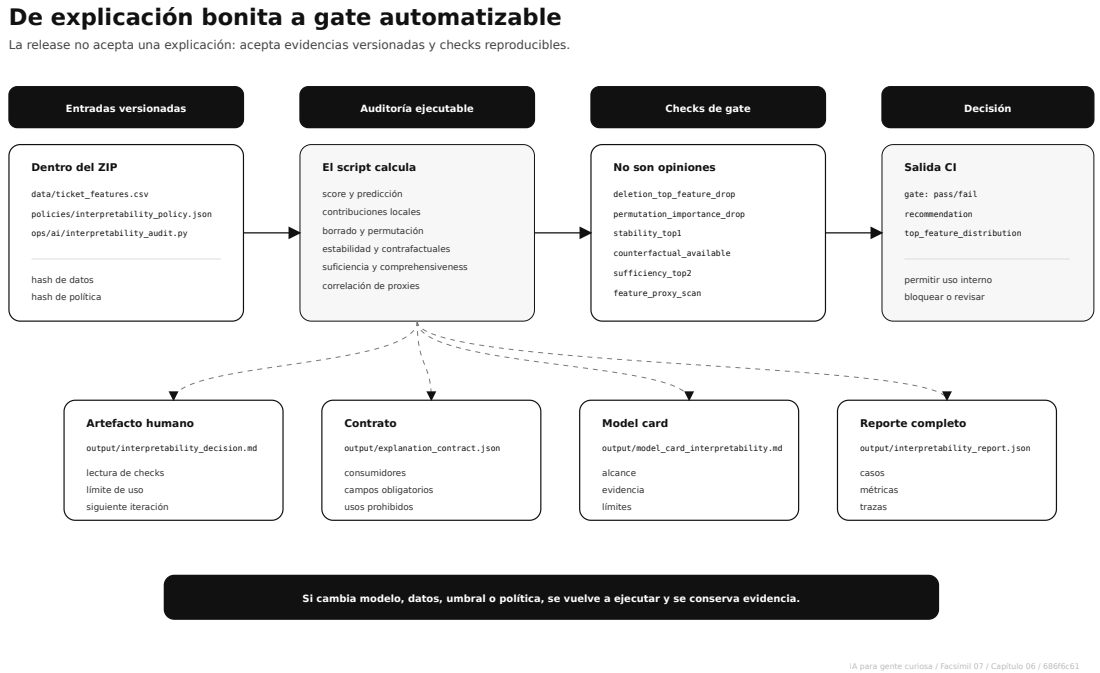
Esto conecta con las model cards, pero baja a operación. Una model card explica el sistema; el contrato de explicación fija cómo se puede consumir una explicación concreta en una run concreta.

Tests de explicación en CI

Si una explicación forma parte de una release, también debería tener tests. No en el sentido de “la explicación es bonita”, sino en el sentido de que pasa checks mínimos antes de publicar una versión.

El cuaderno produce `output/ci_explanation_gate.json`:

```
{
  "gate": "pass",
  "checks": [
    {"name": "deletion_top_feature_drop", "passes": true},
    {"name": "permutation_importance_drop", "passes": true},
    {"name": "stability_top1", "passes": true},
    {"name": "counterfactual_available", "passes": true},
    {"name": "comprehensiveness_top2", "passes": true},
    {"name": "sufficiency_top2", "passes": true},
    {"name": "feature_proxy_scan", "passes": true}
  ],
  "recommendation": "permitir uso interno con monitorización"
}
```



El gate de explicación convierte interpretabilidad en una tubería reproducible: entradas versionadas, checks, contrato, model card y decisión.

Aquí aparecen dos pruebas que conviene conocer bien. En NLP se conocen como *comprehensiveness* y *sufficiency* dentro de trabajos de evaluación de racionales como ERASER.[23] En el cuaderno las adaptamos a features tabulares y a un score probabilístico: no estamos inventando una métrica nueva, estamos usando la misma idea de quitar o conservar la evidencia que la explicación dice que importa.

$$C_K(x) = \hat{p}(x) - \hat{p}(x_{\setminus S_K})$$

En palabras: quitamos las features que la explicación señala como importantes. Si el score apenas cae, la explicación no está capturando señales que sostengan la decisión.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$C_K(x)$	Comprehensiveness para las K features explicadas.	0,210094 para top-2 medio.
S_K	Conjunto de las K features superiores de la explicación.	`student_wait_days`, `missing_payment`.
$x_{\setminus S_K}$	Caso con esas features neutralizadas.	Ticket sin esas dos señales.
$\hat{p}(x)$	Score original del modelo.	0,859603.

Si quitamos las features que la explicación dice que importan y el score no cae, la explicación no está contando algo fuerte.

La suficiencia mira la pregunta contraria:

$$U_K(x) = |\hat{p}(x) - \hat{p}(x_{S_K})|$$

En palabras: dejamos solo las features que la explicación dice que importan. Si con ellas casi reconstruimos el score, la explicación es más suficiente; si no, hay señales importantes que la explicación está escondiendo.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$U_K(x)$	Diferencia entre score original y score usando solo las K features explicadas.	0,063245 para top-2 medio.
x_{S_K}	Caso donde conservamos las features explicadas y neutralizamos el resto.	Ticket reducido a sus dos señales principales.
$\hat{p}(x_{S_K})$	Score usando solo las features explicadas.	Score reconstruido desde top-2.

Si las features explicadas bastan para reconstruir casi todo el score, $U_K(x)$ será bajo. Si no bastan, quizá la explicación ha omitido una señal importante.

En la ejecución actual:

CHECK	RESULTADO	LECTURA
`deletion_top_feature_drop`	0,444993	La feature superior tiene efecto medible.
`permutation_importance_drop`	0,25	Hay features globales con impacto real.
`stability_top1`	0,9667	La explicación local es estable ante perturbación pequeña.
`comprehensiveness_top2`	0,210094	Al quitar las dos features principales, el score cae.
`sufficiency_top2`	0,063245	Las dos features principales aproximan bastante el score.
`feature_proxy_scan`	0,8837	Hay correlación alta entre `prior_cases` y `student_wait_days`; conviene revisarla.

Ese último punto es muy de ingeniería. Una correlación alta no prueba causalidad ni invalida el modelo, pero sí abre una tarea: comprobar si dos features están contando casi lo mismo, si una funciona como proxy de otra o si el dataset necesita rediseño.

Producción: deriva de explicaciones y trazas

En producción no basta con guardar predicciones. Si la explicación influye en revisión, soporte o producto, conviene guardar eventos de explicación:

```
{
  "case_id": "t001",
  "model_version": "support-prioritizer-linear-v1",
  "score": 0.859603,
  "prediction": 1,
  "top_features": ["student_wait_days", "missing_payment", "prior_cases"],
  "data_hash_sha256": "ec7bf...",
  "policy_hash_sha256": "9c167..."
}
```

Con esos eventos podemos medir deriva explicativa. Una forma sencilla es comparar la distribución de feature principal entre dos ventanas mediante distancia de variación total, una distancia estándar entre distribuciones de probabilidad.[24]

$$D_{TV}(P_t, P_{t-1}) = \frac{1}{2} \sum_f |P_t(f) - P_{t-1}(f)|$$

En palabras: sumamos cuánto cambió la proporción de cada razón principal y dividimos por dos para obtener una distancia entre 0 y 1. Si se acerca a 0, las razones dominantes se parecen; si crece, el sistema está explicando por motivos distintos.

SÍMBOLO	SIGNIFICADO	EJEMPLO
$P_t(f)$	Proporción de casos donde la feature f fue la explicación principal en la ventana t.	`deadline_hours = 0,50`.
$P_{t-1}(f)$	La misma proporción en la ventana anterior.	`deadline_hours = 0,40`.
D_{TV}	Distancia de variación total entre distribuciones.	0 significa sin cambio agregado.

En la muestra actual, la distribución de feature principal queda así:

FEATURE PRINCIPAL	PROPORCIÓN
`deadline_hours`	0,50
`student_wait_days`	0,30
`missing_payment`	0,15
`prior_cases`	0,05

Si en la siguiente release `prior_cases` pasa de 0,05 a 0,45, no basta con decir “el accuracy sigue bien”. Algo cambió en la razón operativa de las decisiones. Puede ser un cambio de datos, un cambio de política, un error de feature engineering o una señal nueva real. Hay que investigarlo.

LLMs: explicación textual, traza y mecanismo

En modelos de lenguaje hay una confusión habitual: pedir al modelo que explique su respuesta y tratar esa explicación como mecanismo interno. Son cosas distintas.

NIVEL	QUÉ ES	QUÉ PUEDE APORTAR	QUÉ NO GARANTIZA
Explicación textual	Una justificación generada en lenguaje natural.	Puede ayudar a revisar una respuesta.	No prueba cómo se produjo la salida.
Traza operativa	Prompt, mensajes, herramientas, documentos, scores, coste y eventos.	Permite depurar una run.	No abre las capas internas del modelo.
Evidencia externa	Chunks, citas, resultados de herramientas y validaciones.	Permite comprobar afirmaciones.	No demuestra causalidad interna.
Interpretabilidad mecanicista	Análisis de activaciones, circuitos o intervenciones internas.	Puede estudiar mecanismos concretos.	Es costosa, parcial y no siempre trasladable a producto.

Para ingeniería aplicada, muchas veces la traza vale más que una explicación verbal. Si un agente consulta una herramienta, recupera un documento, cambia una respuesta y pide revisión por score bajo, eso se puede auditar. Si solo dice “he razonado cuidadosamente”, no tenemos suficiente.

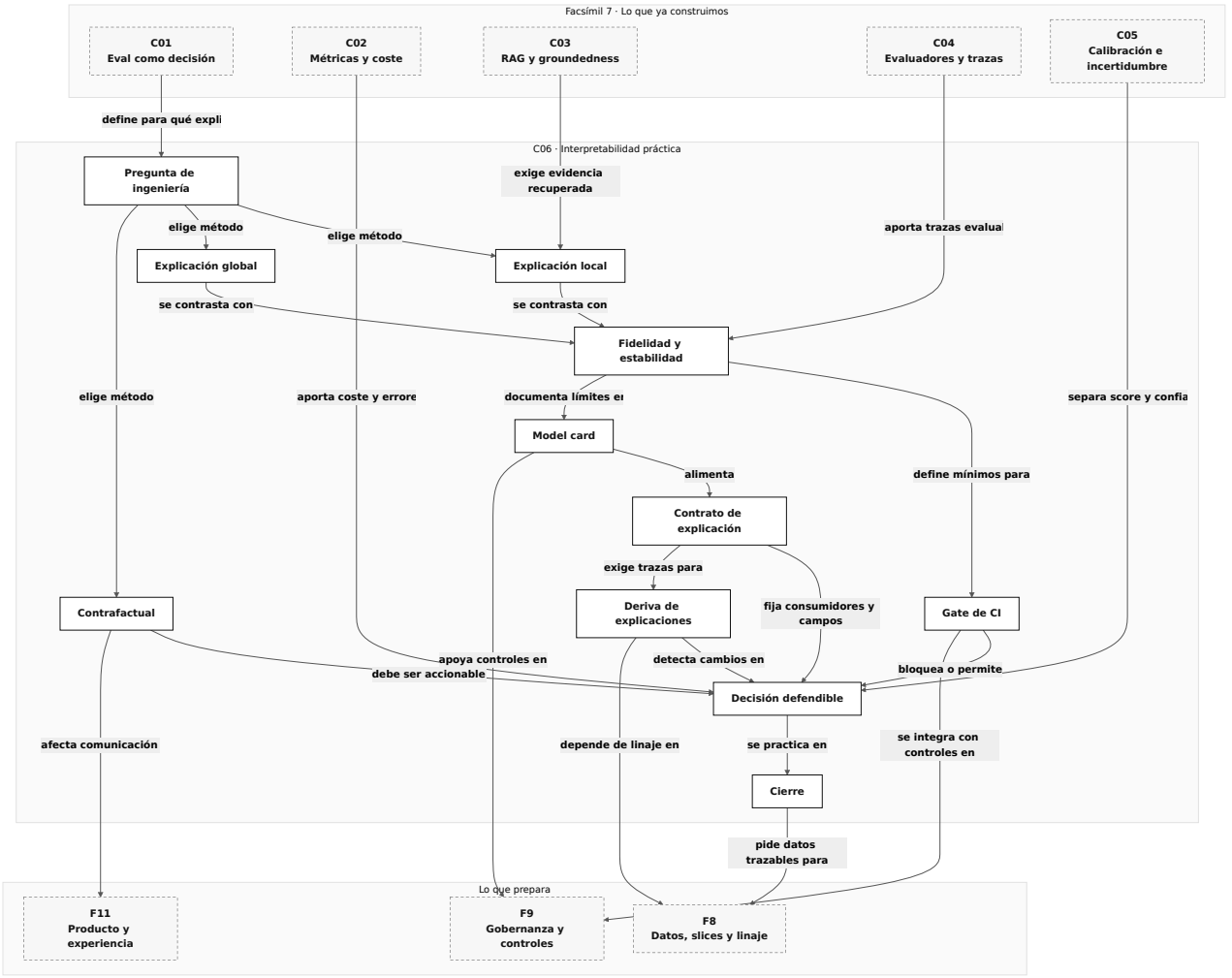
Una regla práctica para alumnos:

> En LLMs, no confundas explicación narrada con evidencia. Guarda trazas, contratos y resultados verificables.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Aceptar explicaciones porque suenan bien	La explicación textual parece convincente.	Separar plausibilidad de fidelidad.
Usar un método para todo	LIME, SHAP o saliency parecen universales.	Empezar por la pregunta de ingeniería.
Enseñar atribuciones sin pruebas	La tabla queda elegante.	Añadir borrado, permutación y estabilidad.
Proponer contrafactuales imposibles	La optimización encuentra cambios absurdos.	Filtrar por accionabilidad y aceptabilidad.
Confundir atención con explicación	Un peso de atención parece intuitivo.	Verificar si cambia la salida y si el mecanismo lo sostiene.
Olvidar los slices	La explicación global tapa segmentos malos.	Auditar por slice y por caso frontera.
No declarar quién puede usar la explicación	El mismo artefacto se usa para diagnóstico, soporte y comunicación externa.	Escribir contrato de explicación.
Dejar explicaciones fuera de CI	La release pasa métricas, pero cambia la lógica explicativa.	Añadir gate con borrado, suficiencia, estabilidad y proxies.
No vigilar deriva de razones	El modelo sigue acertando, pero decide por señales distintas.	Monitorizar distribución de top features.
Tratar proxy como causalidad	Una correlación alta parece una explicación causal.	Revisar pares de features y validar con datos o dominio.
Confundir explicación textual de LLM con mecanismo	La respuesta suena razonada.	Pedir trazas, evidencias y contratos de salida.

Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN BREVE
Interpretabilidad	Capacidad de entender una decisión o comportamiento con una finalidad concreta.
Explicación local	Explicación de una predicción concreta.
Explicación global	Resumen del comportamiento general del modelo.
Fidelidad	Correspondencia entre explicación y comportamiento real del modelo.
Plausibilidad	Facilidad con la que una persona acepta una explicación como razonable.
Atribución	Reparto de una salida entre features, tokens, regiones o componentes.
LIME	Aproximación local de un modelo complejo mediante un modelo interpretable.
SHAP	Marco de atribución basado en valores de Shapley.
Integrated Gradients	Método de atribución que integra gradientes desde una línea base hasta la entrada.
Grad-CAM	Método visual que localiza regiones relevantes para una clase.
Contrafactual	Cambio mínimo de entrada que produciría otra decisión.
Prueba de borrado	Procedimiento que neutraliza una señal explicada para comprobar cuánto cambia la salida.
Importancia por permutación	Caída de métrica al desordenar una feature.
Estabilidad explicativa	Grado en que una explicación conserva sus señales principales ante perturbaciones pequeñas y razonables.
Sanity check	Prueba para detectar explicaciones que no responden al modelo o datos reales.
TCAV	Técnica que mide sensibilidad a conceptos definidos por personas.
Interpretabilidad mecanicista	Estudio de componentes internos y circuitos del modelo mediante análisis e intervenciones.
Contrato de explicación	Acuerdo técnico que fija finalidad, consumidores, campos obligatorios, linaje y usos excluidos.
Comprehensiveness	Prueba que elimina las features explicadas y comprueba cuánto cae el score.

TÉRMINO	DEFINICIÓN BREVE
Suficiencia	Prueba que conserva solo las features explicadas y comprueba cuánto se parece el score al original.
Deriva de explicaciones	Cambio temporal o entre versiones en las razones principales del modelo.
Proxy	Feature que puede representar indirectamente otra variable o mezclar señales que conviene separar.
Model reliance	Dependencia de un modelo respecto a una variable cuando el rendimiento cae al romper su asociación con la salida.
Distancia de variación total	Distancia estándar entre distribuciones de probabilidad que sirve para comparar cambios agregados de razones principales.
Recourse	Cambio accionable que una persona o equipo puede realizar para mover una decisión o resolver un caso.

Antes de pasar página

Antes de cerrar el facsímil, deberías poder responder:

1. ¿Qué diferencia hay entre interpretabilidad, explicación y transparencia?
2. ¿Por qué una explicación plausible puede ser infiel?
3. ¿Cuándo preferirías un modelo interpretable frente a explicar una caja negra?
4. ¿Qué pregunta responde LIME y qué no responde?
5. ¿Qué aporta SHAP y qué supuestos conviene revisar?
6. ¿Por qué Integrated Gradients depende de una línea base?
7. ¿Qué comprobarías antes de confiar en un mapa visual?
8. ¿Qué hace que un contrafactual sea accionable?
9. ¿Cómo usarías pruebas de borrado y permutación?
10. ¿Qué debería entrar en una model card sobre interpretabilidad?
11. ¿Qué campos mínimos pondrías en un contrato de explicación?
12. ¿Qué diferencia hay entre comprehensiveness y suficiencia?
13. ¿Qué señal te daría una deriva de explicaciones?
14. ¿Por qué una correlación alta entre features puede pedir revisión?
15. ¿Qué artefactos debería producir una auditoría de interpretabilidad?
16. ¿Cómo conecta interpretabilidad con EvalOps y calibración?

En resumen

IDEA	QUÉ TE LLEVAS
Interpretar es responder una pregunta situada.	No existe “explicación general” útil para todo.
Plausibilidad no basta.	Una explicación debe probar fidelidad, estabilidad y utilidad.
Cada método tiene límites.	LIME, SHAP, Grad-CAM, contrafactuales y TCAV responden cosas distintas.
Los contrafactuales necesitan criterio humano.	Mínimo no significa accionable ni aceptable.
Las explicaciones se documentan.	Model card, datos, thresholds, checks y decisión.
Las explicaciones también tienen contrato.	Deben declarar finalidad, consumidores, linaje y campos obligatorios.
Las explicaciones se testean.	CI puede revisar borrado, suficiencia, estabilidad, proxies y contrafactuales.
Las explicaciones pueden derivar.	En producción conviene vigilar qué razones dominan cada versión.
El facsímil se cierra con criterio.	El cierre integra métricas, RAG, evaluadores, calibración e interpretación en una sola decisión.

Cuadernos para practicar

Has evaluado sistemas a lo largo del facsímil; estos cuadernos te dejan tocar las dos decisiones que más se equivocan: fiarte de una probabilidad y elegir un umbral. 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.

Calibración: cuando el modelo dice «90%», ¿es un 90%?

Qué practicas: medir si las probabilidades de un modelo significan lo que dicen, y corregirlas. **Dónde encaja:** capítulo 5 (calibración e incertidumbre: de *scores* a decisiones). **Qué necesitas:** un navegador. Corre en CPU; sin claves.

Entrenas un modelo que acierta bastante y luego haces la otra pregunta, la que casi nadie hace: ¿son fiables sus porcentajes? Con un *reliability diagram* comparas lo prometido con lo cumplido y resumes el desajuste en un número, el ECE, que aquí sale

en **0,092** (un modelo optimista que promete más acierto del que cumple). Después lo corriges con *temperature scaling* —dividir los *logits* por una temperatura ajustada— y el ECE baja a **0,025** sin cambiar ni una predicción: solo recalibras la confianza.

Si decides con un umbral («bloquea si supera 0,8»), más te vale que ese 0,8 sea de verdad un 80%. Un modelo descalibrado se equivoca con total aplomo.

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

El coste del error: dónde poner el umbral

Qué practicas: elegir el umbral de decisión por dinero, no por defecto. **Dónde encaja:** capítulo 2 (métricas clásicas: matriz de confusión y coste del error). **Qué necesitas:** un navegador. Corre en CPU; sin claves.

Montas un detector de fraude (clase rara, 8%) y le pones precio a cada error: un fraude que se cuela cuesta 100 €, una transacción legítima bloqueada solo 5. Recorres todos los umbrales y dibujas la curva del coste. El umbral por defecto de 0,5 cuesta **7.735 €**; el óptimo está en **0,07** —lejísimos de 0,5— y cuesta **4.780 €**: casi **3.000 € de ahorro** por mover un número. Como dejar pasar un fraude es veinte veces más caro que una falsa alarma, conviene ser desconfiado.

El acierto es mala guía cuando una clase es rara y un error cuesta más que otro. La métrica honesta es el coste, y el umbral es la palanca: casi nunca vale 0,5.

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

Recursos para seguir: leer, construir y experimentar

Evaluar es decidir. Has visto métricas clásicas, evaluación de RAG, jueces LLM, calibración e interpretabilidad. Aquí tienes por dónde seguir practicándolo.

Para experimentar sin código. En las cajas «Pruébalo en 5 minutos» lo tocaste: MLU-Explain de Amazon (`mlu-explain.github.io`) para mover el umbral de una matriz de confusión y ver precisión y recall pelearse; y los playgrounds (`platform.openai.com` , `aistudio.google.com` , `console.anthropic.com`) para comprobar a mano la fidelidad de un RAG, el ruido y los sesgos de un juez LLM, y la calibración de las confianzas de un modelo.

Para construir. Para evaluar en serio: OpenAI Evals y Promptfoo para montar suites de casos y comparar prompts y modelos; Ragas para métricas de RAG; y plataformas como LangSmith para datasets, trazas y comparación de runs. Empieza con un runner propio pequeño para entender el contrato antes de adoptar una plataforma.

Para leer. Cada capítulo enlaza sus fuentes en «Para saber más»; las ideas clave son la matriz de confusión y el coste del error, la fidelidad y la abstención en RAG, y la calibración. Convertir todo eso en un gate de release defendible es lo que separa «la métrica salió alta» de «puedo justificar por qué publico o bloqueo».

Para saber más

Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M. y Kim, B. (2018). Sanity Checks for Saliency Maps. *Advances in Neural Information Processing Systems*. https://papers.nips.cc/paper/2018/hash/294a8ed24b1ad22ec2e7efea049b8737-Abstract.html

Breiman, L. (2001). Random Forests. *Machine Learning*, 45, 5-32. https://doi.org/10.1023/A:1010933404324

Captum. (2026). *Model Interpretability for PyTorch*. https://captum.ai/

Cover, T. M. y Thomas, J. A. (2006). *Elements of Information Theory* (2.ª ed.). Wiley.

DeYoung, J., Jain, S., Rajani, N. F., Lehman, E., Xiong, C., Socher, R. y Wallace, B. C. (2020). ERASER: A Benchmark to Evaluate Rationalized NLP Models. *Proceedings of ACL*. https://aclanthology.org/2020.acl-main.408/

Doshi-Velez, F. y Kim, B. (2017). *Towards A Rigorous Science of Interpretable Machine Learning*. arXiv. https://arxiv.org/abs/1702.08608

Fisher, A., Rudin, C. y Dominici, F. (2019). All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. *Journal of Machine Learning Research*, 20(177), 1-81. https://jmlr.org/papers/v20/18-760.html

Hastie, T., Tibshirani, R. y Friedman, J. (2009). *The Elements of Statistical Learning* (2.ª ed.). Springer. https://web.stanford.edu/~hastie/ElemStatLearn/

InterpretML. (2026). *InterpretML documentation*. https://interpret.ml/

Jacovi, A. y Goldberg, Y. (2020). Towards Faithfully Interpretable NLP Systems: How Should We Define and Evaluate Faithfulness? *Proceedings of ACL*. https://aclanthology.org/2020.acl-main.386/

Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viégas, F. y Sayres, R. (2018). Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors. *ICML*. https://arxiv.org/abs/1711.11279

Lipton, Z. C. (2018). The Mythos of Model Interpretability. *Communications of the ACM*, 61(10), 36-43. https://doi.org/10.1145/3233231

Lundberg, S. M. y Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems*. https://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions

Meng, K., Bau, D., Andonian, A. y Belinkov, Y. (2022). Locating and Editing Factual Associations in GPT. *Advances in Neural Information Processing Systems*. https://papers.nips.cc/paper_files/paper/2022/hash/6f1d43d5a82a37e89b0665b33bf3a182-Abstract-Conference.html

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D. y Gebru, T. (2019). Model Cards for Model Reporting. *FAT*, 220-229. https://doi.org/10.1145/3287560.3287596

Ribeiro, M. T., Singh, S. y Guestrin, C. (2016). Why Should I Trust You? Explaining the Predictions of Any Classifier. *KDD*, 1135-1144. https://doi.org/10.1145/2939672.2939778

Rudin, C. (2019). Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, 1, 206-215. https://doi.org/10.1038/s42256-019-0048-x

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. y Batra, D. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *ICCV*. https://doi.org/10.1109/ICCV.2017.74

SHAP. (2026). *SHAP documentation*. https://shap.readthedocs.io/

scikit-learn. (2026). *Permutation feature importance*. https://scikit-learn.org/stable/modules/permutation_importance.html

Seldon. (2026). *Alibi Explain documentation*. https://docs.seldon.ai/alibi-explain

Sundararajan, M., Taly, A. y Yan, Q. (2017). Axiomatic Attribution for Deep Networks. *ICML*, 3319-3328. https://proceedings.mlr.press/v70/sundararajan17a.html

Tabassi, E. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.100-1

Wachter, S., Mittelstadt, B. y Russell, C. (2017). Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *Harvard Journal of Law and Technology*, 31, 841-887. https://arxiv.org/abs/1711.00399

Notas

1. Lipton, Z. C. (2018). The Mythos of Model Interpretability. *Communications of the ACM*, 61(10), 36-43. https://doi.org/10.1145/3233231 ↩

2. Doshi-Velez, F. y Kim, B. (2017). *Towards A Rigorous Science of Interpretable Machine Learning*. arXiv. https://arxiv.org/abs/1702.08608 ↩

3. Jacovi, A. y Goldberg, Y. (2020). Towards Faithfully Interpretable NLP Systems: How Should We Define and Evaluate Faithfulness? *ACL*. https://aclanthology.org/2020.acl-main.386/ ↩

4. Ribeiro, M. T., Singh, S. y Guestrin, C. (2016). Why Should I Trust You? Explaining the Predictions of Any Classifier. *KDD*, 1135-1144. https://doi.org/10.1145/2939672.2939778 ↩

5. Lundberg, S. M. y Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. *NeurIPS*. https://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions ↩

6. Sundararajan, M., Taly, A. y Yan, Q. (2017). Axiomatic Attribution for Deep Networks. *ICML*, 3319-3328. https://proceedings.mlr.press/v70/sundararajan17a.html ↩

7. Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. y Batra, D. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *ICCV*. https://doi.org/10.1109/ICCV.2017.74 ↩

8. Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M. y Kim, B. (2018). Sanity Checks for Saliency Maps. *NeurIPS*. https://papers.nips.cc/paper/2018/hash/294a8ed24b1ad22ec2e7efea049b8737-Abstract.html ↩

9. Wachter, S., Mittelstadt, B. y Russell, C. (2017). Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *Harvard Journal of Law and Technology*, 31, 841-887. https://arxiv.org/abs/1711.00399 ↩

10. Rudin, C. (2019). Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, 1, 206-215. https://doi.org/10.1038/s42256-019-0048-x ↩

11. Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viégas, F. y Sayres, R. (2018). Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors. *ICML*. https://arxiv.org/abs/1711.11279 ↩

12. Meng, K., Bau, D., Andonian, A. y Belinkov, Y. (2022). Locating and Editing Factual Associations in GPT. *NeurIPS*. https://papers.nips.cc/paper_files/paper/2022/hash/6f1d43d5a82a37e89b0665b33bf3a182-Abstract-Conference.html ↩

.3. Hastie, T., Tibshirani, R. y Friedman, J. (2009). *The Elements of Statistical Learning* (2.ª ed.). Springer. https://web.stanford.edu/~hastie/ElemStatLearn/ ↩

.4. Breiman, L. (2001). Random Forests. *Machine Learning*, 45, 5-32. https://doi.org/10.1023/A:1010933404324 ↩

.5. Fisher, A., Rudin, C. y Dominici, F. (2019). All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. *Journal of Machine Learning Research*, 20(177), 1-81. https://jmlr.org/papers/v20/18-760.html ↩

.6. Wachter, S., Mittelstadt, B. y Russell, C. (2017). Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *Harvard Journal of Law and Technology*, 31, 841-887. https://arxiv.org/abs/1711.00399 ↩

.7. SHAP Documentation. (2026). https://shap.readthedocs.io/ ↩

.8. Ribeiro, Singh y Guestrin, 2016. ↩

.9. Captum. (2026). Model Interpretability for PyTorch. https://captum.ai/ ↩

:0. InterpretML. (2026). https://interpret.ml/ ↩

:1. Seldon. (2026). Alibi Explain documentation. https://docs.seldon.ai/alibi-explain ↩

:2. scikit-learn. (2026). Permutation feature importance. https://scikit-learn.org/stable/modules/permutation_importance.html ↩

:3. DeYoung, J., Jain, S., Rajani, N. F., Lehman, E., Xiong, C., Socher, R. y Wallace, B. C. (2020). ERASER: A Benchmark to Evaluate Rationalized NLP Models. *Proceedings of ACL*. https://aclanthology.org/2020.acl-main.408/ ↩

:4. Cover, T. M. y Thomas, J. A. (2006). *Elements of Information Theory* (2.ª ed.). Wiley. ↩