Facsímil 07

Evaluar, calibrar e interpretar

Capítulo 01

Qué es una eval y qué decisión permite tomar

686f6c61 · 2026

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.

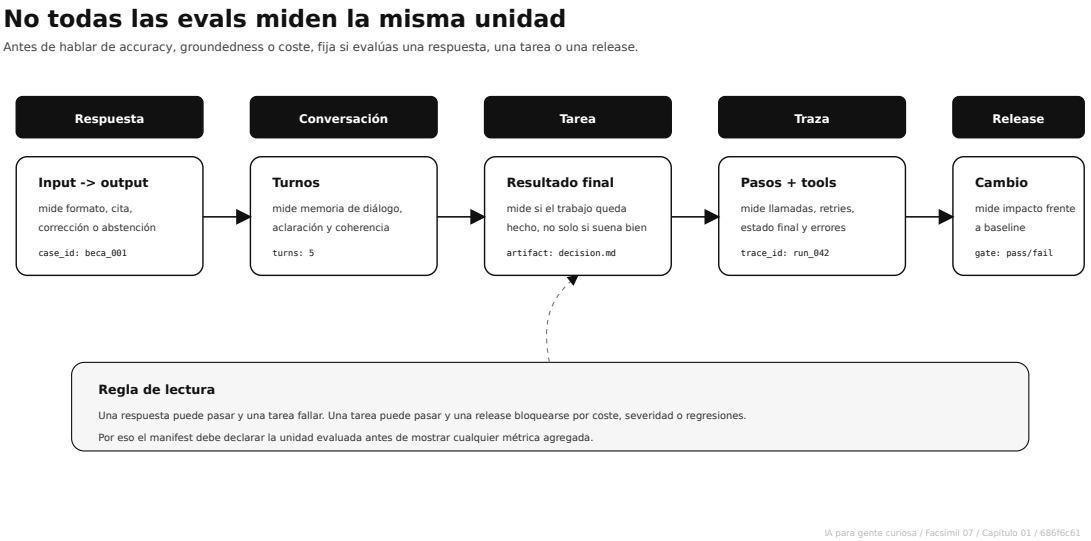
Unidad de evaluación: qué estás midiendo exactamente

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]

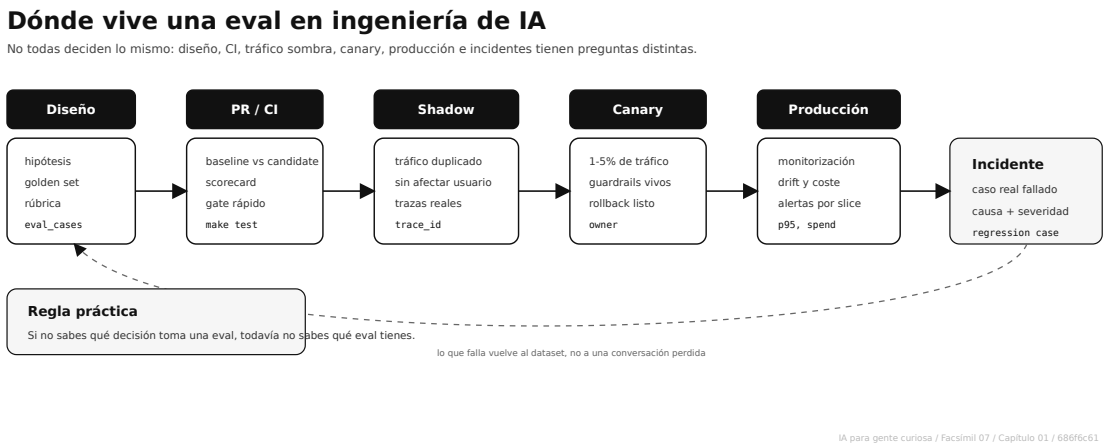
La conclusión práctica es sencilla: el mercado tiene herramientas distintas, pero la anatomía se repite. Necesitas casos, tarea, evaluadores, métricas, trazas, comparación y decisión.

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.

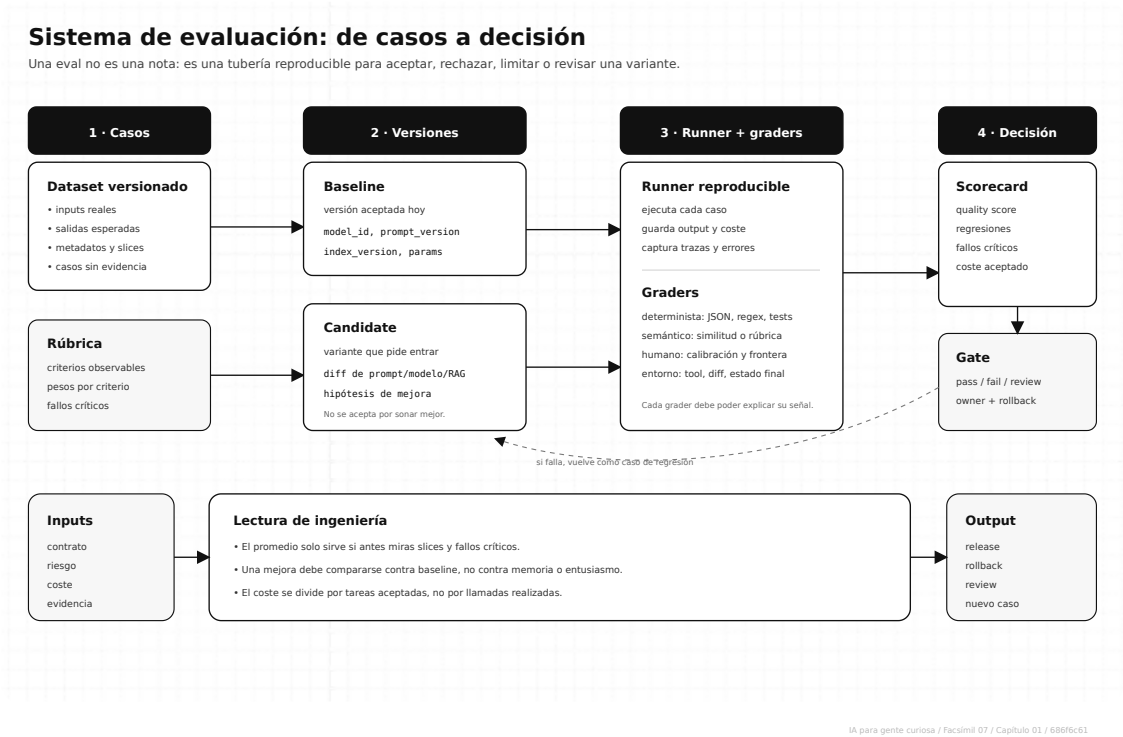
Este capítulo se centra sobre todo en la eval offline con gate de release, porque es la primera que necesitas para trabajar con orden. Pero no la veas como una isla: el mismo dataset puede crecer con incidentes de producción, el mismo manifest puede alimentar auditoría, y el mismo gate puede acabar en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

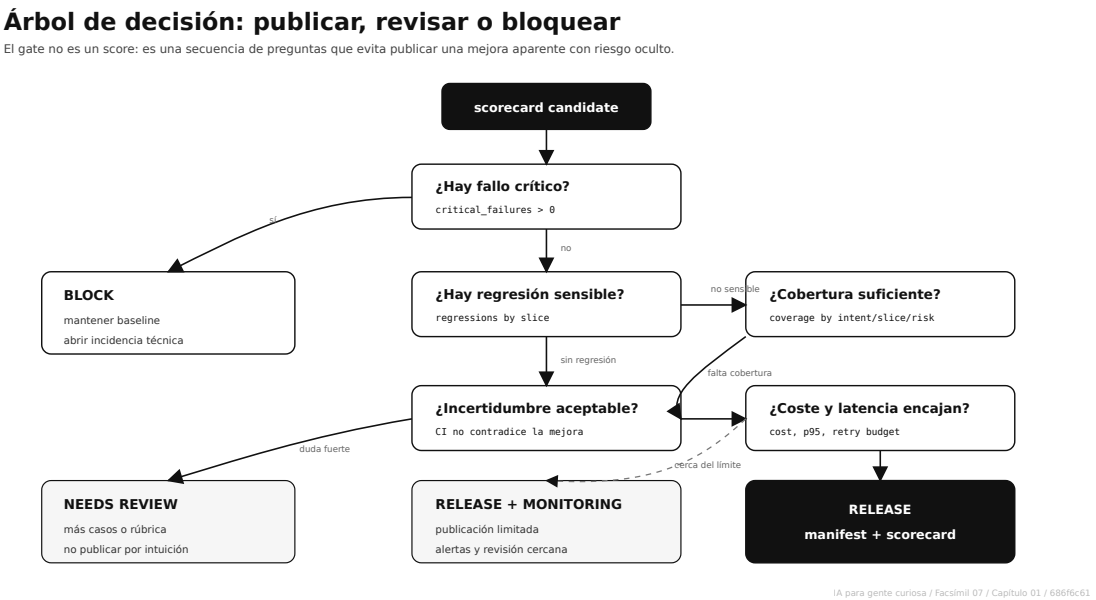
```
aceptar si:
  quality_score >= quality_threshold
  critical_failures == 0
  cost_per_accepted <= cost_budget
```

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**.

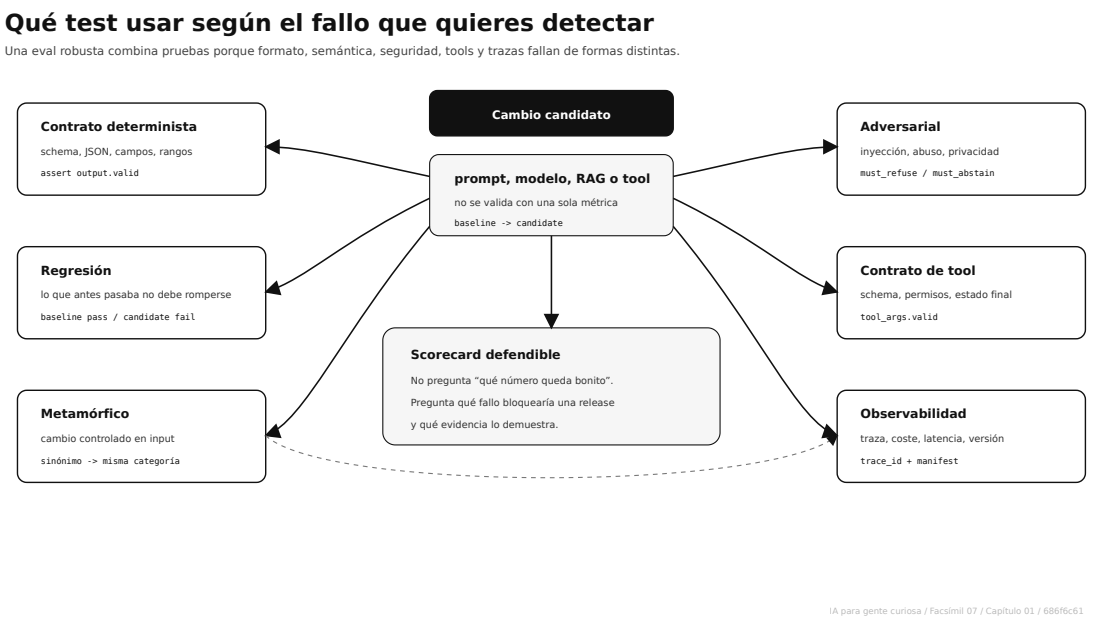
Familias de tests de ingeniería para evals

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.



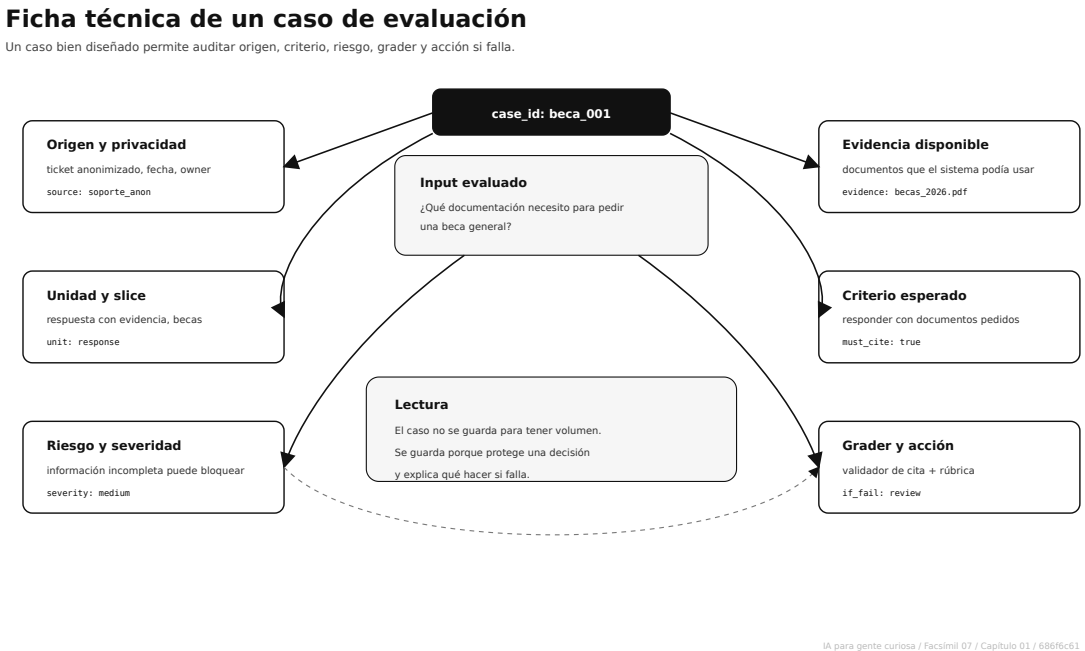
Familias de tests: cada prueba existe para detectar un tipo de fallo y sostener una decisión concreta.

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.

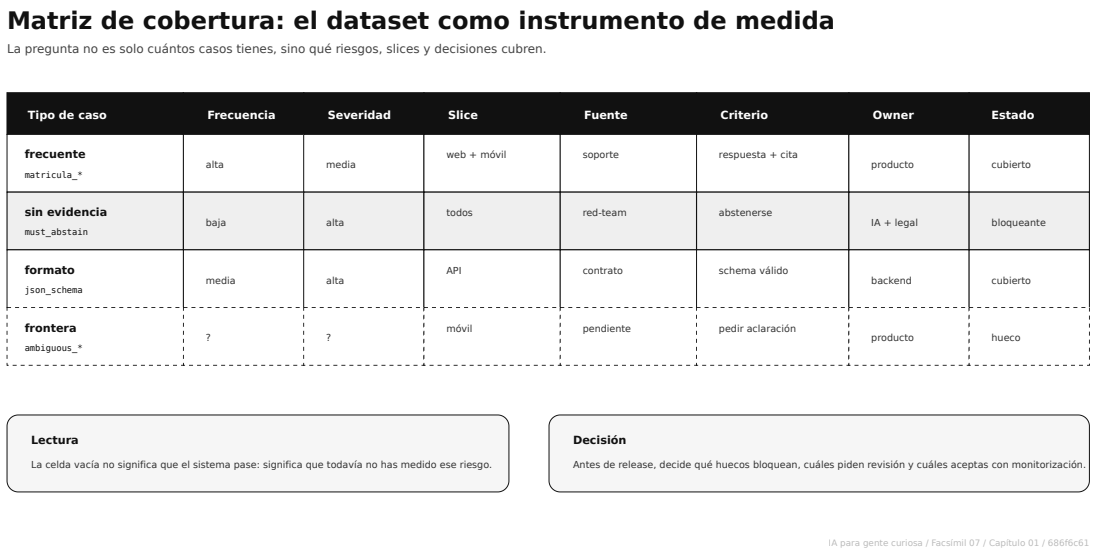
La documentación de modelos también importa. Las model cards nacen para registrar detalles de uso previsto, factores, métricas, datos de evaluación y consideraciones de despliegue.[11] En un sistema aplicado, la scorecard de eval cumple una función parecida para una release concreta: dice qué hemos probado,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

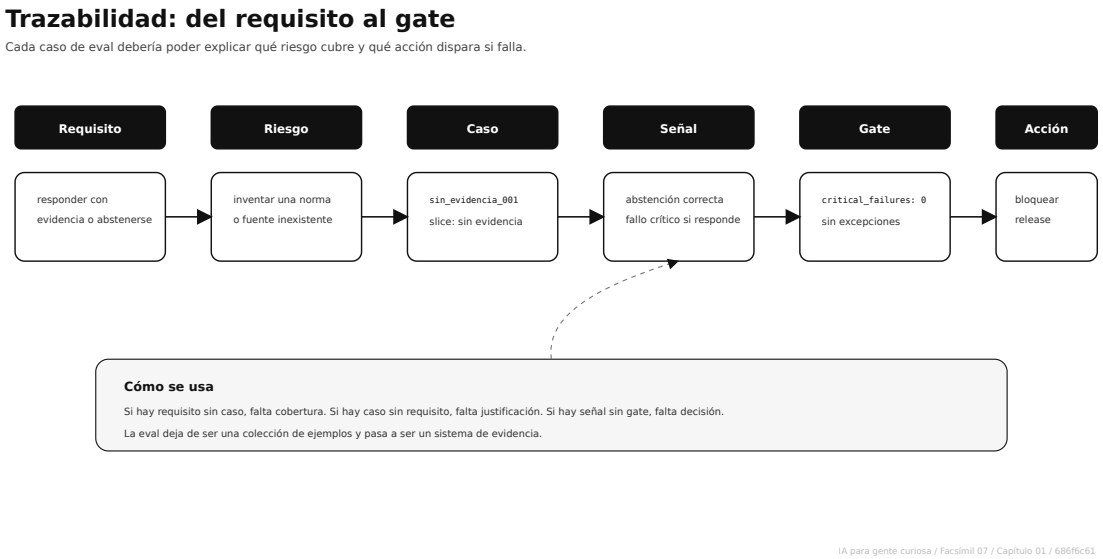
Matriz de trazabilidad: por qué existe cada caso

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.

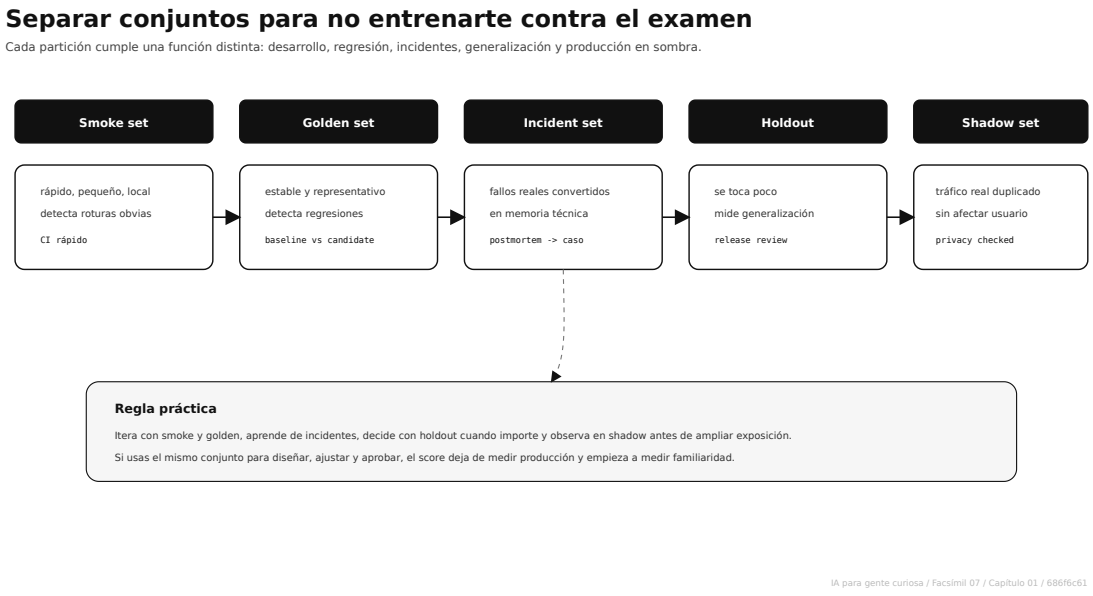
Cómo no sobreajustar contra tu propia eval

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.

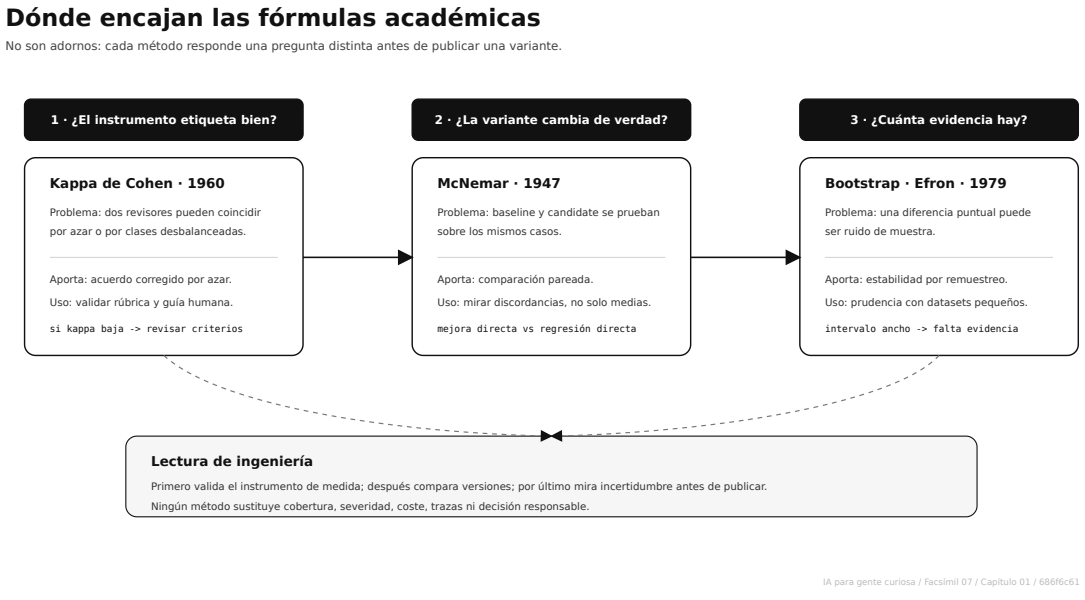
Fórmulas académicas que sí aparecen

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

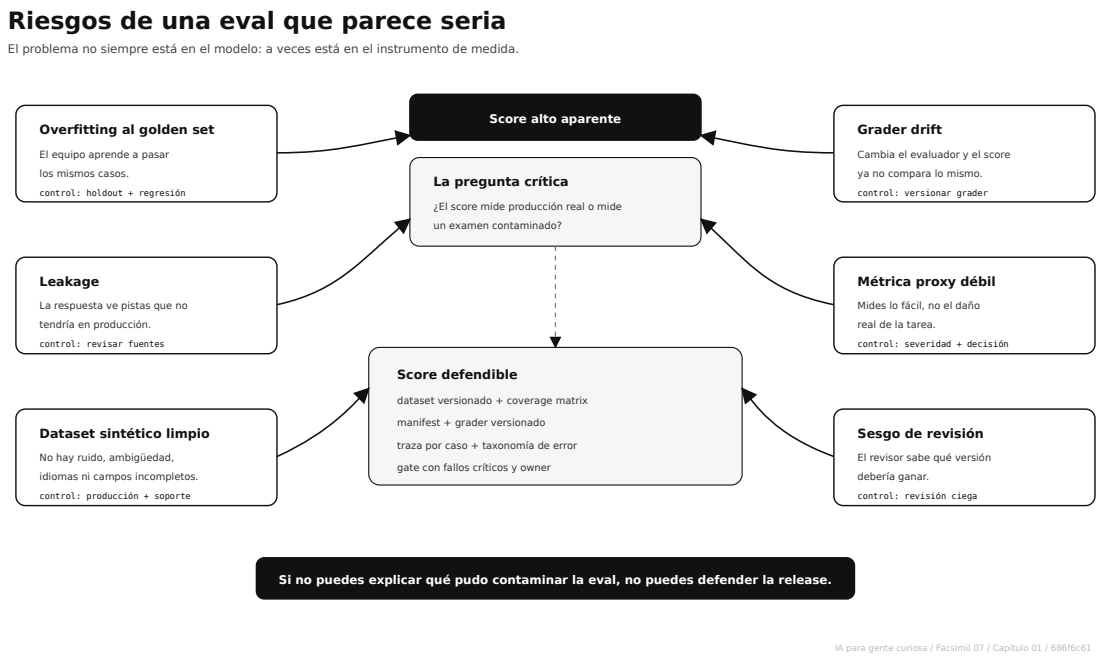
Una eval puede tener JSON, dashboards y fórmulas y aun así estar mal diseñada. El problema no es que falte tecnología; es que el instrumento de medida se haya contaminado. En ingeniería de IA esto pasa mucho porque los equipos iteran rápido: miran los fallos, ajustan prompt, vuelven a mirar los mismos 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.

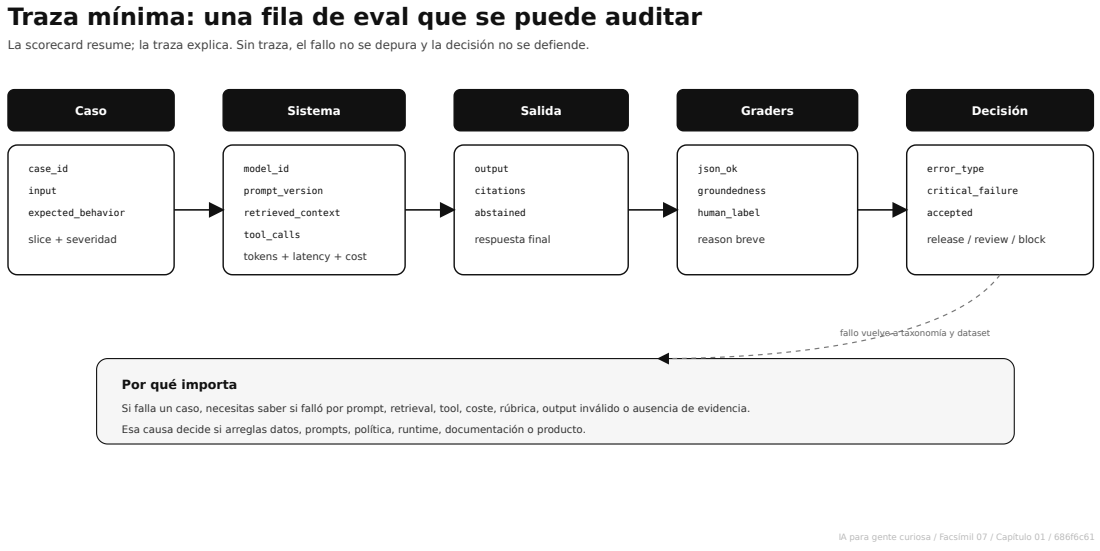
PASO	QUÉ OCURRE	ARTEFACTO ESPERADO
Cambio	Alguien modifica prompt, modelo, RAG, tool o política.	Diff revisable.
Ejecución	El runner evalúa baseline y candidate con el mismo dataset.	`eval_scorecard.json`.
Gate	Se aplican umbrales, fallos críticos, coste y regresiones.	Decisión `release`, `needs_review` 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

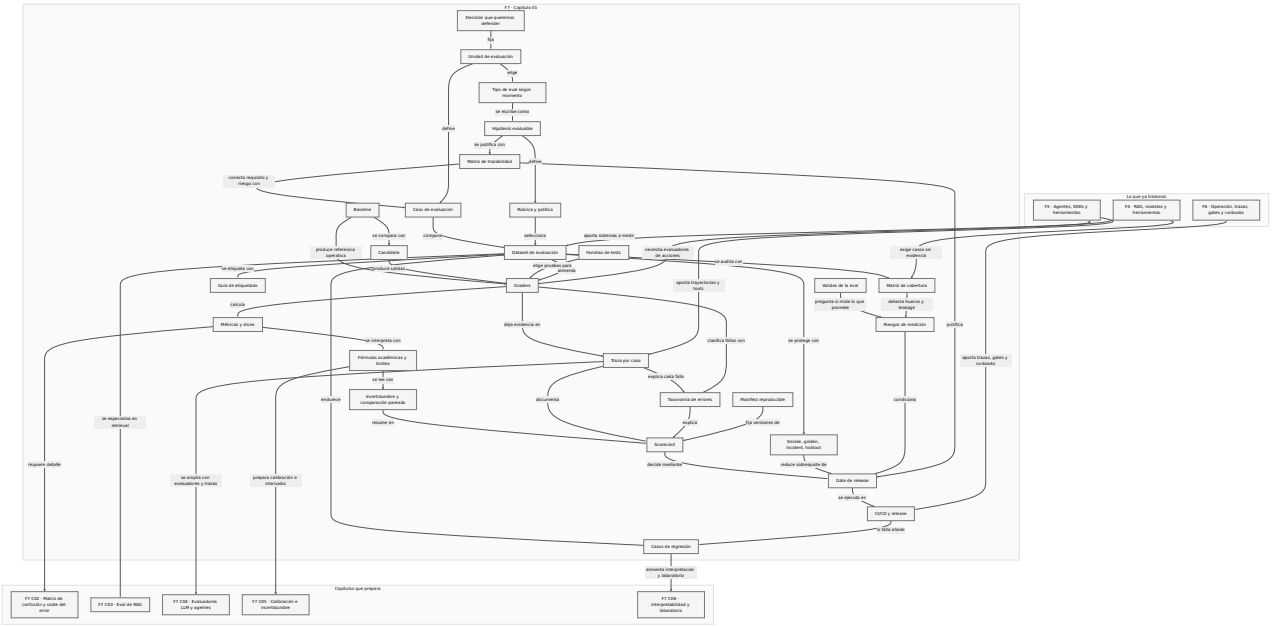
Porque una eval te protege de publicar por sensación. En sistemas de IA, una demo buena suele enseñar lo que el sistema puede hacer; una eval útil enseña lo que todavía puede romper. Esa diferencia cambia conversaciones enteras: en vez de discutir si “parece mejor”, el equipo mira casos, regresiones,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