Facsímil 07

Evaluar, calibrar e interpretar

Capítulo 04

Evaluadores LLM y agentes: rúbricas, trazas y coste

686f6c61 · 2026

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.

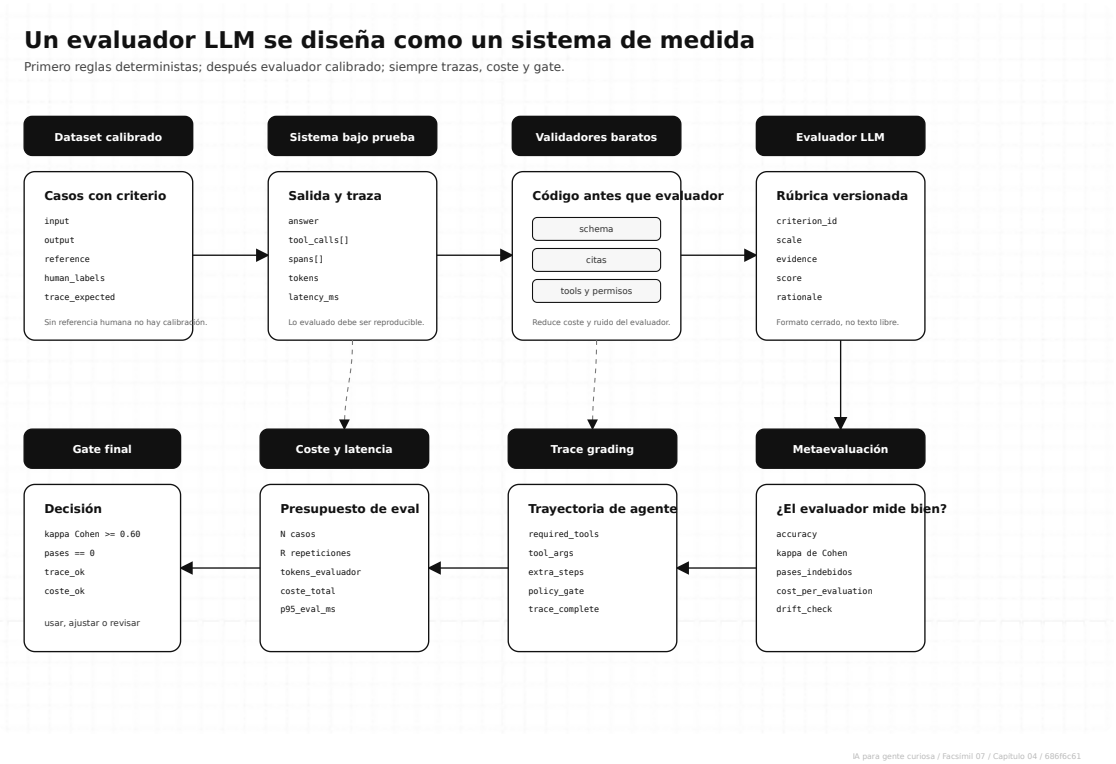
En castellano usaré **evaluador LLM**. En documentación y papers aparece a menudo como `LLM-as-a-judge`; lo citaremos así cuando sea el nombre técnico de la fuente, pero en el cuerpo del capítulo hablaremos de evaluadores.

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

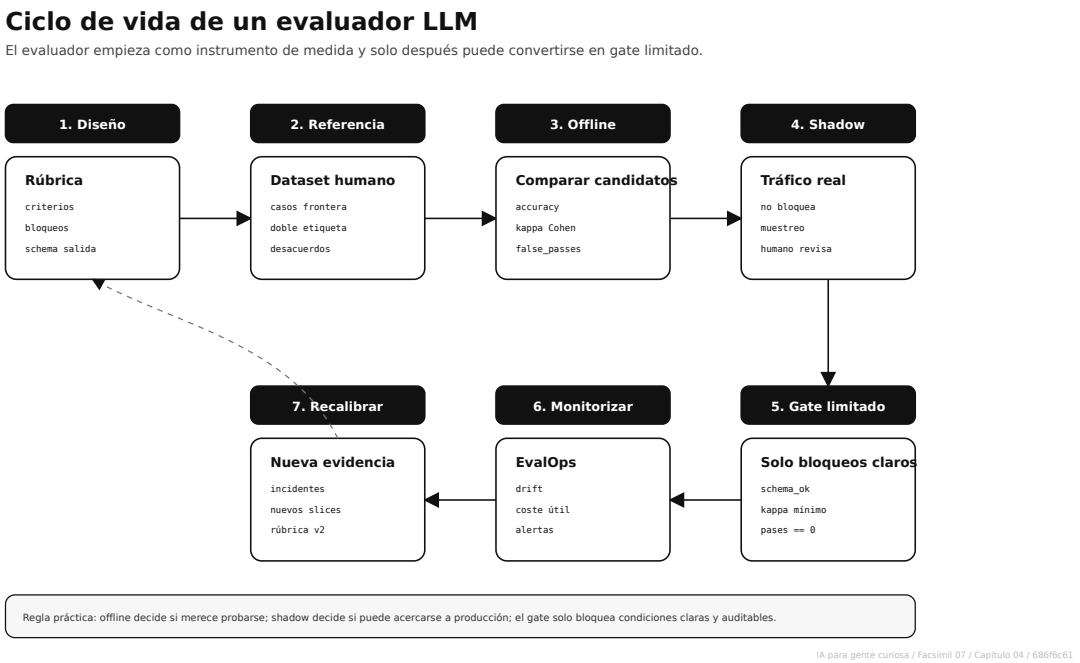
Antes de usar un evaluador como gate, pediría esto:

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**.

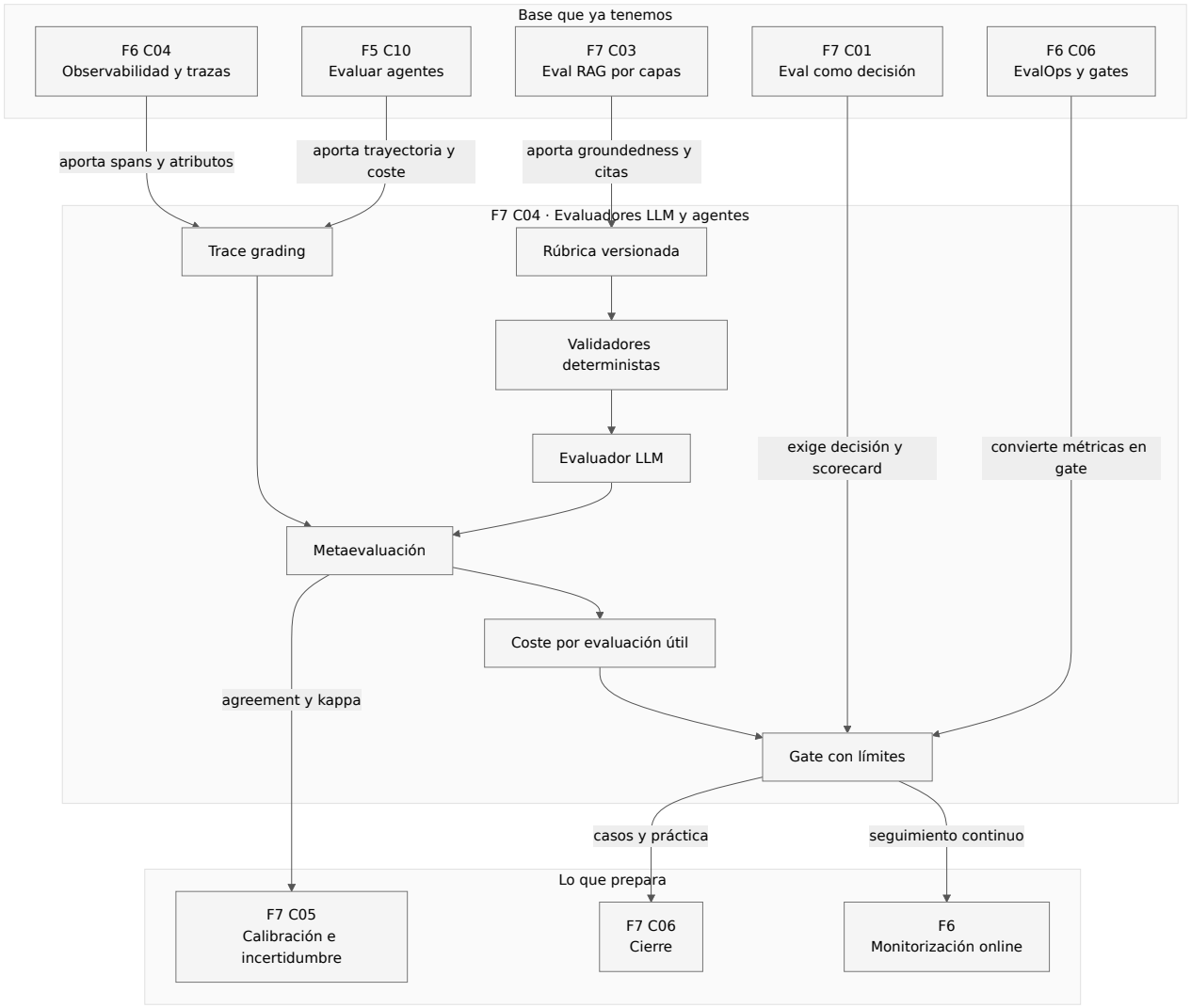
Por qué debería importarte

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. ↩