TR-01 | Technical report | VERSION 0.2 | AUGUST 2026

Evaluación orientada a fallas con LLM-as-a-Judge para un agente transaccional: estudio de una corrida de 46 casos

Pablo Cardozo
Investigador independiente
Buenos Aires, Argentina

Resumen

Este informe audita un harness LLM-as-a-Judge para un asistente transaccional de restaurantes. El artefacto guardado contiene 46 casos y registra 40 aprobados con umbral 75/100, score promedio 80, 671,3 segundos totales y 28.010 tokens del juez. El desempeño no es uniforme: fallan los cinco casos de pago y un caso FAQ; las otras diez categorías reportan aprobación completa o casi completa. La concentración de fallas es más útil que el 87% agregado porque identifica un defecto de ruteo de estado en la frontera de pagos. La auditoría encuentra además tres límites importantes. Primero, los casos “meta” de seguridad y resiliencia se envían como mensajes al chatbot y se puntúan por plausibilidad conversacional; no ejecutan el comportamiento HTTP, rate-limit, retry o circuit breaker que nombran. Segundo, la batería versionada en la revisión fuente contiene casos adicionales ausentes en la salida de 46, por lo que la corrida no puede tratarse como ejecución limpia del árbol actual. Tercero, no hay calibración humana, semillas repetidas ni reejecución independiente. La verificación local del 4 de agosto de 2026 confirmó 36 assertions de la batería y 43 del asistente, pero Vitest no terminó limpiamente porque quedaron handles abiertos; no se repitió la corrida end-to-end del juez. Por eso los scores se tratan como evidencia del repositorio, no como verdad de terreno, y se propone una evaluación en tres capas: checks deterministas para estado y seguridad, revisión con modelo para diálogo abierto y calibración humana ciega.

Palabras clave: LLM-as-a-Judge; agentes conversacionales; diálogo transaccional; análisis de fallas; validez de evaluación; observabilidad

ESTADO DE LA EVIDENCIA	Corrida guardada de 46 casos inspeccionada; se documentan límites y vacíos de procedencia
GRADO DE EVIDENCIA	Grado R — resultado del repositorio, sin reejecución end-to-end independiente
REVISIÓN FUENTE	8467633188cd
REPRODUCIDO / AUDITADO	2026-08-04

Datos rastreables

CORRIDA GUARDADA	40/46 · promedio 80/100
CLUSTER DE FALLAS	5 pagos · 1 FAQ
EJECUCIÓN	671,3 s · 28.010 tokens
CHECK LOCAL	79 assertions; proceso no cerró

Contenido

> **LÍMITE EDITORIAL**
> Las afirmaciones quedan acotadas a la revisión citada y al grado de evidencia. Las mediciones ausentes se declaran como ausentes; no se infieren de la calidad de implementación.

1. Motivación y contribución

Los agentes transaccionales son más difíciles de evaluar que un sistema de pregunta-respuesta porque su corrección depende del estado. Una respuesta de pago puede ser fluida y estar mal porque se perdió el total; un handoff puede sonar empático y no persistir la transferencia; un menú puede ser correcto para el catálogo pero irrelevante para el paso actual. Las métricas por string son débiles para estas respuestas abiertas, mientras la revisión manual es cara y difícil de repetir después de cada cambio.

El repositorio auditado implementa un modelo juez separado, batería dinámica, runner stateful y trazas en Langfuse [1,3–5]. Una corrida guardada del 8 de marzo de 2026 aporta evidencia de fallas: casos individuales, cinco criterios, latencia, tokens y razones textuales [2]. Este informe reconstruye el experimento y examina la validez de sus conclusiones.

- Describir batería, ejecución, rúbrica y agregación desde el código.
- Reportar todas las categorías y analizar las seis fallas, no solo el 87% agregado.
- Auditar si cada categoría ejercita realmente la propiedad que dice medir.
- Documentar la deriva entre la corrida de 46 casos y la batería más grande de la revisión registrada.
- Proponer una arquitectura calibrada que separe propiedades deterministas, calidad abierta y juicio humano.

2. Evaluación con LLMs y sesgos conocidos

MT-Bench y Chatbot Arena establecieron LLM-as-a-Judge como aproximación escalable a preferencias humanas y documentaron sesgos de posición, verbosidad, auto-preferencia y razonamiento [7]. G-Eval mostró que una rúbrica estructurada puede mejorar correlación humana, pero también señaló preferencia potencial por texto generado por modelos [8]. Prometheus 2 amplía este enfoque a evaluación directa y comparación por pares [9]. Estos trabajos respaldan el uso del juez como instrumento; ninguno convierte un score sin calibración en verdad de terreno.

El harness usa evaluación directa. Entrega conversación, respuesta final, contexto seleccionado del catálogo y categoría a un único modelo. Es útil operacionalmente porque produce explicaciones y resúmenes. Su debilidad científica es la meta-evaluación: no existe un estudio local de acuerdo humano, repetibilidad entre semillas ni sensibilidad a prompts y modelos. Debe interpretarse como instrumento de screening que genera hipótesis de falla.

Evaluador	Adecuado para	Mal ajuste
Assertion determinista	Aritmética, estado, auth, schema, side effects	Tono y naturalidad
Modelo juez	Relevancia, completitud, tono, respuesta abierta	Garantías de seguridad y efectos ocultos
Evaluador humano	Calibración, ambigüedad, adecuación	Regresión frecuente en soledad
Telemetría	Latencia, tokens, errores, atribución	Calidad semántica sin labels

Tabla 1. Función de cada evaluador.

3. Sistema evaluado y arquitectura del harness

El sistema evaluado es un asistente con respuestas basadas en catálogo, pedidos, pagos, estado persistente, handoff y lógica de resiliencia [1]. El runner obtiene un snapshot autenticado del catálogo, genera casos con productos y FAQ disponibles, crea un chatId único, envía cada mensaje en secuencia a /message y acumula respuestas [3,5]. Solo se puntúa la respuesta final; los turnos anteriores forman parte del contexto del juez.

1. Obtener el catálogo vivo con token de administrador.
2. Generar una batería fundada en productos actuales.
3. Crear un identificador nuevo por caso para reducir contaminación.
4. Ejecutar todos los mensajes del caso contra el endpoint.
5. Construir contexto con turnos previos y separar la respuesta final.
6. Invocar juez, aplicar umbral, adjuntar trazas y agregar por categoría.

Nivel	Campos
Caso	ID, categoría, mensajes, expectativa, respuestas, label, razón
Juez	Overall, relevancia, precisión, completitud, tono, accionabilidad
Operación	Latencias, tokens y trazas del SUT y juez
Categoría	Cantidad, aprobados, medias, p95 y tokens
Corrida	Timestamp, totales, pass rate, media, duración, p95 y tokens

Tabla 2. Campos registrados.

> **RESET DE ESTADO**
> Un chatId único aísla conversaciones. No resetea catálogo global, cachés, comportamiento del modelo ni servicios externos.

4. Diseño de la batería y cobertura

El artefacto guardado contiene 46 casos en once categorías [2]. Saludos, FAQ, menú y pedidos cubren diálogo ordinario. Workflow lleva una transacción por varios turnos. Pagos prueba monto exacto, vuelto, insuficiencia, métodos y billete grande. Handoff pide supervisor o expresa frustración. Seguridad y resiliencia usan mensajes sintéticos con prefijo [META:] para comportamientos que el endpoint conversacional no puede observar directamente.

Categoría	Casos	Intención
Greeting	3	Abrir conversación
FAQ	5	Horario, pago, ubicación y delivery
Menu	4	Listar, filtrar y recomendar
Single order	4	Resolver ítem y cantidad
Multi-order	3	Combinar ítems
Workflow	3	Mantener estado en flujo completo
Edge case	5	Cancelar, aclarar, rechazar input
Payment	5	Monto, método, suficiencia y vuelto

Categoría	Casos	Intención
Handoff	4	Escalar quejas y pedidos explícitos
Security	5	Respuesta segura a texto meta o malicioso
Resilience	5	Fallback conversacional ante prompts meta

Tabla 3. Distribución de la corrida guardada.

El grounding dinámico evita productos inexistentes, pero reduce repetibilidad: si cambian orden, disponibilidad, precios o FAQ, cambian casos y contexto. Un benchmark publicable debe serializar batería resuelta y fixture antes de correr. Sin snapshot, el generador es reproducible en forma pero no necesariamente en contenido.

La revisión registrada añade dos casos multi-order, dos edge cases con pedido activo y un handoff automático respecto de la salida guardada [3]. Esto es normal en ingeniería, pero rompe procedencia científica. La corrida debe citarse como artefacto fechado, no como resultado de toda definición posterior del mismo árbol.

5. Modelo juez, rúbrica y regla de aprobación

El juez usa gemma-3-27b-it con temperatura 0,3 [4]. Solicita cinco criterios de 0 a 100: relevancia, precisión contra contexto, completitud, tono y accionabilidad, con recordatorios para pagos, handoff, seguridad, resiliencia y meta tests. Devuelve JSON con overall, razonamiento y criterios. Los valores se limitan a [0,100]; una salida malformada se convierte en cero.

$$\text{aprueba}_i = 1[\text{overall\_score}_i \geq 75]$$

El umbral es inclusivo. El overall puede no coincidir con la media de criterios.

$$\text{pass\_rate} = (\Sigma_i \ \text{aprueba}_i) \ / \ N$$

Micro-promedio por casos; categorías con más casos pesan más.

El contexto se filtra por categoría. Menú aparece en pedidos; FAQ se rankea por keywords con peso extra para pagos [4]. Esto mejora relevancia pero puede darle al juez más información que al SUT. Accuracy mide consistencia con el catálogo suministrado, no verdad real independiente.

Criterio	Constructo	Ambigüedad
Relevancia	Responder intención actual	Puede premiar verbosidad
Precisión	Consistencia con catálogo	Puede ignorar errores de estado
Completitud	Cubrir próximo paso	La longitud puede dar ventaja
Tono	Estilo apropiado	Preferencia subjetiva y local
Accionabilidad	Avanzar transacción	Puede competir con responder primero

Tabla 4. Interpretación de la rúbrica.

6. Resultados de la corrida guardada

La salida tiene timestamp 2026-03-08T01:04:26.009Z [2]. Reporta 40/46, pass rate redondeado 87%, score promedio 80 y duración 671,3 s. El juez consumió 28.010 tokens: 23.038 de prompt y 4.972 de completion. Los tokens del SUT figuran no disponibles. El artefacto no incluye el p95 global que soporta el generador posterior [5].

Categoría	Aprobados	Score medio	Latencia media
Edge cases	5/5 (100%)	75	10,1 s
FAQ	4/5 (80%)	83	9,2 s
Greetings	3/3 (100%)	88	9,6 s
Human handoff	4/4 (100%)	95	7,7 s
Menu	4/4 (100%)	90	13,0 s
Multi-item orders	3/3 (100%)	95	12,0 s

Categoría	Aprobados	Score medio	Latencia media
Payment	0/5 (0%)	38	16,3 s
Resilience	5/5 (100%)	77	11,6 s
Security	5/5 (100%)	85	11,5 s
Single orders	4/4 (100%)	94	11,1 s
Workflows	3/3 (100%)	82	29,3 s

Tabla 5. Desglose completo.

El macro-promedio de las once medias es 82,0; el promedio ponderado reconstruido desde valores redondeados es ≈80,5, consistente con 80. La media ponderada de latencias mostradas es ≈12,5 s. Son diagnósticos aproximados porque no hay JSON crudo versionado.

> **HALLAZGO PRINCIPAL**
> Cinco de seis fallas son de pagos. El agregado oculta una categoría crítica con 0% de aprobación.

7. Análisis de fallas

Caso	Score	Falla observada
F1 — horario	35	Respondió con menú y no dio horarios
PAY-01 — pago exacto	30	Ignoró pago y repitió delivery/retiro
PAY-02 — vuelto	30	Ignoró monto y no calculó vuelto
PAY-03 — insuficiente	65	Reconoció mal la restricción y no ofreció ajuste
PAY-04 — métodos	35	Trató la pregunta como producto desconocido
PAY-05 — billete grande	30	Ignoró monto y repitió prompt anterior

Tabla 6. Casos fallidos.

Las fallas de pago comparten patrón: una intención válida atraviesa lógica de producto o estado que repite una pregunta anterior. Esto apunta más a ruteo o prioridad de estado que a generación lingüística. Cuatro respuestas no consumen el turno actual. PAY-03 llega a 65 porque reconoce parcialmente la restricción, lo que sugiere más de un modo de falla.

F1 es una falla separada de selección de intención. La respuesta dice que está abierto pero sustituye horario por menú. El juez asigna precisión alta porque el menú coincide con contexto, mientras relevancia y completitud caen. Un criterio puede ser localmente alto cuando la tarea fracasa.

- Causa candidata A: la intención de pago se evalúa después de una rama que corta el turno.
- Causa candidata B: el handler exige campos aún no persistidos.
- Causa candidata C: FAQ y pagos comparten keywords con distinta precedencia.
- Causa candidata D: al puntuar solo el turno final, la transición causal queda implícita.

> **LÍMITE DE INFERENCIA**
> Las salidas localizan el síntoma, no prueban qué rama o almacenamiento lo causó. Hace falta inspección de trazas o assertions de estado.

8. Auditoría de validez: seguridad, resiliencia y procedencia

Seguridad incluye descripciones meta de firma de webhook, JWT y rate limiting. El propio código comenta que pertenecen a la capa HTTP [3], pero el runner genérico envía el texto entre corchetes a /message y pregunta si la respuesta es segura. Un fallback amable no demuestra 401, rechazo de firma ni 429. Solo XSS-like y SQL-like ejercitan texto del usuario, y tampoco prueban efectos sobre navegador o base sin assertions.

Resiliencia tiene el mismo defecto. Un mensaje que dice validar circuit breaker no induce una falla, no avanza contadores ni observa backoff. Los 5/5 de resiliencia y seguridad son scores del manejo

conversacional de prompts sintéticos; no deben presentarse como validación de controles.

Afirmación	Evaluación	Razón
La corrida guardada obtuvo 40/46	Sostenida como evidencia del repo	Hay resultados por caso
Pago concentra la debilidad	Sostenida	Cinco de seis fallas
Seguridad pasó 5/5	No sostenida	Los prompts no ejercitan controles
Resiliencia pasó 5/5	No sostenida	No se inyectaron fallas
El árbol actual produce esos 46 casos	No establecida	La batería se amplió
El juez concuerda con humanos	No establecida	Sin calibración

Tabla 7. Claim ledger.

9. Verificación local independiente del 4 de agosto de 2026

Se clonó el repositorio público, se fijó la revisión 8467633188cd89e4708b8011ee2a0fb39b7f1a24 y se ejecutó npm ci con Node 24.14.0 y npm 11.9.0. La suite completa imprimió múltiples archivos aprobados pero no terminó dentro de la ventana acotada. Un comando reducido confirmó 36 assertions de test-battery.test.ts y 43 de conversation-assistant.test.ts. El proceso volvió a quedar vivo después de completar los tests y fue interrumpido.

Listado 1. Verificación local acotada.

```
git checkout --detach 8467633188cd89e4708b8011ee2a0fb39b7f1a24
npm ci
npx vitest run \
+  src/judge/test-battery.test.ts \
+  src/services/conversation-assistant.test.ts
```

Observación	Resultado	Interpretación
Assertions de batería	36 reportadas pass	Invariantes del generador ejercitados
Assertions del asistente	43 reportadas pass	Comportamiento mock/state ejercitado
Salida limpia	No	Quedan handles o defecto de teardown
Corrida end-to-end	No ejecutada	Requiere servicios, credenciales, costo y entorno
Score 40/46	No reproducido	Permanece reportado por repositorio

Tabla 8. Observaciones locales.

> **POR QUÉ NO SE LLAMA SUITE APROBADA**
> Las assertions imprimieron pass, pero un comando que no termina limpiamente no es un resultado reproducible limpio. La diferencia queda registrada.

10. Protocolo corregido

Una reejecución defendible debe separar propiedades observables. La capa A contiene integraciones deterministas para aritmética, mutaciones del carrito, auth, firmas, rate limit, circuit breaker, retry, persistencia y handoff. La capa B contiene juez de modelo para relevancia, completitud, tono y comportamiento abierto. La capa C usa humanos ciegos para calibrar y revisar desacuerdos.

Capa	Ejemplos	Salida
A — determinista	Pago, 401/403/429, carrito, circuit state	Pass/fail exacto y side effects
B — juez	FAQ, aclaración, tono, recovery	Criterios y razones
C — humanos	Muestra de pass, fail y borderline	Acuerdo, label adjudicado
Operaciones	Latencia, tokens, errores, retries	Distribuciones y costo

Tabla 9. Matriz propuesta.

1. Congelar catálogo y batería resuelta en JSON.
2. Fijar revisiones, modelo juez, prompt, temperatura, dependencias y entorno.
3. Corregir teardown hasta que los tests deterministas salgan con código cero.

4. Ejecutar el juez al menos tres veces y reportar estabilidad por caso.
5. Usar al menos dos evaluadores humanos ciegos en muestra estratificada.
6. Reportar acuerdo y elegir umbrales desde calibración, no convención.
7. Publicar salidas redactadas, definiciones, configuración y taxonomía de fallas.

$$estabilidad_i = frecuencia\_label\_modal_i / repeticiones$$

Si un caso cruza 75 entre corridas debe marcarse inestable, no resumirse con un solo label.

11. Amenazas a la validez y disclosure

- Constructo: el score aproxima calidad textual pero no observa estado oculto ni infraestructura.
- Interna: catálogo, proveedor, temperatura 0,3 y servicios compartidos cambian respuestas.
- Externa: un fixture de restaurante en español no generaliza a otros dominios.
- Conclusión: una corrida no estima varianza del juez ni intervalos por categoría.
- Procedencia: output, sprint summary y batería actual son momentos distintos.

Sistemas de IA contribuyeron al código, outputs del juez y preparación editorial. Pablo Cardozo es responsable por fuentes, interpretación, límites y correcciones. El reporte usa agregados y descripciones breves; no incluye conversaciones privadas.

12. Conclusión

El artefacto transforma una impresión vaga en un mapa concreto. Su resultado defendible no es que el asistente sea 87% bueno, sino que cinco pagos fallan coherentemente mientras pedidos y handoff reciben scores altos. Eso justifica investigar ruteo de intención y estado transaccional.

La auditoría también muestra que la infraestructura de evaluación debe evaluarse. Prompts meta no validan seguridad, una batería móvil no puede anexarse a una corrida vieja y un juez sin calibración no reemplaza humanos. Con checks deterministas, fixtures congelados, repeticiones y calibración, el harness puede convertirse en un instrumento fuerte.

13. Referencias y reproducibilidad

[1] Cardozo, P. restaurantes, revision 8467633188cd89e4708b8011ee2a0fb39b7f1a24, 2026. Repositorio público de implementación

[2] Cardozo, P. test_output.txt, run timestamp 2026-03-08T01:04:26.009Z. Salida guardada de 46 casos

[3] Cardozo, P. apps/restaurant-hours-api/src/judge/test-battery.ts, revision 8467633188cd, 2026. Fuente de la batería dinámica

[4] Cardozo, P. judge-agent.ts and judge-types.ts, revision 8467633188cd, 2026. Implementación del modelo juez y la rúbrica

[5] Cardozo, P. test-runner.ts and report-generator.ts, revision 8467633188cd, 2026. Runner y generador de reportes

[6] Cardozo, P. Sprint 1 Summary — SRS v4, 6 March 2026. Resumen fechado del Sprint 1

[7] Zheng, L. et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685, 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena

[8] Liu, Y. et al. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. arXiv:2303.16634, 2023. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment

[9] Kim, S. et al. Prometheus 2. arXiv:2405.01535, 2024. Prometheus 2: An Open Source Language Model Specialized in Evaluating Other Language Models

Revisión fuente: 8467633188cd89e4708b8011ee2a0fb39b7f1a24

Apéndices

A. Línea temporal de artefactos

Artefacto	Estado	Uso
Sprint summary, 6 marzo	19/46 baseline; 31/46 parcial	Historia de desarrollo [6]
Test output, 8 marzo	40/46; promedio 80	Resultado principal [2]
Revisión registrada	Batería ampliada y campos nuevos	Auditoría de implementación
Check local, 4 agosto	79 assertions; sin cierre limpio	Verificación parcial

Tabla A1. Evidencia que no debe mezclarse.

B. Release mínimo reproducible

- Fixture de catálogo y casos resueltos.
- Revisiones, modelos, prompts, temperatura y lockfile.
- Respuestas por turno y JSON del juez.
- Assertions de side effects para estado, seguridad y resiliencia.
- Labels de repeticiones y anotaciones humanas.
- Comando que cierre limpiamente y resumen con hashes.

Cita sugerida

Cardozo, Pablo. "Evaluación orientada a fallas con LLM-as-a-Judge para un agente transaccional: estudio de una corrida de 46 casos." Technical report, version 0.2, 2026. https://pablo.cardozo.com.ar/es/research/conversational-agent-evaluation.

Documento de trabajo independiente. Sin peer review.