Failure-Focused LLM-as-a-Judge Evaluation for a Transactional Conversational Agent: A 46-Case Repository Study

Pablo Cardozo
Independent researcher · Buenos Aires, Argentina

Abstract

An aggregate score can hide the defect that matters. In a saved 46-case run for a transactional restaurant assistant, 40 cases pass at a threshold of 75/100 and the mean score is 80. Yet all five payment cases fail; only one failure appears elsewhere. That concentration localizes the symptom at the payment boundary, though the artifact cannot prove which branch or persistence layer caused it. This report reconstructs the model-judge harness and audits what its scores can support. Cases labelled as security or resilience checks are only sent to the chatbot and judged for conversational plausibility; they never exercise the HTTP, rate-limit, retry, or circuit-breaker behavior named by the cases. The checked-in battery also contains cases absent from the saved output, and the result has no human calibration, repeated seeds, or independent end-to-end rerun. A partial local check on 4 August 2026 confirmed 79 assertions, but Vitest retained open handles and did not terminate cleanly. The saved scores are therefore treated as repository observations, not ground truth. A defensible next version would use deterministic assertions for state and security, reserve model review for open-ended dialogue, and calibrate that review against blinded human ratings.
Keywords: LLM-as-a-Judge; conversational agents; transactional dialogue; failure analysis; evaluation validity; observability

STATUS	Saved 46-case run inspected; provenance gaps and test-design limits documented · Partial local verification: 2026-08-04

SAVED RUN	40/46 pass · mean 80/100
FAILURE CLUSTER	5 payment · 1 FAQ
RUNTIME	671.3 s · 28,010 judge tokens
LOCAL CHECK	79 assertions passed; process hung

Contents

1. The failure cluster hidden by the headline

Transactional agents are harder to evaluate than single-turn question answering because correctness depends on state. A payment response can be linguistically fluent yet wrong because the cart total was lost; a handoff can sound empathetic yet fail to persist the transfer; a menu answer can be accurate for the catalog but irrelevant to the user's current step. Reference-string metrics are weak for these open-ended responses, while manual review is expensive and difficult to run after each change.

The audited repository implements a separate model judge, a dynamically generated battery, a stateful system-under-test runner, and Langfuse tracing [1,3–5]. A saved run from 8 March 2026 contains case-level outputs, five criterion scores, latency, token counts, and textual reasons in addition to the aggregate [2]. This report reconstructs that experiment and tests the validity of the conclusions one may draw from it.

- Describe the battery, execution path, judge rubric, and aggregation logic from source.
- Report the complete category-level result and analyze the six failures rather than presenting only the 87% headline.
- Audit whether the named test categories actually exercise the properties they claim to measure.
- Document provenance drift between the saved 46-case run and the larger battery in the recorded source revision.
- Propose a calibrated evaluation architecture that separates deterministic properties, open-ended quality, and human judgment.

2. LLM-based evaluation and known judge bias

MT-Bench and Chatbot Arena established LLM-as-a-Judge as a scalable way to approximate human preference on open-ended assistant responses, while explicitly documenting position, verbosity, self-enhancement, and reasoning biases [7]. G-Eval showed that a structured rubric and form-filling output can improve correlation with human evaluation for generation tasks, but also reported potential preference for model-generated text [8]. Prometheus 2 extends the evaluator-model approach to both direct scoring and pairwise ranking with user-defined criteria [9]. These works support using a judge as an instrument; none implies that an uncalibrated judge score is ground truth.

The audited harness uses direct assessment, not pairwise comparison. It supplies the conversation, final answer, selected catalog context, and category to one judge model. The run produces explanations and category summaries. Its main scientific weakness is meta-evaluation: there is no local study of agreement with human raters, repeatability across seeds, or sensitivity to prompt and model changes. It can screen for failures and generate hypotheses about their causes; it cannot establish ground truth.

Evaluator	Best suited for	Poor fit
Deterministic assertion	Arithmetic, state transitions, auth, schema, side effects	Tone and naturalness
Model judge	Relevance, completeness, tone, open-ended response quality	Security guarantees and hidden side effects
Human rater	Calibration, ambiguity, business appropriateness	High-frequency regression alone
Telemetry	Latency, tokens, errors, trace attribution	Semantic quality without labels

Table 1. Role of each evaluator in a defensible test program.

3. System under test and evaluation architecture

The system under test is a restaurant assistant with catalog-backed answers, ordering, payment handling, persistent conversation state, human handoff, and resilience logic [1]. The test runner first obtains an authenticated catalog snapshot. It then generates cases using available product names and FAQ entries, creates a unique chat identifier per case, sends every user message in sequence to the /message endpoint, and accumulates the observed responses [3,5]. Only the final response is scored, while earlier turns are embedded in the judge's conversation context.

1. Fetch the live catalog using an admin token.

2. Generate a battery whose product references are grounded in that catalog.
3. Create a new conversation identifier for each test to reduce state contamination.
4. Execute all messages of a test sequentially against the system endpoint.
5. Construct context from user turns and non-final bot replies; pass the final reply separately.
6. Invoke the judge, apply the threshold, attach SUT and judge traces, and aggregate by category.

Level	Fields
Case	ID, category, messages, expected behavior, replies, pass/fail, reasoning
Judge	Overall, relevance, accuracy, completeness, tone, actionability
Operations	SUT latency, judge latency, SUT tokens, judge tokens, trace identifiers
Category	Count, passes, mean score, mean latency, p95 latency, token totals
Run	Timestamp, totals, pass rate, mean score, duration, p95, token totals

Table 2. Recorded result fields.

State reset A unique chat identifier isolates cases at the conversation layer. It does not necessarily reset global catalog state, provider caches, model behavior, or external services; those remain shared run-level factors.

4. Battery design and task coverage

The saved artifact contains 46 cases across eleven categories [2]. Greeting, FAQ, menu, and ordering cases assess ordinary dialogue. Workflow cases carry a transaction across several turns. Payment cases probe exact payment, change, insufficient funds, method selection, and a large bill. Handoff cases ask for a supervisor or express frustration. Security and resilience cases use synthetic messages prefixed with [META:] for behaviors that the chatbot endpoint cannot directly observe.

Category	Cases	Primary intent
Greeting	3	Open conversation appropriately
FAQ	5	Answer hours, payment, location, delivery
Menu	4	List, filter, recommend available products
Single order	4	Resolve item and quantity
Multi-order	3	Combine or update multiple items
Workflow	3	Maintain state through complete order flows
Edge case	5	Cancel, clarify, reject unknown input
Payment	5	Interpret amount, method, sufficiency, change
Handoff	4	Escalate complaints and explicit requests
Security	5	Safe conversational response to meta or malicious text
Resilience	5	Conversational fallback to meta failure prompts

Table 3. Distribution in the saved 46-case run.

Dynamic grounding avoids hard-coding a product that is absent from the current catalog. It also reduces repeatability: if product order, availability, prices, or FAQ text changes, the generated cases and judge context change. A publishable benchmark should serialize the resolved battery and catalog fixture before execution. Without that snapshot, the source generator is reproducible in form but not necessarily in content.

The recorded source revision adds two multi-order cases, two active-order edge cases, and an automatic-handoff case relative to the saved output, producing a larger conceptual battery than the 46-case artifact [3]. This is not a problem for ongoing engineering; it is a provenance problem for research reporting. The saved run must be cited as its own dated artifact rather than as a result of every later test definition present in the same repository tree.

5. Judge model, rubric, and pass rule

The judge uses gemma-3-27b-it at temperature 0.3 [4]. Its system prompt requests five 0–100 criteria: relevance, accuracy against supplied context, completeness, tone, and actionability. It adds category-specific reminders for payment, handoff, security, resilience, and meta tests. The model returns JSON containing an overall score, reasoning, and criteria. Values are clamped to [0,100]; malformed output becomes a zero-score failure.

FORMULA	$\text{pass}_i = 1[\text{overall\_score}_i \geq 75]$
The pass threshold is inclusive. The judge may provide an overall score that is not the arithmetic mean of the five criteria.	

FORMULA	$\text{pass\_rate} = (\Sigma_i \ \text{pass}_i) \ / \ \text{N}$
A micro-average over cases. Categories with more cases receive more weight.	

Catalog context is filtered by category. Menu entries are included for order-related cases; selected FAQ entries are ranked by keyword overlap, with extra weight for payment topics [4]. This improves relevance and lowers prompt size, but it can also make the judge more informed than the system under test. Accuracy scores should therefore be interpreted as consistency with supplied catalog context, not independently verified real-world correctness.

Criterion	Intended construct	Ambiguity
Relevance	Responds to current user intent	May reward verbosity that mentions more intents
Accuracy	Consistent with supplied catalog	Judge sees curated context and may overlook state errors
Completeness	Covers necessary next step	Longer answers may receive an advantage
Tone	Appropriate conversational style	Locale and brand preference are subjective
Actionability	Moves transaction forward	Can conflict with answering the immediate question first

Table 4. Rubric interpretation and principal ambiguity.

6. Saved-run results

The saved console artifact is timestamped 2026-03-08T01:04:26.009Z [2]. It reports 40 passing cases out of 46, an 87% rounded pass rate, a mean overall score of 80, and 671.3 seconds of total duration. Judge usage is 28,010 tokens: 23,038 prompt and 4,972 completion. SUT token usage is recorded as unavailable. The artifact does not contain the run-level p95 later supported by the checked-in report type and generator [5].

Category	Passed	Mean score	Mean latency
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
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

Table 5. Complete category breakdown from the saved artifact.

The macro-average of the eleven displayed category means is 82.0, while the weighted average reconstructed from rounded category means is approximately 80.5, consistent with the reported case-level mean of 80 after rounding. The weighted mean of displayed category latencies is approximately 12.5 seconds. These derived values are diagnostic only because category means are rounded and the raw JSON report is not checked in.

Primary finding Five of the six failures are payment cases. The aggregate pass rate conceals a complete failure of one transaction-critical category.

7. Failure analysis

Case	Score	Observed failure
F1 — hours	35	Answered with a menu listing instead of operating hours
PAY-01 — exact payment	30	Ignored payment statement and repeated delivery/pickup question
PAY-02 — change	30	Ignored amount and failed to calculate change
PAY-03 — insufficient amount	65	Acknowledged constraint poorly and did not offer a useful adjustment
PAY-04 — methods	35	Treated payment question as an unidentified product
PAY-05 — large bill	30	Ignored amount and repeated an earlier state prompt

Table 6. Failed cases and observed behavior.

The payment failures share one pattern: a valid payment intent is routed through product or order-state logic that repeats a previous question. This is stronger evidence for a routing or state-priority defect than for a language-generation defect. Four failed responses leave the current turn unconsumed, which makes tone secondary. PAY-03 reaches 65 because it partially recognizes the constraint, suggesting the category contains more than one failure mode.

F1 is a separate intent-selection failure. The answer states that the restaurant is open but substitutes a menu for the actual hours. The judge assigns accuracy 100 in the stored trace because the listed products match catalog context, while relevance and completeness collapse. This illustrates why an overall score is not enough: a criterion can be locally high even when the response fails the user's task.

- Candidate cause A: payment intent is evaluated after an order-state branch that short-circuits the current message.
- Candidate cause B: payment handling requires fields not yet committed to persistent state.
- Candidate cause C: FAQ and payment routes use overlapping keywords but different precedence.
- Candidate cause D: the judge evaluates only the final turn, so the exact state transition causing the failure remains implicit.

Inference boundary The saved outputs localize the symptom. They do not prove which branch or persistence layer caused it; that requires trace-level inspection or deterministic state assertions.

8. Validity audit of security, resilience, and provenance

The security category includes meta descriptions for webhook signature validation, JWT authentication, and rate limiting. In the battery source, comments acknowledge that these properties belong at the HTTP layer [3]. The generic runner nevertheless sends the bracketed meta text to /message and asks the judge whether the conversational reply is safe. A polite fallback cannot establish that an unauthenticated admin request returns 401, that a webhook signature is rejected, or that a rate limiter returns 429. Only the XSS-like and SQL-like text cases exercise user-message handling, and even those do not prove database or browser safety without side-effect assertions.

The same problem affects resilience. A message saying that a circuit breaker or retry should be validated does not induce a provider failure, advance a failure counter, observe an open state, or measure backoff. The reported 5/5 resilience and 5/5 security results are scores for conversational handling of synthetic prompts. They must not be presented as security or fault-tolerance validation.

Claim	Assessment	Reason
The saved run scored 40/46	Supported as repository evidence	Dated output contains case-level results
Payment is the dominant observed weakness	Supported	Five of six failures are payment cases
Security controls passed 5/5	Not supported	Meta prompts did not exercise controls
Resilience mechanisms passed 5/5	Not supported	No faults or state transitions were injected
The current source tree yields the saved 46 cases	Not established	Battery definitions have since expanded
Judge scores match human judgment	Not established	No calibration or agreement study

Table 7. Claim ledger for the 46-case artifact.

9. Partial local verification on 4 August 2026

The public repository was cloned, detached at revision 8467633188cd89e4708b8011ee2a0fb39b7f1a24, and installed with npm ci under Node 24.14.0 and npm 11.9.0. The Cuqui repository is not part of this report; all counts below refer to restaurantes. Running the full Vitest suite produced many passing files but did not terminate within the bounded observation window. A narrower command confirmed that all 36 assertions in test-battery.test.ts and all 43 assertions in conversation-assistant.test.ts reported pass. The process again remained alive after test completion and was interrupted.

Listing 1. Bounded local check.

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

Observation	Result	Interpretation
Battery unit assertions	36 reported pass	Generator invariants exercised locally
Assistant assertions	43 reported pass	Mocked/state behavior exercised locally
Clean process exit	No	Open handles or teardown defect remains
End-to-end judge run	Not executed	Would require live services, credentials, cost, and a pinned environment
Saved 40/46 score	Not reproduced	Remains repository-reported

Table 8. Local observations.

Why this is not called a passing local suite Assertions printed as passed, but a test command that does not terminate cleanly is not a clean reproducible result. The distinction is recorded rather than hidden.

10. Corrected evaluation protocol

A defensible rerun should split the battery by observable property. Layer A contains deterministic integration tests for arithmetic, cart mutations, auth responses, webhook signatures, rate limiting, circuit-breaker state, retry timing, persistence, and handoff side effects. Layer B contains model-judge cases for relevance, completeness, tone, and natural multi-turn behavior. Layer C contains blinded human ratings used to calibrate Layer B and review disagreements.

Layer	Examples	Output
A — deterministic	Payment math, 401/403/429, cart state, circuit state	Exact pass/fail plus side-effect log
B — model judge	FAQ relevance, clarification, tone, recovery wording	Criteria scores and explanations
C — human calibration	Stratified sample of pass, fail, and borderline cases	Agreement, adjudicated label, comments
Operations	Latency, tokens, provider errors, retries	Per-stage distributions and cost

Table 9. Proposed experiment matrix.

1. Freeze the catalog fixture and resolved test battery as versioned JSON.
2. Pin SUT revision, judge model, prompt, temperature, dependencies, deployment, and environment.
3. Run deterministic tests once per revision and fix teardown until the command exits zero cleanly.
4. Run the model-judge layer at least three times; report mean, range, and label stability per case.
5. Blind at least two human raters to the automatic score on a stratified subset.
6. Report agreement, adjudicate disagreements, and choose thresholds from calibration rather than convention.
7. Publish raw redacted outputs, test definitions, configuration, and failure taxonomy.

FORMULA

$$\text{stability}_i = \text{modal\_label\_count}_i \;/\; \text{repeated\_runs}$$

A case that crosses the 75 threshold across repeated model-judge runs should be marked unstable, not summarized by one label.

11. Threats to validity and disclosure

- Construct validity: an LLM score approximates conversational quality but cannot observe hidden state or infrastructure controls from text alone.
- Internal validity: catalog changes, provider drift, temperature 0.3, and shared services can change responses across runs.
- External validity: one Spanish-language restaurant fixture does not generalize to other transactional domains or risk levels.
- Conclusion validity: one run provides no estimate of judge variance and no confidence interval for category performance.
- Provenance validity: saved output, sprint summary, and current battery represent different points in an evolving repository.

AI systems contributed to the application code, judge outputs, and editorial preparation of this report. Pablo Cardozo is responsible for the source selection, interpretation, claim boundaries, and corrections. The repository output is quoted through aggregate numbers and short failure descriptions; no client or private user conversation is used.

12. Conclusion

The 46-case artifact is useful because it turns a vague impression of chatbot quality into a concrete failure map. Its most defensible result is not that the assistant is 87% good; it is that five payment cases fail in a coherent way while most ordinary ordering and handoff cases receive high judge scores. That pattern justifies targeted investigation of intent routing and transaction state.

The audit applies the same standard to the harness itself. Meta prompts cannot validate security controls, a moving battery cannot be silently attached to an older run, and an uncalibrated judge cannot substitute for humans. Deterministic control tests, frozen fixtures, repeated judge runs, and blinded calibration would turn the existing harness into a regression instrument whose limits are known.

A. Artifact timeline

Artifact	Reported state	Use in this report
Sprint summary, 6 March 2026	19/46 baseline; 31/46 partial	Development history only [6]
Saved test output, 8 March 2026	40/46; mean 80; six listed failures	Primary repository result [2]
Recorded source revision	Expanded battery and newer report fields	Implementation audit [1,3–5]
Local check, 4 August 2026	79 assertions reported pass; no clean exit	Partial independent verification

Table A1. Distinct evidence snapshots that must not be conflated.

B. Minimum reproducibility release

- Frozen catalog fixture and resolved case JSON.
- Exact SUT and judge revisions, models, prompts, temperature, and dependency lockfile.
- Raw per-turn SUT responses and per-case judge JSON.
- Deterministic side-effect assertions for state, security, and resilience.
- Repeated-run labels and human calibration annotations.
- A command that exits cleanly and a machine-readable summary with hashes.

C. References and reproducibility

1. Cardozo, P. restaurantes, revision 8467633188cd89e4708b8011ee2a0fb39b7f1a24, 2026. Source ↗
2. Cardozo, P. test_output.txt, run timestamp 2026-03-08T01:04:26.009Z. Source ↗
3. Cardozo, P. apps/restaurant-hours-api/src/judge/test-battery.ts, revision 8467633188cd, 2026. Source ↗
4. Cardozo, P. judge-agent.ts and judge-types.ts, revision 8467633188cd, 2026. Source ↗
5. Cardozo, P. test-runner.ts and report-generator.ts, revision 8467633188cd, 2026. Source ↗
6. Cardozo, P. Sprint 1 Summary — SRS v4, 6 March 2026. Source ↗
7. Zheng, L. et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685, 2023. Source ↗
8. Liu, Y. et al. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. arXiv:2303.16634, 2023. Source ↗
9. Kim, S. et al. Prometheus 2. arXiv:2405.01535, 2024. Source ↗

Suggested citation

Cardozo, Pablo. “Failure-Focused LLM-as-a-Judge Evaluation for a Transactional Conversational Agent: A 46-Case Repository Study.” Technical report, v0.2, 2026.

Independent working document. Not peer reviewed. Claims are limited to the artifacts and source revision named above.