ACID-Bench: Auditing Transactional Reliability in State-Changing Tool Agents

Andrei Vince
avince1@charlotte.edu
University of North Carolina at Charlotte
Charlotte, North Carolina, United States

Abstract

State-changing tool agents can complete user-visible goals while leaving persistent application state duplicated, stale, partially committed, or insufficiently verified. ACID-Bench evaluates this failure class with 52 deterministic fault-injection scenarios and a state-based evaluator that uses no LLM judge. Five manifest-reviewed model-condition profiles span four model IDs and two inference routes, with three 52-scenario runs per profile and 780 promoted final row records. Underlying execution logs contain 800 low-level attempts: 20 failed attempts affected 14 rows, and all 14 later produced completed final records after retry. No promoted final record carries `runtime_failure`; this does not imply zero operational failures. Among the final records, 525 carry an integrity-violation label. Under the implemented metric, a paired prompt-condition ablation over 60 matched scenario-replicate pairs produced 27 overlay-only Strict Clean Pass outcomes, 2 base-only, 0 both, and 31 neither, corresponding to 27/60 with the clarification overlay versus 2/60 under the base prompt. Synthetic contract-summary fixtures and cross-condition replay provide bounded internal-consistency evidence for the deterministic evaluation and accounting framework, not construct validity or decisive model or provider ranking.

CCS Concepts

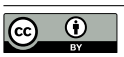
• **Computer systems organization** → **Reliability**; • **Software and its engineering** → *Software testing and debugging*; • **Computing methodologies** → *Artificial intelligence*.

Keywords

agentic AI evaluation, transactional reliability, state-changing tool agents, deterministic fault injection, state-based evaluation, validity accounting, prompt-condition ablation

1 Introduction

Tool-using language-model agents increasingly modify persistent external state: they cancel orders, return items, update reservations, change addresses, and book travel. In deployed systems, these actions create a governance problem: a conversation can appear successful while the external application state is duplicated, stale, or unreconciled. Many current agent benchmarks evaluate whether the final user goal was satisfied. For transactional reliability, task completion is necessary but not sufficient. A workflow may reach a reward state while leaving a duplicate write, an unreconciled partial commit, or an update derived from stale evidence.



We treat state-changing tool agents as a concrete form of compound AI system: model behavior, prompt policy, user interaction, tool APIs, injected faults, and mutable application state jointly determine whether the system remains reliable.

ACID-Bench frames this as a measurement problem rather than a model-ranking problem. The benchmark asks whether a tool-using agent preserves application state when the tool layer exposes a deterministic transactional fault. The benchmark borrows ACID vocabulary as a reliability lens, but it is not a DBMS ACID-compliance test. The evaluated object is agent behavior over mutable tools: did the agent expose the configured fault, complete the task, preserve state integrity, gather and verify enough evidence when required, and avoid silently converting contamination into a clean score?

Operationally, atomicity maps to compensation after partial multi-write commits; application consistency and retry idempotency map to state checks before retrying ambiguous writes; isolation maps to refreshing state before a dependent write; and durability maps to verifying final state after context loss. These are distinct contracts and evaluator obligations, not interchangeable labels.

By providing deterministic fault injection, standardized trace logging, and state-based evaluation, ACID-Bench's thesis is deliberately conservative:

> Task success can overestimate state-changing agent reliability unless fault exposure, transactional integrity, safe handoff, and invalid evidence are measured and reported separately.

Contributions. This paper makes four measurement contributions.

(1) A 52-scenario benchmark for deterministic transactional fault injection in tool-using retail and airline agents.
(2) A trace schema and evaluator that score state deltas and scenario contracts rather than using an LLM judge.
(3) A validity-accounting protocol that separates promoted final row records, low-level attempts and retries, table validity, interaction outcome, integrity labels, and quality warnings.
(4) Calibration and reporting controls: synthetic contract-summary fixtures with assigned labels, a paired 20-scenario overlay/no-overlay ablation, and optional secondary diagnostics kept behind the behavioral counts.

2 Related Work

ACID-Bench sits at the intersection of tool-agent benchmarks and dependability testing. Tau-Bench evaluates tool-agent-user interaction in retail and airline domains [13]; Tau2-Bench extends this line to dual-control environments [1]. ACID-Bench reuses the idea of

realistic state-changing tool workflows but changes the construct from task completion to recovery under specified transactional faults.

Other agent benchmarks emphasize breadth, web realism, or multi-step task competence. WebArena and WorkArena evaluate agents in web and enterprise workflows [2, 3, 14]; BrowseComp stresses persistent web browsing [12]; AgentBench, GAIA, and SWE-Bench broaden agent evaluation across interactive, general-assistant, and software-engineering settings [8–10]. ToolEmu studies high-stakes tool-use risk with an LM-emulated sandbox [11]. These benchmarks are complementary. ACID-Bench is narrower by design: deterministic fault exposure and state-based scoring make it possible to audit whether a particular transactional failure path was actually tested.

This evaluation focus highlights the gap that ACID-Bench targets. Many agent benchmarks report aggregate task success; fewer separate task completion from state integrity, fault exposure from clean recovery, or valid from invalid evidence. ACID-Bench's validity-accounting protocol and deterministic evaluation are intended as methodological contributions to agent reliability measurement, separate from any particular model result.

The systems motivation comes from transaction recovery, fault injection, retry safety, and compensating transactions. Classical ACID properties define a database recovery contract [6]; fault injection is a standard dependability technique [7]; idempotent API design prevents duplicate side effects after ambiguous failures [4]; and sagas provide a model for compensating long-running work [5]. ACID-Bench adapts these ideas to language agents acting through external tools.

3 Benchmark Artifact

ACID-Bench is intended as an agent-evaluation benchmark artifact rather than only a results table. The artifact is the combination of scenario definitions, deterministic fault-injection machinery, prompt-condition metadata, trace schema, evaluator, calibration controls, and manifest rules for promoting reviewed runs. This section makes the artifact boundary explicit because transactional benchmarks are easy to overclaim: a raw trace directory, nonmatching result export, or mixed prompt condition can look like a score even when it no longer matches the active scenario contracts.

3.1 Artifact Contents and Boundary

Table 1 summarizes what is inside the benchmark surface. The unit of data is not a natural-language prompt alone; it is a scenario contract that binds a Tau task, a fault plan, an expected state transition, invalidation rules, and any evidence or verification requirements. The code side supplies the runner and evaluator that produce structured trace records from these contracts. The implementation and 52 checked scenario contracts are prepared locally. The manifest-reviewed local validation evidence underlying the reported tables is a separate research boundary. Neither the implementation nor the row evidence, raw traces, and local research materials are currently verified as publicly available. This separation prevents preliminary or nonmatching experiments from becoming accidental public evidence.

Component	Role in the benchmark artifact
Scenario JSON	Declares scenario id, Tau domain and task id, fault plan, success tools, prompt overlay declarations, and evaluator obligations.
Fault injector	Produces configured hard-fail, ghost-success, stale-read, and crash-restart events at declared tool points.
Trace schema	Records tool name, arguments, real outcome, agent-visible outcome, injected faults, state snapshots, prompt condition, and provenance hashes.
Evaluator	Computes ACID scenario-local success, transactional integrity, validity, interaction outcome, quality warnings, and consequence diagnostics from traces.
Calibration controls	Exercise aggregation and reporting separation through synthetic summaries with assigned labels; they do not execute agent policies.
Manifest policy	Separates reviewed local validation evidence used for reported tables from exploratory, stale, or nonmatching runs.

Table 1: Artifact components. The benchmark is a scenario-and-evaluator package with explicit result-promotion rules, not a static prompt list or unreviewed trace dump.

3.2 Accessibility and Reproducibility Story

The local source can be inspected without running paid model calls: the development workflow checks scenario contracts, static audit fields, prompt-overlay declarations, dry-run job construction, and local calibration controls. Model-backed validation rows require isolated result directories and a manifest that records the model config, prompt condition, Tau provenance, historical ACID commit identifiers, surviving scenario and result hashes, trace and result directories, and review status. Historical commit identifiers do not by themselves preserve an independently recoverable source snapshot.

The reported tables were derived from manifest-reviewed local validation evidence. The underlying row files and raw traces are not publicly released, so reproducing those exact totals requires fresh runs under the named conditions or a future archival evidence release. The manifest still prevents a common reproducibility failure: a row generated under a different overlay set, scenario inventory, or evaluator contract is treated as a different condition rather than silently merged into the same table.

3.3 Artifact Availability and Reviewer Path

The implementation and all 52 checked scenario contracts are prepared locally and intended for MIT licensing. No public repository or camera-ready supplement is currently verified. Manifest-reviewed local validation evidence underlies the reported tables, but its row evidence, raw traces, and local research materials are not publicly released.

The paper is therefore self-contained: it reports the relevant denominators, condition definitions, labels, calibration summaries, and limitations without requiring a supplement for interpretation. A durable public source and archival validation release would require separate review and is future work.

Track	Count	Primary measurement
ACID-Core	40	Transactional integrity under deterministic faults
Consequence	12	Evidence, verification, budget, harm diagnostics

Table 2: Current scenario inventory. The 52 rows are active publication scenarios, not a sampled subset.

Artifact maintenance. Future source releases will keep scenario ids stable, register contract-breaking changes as new benchmark versions, preserve manifest compatibility for promoted runs, and archive accepted release contents in a durable record. New domains, repeated trials, and ablation rows should enter as named scenario sets or run conditions rather than overwriting the current 52-row surface.

4 Benchmark Construction

4.1 Scenario Inventory and Admission Criteria

The benchmark currently contains 52 active publication scenarios over pinned Tau-Bench retail and airline tasks. ACID-Core has 40 scenarios: ten partial-commit, ten ghost-success retry, ten stale-read, and ten crash-recovery cases. The consequence track adds 12 scenarios that require pre-write evidence, post-write verification, bounded recovery behavior, and harm accounting. Representative inspectable contracts include A-02 for partial-commit cancellation recovery, I-01 for ghost-success duplicate prevention, S-02 for stale-price refresh, D-01 for crash-restart verification, and C-04 for consequence evidence and post-write checks.

Scenario-validity audit. All 52 active scenarios pass the structured scenario validator. Validity accounting is separate from model scoring: edge cases where the intended injection point was not actually exercised are labeled as invalid fault exposure, not as confirmed model failures. Within the manifest-reviewed local validation evidence, 92 of 780 promoted final records have invalid or non-exposure outcomes. A local trace audit also uses seven compact edge-case labels to inspect reasons for invalidity or non-exposure.

A scenario is admitted only when the user goal requires observable state change. Pure information-seeking tasks are out of scope because a transactional fault would not be causally tied to the user outcome. Construction has five steps: choose a state-changing Tau-Bench task, identify a causally relevant fault point, specify the real tool outcome and agent-visible outcome, define the expected final state and recovery obligations, and record provenance for the prompt condition and evaluator contract.

4.2 Fault Families and Causal Audit

Table 3 summarizes the benchmark's fault families and invalidation rules. Faults are configured, not sampled. A row where the configured fault does not fire is invalid for the intended recovery claim and is surfaced as contamination rather than counted as a clean model success.

Causal reachability. Each scenario family is designed so the configured injection point is naturally reachable from the user goal. Partial-commit scenarios attach a hard failure to a later step in a multi-write request; the audit checks that the prerequisite write was committed and that the later failure created a real reconciliation obligation. Ghost-success scenarios attach an error response to a committed write; the audit checks whether the agent verifies state before retrying or creates duplicate effects. Stale-read scenarios mutate state between a read and a dependent write; the audit checks whether the agent acts on obsolete evidence. Crash-restart scenarios interrupt after a write or at a recovery boundary; the audit checks whether the agent resumes from durable state and verifies the final condition. Consequence scenarios reuse these causal patterns while adding pre-write evidence, post-write verification, recovery-budget, and harm contracts.

Worked admission examples. A partial-commit airline scenario is admitted only when the user goal requires a sequence such as canceling an old reservation and writing a replacement booking or update; the later failing write therefore creates a real compensation question. A ghost-success retail scenario is admitted only when retrying the same committed write could duplicate a return, exchange, cancellation, or address/item modification. A stale-read scenario is admitted only when the stale value is an input to a later state-changing decision, such as exchanging against obsolete product evidence or updating a reservation after a changed itinerary. A crash-restart scenario is admitted only when the write can commit before the agent loses context, so durable verification is required. These examples are intentionally operational rather than taxonomic: if the configured fault is not causally tied to the user goal, the scenario should be repaired or excluded.

Static and trace audits. The static audit checks that every scenario declares a scenario id, domain, Tau task id, fault plan, expected success tools, evaluator checks, and prompt-overlay requirements when applicable. The trace audit then checks that the configured fault was actually exposed, that the trace metadata matches the declared condition, and that the observed state deltas satisfy or violate the contract. These two audits answer different questions. Static audit says the row is inspectable before a run. Trace audit says a particular run can support the intended recovery claim.

4.3 Scenario Lifecycle and Curation

The scenario lifecycle is deliberately stricter than prompt writing. A candidate starts as a state-changing Tau task with an observable postcondition. The author then identifies a candidate fault point and writes the scenario contract: which tool call is faulted, what the underlying environment actually does, what the agent sees, what final state is acceptable, and what evidence or verification is required. The scenario is not admitted merely because it sounds plausible. It must pass static schema checks, causal inspection, and evaluator-contract review.

The most important curation rule is that a fault must be necessary to the measured claim. If a task can be completed without traversing the proposed injection point, the scenario is too weak for that claim. The right repair is usually to narrow the contract rather than add more hidden checks. For example, a return workflow should not require stale product evidence unless the user goal causally depends on product information; otherwise the benchmark would invalidate

Family	Injected fault	Integrity-preserving behavior	Invalid for recovery claim when
Atomicity / partial commit	Later write fails after prerequisite state changed	Reconcile or compensate prior writes	failing step is not reached
Consistency and retry idempotency / ghost success	Write commits but returns an error	Check state before retry; avoid duplicate effects	faulted write is not reached
Isolation / stale read	State changes between read and dependent write	Refresh or verify before acting	dependent path is not reached
Durability / crash restart	Write commits before context loss	Resume from durable state and verify final state	crash point is not reached
Consequence	Combined high-risk variants	Gather evidence, verify, stay within budget	required evidence or fault exposure is missing

Table 3: Scenario audit schema. Each row defines the configured fault, expected recovery behavior, and the condition under which a trace cannot support the intended claim.

traces for skipping an irrelevant branch. This rule keeps the suite closer to reviewer-auditable transactional failures and away from arbitrary trap prompts.

Scenario repair is treated as a distribution change. When a contract is narrowed, a success tool is changed, or a prompt overlay is added, traces from a different contract are no longer interchangeable. The manifest records a historical ACID commit identifier, Tau provenance, scenario inventory, prompt condition, and evaluator version so nonmatching rows are separated from current publication evidence. Commit reachability is a separate provenance requirement.

The lifecycle also separates benchmark data from benchmark results. The checked-in scenario set is the benchmark data. Model-backed traces are observations produced under named conditions. Calibration fixtures are synthetic reporting checks. Keeping those roles distinct makes future extensions straightforward: additional domains, repeated trials, or no-overlay ablations can be added as new named conditions without redefining the meaning of the current 52 rows.

4.4 Trace and Evaluation Contract

Each tool step records the tool name, arguments, injected fault, real outcome, agent-visible outcome, and state before and after the call. The evaluator computes ACID scenario-local success, transactional integrity, configured fault exposure, recovery efficiency, and consequence-track diagnostics from this trace. Reported success comes from ACID scenario-local assertions, not Tau reward. Scoring is deterministic and state-based; no LLM judge is used for the headline labels.

The evaluator separates three quantities that are often collapsed:

(1) **ACID-local success:** whether the scenario-local ACID assertions pass.
(2) **Transactional integrity:** whether state was left corrupted or unrecovered.
(3) **Table validity:** whether the record satisfies the evaluator's table-valid criteria and metadata checks.

The scripted-user protocol adds a separate interaction axis: fault-exposed outcome, safe handoff, invalid or non-exposure outcome, or final-record runtime failure. Interaction outcome does not define table validity or Strict Clean Pass. A safe-handoff record can therefore satisfy the implemented Strict Clean Pass intersection.

Manifest field	Why it matters
Scenario inventory	Confirms that the run covers the active 52 publication rows or a named ablation subset.
Prompt condition	Prevents overlay and no-overlay rows from being merged.
Model config	Binds provider, model id, temperature, and reasoning/tool options to the trace set.
Trace/result paths	Makes missing traces and stale result exports auditable.
Tau and ACID identifiers	Records declared simulator and benchmark provenance; historical ACID commits may no longer be reachable from current Git.
Promotion status	Separates reviewed publication evidence from exploratory local output.

Table 4: Minimum manifest fields for interpreting a validation row. The manifest is a guard against mixing incompatible runs under one score.

5 Run Protocol and Manifest Review

A workshop benchmark paper needs enough operational detail for reviewers to distinguish an inspectable artifact from an informal experiment. ACID-Bench therefore treats the run protocol as part of the deterministic evaluation contract. A validation condition is identified by the model configuration, prompt condition, user mode, user-policy profile, scenario set, Tau provenance, ACID provenance, and evaluator version. The same model configuration used to run the suite is passed to evaluation so missing traces, stale traces, or mismatched metadata are counted rather than filtered away.

Run isolation. Each model-backed condition uses isolated trace and result directories. This matters for both scientific validity and cost reporting. If traces from different model conditions share a directory, the evaluator can no longer distinguish missing rows from stale rows. If multiple OpenRouter model conditions share one account-usage window, per-model cost deltas cannot be reported cleanly. The benchmark's cost policy is therefore separate from scoring but aligned with the same principle: one named condition per auditable measurement window.

Promotion review. Promotion is not triggered by successful compilation or by the existence of a result JSON. A row is eligible for

publication evidence only after the scenario inventory matches the active contracts, the prompt metadata matches the declared condition, configured faults and invalidating issues are accounted for, and non-invalidating warnings remain visible. Historical rows that predate contract repairs or scenario additions remain internal evidence unless rerun or explicitly reviewed under a compatible manifest.

Local rerun path and limit. The local development workflow audits the static scenario appendix, scenario JSON, dry-run job construction, and synthetic calibration fixtures without paid model calls. A fresh model-backed rerun requires model calls, isolated directories, manifest capture, and post-run evaluation under a named current contract. It cannot independently reconstruct the historical execution-time ACID source snapshot because the named commits are no longer reachable. Stale, diagnostic, and nonmatching rows remain outside the reported evidence boundary.

6 Scoring and Validity Accounting

6.1 Metric Definitions

An *execution attempt* is one low-level run attempt. A *promoted final row record* is the post-retry scenario-run record included in the reported table. The five profiles contain 780 promoted final row records, while the underlying logs contain 800 execution attempts. Twenty failed attempts affected 14 rows; all 14 later produced completed final records after retry.

A row is *table-valid* when it satisfies the evaluator's implemented validity criteria and metadata checks. Reported *success* is ACID scenario-local assertion success, not Tau reward. Across the 780 promoted final records, the two labels disagree on 285 records: 44 are ACID-local successes with Tau failure, while 241 are ACID-local failures with Tau success. This establishes that the metrics are distinct, not that ACID-local labels are inherently superior. Transactional integrity is preserved when the final state has no integrity-violation label under the scenario contract. The implemented *Strict Clean Pass* metric is the intersection of a table-valid row, ACID-local success, and no integrity violation. It does not require the raw `fault_exposed` interaction outcome. SCP is an accounting intersection, not a measure of strict autonomous recovery. The reported total is 234/780, including 18 records whose separate interaction outcome is safe handoff.

Validity and interaction outcome remain separate. The 688 table-valid records comprise 554 fault-exposed outcomes and 134 safe handoffs; the other 92 records are invalid or non-exposure outcomes. No promoted final record carries `runtime_failure`. That final-record fact does not mean zero operational failures because the attempt logs contain the 20 failed attempts above. All headline rates are micro-averages over final scenario-run records. Family and domain slices are diagnostics rather than independent claims about every retail or airline workflow.

6.2 Why Validity and Interaction Outcome Are Separate

Raw fault exposure is an interaction outcome, not a synonym for table validity. In the reported evidence, 688 records are table-valid: 554 have a fault-exposed outcome and 134 have a safe-handoff outcome. The remaining 92 records are invalid or non-exposure outcomes. A non-exposure record does not support a recovery-under-fault claim, but it remains part of final-row accounting. Reporting validity and interaction outcome separately prevents non-exposure from being treated as exposed-fault recovery or silently removed from the denominator.

Synthetic fixture	Success	Target	Integrity
Assigned ceiling	52/52	52/52	preserved
Assigned handoff	0/52	52/52	preserved
Assigned retry	0/52	52/52	violated
Assigned stop-after-failure	0/52	52/52	preserved
Assigned task-only	52/52	52/52	violated

Table 5: Synthetic contract-summary fixtures with assigned intended labels. Fault-target coverage is an assigned summary field, not the runtime `fault_exposed` interaction outcome. These fixtures test aggregation and reporting separation only; they do not establish mechanical solvability or actual policy performance.

6.3 Quality Warnings

The evaluator also reports non-invalidating quality warnings. Examples include deviations from a canonical tool chain, missing post-write checks, missing pre-write reads, and recovery-budget overruns. These warnings remain visible because they affect interpretation, especially in the consequence track. They do not automatically invalidate a row unless the scenario contract marks the missing behavior as required for the intended claim. This avoids two distortions: hiding weak evidence behind a clean score, or treating every procedural deviation as equivalent to state corruption.

Secondary diagnostic index. The Descriptive Triage Index (DTI) is an optional diagnostic that decomposes non-corrupt behavior patterns, including implemented SCP, safe handoff, and fault-exposed progress. It is kept outside the main validation table because its weights are sensitivity-pending policy choices; the artifact documentation records the formula and sensitivity notes.

7 Empirical Calibration and Validation

7.1 Synthetic Contract-Summary Fixtures

ACID-Bench includes synthetic contract-summary fixtures with assigned intended labels, generated outside the model-condition validation surface. Their labels are not produced by executing an agent, oracle, simulator, or task-only policy. The fixtures test only whether aggregation and reporting preserve separations among ACID-local success, integrity, and assigned behavior categories. They do not establish mechanical solvability or actual policy performance.

Worked example. The assigned task-only fixture supplies 52/52 ACID-local success labels, 52/52 assigned fault-target coverage, and 52/52 integrity-violation labels, so the aggregator returns 0/52 SCP. This confirms that the reporting pipeline can keep assigned success and integrity labels separate. It is not an observation of an implemented task-only policy.

7.2 Model-Condition Profiles: Integrity Violations Dominate

The reported results comprise five replicated model-condition profiles spanning four model IDs and two inference routes. Each profile contains three current-contract 52-scenario runs. The manifest prevents accidental mixing of nonmatching evidence with the current scenario inventory. Repeated ablations, broader route and model coverage, and uncertainty analysis remain necessary for statistically decisive comparison claims.

The generated table's `Final rows` column reports 780 promoted final row records, 156 per profile, not a count of low-level attempts. Underlying execution logs contain 800 attempts. Twenty failed attempts affected 14 rows, and all 14 later produced completed final records after retry. No promoted final record carries `runtime_failure`, but the attempt history precludes interpreting this as zero operational failures.

Validity and interaction outcome are separate. The 688 table-valid records comprise 554 fault-exposed outcomes and 134 safe handoffs; 92 records are invalid or non-exposure outcomes. Of the 525 integrity-violation labels, 335 occur on fault-exposed outcomes, 98 on safe-handoff outcomes, and 92 on invalid or non-exposure outcomes. Thus, not all 525 labels are evidence of integrity failure after an exposed fault.

The implemented Strict Clean Pass total is 234/780, including 18 records also classified as safe handoff. Across profiles, implemented SCP ranges from 57/156 to 36/156. The descriptive row-level Wilson 95% intervals for this implemented metric are 36.5% [29.4%, 44.3%] and 23.1% [17.2%, 30.3%], respectively. The intervals overlap and do not estimate between-run model or provider variance. These counts provide cross-condition replay and internal-consistency evidence for the deterministic evaluation and accounting framework, not construct validity or an ordering claim. The underlying row evidence and raw traces are not publicly released.

What the table shows. The table's diagnostic value is the split among promoted final records, table validity, interaction outcome, integrity labels, and chain warnings. Safe handoff and SCP are not mutually exclusive: 18 safe-handoff records satisfy the implemented SCP intersection. Likewise, an integrity label can occur on a fault-exposed, handoff, or invalid/non-exposure outcome. A single ACID-local success number would hide these differences. The current table body is intentionally limited to manifest-reviewed replicated profiles: it prevents nonmatching rows from being mixed with the frozen scenario contracts. Future promoted rows must still report invalid exposure as its own evidence class rather than treating non-exposure as clean recovery or confirmed model failure.

Consequence diagnostics. The consequence track is stricter than ACID-Core because it asks whether the agent had enough pre-write evidence, verified after acting, stayed within a recovery budget, and avoided user-harm penalties. The relevant finding is diagnostic rather than comparative: the benchmark surface is designed to show whether a run mostly hands off, exposes faults, gathers evidence, verifies outcomes, or overruns a recovery budget.

Interpretation. The current accounting supports bounded measurement claims. The synthetic fixtures show that the aggregator can preserve assigned success and integrity labels, but they do not establish scenario solvability or policy behavior. The final-row table separately reports validity, interaction outcome, integrity, and warnings, while low-level attempt failures remain separate operational evidence. These categories support audit without reducing each condition to one score.

7.3 Prompt Condition Ablation: Overlay Matters

Prompt overlays are part of the named run condition, not hidden cleanups. Publication/default runs use the ACID clarification overlay and a deterministic scripted-user policy. Trace metadata records the overlay set, overlay hash, applied overlay ids, base instruction hashes, and effective instruction hashes. If a scenario declares prompt overlays but a run omits or mismatches them, the evaluator invalidates the trace for that condition. Some overlays ask the interaction to re-check before retrying or verify after writes, so those rows measure behavior under a clarified instruction condition rather than independent discovery of recovery strategy.

The 20-scenario paired ablation directly tests whether ACID clarification overlays are merely metadata or whether they materially shape behavior. ACID clarification overlays are not a cosmetic prompt setting; they materially change measured recovery behavior: across three replicates, the canonical DeepSeek V4 Flash thinking-max condition received 27/60 implemented SCP labels under the clarification overlay and 2/60 implemented SCP labels under the base Tau prompt with no overlay. The overlay arm has 52/60 table-valid records, decomposed into 45 fault-exposed outcomes and 7 safe handoffs. The base arm has 41/60 table-valid records, decomposed into 34 fault-exposed outcomes and 7 safe handoffs. The correct interpretation is not that the base Tau condition is a worse model comparison row. It is that ACID-Bench's default validation condition is a controlled, clarified measurement condition, and overlay effects must be reported as part of the measurement contract.

Across the 60 matched scenario-replicate pairs, 27 were overlay-only SCP, 2 were base-only SCP, 0 were SCP under both conditions, and 31 were SCP under neither. These paired counts describe the fixed model condition, scenario subset, replicate structure, tool environment, and scripted-user protocol. They do not establish a general prompt effect beyond that protocol.

This ablation is therefore evidence for a measurement-design claim rather than a model-quality claim: prompt condition must be treated as part of the benchmarked system. Merging clarified and unclarified runs would overstate comparability, while omitting prompt-condition metadata would make the result difficult to audit. This is consistent with treating the benchmarked system as compound: model behavior, prompt policy, and tool environment are jointly measured, not independently scored.

Each prompt arm contains three runs and 60 promoted final row records. Neither arm has a final-record runtime label; the low-level attempt history remains separate. Table 7 therefore keeps only the columns needed to compare the two prompt conditions legibly.

7.4 Reading Future Validation Rows

The validation table is compact because each column protects a different interpretation boundary. Promoted final row records are

Condition	Final rows	Table-valid	Fault-exposed outcomes	Safe handoff outcomes	Integrity labels	Implemented SCP	Chain warnings	Final runtime labels
DeepSeek non-thinking	156	139	122	17	98	53/156 (0.340)	58	0
DeepSeek thinking-max	156	140	120	20	96	57/156 (0.365)	55	0
DeepSeek Pro thinking-max	156	142	110	32	105	49/156 (0.314)	63	0
MiniMax M2.7 reasoning-low	156	136	107	29	109	39/156 (0.250)	76	0
Kimi K2.6 high	156	131	95	36	117	36/156 (0.231)	80	0

Table 6: Reviewed current-contract model-condition accounting. In this generated table, `Final rows` denotes promoted final row records, not low-level execution attempts. Validity, interaction outcome, integrity labels, warnings, and implemented Strict Clean Pass remain separate. Rows must match the scenario inventory and pass manifest review before promotion.

Prompt	Valid	Fault	Handoff	SCP
Clarification overlay	52/60	45	7	27/60
Base Tau prompt	41/60	34	7	2/60

Table 7: Replicated 20-scenario prompt-condition ablation over the reviewed `overlay-ablation-20-v1` scenario set. The surrounding text reports the fixed model condition, run count, final-record denominator, and final runtime-label count.

the reporting denominator; underlying attempts and retries are the operational execution history. Table validity is separate from interaction outcome. In the current evidence, the 688 table-valid records split into 554 fault-exposed outcomes and 134 safe handoffs, while 92 records are invalid or non-exposure outcomes. Chain warnings are trace-shape diagnostics. SCP is the implemented intersection of table validity, ACID-local success, and no integrity label.

This means the table should be read as an accounting profile, not as a sorted score sheet. A single condition can have low SCP because of table invalidity, failed ACID-local assertions, or integrity labels. Interaction outcome must be read separately because 18 SCP records are safe handoffs. The consequence-track diagnostics ask additional questions about pre-write evidence, post-write verification, bounded recovery, and exploratory harm penalties.

The current evidence surface therefore provides calibration and cross-condition evidence for the accounting scheme, not a basis for settling model or provider choice. Sparse strict passes mean small count differences are fragile. Three replicated runs per profile do not estimate between-run model or provider variance. Prompt overlays and deterministic scripted-user behavior are named conditions rather than neutral background assumptions. Secondary index values change when handoff and fault-progress weights change, so they are kept outside the main validation table and behind the core behavioral counts. These facts do not weaken the benchmark artifact; they are exactly why the benchmark reports the underlying evidence categories instead of a single opaque score.

7.5 Claim Boundary and Future Evidence

The current evidence supports a measurable gap between ACID-local success and transactional integrity, while preserving the limits of each label. Of 525 integrity labels, 335 occur on fault-exposed outcomes, 98 on safe-handoff outcomes, and 92 on invalid or non-exposure outcomes. The attempt logs separately record 20 failed attempts affecting 14 rows before completed retries. Synthetic fixtures provide aggregation checks only. Together, these surfaces separate final-row accounting from execution history without treating every integrity label as exposed-fault evidence.

The current evidence does not establish stable model or provider selection guidance. That claim would require broader inference-route and model coverage, uncertainty analysis, stable configurations, and repeated no-overlay ablations across additional conditions if prompt-overlay effects are part of the question. The benchmark is ready to support that style of evidence; this paper reports the narrower source artifact and current-contract validation state.

Next empirical evidence. Future model-backed extensions should add broader inference-route and model coverage, repeated prompt-condition ablations, and uncertainty analysis, promoted only after manifest review.

8 Artifact Use, Ethics, and Limits

8.1 Intended Use

ACID-Bench is intended for evaluating a narrow reliability construct: state-changing tool-agent behavior under deterministic transactional faults. A reader can use it to audit whether a run exposed the intended fault, whether final state was preserved, whether the agent escalated safely, and whether evidence or verification obligations were met. It is not a deployment-safety certification, a formal database ACID-compliance test, or a claim that the retail and airline domains cover every consequential setting.

8.2 Artifact Boundary

The artifact boundary is part of the ethics story. The implementation and 52 checked scenario contracts are prepared locally and intended for MIT licensing. Manifest-reviewed local validation evidence underlies the paper's reported tables, but row evidence, raw traces, and local research materials are not publicly released. This boundary protects participants and readers from accidental leakage and from result rows that do not match the current benchmark contract. A durable public source and reviewed archival validation release remain future work.

8.3 Limitations and Future Work

Domain coverage. The current tasks are retail and airline workflows. They cover concrete state changes but not payments, healthcare, industrial control, or long-running enterprise processes.

Construct validity. ACID is used as an agent-reliability lens, not a claim about the simulator's database internals. Scenario contracts must remain causally tied to the user goal; otherwise the benchmark can invalidate traces for a fault path the task did not naturally require. Local validation review included an 18-row trace-audit sample, with seven compact invalid-exposure edge-case labels where exact prerequisite or trigger matching is stricter than the semantic task path. Those local materials are not publicly released. We treat such rows as invalid exposure and measurement-calibration evidence, not as successful recovery and not as confirmed model failure.

Prompt and scripted-user effects. Prompt overlays and deterministic user policies improve control and reproducibility, but they also shape behavior. The paired no-overlay result shows that recovery behavior changes sharply when ACID clarifications are removed. For this reason, the default model-condition profiles should be read as clarified-condition evidence under the implemented metric, and no-overlay evidence should be reported separately rather than merged into the same comparison surface.

Lifecycle and governance frameworks. Real-time post-market monitoring is most useful when it is anchored to clear pre-deployment behavioral baselines. ACID-Bench provides such a baseline for transactional integrity: before monitoring drift, regressions, or API-induced behavior changes in deployed agents, evaluators need a deterministic way to test whether an agent preserves external state under known transactional faults. In this sense, ACID-Bench is complementary to lifecycle monitoring rather than a replacement for it: it defines auditable failure modes and trace-level metrics that can later be reused in continuous evaluation pipelines.

Model-condition and sampling scope. The five three-replicate model-condition profiles span four model IDs and two inference routes. They provide cross-condition evidence about the implemented accounting labels, but they do not justify decisive model or provider comparison claims. SCP is an accounting intersection under the current evaluator, not an autonomous-recovery or automation-readiness measure. Future selection claims require broader inference-route and model coverage, repeated ablations, uncertainty analysis, and stable configurations.

Metric design. Evidence, verification, budget, harm, and secondary index weights are benchmark-design choices. They are useful for diagnostics, but they are not calibrated economic or safety utilities.

Reproducibility. Promotion requires isolated trace and result directories, the same model config for running and evaluation, prompt-condition metadata, scenario inventory checks, and manifest review. Scenario, manifest, result, and table hashes survive. Historical ACID commit hashes remain in the manifests, but exact run-time source snapshots are not reachable from current Git history after the privacy rewrite. A future archival release must preserve reviewed source snapshots in addition to hashes and identifiers.

Ethics. ACID-Bench evaluates agents in simulated environments and should not be presented as deployment certification. The benchmark is intended to reduce overclaiming by exposing state-corruption risks, not to imply that passing the suite is sufficient for production use in consequential domains.

Falsifiability. The benchmark's claims would weaken if reviewer audit found many injection points that are not causally required by the user goals, if trigger matching repeatedly blocked semantically valid task paths, if no-overlay and clarified prompt conditions produced the same failure modes after broader replication, or if the synthetic fixture labels failed to aggregate as specified. Those failure modes would be measurement evidence rather than model-quality evidence.

9 Conclusion

ACID-Bench shows that ACID scenario-local success, transactional integrity, table validity, and interaction outcome are distinct accounting dimensions for state-changing tool agents. The five model-condition profiles contain 780 promoted final row records across four model IDs and two inference routes. Underlying logs contain 800 attempts, including 20 failed attempts affecting 14 rows before completed retries. No promoted final record carries `runtime_failure`, which does not mean zero operational failures. Of 525 integrity labels, 335 occur on fault-exposed outcomes, 98 on safe handoffs, and 92 on invalid or non-exposure outcomes. Implemented SCP is an accounting intersection totaling 234/780, including 18 safe handoffs. The fixed-protocol ablation reports 27/60 SCP with clarified instructions versus 2/60 under the base prompt, with paired counts 27 overlay-only, 2 base-only, 0 both, and 31 neither. Synthetic fixtures check aggregation only and do not establish scenario solvability. These results support validity accounting and prompt-condition reporting, not decisive model or provider ranking.

References

[1] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. Tau2-Bench: Evaluating Conversational Agents in a Dual-Control Environment. *arXiv preprint arXiv:2506.07982* (2025). doi:10.48550/arXiv.2506.07982

[2] Leo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. 2024. WorkArena++: Towards Compositional Planning and Reasoning-based Common Knowledge Work Tasks. *arXiv preprint arXiv:2407.05291* (2024). doi:10.48550/arXiv.2407.05291

[3] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Leo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. WorkArena: How Capable Are Web Agents at Solving Common Knowledge Work Tasks? *arXiv preprint arXiv:2403.07718* (2024). doi:10.48550/arXiv.2403.07718

[4] Malcolm Featonby. 2020. Making Retries Safe with Idempotent APIs. Amazon Builders' Library. https://aws.amazon.com/builders-library/making-retries-safe-with-idempotent-apis/.

[5] Hector Garcia-Molina and Kenneth Salem. 1987. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*. ACM, 249–259. doi:10.1145/38713.38742

[6] Theo Haerder and Andreas Reuter. 1983. Principles of Transaction-Oriented Database Recovery. *Comput. Surveys* 15, 4 (1983), 287–317. doi:10.1145/289.291

[7] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer. 1997. Fault Injection Techniques and Tools. *Computer* 30, 4 (1997), 75–82. doi:10.1109/2.585157

[8] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *arXiv preprint arXiv:2310.06770* (2023). doi:10.48550/arXiv.2310.06770

[9] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. AgentBench: Evaluating

LLMs as Agents. *arXiv preprint arXiv:2308.03688* (2023). doi:10.48550/arXiv.2308.03688

[10] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. GAIA: A Benchmark for General AI Assistants. *arXiv preprint arXiv:2311.12983* (2023). doi:10.48550/arXiv.2311.12983

[11] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. 2024. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. In *The Twelfth International Conference on Learning Representations*. doi:10.48550/arXiv.2309.15817

[12] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025. BrowseComp: A Simple Yet Challenging Benchmark for Browsing Agents. *arXiv preprint arXiv:2504.12516* (2025). doi:10.48550/arXiv.2504.12516

[13] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. Tau-Bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv preprint arXiv:2406.12045* (2024). doi:10.48550/arXiv.2406.12045

[14] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2023. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint arXiv:2307.13854* (2023). doi:10.48550/arXiv.2307.13854