

When a Tool Times Out: Recovery Contracts for Stateful AI Agents

Veera Ravindra Divi
Independent Researcher
Austin, TX
ravi3divi@gmail.com

Abstract

A tool timeout leaves an AI agent with two uncertainties: whether an action happened and whether repeating it is safe. A language model can generate a plausible recovery plan while creating a second purchase, refund, message, or infrastructure change. This white paper proposes a recovery contract that separates model planning from durable action identity, authoritative outcome reconciliation, and permission to resume. It specifies a state machine, adapter requirements, and a fault-injection evaluation protocol focused on externally visible side effects. The proposal combines established distributed-systems principles with recent agent-reliability evidence; it does not claim a new transaction primitive. No implementation or empirical evaluation is reported. Its practical contribution is a testable architecture for making recovery decisions auditable and for distinguishing safe suspension from successful completion.

The failure that a retry hides

An agent asks a procurement service to create an order. The service commits the order, but its response disappears. The agent sees a timeout, infers that the action failed, and creates another order using a new request identifier. Both orders are valid from the service perspective. The agent may even report a successful recovery. The defect is a mismatch between what the agent knows and what the external system has done. This is an illustrative scenario, not an observed incident.

This problem predates language models. Amazon's engineering guidance explains why a timed-out request can still produce a side effect, and why safe retries require an appropriate idempotency contract [7], [8]. Agents add a distinctive failure path: instead of replaying a fixed request, they can rewrite parameters, choose another tool, delegate the action, or interpret an unverified narrative as evidence. A retry loop around the model therefore cannot substitute for an action protocol.

The central recommendation is to make uncertain outcomes explicit and durable. Before the agent can resume dependent work, a controller must establish whether the original action committed, definitely did not commit, or remains unresolved. If the adapter cannot establish an outcome or safely replay the same intent, the workflow should suspend. Suspension is a valid safety outcome, but it must not be counted as task success.

What the literature establishes

Recent research supports examining reliability separately from benchmark accuracy. Rabanser and colleagues organize agent reliability around consistency, robustness, predictability, and safety rather than one success score [1]. Their fault-injection experiments motivate stress testing, but synthetic fault probabilities and recovery rules are not estimates of a particular production service. A deployment needs its own failure model and its own definition of unacceptable side effects.

Yao and colleagues' τ -bench evaluates terminal database state and repeated-trial reliability [4]. This is a stronger starting point for transactional workflows than judging whether a final answer sounds successful. Cemri and colleagues identify system design, inter-agent misalignment, and task-verification failures [5]. AgentRx generates evidence-bearing constraint violations to help localize failed trajectories [2]. Its diagnostic purpose differs from enforcing a commit boundary during execution.

Yang and colleagues show that agents can adopt corrupted tool returns across search, delegated agents, and code execution; their evaluated interventions provide partial mitigation [3]. ReliabilityBench separately explores perturbations and tool failures [6]. Together these studies motivate distrust of unverified outcomes. They do not establish that an LLM can infer whether an external mutation committed after a lost response. That question requires a service contract or authoritative observation.

The contribution here is consequently a systems synthesis and benchmark specification. Idempotency keys, fencing, durable logs, and reconciliation are established mechanisms. The proposed distinction is where they constrain agent behavior: an unresolved action becomes a barrier to semantic replanning, cross-tool replacement, and downstream success reporting. The proposal requires implementation and evaluation before any claim of improved reliability or acceptable overhead.

Define the recovery contract before execution

Each mutating tool adapter should publish five properties: its effect class, supported intent identifier, authoritative outcome lookup, deduplication scope and retention, and available compensation. Effect classes distinguish read-only operations, naturally idempotent writes, token-deduplicated writes, compensable effects, and irreversible effects. These labels are properties of a documented API and its actual deployment; the model must not assign them from a tool name.

Before dispatch, the controller durably records an action envelope containing workflow identity, logical action identity, actor and tenant, canonical arguments, argument digest, tool and schema version, authorization reference, deadline, and proposed postcondition. A logical action identifies one authorized intent, not one network attempt. Retransmission retains that identity. A genuinely new intent receives a different identity only after the old action is resolved or explicitly accepted as unresolved by an authorized operator.

The digest binds retries to the same argument semantics and detects accidental parameter changes. It is not itself the deduplication key: two intentionally identical orders may be legitimate. A durable unique intent identifier expresses that distinction. This follows the caller-provided identifier approach described in [7]. Canonicalization must be schema-specific; sorting JSON keys alone does not normalize currencies, time zones, omitted defaults, or account identifiers.

Admission checks the token's current age, remaining retention, possible delivery delay, and replay horizon against the adapter's documented deduplication guarantee; remaining workflow time alone is insufficient. Retention expiration is a semantic boundary, not merely a cache eviction. A replay after the service forgets a token can become a new action. The controller must therefore refuse automatic replay when token validity is uncertain and reconciliation is unavailable. A request deadline cannot retract work already accepted by a remote service.

A state machine that preserves uncertainty

The proposed controller uses PREPARED, DISPATCHED, UNKNOWN, COMMITTED, NOT_COMMITTED, SUSPENDED, and COMPENSATED states. PREPARED means the envelope is durable and no dispatch has been attempted by the controller. DISPATCHED means an attempt may be in flight. A timeout, connection loss, or controller crash after dispatch produces UNKNOWN unless a documented response proves an outcome. COMMITTED requires trusted evidence for the requested postcondition. NOT_COMMITTED requires an authoritative negative outcome with sufficient finality to exclude a late commit. Before any network send, the controller durably enters DISPATCHED; a crash thereafter is reconciled as UNKNOWN even if no request actually left the process. This write-ahead rule prevents a restarted controller from treating an already-sent action as safely unattempted.

A validated pre-dispatch rejection can establish NOT_COMMITTED, but a generic error message cannot. A status lookup returning no current record may reflect replication lag, an in-flight request, or a lookup using the wrong identifier. The adapter must define the visibility and finality of a negative lookup. Without that guarantee the state remains UNKNOWN. The model receives the uncertainty explicitly rather than a generic failure string that encourages a fresh attempt.

In UNKNOWN, the controller first queries authoritative status using the original intent identity. If the service supports token-deduplicated replay and the token remains valid, it may replay the original envelope within a bounded attempt budget. Otherwise it suspends for reconciliation or human handling. A substitute purchase through another vendor is a new effect and is blocked while the original purchase is unresolved. Changing the tool does not erase the original uncertainty.

COMMITTED releases dependent actions only when their prerequisites remain valid. The controller may need to recheck price, inventory version, permission scope, or user intent before continuing. NOT_COMMITTED permits a new plan under current authorization. COMPENSATED records a confirmed compensating effect against an earlier

committed action; it does not rewrite history to pretend nothing occurred. SUSPENDED retains all identifiers and evidence so another process can continue the same reconciliation.

Figure 1 summarizes the recovery path after an ambiguous mutation. The outcome states describe evidence about the external effect, not the model's confidence.

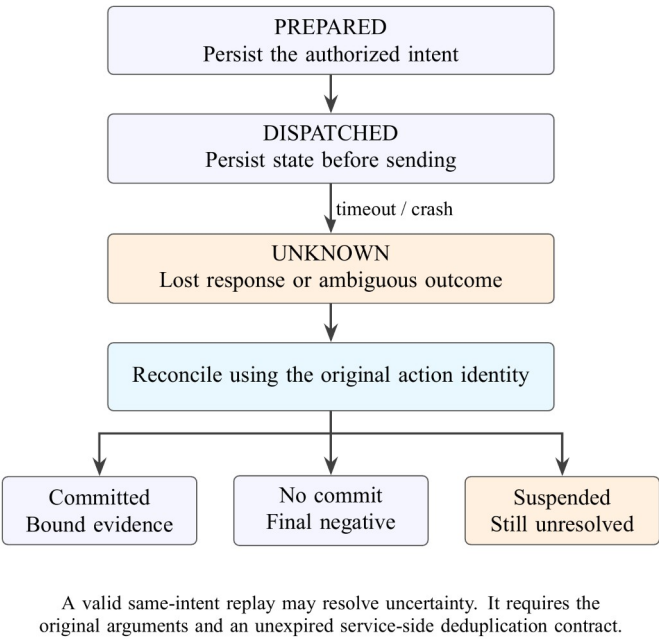


Figure 1. Recovery after an ambiguous mutation. A timeout preserves uncertainty; only authoritative evidence or a valid same-intent replay may resolve it. Suspension preserves safety without counting as task completion.

Table 1. Proposed recovery transitions. A safety state is not task completion.

State	Required evidence	Permitted next step
Prepared	Durable intent; no send	Persist DISPATCHED before send
Unknown	Outcome unresolved	Lookup, valid same-intent replay, or suspend
Committed	Bound receipt and postcondition	Recheck prerequisites; release dependents
Not committed	Final authoritative negative	Replan under current authority
Suspended	Evidence unavailable	Retain identity; reconcile later
Compensated	Confirmed separate effect	Record history; do not erase original

Concurrency, crashes, and the limits of local control

A durable log alone does not prevent two workers from dispatching the same logical action. The controller needs an atomic claim on workflow progress, with a lease or ownership epoch and compare-and-set transitions. A stale worker can still have an in-flight network request after losing ownership. Therefore a local ownership change is insufficient for at-most-once external effects: service-side deduplication or a server-enforced fencing contract must handle overlap.

The dispatcher records the envelope before sending and persists outcome evidence before advancing the workflow. A crash between these steps creates an uncertain action, which the replacement worker reconciles. An outbox pattern can make local intent publication atomic with local state, but it cannot make an arbitrary external API participate in the same database transaction. The architecture deliberately retains this boundary rather than asserting end-to-end exactly-once execution.

An adapter may promise that valid replays produce no additional effect only within a defined caller scope, region, resource scope, and retention interval. Tests must include each boundary. The controller must not claim stronger

guarantees than the service provides. If the service lacks a token and has no authoritative lookup, automatic recovery of a non-idempotent mutation cannot be made safe merely by improving the prompt. Suspending that operation is the honest behavior.

Compensation requires another action envelope and can itself time out. A cancellation is not always an inverse: a message has already been read, a fee may remain, or fulfillment may have started. The controller records compensation preconditions and unresolved compensation outcomes separately. It must not compensate an UNKNOWN original action based only on speculation, because the compensating action could affect an unrelated resource or race with a late commit.

Evidence the planner may use

The planner receives a structured recovery view: logical action identity, controller state, allowed next operations, evidence references, unresolved assumptions, and remaining budget. It can propose diagnostics or explain the situation, but it cannot assert COMMITTED by generating text. State transitions belong to the controller and validated adapter responses. Success reporting should cite the resolved action identity and its postcondition evidence.

Transport authentication proves which service supplied a response; it does not prove the underlying business state is correct. A receipt must be bound to the correct actor, resource, arguments, and intent. Where a postcondition is observable, the controller verifies it through a documented endpoint or trusted event stream. A receipt for the wrong order is not sufficient. Contradictory evidence reopens reconciliation rather than being resolved by whichever sentence the model finds more persuasive.

This division also supports diagnosis. AgentRx illustrates the value of evidence-bearing validation logs [2], but an LLM judge can mistake downstream symptoms for root causes. The recovery log should therefore retain raw protocol facts separately from model interpretations: dispatch timestamps, token scope, lookup version, transition reason, and receipt digest. Sensitive payloads need access controls and bounded retention; hashes do not automatically anonymize low-entropy personal information.

A benchmark specification with observable side effects

The proposed benchmark uses a controlled service emulator whose ground-truth ledger is hidden from the agent and visible to the evaluator. Each task defines an authorized goal, permitted actions, external-effect constraints, and a completion deadline. Procurement, refunds, reservation updates, and infrastructure provisioning provide different effect semantics. These are candidate task families, not a released dataset.

Fault injection must distinguish failure before acceptance, failure after durable commit but before response delivery, duplicate delivery, delayed delivery after the client deadline, stale status reads, token expiration, controller crash, worker overlap, and successful transport with a semantically wrong receipt. Pre-commit and post-commit response loss are separate conditions. A benchmark that replaces both with the same exception cannot identify whether a recovery policy duplicates effects.

Compare a fixed bounded-retry baseline, model-directed replanning, and the proposed contract controller using the same model, tools, authorization rules, and token budget. Add an oracle-outcome controller only as an upper reference, because real agents do not observe the hidden ledger. Ablations remove argument binding, authoritative lookup, or the uncertainty barrier individually. When removing service deduplication, report the changed assumptions rather than treating it as a controller-only comparison.

Primary outcomes should be reported separately: authorized task completion by deadline; duplicate or unauthorized side effects per initiated workflow; false success reports; unresolved outcomes at deadline; and time to reconcile. Also report token cost, tool attempts, human interventions, and completion conditional on no safety violation. A controller that suspends every task can have no duplicates and still be useless. A controller that succeeds after duplicating an order must not receive an unqualified success label.

Use paired task/fault schedules across policies, multiple independent stochastic runs, and confidence intervals clustered by task or workflow family. Declare sample size and exclusion rules before evaluation. Report each fault class independently, including all failed and suspended runs. Rare-event uncertainty matters: observing zero duplicate effects in a finite test does not establish a zero production risk. A statistical upper bound must state its independence assumptions and denominator.

Deployment decisions and open questions

Adoption can begin with one high-impact mutating adapter rather than a complete agent framework replacement. First document its idempotency semantics and status consistency. Then implement durable identities and UNKNOWN handling, shadow-test reconciliation against a non-mutating replay environment, and inject post-commit response loss. Permit automatic replay only for the adapter classes whose guarantees have been exercised. Monitor suspension and reconciliation rates alongside task completion.

The main costs are additional storage, status calls, adapter engineering, and longer latency under uncertainty. Existing APIs may not expose enough evidence. Permission changes during reconciliation can prevent a lookup that was previously allowed. Human operators can also issue a duplicate action outside the controller, so the action identity should be visible in support tools. The boundary of the safety claim must include all actors permitted to produce the same effect.

This white paper provides no measured overhead, production failure frequency, or evidence that the proposed contract outperforms existing orchestrators. Its protocol depends on correct adapters, durable storage, authorization enforcement, and honest service semantics. Byzantine services, compromised credentials, and cross-service atomic transactions require additional mechanisms. The proposal is intended to make these dependencies explicit and testable.

For stateful agents, recovery quality depends on preserving what remains unknown. The useful boundary is concrete: after an ambiguous mutation, the agent may gather evidence or replay a documented idempotent intent, but it may not silently replace that intent and declare success. Testing that boundary against the external ledger turns a reassuring recovery narrative into an engineering claim that can be checked.

Acknowledgment

Prepared as an independent research white paper. The views expressed do not represent an employer. Proposed mechanisms require implementation and empirical validation.

References

- [1] S. Rabanser et al., "Towards a Science of AI Agent Reliability," ICML, 2026; arXiv:2602.16666v3. <https://arxiv.org/abs/2602.16666v3>
- [2] S. Barke et al., "AgentRx: Diagnosing AI Agent Failures from Execution Trajectories," arXiv:2602.02475v2, 2026; accepted to EMNLP Findings per manuscript metadata. <https://arxiv.org/abs/2602.02475v2>
- [3] H. Yang et al., "Agents' Overreliance on Unreliable Tools," arXiv:2609.05587v2, 2026, preprint. <https://arxiv.org/abs/2609.05587v2>
- [4] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan, "τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains," arXiv:2406.12045, 2024. <https://arxiv.org/abs/2406.12045>
- [5] M. Cemri et al., "Why Do Multi-Agent LLM Systems Fail?," arXiv:2503.13657v3, 2025. <https://arxiv.org/abs/2503.13657v3>
- [6] A. Gupta, "ReliabilityBench: Evaluating LLM Agent Reliability Under Production-Like Stress Conditions," arXiv:2601.06112, 2026, preprint. <https://arxiv.org/abs/2601.06112>
- [7] M. Featonby, "Making retries safe with idempotent APIs," Amazon Builders' Library, undated; accessed Oct. 27, 2026. <https://aws.amazon.com/builders-library/making-retries-safe-with-idempotent-APIs/>
- [8] M. Brooker, "Timeouts, retries, and backoff with jitter," Amazon Builders' Library, Dec. 5, 2019. <https://aws.amazon.com/builders-library/timeouts-retries-and-backoff-with-jitter/>