Semantic State Diffs as an Evaluation Layer for Long-Horizon Agents

George Weale
Columbia University
New York, NY, USA
george.weale@columbia.edu

***Abstract*—Long-horizon agents are often evaluated by final text, task labels, or a binary success signal. Those measures can overlook a central property of tool-using systems: an agent changes external state. It may create a ticket with the wrong owner, update the correct record with an invalid field, duplicate a payment, or make a harmless-looking but irreversible change. This paper develops semantic state diffs, an evaluation layer that compares observed and expected state transitions under an explicit task contract. The framework represents state as typed entities and relations, normalizes only declared representation differences, assigns tri-state verdicts, and retains evidence for every comparison. It defines metrics for required changes, forbidden changes, uncertainty, and irreversible side effects, together with a benchmark methodology and statistical protocol. Semantic state diffs make outcome evaluation sensitive to both task completion and the safety of external effects.**

***Index Terms*—agent evaluation, state transition, tool use, semantic equivalence, provenance, long-horizon agents**

I. Introduction

Agents increasingly act through browsers, repositories, databases, and business APIs. Existing benchmarks have advanced realistic evaluation of tool use and web interaction [1], [2], while software-engineering benchmarks test issue resolution against executable repositories [3]. Still, a textual final answer can be correct when state is wrong, and a superficially successful action can conceal extra or irreversible effects. A rigorous evaluation therefore accounts for both an agent's response and the state transition it produces.

This paper develops a semantic state-diff layer for that evaluation. The layer evaluates a declared set of state entities, permitted equivalences, and side-effect rules. It makes the task boundary legible by specifying required changes, prohibited changes, unavailable observations, and permitted normalizations.

II. Motivation and Related Work

The motivation is a mismatch between the action surface of an agent and the evidence used to judge it. A task such as "update the customer record and notify the owner" has several independently meaningful effects: the target identity must be correct, the selected fields must change, the owner relation must be valid, notification delivery must be observed or explicitly unknown, and unrelated records must remain unchanged. A final language response may summarize the intended action accurately while one of those effects is absent or while an extra destructive action has occurred. Conversely, exact byte-for-byte comparison can reject a correct change when an environment rotates an opaque identifier or records a timestamp at a different approved resolution.

Agent benchmarks have raised the realism of this evaluation setting. WebArena uses realistic websites and user tasks [1]; OSWorld evaluates actions across real computer environments [2]; AgentBench collects heterogeneous environments [6]; and GAIA emphasizes questions requiring external tools and multi-step reasoning [7]. SWE-bench grounds a class of software tasks in executable repositories and tests [3]. These benchmarks provide valuable outcome signals, but a test suite, webpage screenshot, or final answer does not automatically expose every business-relevant state effect. The state-diff layer composes with these benchmarks by adding explicit external-state evidence.

The design also draws on transaction processing. A completed request is not synonymous with a completed state transition when network partitions, retries, or asynchronous work are involved. Transaction systems distinguish durable commit, retry behavior, isolation, and recovery [5]; sagas make compensating actions explicit when a long-running task cannot be atomic [10]. Those concepts motivate the framework's reversibility and evidence fields. They encourage an evaluator to record what was observed, what is safely repeatable, and which effects require special review across third-party APIs.

Reliable systems design also depends on preserving uncertainty rather than coercing it into a success label. Lampson's guidance on system design emphasizes explicit interfaces and failure handling [11]. The NIST AI Risk Management Framework similarly foregrounds documented context and measurement limits [12]. The tri-state verdict in this paper translates that stance into an evaluator: an absent or stale observation is an inconclusive outcome, not an opportunity for a friendly answer model to guess that a state transition probably occurred.

III. Contract-Scoped State Model

Let the observable state before and after a run be typed graphs

$$S_t = (N_t, E_t, \tau, \alpha), \quad t \in \{0, 1\}, \tag{1}$$

where nodes N_t represent entities, edges E_t represent typed relations, τ assigns an entity type, and α maps attributes to

typed values. A task contract is

$$\mathcal{C} = (\Omega, \Phi, \nu, \Gamma, \Lambda, \mathcal{A}). \tag{2}$$

Here Ω bounds observable entities and fields; Φ is the set of required postconditions; ν is a versioned canonicalization function; Γ is a set of forbidden deltas; Λ labels reversibility and authorization requirements; and $\mathcal{A}$ specifies evidence sources and freshness constraints. The contract is fixed before evaluation. A post-hoc normalizer that suppresses a defect is a changed oracle, not a harmless implementation detail.

The raw state difference is $\Delta(S_0, S_1)$. The semantic difference is

$$\delta_{\mathcal{C}}(S_0, S_1) = \Delta(\nu_{\mathcal{C}}(S_0), \nu_{\mathcal{C}}(S_1)) \restriction \Omega. \tag{3}$$

Canonicalization may, for example, remove declared ephemeral identifiers or compare timestamps within an approved resolution. It MUST NOT merge distinct principals, erase unauthorized writes, or ignore an error simply because a final answer sounds plausible.

Each atomic delta is a typed tuple $d = (op, id, path, v_0, v_1, \ell, e)$, where $op \in \{\mathsf{create}, \mathsf{update}, \mathsf{delete}, \mathsf{link}, \mathsf{unlink}\}$; $\ell \in \{\mathsf{reversible}, \mathsf{conditional}, \mathsf{irreversible}\}$; and e identifies retained evidence. This representation supports entity-level and field-level comparison, rather than flattening all outcomes to text.

IV. Verdicts and Metrics

Let R be the required semantic delta set and O the observed delta set after canonicalization. The evaluator returns

$$V_{\mathcal{C}} \in \{\mathsf{MATCH}, \mathsf{DIVERGE}, \mathsf{INCONCLUSIVE}\}. \tag{4}$$

MATCH requires that every required predicate in Φ holds, no forbidden predicate in Γ holds, and all necessary evidence satisfies $\mathcal{A}$. DIVERGE requires a witnessed unmet requirement, forbidden effect, or contract violation. INCONCLUSIVE is mandatory when evidence is missing, stale, contradictory, or outside the observation boundary. Forced binary scoring would otherwise reward systems that hide uncertainty.

Contracts should support both exact predicates and relational predicates. An exact predicate might require a designated status field to equal an approved value. A relational predicate might require that an ownership edge point to a principal satisfying a role constraint, without requiring a particular opaque account identifier. Contracts should also state negative requirements, such as "no invoice may be created" or "no permission may broaden." This is essential because a delta set that includes all required changes can still contain a harmful extra write.

Evidence freshness is part of the method, not an implementation afterthought. For a state observation $a \in \mathcal{A}$, the contract should set a collection time, source identity, and acceptable staleness interval. If an asynchronous API has acknowledged a write but the authoritative state read is unavailable, the evaluation is inconclusive unless the contract explicitly names the acknowledgement as sufficient evidence. This prevents the same raw tool response from being overinterpreted across tasks with different risk profiles.

For interpretable reporting, define required-change recall and precision

$$P_R = \frac{|O \cap R|}{|O|}, \qquad Q_R = \frac{|O \cap R|}{|R|}, \tag{5}$$

but report them alongside a weighted forbidden-effect rate

$$F = \frac{\sum_{d \in O \cap \Gamma} w(\ell(d))}{\max(1, \sum_{d \in O} w(\ell(d)))}. \tag{6}$$

Weights should be predeclared, with irreversible changes receiving greater scrutiny. Equations (5)–(6) are diagnostic summaries, not a license to trade one prohibited destructive action for many correct edits.

V. Evaluation Methodology

The evaluation constructs sandboxed tasks in at least three domains: issue tracking, repository maintenance, and account administration. Every task supplies an initial-state snapshot, task contract, allowable actions, expected state predicates, forbidden predicates, and reset mechanism. The benchmark excludes secrets, production accounts, and unbounded live systems.

The protocol compares at least four evaluators: final-text judgment only; exact raw-state equality; contract-scoped semantic diff; and semantic diff with evidence-freshness enforcement. Annotators blind to evaluator output label a stratified sample of runs as correct, incorrect, or unresolvable. Primary measurements are agreement with adjudicated labels, false acceptance of harmful effects, false rejection due to representational variation, inconclusive rate, and diagnostic localization time. Confidence intervals should be bootstrapped over tasks and runs, with domains reported separately rather than pooled into a single headline score [4].

The benchmark should pre-register an action taxonomy. Suggested classes are: read-only retrieval; reversible update; conditionally reversible update; irreversible deletion; permission change; and externally visible notification. Each task should declare whether human authorization is a precondition, whether the test harness provides a compensating action, and how reset is verified. This avoids comparing a harmless sandbox note with a privileged role assignment as though they had equal risk.

The study reports evaluator failures separately from agent failures. A semantic diff can be wrong because a normalizer is unsound, a snapshot is stale, an entity-matching rule is ambiguous, or the contract failed to name an important invariant. Error analysis records which component created the error and whether a reviewer can correct it from retained evidence. The same protocol supports a direct comparison of diagnostic value against conventional task-success scores.

The benchmark should inject controlled failures: correct entity with wrong field, correct field with wrong principal, duplicate action after timeout, stale read before write, unauthorized irreversible change, and a valid result with missing

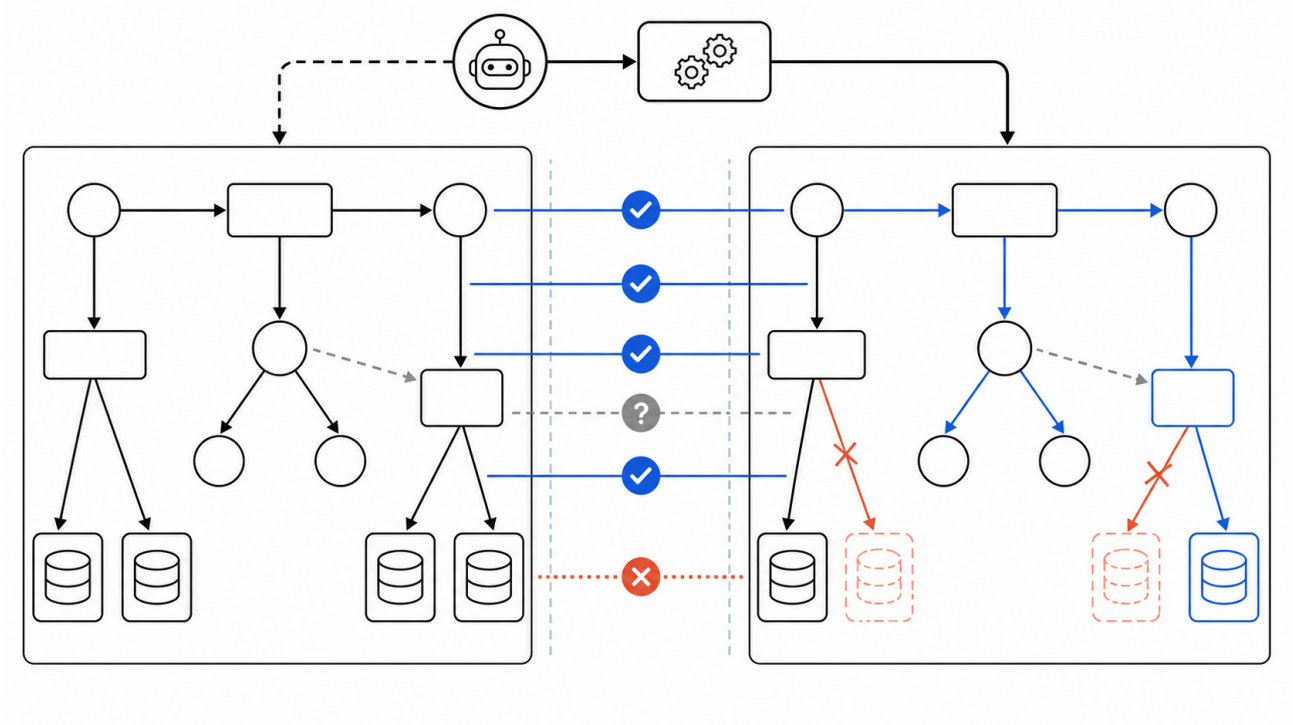

Fig. 1. Before/after typed state graphs with allowed, forbidden, and insufficient-evidence comparison paths.

TABLE I
SEMANTIC STATE-DIFF CONTRACT ARTIFACTS AND THEIR EVALUATION ROLES.

Artifact	Minimum recorded content	Evaluation purpose
Observation boundary Ω	Entity types, fields, relations, data sources, and snapshot timing	States exactly which portion of external state is being judged.
Required predicates Φ	Expected entity creation, field values, links, ordering constraints, and authorization requirements	Defines the changes necessary for a match.
Forbidden deltas Γ	Prohibited writes, deletions, privilege expansion, duplicate actions, and unrelated mutations	Detects harmful extra behavior even when required changes occurred.
Canonicalizer ν	Approved identifier matching, timestamp resolution, ordering rules, and version identifier	Makes permitted representation differences auditable and repeatable.
Side-effect labels Λ	Reversibility, compensation route, authorization level, and retry-safety class	Makes evaluation sensitive to risk, not just count of edits.
Evidence policy $\mathcal{A}$	Source identity, collection time, freshness window, retention and redaction policy	Preserves an inconclusive outcome when state cannot be established.

evidence. These cases test whether the evaluator distinguishes a safe state transition from a plausible transcript. Ablations remove canonicalization, side-effect labels, and evidence checks in turn.

VI. Contract Authoring and Adjudication

The hard part of a semantic diff is not graph subtraction; it is agreeing on what the task permits. Contract authors should begin with a short plain-language statement of the intended state transition, then translate it into typed entities, fields, relations, and postconditions. The translation should be reviewed by someone who owns the workflow, not only by the evaluator implementer. A contract for "close this support case" may require a status change, a closure timestamp, a resolution code, and a relation to an authorized actor, while forbidding deletion of the case history or changes to other cases. The resulting predicates are narrower and more auditable than a generic task-success label.

Every normalizer should be treated as policy. The contract must state which fields are ignored, which identifiers may be matched by an alternate key, which timestamp tolerance is acceptable, and whether ordering has semantic meaning. The evaluator should preserve both raw and canonical forms. A reviewer must be able to see that a value was hidden because it was a declared transient identifier rather than because it was inconvenient to compare. A change to a normalizer is a new evaluator version and requires rerunning affected tasks.

Adjudication should use a structured disagreement protocol. When a run is marked divergent, the dossier includes the relevant before and after fragments, matched entities, unmatched

entities, triggered predicates, evidence source identifiers, and the agent action trace. When it is marked inconclusive, the dossier additionally names the missing observation and the rule that made it necessary. Reviewers may then classify the case as an agent failure, evaluator failure, task-contract ambiguity, or environment failure. This taxonomy prevents performance claims from being inflated by silently reclassifying evaluator blind spots as agent errors.

The framework applies least privilege. Contracts for privileged actions require explicit evidence of authorization, a recorded actor identity, and a reversibility label that is more conservative than the tool's own success response. A test system uses disposable accounts and an independently verified reset. Where an action cannot be safely simulated, the contract forbids execution and evaluates the agent's ability to request approval or produce a dry-run plan instead. These controls align evaluation practice with the reliability being measured.

A public, sandboxed contract suite uses versioned state snapshots and reviewer labels. It tests whether independent teams write compatible contracts for the same task, whether the layer detects seeded side effects missed by text scoring, and how often its inconclusive state prevents a false pass. Versioned fixtures and adjudication records make those comparisons reproducible across evaluator revisions.

VII. Reference Evaluator Architecture

The reference evaluator has five separated stages: capture, resolution, canonicalization, policy evaluation, and evidence packaging. Capture reads before and after snapshots from the sources permitted by $\mathcal{A}$ and records collection metadata without deciding correctness. Resolution maps raw objects to contract entity types and produces candidate identity matches. Canonicalization applies only the versioned rules in ν. Policy evaluation computes required and forbidden predicates over the resulting graph. Evidence packaging emits a human-reviewable dossier. Separation matters because an identity-matching error should be diagnosed as such, rather than appearing as an unexplained model failure.

Entity resolution requires conservative defaults. A stable primary key is preferable when one exists. When a contract permits matching by a compound natural key, the rule should specify its fields, uniqueness assumptions, and tie handling. If two candidate entities match the same expected entity, or if a relation can be interpreted in more than one way, the evaluator should return INCONCLUSIVE unless the contract resolves the ambiguity. Guessing a match to improve an aggregate score is a silent change in risk policy.

Canonicalization operates after raw capture and before comparison, but it must never modify evidence in place. A timestamp bucket, Unicode normalization, or approved ordering rule can be represented as a deterministic transformation with an input and output digest. Field-level canonicalization is preferable to a broad serializer-level "cleanup" because it makes the affected semantics legible. Any rule that removes a field from comparison should cite the task reason: nondeterminism, privacy, or an explicitly declared representation boundary. The absence of a reason is itself useful review evidence.

Policy evaluation should first evaluate negative constraints. An observed destructive write or privilege expansion may require immediate divergence even if later predicates also pass. It should then evaluate required predicates, relationship constraints, and evidence freshness. The final verdict is accompanied by a minimal counterexample when possible: the smallest state fragment, predicate, and evidence source that demonstrate the decision. Minimality is a usability goal, not a formal guarantee. Large state graphs may require a summarized difference plus a reference to full retained evidence.

The architecture deliberately leaves agent planning and model grading outside the evaluator. An agent can be scored by another method for reasoning quality, explanation, or efficiency; semantic state diffs report only the declared transition. This separation also helps benchmark builders: they can reuse the same state contract across several agents and action interfaces, while publishing a change log whenever the domain schema or canonicalization policy moves. The reference implementation is sandbox-first, deterministic where possible, and accompanied by adversarial tests for stale reads, duplicate requests, and ambiguous identities.

VIII. Reproducibility and Audit Record

Every evaluated run should produce a compact audit bundle. The bundle contains the contract identifier and version, before and after snapshot digests, resolved entity mappings, canonicalizer version, predicates evaluated, evidence-source metadata, agent action identifiers, and final verdict. It should include enough bounded state fragments to reproduce the decision, with references to protected raw snapshots when direct retention is disallowed. A reviewer must be able to distinguish a missing entity from an entity that was captured but excluded by the contract.

Contract changes should be handled as evaluator changes. A benchmark maintainer who expands Ω, alters an identifier-matching rule, or changes a timestamp tolerance should issue a new contract version and report which prior tasks are affected. The release should include a migration note explaining whether the change modifies task semantics, only evidence collection, or only presentation. Historical scores should not be silently recomputed under a new normalizer and displayed as though they were directly comparable with the old scores.

The audit process should test the evaluator itself with fixtures designed to fail at each stage. Capture tests omit an authoritative snapshot; resolution tests introduce duplicate natural keys; canonicalization tests use an unapproved representation difference; policy tests add a forbidden write; and packaging tests remove an evidence reference. The expected result is not always divergence: an unavailable authoritative source should produce INCONCLUSIVE. These meta-tests are essential because an evaluation layer that cannot explain its own failure modes should not be used to certify long-horizon actions.

For privacy-sensitive domains, the bundle supports tiered disclosure. Public benchmark artifacts may contain synthetic entities and opaque digests, reviewers may receive redacted fragments under access control, and a restricted auditor may inspect the original evidence. The contract states which tier is sufficient for each verdict. This preserves the distinction between reproducibility, which requires a defensible procedure, and unrestricted data release, which may be unsafe or prohibited. Audit evaluation reports the cost and inter-reviewer agreement of this process so that retained evidence is assessed for practical interpretability.

IX. Relationship to Existing Evaluation

WebArena and OSWorld demonstrate that embodied task completion requires interaction with changing environments [1], [2]. SWE-bench grounds software tasks in tests, but a test suite may be incomplete or may not expose all external artifacts [3]. The state-diff layer complements task success metrics, unit tests, policy checks, and human review. It makes the target state transition a first-class object and captures the information needed to audit a disputed outcome.

This distinction also aligns with established ideas in database transactions and distributed systems. Idempotency, atomicity, and durable observation change the meaning of an action after a timeout; an HTTP success code is not enough evidence that the desired state is present [5]. The framework documents enough contractual context to separate observed state from unresolved distributed-consistency effects.

X. Limitations and Governance

The framework depends on a good contract. Important outcomes may be unobservable, canonicalization can encode policy mistakes, and state snapshots can expose sensitive information. Semantic matching can be expensive for large graphs, while causal attribution from a delta to a particular agent action may remain ambiguous. Moreover, a perfect evaluator cannot make an unsafe authorization policy safe. Systems using this method should minimize retained data, encrypt evidence, enforce sandboxing for evaluation, and require human authorization for actions whose reversibility label demands it.

XI. Conclusion

Semantic state diffs provide a precise complement to answer-centric agent evaluation. By specifying an observation boundary, expected and forbidden transitions, normalization policy, evidence requirements, and an inconclusive outcome, the framework makes long-horizon task completion reviewable at the level of external effects. This approach supports reproducible comparison, safer benchmark design, and clearer diagnosis when an agent, evaluator, or environment produces an uncertain outcome.

References

[1] S. Zhou et al., "WebArena: A realistic web environment for building autonomous agents," in *ICLR*, 2024.

[2] T. Xie et al., "OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments," in *NeurIPS*, 2024.

[3] C. E. Jimenez et al., "SWE-bench: Can language models resolve real-world GitHub issues?," in *ICLR*, 2024.

[4] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. New York, NY, USA: Chapman and Hall, 1993.

[5] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann, 1993.

[6] X. Liu et al., "AgentBench: Evaluating LLMs as agents," in *ICLR*, 2024.

[7] G. Mialon et al., "GAIA: A benchmark for general AI assistants," in *ICLR*, 2024.

[8] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in *ICLR*, 2023.

[9] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," in *NeurIPS*, 2023.

[10] H. Garcia-Molina and K. Salem, "Sagas," in *Proc. ACM SIGMOD*, 1987, pp. 249–259.

[11] B. W. Lampson, "Hints for computer system design," *ACM SIGOPS Operating Systems Review*, vol. 17, no. 5, pp. 33–48, 1983.

[12] National Institute of Standards and Technology, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, NIST AI 100-1, 2023.