

A transaction control layer for governed agentic commerce: reference architecture implementation and controlled evaluation

Nixalkumar Patel¹ and Krunal Panchal ²

¹ Independent Researcher, New York, United States

² Independent scholar, United States

Corresponding Author: Krunal Panchal, **E-mail:** Krunal.cp@live.com

ABSTRACT

A weak answer from a shopping assistant may inconvenience a customer. A weak action from a commerce agent can move money, reserve inventory, change a delivery commitment, or issue a refund. Once an agent can create those effects, the central question is no longer whether its response sounds correct. The question is whether the action is still authorized and commercially valid when the enterprise system commits it. This study introduces a Transaction Control Layer (TCL) that separates an agent's proposal from permission to execute. TCL binds each approved action to grounded intent, delegated authority, current authoritative state, a short validity period, and a stable idempotency key. It then records and reconciles the resulting side effects. We formalize commit validity across five dimensions and define five operational decisions: allow, modify, hold, deny, and escalate. The design is examined with CAV-Bench v1.0.0, an Apache-2.0 reference artifact containing 40 synthetic scenarios and five deterministic architecture profiles. The direct profile reaches an outcome success rate of 0.925, yet only 0.250 of its runs are commit-valid. Adding policy checks, commit-time guards, reconciliation, and lifecycle recovery raises commit-valid success to 0.500, 0.750, 0.875, and 1.000. These values show that the benchmark responds to the controls it was designed to test. They do not estimate real model failure rates or establish production effectiveness. An external replication protocol is therefore included as the next step toward independent validation.

KEYWORDS

AI agents, agentic commerce, transaction control, authorization, concurrency control, idempotency, reconciliation, service governance, benchmark

ARTICLE INFORMATION

ACCEPTED: 02 March 2026

PUBLISHED: 31 October 2026

DOI: 10.32996/jcsts.2026.8.12.4

1. Introduction

A weak answer from a shopping assistant may inconvenience a customer. A weak action from a commerce agent can change inventory, move money, alter a delivery commitment, or create a refund that another system must honor. That difference matters because an enterprise transaction is rarely contained in one model response or one API call. Price may come from a commerce engine, inventory from an ERP or warehouse system, payment status from a processor, and the delivery promise from an order or transportation platform. Each source can change while the agent is planning.

Emerging agent and commerce protocols improve how agents, merchants, and payment participants exchange requests and credentials [1], [2]. Current benchmarks also show that tool use, policy adherence, and adversarial resilience remain difficult in interactive environments [3]-[7]. Those advances are necessary, but they leave a narrower systems question unresolved: what must still be true at the instant when a proposed action becomes an external side effect?

Consider an agent that observes an in-stock item at an approved price, then continues reasoning while inventory, price, delivery capacity, or customer authority changes. A policy check can be correct when it runs and still be stale by the time the order commits. A different failure appears when a downstream service times out after accepting a request. If the agent treats silence as failure and retries without reconciliation, it can create a second order or charge. In either case, an end-state

checker may later find the requested outcome and report success, even though the path contained an unauthorized, stale, duplicate, or partially recovered effect.

We use commit-before-validation to describe this family of failures. The term does not imply that enterprise systems lack transaction controls. Optimistic concurrency control, conditional writes, idempotency, and compensation are well-established mechanisms [8], [9]. The problem is that agentic execution must combine those mechanisms with user intent and delegated authority, then apply them across services that do not share one transaction manager.

The work makes four contributions:

A transaction envelope that distinguishes an agent's proposed action from permission to commit it.

A five-dimensional definition of commit validity covering intent, authority, temporal state, execution integrity, and recovery.

A protocol-independent architecture built around version-bound authorization, idempotent execution, effect reconciliation, and evidence capture.

A reproducible controlled ablation using CAV-Bench v1.0.0, together with a protocol for independent replication.

The study is intentionally architectural. It tests whether the benchmark distinguishes incremental controls and whether outcome-only scoring can hide invalid commits. It does not estimate how often a language model will fail in production.

2. Background and Related Work

2.1. What Current Agent Evaluations Reveal

WebArena established that agents struggle with realistic, multi-step work across functional web applications [3]. AgentBench broadened the evaluation surface and connected failures to reasoning, decision making, and instruction following [4]. Tau-bench brought users, domain policies, APIs, and database-state checks into the same evaluation loop [5]. Together, these benchmarks show why final-answer accuracy is too narrow for an agent that acts.

The remaining gap is historical validity. A final database state can confirm that the goal was reached, but it may not reveal that an intermediate action exceeded authority, that a duplicate effect briefly existed, or that a compensating action later concealed an invalid commit. CAV-Bench therefore treats the committed side-effect history, rather than the agent's completion claim, as the evidence used for scoring.

Security-oriented work addresses a different but complementary risk. AgentDojo measures utility and resistance to prompt injection [6], while Agent-SafetyBench covers a broader set of unsafe agent behaviors [7]. TCL does not assume an attack. It treats every consequential proposal as potentially wrong because stale state, retries, mis-scoped delegation, and partial service outcomes can occur in ordinary operations.

2.2. What Commerce Protocols Do Not Decide

The Agentic Commerce Protocol defines how buyers, their agents, and businesses interact to complete purchases [1]. Visa's Trusted Agent Protocol gives merchants a cryptographic basis for recognizing a trusted agent and validating signed requests [2]. These mechanisms provide valuable identity and message-integrity signals. They do not decide whether the merchant's price, inventory, fulfillment capacity, or account state is still valid when the business system accepts the action. That distinction shapes the proposed architecture. TCL sits behind the interaction protocol and in front of the consequential service boundary. It can accept identity and delegation evidence from a protocol, but it resolves volatile commercial facts from the enterprise systems that own them.

2.3. Transaction Processing and the Remaining Composition Problem

Optimistic concurrency control verifies that relevant data has not changed before a write [8]. Sagas organize long-lived work into steps with compensating actions [9]. Idempotency and operation-status queries help services survive retries and ambiguous responses. TCL does not claim any of these mechanisms as new. Its contribution is to compose them with grounded intent and delegated authority at one agent-facing execution boundary, then measure the validity of the committed history without allowing the agent to grade itself.

This execution-centered interpretation also complements AI governance frameworks. The NIST AI Risk Management Framework emphasizes governance, measurement, and accountability [10], while its Generative AI Profile adds risks specific to generative systems [11]. TCL narrows those principles into decisions and evidence for consequential transactions.

3. Problem Model

3.1. Transaction Envelope

An agent produces a proposed consequential action q :

$$q = \langle p, a, r, \theta, l, C, V_0, E_0, \delta \rangle \quad (1)$$

Here p identifies the principal and delegation chain; a is the action; r is the resource; θ contains normalized parameters; l records grounded intent constraints; C contains contextual constraints; V_0 identifies the versions of authoritative state

observed during planning; E_0 contains supporting evidence; and δ is the decision deadline. The envelope makes the proposal explicit, but it does not authorize execution.

Given authoritative state S at time t and applicable policies P , TCL returns one of five decisions:

$D(q, S_t, P) \in \{\text{allow, modify, hold, deny, escalate}\}$ (2)

Allow produces a short-lived commit capability κ . The capability binds the approved action hash to the principal, resource scope, expected state versions, policy version, obligations, idempotency key, and expiry. The short lifetime is operationally important. Without it, an agent could retain a once-valid approval and present it after the commercial facts or delegation had changed. Modify creates a revised proposal that must be acknowledged or evaluated again. Hold delays execution while a resolvable condition is checked. Deny stops the action. Escalate transfers a bounded decision to an authorized reviewer with the uncertainty and observed effects attached.

3.2. Commit Validity

For a committed action c and execution trace τ , five predicates define whether the commit is valid:

$G(c)$ intent remains grounded in the user's expressed constraints (3)

$A(c)$ the principal and delegation authorize the action (4)

$T(c)$ business preconditions remain current at commit (5)

$X(c, \tau)$ external effects have the required cardinality and consistency (6)

$R(c, \tau)$ partial or ambiguous outcomes are resolved truthfully (7)

Recovery applies only when the scenario contains a partial or ambiguous outcome. Commit validity is conjunctive:

$CV(c, \tau) = G(c) \wedge A(c) \wedge T(c) \wedge X(c, \tau) \wedge R(c, \tau)$ (8)

The conjunctive rule reflects how consequential systems are reviewed in practice. A correct price cannot offset an unauthorized principal, and a later cancellation does not erase the fact that a duplicate charge occurred. Compensation may restore business state, but the historical commit remains invalid for measurement and audit.

3.3. Failure Classes and Required Controls

Table 1 Failure classes and transaction-control responses

Failure class	Commerce example	Why outcome scoring can miss it	Required TCL control
Intent drift	Agent upgrades shipping or substitutes an item outside the customer's conditions.	The order completes, but not on the customer's terms.	Canonical intent constraints, parameter comparison, and modify or escalate.
Authority defect	A delegated agent issues a refund above its scope or changes another tenant's resource.	The target state exists even though the actor lacked authority.	Principal and delegation verification, tenant scope, resource scope, amount limit, and policy version.
Stale state	Price, inventory, eligibility, or the fulfillment promise changes after planning.	An order can exist even though its commercial premise expired.	Version-aware reads, commit-time preconditions, conditional writes, and bounded capability lifetime.
Execution ambiguity	A service accepts a request but times out; an unguarded retry duplicates the effect.	The intended state exists alongside an extra charge or order.	Stable idempotency key, operation-status query, and append-only effect ledger.
Partial outcome	Payment succeeds but order creation fails, or one write conflicts.	A coarse success flag hides a stranded side effect.	Reconciliation, policy-approved compensation, bounded hold or escalation, and truthful reporting.

4. Transaction Control Layer Architecture

TCL places a deterministic execution boundary between agent planning and enterprise APIs. The agent may reason probabilistically and revise its plan. It cannot bypass the boundary when the action can change authoritative business state. This separation keeps the model useful for planning while preventing its own confidence or completion claim from becoming transaction authority.

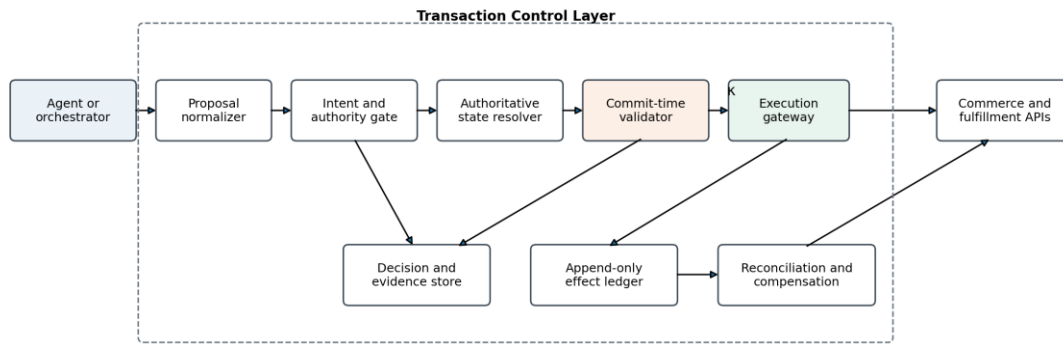


Figure 1 Reference architecture and trust boundary

4.1. Components and Design Rationale

The proposal normalizer gives downstream controls one stable representation of the requested action. This matters in an enterprise environment where the same business action may arrive through a conversational interface, an agent protocol, a workflow engine, or a conventional API. Without normalization, policy logic becomes coupled to each channel's payload. The intent and authority gate answers two different questions. Intent establishes what the customer actually requested, including limits and conditions. Authority establishes who may act, on which resource, in which tenant, and up to what scope. Combining the questions in one opaque allow flag would make later review difficult, so TCL records the predicates separately.

The authoritative state resolver reads volatile facts from the systems that own them. Conversational memory is not an authoritative source for inventory, payment status, account eligibility, or a delivery promise. The commit-time validator binds the latest versions to κ and checks them again at the gateway. Where the target service supports conditional writes, the expected version travels with the write. Where it does not, TCL narrows the remaining time-of-check to time-of-use interval and treats the residual uncertainty as something reconciliation must detect.

The execution gateway enforces the scope of κ and attaches a stable idempotency key. A timeout does not prove that the target service rejected the action, so retrying immediately is unsafe. The gateway first records what it observed and asks the target service for operation status when that capability exists. The effect ledger preserves the externally observed history. Reconciliation then decides whether the action committed, failed, duplicated, or left a partial state that requires compensation, hold, or escalation.

4.2. Illustrative Policy Schema

A useful policy must be enforceable by software and understandable during review. The schema below separates identity, intent, authority, state, execution, and recovery obligations. A production policy can add domain-specific rules without changing the commit protocol.

```

policy_id: refund-agent-v3
applies_to: {action: issue_refund, tenant: merchant-A}
principal:
require_signed_delegation: true
allowed_roles: [service_agent]
intent:
required_fields: [order_id, amount, reason]
amount_lte_user_authorized: true
authority:
max_amount: 500.00
resource_owner: current_customer
state:
authoritative_reads: [order.status, payment.version]
commit_preconditions:
- order.status in [PAID, PARTIALLY_FULFILLED]

```

- payment.version == observed.payment.version

execution:

idempotency_scope: [tenant, action, order_id, request_id]

max_effects: 1

recovery:

ambiguous: QUERY_STATUS_THEN_HOLD

partial: ESCALATE_WITH_EFFECT_SUMMARY

decision_ttl_ms: 3000

Listing 1 Illustrative transaction-control policy

4.3. Commit Protocol

The protocol keeps proposal review, commit authorization, and outcome verification distinct. That separation makes it possible to identify whether a failure began in interpretation, permission, stale state, execution, or recovery.

Normalize the proposed call into q and reject any missing critical field.

Resolve the principal, agent identity, delegation chain, tenant, and resource scope.

Evaluate intent and authority. Return modify, hold, deny, or escalate when execution would be unsafe.

Read authoritative preconditions and versions into the commit read set V_c .

Evaluate the policy with V_c and issue signed, short-lived κ bound to $H(q)$, V_c , the policy version, obligations, and an idempotency key.

At the gateway, verify the signature, expiry, action hash, policy version, and current state versions.

Execute only the action and parameters authorized by κ .

Record the request and observed response in the effect ledger. Do not infer failure from a timeout alone.

Query operation status when the result is ambiguous, then reconcile observed effects against the expected cardinality.

Compensate only when policy permits it. Otherwise hold or escalate and report the actual partial state.

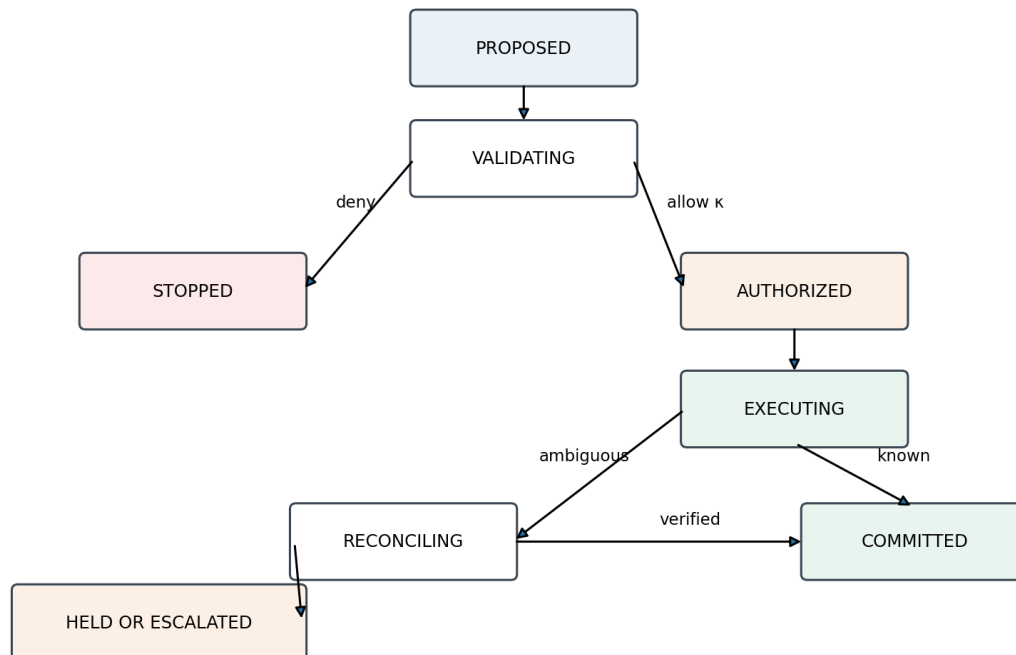


Figure 2 Controlled transaction lifecycle

5. Reference Implementation

The implementation target is middleware that can sit beside existing commerce, payment, order, and fulfillment services. It is not a replacement for those systems or their transaction controls. CAV-Bench v1.0.0 provides the benchmark-facing implementation: a stable adapter protocol, deterministic tool environments, authoritative state and version history, an

append-only side-effect ledger, recovery events, a private oracle, and an evaluator that derives validity without trusting adapter-supplied labels [12].

5.1. Evaluator Trust Boundary

An adapter may submit a proposed action, trace metadata, and a claimed completion status. It cannot declare itself commit-valid. The evaluator derives each result from the private scenario oracle, authoritative version history, canonical environment events, the side-effect ledger, and environment-recorded recovery or escalation. A contract test forges adapter fields such as `commit_valid` and confirms that the evaluator's result does not move [12].

This boundary is central to the research design. If the same component that acts can also define whether its action was valid, the metric becomes another self-report. Independent grading does not eliminate benchmark bias, but it prevents that direct circularity.

5.2. Decision Record

Each attempted transaction produces a decision record z :

$$z = \langle \text{id}, H(q), p, P_v, V_o, V_c, D, H(k), L_o, s^f \rangle \quad (9)$$

P_v identifies the policy version, L_o points to the observed-effect ledger, and s^f records the evaluator-derived terminal state. Hashes can reduce duplication of large or sensitive payloads, but production deployments still need explicit access, retention, and privacy controls.

5.3. Integration Modes

The reference artifact exposes a Python adapter interface and an experimental loopback REST gateway. A conventional service, agent framework, or future MCP or A2A client can map a proposed tool call to the adapter protocol without changing evaluator internals. New domains enter through a versioned scenario schema, and CI can stop a build when `commit-valid` success falls below a configured threshold [12]. The REST gateway remains experimental and has not received external technical review. It is not presented as a production security boundary.

6. Controlled Evaluation

6.1. Research Questions

RQ1. Can end-state success materially overstate the validity of consequential commits?

RQ2. Does the benchmark respond when the architecture adds the controls specified by TCL?

RQ3. Which residual failures require commit-time and post-execution controls rather than pre-execution policy alone?

6.2. Scenario Pack and Architecture Profiles

The `core-v1` pack contains 40 public, synthetic scenarios across four failure families. Five deterministic, non-LLM profiles run through the same capability-parameterized engine. Direct execution has no intent gate, authority gate, commit guard, reconciliation, or recovery. Policy-gated execution adds intent and authority checks. Commit-guarded execution also validates versioned state at commit. The reconciled profile adds stable idempotency keys and operation-status queries. Full lifecycle execution adds compensation, bounded escalation, and truthful reporting of partial state [12].

Because the environment and profiles are deterministic, the reported proportions describe this constructed corpus exactly. Confidence intervals would imply a population inference that the experiment does not support.

6.3. Metrics

Outcome success rate, or OSR, asks whether the final state satisfies the goal. Policy-aware outcome success rate, or PAOSR, also requires the absence of intent and authority violations. Commit-valid success rate, or CVSR, requires every applicable validity dimension to pass. The two gaps are:

$$6.4. \text{VG} = \text{OSR} - \text{CVSR} \quad \text{PAVG} = \text{PAOSR} - \text{CVSR} \quad (10)$$

CVSR is non-compensatory and comes from evaluator-side evidence. A later correction can restore the desired business state, but it does not turn an earlier unauthorized or duplicate effect into a valid commit.

6.5. Results

Table 2 Controlled architecture ablation across 40 scenarios

Profile	OSR	PAOSR	CVSR	VG
Direct	0.925	0.750	0.250	0.675
Policy gated	1.000	1.000	0.500	0.500
Commit guarded	1.000	1.000	0.750	0.250
Reconciled	1.000	1.000	0.875	0.125
Full lifecycle	1.000	1.000	1.000	0.000



Figure 3 Outcome success remains high while commit-valid success exposes missing controls

The direct profile reaches OSR of 0.925 but CVSR of 0.250, leaving a validity gap of 0.675. Even PAOSR of 0.750 exceeds CVSR by 0.500. In this corpus, reaching the requested end state and avoiding an intent or authority violation still does not establish that the transaction used current state, created the correct number of effects, or recovered truthfully.

CVSR rises as each control family is enabled: 0.250 for direct execution, 0.500 with policy gates, 0.750 with the commit guard, 0.875 with reconciliation, and 1.000 with full lifecycle handling. The progression shows that the evaluator distinguishes the architectural changes it was built to test. The final value does not show that TCL prevents every real-world failure. The scenarios and deterministic profiles were constructed around known control paths.

The residual gaps also explain why one general policy check is insufficient. Intent and authority checks close part of the gap, but they cannot detect a price or inventory change that occurs after the check. Commit-time validation addresses that state transition. It still cannot determine whether a timed-out service accepted the request, which requires idempotency and status reconciliation. Partial outcomes require a final layer of recovery and truthful reporting.

6.6. Reproducibility

The immutable v1.0.0 archive and canonical results are identified by DOI 10.5281/zenodo.21364386 [12]. Core execution requires Python 3.11 through 3.13 and does not require network access. A clean replication installs the development dependencies, runs the test suite, and executes cavbench ablate. The run directory preserves the manifest, per-scenario traces, evaluation records, and machine-readable summaries. Replicators should retain the dependency versions and run metadata with their results.

7. Independent Evaluation Protocol

Controlled reproducibility and independent validation answer different questions. The current experiment shows that the benchmark responds to known architectural controls in a deterministic environment. An outside evaluator must show whether another team can reproduce the result, extend the artifact without weakening its trust boundary, and apply the method to a service boundary the author does not control.

A credible replication begins with the immutable artifact rather than an author-generated results summary. The evaluator should record its identity, organization, environment, source hash, dependency-lock hash, and run dates. It should preserve the complete run directory and confirm that forged adapter validity fields cannot change the score. These steps make the chain of evidence inspectable.

Reproducing the canonical table alone is useful but limited. The evaluator should also implement at least one adapter or scenario pack without changing evaluator internals. A stronger study would connect a non-production commerce service and inject stale inventory or price, narrowed delegation, timeout after success, duplicate retry, partial multi-service completion, and inconsistent success reporting. Baseline and TCL-enabled configurations must use the same scenarios and disclose every exclusion.

The study should report validity metrics together with operational costs: median and tail decision latency, authoritative reads per attempted commit, modification and denial rates, hold and escalation rates, reconciliation time, duplicate or partial effects, and false denials on valid actions. The evaluator should document defects, changes, and before-and-after results. A signed factual statement can then identify what was tested and what changed. Until that evidence exists, the appropriate claim remains a reference architecture with controlled evaluation.

8. Discussion

8.1. Deployment Choices

TCL can run as a shared gateway, an embedded library, or a sidecar. A shared gateway gives the organization one policy and evidence boundary, but it adds a network dependency and can concentrate operational risk. An embedded library removes that hop but makes consistent policy versions and evidence formats harder to maintain across teams. A sidecar can follow service ownership while receiving centrally managed policy. The correct choice depends on latency, failure isolation, and who owns the authoritative commit.

Authoritative reads will usually dominate decision latency. Implementations can cache stable identity attributes, batch independent reads, and resolve unrelated preconditions in parallel. Volatile commercial facts should remain version-bound. When a business accepts more risk for a low-value action, the policy should express that choice explicitly instead of allowing the agent to bypass the layer.

Human review is one possible decision outcome, not a substitute for transaction controls. An escalation should carry the exact proposal, the failed or uncertain predicate, the authoritative versions, the effects already observed, and the decision deadline. Otherwise, the reviewer receives the same ambiguity that caused the automated path to stop.

8.2. Limitations and Threats to Validity

The evaluation contains 40 public synthetic scenarios and five deterministic non-LLM profiles. A model can memorize public cases, the corpus does not cover every consequential-action failure, and deterministic profiles do not represent the variance of frontier models. The environment assumes cooperating adapters and is not a sandbox for malicious code. The experiment does not measure throughput, tail latency, operator burden, user acceptance, false positives, or economic impact.

Construct validity depends on the five-dimensional definition. Some deployments may classify compensated effects differently. We retain a non-compensatory historical measure because an unauthorized or duplicate side effect remains relevant after correction. The conclusions are descriptive rather than statistical. External validity requires evaluator-owned integrations and scenarios.

TCL also cannot create atomicity when a remote API provides neither conditional writes nor a queryable idempotent operation. In that setting, it can narrow the risk window, expose uncertainty, and trigger recovery, but it cannot prove that the remote service produced exactly one effect. Privacy and regulatory requirements for evidence retention remain deployment-specific and fall outside this study.

9. Conclusion

A commerce agent needs a clear boundary between proposing an action and receiving authority to commit it. TCL places grounded intent, delegated authority, current enterprise state, idempotent execution, reconciliation, and recovery at that boundary. The controlled ablation shows why outcome success alone can be misleading: a system may reach the requested end state while the committed history contains a stale, duplicate, unauthorized, or poorly recovered effect. Successive controls close the measured gap in the constructed corpus. Independent replication with evaluator-owned adapters and a real service boundary is the necessary next test.

Funding: Funding information was not supplied in the manuscript.

Conflicts of Interest: A conflict-of-interest declaration was not supplied in the manuscript.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers.

References

- Agentic Commerce Protocol Project, "Agentic Commerce Protocol," 2025. Available: <https://github.com/agentic-commerce-protocol/agentic-commerce-protocol>. Accessed: Sep. 14, 2026.
- C. Autio et al., Artificial Intelligence Risk Management Framework Generative Artificial Intelligence Profile, NIST AI 600-1, 2024. <https://doi.org/10.6028/NIST.AI.600-1>
- E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer, and F. Tramer, "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," Advances in Neural Information Processing Systems, 2024.
- H. Garcia-Molina and K. Salem, "Sagas," Proc. ACM SIGMOD International Conference on Management of Data, pp. 249-259, 1987. <https://doi.org/10.1145/38713.38742>
- H. T. Kung and J. T. Robinson, "On Optimistic Methods for Concurrency Control," ACM Transactions on Database Systems, vol. 6, no. 2, pp. 213-226, 1981. <https://doi.org/10.1145/319566.319567>

- N. Patel, CAV-Bench Commit-Time Action Validity Benchmark, version 1.0.0, Zenodo, 2026. <https://doi.org/10.5281/zenodo.21364386> Source: <https://github.com/Harimay23/cav-bench>.
- National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework, NIST AI 100-1, 2023. <https://doi.org/10.6028/NIST.AI.100-1>
- S. Yao, N. Shinn, P. Razavi, and K. Narasimhan, "Tau-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains," arXiv:2406.12045, 2024.
- S. Zhou et al., "WebArena: A Realistic Web Environment for Building Autonomous Agents," Proc. International Conference on Learning Representations, 2024.
- Visa, "Trusted Agent Protocol: Getting Started," 2025. Available: <https://developer.visa.com/capabilities/trusted-agent-protocol/docs-getting-started>. Accessed: Sep. 14, 2026.
- X. Liu et al., "AgentBench: Evaluating LLMs as Agents," Proc. International Conference on Learning Representations, 2024.
- Z. Zhang et al., "Agent-SafetyBench: Evaluating the Safety of LLM Agents," arXiv:2412.14470, 2024.