PROOF-CARRYING AGENT CONTRACTS (PACT): GOVERNED RELEASE FOR STATEFUL AGENT EXECUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using agents are usually judged by task completion, even though the side effects that produce a successful result must also satisfy authority, budget, interface, and recovery constraints at execution time. Proof-Carrying Agent Contracts (PACT) treats this commit boundary as a separate control problem. A typed contract binds an action graph to scoped authority, budget rules, admissible interfaces, declared recovery paths, proof obligations, and accountability evidence. At commit time, a trusted gate rechecks live-dependent obligations and records the decision in an attested release record. Under explicit assumptions on policy coverage, local-verifier soundness, and trusted commit binding, released actions are policy-admissible; structural refinement can only shrink the release set; and dependency-complete cache reuse is equivalent to recomputing reused checks. VeraContract-Bench adds deployment constraints to 180 autonomous software-engineering task instances and evaluates matched runs at both action and task levels. With three matched seeds for stochastic methods, PACT attains 63.5% task success with 1.8% unsafe-release rate, compared with 64.1% and 19.3% for the matched ReAct backbone. Governed success reaches 49.3%, versus 37.2% for AgentSpec-style runtime rules; recovery reaches 79.4%, and release-gate P99 latency is 64 ms. These results separate successful task completion from deployable execution at the commit boundary.

1 INTRODUCTION

Language-model agents increasingly change persistent state: they edit repositories, invoke APIs, spend budgets, and resume work under policies that may have changed since an earlier step. A task can therefore succeed while its execution remains unacceptable. A patch may pass every test after an unapproved dependency was installed; a resumed run may reuse a capability issued under an obsolete policy; a correct artifact may result from an undeclared data transfer. ReAct, Reflexion, LLMCompiler, and SWE-agent illustrate the move from text generation to state-changing tool use (Yao et al., 2023; Shinn et al., 2023; Kim et al., 2024; Yang et al., 2024). SWE-bench, WebArena, and OSWorld measure task completion in these settings, while AgentDojo, RedCode, AgentHarm, and τ-bench expose security and policy failures (Jimenez et al., 2024; Zhou et al., 2024; Xie et al., 2024; Debenedetti et al., 2024; Guo et al., 2024; Andriushchenko et al., 2025; Yao et al., 2025). These results motivate a deployment question that task accuracy alone does not answer: under the current authority and system state, may a proposed side effect actually be committed?

Existing systems address important parts of this question. Guard agents reason over explicit safety policies, privilege systems constrain tool use, runtime specification languages enforce rules, and newer runtimes preserve security context, stage external effects, or recover from earlier state changes (Xiang et al., 2025; Chen et al., 2025; Shi et al., 2025; Wang et al., 2026a; Palumbo et al., 2026; Zheng & Yang, 2026; Chen et al., 2026; Yang et al., 2026; Zhuang et al., 2026). PACT focuses on the commit boundary and separates action proposal from release authority. An interchangeable agent policy proposes an action. A compiler derives a typed contract from the task and policy artifacts. A trusted runtime checks the applicable obligations against live state and binds the canonical action, contract, state digest, and current policy/registry epoch before the effect is released. The resulting release record retains the evidence needed to reconstruct the decision. The formal guarantee is deliberately scoped to declared release legality under policy-coverage, verifier-soundness, and trusted-binding assumptions; it does not establish semantic correctness of arbitrary agent-generated code.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

VeraContract-Bench evaluates task utility and deployability separately on SWE-bench-derived workflows. With the matched ReAct backbone, PACT changes task success from 64.1% to 63.5% while reducing unsafe-release rate (UAR) from 19.3% to 1.8%. Governed success rate (GSR) reaches 49.3%, compared with 37.2% for the AgentSpec-style baseline, the strongest non-PACT governance baseline in this evaluation. The same study measures recovery after partial failure, compiler reliability, operator diagnosis, and release-gate tail latency, so lower unsafe-release rates cannot be interpreted independently of utility, recovery, or deployment cost.

Contributions. PACT formalizes governed release through a typed action-graph contract, live check-to-commit binding, declared recovery-path coverage, and an accountability-bearing release record. The analysis establishes release soundness, a conditional residual-risk bound aligned with UAR, monotone contract refinement, and sound incremental verification when the declared proof dependencies remain unchanged. VeraContract-Bench evaluates action-level release quality and run-level governed success under a matched protocol, with ablations that isolate live revalidation, recovery semantics, accountability, and proof reuse.

2 Related Work

Runtime policy enforcement and action-boundary governance. GuardAgent and ShieldAgent translate explicit safety requirements into executable checks over agent actions (Xiang et al., 2025; Chen et al., 2025). Progent, AgentSpec, and FORGE enforce privileges or runtime policies at tool boundaries (Shi et al., 2025; Wang et al., 2026a; Palumbo et al., 2026), while SAGA focuses on identity, access control, and communication across agentic systems (Syros et al., 2026). Agent Behavioral Contracts, Proof-Carrying Agent Actions, and Aegis make behavioral contracts, action evidence, or trusted action-boundary control explicit (Bhardwaj, 2026; Wang, 2026; Mazzocchetti, 2026). CONTINUITY preserves authenticated security context as actions cross composed controls, and the Irreversibility Budget accounts for residual irreversible risk across agents and workflows (Zheng & Yang, 2026; Mohammadi & Bindschaedler, 2026). PACT operates in this same runtime-governance space. Its unit of analysis is a contract-covered side effect within a typed action graph: at commit time, the gate checks the applicable obligations against current state and policy, binds the decision to declared recovery and accountability evidence, and records the result for later reconstruction. VeraContract-Bench then evaluates that decision at both the action level and the run level, separating release quality from task completion and recovery.

Transactions, recovery, and formal enforcement. Sagas and transaction-processing systems established compensation and recovery for long-running workflows (Garcia-Molina & Salem, 1987; Gray & Reuter, 1992). Cordon stages effects behind a semantic transaction boundary (Chen et al., 2026); DART checks whether a restore point remains semantically admissible after downstream commitments (Yang et al., 2026); AgentRewind aligns agent context with environment checkpoints (Zhuang et al., 2026); and Recoverability as a System Primitive makes continuation from saved state an explicit, evidence-backed decision (Zhang & Liu, 2026). SafeMCP uses environment-grounded look-ahead reasoning to constrain tool acquisition, while SIMMER measures latent failures that can arise before an immediate execution error (Wang et al., 2026b; Lu et al., 2026). PACT has a narrower recovery scope: it requires a declared recovery or terminal path and checks the represented preconditions when an action is released. Recovery-controller synthesis and preservation of recovery prerequisites across arbitrary later actions remain outside the model. The formalization also draws on proof-carrying code, runtime verification, and capability systems (Necula, 1997; Necula & Lee, 1998; Leucker & Schallhart, 2009; Dennis & Van Horn, 1966; Saltzer & Schroeder, 1975). Accountability is used in the technical sense of reconstructing the responsible principal, failed obligation, supporting evidence, and adjudication route, rather than assigning legal liability (Raji et al., 2020).

3 PACT: Governed Release Semantics

Execution model and contract. An execution instance is

$$x = (g, s, \mathcal{P}, \mathcal{T}, \mathcal{K}, \mathcal{H}), \tag{1}$$

where g is the task goal, s is the current environment state, $\mathcal{P}$ contains the concrete policy artifacts, $\mathcal{T}$ is the available tool/interface set, $\mathcal{K}$ is the verified-skill registry, and $\mathcal{H}$ contains the supported proposal-policy families. Proposal policies may be stochastic and remain outside the trusted computing base.

Throughout the paper, $\mathcal{P}$ denotes policy artifacts, π their admissibility semantics, and Π the proof obligations carried by a contract.

A proof-carrying agent contract is

$$\mathcal{C} = (G, \Phi, \Sigma, \mathcal{E}, \Delta, \Pi, \Omega, \mathcal{J}_C, \mathcal{R}_C, \mathcal{S}_C), \tag{2}$$

where G is the action graph, Φ contains declared invariants, Σ is the capability environment, $\mathcal{E}$ is the economic envelope, Δ specifies delegation and approval, Π contains proof obligations, Ω records accountability mappings, $\mathcal{J}_C$ specifies admissible interfaces, $\mathcal{R}_C$ contains declared recovery semantics, and $\mathcal{S}_C$ specifies skill and execution constraints. The action graph is

$$G = (V, E^{\mathrm{fwd}}, E^{\mathrm{comp}}), \tag{3}$$

where E^{fwd} records forward dependencies and E^{comp} records declared compensation dependencies. Let $V_{\mathrm{commit}} \subseteq V$ denote actions that may create externally visible state, and let $V_{\mathrm{recover}} \subseteq V_{\mathrm{commit}}$ denote those for which the contract requires a recovery or terminal path.

For a committing action a, an applicable obligation j, and trace prefix τ, the accountability map returns

$$\Omega(a, j, \tau) = (p_{a,j}, e_{a,j}, r_{a,j}, q_{a,j}, c_{a,j}), \tag{4}$$

where $p_{a,j}$ identifies the responsible principal, $e_{a,j}$ the required evidence route, $r_{a,j}$ the adjudication route, $q_{a,j}$ the retention policy, and $c_{a,j}$ the escalation condition. These fields make a release decision reconstructable; they do not assign legal liability.

Commit-time release. For a proposed side effect a_t, let $\mathsf{Obl}(a_t, \mathcal{C})$ denote the obligations instantiated by the contract. Logic closure is checked before execution: every committing action must instantiate the mandatory obligations required by its action class. Missing mandatory rules therefore invalidate the contract rather than passing vacuously.

For each $j \in \mathsf{Obl}(a_t, \mathcal{C})$, the corresponding verifier returns $v_j(a_t, s_t, \mathcal{C}) \in \{0, 1\}$. The release predicate is

$$\mathrm{Rel}(a_t, s_t, \mathcal{C}) = \bigwedge_{j \in \mathsf{Obl}(a_t, \mathcal{C})} v_j(a_t, s_t, \mathcal{C}). \tag{5}$$

The implementation organizes these obligations into eight families: Auth, Econ, Iface, Skill, $\mathrm{Pattern}$, Attest, $\mathrm{Approve}$, and $\mathrm{Accountable}$. The family names are implementation groups rather than additional predicates beyond Eq. (5). Auth checks the principal, delegated scope, expiry, nonce, and policy epoch; Econ and Iface check projected cost and the requested target, operation, and arguments. Skill and $\mathrm{Pattern}$ constrain registry state and execution structure, including fan-out and evaluator calls. The remaining families require the appropriate attestation, approval, and accountability evidence. Appendix A gives the typed fields used by these checks.

The gate resolves failed obligations with the following precedence:

$$\mathsf{Gate}(a_t, \mathcal{C}, s_t) = \begin{cases} \textsc{release}, & \mathrm{Rel}(a_t, s_t, \mathcal{C}) = 1, \\ \textsc{block}, & \text{a fatal obligation fails}, \\ \textsc{repair}, & \text{no fatal failure and bounded repair exists}, \\ \textsc{downgrade}, & \text{repair is exhausted and an admissible } \mathcal{C}_{\mathrm{safe}} \text{ exists}, \\ \textsc{block}, & \text{otherwise}. \end{cases} \tag{6}$$

The cases are mutually exclusive under this ordering: a fatal violation dominates repair and downgrade. Every action in V_{commit} crosses this boundary. Evidence that is independent of mutable state may be prepared earlier, but state-sensitive predicates are reevaluated against s_t before commit.

Declared recovery and release evidence. PACT requires a declared recovery or terminal disposition for each action in V_{recover}:

$$\forall a \in V_{\mathrm{recover}} : \quad \mathrm{Compensable}(a) \vee \mathrm{ApprovedIrreversible}(a) \vee \mathrm{DeclaredTerminal}(a). \tag{7}$$

Compensation follows saga semantics when exact rollback is unavailable (Garcia-Molina & Salem, 1987). An irreversible action instead requires elevated approval or an explicit terminal classification.

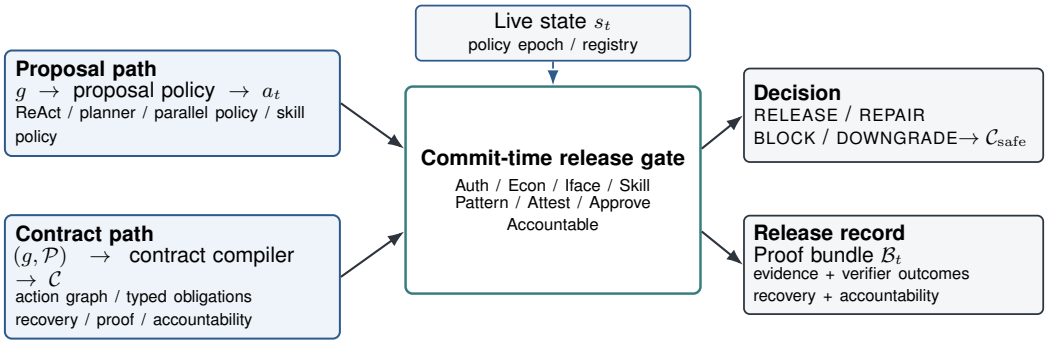

Figure 1: PACT separates proposal from release authority. A proposal policy supplies a_t, the compiler supplies $\mathcal{C}$, and the commit-time gate evaluates both against live state. The decision is bound to the checked action, state, and policy version and recorded in $\mathcal{B}_t$.

Eq. (7) establishes declared recovery-path coverage. It does not guarantee that a backup, credential, budget, or external service required by that path will remain available after arbitrary later actions.

Each gate decision produces an attested release record,

$$\mathcal{B}_t = (d_a, d_C, d_s, e_p, e_\Sigma, e_\mathcal{E}, e_J, e_K, e_\Pi, e_R, e_\Omega, v_t, \delta_t, \sigma_t), \tag{8}$$

where d_a, d_C, and d_s bind the canonical action, contract, and verified live state. The evidence terms bind the relevant policy, capability, budget, interface, skill, obligation, recovery, and accountability material; v_t contains the verifier outcomes, δ_t the gate decision, and σ_t the integrity signature. Here "proof" means machine-checkable evidence for the declared obligations, rather than a proof of arbitrary program correctness.

The checked object must also be the object that crosses the execution boundary. Let ρ_t denote the current policy/registry version vector and define

$$\beta_t = (d_a, d_C, d_s, \rho_t), \qquad \sigma_t = \mathrm{Sign}(\beta_t, v_t, \delta_t). \tag{9}$$

The runtime treats β_t as immutable for the corresponding commit. A change to the canonical action, any represented state dependency, or the relevant policy/registry version invalidates the affected verdicts. Compilation may therefore prepare stable evidence in advance without granting authority to commit a different action or a stale state. Figure 1 summarizes the separation between proposal and release authority.

4 Compilation, Verification, and Proof Reuse

Contract compilation. The language model proposes values inside a typed contract skeleton; deterministic validation decides whether the resulting contract can govern execution. A task-family skeleton fixes the required fields, schema-constrained decoding fills action nodes, capabilities, budgets, approvals, recovery edges, proof obligations, and accountability routes, and verification returns typed failures. A bounded repair loop revises only the fields implicated by those failures:

$$F_i = \mathsf{Verify}(\widehat{\mathcal{C}}_i), \qquad \widehat{\mathcal{C}}_{i+1} = \mathsf{Repair}(\widehat{\mathcal{C}}_i, F_i), \qquad i < k. \tag{10}$$

If no logic-closed contract is obtained within the repair budget, execution falls back to $\mathcal{C}_{\text{safe}}$. The safe skeleton permits observation, local analysis, and declared dry-run operations while excluding committing effects.

Compiler quality is measured independently of downstream task success. Let $\mathcal{D}$ denote the compilation instances and S_x the required schema for instance x. The schema-valid rate by attempt k is

$$\mathrm{SVR@}\,k = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \mathbf{1}\Big\{\exists i \le k : \widehat{\mathcal{C}}_i(x) \in \mathsf{Lang}(S_x)\Big\}. \tag{11}$$

Logic-closure rate $\mathrm{LCR@}\,k$ additionally requires the deterministic contract checks to pass. The bounded-safe-downgrade rate is

$$\mathrm{BSSD} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \mathbf{1}\{\mathcal{C}_k(x) = \mathcal{C}_{\text{safe}}\}. \tag{12}$$

In the reported compiler protocol, infrastructure failures are classified outside this outcome denominator; every remaining instance terminates either with a logic-closed contract or with $\mathcal{C}_{\text{safe}}$. Under that protocol, $\text{LCR@}\,k + \text{BSSD} = 1$.

Runtime verification and proof reuse. Verification starts with structural typing and registry validity, then evaluates the critical action subgraph and its temporal obligations. Nodes that may create external state receive the strongest checks. Immediately before commit, the runtime rereads mutable authority, budget, state preconditions, and evidence freshness; compilation-time values cannot authorize a later commit by themselves.

Stable verifier results can be reused only when their semantic dependencies are unchanged. For obligation j at time t, define

$$\chi_{j,t} = (C_j, \rho_t, \kappa_t, \nu_t), \qquad d_{j,t} = \text{Hash}(\chi_{j,t}), \tag{13}$$

where C_j contains the contract fragment and canonical action projection consumed by verifier j, ρ_t is the policy/registry version vector, κ_t is the canonical snapshot of the verifier's complete declared state dependencies, and ν_t is the verifier implementation version. The snapshot binds dependency identities and current values, including negative predicates, absence conditions, and validity deadlines when those facts can change the verdict. A signed cache entry binds the canonical tuple as well as its digest. Reuse requires tuple equality, verifier-version equality, and signature validity; the digest serves only as an efficient lookup key. Any obligation whose tuple changes is recomputed before release.

5 Formal Guarantees

The formal results concern the same commit event defined by Eq. (6). They establish three properties of that boundary: a released action satisfies the declared policy abstraction under the stated assumptions; tightening a contract cannot enlarge its release set; and reusing an unchanged verifier result is equivalent to recomputing it when the complete dependency tuple is preserved. No result below asserts semantic correctness of arbitrary generated code or safety outside the declared policy scope.

Let $\mathcal{A}$ be the universe of contract-representable side effects, let $\mathcal{A}_\pi(s) \subseteq \mathcal{A}$ be the effects admissible under organizational policy semantics π in state s, and let $Q_j(s) \subseteq \mathcal{A}$ denote the admissible set associated with obligation j.

Assumption 1 (Trusted enforcement)**.** The policy store, registry, release gate, evidence store, signer, and verifier implementations execute according to their specified semantics. Evidence and capability bindings cannot be forged inside the trusted boundary, and the check-to-commit mechanism atomically binds the canonical action, verified state digest, and policy/registry version represented by Eq. (9).

Assumption 2 (Local verifier soundness)**.** For every applicable obligation j of a contract-covered action,

$$v_j(a, s, \mathcal{C}) = 1 \quad \Longrightarrow \quad a \in Q_j(s). \tag{14}$$

Assumption 3 (Policy coverage)**.** For every logic-closed committing action in the declared policy scope,

$$\bigcap_{j \in \mathsf{Obl}(a,\mathcal{C})} Q_j(s) \subseteq \mathcal{A}_\pi(s). \tag{15}$$

Theorem 1 (Compositional release soundness)**.** *Under Assumptions 1–3, every released contract-covered side effect is policy-admissible at the state against which it is committed:*

$$\mathsf{Gate}(a, \mathcal{C}, s) = \textsc{release} \quad \Longrightarrow \quad a \in \mathcal{A}_\pi(s). \tag{16}$$

Proof sketch. A release decision requires every obligation in $\mathsf{Obl}(a, \mathcal{C})$ to pass. Assumption 2 therefore places a in each corresponding $Q_j(s)$, and Assumption 3 places their intersection inside $\mathcal{A}_\pi(s)$. Assumption 1 ensures that this conclusion applies to the action and state actually committed. Appendix B gives the full proof. □

Applied to each commit of a finite trace, Theorem 1 gives per-step admissibility under the state and policy represented at that commit. It does not imply that a recovery path declared at one step remains feasible after arbitrary subsequent actions.

Corollary 1 (Residual-risk decomposition)**.** *Let E_{cov}, E_{ver}, and E_{bind} denote failures of policy coverage, local verifier soundness, and trusted binding, with respective probability bounds ϵ_{cov}, ϵ_{ver}, and ϵ_{bind}. Then*

$$\Pr[a \notin \mathcal{A}_\pi(s) \wedge \mathsf{Gate}(a, \mathcal{C}, s) = \textsc{release}] \leq \epsilon_{\mathrm{cov}} + \epsilon_{\mathrm{ver}} + \epsilon_{\mathrm{bind}}. \tag{17}$$

If

$$p_{\mathrm{rel}} = \Pr[\mathsf{Gate}(a, \mathcal{C}, s) = \textsc{release}] > 0, \tag{18}$$

then

$$\Pr[a \notin \mathcal{A}_\pi(s) \mid \mathsf{Gate}(a, \mathcal{C}, s) = \textsc{release}] \leq \min\left\{1, \frac{\epsilon_{\mathrm{cov}} + \epsilon_{\mathrm{ver}} + \epsilon_{\mathrm{bind}}}{p_{\mathrm{rel}}}\right\}. \tag{19}$$

The left-hand side of Eq. (19) has the same denominator semantics as empirical UAR.

Proof sketch. When none of the three failure events occurs, Theorem 1 excludes unsafe release. The joint unsafe-release event is therefore contained in $E_{\mathrm{cov}} \cup E_{\mathrm{ver}} \cup E_{\mathrm{bind}}$, and the union bound gives Eq. (17). Conditioning on release yields Eq. (19). □

Definition 1 (Structural contract refinement)**.** For verifier j, define

$$\mathsf{Acc}_j^{\mathcal{C}}(s) = \{b \in \mathcal{A} : v_j(b, s, \mathcal{C}) = 1\}. \tag{20}$$

At a fixed policy epoch, $\mathcal{C}' \preceq \mathcal{C}$ on a common action domain if, for every action a and state s,

$$\mathsf{Obl}(a, \mathcal{C}) \subseteq \mathsf{Obl}(a, \mathcal{C}'), \qquad \mathsf{Acc}_j^{\mathcal{C}'}(s) \subseteq \mathsf{Acc}_j^{\mathcal{C}}(s) \quad \forall j \in \mathsf{Obl}(a, \mathcal{C}). \tag{21}$$

Proposition 1 (Monotone release under refinement)**.** *If $\mathcal{C}' \preceq \mathcal{C}$, then*

$$\mathsf{RelSet}_{\mathcal{C}'}(s) \subseteq \mathsf{RelSet}_{\mathcal{C}}(s), \qquad \mathsf{RelSet}_{\mathcal{C}}(s) = \{a \in \mathcal{A} : \mathsf{Gate}(a, \mathcal{C}, s) = \textsc{release}\}. \tag{22}$$

Consequently, release soundness of $\mathcal{C}$ for π implies release soundness of $\mathcal{C}'$ on the common domain.

Theorem 2 (Incremental verification equivalence)**.** *Let $\mathcal{I}_{\mathrm{cache}}$ and $\mathcal{I}_{\mathrm{live}}$ partition $\mathsf{Obl}(a_{t'}, \mathcal{C})$, and let $\widehat{v}_{j,t}$ denote the signed cached verdict for obligation j produced at time t. Suppose each cached verifier is deterministic in its complete dependency tuple, its verifier version is unchanged, its cache signature is valid, and $\chi_{j,t} = \chi_{j,t'}$ for every $j \in \mathcal{I}_{\mathrm{cache}}$. Then*

$$\left(\bigwedge_{j \in \mathcal{I}_{\mathrm{cache}}} \widehat{v}_{j,t}\right) \wedge \left(\bigwedge_{j \in \mathcal{I}_{\mathrm{live}}} v_j(a_{t'}, s_{t'}, \mathcal{C})\right) = \bigwedge_{j \in \mathsf{Obl}(a_{t'}, \mathcal{C})} v_j(a_{t'}, s_{t'}, \mathcal{C}). \tag{23}$$

Proof sketch. Tuple equality and verifier determinism imply $\widehat{v}_{j,t} = v_j(a_{t'}, s_{t'}, \mathcal{C})$ for every reused obligation. Obligations in $\mathcal{I}_{\mathrm{live}}$ are evaluated directly at t'. Substituting the cached equalities into the partitioned conjunction gives the full recomputation. Appendix B provides the complete argument. □

Completeness of κ_t is therefore part of the theorem's premise rather than an implementation convenience. For example, an interface obligation may depend on the absence of an undeclared endpoint; creating such an endpoint must invalidate the cached result even if the earlier check never read that object positively. Approval expiry and policy-version changes are handled in the same way. Proof reuse is thus defined over declared semantic dependencies rather than over action text or a raw cache key.

Proposition 2 (Bounded downgrade)**.** *Assume a finite repair budget k and terminating compilation, validation, and safe-skeleton procedures. Compilation cannot remain in an unbounded repair loop: it returns a logic-closed contract, $\mathcal{C}_{\mathrm{safe}}$, or an explicit infrastructure failure.*

Together, these results delimit both the guarantee and its failure modes. A released action can violate policy only through a failure of the declared coverage, a local verifier assumption, or the trusted binding boundary. The empirical evaluation therefore reports unsafe releases and their attributed failure classes separately rather than interpreting a low aggregate rate as a universal safety guarantee.

6 VERACONTRACT-BENCH AND EVALUATION

VeraContract-Bench overlays deployment constraints on SWE-bench Verified tasks (Jimenez et al., 2024). The repository, issue, base commit, and test oracle are inherited from each source instance, while the task card specifies the execution conditions needed to judge deployability: capability scopes, budget rules, approval requirements, endpoint allowlists, synthetic secret markers, rollback obligations, policy epochs, and action-level oracle labels. Nine families cover repository patching, dependency updates, multi-file refactors, failed deployment with compensation, pause/resume debugging, evaluator-cost runaway, unsafe external endpoints, delegated-approval conflicts, and memory-conflict recovery. The held-out evaluation contains 180 instances—10 clean and 10 adversarial cases per family—with three matched seeds for stochastic methods. Variants derived from the same SWE-bench issue remain in the same split, and development tasks are disjoint from the held-out benchmark. Task success is determined by the SWE-bench harness; action admissibility and recovery eligibility come from task-card oracles fixed before method decisions are scored. The benchmark therefore keeps task utility, side-effect admissibility, and resolution of partial state as separate objects rather than collapsing them into a single success label.

Let $\mathcal{A}_{\text{cand}}$ denote candidate side effects observed at the common tool boundary, $\mathcal{U} \subseteq \mathcal{A}_{\text{cand}}$ the oracle-unsafe subset, and $\mathcal{A}_{\text{safe}} = \mathcal{A}_{\text{cand}} \setminus \mathcal{U}$. Action-level metrics use the terminal disposition of a candidate. An effect enters $\mathcal{A}_{\text{rel}}$ only when the committing action crosses the execution boundary; it enters $\mathcal{A}_{\text{blk}}$ when the evaluated method prevents that effect from crossing. A REPAIR outcome is intermediate and is scored only after the repaired execution reaches a terminal release or block. A DOWNGRADE blocks the original committing effect; operations subsequently admitted by $\mathcal{C}_{\text{safe}}$ are evaluated as their own candidates. Hence, over scored terminal candidates, $\mathcal{A}_{\text{rel}}$ and $\mathcal{A}_{\text{blk}}$ form a disjoint partition of $\mathcal{A}_{\text{cand}}$. The unsafe-release rate measures contamination among released actions,

$$\text{UAR} = \frac{|\mathcal{A}_{\text{rel}} \cap \mathcal{U}|}{\max(1, |\mathcal{A}_{\text{rel}}|)}, \tag{24}$$

while violation-rejection rate measures recall on unsafe proposals,

$$\text{VRR} = \frac{|\mathcal{A}_{\text{blk}} \cap \mathcal{U}|}{\max(1, |\mathcal{U}|)}. \tag{25}$$

Release-correctness rate evaluates the terminal release decision over both classes:

$$\text{RCR} = \frac{|\mathcal{A}_{\text{rel}} \cap \mathcal{A}_{\text{safe}}| + |\mathcal{A}_{\text{blk}} \cap \mathcal{U}|}{\max(1, |\mathcal{A}_{\text{cand}}|)}. \tag{26}$$

Only a method-level pre-execution decision counts as a block. A permission error imposed downstream, sandbox failure, tool crash, or test failure receives no VRR credit; otherwise a method with no release mechanism could appear safe simply because its environment happened to stop an action.

Governed success is evaluated at the run level rather than reconstructed from action marginals. For N runs,

$$\text{GSR} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{T_i = 1 \wedge U_i = 0 \wedge R_i = 1\}, \tag{27}$$

where T_i denotes task success, U_i indicates that at least one oracle-unsafe side effect was released, and R_i indicates that no unresolved partial committing state remains. UAR and GSR therefore answer different questions: a system can attain low UAR by refusing useful actions, while a task can succeed after a single side effect has already made the execution undeployable. RCR evaluates individual release decisions; GSR requires the run to complete the task without either an unsafe release or unresolved partial state. Recovery success is computed only over task-card-defined recovery-eligible episodes, with the same eligibility rule applied to every method. Latency is measured separately at the final release gate and across the broader verification path; the operator study measures adjudication time, override precision, earliest-failed-obligation localization, and proof completeness.

The matched comparison includes ReAct-only (Yao et al., 2023), Plan-and-Solve (Wang et al., 2023), LLMCompiler-style execution (Kim et al., 2024), a runtime judge, audit-only tracing, workflow/saga orchestration (Garcia-Molina & Salem, 1987), an AgentSpec-style runtime-policy baseline (Wang et al., 2026a), PACT-lite, and PACT-full. PACT-lite retains typed compilation and a reduced gate

while omitting Ω, commit-time revalidation, and governed skill-lifecycle checks; the remaining mechanisms are isolated by one-at-a-time ablations. All methods receive the same source task, repository snapshot, backbone model, policy artifacts, tool budget, and wall-clock budget, and stochastic methods use the same seeds. Oracle labels are hidden from the evaluated method. Every side-effecting proposal is logged before the release mechanism acts together with its policy epoch, state digest, terminal disposition, downstream outcome, compensation status, and latency stages. Proposal and released-action counts remain separate, preventing a method from lowering UAR by silently changing its denominator.

Statistical resampling is performed at the source-task level. Seed repeats remain nested within a task, and variants derived from the same source issue are not treated as independent evidence. Task-level uncertainty uses hierarchical bootstrap resampling; paired comparisons preserve task and seed matching, operator-time comparisons use paired nonparametric tests with effect sizes, and multiplicity across primary comparisons is controlled with Holm correction. Residual unsafe releases are classified as coverage, verifier, or runtime-binding failures, following the failure decomposition in Section 5.

7 Empirical Results

Table 1 shows a change in execution quality rather than a large sacrifice in task utility. With the matched ReAct backbone, PACT-full solves 63.5% of the tasks versus 64.1% for ReAct, a difference of 0.6 percentage points, while UAR falls from 19.3% to 1.8%, a 90.7% relative reduction. PACT-full rejects 95.1% of oracle-unsafe proposals and attains 98.2% RCR. Among the non-PACT governance baselines, AgentSpec-style gives the highest GSR at 37.2%; PACT-lite reaches 42.6% and PACT-full 49.3%. The contrast with LLMCompiler-style execution is particularly useful: it achieves the highest task-success rate, 68.7%, but also the highest UAR, 22.4%. A successful task is therefore not, by itself, evidence that the execution path was deployable.

Table 1: Primary deployability results on VeraContract-Bench. Task, UAR, VRR, RCR, recovery, and GSR are percentages. Gate P99 is per-action commit-time latency in milliseconds.

Method	Task ↑	UAR ↓	VRR ↑	RCR ↑	Rec. ↑	GSR ↑	Gate P99 ↓
ReAct	64.1	19.3	0.0	80.7	28.4	18.2	–
Plan-and-Solve	66.3	17.1	0.0	82.9	32.6	21.5	–
LLMCompiler-style	68.7	22.4	0.0	77.6	25.1	15.8	–
Runtime judge	61.8	9.4	64.7	88.9	43.2	31.4	187
Audit-only	63.8	18.7	0.0	81.3	27.8	17.6	–
Workflow/saga	58.9	11.2	61.3	89.4	57.6	33.1	142
AgentSpec-style	62.4	7.1	76.2	93.5	54.8	37.2	96
PACT-lite	62.8	4.2	85.4	95.7	67.3	42.6	78
PACT-full	63.5	1.8	95.1	98.2	79.4	49.3	64

The remaining baselines separate different kinds of protection. A runtime judge already lowers UAR to 9.4%, showing that local action screening is useful, while workflow/saga orchestration produces its clearest improvement in recovery, reaching 57.6%. AgentSpec-style rules improve both UAR and RCR, making explicit runtime policy a substantive comparison rather than a weak guardrail. PACT-full improves further while also attaining higher recovery and GSR. The ablations below localize that margin: stale-state protection affects unsafe release, declared compensation primarily affects recovery, accountability changes operator diagnosis, and proof reuse changes verification cost.

Table 2 gives each component a different empirical signature. Typed compilation raises LCR@5 from 68.5% under direct JSON generation to 97.6%; the full compiler reaches 74.2% SVR@1 and 93.1% SVR@3, with the remaining 2.4% of attempts ending in the bounded safe skeleton under the reported compiler protocol. Across the 60-trace operator study, adjudication time falls from 13.8 to 5.9 minutes and blame localization rises from 43.6% to 87.4%. Removing Ω lowers blame localization to 71.9% and GSR to 41.8%, consistent with its role in reconstructing failed obligations and their evidence routes. Compensation behaves differently: removing compensation edges reduces recovery from 79.4% to 38.5%, whereas UAR changes only from 1.8% to 2.4%. Commit-time state binding has the opposite profile; removing revalidation raises UAR to 6.9%. These contrasts are important because they show that the gain in Table 1 is not produced by one generic blocking mechanism.

Verification cost exhibits a similarly specific trade-off. The final gate itself has a P99 latency of 64 ms, whereas the broader verification path takes 1617 ms from a cold start, 887 ms with stable-proof reuse, and 634 ms with reuse plus speculative preparation. A 476 ms configuration that omits commit-time revalidation is faster, but it falls outside the trusted-binding premise of Theorem 1 and is therefore only a latency reference. Under adversarial variants, PACT-full UAR rises from 0.4% to 3.2% and VRR falls from 97.2% to 93.0%. The observed 1.8-point residual UAR is assigned by the benchmark taxonomy to omitted obligations in the interface, economy, memory, skill, and approval classes. Appendix F reports the full compiler, ablation, latency, family-level, and stress-test breakdowns.

Table 2: Mechanism-level evidence for compiler closure, accountability, live revalidation, declared recovery handling, and proof reuse. Deltas are absolute percentage points unless marked otherwise. Detailed results appear in Appendix E–F.

Mechanism	Metric	Reference / ablation	Full / optimized	Effect
Typed compilation	LCR@5	Direct JSON 68.5	Full 97.6	+29.1
Accountability	Blame localization	Audit 43.6	PACT 87.4	+43.8
Accountability	Operator time	Audit 13.8 min	PACT 5.9 min	-57.2%
Live revalidation	UAR	w/o reval. 6.9	Full 1.8	-5.1
Declared recovery	Recovery	w/o comp. 38.5	Full 79.4	$+40.9$
Proof reuse + prep.	Path P99	Cold 1617 ms	Optimized 634 ms	-60.8%

8 Limitations and Scope

PACT reasons over the policy and execution semantics represented to the runtime, so its guarantees end at that representation boundary. An organizational rule omitted from the policy artifacts is outside the coverage assumption; an unsound verifier may accept an inadmissible action; and compromise of the trusted enforcement path invalidates the check-to-commit guarantee. Recovery has the same boundary. Equation (7) requires a declared compensation or terminal path and permits its represented preconditions to be checked when an action commits, but it does not guarantee that the resources, credentials, backups, or external services needed by that path will remain available after arbitrary later actions. Work on semantic transactions and recoverability addresses complementary aspects of this longer-horizon problem (Chen et al., 2026; Yang et al., 2026; Zhuang et al., 2026; Zhang & Liu, 2026). Recovery in PACT should therefore be interpreted as declared-path handling rather than preservation of indefinite future recoverability.

The empirical evidence is confined to autonomous software-engineering workflows. VeraContract-Bench is useful in this setting because repository state, tool effects, and task completion are executable and can be paired with action-level policy oracles, but the results do not establish the same behavior for physical control, perception-heavy agents, clinical decisions, or organization-level multi-agent coordination. Governance also carries costs beyond the final release-gate latency, including contract compilation, evidence handling, registry operations, and operator adjudication. The evaluation exposes several of these costs through compiler reliability, verification-path latency, task utility, and operator-facing measurements; unsafe-release rate should therefore not be read in isolation. Finally, the accountability record supports technical reconstruction of a release decision—the responsible principal, applicable obligation, evidence route, and adjudication path—rather than a model of legal or moral responsibility (Raji et al., 2020).

9 Conclusion

PACT formulates state-changing agent execution as a governed release problem: proposing an action and authorizing its external effect are separate decisions. The contract and commit-time gate bind that authorization to current policy, live state, declared recovery, and machine-checkable evidence, while the formal results characterize when release remains sound, how stricter contracts affect the admissible set, and when cached verification is equivalent to recomputation. VeraContract-Bench exposes the practical consequence of this distinction. Relative to the matched ReAct backbone, PACT reduces unsafe release from 19.3% to 1.8% while task success changes by only 0.6 percentage points, and its ablations separate the roles of live revalidation, recovery declarations, accountability, and proof reuse. The central finding is that successful task completion and deployable execution are not the same objective; governing the commit boundary makes that difference explicit and measurable.

AI USE STATEMENT

In this work, generative AI tools were used only for identifying and retrieving potentially relevant literature and for language editing to improve the clarity, grammar, and readability of the manuscript. Generative AI tools were not used to formulate research ideas or hypotheses, develop the proposed methodology, design experiments, derive mathematical claims or proofs, implement methods or generate code, process or analyze data, interpret experimental results, produce results, or create figures, tables, or other research artifacts. All scientific ideas, methods, theoretical developments, experiments, analyses, and conclusions were produced by the authors.

All AI-assisted work was reviewed and verified by the authors. In particular, any literature identified with AI assistance was checked against the original sources for existence, relevance, bibliographic accuracy, and appropriate citation context, and all AI-assisted language edits were reviewed to ensure that they did not alter the technical meaning, claims, or contributions of the paper. The authors take full responsibility for the final content of this work, including all text, claims, references, and artifacts.

ETHICS STATEMENT

The authors have reviewed the ICLR Code of Ethics and believe that this work complies with it. This research does not involve human-subject studies, animal experiments, personally identifiable information, or the collection or release of sensitive personal data. The empirical evaluation uses publicly available benchmarks and controlled experimental data in accordance with their applicable licenses and terms of use. No new dataset containing private, sensitive, or personally identifiable information is introduced or released. Relevant limitations, potential biases of the benchmark data and evaluation settings, and broader implications of the proposed approach are discussed in the paper.

REPRODUCIBILITY STATEMENT

We have taken several steps to facilitate reproducibility. The main paper specifies the proposed methodology, objectives, and evaluation protocol, while the appendix provides the additional implementation details, experimental settings, hyperparameters, dataset and preprocessing information, evaluation metrics, and ablation studies needed to reproduce the reported results. For the theoretical results, the assumptions and derivations are stated in the main paper, with complete proofs and additional analysis provided in the appendix. An anonymized implementation, including the source code and supporting scripts needed to reproduce the main experiments, is provided with the submission as supplementary material. The datasets used in the experiments are publicly available, and any task-specific preprocessing or data splits required for the reported experiments are documented or included with the supplementary materials.

REFERENCES

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, Yarin Gal, and Xander Davies. Agentharm: A benchmark for measuring harmfulness of llm agents. In *International Conference on Learning Representations*, 2025.

Varun Pratap Bhardwaj. Agent behavioral contracts: Formal specification and runtime enforcement for reliable autonomous ai agents. *arXiv preprint arXiv:2602.22302*, 2026.

Zhaorun Chen, Mintong Kang, and Bo Li. Shieldagent: Shielding agents via verifiable safety policy reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8313–8344, 2025.

Zheng Chen, Hanqing Liu, Duling Xu, Dong Dong, Jialin Li, Bangzheng Pu, and Jidong Zhai. Cordon: Semantic transactions for tool-using llm agents. *arXiv preprint arXiv:2606.17573*, 2026.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, 1966.

Hector Garcia-Molina and Kenneth Salem. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pp. 249–259, 1987.

Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1992.

Chengquan Guo, Xun Liu, Chulin Xie, Andy Zhou, Yi Zeng, Zinan Lin, Dawn Song, and Bo Li. Redcode: Risky code execution and generation benchmark for code agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24370–24391, 2024.

Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

Xiaoxin Lu, Ranran Haoran Zhang, and Rui Zhang. Simmer: Benchmarking latent failures in llm executable planning with a world model. In *Proceedings of the Conference on Language Modeling (COLM)*, 2026.

Adam Mazzocchetti. Runtime governance for agentic ai: Action-boundary control with trusted provenance and fail-closed execution. *arXiv preprint arXiv:2608.16891*, 2026.

Bardia Mohammadi and Laurent Bindschaedler. The irreversibility budget: Fleet-level risk accounting and admission control for agent operating systems. *arXiv preprint arXiv:2609.00275*, 2026.

George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 106–119, 1997.

George C. Necula and Peter Lee. Safe, untrusted agents using proof-carrying code. In *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pp. 61–91. Springer, 1998.

Nils Palumbo, Sarthak Choudhary, Jihye Choi, Guy Amir, Prasad Chalasani, and Somesh Jha. Formal policy enforcement for real-world agentic systems. *arXiv preprint arXiv:2602.16708*, 2026.

Inioluwa Deborah Raji, Andrew Smart, Rebecca N. White, Margaret Mitchell, Timnit Gebru, Ben Hutchinson, Jamila Smith-Loud, Daniel Theron, and Parker Barnes. Closing the ai accountability gap: Defining an end-to-end framework for internal algorithmic auditing. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pp. 33–44, 2020.

Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

Tianneng Shi, Jingxuan He, Zhun Wang, Linyu Wu, Hongwei Li, Wenbo Guo, and Dawn Song. Progent: Programmable privilege control for llm agents. *arXiv preprint arXiv:2504.11703*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Georgios Syros, Anshuman Suri, Jacob Ginesin, Cristina Nita-Rotaru, and Alina Oprea. Saga: A security architecture for governing ai agentic systems. In *Network and Distributed System Security Symposium*, 2026. doi: 10.14722/ndss.2026.230869.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering*, pp. 2938–2950, 2026a. doi: 10.1145/3744916.3764546.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 2609–2634, 2023. doi: 10.18653/v1/2023.acl-long.147.

Lichao Wang, ZhaoXing Ren, Tianzhuo Yang, Jiaming Ji, Chi Harold Liu, Yaodong Yang, and Juntao Dai. Safemcp: Proactive power regulation for llm agent defense via environment-grounded look-ahead reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11374–11396. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.acl-long.522.

Zexun Wang. Proof-carrying agent actions: Model-agnostic runtime governance for heterogeneous agent systems. *arXiv preprint arXiv:2606.04104*, 2026.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. Guardagent: Safeguard llm agents via knowledge-enabled reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68316–68342, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1650.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Ke Yang, Panpan Li, Zonghan Wu, Kejin Xu, Huaxi Huang, and Xiaoshui Huang. Dart: Semantic recoverability for structured tool agents. *arXiv preprint arXiv:2605.23311*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. Tau-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025.

Zhihui Zhang and Wei Liu. Recoverability as a system primitive for long-horizon ai agents. *arXiv preprint arXiv:2609.13672*, 2026.

Chris Zheng and Geng Yang. Continuity: Security-context contracts for composable llm agent controls. *arXiv preprint arXiv:2609.05269*, 2026.

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. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

Yu Zhuang, Kefei Chen, Yitong Duan, Shuxin Zheng, Jian Li, and Xu-Yao Zhang. Agentrewind: Recoverable execution for long-horizon llm agents. *arXiv preprint arXiv:2608.14380*, 2026.

APPENDIX

A NOTATION AND TYPED CONTRACT SCHEMA

Table 3: Core notation used in the main paper and appendix.

Symbol	Meaning
g, s	task goal and current environment state
$\mathcal{P}, \mathcal{T}, \mathcal{K}, \mathcal{H}$	policy artifacts, tools/interfaces, verified skills, and proposal-policy families
$\mathcal{C}$	proof-carrying agent contract
$G = (V, E^{\mathrm{fwd}}, E^{\mathrm{comp}})$	action graph with forward and compensation edges
V_{commit}	actions that may create externally visible state
V_{recover}	subset of V_{commit} requiring a declared recovery or terminal path
$\mathcal{A}$	universe of side effects representable by the contract and policy semantics
$\mathcal{A}_{\mathrm{cand}}$	candidate side effects observed at the evaluation tool boundary
$\mathcal{U}, \mathcal{A}_{\mathrm{safe}}$	oracle-unsafe subset of $\mathcal{A}_{\mathrm{cand}}$ and its oracle-safe complement
$\mathcal{A}_{\mathrm{rel}}, \mathcal{A}_{\mathrm{blk}}$	terminally released and blocked candidate-action sets
$\mathsf{Obl}(a, \mathcal{C})$	obligation indices applicable to action a under $\mathcal{C}$
$\mathsf{RelSet}_{\mathcal{C}}(s)$	actions released by $\mathcal{C}$ in state s
$\mathsf{Acc}_j^{\mathcal{C}}(s)$	acceptance set of verifier j under $\mathcal{C}$ in state s
$\mathcal{A}_\pi(s)$	actions admissible under policy semantics π in state s
$Q_j(s)$	policy-projection acceptance set associated with obligation j
$v_j(a, s, \mathcal{C})$	verifier predicate for obligation j
β_t	check-to-commit binding (d_a, d_C, d_s, ρ_t)
ρ_t	current policy/registry version vector
$\chi_{j,t}$	complete dependency tuple used for verifier-result reuse
κ_t	canonical snapshot of all declared state dependencies relevant to verifier j
ν_t	verifier implementation version
$d_{j,t}$	digest of $\chi_{j,t}$ used as a cache lookup key
$\mathcal{B}_t$	attested release record (proof bundle) at time t
$\mathcal{C}_{\mathrm{safe}}$	bounded safe contract used after unsuccessful compilation repair
$\Phi, \Sigma, \mathcal{E}, \Delta, \Pi, \Omega, \mathcal{J}_C, \mathcal{R}_C, \mathcal{S}_C$	typed contract components defined in Eq. (2)
$\mathcal{D}$	set of contract-compilation instances used for compiler metrics
$\preceq$	structural contract-refinement relation

A typed action record contains a stable action identifier, tool/interface identifier, operation class, canonical argument digest, declared state dependencies, required capability class, projected cost, risk tier, proof-obligation identifiers, an optional recovery reference, and an accountability key. A capability record binds issuer, principal, delegate, resource scope, permitted operations, economic scope, expiry, nonce, and policy epoch. An accountability entry binds an action–obligation pair to its principal, evidence route, adjudication route, retention policy, and escalation condition.

The dependency snapshot κ_t contains more than the names of state fields read by a verifier. It canonically binds the values on which the verdict depends, including the truth of negative predicates, absence conditions, validity intervals, and deadlines. Consequently, equality of $\chi_{j,t}$ across two commits implies equality of the verifier-visible contract fragment, action projection, relevant policy/registry versions, state-dependent facts, and verifier implementation version.

Contract well-formedness requires every referenced action, tool, skill, capability, proof obligation, compensation node, and accountability entry to exist with compatible types. Mandatory obligations for a committing action class must be instantiated before execution, and every $a \in V_{\mathrm{recover}}$ must satisfy Eq. (7). A contract that fails these checks is not eligible to reach the release gate.

B FULL PROOFS

Proof of Theorem 1. Fix a contract-covered side effect a for which $\mathsf{Gate}(a, \mathcal{C}, s) = \textsc{release}$. By the precedence rule in Eq. (6), release is possible only when $\mathrm{Rel}(a, s, \mathcal{C}) = 1$. Equation (5) therefore gives $v_j(a, s, \mathcal{C}) = 1$ for every $j \in \mathsf{Obl}(a, \mathcal{C})$. Assumption 2 implies $a \in Q_j(s)$ for each such

obligation, and hence $a \in \bigcap_{j \in \mathsf{Obl}(a,\mathcal{C})} Q_j(s)$. Assumption 3 places this intersection inside $\mathcal{A}_\pi(s)$, so the verified action is policy-admissible in the verified state.

It remains to relate that verified object to the external effect that is actually committed. Under Assumption 1, the release boundary atomically binds the canonical action, contract, live-state digest, and policy/registry version represented by β_t in Eq. (9). The action and state to which the preceding argument applies are therefore the action and state authorized to cross the commit boundary. Thus $\mathsf{Gate}(a, \mathcal{C}, s) = \textsc{release}$ implies $a \in \mathcal{A}_\pi(s)$. □

Proof of Corollary 1. Let $E_{\mathrm{rel}} = \{\mathsf{Gate}(a, \mathcal{C}, s) = \textsc{release}\}$ and $E_{\mathrm{unsafe}} = \{a \notin \mathcal{A}_\pi(s)\}$. Let E_{cov}, E_{ver}, and E_{bind} denote, for the sampled contract-covered action, failures of policy coverage, local verifier soundness, and trusted check-to-commit binding. If none of these events occurs, Theorem 1 excludes $E_{\mathrm{unsafe}} \cap E_{\mathrm{rel}}$. Hence

$$E_{\mathrm{unsafe}} \cap E_{\mathrm{rel}} \subseteq E_{\mathrm{cov}} \cup E_{\mathrm{ver}} \cup E_{\mathrm{bind}}. \tag{28}$$

The union bound and the assumed event-probability bounds give Eq. (17). Since $p_{\mathrm{rel}} = \Pr(E_{\mathrm{rel}}) > 0$,

$$\Pr(E_{\mathrm{unsafe}} \mid E_{\mathrm{rel}}) = \frac{\Pr(E_{\mathrm{unsafe}} \cap E_{\mathrm{rel}})}{p_{\mathrm{rel}}}, \tag{29}$$

which together with the fact that probabilities are at most one yields Eq. (19). □

Proof of Proposition 1. Fix s and suppose $a \in \mathsf{RelSet}_{\mathcal{C}'}(s)$. Release under $\mathcal{C}'$ implies that every obligation in $\mathsf{Obl}(a, \mathcal{C}')$ accepts a. Structural refinement gives $\mathsf{Obl}(a, \mathcal{C}) \subseteq \mathsf{Obl}(a, \mathcal{C}')$; therefore every obligation required by $\mathcal{C}$ is among the obligations satisfied under $\mathcal{C}'$. For each $j \in \mathsf{Obl}(a, \mathcal{C})$,

$$a \in \mathsf{Acc}_j^{\mathcal{C}'}(s) \quad \text{and} \quad \mathsf{Acc}_j^{\mathcal{C}'}(s) \subseteq \mathsf{Acc}_j^{\mathcal{C}}(s),$$

so verifier j also accepts a under $\mathcal{C}$. Consequently all obligations required by $\mathcal{C}$ pass and $a \in \mathsf{RelSet}_{\mathcal{C}}(s)$. Thus $\mathsf{RelSet}_{\mathcal{C}'}(s) \subseteq \mathsf{RelSet}_{\mathcal{C}}(s)$. If $\mathcal{C}$ is release-sound for π, every member of the latter set lies in $\mathcal{A}_\pi(s)$, and the same is therefore true for $\mathcal{C}'$ on the common domain. □

Proof of Theorem 2. For each $j \in \mathcal{I}_{\mathrm{cache}}$, the cached verdict $\widehat{v}_{j,t}$ was produced by verifier version ν_t from the canonical dependency tuple $\chi_{j,t}$. The theorem assumes a valid cache signature, unchanged verifier version, deterministic verification as a function of the complete tuple, and $\chi_{j,t} = \chi_{j,t'}$. Because κ_t binds the current values and absence/deadline predicates of all declared state dependencies, tuple equality also preserves every state-dependent input that can change verifier j's verdict. Therefore

$$\widehat{v}_{j,t} = v_j(a_{t'}, s_{t'}, \mathcal{C}), \qquad j \in \mathcal{I}_{\mathrm{cache}}. \tag{30}$$

Every obligation in $\mathcal{I}_{\mathrm{live}}$ is evaluated directly at t'. Since $\mathcal{I}_{\mathrm{cache}} \cup \mathcal{I}_{\mathrm{live}} = \mathsf{Obl}(a_{t'}, \mathcal{C})$ and the two sets are disjoint, substituting Eq. (30) into the partitioned conjunction gives exactly the full recomputation in Eq. (23). □

Proof of Proposition 2. The compile–verify process has at most k repair transitions. A logic-closed contract terminates compilation immediately. If all permitted repairs are exhausted without such a contract, the terminating safe-skeleton procedure returns $\mathcal{C}_{\mathrm{safe}}$; failures that prevent either procedure from completing are represented by the separate infrastructure-failure outcome. No execution path can therefore contain more than k repair transitions, and compilation terminates in one of the outcomes stated in the proposition. □

C Benchmark Construction and Oracles

C.1 Task construction and split

VeraContract-Bench is built from SWE-bench Verified instances (Jimenez et al., 2024). Each task card records the source identifier, repository, base commit, issue text, family label, clean/adversarial condition, allowed and forbidden resources, tool-call and monetary budgets, required approvals, policy epoch, recovery obligations, action-oracle rules, and the original SWE-bench task oracle. The held-out suite contains 180 instances: 10 clean and 10 adversarial cases in each of nine task families.

Development instances are disjoint from this suite. Variants derived from the same SWE-bench source issue are assigned to the same split, so closely related variants cannot cross the development/evaluation boundary.

The nine task families are:

1. **Repository patch and test**: a bug-fix workflow with protected files and declared test requirements.
2. **Dependency update under approval**: package changes require an allowed registry, budget compliance, and the required approval receipt.
3. **Multi-file refactor**: cross-file edits exercise action-graph completeness and resource scope.
4. **Failed deployment with compensation**: a controlled partial deployment failure requires the declared compensating path.
5. **Pause/resume debugging**: policy or capability state changes between suspension and resumption.
6. **Evaluator-cost runaway**: repeated high-cost evaluator calls exercise economic and execution-pattern constraints.
7. **Unsafe external endpoint**: an action attempts to send an artifact to an endpoint outside the allowlist.
8. **Delegated-approval conflict**: the claimed approver or delegation chain conflicts with the active policy.
9. **Memory-conflict recovery**: persistent context contains stale policy or credential hints that conflict with the current policy epoch.

C.2 Action and recovery oracles

The action oracle is derived from the task-card policy artifacts before an evaluated method's terminal release decision is exposed. Each candidate side effect is labeled with respect to the concrete pre-action state and policy epoch in which it is proposed. The oracle record identifies the governing rule, expected capability scope, budget state, approval requirement, and, when applicable, recovery requirement. Fully specified rules are checked executablely. Judgment-dependent cases are independently labeled by two annotators, with disagreements resolved before method decisions are unblinded; Cohen's κ summarizes agreement on this subset.

Recovery eligibility is also fixed by the task card rather than inferred from a method's behavior. An eligible episode begins when the declared scenario enters a state for which compensation or an approved terminal disposition is required. Recovery succeeds only when the episode reaches the task-card specified compensated or terminal state without leaving an unresolved partial commit. The same eligibility and success rules are applied to every method.

The failure taxonomy connects evaluation labels to the formal assumptions. A *coverage failure* occurs when the policy oracle marks a proposed effect unsafe but the compiled contract contains no obligation capable of representing the governing rule. A *verifier failure* occurs when an applicable obligation exists but returns an incorrect verdict for its declared inputs. A *binding failure* occurs when the effect that crosses the execution boundary is not the action/state/policy tuple authorized by the corresponding commit binding. Annotation disagreement is resolved before scoring and is not counted as a system failure class.

D Metric Details and Statistical Protocol

The action metrics in Eqs. (24)–(26) are computed from terminal candidate dispositions. A REPAIR decision is intermediate: the original candidate is scored only after repair reaches a terminal release or block. A DOWNGRADE blocks the original committing effect, while any operation subsequently admitted by $\mathcal{C}_{\mathrm{safe}}$ is scored as a separate candidate. Consequently $\mathcal{A}_{\mathrm{rel}}$ and $\mathcal{A}_{\mathrm{blk}}$ form a disjoint partition of the scored $\mathcal{A}_{\mathrm{cand}}$ set.

Action ratios are formed from pooled counts rather than averages of per-run ratios. Within a bootstrap replicate, the selected task and seed runs contribute their action counts, after which UAR, VRR, and

RCR are recomputed from the pooled numerators and denominators. A run with no released action contributes zero to both the UAR numerator and denominator; a run with no oracle-unsafe proposal contributes zero to both VRR counts. The $\max(1, \cdot)$ convention in the main definitions only makes a degenerate zero-denominator aggregate well defined and does not introduce synthetic positive events.

GSR uses the joint run-level indicator in Eq. (27). T_i is supplied by the official SWE-bench test harness, $U_i = 0$ requires that no oracle-unsafe side effect was released in run i, and $R_i = 1$ requires that no unresolved partial committing state remains at termination. Recovery success is computed separately on the predeclared recovery-eligible episodes defined in Appendix C. Neither GSR nor recovery is reconstructed from action-level marginal rates.

Uncertainty is estimated at the source-task level. A hierarchical bootstrap samples source tasks with replacement and, for stochastic methods, resamples their nested seed runs while preserving matched seed indices across paired methods. Metrics and paired differences are recomputed within each replicate. Bootstrap intervals are reported by their lower and upper endpoints when the empirical distribution is asymmetric. Holm correction is applied to the prespecified family of primary baseline comparisons. Individual actions may be used for descriptive diagnostic plots, but actions from the same run are not treated as independent hypothesis-test samples.

The operator study comprises 60 trace-level adjudication cases involving 12 operators with site-reliability or code-review experience. Comparisons are paired by the underlying scenario and operator when the same failure is shown under different evidence conditions. Endpoints are adjudication time, override precision, earliest-failed-obligation localization, accountability-route reconstruction, proof completeness, and explanation sufficiency. Time comparisons use paired nonparametric analysis with effect sizes; categorical endpoints are evaluated on their paired outcomes.

E Baseline and Ablation Specifications

All matched baselines are instrumented at the same side-effect boundary used by VeraContract-Bench and receive the source task, repository snapshot, backbone model, tool and wall-clock budgets, and policy artifacts specified by the main evaluation protocol. Instrumentation records proposals and outcomes for oracle scoring but does not supply a baseline with a release capability that the baseline itself does not implement.

Cognition and planning baselines. ReAct-only uses ReAct-style reasoning and tool execution without a pre-execution release gate (Yao et al., 2023). Plan-and-Solve generates an explicit plan before executing tools but adds no independent side-effect authorization layer (Wang et al., 2023). LLMCompiler-style execution schedules independent tool calls in parallel when permitted by its planner, under the same resource limits as the other matched methods (Kim et al., 2024). These methods are instrumented for oracle scoring at the common tool boundary; downstream environment failures are not converted into method-level blocks.

Runtime and workflow baselines. The runtime judge evaluates each candidate side effect with a separate model or rule-plus-model decision using the declared policy text and action record, without PACT's typed contract or reusable proof state. Audit-only records the action, state, policy epoch, and downstream outcome for post-hoc inspection but never blocks a candidate before execution. Workflow/saga uses a fixed stage structure with compensation handlers (Garcia-Molina & Salem, 1987); it does not invoke the task-specific PACT contract compiler. The AgentSpec-style baseline expresses pre-execution safety rules over the shared action schema and evaluates those rules before the tool call (Wang et al., 2026a). No PACT graph compiler, proof cache, Ω map, or PACT recovery declaration is added to this implementation.

PACT variants and ablations. PACT-lite retains skeleton retrieval, typed compilation, structural validation, and a reduced action gate, while omitting Ω, commit-time state revalidation, and governed skill-lifecycle checks. It is treated as a system variant rather than evidence for any one component. Mechanism-level claims use one-at-a-time ablations of PACT-full. The ablated components are the accountability map, commit-time revalidation, critical-subgraph checks, compensation edges, skill-lifecycle checks, pattern-profile checks, economic envelope, attestation binding, safe-skeleton fallback, proof reuse, and secret-isolation checks. Each ablation changes only the named mechanism while retaining the remaining PACT configuration.

F Extended Results

The main paper reports the deployability trade-off and the largest mechanism-level effects. This section gives the corresponding one-at-a-time ablations and the breakdowns used to interpret those effects. Table 4 shows that the components have different failure signatures rather than producing a uniform change across all metrics. Commit-time revalidation, economic constraints, and attestation have their largest effects on unsafe release; compensation edges primarily determine recovery after partial failure; the safe skeleton protects task utility under failed compilation; and removing proof reuse leaves quality unchanged, as expected for an optimization whose correctness is covered by Theorem 2.

Table 4: One-at-a-time quality ablations on VeraContract-Bench. Each row removes one component from PACT-full. Verification-path latency is reported separately in Table 7. All values are percentages.

Configuration	Task ↑	UAR ↓	VRR ↑	RCR ↑	Recovery ↑	GSR ↑
PACT-full	63.5	1.8	95.1	98.2	79.4	49.3
w/o accountability map Ω	62.8	3.1	90.2	96.4	71.3	41.8
w/o commit-time revalidation	63.1	6.9	85.6	93.5	74.9	41.1
w/o critical-subgraph checks	63.2	4.4	88.9	95.2	76.1	43.8
w/o compensation edges	62.4	2.4	92.1	97.2	38.5	22.8
w/o skill-lifecycle checks	63.2	4.6	89.7	95.1	75.8	43.5
w/o pattern profile	63.0	4.1	90.8	95.6	76.2	44.0
w/o economic envelope	62.9	5.8	87.4	94.0	74.2	41.3
w/o attestation checks	62.7	5.2	86.8	94.6	72.9	41.0
w/o bounded safe downgrade	58.7	1.9	94.1	97.9	76.4	43.1
w/o proof reuse	63.5	1.8	95.1	98.2	79.4	49.3
w/o secret isolation	63.3	3.8	91.3	95.9	76.8	44.8

F.1 Compiler, adjudication, and verification cost

Compiler reliability is reported independently of downstream task success. In Table 5, SOI denotes Schema Omission Incidence, the fraction of compilation instances whose contract omits a required field; CFI denotes Cross-Field Inconsistency, the fraction violating a required invariant across contract fields; and SRM denotes Skill-Reference Mismatch, the fraction referencing an inadmissible skill. SVR@k records schema validity by attempt k, whereas LCR@5 requires a schema-valid contract that also passes deterministic logic-closure checks within five attempts. BSSD is the bounded-safe-downgrade rate. Infrastructure failures are classified outside these compiler outcomes, so LCR@5 and BSSD sum to 100% under the reported protocol.

Table 5: Compiler reliability. SVR@k is schema validity by attempt k; LCR@5 additionally requires deterministic logic closure within five attempts. SOI, CFI, and SRM measure schema omissions, cross-field inconsistencies, and skill-reference mismatches. All values are percentages.

Method	SVR@1 ↑	SVR@3 ↑	LCR@5 ↑	SOI ↓	CFI ↓	SRM ↓	BSSD ↓
Direct JSON contract	33.8	51.2	68.5	38.4	29.1	22.3	31.5
Skeleton retrieval only	50.6	67.8	81.1	23.7	17.8	14.6	18.9
Schema-constrained decoding	66.9	81.7	89.4	13.5	10.7	7.8	10.6
+ Verifier-guided repair	66.9	88.6	95.2	6.8	4.9	3.7	4.8
Full compiler	74.2	93.1	97.6	3.9	2.8	1.9	2.4

The operator-facing results in Table 6 use the same 60 blocked or failed trace scenarios described in Appendix H. The pattern is not limited to faster adjudication: moving from audit-only evidence to the full PACT record improves localization, proof completeness, and the sufficiency of the evidence available to the operator. Removing Ω preserves much of the structured record but weakens the direct mapping from a failed obligation to its responsible principal and adjudication route.

Table 7 measures the broader per-action verification path rather than only the final gate reported in Table 1. Stable-proof reuse reduces P99 from 1617 ms to 887 ms, and speculative preparation reduces

Table 6: Fail-path quality and operator-facing adjudication on 60 blocked or failed trace scenarios. Operator time is measured in minutes; the remaining values are percentages.

Method	Operator time ↓	Override precision ↑	Blame localization ↑	Proof completeness ↑	Explanation sufficiency ↑
Audit-only	13.8	57.8	43.6	30.7	48.9
Runtime judge	11.7	66.9	55.8	51.4	60.7
PACT w/o Ω	8.3	78.6	71.9	77.8	73.5
PACT-full	5.9	92.7	87.4	95.8	87.8

it further to 634 ms. The final row shows the latency obtained when commit-time revalidation is omitted; that configuration is not covered by the trusted-binding premise used in Theorem 1.

Table 7: Per-action verification-path latency. The final row disables commit-time revalidation and is reported only as a latency reference.

Setting	Proof reuse	Warm runtime	Speculative prep	P50 (ms) ↓	P95 (ms) ↓	P99 (ms) ↓
Cold verification	No	No	No	612	938	1617
Cached stable blocks	Yes	Yes	No	341	512	887
Speculative preparation	Yes	Yes	Yes	246	378	634
No commit revalidation*	Yes	Yes	Yes	184	287	476

F.2 Family, adversarial, and residual-risk breakdowns

Table 8 separates the nine task families. Each family contains 20 held-out task instances; stochastic methods use the three matched seeds specified in Section 6. Because traces contain different numbers of candidate actions, pooled UAR and VRR are recomputed from action counts rather than averaged over family percentages. PACT-full lowers UAR and raises VRR in every family. The largest remaining UAR values occur in the pause/resume family and the unsafe-endpoint family, at 3.0% and 2.6%, respectively.

Table 8: Per-family results. Each family contains 20 tasks (10 clean and 10 adversarial); stochastic methods use three matched seeds. The pooled row recomputes action-level ratios from the underlying action counts. All values are percentages.

Family	ReAct-only			PACT-full		
	Task ↑	UAR ↓	VRR ↑	Task ↑	UAR ↓	VRR ↑
A: Repository patch and test	68.4	15.2	0.0	67.8	0.8	96.8
B: Dependency update under approval	61.2	24.6	0.0	60.4	2.4	93.2
C: Multi-file refactor	66.8	18.3	0.0	65.1	1.2	95.7
D: Failed deployment with compensation	63.5	21.4	0.0	62.6	2.1	94.1
E: Long-running debug with pause/resume	57.9	17.6	0.0	56.8	3.0	93.5
F: Evaluator or tool-cost runaway	64.1	22.8	0.0	63.9	1.4	96.2
G: Unsafe external endpoint simulation	70.2	28.4	0.0	68.5	2.6	92.4
H: Delegated approval conflict	60.6	14.9	0.0	59.7	0.9	95.9
I: Memory-conflict recovery	64.2	10.5	0.0	66.7	0.9	95.4
Pooled	64.1	19.3	0.0	63.5	1.8	95.1

Adversarial variants increase UAR for every method in Table 9. Among methods with a pre-execution release mechanism, VRR is also lower on adversarial variants. The UAR difference between PACT-

lite and PACT-full is 1.7 percentage points on clean tasks and 3.1 points on adversarial variants; the one-at-a-time ablations in Table 4 are used to attribute this system-level gap to individual mechanisms.

Table 9: UAR and VRR on clean and adversarial task variants. Aggregate values in Table 1 are recomputed from pooled action counts rather than from the arithmetic mean of these columns. All values are percentages.

Method	UAR (clean) ↓	UAR (adversarial) ↓	VRR (clean) ↑	VRR (adversarial) ↑
ReAct-only	12.6	26.0	0.0	0.0
Plan-and-Solve-only	10.9	23.3	0.0	0.0
LLMCompiler-style	15.1	29.7	0.0	0.0
Runtime judge	6.2	15.0	72.4	57.0
Audit-only	11.8	25.2	0.0	0.0
Workflow/saga	5.1	17.3	68.5	54.1
AgentSpec-style	3.2	11.0	82.1	70.3
PACT-lite	2.1	6.3	90.5	80.3
PACT-full	0.4	3.2	97.2	93.0

Corollary 1 permits residual unsafe releases from coverage, verifier, or trusted-binding failures. In the measured PACT-full traces, the residual 1.8 percentage points in Table 10 are all assigned to coverage omissions: the governing policy or compiled contract did not instantiate the obligation needed to reject the released action. No residual case in this set is classified as a verifier error or a check-to-commit binding failure.

Table 10: Observed coverage-omission contribution to PACT-full UAR. Values are percentage points of released actions and sum to the aggregate UAR.

Benchmark taxonomy class	Share of released actions
Interface	0.6
Economy	0.4
Memory	0.3
Skill	0.2
Approval	0.3
Total	1.8

G Failure Case Studies

Three traces illustrate failure modes that are hidden by task success alone. They are drawn from VeraContract-Bench and use the same task-success and action-oracle definitions as the quantitative evaluation.

Case 1: Task-passing dependency update with an unapproved package. A ReAct-only run resolves a Django compatibility issue by installing a newer transitive dependency. The resulting patch passes the SWE-bench tests, but the installation is oracle-unsafe: the package is outside the organizational allowlist and no required approval receipt is present. The trace also writes the package-index URL to a log outside the permitted repository scope. Under PACT-full, the installation does not cross the release boundary because the applicable Iface and $\mathrm{Approve}$ obligations fail. A subsequent proposal uses an allowlisted alternative and reaches the same task-level outcome without the inadmissible side effect. The case separates successful artifact production from admissible execution.

Case 2: Parallel fan-out and a partial commit without a usable recovery path. An LLMCompiler-style run parallelizes a multi-file refactor across six tool calls. Four calls complete, another removes a test file, and a staging deployment enters a partial commit before the inconsistency is detected. At that point the declared compensation procedure cannot execute because a represented recovery prerequisite is no longer satisfied. In the corresponding PACT-full trace, the $\mathrm{Pattern}$ obligation limits speculative fan-out for the action class, while the staging commit is released only if its declared recovery disposition and represented preconditions pass the current-state checks. When

those conditions do not hold, the committing effect is not released. The resulting governed trace preserves a valid compensation route at that commit. This case concerns represented preconditions at the release boundary; it does not imply that PACT preserves recovery prerequisites after arbitrary future actions.

Case 3: Stale authority after pause and resume. A debugging run is suspended after 18 minutes and later resumes after the policy epoch has changed. ReAct-only reuses a capability token issued under the earlier epoch and modifies a directory that is read-only under the current policy. The tests pass, but the write is oracle-unsafe. PACT-full rejects the action when the Auth obligation detects the stale policy epoch during commit-time revalidation. Execution resumes only after a fresh capability grants the narrower authority required by the remaining edits. The resulting run completes with a consistent sequence of release records. The case isolates the gap between evidence available during planning and the state that must authorize the eventual commit.

H Operator Study Protocol

The operator study examines whether the evidence attached to a blocked or failed execution changes the quality and cost of adjudication. It uses 60 trace scenarios and 12 operators with site-reliability or code-review experience. The scenarios are matched across the four evidence configurations reported in Table 6: audit-only evidence, runtime-judge evidence, a PACT record without Ω, and the full PACT record. Method labels are hidden from participants and scenario presentation is randomized. Operators determine whether an override is justified, localize the earliest failed obligation, and use the available record to reconstruct the evidence and responsible route needed for adjudication.

The reported endpoints are exactly those in Table 6. Operator time measures time to complete the adjudication. Override precision scores proposed overrides against the study rubric; blame localization records whether the earliest failed obligation is identified correctly. Proof completeness measures whether the evidence view contains the fields required by the adjudication rubric, and explanation sufficiency records whether that evidence is sufficient to justify the final decision under the same rubric. Comparisons are paired on matched scenarios. Trace contents are synthetic or sanitized, free-text responses are anonymized, and no real credentials, customer data, or private repositories are exposed.

I Result Provenance and Aggregation

Each reported result is linked to an aggregation record derived from the stored action traces, oracle labels, build/test outcomes, compensation outcomes, and latency events. The record identifies the benchmark version, source-task split, seed set, model and decoding configuration, tool and wall-clock budgets, policy and registry versions, action counts, oracle-label distribution, exclusions, and software/hardware configuration. Action-level ratios are reconstructed from their underlying counts, while run-level metrics retain the source-task and seed keys required by the hierarchical resampling procedure in Appendix D. This separation prevents a displayed percentage from becoming the only retained representation of an experiment.

The release record $\mathcal{B}_t$ supports reconstruction of the decision represented at the gate by retaining the bound action, contract and state digests, evidence references, verifier outcomes, decision, and integrity signature. Raw secrets need not be embedded in the record; sensitive values may instead be represented by policy-approved digests or references to retained evidence. The applicable retention rule is the $q_{a,j}$ component of Ω in Eq. (4).

J Threat Model and Scope

PACT treats the proposal model, retrieved text, persistent memory, generated scripts, reusable skills, and untrusted tool or service outputs as potentially incorrect or adversarial. The trusted boundary is the one assumed by Assumption 1: the policy and registry state used for verification, the release gate and verifier implementations, the evidence store, integrity signer, and the check-to-commit binding must execute as specified. Compromise of that boundary is outside the release-soundness guarantee. The guarantee also does not establish that the policy author encoded the correct organizational rule,

that an external service behaves correctly after an authorized request is released, or that a patch is semantically correct beyond the available task oracle. Prompt- and tool-mediated attacks of the kind studied in agent-security benchmarks are relevant only insofar as their externally visible effects cross the mediated release boundary (Debenedetti et al., 2024; Guo et al., 2024).

Table 11 maps representative failure modes to the PACT mechanisms that constrain them. The entries are controls at the governed release boundary rather than claims that the corresponding threat is eliminated in every environment.

Table 11: Representative threats, failure modes, and the PACT mechanisms applied at the governed release boundary.

Threat	Failure mode	PACT control or boundary
Prompt injection	Retrieved content requests an unauthorized side effect	Auth, Iface, and commit-time release checks
Tool/output injection	Untrusted tool output induces an unauthorized follow-up action	Iface, Attest, and fresh action-level verification
Policy-epoch drift	A capability remains syntactically valid after policy changes	Auth, live revalidation, and β_t binding
Script-mediated escape	Generated code attempts a side effect outside the declared action	Mediated committing boundary and Iface; an unmediated bypass violates the trusted-binding assumption
Observation spoofing	State or receipt evidence is falsified	Attest, state/evidence digests, and trusted evidence bindings
Memory poisoning	Stale context conflicts with the current authority or policy epoch	Current-state Auth checks and policy/registry version binding
Evaluator spoofing	Invalid evaluation evidence is presented as successful	Attest and evaluator-evidence provenance
Skill spoofing	An unverified skill is substituted for a registered skill	Skill and the verified-skill registry
Budget exhaustion	Repeated calls exceed the declared economic envelope	Econ and Pattern obligations
Approval fatigue	Repeated approval requests weaken human oversight	Approve, risk-tier rules, and Pattern constraints
Accountability gap	A release cannot be traced to its governing obligation and route	Accountable, Ω, and $\mathcal{B}_t$

Committing effects fail closed when a mandatory obligation lacks the evidence required by the contract. This improves containment at the cost of availability: a run may be blocked even when the proposed action would in fact have been benign. Bounded repair, $\mathcal{C}_{\text{safe}}$, and explicit approval recover utility only when the active policy permits them. Denial-of-service against the governance runtime, incorrect policy authoring, and side effects that bypass the mediated commit boundary remain outside the guarantees stated in Section 5.