

# AARC: A Machine-Verifiable Audit and Reliability Contract for Tool-Using AI Agents

Stamatis-Christos Saridakis

2026

**Manuscript revision 1.1.1 (October 2, 2026).** This revision updates literature positioning and publication metadata only; the normative AARC contract, schemas, reference monitors, and conformance semantics remain v1.1.0.

## Abstract

Tool-using AI agents increasingly execute long-horizon workflows whose correctness depends not only on model outputs but also on authorization, execution order, policy state, and the integrity of the resulting evidence. Existing agent benchmarks expose substantial inconsistency and security failures, while recent runtime-enforcement systems focus on policy languages, action validation, or predictive risk control. This paper addresses a complementary systems problem: *how can heterogeneous agent runtimes emit a portable execution record that can be independently checked for a defined set of reliability invariants?*

We introduce the Agentic Audit & Reliability Contract (AARC), a model- and policy-engine-independent runtime contract for externally observable agent execution. AARC v1.1.0 defines machine-verifiable runtime-event and state-vector schemas; immutable role, objective, and policy anchors; an RFC 8785 canonical hash chain; fail-closed tool authorization receipts; separated Actor–Critic–Judge decision validation; and append-only change provenance. The contract deliberately excludes hidden chain-of-thought and does not equate trace conformance with semantic correctness.

The accompanying artifact provides executable Python and TypeScript reference monitors, JSON Schema validation, cross-language canonicalization vectors, adversarial fault injection, and a release pipeline that fails closed if conformance, bibliography, paper, or packaging checks fail. The deterministic conformance suite accepts the clean reference trace and rejects all 11 injected structural and semantic violations, including attacks whose hash chains are recomputed after mutation. These results establish enforcement of the stated contract invariants—not general agent safety—and provide a reproducible foundation for integrating stronger policy engines and agent benchmarks.

## 1 Introduction

Language-model agents have moved from text generation toward persistent systems that call APIs, modify external state, communicate with users, and coordinate multi-step workflows. This shift changes the reliability problem. A successful final answer is no longer sufficient evidence that the execution was safe, authorized, reproducible, or even internally consistent. Reliability evaluations have therefore begun to emphasize consistency, robustness, predictability, and safety rather than a single task-success score [12]. Interactive benchmarks such as  $\tau$ -bench and  $\tau^2$ -Bench expose repeated-run inconsistency and policy violations in realistic tool-use settings [16, 2], while AgentDojo, Agent Security Bench, and DUMA-Bench demonstrate that tool-using agents remain vulnerable to prompt injection, memory poisoning, and interaction-dependent attacks [7, 18, 1].

A second line of work moves enforcement outside the language model. AgentSpec defines customizable runtime rules; Pro2Guard adds proactive probabilistic risk estimation; VIGIL checks temporal and value-flow policies over finite execution traces; and ActGov validates proposed tool actions against task-scoped authorization and security policies [14, 15, 10, 19]. These systems establish that runtime mediation is an important safety boundary. They do not, however, eliminate a separate interoperability problem: different agent stacks still encode identity, intent, authorization, state transitions, approval evidence, remediation, and audit history in incompatible ways.

AARC targets this missing layer. Its purpose is not to replace a policy language, an agent benchmark, a model evaluator, or a security monitor. It specifies a small external contract that those systems can emit and verify. The central design principle is that claims about an agent run should be grounded in machine-checkable evidence that survives implementation boundaries.

This paper makes five contributions:

1. **A portable runtime contract.** We define versioned JSON schemas for runtime events and state snapshots, including explicit identity, intent, trace, tool, uncertainty, policy, assumption, and evidence fields.
2. **Machine-checkable safety invariants.** We formalize event-order integrity, immutable anchors, permit-bound tool execution, separated approval identities, and terminal lifecycle semantics.
3. **Cross-runtime cryptographic determinism.** Event hashing uses RFC 8785 JSON Canonicalization Scheme (JCS) [13], avoiding language-dependent JSON serialization in cryptographic commitments.
4. **Executable reference enforcement.** Python and TypeScript monitors implement the same observable contract, including fail-closed tool authorization and offline verification.
5. **Adversarial conformance evidence.** The repository contains a deterministic suite that attacks both byte-level integrity and higher-level semantics, including rehashed forgeries, and makes the results a release gate.

AARC uses the normative terms MUST, SHOULD, and MAY according to BCP 14 [3, 9]. The paper describes artifact version v1.1.0. The deterministic hash of the normative v1.1.0 specification set used by the build is 07cad58d6a584657e8a07b5671413de5a64495313ade55b8e8398f99a9b57bf6.

## 2 Problem Statement and Scope

### 2.1 Execution reliability as an evidence problem

Consider a tool-using agent that produces a correct user-visible answer after several API calls. The final answer does not by itself establish that: (i) the same task policy remained active throughout the run; (ii) each side effect was authorized before execution; (iii) the recorded sequence was not reordered or altered; (iv) an approval was produced by a distinct reviewer rather than self-approved; or (v) the log contains enough information for an external auditor to reconstruct the decision boundary.

AARC treats these questions as properties of the *execution envelope*. The model may remain stochastic and opaque; the envelope is deterministic and verifiable.

### 2.2 What AARC does not claim

AARC conformance does not prove that a model’s factual statements are true, that a policy is ethically correct, that a tool itself is uncompromised, or that a compliant run will complete its task successfully. A hash chain without a trusted checkpoint or signature does not prove who authored the history. Likewise, distinct logical Actor, Critic, and Judge identifiers do not guarantee organizational independence if a single compromised principal controls all three identities.

These exclusions are intentional. Overstating such guarantees would convert an audit contract into a security claim it cannot support.

## 3 Related Work

### 3.1 Agent reliability and evaluation

$\tau$ -bench introduced repeated-trial reliability through the  $\text{pass}^k$  metric for tool-agent-user interaction [16];  $\tau^2$ -Bench extended evaluation to dual-control interaction [2]. More broadly, Rabanser et al. argue that

aggregate success rates hide operational differences across consistency, robustness, predictability, and safety [12]. Gao and Zhou further show that benchmark scores can be weakly supported when outcome checks do not retain sufficient execution evidence [8]. AARC is aligned with this evidence-oriented view: it does not define task success, but it gives evaluators a standard execution record against which additional outcome evidence can be attached.

Security benchmarks stress different parts of the agent stack. AgentDojo evaluates prompt injection in agents that act over untrusted tool data [7]; Agent Security Bench covers attacks across prompts, tools, and memory [18]; DUMA-Bench studies security under interactive dual control [1]. AARC can be used around such benchmarks to standardize the provenance and authorization evidence generated by a run, but it is not itself an attack benchmark.

## 3.2 Observability standards

OpenTelemetry’s Generative AI semantic conventions standardize cross-platform telemetry for agent invocations, model calls, and tool execution [11]. This is an important interoperability substrate: polyglot agent stacks can emit consistently named spans and attributes into existing observability backends. AARC operates at a different semantic layer. An OpenTelemetry span can describe that a tool call occurred; AARC specifies whether the corresponding execution is conforming with an immutable task anchor, a prior authorization receipt, ordered integrity, and lifecycle rules. The two approaches are therefore complementary: AARC events can be exported through OpenTelemetry adapters, while OpenTelemetry remains the transport and observability convention rather than the authorization or conformance contract.

## 3.3 Runtime enforcement

AgentSpec shows that structured runtime rules can prevent unsafe actions with low enforcement overhead [14]. Pro2Guard adds predictive intervention using probabilistic model checking [15]. VIGIL translates behavioral requirements into trace-aware symbolic constraints for runtime checking [10]. ActGov performs policy-constrained validation of proposed actions and verifies policy evolution with SMT-based counterexample checking [19]. SABER focuses attention on small state-mutating actions whose local errors can have large downstream consequences [6].

AARC is intentionally orthogonal to these policy mechanisms. It standardizes the evidence that an authorization decision occurred, what request it bound, which policy anchor governed it, and whether later execution matched that permit. An AARC Tool Gateway may therefore delegate policy evaluation to AgentSpec, VIGIL, ActGov, a capability system, an ABAC/RBAC engine, or a deployment-specific decision point.

## 3.4 Transactional containment and agentic transaction processing

Cordon introduces a semantic-transaction boundary for tool-using agents: reversible local mutations and outward-facing effects are staged, the composed execution flow is validated before commit, and recovery metadata is retained for rollback and audit [5]. Mnemosyne develops Agentic Transaction Processing (ATP), treating generated workflow actions as untrusted proposals until deterministic admission under an executable constraint set; its runtime combines an append-only transition log, effective-state projection, dependency-safe compensation, and bounded reactive repair [4]. These systems are substantially closer to AARC than per-call guardrails because they also move authority from model-generated text into runtime-controlled state transitions.

The emphasis remains different. Cordon and ATP define transaction, staging, admission, and repair semantics for consequential workflows. AARC defines a portable evidence and conformance contract that heterogeneous runtimes can emit and independently verify, including canonical event commitments, immutable anchors, authorization receipts, separated approval evidence, and lifecycle closure. A transactional runtime can therefore emit AARC records without AARC prescribing that runtime’s transaction or repair algorithm.

### 3.5 Concurrent and subsequent assurance-contract work

The publication-grade AARC v1.1.0 repository state was committed publicly on September 25, 2026 at Git commit `3aa462a04dd5d0b5778012577fd379e34bd0e71f`. The present v1.1.1 manuscript revision leaves the normative v1.1.0 contract unchanged and updates literature positioning only.

Zabolotnii’s *Runtime Assurance Contract* (RAC), submitted September 30, 2026, formalizes how runtime evidence changes an agent’s operational authority through autonomy boundaries, component eligibility, evidence state, transition policy, human-review capacity, and mandatory non-compensatory gates [17]. RAC overlaps with AARC in externalizing consequential authorization from the model, carrying explicit policy/evidence context, and using fail-closed runtime gating. Its central problem is nevertheless different: RAC asks when accumulated evidence permits an authority transition and prevents mandatory failures from being offset by aggregate scores, whereas AARC asks how an executed agent run can expose a portable, machine-verifiable record of ordering, anchors, authorization, approval, and provenance across runtimes.

RAC is therefore treated here as subsequent related work rather than as an influence on the September 25 AARC v1.1.0 artifact. This chronology establishes version provenance only; it does not imply dependence or copying.

## 4 Design Goals

AARC v1.1.0 follows six goals.

1. **External verifiability.** Conformance must be decidable without access to hidden chain-of-thought.
2. **Fail-closed side effects.** A tool request without a valid authorization decision must not become a conforming execution.
3. **Stable intent anchors.** Role, objective, and policy identity must be immutable within a run.
4. **Cross-language determinism.** Cryptographic event commitments must not depend on a language’s default JSON serializer.
5. **Policy-engine independence.** The evidence contract should not prescribe a particular authorization language.
6. **Versioned evolution.** Breaking semantic changes require a new specification version; change history must be auditable rather than silently rewritten.

Figure 1 shows the architecture.

## 5 Formal Execution Model

### 5.1 Runs, events, and canonical commitments

Let a run be

$$R = (E, A, \Pi),$$

where  $E = (e_1, \dots, e_n)$  is the committed event sequence,  $A$  is the immutable anchor tuple, and  $\Pi$  is the verifier policy relevant to externally visible actions.

For each event  $e_i$ , define:

$$e_i = (id_i, type_i, seq_i, run_i, task_i, A_i, p_i, h_{i-1}, h_i),$$

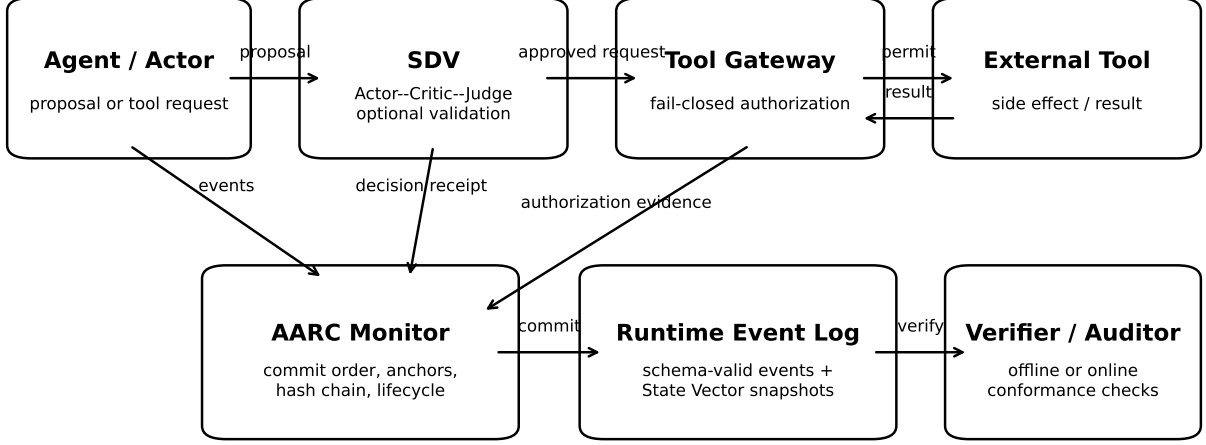
where  $p_i$  is the event payload and  $h_i$  is its commitment. The genesis predecessor is

$$h_0 = 0^{256}.$$

For  $i \geq 1$ ,

$$h_i = \text{SHA256}(\text{JCS}(e_i \setminus \{h_i\})),$$

## AARC v1.1.0: observable reliability contract around the model



The model may propose actions; external runtime components remain authoritative for conformance and side effects.

Figure 1: AARC places a verifiable execution contract around the model. The model proposes; external runtime components remain authoritative for approval, side effects, commit order, and verification.

where JCS is RFC 8785 canonical JSON [13].

The anchor tuple is:

$$A = (H(\text{role}), H(\text{objective}), H(\text{policy})).$$

A conforming trace requires  $A_i = A$  for every event.

### 5.2 Tool authorization relation

Let a tool request be  $q = (t, d_a, r)$ , where  $t$  is the tool identity,  $d_a$  is the canonical argument digest, and  $r$  is the request event identifier. An authorization event yields permit

$$p = (t, d_a, r, \pi, a),$$

binding the tool, argument digest, request, policy anchor  $\pi$ , and authorization-event identifier  $a$ .

For an execution event  $x$ , the verifier requires:

$$\text{Exec}(x) \Rightarrow \exists p [x.\text{permit} = p.a \wedge x.\text{tool} = p.t \wedge x.\text{request} = p.r].$$

When the verifier is configured with an external allow-set or policy view, it also requires  $p.t \in \Pi$ .

### 5.3 Separated decision validation

For optional Actor–Critic–Judge validation, an approval receipt is:

$$d = (a, c, j, h_p, h_c, \mathcal{E}, v),$$

where  $a, c, j$  are Actor, Critic, and Judge identities;  $h_p$  and  $h_c$  are proposal and critique digests;  $\mathcal{E}$  is the evidence-reference set; and  $v$  is the verdict. For  $v = \text{approved}$ :

$$a \neq c, \quad a \neq j, \quad c \neq j, \quad |\mathcal{E}| > 0.$$

AARC reserves the acronym *ACJ* for the Agentic Change Journal. The Actor–Critic–Judge pattern is therefore referred to descriptively or as Separated Decision Validation (SDV).

## 5.4 Verifier properties

The reference verifier checks the following safety properties.

**P1: Ordered integrity.** Sequence numbers are contiguous, each predecessor hash equals the prior committed hash, and each event commitment recomputes under JCS.

**P2: Anchor invariance.** All events carry the genesis role, objective, and policy anchors.

**P3: Permit safety.** No `tool.executed` or `tool.failed` event is valid without a matching authorization permit.

**P4: Non-self-approval evidence.** An approved SDV receipt requires pairwise-distinct role identities plus proposal, critique, and evidence commitments.

**P5: Lifecycle closure.** A run begins with `task.start` and a completed trace ends with exactly one terminal event. No event may follow termination.

## 5.5 Conditional guarantees

P1 detects mutation, deletion, duplication, and reordering relative to a trusted checkpoint, assuming SHA-256 collision resistance. It does not prevent an attacker who controls the entire committer from fabricating a new internally consistent history and replacing every checkpoint. Deployments that require origin authenticity must add signatures, authenticated storage, or a transparency log. Similarly, P3 prevents unauthorized execution only when the Tool Gateway is an unavoidable side-effect boundary.

These conditions are part of the contract rather than implementation footnotes.

## 6 State Vector and Audit Evidence

The AARC State Vector is a schema-valid snapshot intended for independent audit. Version v1.1.0 requires:

- **Identity:** agent, root-agent, optional parent-agent, role, and role digest;
- **Intent:** immutable objective and policy digests;
- **Trace coordinates:** run, task, step, sequence, and predecessor event commitment;
- **Inputs and outputs:** structured, deployment-defined summaries;
- **Tools:** structured tool-status records;
- **Confidence metadata:** optional confidence, uncertainty, and evidence-coverage values;
- **Policy:** an explicit policy identifier and deployment-defined constraints;
- **Assumptions and evidence references:** externally inspectable statements and pointers.

The schema deliberately records *declared assumptions*, not internal reasoning traces. This separation supports auditability without making hidden chain-of-thought a conformance dependency.

| Threat                               | AARC mechanism                                                                          | Required assumption / residual risk                                                             |
|--------------------------------------|-----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| Post-commit log mutation             | Sequence and JCS/SHA-256 hash-chain verification                                        | A trusted checkpoint, signature, or protected copy is needed against total-history replacement. |
| Role/objective/policy drift          | Immutable genesis anchors checked on every event                                        | The authoritative monitor itself must not be compromised.                                       |
| Tool execution without authorization | Request–authorization–execution permit binding                                          | The gateway must be an unavoidable side-effect path.                                            |
| Self-approval                        | Pairwise-distinct Actor/Critic/Judge identities and evidence-bound receipts             | Distinct identifiers do not prove independent organizational control.                           |
| Ambiguous cross-runtime hashing      | RFC 8785 JCS canonicalization and shared test vectors                                   | Inputs must remain in the I-JSON/JCS domain.                                                    |
| Missing audit evidence               | Schema-required fields, explicit assumptions, evidence references, ACJ and patch ledger | AARC cannot guarantee that supplied evidence is truthful or complete beyond the schema.         |

Table 1: Threat model and the assumptions under which AARC mechanisms are meaningful.

## 7 Tool Gateway and Change Provenance

### 7.1 Fail-closed Tool Gateway

The Tool Gateway mediates side effects outside model-generated text. A request produces `tool.requested`; the gateway commits exactly one authorization or denial decision; execution references the authorization event. Unknown tools, malformed policy state, missing permits, or verifier errors must resolve to denial.

AARC does not prescribe how the policy decision is computed. This is the main interoperability boundary with specialized runtime-enforcement research.

### 7.2 Agentic Change Journal

The Agentic Change Journal (ACJ) records material changes to model, prompt, policy, configuration, code, or runtime posture. Published entries are append-only: corrections supersede older entries rather than rewriting them. The patch ledger links remediation artifacts and verification evidence to these change records.

This distinction is important operationally. Runtime events answer “what happened in this run?”; ACJ and patch-ledger records answer “what changed in the system, why, and how was the change verified?”

## 8 Threat Model

Table 1 summarizes the intended boundary.

## 9 Reference Implementation

The artifact contains independent Python and TypeScript monitors. Both implement event commitments, immutable anchors, tool request/permit binding, SDV receipts, and offline trace verification. Python uses the Trail of Bits RFC 8785 implementation; TypeScript uses the RFC-referenced `canonicalize` implementation. A shared canonicalization vector includes a floating-point normalization case that would fail under naive language-default serialization.

Two JSON Schemas define the wire contract: `AARC_RUNTIME_EVENT.schema.v1_1_0.json` and `AARC_STATE_VECTOR.schema.v1_1_0.json`. The schemas themselves are validated against JSON Schema Draft 2020-12, after which positive runtime fixtures are validated.

| Injected fault                 | Rehashed after attack? | Expected detector                  |
|--------------------------------|------------------------|------------------------------------|
| Missing event                  | No                     | sequence/hash continuity           |
| Reordered events               | No                     | sequence/hash continuity           |
| Duplicated event               | No                     | sequence continuity                |
| Payload tamper                 | No                     | event hash                         |
| Policy-anchor drift            | Yes                    | anchor invariance                  |
| Sequence gap                   | Yes                    | contiguous ordering                |
| Broken predecessor link        | No                     | hash continuity                    |
| Execution with forged permit   | Yes                    | permit relation                    |
| Unauthorized tool substitution | Yes                    | verifier policy + permit semantics |
| Actor self-approval            | Yes                    | SDV identity separation            |
| Missing terminal event         | Yes                    | lifecycle closure                  |

Table 2: Adversarial conformance cases. Rehashed cases are intentionally important: they test semantic invariants rather than merely detecting stale checksums.

The reference implementation intentionally keeps the policy engine simple. Its allow-set is not presented as a state-of-the-art policy language; it is a minimal enforcement point used to test the AARC evidence semantics. Production deployments can substitute a stronger decision engine while retaining the same request, authorization, and execution bindings.

## 10 Evaluation

### 10.1 Research questions

We evaluate four questions.

- RQ1** Does a valid reference execution satisfy both schemas and the semantic verifier?
- RQ2** Does the verifier reject representative structural and semantic corruptions, including attacks that recompute the hash chain?
- RQ3** Do Python and TypeScript agree on the canonical cryptographic input for a cross-language JCS test vector?
- RQ4** Does verification scale linearly with event count in the reference implementation?

### 10.2 Adversarial conformance suite

The clean fixture includes a genesis event, state snapshot, valid SDV approval, authorized tool request, permit, execution result, and terminal event. The benchmark then creates 11 corrupted traces.

The deterministic release benchmark accepts the clean trace and rejects all 11 injected faults, for an injected-fault detection rate of 11/11 under this test model. This number must not be interpreted as an empirical estimate of real-world attack coverage. It establishes that the reference verifier enforces the enumerated invariants against these fixtures.

Figure 2 summarizes the verification stack.

### 10.3 Cross-language canonicalization

The interoperability test canonicalizes

$$\{\text{"b"} : 1.0, \text{"a"} : \text{"x"}\}$$

to the RFC 8785 representation

$$\{\text{"a"} : \text{"x"}, \text{"b"} : 1\},$$

## Verification stack and deterministic fault-injection coverage

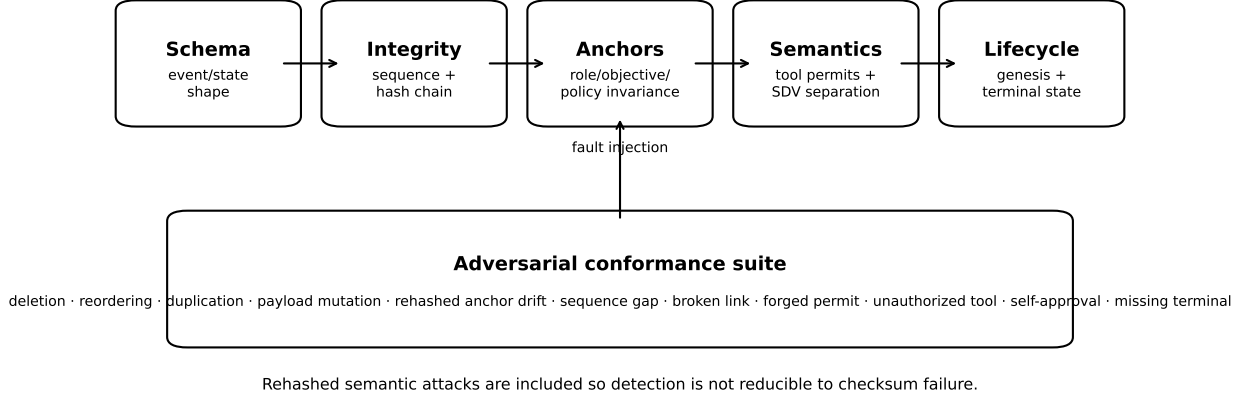


Figure 2: Verification proceeds from schema shape through integrity, anchors, semantic permit/approval constraints, and lifecycle closure.

| Events | Commit $\mu\text{s}/\text{event}$ | Verify $\mu\text{s}/\text{event}$ | Commit events/s | Verify events/s |
|--------|-----------------------------------|-----------------------------------|-----------------|-----------------|
| 102    | 31.83                             | 20.00                             | 31,414          | 50,001          |
| 1002   | 31.14                             | 19.88                             | 32,108          | 50,304          |
| 5002   | 31.22                             | 20.73                             | 32,034          | 48,229          |

Table 3: Median AARC v1.1.0 reference-monitor scaling over five repetitions. Per-event cost remains approximately constant across a  $49\times$  increase in trace length.

then checks the shared SHA-256 digest `cdab067e9f3beb32d1252cfd63e492592fecbf591b0d08cadb24bb17f3864246` in both implementations. This catches a concrete failure mode in naive cross-language hashing, where Python and JavaScript may serialize numerically equivalent values differently.

### 10.4 Scaling results

We measured commit and verification cost on traces containing 100, 1000, and 5000 heartbeat events, taking the median of five repetitions for each size. Table 3 reports the September 25, 2026 publication-gate run on a GitHub-hosted Ubuntu 24.04 runner with CPython 3.12.14. These values are environment-specific and are retained as machine-readable CI artifacts rather than treated as normative constants.

The near-constant per-event cost supports the expected linear behavior of the single-pass verifier: from 102 to 5002 events, median verification cost changes from 20.00 to 20.73  $\mu\text{s}/\text{event}$ , while commit cost changes from 31.83 to 31.22  $\mu\text{s}/\text{event}$ . The dominant operations are JCS canonicalization, SHA-256, and small constant-size semantic checks. These measurements characterize the reference implementation only; they do not include networked policy-engine, storage, signature, or tool-execution latency.

### 10.5 Interpretation

The evaluation is intentionally narrower than AgentSpec, VIGIL, ActGov, or agent-security benchmarks: it does not measure how often AARC discovers previously unknown unsafe model behavior. It tests whether an implementation faithfully enforces the AARC contract. This distinction preserves a clean separation between *contract conformance* and *policy effectiveness*.

## 11 Reproducibility and Release Discipline

The publication pipeline is part of the artifact. Every candidate build:

1. validates both JSON Schemas and a positive fixture;
2. runs Python unit and adversarial conformance tests;
3. emits machine-readable conformance and scaling results;
4. compiles and tests the TypeScript implementation;
5. regenerates architecture figures;
6. computes a deterministic hash over the normative v1.1.0 specification files;
7. fails on LaTeX or BibTeX errors; and
8. builds an arXiv source bundle containing the exact normative specification snapshot used by the paper.

This closes a common gap between a paper’s prose claims and the versioned artifact readers can actually inspect. The embedded specification hash lets a reader detect a paper/spec mismatch at build time.

## 12 Limitations

AARC has several important limitations.

First, the current artifact evaluates conformance properties on synthetic traces; it does not yet report end-to-end task utility or attack success rate on AgentDojo,  $\tau$ -bench, or DUMA-Bench. Those benchmarks measure questions that AARC deliberately does not collapse into the conformance score.

Second, the reference Tool Gateway uses a minimal allow-set. Rich temporal, argument, value-flow, and probabilistic policies are better delegated to systems such as VIGIL, ActGov, AgentSpec, or Pro2Guard.

Third, the event hash chain is not a signature scheme. Strong provenance requires authenticated checkpoints or an external transparency mechanism.

Fourth, SDV role separation is syntactic unless identities are bound to independent authenticated principals.

Finally, AARC standardizes evidence structure but cannot guarantee the semantic truth of evidence supplied by a compromised upstream component.

## 13 Discussion and Future Work

The most useful next step is integration rather than adding another policy language. AARC adapters can wrap modern runtime enforcers and export a common event stream, allowing the same audit verifier to compare heterogeneous agent stacks. Similarly, AgentDojo or  $\tau$ -family trajectories can be enriched with AARC authorization and provenance receipts so benchmark outcomes are accompanied by machine-verifiable execution evidence.

A second direction is authenticated provenance. The current contract already separates hash-chain integrity from authenticity; future profiles can specify detached signatures, hardware-backed identities, or transparency-log checkpoints without changing the event semantics.

A third direction is distributed and multi-agent causality. The v1.1.0 schema carries root and parent agent identifiers but does not yet define a full distributed partial-order protocol. Extending AARC with signed causal links while retaining deterministic local verification is a natural next step.

## 14 Conclusion

AARC addresses a narrow but increasingly important reliability problem: independent verification of the execution envelope around a tool-using AI agent. Version v1.1.0 turns earlier descriptive claims into executable contract semantics: explicit schemas, immutable anchors, RFC 8785 event commitments, fail-closed tool permits, separated approval evidence, append-only change provenance, and cross-language reference monitors.

The main result is not that AARC makes an arbitrary agent safe. It is that a specific class of runtime claims can be represented portably and checked deterministically, with the paper, specification, implementation, tests, and release artifacts cryptographically tied to the same versioned contract. That provides a practical substrate on which stronger policy engines, security benchmarks, and reliability evaluations can build.

## References

- [1] Ivan Aleksandrov, German Kochnev, Sabrina Sadiekh, and Yaroslav Rogoza. Duma-bench: A dual-control multi-agent benchmark for evaluating llm agent security, 2026. arXiv:2609.24662.
- [2] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan.  $\tau^2$ -Bench: Evaluating Conversational Agents in a Dual-Control Environment, 2025. arXiv:2506.07982.
- [3] Scott Bradner. Key words for use in rfc's to indicate requirement levels. RFC 2119, 1997. doi:10.17487/RFC2119.
- [4] Edward Y. Chang and Longling Geng. Mnemosyne: Agentic transaction processing for validating and repairing ai-generated workflows, 2026. arXiv:2607.00269.
- [5] Zheng Chen, Hanqing Liu, Duling Xu, Dong Dong, Jialin Li, Bangzheng Pu, and Jidong Zhai. Cordon: Semantic transactions for tool-using llm agents, 2026. arXiv:2606.17573.
- [6] Alejandro Cuadron, Pengfei Yu, Yang Liu, and Arpit Gupta. Saber: Small actions, big errors — safeguarding mutating steps in llm agents. In *ICLR 2026 Workshop on Memory for LLM-Based Agentic Systems*, 2026.
- [7] 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, Datasets and Benchmarks Track*, 2024.
- [8] Shanshan Gao and Liyi Zhou. Can agent benchmarks support their scores? evidence-supported bounds for interactive-agent evaluation, 2026. arXiv:2605.10448.
- [9] Barry Leiba. Ambiguity of uppercase vs lowercase in rfc 2119 key words. RFC 8174, 2017. doi:10.17487/RFC8174.
- [10] Ying Li, Yanju Chen, Hongbo Wen, Bosi Zhang, Hanzhi Liu, Peiran Wang, Yu Feng, and Yuan Tian. Vigil: Runtime enforcement of behavioral specifications in ai agent skills, 2026. arXiv:2606.26524.
- [11] OpenTelemetry Project. Semantic conventions for generative ai. OpenTelemetry Semantic Conventions, 2026. Accessed September 2026.
- [12] Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of ai agent reliability, 2026. arXiv:2602.16666.
- [13] Anders Rundgren, Brady Jordan, and Samuel Erdtman. Json canonicalization scheme (jcs). RFC 8785, 2020. doi:10.17487/RFC8785.
- [14] Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE)*, 2026. doi:10.1145/3744916.3764546.

- [15] Haoyu Wang, Christopher M. Poskitt, Jun Sun, and Jiali Wei. Pro2guard: Proactive runtime enforcement of llm agent safety via probabilistic model checking, 2025. arXiv:2508.00500.
- [16] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan.  $\tau$ -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains, 2024. arXiv:2406.12045.
- [17] Serhii Zabolotnii. Trust is not a score: Runtime assurance contracts for high-risk ai agents, 2026. arXiv:2609.39717, submitted September 30, 2026.
- [18] Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents. In *International Conference on Learning Representations*, 2025. arXiv:2410.02644.
- [19] Kaiyuan Zhang, Yuke Peng, Ke Jiang, and Yinqian Zhang. Actgov: Governing llm agent actions via policy-constrained validation, 2026. arXiv:2609.24446.