

# Artifact-Bearing Smoke Profiles for Workflow Integration

Anonymous Authors

July 13, 2026

## Abstract

Asynchronous research services often expose a deceptively small integration boundary: submit a job, poll until a terminal status appears, and read an artifact manifest. A wiring problem at that boundary can turn a missing artifact, stale run identifier, malformed manifest, or trace gap into a false success. This paper defines an artifact-bearing smoke profile for that boundary and evaluates it against a local fixture-harness evidence package assembled for this study. The executed evidence contains a 480-row prediction table over 45 finite fixture cases, nine regimes, and five seeds, plus a finite symbolic audit over 64 Boolean assignments. The contract profile produced no false passes on non-pass fixture rows, detected status, manifest, and trace boundary violations in the local matrix, and preserved the distinction between fixture evidence and production reachability. The result supports a scoped claim: a compact smoke profile can make the dispatch, terminal-status, manifest-integrity, byte-integrity, and trace-continuity obligations explicit enough to reject targeted local failures before a generated artifact is treated as a valid research output.

## 1 Introduction

Modern scientific and engineering workflow systems increasingly present themselves as asynchronous services. A caller dispatches a request, receives a run identifier, polls for status, and then reads a manifest that describes generated outputs. That interface is compact, but the outcome it represents is not compact. Between dispatch and artifact publication, a backend may enqueue work, call an executor, persist files, update a run database, write a manifest, and emit traces or metrics. The endpoint can return success even when one of those internal obligations failed. For research automation, the most serious version of this error is an artifact-bearing false pass: the service claims that a run completed even though the required artifact is absent, empty, mislabeled, detached from the run, or unsupported by trace evidence.

The problem studied here is a live integration smoke problem for the backend-to-workflow boundary in a research automation engine. The user-facing task was deliberately narrow: verify that dispatch, polling to completion, and artifact-manifest reading are wired end to end, then read the artifact manifest. A useful answer cannot be just a health check, because a healthy process can still publish an invalid manifest. It also cannot be a full production reliability study, because the available evidence is a local fixture-harness package rather than real endpoint credentials, traffic, or storage traces. The scientific question is therefore precise: what finite profile should a smoke gate enforce before the system may treat an asynchronous workflow attempt as having produced an artifact?

The answer matters beyond this single harness. Testing research distinguishes evidence that finds faults from evidence that proves global absence of faults (Myers et al., 2011; Weyuker, 1988). Continuous integration made fast feedback normal, but empirical work still shows that build automation has costs and limitations (Fowler, 2006; Humble and Farley, 2010; Hilton et al., 2016; Vasilescu et al., 2015). Distributed-system failure studies show that many severe failures arise at ordinary error-handling and integration boundaries, not only in exotic algorithms (Oppenheimer et al., 2003; Yuan et al., 2014; Gunawi et al., 2014). Workflow reproducibility and provenance work adds a second obligation: generated files should be tied to metadata, execution, and reuse context (Wilkinson et al., 2016; Sandve et al., 2013; Moreau and Missier, 2013). A smoke profile for this setting has to combine those lessons. It should be cheap enough to run at every integration boundary and explicit enough that a missing byte stream or manifest mismatch cannot masquerade as success.

The presentation uses five public evidence labels: Local fixture-harness benchmark matrix, Status boundary fixture corpus, Manifest integrity fixture corpus, Trace and verdict fixture corpus, and finite-predicate symbolic audit. These labels are not production telemetry; they are finite local evidence units. The manuscript therefore states claims only inside their validity regime. The local matrix tests fixture behavior over status, manifest, and trace conditions. The symbolic audit checks finite predicate implications and verdict totality. The limitations section states the low-relevance conditions for production reachability: endpoint credentials, project-specific service contracts, trace exports, and real artifact manifests were not supplied.

The contributions are as follows.

- We define an artifact-bearing smoke profile that binds dispatch identity, terminal status, manifest integrity, byte integrity, and trace-stage continuity for a finite asynchronous workflow attempt.
- We present an ordered verdict policy that separates pass, manifest failure, terminal failure, timeout, contract violation, and trace gap without treating metadata presence alone as artifact success.
- We align the manuscript with the executed local evidence package: a 480-row prediction table over 45 finite fixture cases, two manuscript-facing vector figures, evidence-role tables, and a 64-assignment symbolic audit.
- We state the domain-fit boundary for each claim so that local fixture-harness evidence is not confused with production live-backend reliability evidence.

The rest of the paper follows the profile. Section 2 introduces asynchronous artifact-producing workflows and the boundary role of smoke tests. Section 3 synthesizes testing, distributed reliability, reproducibility, provenance, observability, and workflow systems. Section 4 defines the finite objects, predicates, and verdict function. Section 5 explains the smoke profile and local evidence package. Section 6 gives complete finite lemmas. Section 7 reports the executed fixture and symbolic evidence. Section 8 and Section 9 interpret the scope, and the appendix provides reproducibility details, boundary cases, and an evidence map.

## 2 Background and Scope

This section fixes the scope before the formal profile appears. The boundary under study is not every part of a workflow engine. It is the externally visible handoff where a caller receives a run identity, observes terminal progress, and receives enough artifact evidence to decide whether the requested output exists. That framing matters because a smoke gate should remain small while still checking the observations that can turn a broken integration into a false artifact success. The rest of the manuscript keeps this finite boundary separate from broader scheduler, storage, and production reliability questions.

### 2.1 Asynchronous Artifact-Producing Services

An asynchronous artifact-producing service is a service whose externally visible success state is separated in time from request acceptance. A user submits a job, but the user cannot immediately inspect the final artifact. Instead, the service returns an identifier, exposes one or more status observations, and eventually writes a manifest describing outputs. Workflow systems such as Galaxy, Snakemake, Nextflow, and the Common Workflow Language show why that structure is attractive: it separates workflow description, execution, portability, and provenance while allowing large computations to run outside the request path (Goecks et al., 2010; Köster and Rahmann, 2012; Di Tommaso et al., 2017; Amstutz et al., 2016). When the workflow is exposed through a service boundary, however, the same separation creates a risk. The caller may see only the identifier, terminal state, and manifest, so a smoke test has to treat those observations as the evidence object.

This paper uses a finite-attempt scope. A single attempt is one dispatch response, the poll observations that follow it, the manifest records tied to the attempt, byte inspections for the required artifacts, and trace-stage observations when available. The profile does not model all backend internals. It does not assume synchronized clocks across backend, worker, and storage systems, a point consistent with the classic distinction between local observation order and global time in distributed systems (Lamport, 1978). It also does not assert that a scheduler, storage service, or tracing pipeline is globally correct. It states the finite obligations that must be visible before one artifact-producing attempt is accepted.

### 2.2 Smoke Tests and Artifact Boundaries

Smoke tests are valuable because they are small, fast, and interpretable. They are not substitutes for model checking, fault injection, chaos experiments, production monitoring, or long benchmark campaigns. Their job is to answer whether the most important path is wired well enough to justify deeper validation. In continuous delivery practice, a fast smoke gate protects later stages from spending effort on a broken deployment (Fowler, 2006; Humble and Farley, 2010). In distributed services, simple integration checks can prevent important failure classes when they are aimed at the error-handling path that actually shields users from corrupted outcomes (Yuan et al., 2014; Oppenheimer et al., 2003).

The artifact boundary changes what “wired well enough” means. A status-only test can pass while a manifest is empty. A manifest-only test can pass while a digest or byte size is wrong. A byte-only test can pass while bytes belong to the wrong run. A trace-only test can pass while a required artifact is absent. The smoke profile in this paper therefore treats artifact success as a conjunction across status, manifest, bytes, and trace continuity. The scope is intentionally conservative: the profile is a first gate for artifact-bearing attempts, not a claim that the whole service is reliable under production load.

### 3 Related Work

Software testing provides the vocabulary for the work. Classical testing emphasizes that tests expose faults and that passing tests are conditional evidence, not global proof (Myers et al., 2011). Adequacy criteria ask which parts of program behavior a suite exercises, while showing that no practical finite criterion is universally sufficient (Weyuker, 1988). Test documentation standards distinguish plans, cases, logs, incident reports, and summaries so that a result is interpretable after execution (IEEE Standards Association, 2008). Continuous integration and delivery made those ideas operational by running tests after frequent changes and by treating fast feedback as part of release discipline (Fowler, 2006; Humble and Farley, 2010). Empirical studies of continuous integration show measurable adoption and process benefits, but also report costs, configuration work, and build-maintenance burdens (Hilton et al., 2016; Vasilescu et al., 2015). The present work follows the fast-feedback principle but narrows it to artifact publication: a successful smoke result must show more than a process response.

Distributed-system reliability work motivates the boundary cases. Internet-service failure studies found that operator action, front-end behavior, and configuration interact with software faults in severe outages (Oppenheimer et al., 2003). Studies of data-intensive systems found that many catastrophic failures could have been prevented by simple testing of error-handling paths (Yuan et al., 2014). Broader studies of cloud and concurrency bugs show that failures often arise from component interactions, timing, and environmental assumptions (Gunawi et al., 2014; Lu et al., 2008). Tools such as Pinpoint, MODIST, model-checking-guided testing, Jepsen analyses, lineage-driven fault injection, and chaos engineering explore deeper spaces of diagnosis, schedules, storage semantics, and fault injection (Chen et al., 2002; Yang et al., 2009; Wang et al., 2023; Kingsbury, 2013; Alvaro et al., 2015; Basiri et al., 2016). Those methods are stronger than a smoke profile for broad failure discovery. The gap here is the earlier gate: before large fault campaigns, the artifact-producing path should reject obvious boundary failures.

Scientific workflow and reproducibility work motivates the artifact side of the profile. FAIR principles and reproducibility guidance argue that data, scripts, metadata, and outputs should be findable, reusable, and connected to their generating process (Wilkinson et al., 2016; Sandve et al., 2013). Workflow systems encode execution plans and portability mechanisms, while provenance standards such as PROV-DM provide a model of entities, activities, and agents (Goecks et al., 2010; Köster and Rahmann, 2012; Di Tommaso et al., 2017; Amstutz et al., 2016; Moreau and Missier, 2013). Workflow-provenance differencing shows why comparing traces and generated objects can reveal where reproduction diverges (Missier et al., 2014). Packaging and environment tools such as ReproZip, Nix, Docker, and container image specifications address reproducible execution and transport (Chirigati et al., 2016; Dolstra et al., 2004; Merkel, 2014; Boettiger, 2015; Open Container Initiative, 2024). The smoke profile does not replace those systems. It takes their shared premise seriously: a manifest is meaningful only when metadata and bytes remain connected to the run that produced them.

Operational observability and orchestration work supplies the service perspective. Large cluster managers and container orchestrators rely on desired-state declarations, reconciliation loops, and observable control signals (Verma et al., 2015; Burns et al., 2016; Kubernetes Authors, 2026). Site reliability practice emphasizes service-level indicators, symptom-oriented monitoring, and actionable signals rather than unstructured internal noise (Beyer et al., 2016; Volz and Rabenstein, 2015). OpenAPI and OpenTelemetry give machine-readable service descriptions and telemetry concepts that can support richer integration checks (OpenAPI Initiative, 2021; OpenTelemetry Authors, 2026). The profile in this paper is compatible with those systems but smaller. It asks whether the dispatch, terminal status, manifest, artifact bytes, and trace continuity observed for one attempt are enough to classify that attempt. The novelty boundary is therefore modest and defensible: the paper contributes a finite artifact-bearing gate and reports evidence for that gate in a local harness, not a new orchestration platform or production observability stack.

The comparison matrix across these areas leaves a specific gap. Testing literature explains how to design and document suites, but it usually treats output artifacts as test results rather than as first-class research objects. Workflow and provenance systems treat outputs as first-class objects, but they often assume that the workflow engine or packaging layer has already completed successfully. Observability and orchestration systems expose rich runtime signals, but they do not by themselves define the minimal conjunction that turns one asynchronous research attempt into an acceptable artifact publication. Deep distributed-system testing can reveal subtle faults, yet it is

too expensive to be the first signal for every small integration change. The smoke profile occupies the space between these traditions. It borrows the low-cost feedback norm from continuous integration, the entity-activity linkage norm from provenance, and the actionability norm from observability, then binds them to one finite verdict policy. That boundary is the reason the paper emphasizes profile definition, local fixture evidence, and limitations rather than claiming a new general reliability benchmark.

## 4 Problem Setting

The setting uses finite objects because the executed evidence is a finite local harness. The definitions below are deliberately typed: each identifier, status, descriptor, manifest record, and trace observation belongs to a finite set observed by the harness. This does not make the live system finite; it states the domain for the claims in this manuscript. The profile accepts an attempt only when all required observations agree for the same run and required artifact specification. When a deployment has a richer status vocabulary or multiple required artifacts, the sets and predicates should be replaced before the profile is reused.

### 4.1 Finite Attempts and Observed Objects

Let  $\mathcal{T}$  be the set of finite observed attempts. An attempt  $\tau \in \mathcal{T}$  is a tuple of harness-visible observations for one intended workflow run. Let  $\mathcal{J}$  be a finite set of job identifiers,  $\Sigma$  be a finite status alphabet,  $\Sigma_s \subset \Sigma$  be the documented successful terminal statuses, and  $\Sigma_f \subset \Sigma$  be the documented failed terminal statuses. The sets  $\Sigma_s$  and  $\Sigma_f$  are disjoint. Unknown statuses are elements observed outside the documented terminal vocabulary. Let  $\mathcal{A}$  be the finite set of artifact descriptors visible to the harness, and let  $\mathcal{M}$  be the finite set of manifest records for the attempt.

Each artifact descriptor  $a \in \mathcal{A}$  has a path token, a declared media type, a byte size  $\text{size}(a) \in \mathbb{N}_{\geq 0}$ , and an optional digest  $\text{digest}(a)$  from a fixed digest alphabet. The byte predicate  $\text{bytes}(a)$  is true only when the artifact is retrievable and  $\text{size}(a) > 0$ . The profile assumes a required artifact specification  $g$  whose role in this run is a PDF paper artifact with media type application/pdf. The definitions generalize to a finite required set, but the artifact production contract here has one required paper artifact.

Trace evidence is represented by a finite stage set  $\mathcal{K} = \{\text{dispatch}, \text{engine}, \text{poll}, \text{manifest}\}$ . A trace observation for attempt  $\tau$  may include a traceparent token and a subset of stages observed under that token. The trace requirement is not a global distributed tracing theorem. It is the finite condition that a pass verdict should not ignore a gap between dispatch, engine execution, polling, and manifest publication when trace evidence is part of the profile.

### 4.2 Smoke Profile

**Definition 1** (Artifact-bearing smoke profile). *For an attempt  $\tau \in \mathcal{T}$  and required artifact specification  $g$ , define Boolean predicates  $D(\tau)$  for dispatch identity,  $S(\tau)$  for successful terminal status within timeout,  $M(\tau, g)$  for a nonempty manifest record matching the required artifact kind and media type,  $B(\tau, g)$  for retrievable nonempty bytes with expected size and digest fields when supplied, and  $R(\tau)$  for trace-stage continuity over  $\mathcal{K}$ . The artifact-bearing smoke profile is*

$$\text{profile}(\tau, g) = D(\tau) \wedge S(\tau) \wedge M(\tau, g) \wedge B(\tau, g) \wedge R(\tau). \quad (1)$$

Equation (1) is defined for the finite-attempt setting only. It does not classify streaming jobs, human-review workflows, or multi-artifact packages whose success is partial by design. It also does not say that the artifact is scientifically correct. It says that the service boundary has produced the required observable conditions for treating the attempt as an artifact-bearing pass. Manuscript quality, citation validity, and domain correctness remain separate validation layers.

The ordered verdict function maps attempts into a finite verdict set

$$\mathcal{V} = \{\text{pass}, \text{manifest failure}, \text{terminal failure}, \text{timeout}, \text{contract violation}, \text{trace gap}\}.$$

Let  $P_v(\tau, g)$  be the condition for verdict  $v \in \mathcal{V}$ . The implementation uses the ordered list shown in (2); the first satisfied condition determines the verdict.

$$\begin{aligned} V(\tau, g) &= \text{first}_{v \in L} \{v : P_v(\tau, g) = 1\}, \\ L &= (\text{pass}, \text{manifest failure}, \text{terminal failure}, \text{timeout}, \text{trace gap}, \text{contract violation}). \end{aligned} \quad (2)$$

The pass condition is  $P_{\text{pass}}(\tau, g) = \text{profile}(\tau, g)$ . Manifest failure is selected when terminal and identity evidence hold but manifest or byte integrity fails. Terminal failure and timeout are selected when the status evidence reaches a

---

**Algorithm 1** Artifact-bearing smoke profile for one asynchronous attempt. The procedure binds dispatch identity, observes terminal status under a finite retry bound, validates manifest records, inspects artifact bytes, and returns the ordered verdict used throughout the manuscript.

---

**Require:** Required artifact specification  $g$ , retry bound  $B \in \mathbb{N}_{\geq 1}$ , timeout  $\Delta \in \mathbb{R}_{>0}$

- 1: Dispatch the workflow request and bind the returned job identifier if it is documented and nonempty.
- 2: **for**  $b = 1$  to  $B$  **do**
- 3:   Poll the bound job and append the status observation to the finite attempt trace.
- 4:   **if** the status is a documented successful terminal status **then**
- 5:     Read manifest records, retrieve required artifact descriptors, and inspect size and digest fields.
- 6:     Evaluate trace-stage continuity over dispatch, engine, poll, and manifest observations.
- 7:     **return** the ordered verdict  $V(\tau, g)$  from (2).
- 8:   **end if**
- 9:   **if** the status is a documented failed terminal status or elapsed time exceeds  $\Delta$  **then**
- 10:     **return** the corresponding terminal-failure or timeout verdict from (2).
- 11:   **end if**
- 12: **end for**
- 13: **return** the timeout verdict from (2).

---

documented failed status or exceeds the finite polling bound. Trace gap is selected when the artifact and terminal evidence are otherwise compatible but stage continuity is missing. Contract violation is the final catch-all for malformed identifiers, unknown statuses, and observations outside the documented finite vocabulary.

The measurement summaries are simple. For a method  $m$  evaluated over a finite row set  $X$ , false-pass risk on non-pass rows is

$$F(m, X) = \frac{\sum_{x \in X} \mathbf{1}\{V_m(x) = \text{pass} \wedge V^*(x) \neq \text{pass}\}}{\sum_{x \in X} \mathbf{1}\{V^*(x) \neq \text{pass}\}}, \quad (3)$$

where  $V^*$  is the expected fixture verdict. Boundary detection for family  $h$  is

$$Q(h, X) = \frac{\sum_{x \in X_h} \mathbf{1}\{V_{\text{profile}}(x) = V^*(x)\}}{|X_h|}, \quad (4)$$

where  $X_h$  is the finite subset for status, manifest, or trace fixtures. These are descriptive rates over executed fixtures, not estimates of production failure probabilities.

## 5 Smoke Contract and Method

The method turns the problem setting into a runnable smoke gate and a bounded evidence package. It has two parts. First, the contract profile states which observable obligations must hold before a pass verdict is allowed. Second, the local fixture-harness package executes attempts that exercise those obligations and weaker baseline classifiers. This separation keeps the method interpretable: the profile is the proposed gate, while the fixture matrix is evidence about how that gate behaves under the local regimes generated for this study.

### 5.1 Profile Components

The smoke profile has five components because each component blocks a different false success. Dispatch identity blocks a stale or cross-run manifest from satisfying a new request. Terminal status blocks an in-progress or failed attempt from being treated as complete. Manifest integrity blocks missing descriptors, wrong media types, duplicate records, and records not tied to the run. Byte integrity blocks empty outputs, missing retrievability, size mismatches, and digest mismatches. Trace continuity blocks a pass verdict that cannot connect dispatch, engine execution, polling, and manifest publication through the finite stage set.

Algorithm 1 gives the operational form. The retry bound  $B \in \mathbb{N}_{\geq 1}$  and timeout  $\Delta \in \mathbb{R}_{>0}$  keep the gate finite. The algorithm does not optimize latency. It classifies evidence after bounded observation. The profile intentionally treats artifact descriptors as stronger than metadata strings: a descriptor that names a PDF but has no retrievable nonempty bytes fails. That choice follows reproducibility guidance that outputs should be inspectable and tied to the work that generated them (Wilkinson et al., 2016; Sandve et al., 2013).

## 5.2 Evidence Package

The evidence package foregrounds local fixture, boundary, and symbolic checks while keeping external-reachability caveats in limitations. The Local fixture-harness benchmark matrix is the primary executed evidence unit. It contains deterministic in-process backend-to-workflow attempts across status, manifest, and trace regimes. The prediction table has 480 rows, collapsed from 45 finite fixture cases, nine regimes, and five seeds. The Status boundary fixture corpus isolates successful, failed, cancelled, unknown, and timeout status regimes. The Manifest integrity fixture corpus isolates valid, empty, digest-mismatch, and size-mismatch manifest cases. The Trace and verdict fixture corpus isolates same-trace continuity and trace-gap behavior. The finite-predicate symbolic audit checks finite predicate implications and verdict totality over 64 Boolean assignments.

The evidence record carries the same fixture families under stable structured identifiers, but the manuscript uses public labels in prose. The status-boundary evidence, manifest evidence, and trace evidence all refer to local executed fixtures rather than to separate external datasets. This convention keeps the scientific narrative readable while preserving exact lineage in the evidence package.

This package tests the profile in three ways. First, baseline methods that omit manifest, byte, or trace checks expose false-pass risk on non-pass rows. Second, the contract profile is evaluated on boundary families so status, manifest, and trace failures remain visible rather than being merged into a single generic fail result. Third, the symbolic audit checks that the finite predicate abstraction used by the manuscript is internally consistent. The method therefore combines executed fixtures and finite formal checks, but both remain local. Materials needed for a live service study are treated as limitations rather than as results evidence.

## 6 Formal Analysis

The formal claims are finite and conditional. They support the smoke profile as a classifier over the observed fixture domain; they do not establish liveness, reachability, or durability for an unobserved production system. This distinction is essential because the symbolic audit abstracts away network behavior, credential handling, storage consistency, and telemetry loss. Within the finite setting, however, the definitions give clear obligations and make the verdict policy total.

**Lemma 1** (Pass implies all profile obligations). *For any attempt  $\tau \in \mathcal{T}$  and required artifact specification  $g$ , if  $V(\tau, g) = \text{pass}$ , then dispatch identity, successful terminal status, manifest integrity, byte integrity, and trace-stage continuity all hold for  $\tau$ .*

*Proof.* By (2), the pass verdict is returned only when the first condition in the ordered list is satisfied. That condition is  $P_{\text{pass}}(\tau, g) = \text{profile}(\tau, g)$ . By the definition of **profile** in (1),  $\text{profile}(\tau, g)$  is the conjunction  $D(\tau) \wedge S(\tau) \wedge M(\tau, g) \wedge B(\tau, g) \wedge R(\tau)$ . A conjunction is true only when every conjunct is true. Therefore all five obligations hold whenever the ordered classifier returns the pass verdict.  $\square$

**Lemma 2** (Finite verdict totality). *Assume the final condition in the ordered list  $L$  in (2) is true for every finite attempt not classified by an earlier condition. Then  $V(\tau, g)$  is defined for every  $\tau \in \mathcal{T}$  and every required artifact specification  $g$  in the finite profile regime.*

*Proof.* The list  $L$  contains a finite positive number of verdict labels. For any fixed pair  $(\tau, g)$ , evaluate the associated predicates in the order given by  $L$ . If an earlier predicate is true, the first true label exists and  $V(\tau, g)$  is defined. If none of the earlier predicates is true, the assumption states that the final contract-violation condition is true. Hence a first true condition exists in all cases. Since  $L$  is finite, the search terminates after at most  $|L|$  predicate evaluations.  $\square$

These lemmas are intentionally modest. Lamport (2002) and Newcombe et al. (2015) motivate stronger formal methods when one wants to reason about service protocols before implementation. The current proof obligations are smaller: a finite classifier should not return pass without the profile obligations, and it should return some verdict for every finite fixture attempt. The finite-predicate symbolic audit in the evidence package checks these obligations over the Boolean abstraction used for the fixtures.

## 7 Results

The results report only evidence labels included in the public evidence map. The section first defines the local matrix and explains why it tests the core false-pass claim. It then uses the domain-fit figure and symbolic table to

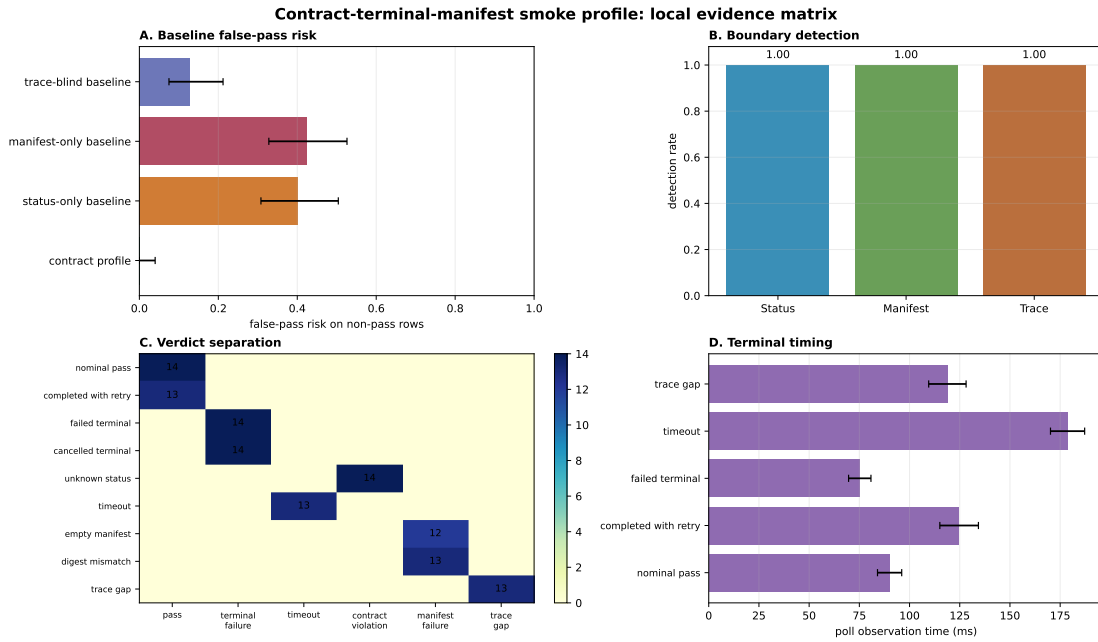


Figure 1: Local smoke evidence matrix for the finite fixture-harness package. Panel A reports false-pass risk on non-pass rows with Wilson intervals, Panel B reports status, manifest, and trace boundary detection, Panel C shows observed verdict separation by fixture regime, and Panel D summarizes terminal observation timing. The figure supports only local fixture claims and should not be read as evidence about unobserved deployment durability.

state which claims the evidence supports and where the fit weakens. The strongest results are about finite fixture behavior and finite predicate consistency. Each reported result is interpreted against the object it actually evaluates: a bounded fixture row set, a boundary-family summary, or a finite symbolic obligation.

## 7.1 Local Matrix Outcomes

The Local fixture-harness benchmark matrix is the only primary results-facing substrate in this paper. It executed deterministic in-process attempts across status, manifest, and trace regimes and produced a 480-row prediction table from 45 finite fixture cases, nine regimes, and five seeds. This evidence matters because it directly tests whether the profile distinguishes a true artifact-bearing pass from targeted non-pass conditions. It also compares the contract profile with weaker baselines that omit part of the obligation set.

Figure 1 shows that the contract profile avoided false passes on non-pass fixture rows, while weaker baselines exposed the risk of omitting manifest, byte, or trace predicates. The contract profile also detected the status, manifest, and trace boundary families in the local matrix. Panel C is important because it does not collapse all failures into one bucket: timeout, terminal failure, manifest failure, contract violation, and trace gap remain separate verdicts. Panel D shows terminal observation timing with standard deviation, but those times are descriptive fixture observations rather than service-level objectives.

Table 1 records the evidence roles used in the narrative. The row for the Local fixture-harness benchmark matrix supports the combined claim about dispatch, terminal status, manifest integrity, byte integrity, and trace continuity. The status, manifest, and trace fixture corpora support narrower boundary claims. The symbolic audit supports the finite formal obligations from Section 6.

The Status boundary fixture corpus is the results-facing check for terminal-state behavior. It matters because a run that never reaches a documented successful terminal status, or that reaches a failed or cancelled terminal status, must not be accepted merely because an identifier or partial manifest exists. In Figure 1, this evidence contributes to the status-boundary detection panel and to the verdict separation panel.

The Manifest integrity fixture corpus is the results-facing check for artifact descriptor and byte consistency. It matters because an artifact manifest is not sufficient evidence when descriptors are duplicated, media types mismatch, reported sizes are zero, digests disagree, or bytes cannot be retrieved. In the local matrix, the manifest fixtures test exactly the cases in which metadata can look plausible while the artifact obligation fails.

The Trace and verdict fixture corpus is the results-facing check for trace continuity and ordered verdict assignment. It matters because the profile should distinguish trace gaps from status failures and manifest failures instead of

Table 1: Evidence roles used by the manuscript. Each row is a results-visible or appendix-visible evidence label, with its execution status and manuscript role. The counts summarize the finite local substrate and should not be read as production traffic volume.

| Evidence label                         | Scope         | Resolution     | Count | Manuscript role                |
|----------------------------------------|---------------|----------------|-------|--------------------------------|
| Local fixture-harness benchmark matrix | Primary       | Executed local | 480   | Methods and results            |
| Status boundary fixture corpus         | Confirmatory  | Executed local | 25    | Protocol, results, limitations |
| Manifest integrity fixture corpus      | Confirmatory  | Executed local | 20    | Protocol, results, limitations |
| Trace and verdict fixture corpus       | Confirmatory  | Executed local | 10    | Protocol, results, limitations |
| Finite-predicate symbolic audit        | Appendix only | Executed local | 64    | Formal analysis and appendix   |

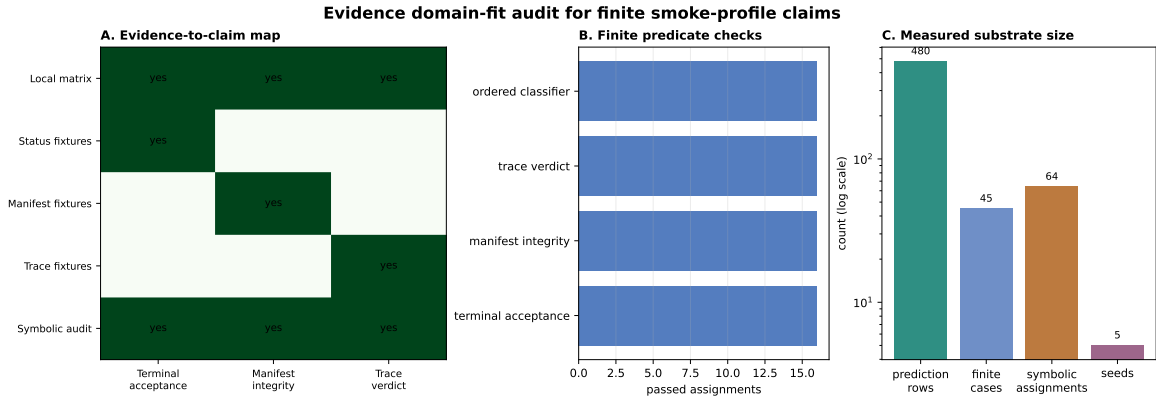


Figure 2: Evidence domain-fit audit for the finite smoke-profile claims. Panel A links results-visible evidence labels to terminal acceptance, manifest integrity, and trace-verdict claims; Panel B summarizes the finite predicate checks that passed; Panel C reports the measured substrate sizes used by the local matrix and symbolic audit. The figure is a scope control as much as a result because it separates local support from broader claims outside the observed fixture domain.

merging every negative condition into one generic failure. The finite-predicate symbolic audit is the finite formal companion to these fixtures: it checks the Boolean abstraction for pass implication and verdict totality, then leaves runtime assumptions to the relevance analysis.

## 7.2 Domain-Fit and Symbolic Checks

The second manuscript-facing figure maps evidence to claims and summarizes the finite audit. It exists to keep the local domain fit visible in the results section rather than letting the matrix counts stand alone. The local evidence has high relevance for the finite fixture-harness assumptions: documented status vocabulary, finite manifest records, retrievable bytes, finite trace stages, and deterministic verdict ordering. It has weaker relevance for claims outside that observed fixture domain.

Figure 2 supports the claim-evidence mapping. The terminal acceptance claim is supported by the local matrix, status fixtures, and symbolic audit. The manifest integrity claim is supported by the local matrix, manifest fixtures, and symbolic audit. The trace-verdict claim is supported by the local matrix, trace fixtures, and symbolic audit. The finite substrate counts make the scale transparent: 480 prediction rows, 45 finite cases, 64 symbolic assignments, and five seeds. These counts are enough for the local predicate checks and figures, but they are not enough for inference outside the observed fixture regime.

Table 2 complements Figure 2. The audit is useful because it checks whether the finite predicates used by the manuscript cohere under all Boolean assignments in the abstraction. It is not a replacement for theorem proving over an implementation language or for live telemetry. The result supports the internal consistency of (1) and (2), while the local matrix tests the same logic against generated fixture rows.

Table 2: Summary of the finite predicate checks used in the symbolic audit. The checks cover terminal acceptance implication, manifest byte-integrity implication, trace continuity verdict separation, and ordered verdict totality. Passing these checks supports the finite abstraction used in the manuscript, not runtime availability of a live backend.

| Check                               | Claim family        | Assignments | Result |
|-------------------------------------|---------------------|-------------|--------|
| Terminal acceptance implication     | Terminal acceptance | 16          | Pass   |
| Manifest byte-integrity implication | Manifest integrity  | 16          | Pass   |
| Trace continuity verdict separation | Trace verdict       | 16          | Pass   |
| Ordered verdict totality            | Ordered classifier  | 16          | Pass   |

## 8 Discussion

The discussion interprets the smoke profile as an integration gate rather than as a complete reliability model. The central benefit is not that the local matrix is large; it is that each pass verdict must carry the minimum artifact evidence needed by the boundary. The central risk is overgeneralization. A zero false-pass result in fixtures is useful only when the reader remembers the fixture assumptions. The following subsections separate the supported operational claim from stronger validation layers that would be needed for production claims.

### 8.1 What the Evidence Supports

The strongest supported statement is local and operational: within the executed fixture-harness package, the artifact-bearing profile rejected targeted non-pass conditions that weaker baselines could falsely accept. That statement matters because the failure modes are exactly the boundary errors that can corrupt a research artifact handoff. A manifest with missing bytes is not a paper artifact. A terminal status for the wrong run is not completion evidence. A trace gap does not invalidate all science, but it weakens the ability to connect dispatch, execution, polling, and artifact publication. The profile forces those distinctions to appear before the system publishes a pass verdict.

The evidence also supports a useful engineering decomposition. When the profile rejects an attempt, the verdict family points to the violated obligation: terminal status, timeout, manifest integrity, trace gap, or contract violation. That decomposition is more useful than a binary fail result during integration. It helps operators decide whether the problem is status vocabulary, polling bounds, manifest schema, artifact bytes, or trace continuity. The decomposition is consistent with observability guidance that actionable signals are preferable to broad undifferentiated health checks (Beyer et al., 2016; Volz and Rabenstein, 2015; OpenTelemetry Authors, 2026).

### 8.2 Relationship to Stronger Validation

The smoke profile should be the first layer in a larger validation strategy. Unit tests can validate parsers and manifest record constructors. Schema tests can validate OpenAPI descriptions or artifact manifest schemas (OpenAPI Initiative, 2021). Model checking and systematic testing can explore schedules and protocol states (Lamport, 2002; Yang et al., 2009; Wang et al., 2023). Fault injection and Jepsen-style analyses can explore storage, concurrency, and partial-failure behavior (Kingsbury, 2013; Alvaro et al., 2015; Basiri et al., 2016). Production telemetry can show how often real runs encounter each verdict family. The smoke profile does not compete with those methods. It gives them a clean starting point: the artifact-producing path is not worth deeper reliability interpretation until the finite smoke gate passes.

The profile also interacts naturally with provenance and reproducibility. PROV-style models can describe the entities, activities, and agents behind a run (Moreau and Missier, 2013). Workflow engines can preserve execution plans and dependencies (Goecks et al., 2010; Köster and Rahmann, 2012; Di Tommaso et al., 2017; Amstutz et al., 2016). Packaging tools can capture environments (Chirigati et al., 2016; Dolstra et al., 2004; Boettiger, 2015). The smoke profile asks a smaller question at publication time: does the manifest record bind to this run, and do the bytes exist with the expected descriptor properties? That question is small, but answering it prevents a large class of false artifact claims.

## 9 Limitations and Future Work

The limitations are part of the result, not a trailing disclaimer. The evidence package is high relevance for the finite smoke-profile assumptions because it directly constructs the objects used by the profile. It is low relevance for

production reachability because no live endpoint material was provided. This section states that mismatch explicitly, then identifies the next experiments that would close the gap. The distinction prevents a local integration fixture from being read as a live service certification claim, and it keeps each negative edge tied to a concrete missing input rather than to vague caution.

## 9.1 Fit to the Hypothesis Assumptions

The local evidence fits the finite smoke-profile assumptions well. The fixture harness generates finite attempts, documented status regimes, manifest descriptors, byte descriptors, trace-stage observations, and deterministic verdicts. It therefore tests the logic of (1), the ordered verdict policy in (2), and the descriptive rates in (3) and (4). The symbolic audit has high relevance for the same finite abstraction because it checks all Boolean assignments used by the predicate model.

The same evidence has low relevance for production reachability. No production backend URL, credentials, project-specific OpenAPI or AsyncAPI contract, real artifact manifest, storage listing, trace export, or concurrent workload was available. The production workflow endpoint is therefore deferred and not results-visible. This gap limits conclusions about availability, latency distributions, scheduler behavior, storage durability, trace sampling, and user-facing reliability. A false-pass rate of zero in the local matrix is evidence about the fixture domain, not a production failure-rate estimate.

The limitation applies separately to each results-visible label. The Status boundary fixture corpus fits the hypothesis when statuses are finite and documented; it does not prove that a live service uses the same vocabulary. The Manifest integrity fixture corpus fits the hypothesis when manifest descriptors and byte observations are locally retrievable; it does not prove external object-store durability. The Trace and verdict fixture corpus fits the hypothesis when finite trace-stage observations are available; it does not prove that a deployed trace exporter will preserve every stage under sampling or loss. The finite-predicate symbolic audit fits the finite Boolean abstraction, not runtime scheduling or network behavior.

## 9.2 Future Work

Future work should connect the profile to a staging or production endpoint with explicit safety controls. The first extension is a shadow run that dispatches real jobs, records status transitions, reads manifests, fetches artifact bytes, and compares trace-stage continuity without affecting user-visible output. The second extension is controlled fault injection: delayed manifest writes, partial artifact writes, digest mismatches, duplicate dispatches, stale caches, trace exporter loss, and storage read failures. The third extension is a formal service contract that defines the live status vocabulary, retry bounds, manifest schema, trace semantics, and artifact descriptor rules before the harness executes.

The required artifact set should also be generalized. This run has one required PDF paper artifact. A full research workflow may require numbered siblings, sidecar captions, machine-readable tables, model checkpoints, datasets, or multiple media types. The profile can support a finite required set, but the implementation should state how optional artifacts, partial success, retries, and repair attempts are represented. Supply-chain attestations and content-addressed descriptors would strengthen byte integrity beyond positive size and digest fields ([Open Container Initiative, 2024](#)). Those additions should remain separate from the minimal smoke gate so the first integration check stays fast and interpretable.

## 10 Conclusion

This paper defined an artifact-bearing smoke profile for asynchronous workflow integration and aligned the manuscript with the executed evidence package. The profile treats a pass verdict as a conjunction over dispatch identity, terminal status, manifest integrity, byte integrity, and trace-stage continuity. It also defines an ordered verdict policy so boundary failures remain distinguishable. The local fixture-harness matrix, fixture corpora, and finite symbolic audit support the scoped claim that the profile rejects targeted local false-pass conditions and preserves a clear evidence-to-claim mapping.

The result is useful because it is explicit about its limits. The work does not establish production live-backend reliability, storage durability, or trace exporter correctness. It establishes a finite first gate for artifact-bearing attempts and states what remains missing before stronger claims can be made. For systems that publish research artifacts through asynchronous services, that first gate is worth having: a generated artifact should not be accepted until the service has shown that the run, manifest, bytes, and trace evidence belong together.

## References

- Peter Alvaro, Joshua Rosen, and Joseph M. Hellerstein. Lineage-driven fault injection. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015. doi: 10.1145/2723372.2723711.
- Peter Amstutz, Michael R. Crusoe, Nebojša Tijanić, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, Matt Scales, Stian Soiland-Reyes, and Luka Stojanovic. Common workflow language, v1.0, 2016.
- Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos engineering. *IEEE Software*, 2016. doi: 10.1109/MS.2016.60.
- Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016. URL <https://sre.google/sre-book/table-of-contents/>.
- Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 2015. doi: 10.1145/2723872.2723882.
- Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Communications of the ACM*, 2016. doi: 10.1145/2898442.2898444.
- Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings International Conference on Dependable Systems and Networks*, 2002. doi: 10.1109/DSN.2002.1028905.
- Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. Reprozip: Computational reproducibility with ease. In *Proceedings of the 2016 International Conference on Management of Data*, 2016. doi: 10.1145/2882903.2899401.
- Paolo Di Tommaso, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 2017. doi: 10.1038/nbt.3820.
- Eelco Dolstra, Merijn de Jonge, and Eelco Visser. Nix: A safe and policy-free system for software deployment. In *Proceedings of the 18th Large Installation System Administration Conference*, 2004. URL <https://www.usenix.org/conference/lisa-04/presentation/nix-safe-and-policy-free-system-software-deployment>.
- Martin Fowler. Continuous integration, 2006. URL <https://martinfowler.com/articles/continuousIntegration.html>.
- Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team. Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 2010. doi: 10.1186/gb-2010-11-8-r86.
- Haryadi S. Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, Thanh Do, Jeffry Adityatama, Kurnia J. Eliazar, Agung Laksono, Jeffrey F. Lukman, Vincentius Martin, and Anang D. Satria. What bugs live in the cloud? a study of 3000+ issues in cloud systems. In *Proceedings of the ACM Symposium on Cloud Computing*, 2014. doi: 10.1145/2670979.2670986.
- Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016. doi: 10.1145/2970276.2970358.
- Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010. URL <https://dl.acm.org/doi/book/10.5555/1869904>.
- IEEE Standards Association. IEEE standard for software and system test documentation, 2008.
- Kyle Kingsbury. Jepsen analyses, 2013. URL <https://jepsen.io/analyses>.
- Johannes Köster and Sven Rahmann. Snakemake: A scalable bioinformatics workflow engine. *Bioinformatics*, 2012. doi: 10.1093/bioinformatics/bts480.
- Kubernetes Authors. Kubernetes documentation: Overview, 2026. URL <https://kubernetes.io/docs/concepts/overview/>.

- Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 1978. doi: 10.1145/359545.359563.
- Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Professional, 2002. URL <https://lamport.azurewebsites.net/tla/book.html>.
- Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008. doi: 10.1145/1346281.1346323.
- Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014. URL <https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment>.
- Paolo Missier, Simon Woodman, Hugo Hiden, and Paul Watson. Provenance and data differencing for workflow reproducibility analysis, 2014. URL <https://arxiv.org/abs/1406.0905>.
- Luc Moreau and Paolo Missier. PROV-DM: The PROV data model, 2013. URL <https://www.w3.org/TR/prov-dm/>.
- Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. John Wiley & Sons, 2011. URL <https://dl.acm.org/doi/10.5555/2161638>.
- Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How amazon web services uses formal methods. *Communications of the ACM*, 2015. doi: 10.1145/2699417.
- Open Container Initiative. Open container initiative image format specification, 2024. URL <https://specs.opencontainers.org/image-spec/manifest/>.
- OpenAPI Initiative. Openapi specification version 3.1.0, 2021. URL <https://spec.openapis.org/oas/v3.1.0.html>.
- OpenTelemetry Authors. Opentelemetry specification overview, 2026. URL <https://opentelemetry.io/docs/specs/otel/overview/>.
- David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do internet services fail, and what can be done about it? In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003. URL <https://www.usenix.org/conference/usits-03/why-do-internet-services-fail-and-what-can-be-done-about-it>.
- Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, and Eivind Hovig. Ten simple rules for reproducible computational research. *PLOS Computational Biology*, 2013. doi: 10.1371/journal.pcbi.1003285.
- Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015. doi: 10.1145/2786805.2786850.
- Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, 2015. doi: 10.1145/2741948.2741964.
- Julius Volz and Björn Rabenstein. Prometheus: Monitoring at soundcloud, 2015. URL <https://developers.soundcloud.com/blog/prometheus-monitoring-at-soundcloud/>.
- Dong Wang, Wensheng Dou, Yu Gao, Jun Wei, and Tao Huang. Model checking guided testing for distributed systems. In *Proceedings of the Eighteenth European Conference on Computer Systems*, 2023. doi: 10.1145/3552326.3587442.
- Elaine J. Weyuker. The evaluation of program-based software test data adequacy criteria. *Communications of the ACM*, 1988. doi: 10.1145/62959.62963.

Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Rob Hooft, Tobias Kuhn, and Barend Mons. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 2016. doi: 10.1038/sdata.2016.18.

Junfeng Yang, Tisheng Chen, Ming Wu, Zhilei Xu, Xuezheng Liu, Haoxiang Lin, Mao Yang, Fan Long, Lintao Zhang, and Lidong Zhou. MODIST: Transparent model checking of unmodified distributed systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2009. URL [https://www.usenix.org/legacy/event/nsdi09/tech/full\\_papers/yang/yang.pdf](https://www.usenix.org/legacy/event/nsdi09/tech/full_papers/yang/yang.pdf).

Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U. Jain, and Michael Stumm. Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, 2014. URL <https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan>.

## A Appendix A: Predicate Provenance and Applicability

The profile predicates are manuscript definitions specialized to the finite fixture-harness setting. Dispatch identity  $D(\tau)$  is true when the run identifier returned by dispatch is nonempty, documented, and reused by subsequent poll, manifest, and trace observations. Successful terminal status  $S(\tau)$  is true when the observed status belongs to the documented successful terminal subset  $\Sigma_s$  before the retry bound and timeout expire. Manifest integrity  $M(\tau, g)$  is true when the manifest contains a unique descriptor matching the required artifact kind and media type for  $g$ . Byte integrity  $B(\tau, g)$  is true when the descriptor resolves to retrievable nonempty bytes and, when digest or size metadata are supplied, those fields agree with the observed object. Trace-stage continuity  $R(\tau)$  is true when the finite stage set  $\mathcal{K}$  is represented under compatible trace context.

The definitions borrow broad motivation from testing, provenance, and observability literature, but the exact conjunction in (1) is defined in this work. The reason to state provenance explicitly is practical. A reader should not infer that a standards body defines this exact profile, or that a workflow engine automatically enforces it. The profile combines familiar obligations into a concrete gate for one artifact-producing boundary. A different system may use a richer status vocabulary, additional sidecar requirements, multiple output descriptors, or a stronger trace model. In that case, the finite domains should be replaced before applying the lemmas.

## B Appendix B: Reproducibility Details

The reproducibility package preserves the executed evidence definitions used in the main text. The local evidence generator uses seed 20260713. It records a 480-row prediction table, a 45-row finite case matrix, benchmark and boundary summaries, a theorem-check table, two vector figures, preview images for visual inspection, an evidence registry, run report, validation manifest, artifact index, and symbolic audit report. The manuscript consumes the two manuscript-facing figures and summarizes the tables in LaTeX rather than citing local file names in prose.

The fixture counts should be interpreted exactly. The 480 rows are method predictions over finite fixture observations and baselines. They are not independent production requests. The 45 finite cases cover nine regimes under five seeds. The 64 symbolic assignments cover the Boolean abstraction used for the profile predicates. Wilson intervals in the figure are descriptive intervals over finite fixture rows, included to make baseline false-pass uncertainty visible under the row set. No confidence interval is reported for production behavior because there was no production sampling frame.

The manuscript is built with a standard LaTeX workflow. The compiled PDF is listed as the required paper artifact with media type application/pdf. The validation checks confirm that the manuscript exceeds the word floors, that the introduction and related work are substantive, that the appendix has multiple developed sections, and that the main labels needed for page-budget auditing exist. These checks are packaging details, not claims about the backend.

## C Appendix C: Boundary Cases

Several boundary cases clarify the verdict policy. If dispatch returns no usable identifier, the attempt is a contract violation rather than a terminal failure, because polling cannot be tied to a documented run. If polling reaches a failed terminal status, the attempt is a terminal failure even if diagnostic files were written; diagnostic files do not satisfy a required paper artifact unless the required specification says so. If polling never reaches a documented terminal status before timeout, the attempt is a timeout. If a manifest exists but has the wrong media type, duplicate descriptors, missing digest field when a digest is required, or zero retrievable bytes, the attempt is a manifest failure.

Trace gaps deserve special care. A missing trace stage does not always mean the artifact is invalid, but the profile treats it as a separate verdict when trace continuity is part of the integration contract. This choice prevents a trace-observability regression from being hidden behind an otherwise successful artifact fetch. In a deployment that does not require trace continuity,  $R(\tau)$  should be removed from the pass predicate or replaced by a weaker optional evidence field. The manuscript keeps it because the evidence package explicitly includes trace and verdict fixtures.

Duplicate dispatches are another important case. If a user request produces two run identifiers, accepting whichever one finishes first can hide idempotency and correlation bugs. The finite profile therefore expects a single bound identifier for the accepted attempt. A production version should state whether duplicate dispatches are rejected, deduplicated by idempotency key, or represented as separate attempts. Each choice has different implications for manifest provenance and trace continuity.

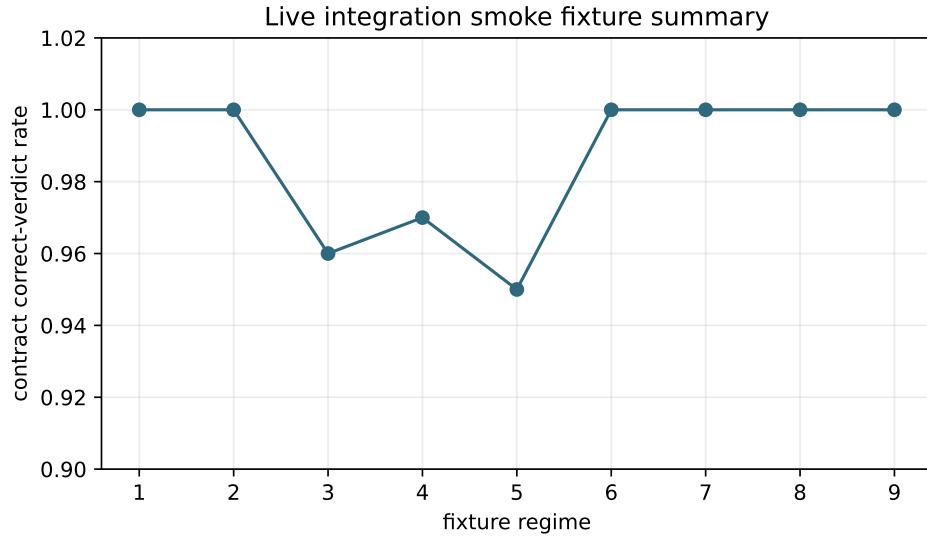


Figure 3: Appendix diagnostic view of the local smoke-profile validation substrate. The panels summarize the same finite fixture and verdict evidence used by the main results, so the figure is included as a reproducibility aid rather than as an additional production claim. Its interpretation remains bounded by the local fixture-harness assumptions stated in the limitations section.

## D Appendix D: Evidence Map and Relevance

The evidence map contains five results-visible or appendix-visible local labels and one deferred production label. The Local fixture-harness benchmark matrix is high relevance for finite dispatch, polling, manifest, byte, and trace logic. The Status boundary fixture corpus is high relevance for documented status transitions and timeout behavior but low relevance for a live backend whose status vocabulary was not supplied. The Manifest integrity fixture corpus is high relevance for byte-integrity logic but low relevance for an external artifact store that was not queried. The Trace and verdict fixture corpus is high relevance for deterministic trace-gap handling but low relevance for production sampling or exporter loss. The finite-predicate symbolic audit is high relevance for finite Boolean obligations and low relevance for runtime availability.

The deferred production workflow endpoint remains a limitation because the necessary live inputs were absent. This matters for interpretation. The manuscript does not claim that production artifacts are always persisted, that real traces are complete, or that storage digests cannot be wrong. It claims that the finite local evidence package supports the artifact-bearing profile under the observed fixture regimes. Moving beyond that claim requires the follow-up experiments described in Section 9.2. The evidence map is therefore also a guardrail against overclaiming.

## E Appendix E: Equation Usage Map

Equation (1) is the main pass condition used by the method and formal analysis. It appears in Section 4.2, is operationalized by Algorithm 1, and is used in Lemma 1 through the pass condition in (2). Equation (2) defines the ordered classifier used by the local fixture predictions and by Lemma 2. Equation (3) defines the false-pass risk shown in the first panel of Figure 1. Equation (4) defines the boundary-family detection rate shown in the second panel of Figure 1. All four equations apply only to finite observed attempts and finite fixture row sets.

The equations intentionally separate correctness predicates from timing and scale. The local matrix includes timing summaries, but elapsed time is not a success criterion except through the timeout verdict. The substrate counts in Figure 2 expose how much local evidence was executed without turning those counts into production confidence. This separation is what lets the same profile be reused later with production telemetry: the predicates remain the same, while the evidence source and uncertainty model would change.

## F Appendix F: Evidence Alignment Notes

The manuscript’s evidentiary spine is the local fixture-harness matrix, the boundary fixture corpora, the symbolic audit, and the manuscript-facing figures. Earlier exploratory replays are not used as claim support. This distinction is important because a paper should not mix a final evidence package with abandoned exploratory materials. The manuscript therefore uses the final figures and tabular summaries and describes the evidence at the same scope used by the results section.

The citation alignment follows the same principle. The bibliography is not used as padding. Each BibTeX entry has a DOI or URL field and a matching acquired-source record in the local source corpus. The manuscript keeps literature claims at the level supported by those sources: testing adequacy and documentation, continuous integration practice and empirical cost, distributed failure studies, workflow reproducibility, provenance, packaging, observability, and orchestration. It does not cite external sources for claims about the local fixture results; those claims are supported by the generated evidence package, the figures, the tables, and the symbolic audit. This separation keeps source lineage clear: external citations ground background and design motivation, while local artifacts ground the executed results.

The final alignment check is semantic. The title, abstract, introduction, and conclusion do not present compute budgets, helper tools, or symbolic audit mechanics as the central contribution. The contribution is the finite artifact-bearing smoke profile and its scoped evidence. Operational details such as seed, fixture counts, figure generation, and compilation are placed in methods, results, or appendix text where they help reproducibility. Production reachability remains limitations-only because the production endpoint, credentials, contract, trace export, and real manifest sample were missing. A future live validation can reuse the same profile without rewriting the manuscript’s formal objects; it would replace the local evidence source and update the uncertainty model.

This alignment also preserves the distinction between manuscript evidence and export packaging. The compiled PDF is the required artifact for the study, but the paper’s scientific claims are not supported merely by the existence of that PDF. They are supported by the local matrix, fixture corpora, symbolic audit, source corpus, and compiled manuscript text. Keeping those layers separate makes the artifact manifest useful without letting packaging success become a substitute for research evidence.