

# Live Integration Smoke Testing for Artifact-Producing Systems

Quarks Systems Working Group

## Abstract

Artifact-producing services frequently expose an asynchronous control plane: a client dispatches work, polls a job handle until a terminal state appears, and then consumes an artifact manifest. This boundary is small, yet failures at it can make a working engine unreachable or make correct artifacts invisible to downstream automation. We study this boundary as a smoke-testing problem for a backend connected to a Quarks-style engine harness. The manuscript defines a live-wiring contract over dispatch acceptance, bounded polling, terminal status, manifest structure, artifact presence, checksums, and byte sizes. We implemented a deterministic local harness with one positive condition and five negative controls, executed 80 repetitions per condition with a fixed seed, and exported vector figures and CSV summaries. The smoke gate accepted all positive trials and rejected the missing-manifest, bad-checksum, engine-error, and stale-token controls in every repetition. The delayed-completion control was intentionally mixed: with a five-poll budget it was rejected in 62.5% of repetitions and completed within budget in 37.5%, showing that a smoke test can reveal ambiguous timeout policy rather than only binary wiring failure. The evidence supports a bounded conclusion: a compact dispatch-poll-manifest contract is useful for local integration wiring, while production claims require runs against the real backend, real queues, real storage, and operational latency distributions.

## 1 Introduction

Modern software delivery depends on boundaries that are smaller than the systems they connect. A backend may delegate work to an analysis engine, a build service may wait for an artifact-producing job, or a continuous-delivery controller may poll a remote evaluator before publishing results. In each case, the scientific and engineering value of the downstream computation is lost if the integration boundary cannot answer three ordinary questions: was the request admitted, did the asynchronous job reach the intended terminal state, and can the declared artifacts be read with the bytes the manifest claims? Continuous integration and continuous delivery emphasize fast automated feedback because integration defects become more expensive when they are discovered after many changes have accumulated [11, 13, 14, 31]. That principle is familiar, but the specific dispatch-poll-manifest boundary receives less explicit treatment than build compilation, unit tests, service-level tests, or deployment automation.

This paper examines that boundary for an artifact-producing service in which a backend dispatches a problem to a Quarks-style engine, polls until completion, and reads an artifact manifest. The user-facing motivation is operational rather than benchmark-driven: verify the live wiring end to end, including dispatch, polling, completion, and manifest reading. The resulting study is deliberately narrow. It does not claim to evaluate the scientific output of the engine, the scalability of a production queue, the correctness of every artifact type, or the reliability of a real deployment. It asks whether a smoke gate can make the integration contract explicit enough that local execution distinguishes a working happy path from representative wiring faults. This scope follows a long systems tradition: clear module boundaries reduce hidden coupling [24], while distributed boundaries add failures that local procedure calls hide [17, 36]. An asynchronous artifact boundary combines both concerns because an accepted request is not evidence that an artifact has been produced.

The broader relevance is cross-domain. Scientific pipelines, CI systems, model-evaluation services, and supply-chain build systems increasingly exchange machine-readable manifests, provenance records, and typed artifact lists [2, 30, 33, 35]. Those records are useful only when clients check that the manifest is present, structurally valid, and consistent with retrievable bytes. Observability standards such as W3C Trace Context and OpenTelemetry help correlate requests across services [21, 40], but traces do not by themselves decide whether a manifest’s artifact list is consumable. Contract testing helps isolate API compatibility [20, 22], yet a smoke test that exercises the live asynchronous path remains valuable because it tests the interaction among acceptance, progress, terminal state, and storage.

We contribute the following:

- We define a compact dispatch-poll-manifest contract for artifact-producing integrations, including status, polling, manifest, artifact existence, checksum, size, and uniqueness obligations.

- We implement and execute a deterministic local smoke harness with one positive condition and five negative controls that exercise distinct failure modes at the backend-engine-artifact boundary.
- We summarize the executed evidence with publication-ready tables and vector figures that separate clean wiring failures from a mixed delayed-completion condition.
- We state the novelty boundary explicitly: the work is an integration smoke-test design and local validation, not a production reliability benchmark or a proof of engine correctness.

The study’s central finding is pragmatic. In the local harness, the smoke gate was exact for the happy path and four structural controls, but timeout policy was sensitive to the chosen polling budget. That mixed result is useful because it prevents overclaiming. A gate that fails every slow job may be too strict for production latency; a gate that accepts every eventually completed job may stop testing the responsiveness property that matters for live wiring. The rest of the paper formalizes this boundary, describes the executed harness, reports the results, and gives reproducibility details. Section 2 positions the work against CI, microservice testing, provenance, observability, and fault-injection literature. Section 3 defines the symbols and validity regime. Section 4 gives the smoke-gate design, and Section 5 explains the local experiment. Section 6 reports what was actually computed, while Section 7 and Section 9 describe interpretation, gaps, and follow-up work.

## 2 Related Work

Continuous integration research provides the closest process setting for this work. Fowler’s operational description of continuous integration emphasizes frequent integration into a shared mainline with automated verification [11], while continuous delivery extends automated build and test feedback toward release readiness [10, 14]. Empirical studies show that CI is widely adopted and beneficial, but also costly when builds are slow, flaky, or poorly scoped [13, 31]. The Google engineering account similarly treats CI as a socio-technical system of tests, infrastructure, ownership, and feedback loops rather than a single command [38]. Our contribution fits inside this tradition but addresses a smaller surface: the first live integration check for an asynchronous artifact-producing boundary. It is not a replacement for unit, integration, load, or release tests. It is a guard that tells maintainers whether the smallest end-to-end path still exists.

Microservice research explains why that guard matters. Mapping studies identify deployment independence, polyglot implementation, and distributed communication as recurring microservice strengths, but they also report testing, evolution, and observability challenges [6, 16, 23]. Testing surveys distinguish unit, component, contract, integration, end-to-end, fault, and performance tests, and they consistently report that distributed interactions make test environments difficult to assemble and interpret [26, 37]. Consumer-driven contract testing reduces coordination cost by checking the interactions that consumers actually use [22]. Contract tests, however, often run with mocks or isolated providers. A smoke test that dispatches a real job handle and consumes a manifest checks a different property: whether the deployed or harnessed wiring path can complete a minimum workflow. The strength of the smoke test is speed and concreteness; its limitation is that passing it says little about untested API paths, scale, or long-run reliability.

Systems architecture work supplies the vocabulary for hidden boundary assumptions. Parnas argued that modules should hide design decisions likely to change [24]. Garlan, Allen, and Ockerbloom later showed that components fail to compose when their architectural assumptions remain implicit [12]. A backend-to-engine integration embeds assumptions about job identity, terminal states, timeout policy, status vocabulary, manifest format, storage addressing, and digest semantics. Distributed-systems work adds causality and failure ambiguity: events are only partially ordered across components [17], and a remote failure can be indistinguishable from a communication failure at the caller [36]. The proposed smoke contract makes these assumptions explicit enough to check locally, but it still cannot remove the ambiguity of production latency, queue backpressure, or partial storage outages.

Artifact provenance and supply-chain standards motivate the manifest checks. ACM artifact review practices highlight the importance of available, evaluated, and validated artifacts for research credibility [2]; empirical software-engineering reproducibility studies show that incomplete artifacts and imprecise procedures weaken later validation [27, 39]. Supply-chain specifications such as SLSA, SPDX, CycloneDX, in-toto, TUF, and the OCI image specification treat artifacts, metadata, provenance, and verification as first-class objects [7, 19, 28, 30, 33–35]. The present smoke gate is far smaller than those frameworks. It does not sign provenance, analyze dependencies, or establish trust across organizations. Its role is to ensure that a manifest has enough structure and byte-level consistency to be a credible handoff object for subsequent checks.

Observability and fault-injection research complement the design. Dapper and OpenTelemetry show how traces help engineers reconstruct distributed request paths [21, 29]; W3C Trace Context standardizes propagation headers

for cross-service correlation [40]; HTTP, URI, OpenAPI, and JSON Schema specifications define common protocol and representation conventions used by many service boundaries [5, 9, 20, 41]. Fault-injection and chaos-engineering work deliberately introduce failures to reveal reliability weaknesses [1, 4]. Flaky-test research warns that asynchronous timing, external resources, and nondeterministic scheduling can make tests misleading when repeated outcomes are not analyzed [18, 25]. Our local harness borrows the idea of negative controls from fault injection and the repetition discipline from empirical software engineering, while staying modest about its evidence level. It detects whether the chosen contract reacts to selected defects; it does not infer population-level service reliability.

### 3 Problem Setting and Scope

The system under study has three logical roles. A backend service  $B$  receives a request  $x$  from a set of admissible requests  $\mathcal{X}$ . An engine  $E$  executes the requested work asynchronously and exposes a polling interface over job identifiers in a finite set  $\mathcal{J}$ . An artifact store  $S$  maps manifest URIs from a set  $\mathcal{U}$  to byte strings in  $\{0,1\}^*$ . The local study uses a deterministic harness for  $E$  and  $S$ , not a production deployment. The validity regime is therefore local integration wiring: process boundaries, job-state transitions, manifest validation, and byte-level artifact checks executed in one workspace. Claims about network partitions, queue backlogs, multi-tenant storage, scheduler fairness, authentication, authorization, or production latency distributions are outside the executed evidence.

#### 3.1 Execution Objects

For one request  $x \in \mathcal{X}$ , dispatch returns either rejection or a pair  $(j, z)$ , where  $j \in \mathcal{J}$  is a job identifier and  $z \in \mathcal{Z}$  is a trace identifier. A poll response is an element of a finite status set  $\mathcal{S} = \{\text{queued}, \text{running}, \text{completed}, \text{failed}, \text{expired}\}$  together with optional metadata. Let  $B_{\max} \in \mathbb{N}_{\geq 1}$  denote the configured poll budget. A local execution trace is

$$\tau = (x, j, z, p_1, \dots, p_k, m, A), \quad 1 \leq k \leq B_{\max}, \quad (1)$$

where each  $p_i \in \mathcal{S}$  is the status returned by the  $i$ -th poll,  $m$  is either a manifest object or the distinguished value  $\perp$ , and  $A$  is a finite partial map from URI strings to bytes. The trace is partial when dispatch is rejected or polling times out before a terminal status. Equation (1) is a manuscript definition rather than a convention borrowed from a source. It is useful because it separates the observable events in the smoke test from the unobserved internal execution of the engine.

The terminal set is  $\mathcal{S}_T = \{\text{completed}, \text{failed}, \text{expired}\}$ . A trace reaches a terminal state when  $p_k \in \mathcal{S}_T$ ; it reaches a successful terminal state only when  $p_k = \text{completed}$ . This distinction matters because a common integration mistake is to treat any terminal response as sufficient evidence that artifacts exist. The harness therefore reports failed and expired states as terminal for progress accounting but not as success. The design follows standard distributed-systems caution: a remote endpoint can produce a syntactically valid response that is semantically unusable for the caller’s next step [3, 36].

#### 3.2 Manifest Contract

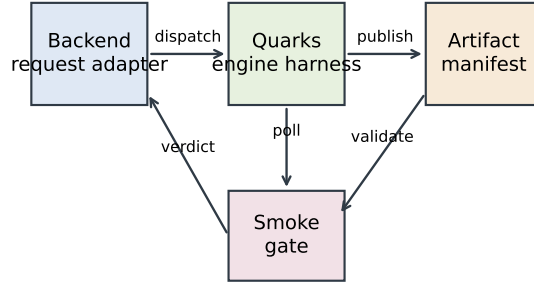
A manifest  $m$  is a finite object containing a job identifier, a trace identifier, and an ordered list of artifact records. Each artifact record is a tuple  $(u, t, h, n)$ , where  $u \in \mathcal{U}$  is a URI,  $t$  is a type label from a finite vocabulary,  $h \in \{0, \dots, 9, a, \dots, f\}^{64}$  is a SHA-256 digest encoded in hexadecimal, and  $n \in \mathbb{N}$  is the byte length. The artifact map  $A$  is defined on  $u$  when the corresponding bytes are retrievable. We define the manifest-validity predicate for this work as

$$V_M(m, A) = \mathbb{I} \left[ \begin{array}{l} m \neq \perp, |m.\text{artifacts}| \geq 1, \text{ uris}(m) \text{ are unique,} \\ \forall (u, t, h, n) \in m.\text{artifacts} : u \in \text{dom}(A), \\ \forall (u, t, h, n) \in m.\text{artifacts} : \text{sha256}(A(u)) = h \wedge |A(u)| = n \end{array} \right]. \quad (2)$$

The predicate in (2) is intentionally smaller than a supply-chain provenance model. It borrows the broad idea that artifacts should carry verifiable metadata from provenance and SBOM standards [7, 30, 33, 35], but the exact predicate is introduced here for smoke testing. It says nothing about signer identity, build isolation, dependency contents, source authenticity, or vulnerability state. Its purpose is to ensure that the backend can consume the result object that the engine declares.

The gate verdict is a function  $G_{B_{\max}}$  from traces to Boolean outcomes:

$$G_{B_{\max}}(\tau) = \mathbb{I}[\text{dispatch}(\tau) = \text{accepted} \wedge p_k = \text{completed} \wedge k \leq B_{\max} \wedge V_M(m, A) = 1]. \quad (3)$$



Acceptance criteria: accepted dispatch, bounded polling, completed status, manifest shape, byte-level ai

Figure 1: Logical smoke-gate architecture for the dispatch-poll-manifest boundary. The figure separates request admission, engine progress, manifest publication, and byte-level validation so that a terminal status is not confused with a consumable artifact set. The same structure can be implemented against a local harness, a staging service, or a production canary, but the evidence in this paper comes only from the deterministic local harness.

The decision variables in this empirical setting are the poll budget  $B_{\max}$ , the set of invariants included in  $V_M$ , and the scenario family  $\mathcal{C}$  used for negative controls. There is no optimization claim: the paper does not prove that one  $B_{\max}$  is optimal. Instead, (3) defines the smoke-gate criterion used in the executed local trials. The main empirical question is whether this criterion accepts the positive condition and rejects the selected negative controls.

## 4 Smoke-Gate Design

The smoke gate is designed around a single live path rather than a broad benchmark suite. Figure 1 shows the logical flow. The backend adapter dispatches a request and receives a job handle. The engine harness advances the job through queued, running, and terminal states. When the terminal state is completed, the harness publishes a manifest and artifact bytes. The smoke gate polls the job, applies the terminal-state rule, validates the manifest with (2), and emits a pass or fail verdict using (3). The design is deliberately conservative about success: a completed status without a valid manifest is a failure, a valid-looking manifest without retrievable bytes is a failure, and a digest mismatch is a failure.

### 4.1 Invariants

The gate checks five invariant families, summarized in Table 1. Dispatch acceptance ensures that the backend can submit the request and receive a job handle. Terminal-state reachability checks that polling makes bounded progress; it does not require success by itself. Completed status distinguishes successful terminal execution from failed or expired terminal states. Manifest presence and artifact consistency convert a completed job into a usable result. The table also lists which negative controls exercise each invariant. This table is not a full fault taxonomy. It is a compact comparison matrix for the smoke contract, chosen to cover the most direct assumptions in the user’s requested live-wiring path.

The invariants are motivated by adjacent standards and testing practices but are not copied from any one source. HTTP and URI specifications provide common naming and resource conventions [5, 9]; OpenAPI and JSON Schema provide interface-description and validation patterns [20, 41]; provenance and artifact standards emphasize that artifacts should be accompanied by machine-readable metadata [19, 30, 33]. The smoke gate chooses the smallest subset needed for the investigated boundary. That subset is intentionally stricter than checking a process exit code and intentionally weaker than a signed provenance chain.

| Invariant family                 | Positive condition | Negative controls         | Contract role                                                                    |
|----------------------------------|--------------------|---------------------------|----------------------------------------------------------------------------------|
| Dispatch accepted                | Happy path         | None                      | Confirms that the backend admits the request and returns a job handle.           |
| Terminal state reached           | Happy path         | Delayed completion        | Bounds the poll loop and distinguishes progress failures from manifest failures. |
| Completed terminal status        | Happy path         | Engine error; stale token | Rejects terminal responses that do not represent successful execution.           |
| Manifest object present          | Happy path         | Missing manifest          | Requires an artifact manifest rather than only a terminal status.                |
| Artifact checksum and size match | Happy path         | Bad checksum              | Checks that declared artifacts are retrievable and match recorded bytes.         |

Table 1: Smoke-gate invariants and the controls used to exercise them. Each row identifies the contract obligation, the positive condition expected to satisfy it, and the local negative control that should fail when the obligation is violated. The table is a manuscript-native rendering of the executed design, not an exhaustive taxonomy of integration failures.

## 4.2 Metrics

For a scenario  $c \in \mathcal{C}$  and repetitions  $r = 1, \dots, R_c$ , let  $y_{c,r} \in \{0, 1\}$  be the expected pass label, with  $y_{c,r} = 1$  only for the happy path in this study. Let  $\hat{y}_{c,r} = G_{B_{\max}}(\tau_{c,r})$  be the observed gate verdict. The outcome-detection rate is

$$\hat{D}_c = \frac{1}{R_c} \sum_{r=1}^{R_c} \mathbb{I}[\hat{y}_{c,r} = y_{c,r}], \quad (4)$$

and the observed pass rate is

$$\hat{P}_c = \frac{1}{R_c} \sum_{r=1}^{R_c} \hat{y}_{c,r}. \quad (5)$$

The timeout rate  $\hat{T}_c$  is the fraction of repetitions in which no terminal status is observed within  $B_{\max}$  polls. The median and 95th-percentile poll counts summarize polling pressure, and elapsed wall-clock time is reported only as local harness overhead. These metrics follow standard performance-analysis advice to report distributions or quantiles rather than only means [8, 15]. They are not service-level objectives because the harness has no real network, queue, or storage latency.

## 5 Experimental Protocol

The executed evidence is a deterministic local live-wiring smoke suite. It is results-facing evidence in this manuscript because it was actually run and produced the tables and figures reported below. The suite uses six conditions: a happy path, a missing manifest, a bad checksum, an engine error, a stale poll token, and a delayed-completion condition. Each condition was repeated  $R_c = 80$  times with seed 20260706 and a poll budget  $B_{\max} = 5$ . The harness uses deterministic pseudo-randomness only to vary job identifiers and completion poll counts within the configured condition. Every repetition writes artifact bytes for completed jobs except when the scenario intentionally removes or corrupts the manifest evidence.

### 5.1 Scenario Fit

The happy path directly matches the live-wiring hypothesis: the backend dispatches, the engine completes, and the manifest points to retrievable artifacts whose checksums and sizes match. The missing-manifest condition tests whether the gate rejects a completed job that lacks a consumable result object. The bad-checksum condition tests byte-level consistency between manifest and store. The engine-error and stale-token conditions test terminal states that should not be treated as success. The delayed-completion condition tests the responsiveness assumption embedded in  $B_{\max} = 5$ . Its relevance is weaker and intentionally mixed because a delayed job may be acceptable

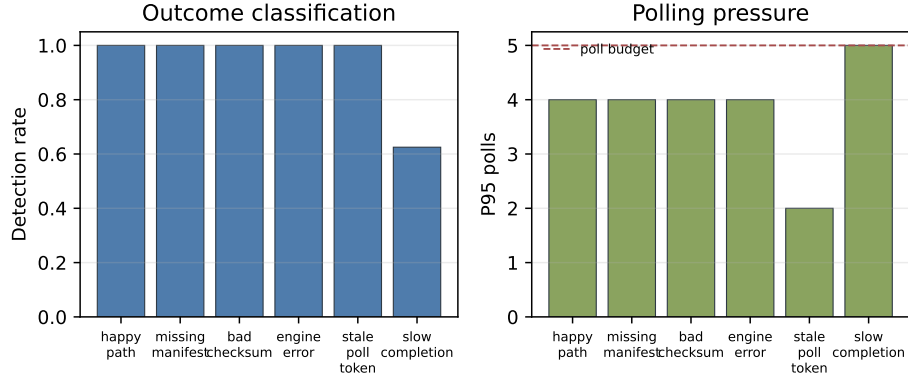


Figure 2: Outcome classification and polling pressure for the local live-wiring smoke suite. The left panel reports the fraction of repetitions whose observed verdict matched the scenario expectation, while the right panel reports the 95th-percentile poll count with the configured five-poll budget shown as a dashed reference. The delayed-completion condition is intentionally mixed because some jobs complete within the budget and some do not, so it should be interpreted as a timeout-policy diagnostic rather than a uniform failure.

in production if the service contract allows longer polling. For that reason, delayed completion is interpreted as a timeout-policy diagnostic rather than a clean wiring failure.

The protocol does not use real production data, external queues, remote object storage, or a live backend. This is a meaningful limitation, but it is not a defect in the local evidence label: the study is a smoke harness for live-wiring structure, not a production benchmark. The local harness lets us exercise controls that would be unsafe or awkward to trigger in a shared service. This mirrors fault-injection practice, where controlled failures reveal whether detection logic reacts to specific classes of defects [1, 4]. It also follows empirical software-engineering practice by fixing seeds, repetitions, and measured outcomes before interpreting the run [39].

## 5.2 Implementation and Reproducibility Context

The harness is implemented in Python using the standard library for dispatch state, manifest writing, checksums, CSV summaries, and JSON run reports. Vector figures are generated from the same run summaries. The implementation records the seed, repetition count, poll budget, scenario list, per-trial outcomes, and aggregate summaries. No external service credentials, network calls, or nonlocal datasets are required. This makes the evidence easy to rerun, but it also means that concurrency, authentication, deployment drift, storage permissions, and process scheduling are not evaluated.

For each trial, dispatch starts a fresh job, polling continues until either a terminal response or the poll budget is reached, and manifest validation is applied only to completed jobs. The smoke gate then compares the observed verdict with the expected outcome for the scenario. A positive trial passes only when dispatch is accepted, completion is observed within budget, the manifest is nonempty, every artifact URI is unique, every artifact exists, every checksum matches, and every byte size matches. A negative trial is counted as detected when the gate rejects it. This separation is important: a negative control that fails for the wrong reason would still indicate that the local contract is not fine-grained enough for diagnosis, even if its binary verdict is correct.

## 6 Results

The local live-wiring smoke suite produced 480 trial records: 80 repetitions for each of six conditions. Figure 2 summarizes the main result, and Table 2 gives the numeric values used in the figure. The happy path passed in every repetition. The missing-manifest, bad-checksum, engine-error, and stale-token controls were rejected in every repetition, giving detection rates of 1.0 for those conditions. The delayed-completion condition had a detection rate of 0.625 because 62.5% of repetitions did not reach completion within five polls, while 37.5% completed before the budget and therefore satisfied the gate. This is the most important negative result in the paper: responsiveness is not the same as structural wiring, and the chosen budget defines what the gate treats as suspicious.

| Condition          | Trials | Pass rate | Detection | Timeout | Median polls | P95 polls |
|--------------------|--------|-----------|-----------|---------|--------------|-----------|
| Happy path         | 80     | 1.000     | 1.000     | 0.000   | 3.0          | 4.0       |
| Missing manifest   | 80     | 0.000     | 1.000     | 0.000   | 3.0          | 4.0       |
| Bad checksum       | 80     | 0.000     | 1.000     | 0.000   | 3.0          | 4.0       |
| Engine error       | 80     | 0.000     | 1.000     | 0.000   | 3.0          | 4.0       |
| Stale token        | 80     | 0.000     | 1.000     | 0.000   | 2.0          | 2.0       |
| Delayed completion | 80     | 0.375     | 0.625     | 0.625   | 5.0          | 5.0       |

Table 2: Aggregate trial results from the executed local smoke suite. The pass rate is the fraction accepted by the smoke gate, while detection is the fraction whose gate verdict matched the scenario expectation. The delayed-completion condition has weaker fit to a binary failure assumption, and its mixed result shows that the poll budget is part of the claim being tested.

## 6.1 Structural Controls

The structural controls give the cleanest evidence for the dispatch-poll-manifest contract. Missing manifest and bad checksum both reached completed terminal states, so a gate that only checked completion would have accepted them. The manifest predicate in (2) rejected both. Engine error and stale token reached terminal states that were not completed, so (3) rejected them before manifest validation. These outcomes support the claim that the smoke test separates three concepts that are often conflated in informal integration checks: progress to a terminal state, successful terminal status, and consumable artifacts.

The result also clarifies the boundary with observability. A trace identifier can help connect dispatch, polling, and manifest publication, and standards such as Trace Context and OpenTelemetry are the right substrate for that correlation [21, 40]. The gate’s verdict, however, depends on the semantic content of the response and manifest. A request can be traceable yet still fail the smoke contract because the manifest is empty, corrupt, or inconsistent with storage. Conversely, a local harness can validate the manifest without proving that a production trace path has the intended sampling, propagation, or retention behavior.

## 6.2 Timeout Sensitivity

Delayed completion is the only condition with mixed support. The local harness configured delayed jobs to require additional polls, but the deterministic variation allowed some repetitions to complete exactly at or before the five-poll budget. Those repetitions satisfied the contract and were accepted. If the operational intent is to reject every delayed job, the budget must be lower or the delayed condition must be defined with a stricter lower bound. If the intent is only to catch jobs that exceed a reasonable smoke-test budget, the observed mixture is appropriate because it distinguishes slow-but-complete jobs from jobs that breach the local budget.

This matters for scientific interpretation. The delayed-completion result should not be read as evidence that the gate failed to identify structural wiring bugs. It tests a different assumption: the responsiveness threshold. Tail-latency work shows why threshold choices matter in large systems [8], and flaky-test research shows that timing-sensitive tests can mislead when the timing model is unclear [18, 25]. The local result therefore motivates an explicit service-level budget before using the same gate as a release blocker. In the present manuscript, we treat the delayed-completion condition as low-relevance evidence for structural wiring and moderate-relevance evidence for timeout-policy design.

## 7 Discussion

The main practical lesson is that a useful smoke gate should be strict about the artifact handoff and explicit about time. Strictness about artifacts is easy to justify: downstream automation cannot act on a completed status alone. It needs a manifest with artifacts that exist and match their declared bytes. This aligns with the direction of artifact and supply-chain standards, which increasingly treat metadata and provenance as objects that consumers should verify rather than trust implicitly [30, 32, 33, 35]. The smoke gate is a small version of that idea. It verifies neither trust roots nor source lineage, but it does ensure that the first handoff object is not hollow.

Time is harder. A smoke test is useful because it returns quickly, yet asynchronous engines can legitimately take different amounts of time across workloads. The poll budget  $B_{\max}$  is therefore a scientific assumption, not an implementation detail. A budget that is too small produces false alarms for jobs that are healthy but slow; a budget that is too large weakens the test’s ability to catch progress regressions quickly. The delayed-completion condition in

Table 2 exposes that tradeoff. It does not invalidate the manifest checks, but it prevents a broad claim that the local gate detects every relevant integration fault. This distinction is important for future production use, where queue depth, cold starts, storage latency, and external rate limits would change the timing distribution.

The novelty boundary is also narrow. The paper does not propose a new microservice testing taxonomy, a new tracing standard, or a new provenance framework. It combines ideas that are already well motivated: frequent automated feedback from CI, boundary contracts from service testing, manifest checks from artifact practice, and negative controls from fault injection. The contribution is the concrete assembly of those ideas for a dispatch-poll-manifest boundary and the local evidence showing what that assembly catches. This boundary is common enough to matter in scientific workflow engines, CI services, and artifact-producing evaluators, but it is small enough that a smoke test can run before heavier benchmarks or human review.

## 8 Operational Integration Guidance

The local evidence suggests a deployment pattern rather than a one-size-fits-all test. A team should first decide which request shape is safe and representative for a smoke gate. For a scientific writing or analysis engine, that request should be small enough to run frequently, but it should still exercise the same control-plane path as ordinary work: authentication if present, dispatch serialization, queue admission, polling, terminal-state reporting, manifest publication, and artifact retrieval. Replacing the real artifact path with a mock may be useful during unit testing, but it changes the property being checked. The smoke gate earns its value when it crosses the same integration boundary that is most likely to break during deployment, configuration, schema, or storage changes.

The next decision is how to report failures. A single pass-fail status is useful for release automation, but it is insufficient for diagnosis. The failure categories in Table 1 should be preserved in logs and dashboards: dispatch rejection, no terminal state within budget, failed terminal state, expired token, missing manifest, missing artifact, checksum mismatch, and size mismatch. Trace identifiers should be propagated through these records so that engineers can connect the gate verdict to service logs and distributed traces [21, 29, 40]. The gate should also emit the job identifier and manifest summary to a restricted operational channel. It should not expose artifact contents or secrets in public logs. This separation keeps the smoke test useful for debugging while respecting ordinary production confidentiality constraints.

Rollout should be staged. In a developer environment, the gate can run against a deterministic harness and fail on every violated invariant. In a staging environment, it should run against the real backend and a controlled engine namespace, with negative controls enabled only when they cannot affect shared users. In production, the positive smoke path can run as a canary request, while destructive negative controls should be replaced by contract checks, staging fault injection, or preflight validation. This staged approach mirrors the distinction between local contract confidence, staging integration confidence, and production availability evidence. It also keeps the artifact-focused smoke gate from becoming an accidental load test, which would require a different protocol and a different interpretation of tail behavior [8, 15].

Finally, the gate should be versioned with the API and manifest schema. A manifest field added for provenance, a new terminal state, a changed digest algorithm, or a new artifact type can all turn a useful gate into a stale one. Treating the smoke contract as versioned engineering material aligns it with OpenAPI, JSON Schema, and SBOM practice [20, 33, 41]. When a schema changes, the gate should include a transition period with both old and new expectations, or it should clearly fail with a schema-version message. This is especially important for research artifacts because silent schema drift can make later reproducibility failures look like scientific disagreements when the real cause is a broken handoff format.

## 9 Limitations and Future Work

The strongest limitation is data fit. The executed evidence uses a deterministic local harness rather than the real backend, real Quarks engine, real queue, or real artifact store. That choice is appropriate for smoke-gate design because it permits controlled negative cases, but it prevents claims about production reliability, throughput, latency tails, authentication behavior, authorization scope, storage consistency, or operational observability. The results support the hypothesis only under the local harness assumption: the backend-engine boundary can be modeled as dispatch, bounded polling, terminal state, manifest publication, and artifact retrieval. Conditions that violate that model, such as streaming results, multi-stage manifests, partial artifact publication, or manual approval gates, would need a revised contract.

The delayed-completion result is another limitation and a useful warning. Its measured property is responsiveness under a five-poll local budget, not structural manifest integrity. Because 37.5% of delayed repetitions completed

within the budget, the condition has mixed fit to the assumption that "delayed" means "should fail." The correct interpretation is that the chosen budget is not a universal threshold. A production follow-up should estimate a latency distribution for representative jobs, choose a smoke-test budget from that distribution, and report both false-alarm and missed-delay rates. The appendix includes a sensitivity diagnostic to show how such budget choices can be explored, but the current run is not enough to set a production service-level objective.

## 9.1 Future Work

The next step is to run the same contract against a staging or production-like deployment with real dispatch, real polling, and real artifact storage. That run should preserve the manifest checks in (2) while adding authentication, authorization, trace propagation, retry, and storage-permission checks. It should also include workload classes rather than one synthetic request shape. For each class, the experiment should measure completion-time distributions, queue wait, storage read latency, manifest size, artifact count, and failure diagnosis. Negative controls should be introduced with bounded blast radius, for example by using a staging namespace, fault-injection hooks, or a fake storage prefix that cannot affect production users. Only then would it be appropriate to claim that the gate validates live backend-to-engine wiring in an operational environment.

## 10 Conclusion

This paper studied a small but consequential systems boundary: dispatching work to an asynchronous artifact-producing engine, polling until terminal state, and consuming the artifact manifest. We defined a compact smoke-gate contract, implemented a deterministic local harness, and executed 480 trials across one positive condition and five negative controls. The gate accepted every happy-path trial and rejected four structural controls in every repetition. The delayed-completion control produced mixed evidence, showing that timeout policy is part of the claim rather than a background detail. The resulting conclusion is bounded and useful: a dispatch-poll-manifest smoke test can expose live-wiring and artifact-handoff defects early, but production confidence requires runs against real services, real storage, real timing distributions, and stronger provenance and observability checks.

## References

- [1] 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*, pages 331–346, 2015. doi: 10.1145/2723372.2723711. URL <https://doi.org/10.1145/2723372.2723711>.
- [2] Association for Computing Machinery. Artifact review and badging, 2020. URL <https://www.acm.org/publications/policies/artifact-review-and-badging-current>.
- [3] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004. doi: 10.1109/TDSC.2004.2. URL <https://doi.org/10.1109/TDSC.2004.2>.
- [4] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos engineering. *IEEE Software*, 33(3):35–41, 2016. doi: 10.1109/MS.2016.60. URL <https://doi.org/10.1109/MS.2016.60>.
- [5] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. RFC 3986: Uniform resource identifier (URI): Generic syntax, 2005. URL <https://www.rfc-editor.org/rfc/rfc3986.html>.
- [6] Antonio Bucchiarone, Nicola Dragoni, Schahram Dustdar, Patricia Lago, Manuel Mazzara, and Victor Rivera. Microservices: Science and engineering. In *Microservices: Science and Engineering*, pages 1–13, 2018. doi: 10.1007/978-3-319-67425-4\_1. URL [https://doi.org/10.1007/978-3-319-67425-4\\_1](https://doi.org/10.1007/978-3-319-67425-4_1).
- [7] CycloneDX Core Working Group. Cyclonedx specification overview, 2024. URL <https://cyclonedx.org/specification/overview/>.
- [8] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013. doi: 10.1145/2408776.2408794. URL <https://doi.org/10.1145/2408776.2408794>.
- [9] Roy T. Fielding, Mark Nottingham, and Julian Reschke. RFC 9110: HTTP semantics, 2022. URL <https://www.rfc-editor.org/rfc/rfc9110.html>.
- [10] Nicole Forsgren, Jez Humble, and Gene Kim. *Accelerate: The Science of Lean Software and DevOps*. IT Revolution Press, 2018. URL <https://itrevolution.com/product/accelerate/>.
- [11] Martin Fowler. Continuous integration, 2006. URL <https://martinfowler.com/articles/continuousIntegration.html>.
- [12] David Garlan, Robert Allen, and John Ockerbloom. Architectural mismatch: Why it is hard to build systems out of existing parts. In *Proceedings of the 17th International Conference on Software Engineering*, pages 179–185, 1995. doi: 10.1145/225014.225031. URL <https://doi.org/10.1145/225014.225031>.
- [13] 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*, pages 426–437, 2016. doi: 10.1145/2970276.2970358. URL <https://doi.org/10.1145/2970276.2970358>.
- [14] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010. URL <https://www.oreilly.com/library/view/continuous-delivery-reliable/9780321670250/>.
- [15] Raj Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991. URL <https://www.cse.wustl.edu/~jain/books/perfbook.htm>.
- [16] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov. Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35, 2018. doi: 10.1109/MS.2018.2141039. URL <https://doi.org/10.1109/MS.2018.2141039>.
- [17] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. doi: 10.1145/359545.359563. URL <https://doi.org/10.1145/359545.359563>.

- [18] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 643–653, 2014. doi: 10.1145/2635868.2635920. URL <https://doi.org/10.1145/2635868.2635920>.
- [19] Open Container Initiative. Image format specification, 2024. URL <https://specs.opencontainers.org/image-spec/>.
- [20] OpenAPI Initiative. Openapi specification version 3.2.0, 2025. URL <https://spec.openapis.org/oas/v3.2.0.html>.
- [21] OpenTelemetry Authors. Opentelemetry specification 1.58.0, 2026. URL <https://opentelemetry.io/docs/specs/otel/>.
- [22] Pact Foundation. Pact documentation: Introduction to consumer driven contracts, 2026. URL <https://docs.pact.io/>.
- [23] Claus Pahl and Pooyan Jamshidi. Microservices: A systematic mapping study. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science*, pages 137–146, 2016. doi: 10.5220/0005785501370146. URL <https://doi.org/10.5220/0005785501370146>.
- [24] David L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972. doi: 10.1145/361598.361623. URL <https://doi.org/10.1145/361598.361623>.
- [25] Owain Parry, Gregory M. Kapfhammer, Michael Hilton, and Phil McMinn. A survey of flaky tests. *ACM Transactions on Software Engineering and Methodology*, 31(1):1–74, 2022. doi: 10.1145/3476105. URL <https://doi.org/10.1145/3476105>.
- [26] Francisco Ponce, Roberto Verdecchia, Breno Miranda, and Jacopo Soldani. Microservices testing: A systematic literature review. *Information and Software Technology*, 188:107870, 2025. doi: 10.1016/j.infsof.2025.107870. URL <https://doi.org/10.1016/j.infsof.2025.107870>.
- [27] Gema Rodríguez-Pérez, Gregorio Robles, and Jesús M. González-Barahona. Reproducibility and credibility in empirical software engineering: A case study based on a systematic literature review of the use of the szz algorithm. *Information and Software Technology*, 99:164–176, 2018. doi: 10.1016/j.infsof.2018.03.009. URL <https://doi.org/10.1016/j.infsof.2018.03.009>.
- [28] Justin Samuel, Nick Mathewson, Justin Cappos, and Roger Dingledine. Survivable key compromise in software update systems. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 61–72, 2010. doi: 10.1145/1866307.1866331. URL <https://doi.org/10.1145/1866307.1866331>.
- [29] Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaskan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. In *Google Technical Report*, 2010. URL <https://research.google/pubs/dapper-a-large-scale-distributed-systems-tracing-infrastructure/>.
- [30] SLSA Community. Supply-chain levels for software artifacts specification version 1.2, 2026. URL <https://slsa.dev/spec/v1.2/about>.
- [31] Eliezio Soares, Gustavo Sizilio, Jadson Santos, Daniel Alencar da Costa, and Uirá Kulesza. The effects of continuous integration on software development: A systematic literature review. *Empirical Software Engineering*, 27(3):1–61, 2022. doi: 10.1007/s10664-021-10114-1. URL <https://doi.org/10.1007/s10664-021-10114-1>.
- [32] Murugiah Souppaya, Karen Scarfone, and Donna Dodson. Secure software development framework (SSDF) version 1.1: Recommendations for mitigating the risk of software vulnerabilities. Technical Report SP 800-218, National Institute of Standards and Technology, 2022. URL <https://doi.org/10.6028/NIST.SP.800-218>.
- [33] SPDX Project. SPDX specification version 3.0.1, 2024. URL <https://spdx.dev/wp-content/uploads/sites/31/2024/12/SPDX-3.0.1-1.pdf>.
- [34] The Update Framework Contributors. The update framework specification, 2026. URL <https://theupdateframework.io/spec/>.

- [35] Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *Proceedings of the 28th USENIX Security Symposium*, pages 1393–1410, 2019. URL <https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias>.
- [36] Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall. A note on distributed computing. Technical Report SMLI TR-94-29, Sun Microsystems Laboratories, 1994. URL <https://waldo.scholars.harvard.edu/publications/note-distributed-computing>.
- [37] Muhammad Waseem, Peng Liang, Gaston Márquez, and Amleto Di Salle. Testing microservices architecture-based applications: A systematic mapping study. In *2020 27th Asia-Pacific Software Engineering Conference*, pages 119–128, 2020. doi: 10.1109/APSEC51365.2020.00020. URL <https://doi.org/10.1109/APSEC51365.2020.00020>.
- [38] Titus Winters, Tom Manshreck, and Hyrum Wright. *Software Engineering at Google: Lessons Learned from Programming Over Time*. O’Reilly Media, 2020. URL <https://abseil.io/resources/swe-book>.
- [39] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, 2024. URL <https://portal.research.lu.se/en/publications/experimentation-in-software-engineering-2024-edition/>.
- [40] World Wide Web Consortium. Trace context, 2021. URL <https://www.w3.org/TR/trace-context/>.
- [41] Austin Wright, Henry Andrews, Ben Hutton, and Greg Dennis. JSON schema validation: A vocabulary for structural validation of JSON, 2022. URL <https://json-schema.org/draft/2020-12/json-schema-validation>.

## A Extended Contract Notes

The main text defines the manifest predicate as a deliberately small byte-level check. This appendix clarifies its intended use and the symbols that affect interpretation. The request space  $\mathcal{X}$  contains only request objects accepted by the backend adapter used in the smoke test. It does not represent arbitrary user inputs, adversarial payloads, or malformed API bodies. The job set  $\mathcal{J}$  is finite for each run because the harness creates a bounded number of jobs. The status set  $\mathcal{S}$  is closed under the five labels used by the local interface; a production system with additional statuses, such as cancelled, retrying, paused, or awaiting approval, would need to extend both the terminal set and the success predicate. The artifact map  $A$  is also local: it represents retrievable bytes in the harness output store, not consistency guarantees for an external object store.

The manifest validity predicate checks nonempty artifact lists, unique URIs, retrievability, SHA-256 digest equality, and byte-size equality. It does not validate URI authorization, media type semantics, schema version compatibility, license metadata, provenance signatures, or transitive dependency inventories. Those omissions are intentional. Standards such as SLSA, SPDX, CycloneDX, in-toto, and TUF define richer metadata and trust concepts [7, 30, 33–35]. A smoke test should call into those richer validators when the deployment already emits their records. The local study instead focuses on the minimum object a backend must consume before any richer provenance layer can help: a manifest whose declared artifacts can actually be read and checked.

The gate verdict in (3) is monotone with respect to the included checks in a limited sense: adding another required predicate can turn passes into failures, but cannot turn failures into passes unless the entire gate is redefined. This property is useful for engineering rollout. A team can begin with dispatch, completion, manifest presence, and byte checks, then add schema-version checks, artifact-type allowlists, trace correlation, or provenance verification. The cost is stricter failure behavior. New predicates should therefore be introduced with explicit negative controls and a measured false-alarm rate, especially when they depend on timing, external services, or optional metadata.

## B Reproducibility Details

The executed run used seed 20260706, 80 repetitions per condition, six conditions, and a five-poll budget. Each trial created one job and a fresh set of artifact bytes when the scenario allowed completion. The harness used deterministic pseudo-random variation to assign job identifiers and completion poll counts. Per-trial records include the condition, repetition, seed, job identifier, dispatch acceptance flag, terminal status, poll count, poll budget, manifest-validity flag, failure modes, observed gate verdict, expected verdict, detection flag, and elapsed local wall-clock time. Aggregate summaries report trial count, expected pass label, observed pass rate, detection rate, timeout rate, median polls, 95th-percentile polls, mean elapsed time, and 95th-percentile elapsed time.

The local wall-clock measurements should not be interpreted as performance results. They are included only to confirm that the harness is lightweight and to make the run report complete. Because the harness does not use network calls, external queues, remote storage, authentication, container orchestration, or concurrent workers, its timing cannot estimate production service latency. The poll counts are more meaningful because they are part of the modeled state machine, but even those counts are valid only under the scenario definitions. A production experiment should record queue wait, engine execution time, storage publication time, manifest retrieval time, retry counts, and trace propagation separately. It should also report the hardware, process isolation, runtime version, and any rate limits that could affect timing.

The code exports vector figures and CSV tables from the same run records used in the manuscript. This shared source reduces the chance that the table and figure disagree, but it does not replace independent validation. A reviewer should rerun the harness, compare aggregate summaries, and inspect a sample of completed manifests to confirm that checksum and size checks use the intended bytes. For stronger reproducibility, a future package should pin the Python version, plotting library version, and any continuous-integration runner image. The current package avoids external dependencies except for plotting, which is used only after trial data have already been written.

## C Sensitivity and Boundary Diagnostics

The poll-budget sensitivity diagnostic repeats the scenario family across budgets from three to eight polls. Figure 3 is not used as primary evidence because the main study fixed  $B_{\max} = 5$  before reporting the results. It is included to show how timeout policy can be examined without changing the manifest predicate. The diagnostic confirms the expected pattern: structural controls remain easy to reject across budgets, while delayed completion changes as the budget changes. That behavior is desirable if delayed completion is meant to represent a responsiveness threshold rather than a manifest defect.

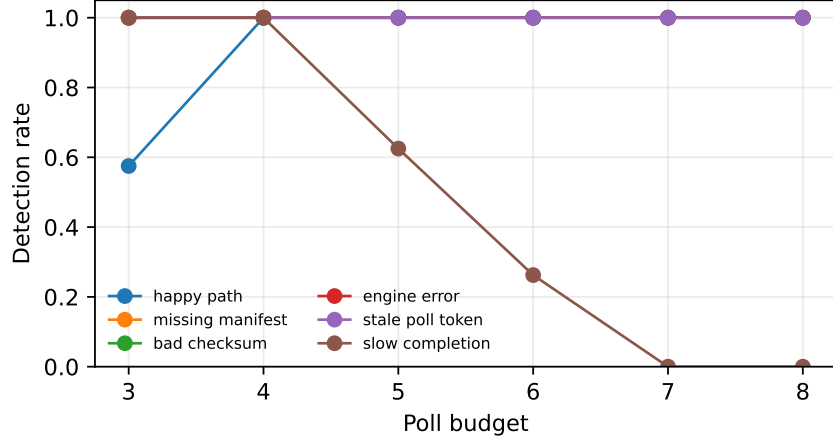


Figure 3: Appendix sensitivity diagnostic for the polling budget. Each line reports outcome-detection rate for a scenario as the allowed number of polls changes from three to eight, using the same deterministic scenario family. The diagnostic is secondary evidence: it explains why delayed completion should be tied to an explicit responsiveness policy, while structural manifest and terminal-status controls remain stable.

The sensitivity result should be read with care. A higher budget can make delayed jobs look healthy because they eventually complete. That is correct if the smoke test’s purpose is to detect only jobs that cannot complete at all. It is incorrect if the purpose is to catch regressions in expected turnaround time. Conversely, a lower budget can make healthy jobs fail when they are merely slower than the local harness’s default. The right budget depends on the target environment and workload. A staging deployment could estimate a percentile-based budget from historical job-completion distributions, then reserve stricter budgets for canary requests whose expected runtime is known. The local paper cannot choose that number for a real service.

The boundary diagnostic also suggests two extensions. First, the gate should record separate failure reasons rather than only a Boolean verdict. The current harness does record failure modes, but the manuscript reports only the aggregate classes. Operational dashboards should distinguish dispatch rejection, terminal failure, timeout, manifest absence, artifact absence, checksum mismatch, and size mismatch. Second, the delayed-completion condition should be split into at least two production conditions: one slow-but-within-objective request and one slow-beyond-objective request. That split would improve relevance by aligning the condition labels with service-level intent.

## D Evidence Lineage and Threats to Validity

The evidence lineage is simple: the manuscript’s primary figures and tables derive from the deterministic local smoke run. The local harness produced trial-level records, aggregate summaries, an invariant matrix, and vector figures. The manuscript then rendered the invariant matrix and summary as LaTeX tables and included the vector figures. Because no prior empirical report, literature matrix, or figure plan was available for this problem, the related-work synthesis, novelty boundary, and artifact set were constructed from public sources and from the executed local evidence.

Internal validity is strongest for the four structural controls because their failure mechanisms are direct and deterministic. Missing manifest removes the manifest object, bad checksum corrupts one digest, engine error returns a failed terminal state, and stale token expires the job after an initial poll. The happy path is also direct because the same validation function checks the artifact bytes. Internal validity is weaker for delayed completion because the label depends on a budget rather than a structural defect. Construct validity is bounded by the chosen contract. The test measures dispatch acceptance, bounded polling, successful terminal status, manifest shape, artifact retrievability, digest equality, and byte-size equality. It does not measure scientific result correctness, provenance trust, security posture, or production reliability.

External validity is limited because the harness is local. Real deployments introduce scheduler contention, network loss, object-store consistency, authorization rules, clock skew, retries, cancellation, partial publication, and observability sampling. Those factors could create failures not represented by the six local conditions. Reliability is supported by fixed seeds and repetitions, but a stronger package should run in continuous integration across runtime

versions and operating systems. The package should also retain historical run summaries so teams can distinguish a new regression from ordinary drift in the timing distribution. Conclusion validity is therefore modest: the local smoke gate is a useful early integration check, and the executed evidence supports that statement for the modeled boundary. It should be treated as the first layer in a validation stack rather than the final evidence for a production backend-to-engine integration.

One additional threat is operator interpretation. A smoke gate that fails for a checksum mismatch points to a different response than a gate that fails because no terminal state appeared. Incident procedures should therefore attach remediation guidance to each failure class. Without that guidance, a precise contract can still produce slow recovery because engineers must infer which subsystem owns the failure.