Executable Assurance

Governed AI as Runtime Infrastructure

A Unified Authority and Evidence Architecture for Regulated Agentic Systems

Ernest Chiweshe

ernest.chiweshe@lumiwright.com

LinkedIn · X @ernest_chiweshe · YouTube · TikTok @ernest.chiweshe

27 October 2026 · Version 17

Not smarter AI.

Trustworthy action, with proof.

Contents

- Abstract
- Architectural intent · October 2026 reference evidence
- 1. Introduction
- 2. Related work
- 3. The architectural gap
- 4. Architecture
- 5. Implementation
- 6. Evaluation
- 7. What Is Established, and the Road Ahead
- 8. Conclusion
- An invitation from a thousand directions
- Author responsibility
- Evidence and Reproducibility
- References

Abstract

Platform and reference scope. The platform is implemented, and was deployed and evaluated in July 2026. The October reference extraction separately reproduces selected mechanisms. **Physical G4 expiry prevention remains OPEN.** Measurement details and deployment-specific qualifications are stated in the evaluation sections.

Author's note (22 October 2026). On the very day I set aside to publish this work, I learned of NVIDIA's announcement of the OpenShell Reference Architecture (21 October 2026), declaring that autonomous agents should "think on Rubin, act on Vera and be governed on BlueField." I welcome this announcement as external validation of the architectural thesis I have built and argued: that model alignment and self-governance are insufficient, that capability is not authority, and that deterministic governance must be separated from execution and enforced as computing infrastructure.

This architecture was specified in late June 2026, its reference cores consolidated on 3 July 2026, and its operational performance measured against hardware security modules on 14–15 July 2026—including a 24-hour soak run of 854,100 consecutively governed decisions across 2,847 batches with zero failed batches, on AWS evaluation infrastructure with staging retention. Eleven regulated domains ran on the same governed core. Those dated, hash-pinned records stand as prior art.

The industry's convergence confirms the necessity of infrastructure-level mediation and clarifies the layers of defense. While platforms like OpenShell provide hardware- and operating-system isolation (the container and network firewall walls), Executable Assurance provides the per-decision cryptographic authority and transactional commitment protocol (the lock in the door). Before any consequential effect is released to an actuator, Executable Assurance requires single-use cryptographic permits bound to exact payload hashes, commit-time database receiver enforcement, receipt-bound reconciliation of `UNKNOWN` states, and tamper-evident evidence committed before the fact. They are complementary, independent layers within the same Swiss-cheese model.

Agentic AI has rapidly acquired the machinery to perform work: models can reason over context, invoke tools, coordinate multi-step activity, and involve people when a workflow pauses. In regulated environments, however, capability is not authority. A system must also establish whether an action is permitted, which exact rule governed it, why its automated checks were trusted, who approved it, whether that person held the required authority at that moment, and whether the proof can be verified after the event.

The central contribution of this paper is the architectural **separation of probabilistic reasoning from deterministic authority to act**, implemented through fail-closed external mediation, cryptographic authorization, and independently verifiable evidence.

This paper presents **Executable Assurance**, a runtime architecture that makes those questions part of execution itself. A deny-by-default authority path classifies the consequence of a proposed action, evaluates content-addressed policy, verifies signed findings, binds required human approval to fresh credential evidence, issues a single-use permit for the exact output bytes, and records the result in a signed hash chain anchored to write-once storage. A separate qualification graph measures the operating characteristics of automated verifiers and lets every governed decision cite the exact, signed, expiring qualification report that justified relying on each checker. One canonical evidence graph then produces regulator, auditor, human, and machine views without creating competing versions of the truth.

The architecture is implemented, and was deployed and evaluated in service in July 2026, as a governance plane over a general-purpose agent platform. It preserves one application abstraction and specializes by domain packs rather than separate governance products. Ten regulated-domain proofs ran on the same core without a new core abstraction, and the eleventh domain, Education, arrived in a stronger form: a **signed, versioned, installable domain pack** that the runtime verifies, activates, governs and can roll back without importing a line of pack code. At the evaluation snapshot, approximately 160 architecture-specific tests, including ten fault-injection drills, ran within a 2,031-test platform suite; the platform has since grown past 14,000 tests. DSSE attestations were independently verified by community tooling (`securesystemslib` 1.4.0). Both AWS KMS (managed Ed25519) and single-tenant CloudHSM (ECDSA P-384 in FIPS 140-3 Level 3 mode) profiles passed 300-decision operational baselines with write-once retention (S3 Object Lock); the CloudHSM STRICT profile then sustained 854,100 governed decisions over a continuous 24-hour soak with zero failed batches, and both failed closed when signing or retention was interrupted. The result is not a claim of automatic regulatory compliance. It is a narrower and more useful engineering contribution: authority becomes executable, failure becomes visible, and every released decision can be traced to the evidence that permitted it. Governance becomes reusable runtime infrastructure rather than a control layer rebuilt inside every application.

Architectural intent

Most security systems infer authority from capability: if an agent holds the credentials or API keys to perform an action, the system permits it. In autonomous systems, this assumption fails. Executable Assurance separates runtime execution from governance execution so that *capability is not authority*. Rather than relying on reactive post-hoc auditing or model alignment, the architecture compiles governance policies into deterministic, before-the-fact runtime gates. Consequential actions pass through a single, fail-closed choke-point that verifies single-use cryptographic permits and commits immutable evidence before any real-world effect is released.

Design causes. Everything else in this paper follows from these:

- **Separation of capability from authority.** Decouple the capability to act (holding database credentials, network sockets or execution environments) from the authority to act (holding a valid cryptographic permit). An agent or model proposes; it never holds the authority to execute on its own.
- **Preventative enforcement before the fact (executable policy).** Transform static, paper-based compliance rules into deterministic, fail-closed runtime gates. Consequential side-effects are blocked before they can occur, rather than detected and audited after the damage is done.
- **Write-once evidence as a precondition for effect.** Tamper-evident proof of authorization, or of refusal, is durably committed before any bytes are released to actuators (write-once retention on the deployed platform; append-only, hash-chained records in the reference). If recording fails, execution aborts.
- **Deterministic single-use permits.** Permissions are bound to exact parameters: who, which tenant, which target, the exact byte hash of the payload, and a strict deadline. Once consumed, the permit is spent and cannot be replayed or extended.
- **Fail-closed default.** In any state of ambiguity, partition, timeout or lost acknowledgement, the system refuses and holds the state as `UNKNOWN` rather than guessing or blindly retrying.

Role. The system does not attempt to solve model alignment, filter prompt semantics, or replace operating-system sandboxing. It is an execution choke-point placed directly between an autonomous proposer and the actuator. In James Reason's Swiss-cheese model it is the preventative gate layer: even if prompt engineering, model safety filters or human review fail, the execution layer refuses to release effects without a valid cryptographic permit.

October 2026 reference evidence update

The deployed platform and this reference remain distinct evidence populations. The private release candidate (candidate004) contains 1,244 source files and passed all 2,950 tests on Windows/PostgreSQL 17 and separately on Linux/PostgreSQL 18, with its source hashes unchanged in both runs. Against the frozen 69-obligation register it records 65 MET, 2 MET WITH LIMITS (A15, B-F7b) and 2 OPEN (R-1b, R-2b). The 63 rows met on the earlier Candidate 062 were carried forward by test-level evidence (1,063 of 1,065 earlier tests present and passing on both systems; the other two deliberately broadened), not re-adjudicated. The register is an evidence trail, not a safety percentage. A locally built release-candidate package passes all 2,826 of its public tests from the package alone (Linux, Python 3.14, PostgreSQL 18); the remaining 124 tests need excluded private inputs (chiefly a synthetic Health pack) and are listed with their reasons.

Governed outbound contracts now demonstrate exact payload admission, duplicate refusal without a second POST or budget spend, and receipt-bound reconciliation. Their destinations are synthetic. Complete mediation, credential withholding and host isolation remain deployment obligations; external HTTP and filesystem boundaries do not become one atomic distributed transaction. UNKNOWN is retained until reconciliation and does not authorize blind resend.

The version 0.1.7 installed-package suite was not a pass: 1,957 passed, 75 failed and 40 errors. Its 115 excluded-asset outcomes comprise 92 Health, 14 owner-record, six publication and three requirements cases. Reproducing the private full-suite result requires the exact retained private inputs. Public installation alone does not reproduce that qualification. Publication cleanup creates a separately identified derivative, not an automatic transfer of qualification to changed bytes.

The historical July measurements retain their original workload and staging retention qualifications. They are not measurements of this October reference. See the repository's `docs/publication-closeout.md` for the dated claim-to-evidence bridge, fingerprints, trust boundary and reproduction scope.

21 October 2026 findings (isolated qualification scripts run against the frozen candidate004; not features shipped in its 1,244 files unless stated).

Deadline at commit (R-1b, OPEN). The database receiver refuses, inside COMMIT, a receipt whose deadline has passed (shipped in candidate004). The deadline is supplied by trusted host validation; the database does not verify permit signatures. Work after that check (commit record, WAL flush, visibility) is not bounded by this mechanism. Observed on Linux/PostgreSQL 18 in two independent runs of 60 samples per condition: check-to-commit-record median about 0.18-0.29 ms and check-to-acknowledgement median about 1.1-1.5 ms, with a maximum of 61.5 ms under continuous checkpoints. On Windows/PostgreSQL 17 (60 samples per condition) the medians were about 0.7-1.2 ms to the commit record and 1.1-2.1 ms to the acknowledgement; in 34 of 180 samples the client acknowledgement time preceded the recorded commit timestamp, and in 2 it preceded the in-COMMIT check reading. The cause was not established, so those orderings are not used as completion evidence. These are observations, not a bound.

Recovery (R-2b, OPEN). Isolated qualification scripts against the frozen candidate demonstrate, under HOLD and without activation: exact replay of a certified archive and transfer to a new database identity with row-for-row equality; revocation of credentials lost in the cut-off tail; fencing of runtime logins; a held

clock floor in the new domain bounded by the old floor and the archive certification time; witness freeze requiring at least n-k+f+1 frozen witnesses; and a permanent hold on a witnessed history lost after the cutoff (tested for the missing-tail condition). Where the source cannot be fenced or the tail is lost, the tested mechanism retains HOLD. Controlled resumption with fresh credentials and new-domain activation is unfinished.

Separate barriers. Against the agent, the authority path and the database duty split act as separate barriers. Against a stolen runtime database credential they can fail together on the same paths: the database enforces duty and structure but does not verify signatures. Neither robustness against a stolen runtime credential nor failure independence of the layers is established. Credential custody and host integrity are separate layers this work depends on, not ones it provides.

How the evidence was produced. Implementation and verification were carried out by internal, separated AI-agent roles (one implementer, one verifier re-deriving results from raw evidence, one internal reviewer and coordinator) under the human author, who remains responsible for the work. This is internal verification, not review by independent parties. Raw evidence for the cited test records is hash-pinned and retained.

1. Introduction

1.1 Capability is not authority

Between 2023 and 2026, agentic AI systems moved from demonstrations to deployed software. The major model providers and the open-source ecosystem converged on a common execution substrate: a model-driven loop, tool invocation, multi-step orchestration, checkpointing, and human-in-the-loop interrupts. This substrate answers the question: *can the system do the work?*

Regulated domains (healthcare, financial services, legal compliance, aviation, critical infrastructure) ask a second question that the substrate cannot answer: *is the system permitted to act on the result — and can it prove why?* A hospital deploying an agent that drafts patient-facing material, or a broker-dealer deploying an agent that drafts research briefs, must eventually answer a regulator who asks: who authorized this output, under which rule (exactly which version, in force at which instant), checked by which verifiers (and why should those verifiers be believed), approved by whom (and how was their licence verified), and where is the tamper-evident record?

Contemporary agent stacks reviewed in this survey typically end at a quality verdict — the output is well-formed, sourced, within policy word-lists, "shippable." Quality gates judge artifacts. The regulator's question is about **authority over actions**, and it requires machinery that content quality pipelines are not typically shaped to hold: actor identity and role, action consequence, tamper-evident history, and data-sensitivity classification.

1.2 Governance as shared infrastructure

Organizations commonly rebuild governance around each AI application: one approval flow for a clinical workflow, another evidence model for financial research, another policy layer for government casework. That repeats controls, fragments evidence, and makes every new application carry its own assurance architecture.

Executable Assurance treats governance as a platform capability. The host platform supplies a common execution substrate; an optional governance plane watches consequential actions through a narrow, one-way interface. Domain packs declare policy, required verifiers, risk tiers, approval authority, and emission

rules. Customer configuration can specialize workflows without changing the authority path. Improvements to the shared path therefore strengthen every governed application that inherits it.

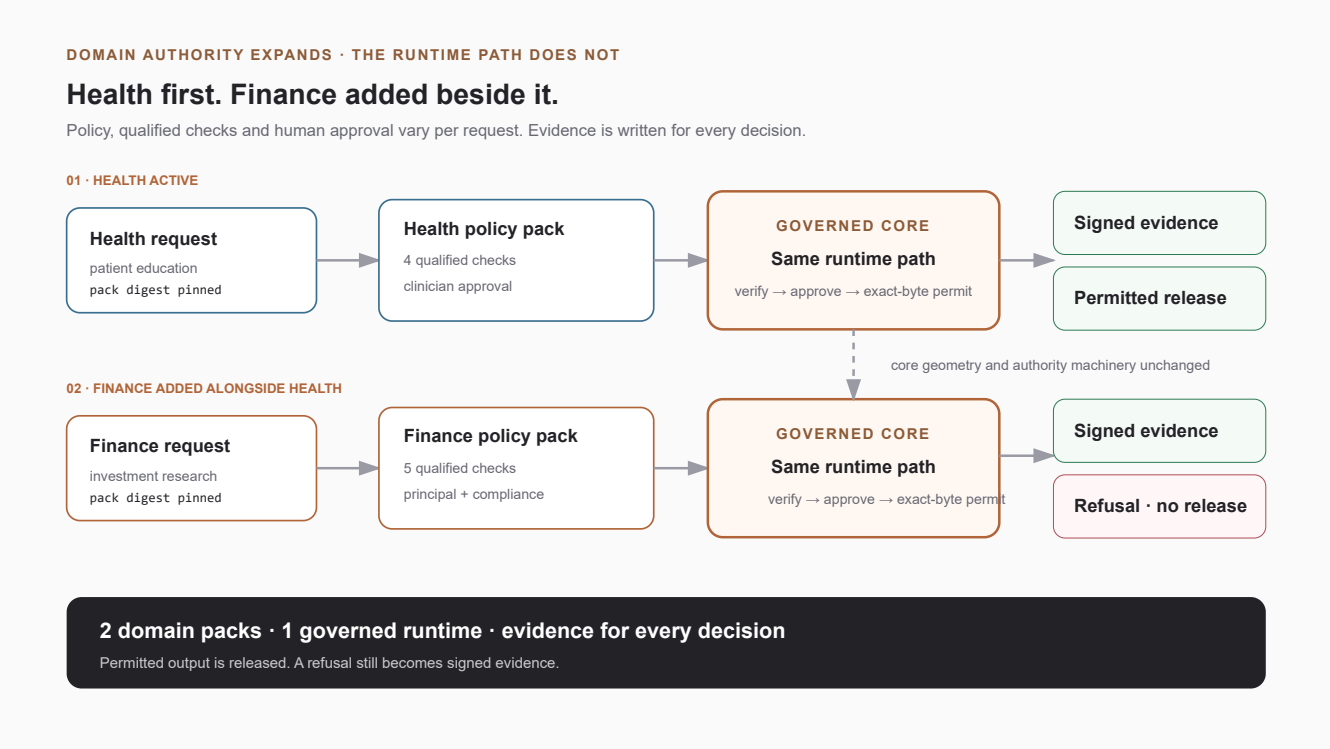


Figure 1. Domain authority expands; the runtime path does not. A domain pack changes policy, qualified checks and approval requirements per request, and every decision writes evidence.

Figure 1 in words. Two rows share one governed core. In the first row a Health request for patient education meets the Health policy pack, with four qualified checks and clinician approval, passes through the core's path of verify, approve and exact-byte permit, and ends in a permitted release with signed evidence. In the second row a Finance request for investment research meets the Finance policy pack, with five qualified checks and approval by a principal and a compliance officer, passes through the very same core path, and this time ends in a refusal with no release, which is also written as signed evidence.

This is the same platform logic that made operating systems, databases, virtualization, and cloud useful: build the dependable substrate once, then specialize above it. The central design move in this paper is **prevention before audit**. Governance is not only a witness that reconstructs an event after execution; for consequential actions, it is a prerequisite for execution. Evidence is generated by the exact same path that decides whether release may occur.

1.3 Central Contribution & Architectural Commitments

The central contribution of this paper is the architectural **separation of probabilistic reasoning from deterministic authority to act**, implemented through fail-closed external mediation, cryptographic authorization, and independently verifiable evidence.

The work makes five architectural commitments:

1. **One application abstraction.** Regulated and unregulated applications remain instances of the same specification (`VerticalSpec`). A regulated domain adds a policy pack, not a parallel platform ("no Health OS").

2. **An orthogonal governance plane.** Governance attaches through an optional declarative block (`RegulatedBlock`) and an injected protocol (`review_draft`, `authorize_release`). The dependency is strictly one-way for core runtime abstractions: the governance plane imports platform plumbing, while the engine, core abstractions, application declarations, quality gate, and provider drivers never import the governance plane (governed components such as the pack runtime build upon it by design).

3. **Two linked trust graphs.** The runtime *decision graph* (assertion → verification → approval → permit → evidence) answers whether a particular action was allowed. The engineering *qualification graph* (verifier → qualification report → dataset → operating characteristics) answers why a verifier should be trusted. A decision cites qualification by content digest.

4. **Credentialed human authority.** Approval is cryptographically bound to a fresh identity and licence-verification receipt issued at the approval instant, rather than to an authenticated click alone.

5. **Canonical evidence, multiple projections.** The evidence graph is authoritative. Regulator, auditor, human, and machine views are pure projections over the same canonical record.

We claim **synthesis, not invention**, and the synthesis is built and running. Every layer stands on mature precedent — safety cases from systems engineering, tool qualification from avionics, capability tokens and signed attestations from systems security, default-deny policy engines, and documentation obligations from AI governance law — and that is a strength: each part is already trusted in its own field. What the survey found nowhere is their composition into one per-decision executable enforcement path for agentic systems, together with two specific mechanisms that are new to this setting: qualification referenced by digest from individual decisions, and approvals cryptographically bound to licence-verification receipts.

1.4 Design Chronology & Evaluation Timeline

To establish clear scientific context without unsupported priority claims, we report the chronological lineage of this work across verifiable repositories and records:

- **Late June 2026 (Architectural Specification & Platform Invariants)**: The architectural invariants of Executable Assurance—including the one-way dependency of the governance plane over the platform, deny-by-default consequence policy with `require-approval | deny` verdicts, multi-tenancy by construction as a mandatory context on every decision, per-tenant model credential resolution (BYOK), clinician approval for health, and the dual-spectrum generality argument ("forgiving and regulated on one core")—were articulated in LumiSpine (the architectural precursor and specification for fail-closed authority gating; evidenced by document file dates from 28 June 2026 and working session records from 30 June 2026, rather than repository commits); per-tenant model credential resolution (BYOK) was named as a design commitment for a future model-gateway adapter. The platform's foundational tenant identity seam (`factory/tenant.py`) dates from the repository's first day (commit `fe495990a`, 31 May 2026).

- **Early July 2026 (Dual Reference Cores & Unification)**: Two parallel, independent reference cores were evaluated head-to-head on 1–2 July 2026 and consolidated into the canonical `regulated_core` package on 3 July 2026 (git tags `regulated-core-baseline`, `v0.1`, and `v0.2`; commit `22848b288`). The dual trust graphs, content-addressed verifier qualification, and license-bound human approval receipts were unified at this milestone.

- **14–15 July 2026 (Operational Measurement Against Hardware HSMs)**: The author had hoped to publish in June but deliberately deferred publication for further validation. Dual reference core consolidation followed in early July, and operational validation against physical cloud

cryptographic infrastructure ran on 14–15 July 2026. The signing and retention seams were integrated with an active, single-tenant AWS CloudHSM cluster operating in FIPS 140-3 Level 3 mode (CMVP Cert #4703) and AWS KMS (managed Ed25519) paired with AWS S3 Object Lock; each profile completed 300-decision operational baselines and live network/storage fault-injection drills, and the CloudHSM STRICT profile executed the continuous 24-hour soak (2,847 batches, 854,100 decisions, zero failed batches; `soak.log`).

- **Current System Release & Workload Breadth (September 2026)**: Following hardware validation, the abstraction was tested across multi-modal creative and organizational workloads, culminating in release tag `v1.9.0` (commit `4e35a6811d3e38cc560473d01e7bcb3eefff00c0`), incorporating domain packs (Education pack #0.1.2), procedural 3D Blender/USD scene composition, deterministic video editing, and PostgreSQL row-level security. Related work published in mid-to-late 2026 is surveyed in Section 2 to frame comparative capability; this work does not derive its runtime abstractions from those subsequent publications.

1.5 Findings & Epistemic Grounding

We summarize seven key findings supported by the implementation, test receipts, and operational evaluation. Each finding is explicitly mapped to its empirical evidence and its falsification criteria:

- **Finding 1 — Executable Composition.** Consequence-aware authorization, credentialed approval, output-bound permits, signed evidence, and write-once retention can operate as one deny-by-default runtime path attached to a general-purpose agent platform without changing its application abstraction.
- *Evidence:* The implementation runs, and was deployed in service at evaluation (July 2026); at the recorded platform release tag `v1.9.0` (commit `4e35a681`; distinct from the 0.1.8rc3 reference package), a single standalone reproduction command (`scripts/reproduce_regulated.py`) executes 149 passed tests (4 skipped by design) across ten fault-injection drills, and the full staged test suite executes 222 passed tests (13 skipped); and the platform's execution engine, application declarations, and quality gates contain zero imports of the governance plane, while governed products build on it by design.
- *Falsification Criteria:* Falsified if attaching the governance plane requires modifying core platform abstractions or breaks unregulated application execution.
- **Finding 2 — Machine-Checkable Trust per Decision.** A decision can carry machine-verifiable evidence of both (a) the measured qualification of its automated checkers (referencing an expiring, content-addressed report by digest), and (b) the approval-time authority of its human approvers (binding approvals to a fresh licence-verification receipt).
- *Evidence:* Both mechanisms are implemented, exercised end to end, and fail closed under drilled failure. The resulting DSSE attestation is verified by independent external tooling (`securesystemslib` 1.4.0). Identity-provider and licence-registry integrations remain deterministic local adapters, as bounded in Section 7.
- *Falsification Criteria:* Falsified if a decision can execute using unmeasured verifiers, unverified approver claims, or stale/expired qualification reports without failing closed.
- **Finding 3 — Reusable Abstraction across Domains.** Eleven regulated domains have run on the one authority path: ten at the evaluation snapshot, with Education added later as the eleventh. The first ten, spanning distinct regulatory philosophies (clinical safety, financial conduct, legal

liability, contractual obligation, drug regulation, aviation safety, occupational safety, administrative law, employment law, critical infrastructure), used identical authority machinery without adding a single core abstraction.

- *Evidence:* Eight domains added after the first two landed as pure additions — zero modifications to any existing file, a checkable property of the git repository history. The eleventh, Education, arrived as signed data through the installable pack format (Finding 5).
- *Falsification Criteria:* Falsified if supporting a new regulatory philosophy requires changing core gate logic or inventing new runtime execution primitives.
- **Finding 4 — Useful Separation of Trust Graphs.** Separating *runtime assurance* ("was this action allowed?") from *engineering assurance* ("should this component be trusted at all?") into two graphs meeting only where evidence cites qualification produces a clean operational boundary: the hot path consumes signed facts without recomputing measurements, while auditors retain a digest-linked route to underlying validation datasets.
- *Evidence:* Hot-path runtime performance remains pure and unaffected by dataset size; auditor projections successfully re-derive full qualification lineage from the pack.
- *Falsification Criteria:* Falsified if hot-path decision latency scales with qualification dataset size or if auditors cannot trace a released decision back to raw checker validation metrics.
- **Finding 5 — Domain packs are a shippable format.** A regulated domain can be delivered as a signed, versioned, multi-artifact bundle of data that the governed runtime verifies, installs, activates atomically, pins per tenant, rolls back and revokes, without importing pack code and without a change to the authority path.
- *Evidence:* The Education formative-feedback pack exists as three signed versions (0.1.0, 0.1.1, 0.1.2) with build provenance and digests in the deployment catalogue; version 0.1.2 is the one activated in the hosted Education pilot, and every activation identifier is carried into release receipts (§4.6, §6.8).
- *Falsification Criteria:* Falsified if installing, upgrading or revoking a pack requires a change to the authority path, or if an unsigned, tampered or revoked pack can govern a release.
- **Finding 6 — Generality Across Workload Kinds (Attempted Agentic OS Abstraction).** In an attempt to define an Agentic Operating System, the runtime abstraction was evaluated across divergent workload categories alongside the high-assurance authority gate: deterministic video editing with cut-plan readiness gates referencing the workflow pattern of OpusClip, multi-modal animated explanation in Lumi Studio referencing procedural Blender/USD composition, and personal assistant and organizational delegation shells referencing OpenClaw and Paperclip. Across the specified integration snapshots for the Lumi Studio port, the core `projectos/engine.py` and `regulated_core` trees were unchanged, while host wiring and adapters specialized execution, demonstrating reuse at those boundaries without claiming all workload code remained invariant.

Evidence: 24 vertical packages under `factory/verticals/`; deterministic cut-plan validation refusing invalid cut boundaries prior to timeline rendering; unchanged `projectos/engine.py` and `regulated_core` blobs across the September 8 studio port (commits `d867f76`, `93d61d2`, `393cd1f`); the Assistant shell (6,921 lines); the Delegation/Paperclip shell (6,268 lines); and native spatial receipts in `output/spatial/`.

Falsification Criteria: Falsified if supporting an unconstrained creative or organizational workload kind requires altering the core runtime engine or introducing a parallel abstraction beside `VerticalSpec`.

- **Finding 7 — Multi-Tenancy by Construction & Two-Tier BYOK Security.** The runtime carries tenant identity through authorization, credential lookup, and persistence, incorporating multi-tenancy by construction and two tiers of Bring Your Own Key (BYOK): operational model/tool credentials and cryptographic signing custody.

Evidence: `factory/tenant.py` threaded through every operation from commit one (`fe495990a`, 31 May 2026), providing immutable tenant context, per-run spend caps (`max_cost_usd`), provider allow-listing (`allowed_providers`), and consequence-tiered policy gates (`can_execute_code` and `can_ship_project`); supported model/agent authentication paths use customer-bound credentials and isolated temporary home environments (`factory/harness_credentials.py`) failing loud with `MissingCredentialError` (arbitrary customer external-MCP onboarding remains unverified); named offline tests exercise specific cross-tenant denials; PostgreSQL Row-Level Security (RLS) is defined and forced across regulated evidence tables (`regulated_core/persistence_pg.py`); and an injectable signing seam admitting customer KMS/CloudHSM modules (evaluated under operator keys). (Limitation: Evaluated in a single-operator deployment with multi-tenant seams intact; not proven at public SaaS scale or penetration-tested; threat model explicitly excludes malicious platform host root).

Falsification Criteria: Falsified if one tenant can resolve another tenant's provider credentials, access another's evidence chains, or exceed declared spend and capability boundaries without failing closed.

1.6 Terminology

Term	Meaning in this paper
Vertical / `VerticalSpec`	An application declared as data and executed by a generic engine; the platform's application abstraction
Governance plane / `regulated_core`	The optional cross-cutting layer enforcing authority; imports platform plumbing, while the core runtime engine, application declarations, quality gate, and provider drivers never import it
Assertion	The unit decisions are made over: content by digest, lineage, confidence, and signed verifications
Consequence policy	A deny-by-default mapping from assertion, action, and context to an authority decision
Verifier	An automated checker producing a signed finding bound to an assertion digest
Qualification	Measured verifier performance sealed in a signed, expiring, content-addressed report
Permit	A single-use, expiring authorization bound to the exact output bytes
Evidence pack	The canonical per-decision record of policy, findings, approvals, receipts, permit, and retention
Projection	A pure stakeholder-specific view over the canonical evidence pack
Domain pack	A signed, versioned, installable bundle of data that specializes the governed core for one domain: vocabulary, schemas, workflow, rulebook, approved sources, conformance cases

---

2. Related work

We organize prior work into five communities, noting for each what it solved and what it does not cover about runtime enforcement for agentic systems. A fuller annotated survey appears in the implementation repository (`docs/regulated-architecture/09-prior-art-and-positioning.md`).

2.1 Safety and assurance cases

Goal Structuring Notation (GSN) [1] and Claims-Arguments-Evidence (CAE) provide a settled ontology — claims, arguments, evidence, context, assumptions — standardized in ISO/IEC 15026-2 [2]. Sector standards (DO-178C in avionics, ISO 26262 in automotive, IEC 62304 and FDA premarket guidance in medical devices) operationalize assurance argumentation for certification. The literature distinguishes the *whole-system, design-time* safety case from *per-decision operational evidence*; recent work on dynamic and continuous assurance (DARPA ARCOS [3], ACCESS [4]) automates the *evaluation* of assurance arguments. AMLAS [5] adapts GSN-style argumentation to machine-learning components; the UK AI Security Institute applies CAE templates to frontier-model safety cases [6, 7]. **Gap for our setting:** assurance cases are predominantly whole-system, design-time documents; the reviewed sources do not establish the complete per-decision runtime combination evaluated here: signed verifier qualification cited by digest, credential-bound human approval, and output-bound single-use authorization enforcing fail-closed gating when evidence is stale or missing.

2.2 AI-specific governance frameworks

NIST's AI Risk Management Framework and its Generative AI Profile [9] define organizational functions (Govern/Map/Measure/Manage) and risk categories; ISO/IEC 42001 [10] defines a certifiable AI management system; the EU AI Act [11] mandates technical documentation (Annex IV), automatic logging (Art. 12), and human oversight (Art. 14) for high-risk systems. **Gap:** these are management-system and documentation obligations specifying governance controls; the surveyed frameworks do not provide per-decision runtime enforcement binding human approval to fresh credential verification or requiring content-addressed, expiring qualification evidence for automated checkers at the point of use.

2.3 Tool and model qualification

DO-330 [12] qualifies software *tools* in avionics with rigor proportional to the consequence of tool error (Tool Qualification Levels), a logic mirrored by ISO 26262's Tool Confidence Levels [13]. The FDA's Predetermined Change Control Plans [14] pre-authorize re-validation protocols for evolving AI medical devices. Documentation schemas (Model Cards [15], Datasheets for Datasets [16]) and content-addressed dataset versioning (DVC) supply the artifact vocabulary. **Gap:** qualification in this line is typically a development-time, per-release activity producing static documents; the reviewed sources do not establish the runtime combination evaluated here where qualified verifier status is machine-checked at the point of use via signed, content-addressed digests that are referenced from individual decisions and expire automatically.

2.4 Runtime authorization and provenance

Capability systems — macaroons [17], Biscuit [18] — provide attenuable, caveat-carrying bearer authorization; SPIFFE/SPIRE [19] provides workload identity. The supply-chain security stack — in-toto attestations in DSSE envelopes [20, 21], SLSA provenance [22], Sigstore with the Rekor transparency log [23, 24] — provides signed, externally verifiable statements about artifacts, with policy-gated verification. C2PA [25] binds signed manifests to content hashes for media provenance. OPA [26] supports programmable authorization policies; Cedar [27] defines default-deny and forbid-overrides-permit semantics. Amazon Bedrock AgentCore Policy [28] applies Cedar to default-deny gating of agent tool calls at a gateway — to our knowledge the closest deployed precedent to our consequence-policy layer. **Gap:** these systems authorize *access* and prove *artifact provenance*; the reviewed sources do not establish the

complete combination evaluated here: evaluating consequence of an AI output's content, requiring authorizing evidence from qualified verifiers, and binding a human approver's professional licence into the authorization ceremony.

2.5 Agentic-AI governance runtimes

LangGraph's interrupt/checkpoint mechanism [29] provides approval pausing via persistence; the Model Context Protocol security principles call for explicit user consent before tool invocation [30]. Guardrail systems (NeMo Guardrails [31], Llama Guard 3 [32], Guardrails AI) attach content checkers to agent pipelines. These frameworks support approval pauses and content filtering, including blocking outputs. **Gap — checked specifically for this paper:** in the reviewed documentation we did not identify this paper's combined contract of professional-credential-bound approval, signed verifier qualification, and output-bound single-use authorization.

2.6 Capability comparison

Table 1 summarizes the survey along the capabilities this architecture composes. Assessments are based on public documentation as of the July 2026 survey; "—" means the capability was not found in the surveyed material, not that it is impossible to add. ● = present; ◐ = partial or adjacent; — = not found.

Table 1. Capability matrix across surveyed systems.

Capability	LangGraph (HITL)	NeMo Guardrails / Llama Guard	Cedar / Bedrock AgentCore	in-toto / Sigstore	watsonx.gov / Credo AI	Executable Assurance (This work)
Deny-by-default action control	—	—	● (tool calls, at gateway)	—	—	●
Consequence policy over output content	—	◐ (partial: blocking filters)	—	—	—	●
Human authorization primitive	● (interrupt/resume)	—	◐ (consent flows)	—	◐ (workflow sign-off)	●
Approval bound to fresh licence evidence	—	—	—	—	—	●
Dual control (two distinct verified identities)	—	—	—	—	◐ (process-level)	●
Signed, expiring qualification cited by digest	—	◐ (published benchmarks, unsigned)	—	—	—	●
Single-use permit bound to exact output bytes	—	—	—	◐ (policy-gated deploy check)	—	●
Signed per-decision chain anchored to WORM	◐ (checkpoints, mutable)	—	—	● (transparency log)	◐ (audit records)	●
Externally verifiable attestation (DSSE/in-toto)	—	—	—	●	—	●
Regulator format documentation projections	—	—	—	—	● (framework mappings)	●
Refuses operation on stale assurance facts	—	—	—	—	—	●

No surveyed system covered the full path; Executable Assurance does. The architecture therefore claims composition rather than reinvention of every layer. Its distinctive mechanisms are the per-decision digest link to verifier qualification (G1) and the cryptographic binding between approval and approval-time credential evidence (G2).

---

3. The architectural gap

Regulated AI needs a continuous chain from proposed action to permitted release. Existing disciplines provide strong pieces of that chain, but they ordinarily stop at different boundaries: policy engines authorize requests, guardrails judge content, approval workflows capture human intent, assurance cases justify systems, and provenance tools attest artifacts. The unresolved architectural problem is to make their assurance facts decisive at the moment an agent attempts a consequential action.

This work addresses three linked gaps identified in the survey:

- **G1 — Qualified machine judgment.** An output-bound, single-use permit is conditioned on signed findings from verifiers whose measured performance is sealed in content-addressed qualification reports and cited by digest from the decision.
- **G2 — Verified human authority.** A human approval is cryptographically bound to identity and licence evidence obtained at the approval instant, including role, status, jurisdiction, and expiry.
- **G3 — One executable path.** Consequence policy, qualification, credentialed approval, permit, signed evidence, and durable retention operate as one fail-closed release path rather than as a management system plus after-the-fact documents.

These gaps define the paper's scope, and the architecture closes all three. Regulatory interpretation and the organization's management system stay with the institution; the architecture gives that institution a runtime substrate whose decisions are enforceable and whose proof is intrinsic to execution.

---

4. Architecture

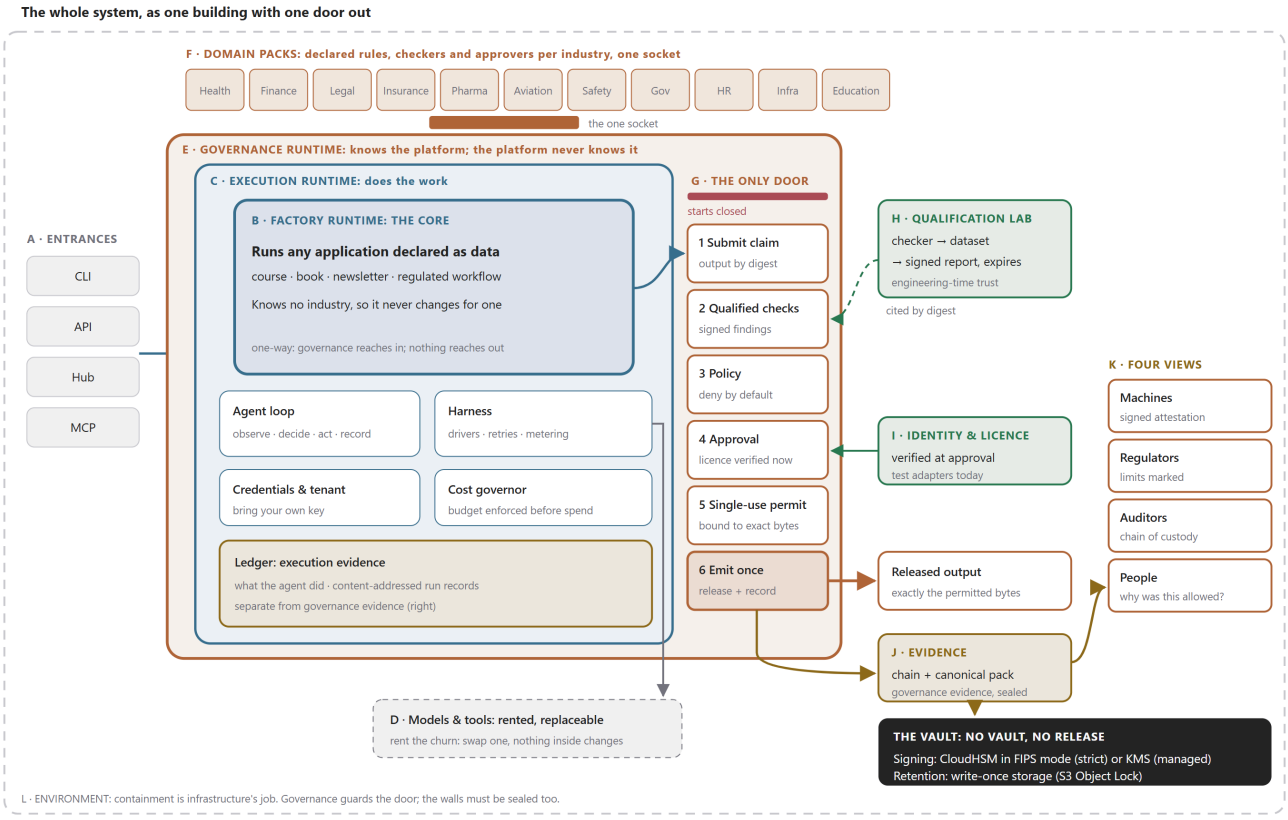


Figure 2. The whole system, as one building with one door out. Nested runtimes do the work; domain packs meet the governance runtime at one socket; every release leaves through the only door, and the evidence it writes serves four views.

Figure 2 in words. A dashed environment boundary surrounds everything. Far left, four entrances: CLI, API, Hub and MCP. In the centre, the factory runtime core sits inside the execution runtime, which sits inside the governance runtime. Along the top of the governance runtime, the domain packs plug into one socket: ten at the evaluation snapshot, eleven in this edition. Below it, outside, rented models and tools connect to the harness. The right wall of the governance runtime is the authority path, six stations from top to bottom, starting closed. Outside to the right, a qualification lab connects to station 2 by a dashed digest line, and an identity and licence registry connects to station 4. Station 6 is the only door out: it releases the exact output and sends the record to the evidence chain, which is signed in a hardware security module or key service vault and kept in write-once storage. The evidence projects into four views: machines, regulators, auditors and people. In this edition the row of domain packs shows eleven: Health, Finance, Legal, Insurance, Pharma, Aviation, Safety, Government, HR, Infrastructure and Education, all meeting the runtime at the one socket.

The figure above is the map for this section: §4.1 is the nesting of the runtimes and the one socket, §4.2 is the only door, §4.3 is the qualification lab, §4.4 is what holds when parts fail, §4.5 is the evidence and its four views, and §4.6 is the row of domain packs.

4.1 One abstraction, one optional plane

The host platform (an agent "factory": multi-tenant loop execution, credential resolution, encrypted artifact storage, run ledger, cost governance) runs *verticals* — applications declared by a `VerticalSpec` and executed by a generic project engine. The architecture's first commitment is that regulated capability is **not** a new

kind of vertical, a new axis, or a separate product. It is a cross-cutting plane, attached through exactly two elements:

```
@dataclass(frozen=True)
class RegulatedBlock:            # optional field on VerticalSpec — pure data
    policy_ref: str              # names the consequence-policy pack
    intended_use: str            # the bounded contract a regulator reads
    prohibited_uses: tuple[str, ...]
    risk_tiers: dict[str, int]
    required_verifiers: tuple[str, ...]
    approver_roles: tuple[str, ...]
    emission_policy: str
```

and an injected protocol (`review_draft`, `authorize_release`) that the platform *defines* and the governance plane *implements*, wired only at the composition root. Two properties follow by construction rather than configuration:

- **One-way dependency.** The governance plane imports the platform's plumbing (signing, ledger, encrypted storage); the platform's engine, core and application declarations never import the plane. Unregulated verticals cannot observe the plane's existence; at the evaluation snapshot all 184 pre-existing vertical tests passed unchanged. Governed products, the pack runtime and the Education workbench among them, are built on the plane deliberately: that is the plane doing its job, and it leaves the engine beneath them untouched.
- **Fail-closed by code shape.** In the engine's release path, the exporter call — the only line that writes output bytes — is dominated by the gate check. A regulated specification with no gate wired refuses to release. There is no configuration in which an unauthorized byte can be written, because no code path reaches the exporter without passing the gate.

Declared *intended use* is load-bearing: regulators regulate systems by purpose. The block makes the boundary machine-checkable — an action outside the declaration is denied as unknown rather than discouraged.

4.2 The authority path (runtime trust)

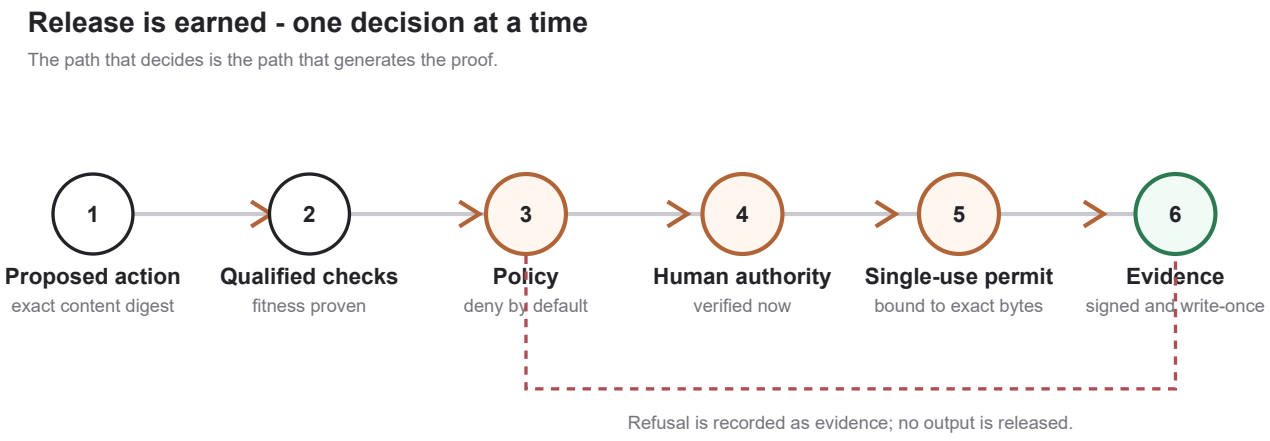


Figure 3. Release is earned one decision at a time. The path that decides is the path that generates the proof.

Figure 3 in words. Six stations in a line, left to right. Proposed action, identified by its exact content digest. Qualified checks, whose fitness is proven. Policy, which denies by default. Human authority, verified now. A single-use permit bound to the exact bytes. Evidence, signed and written once. Beneath the line, a return path shows that a refusal at any station is recorded as evidence and no output is released.

Within the plane, every consequential release traverses one enforced function (`authorize_and_emit`) executing seven checks in fixed order: (1) tenant isolation; (2) a signed, expiring *preflight capability* re-verified on the emit path, so no later step can assert it was allowed; (3) a *producer bar* — the probabilistic producer (the model) may never itself take an authoritative or external action, regardless of policy outcome; (4) authenticity of all attached verifications; (5) the consequence decision; (6) approval binding; (7) permit issuance and single-use emission. Every outcome, including every refusal, is recorded.

Consequence policies. Actions carry an effect on a three-valued risk axis — *reversible* (drafts), *authoritative* (writes to a record of truth), *external* (leaves the system). Policies are deny-by-default and return one of four decisions (allow; allow-with-annotation; require-approval; deny), always with machine-readable reason codes. Unknown content types, actions, or verticals deny automatically. The same assertion under different policies yields different decisions; rigor is a per-vertical dial, which is what lets strict and forgiving verticals share one engine.

Trust anchoring. All attestations are Ed25519 signatures over a canonical byte encoding (floating-point values are rejected in signed bodies; time is integer microseconds). Verification resolves the signer's *name* against a trust directory holding registered public keys; signed objects never vouch for themselves, and each authority (decision recorder, approval, emission, preflight, each verifier) holds a distinct key, so compromise of one cannot forge another. A verification result is bound to the exact assertion by content digest and must be signed by the identity it names; anything unsigned or mis-signed is invisible to policies and therefore denied.

Credentialed approval (G2). A policy's `require-approval` names role groups and a quorum. The workflow refuses self-declared roles: a credential verifier consults an identity provider ("is this an active person in this tenant?") and a licence registry ("does this person hold this role's licence — status, jurisdiction, expiry?") *at the approval instant*, and issues a signed `CredentialReceipt`. The grant binds the receipt's digest; the certificate carries all receipt digests and is itself bound to the digest of the exact requirement and decision record, so an approval cannot be replayed against a re-timed, re-targeted, or content-altered decision. Expired, suspended, wrong-jurisdiction, and wrong-role credentials, and provider outages, all fail closed; a licence revoked between drafting and approval is caught because verification binds to the approval instant. Dual control requires two *distinct* verified identities; one person holding both roles is refused.

Output-bound emission (G1, first half). Approval alone does not release. A single-use, expiring permit binds one tenant, one decision record, one action, and the SHA-256 of the exact output bytes. Altered output voids the permit; a second use is denied and the denial is itself recorded as evidence.

Evidence. Outcomes are appended to a per-tenant hash chain; each signed envelope seals its predecessor's hash and sequence number, so removal or reordering is visible. The chain is periodically anchored to write-once (WORM) storage; for releases with retention wired, a missing retention receipt aborts the release before any byte is exported. Decision records additionally carry the content digest of the **policy document** in force: policies are signed, immutable, content-addressed documents with jurisdictions, effective windows, and supersession-by-digest, so "show me the exact rule text that decided this" has a cryptographic answer rather than a mutable version label.

4.3 The qualification graph (engineering trust; G1, second half)

The runtime machinery presumes trustworthy verifiers; nothing in §4.2 justifies that trust. The architecture therefore holds a second, orthogonal graph, populated at engineering time.

A verifier is *qualified* by measuring it against a labelled dataset whose content hash is its identity (substituted or doctored validation sets are detectable by recomputation). The measurement — true/false positives and negatives; precision, recall, F1 computed in integer arithmetic so no float enters a signed body — together with coverage (jurisdictions, languages, document types) and *known limitations* is sealed in a signed, immutable `QualificationReport` with an expiry. Reports supersede; they are never edited.

The two graphs meet at exactly one seam: the evidence assembler resolves, for each verifier class that attested a decision, the qualification report currently in force, and binds its digest (plus a small human-readable summary) into the evidence pack. The reference-by-digest is a cryptographic commitment to the exact qualification content — structurally analogous to a certificate referencing its issuing authority — giving auditors a one-lookup walk from any released output to the measured performance of every checker that blessed it. Runtime verification remains pure: the hot path never reads datasets or computes metrics.

Two design rules preserve honesty. The reference datasets deliberately include cases the current (deterministic) verifiers miss, and the published characteristics reflect it (e.g., recall 0.75 for the clinical-privacy checker over its golden set): regulator trust attaches to measurement honesty, not to flattering numbers. And deterministic checkers carry no per-result confidence score — their trustworthiness lives at the qualification level; the schema reserves per-call confidence and calibration curves for future probabilistic verifiers.

4.4 Operational assurance

Infrastructure tests ask whether components recover; regulators ask whether guarantees held while components were failing. The architecture defines a catalogue of *failure drills*, each pairing a failure mode with the assurance guarantee that must survive it (e.g., "KMS unreachable during evidence signing → no unsigned evidence can exist"; "WORM unreachable at release → release aborts before any byte is emitted"; "licence revoked mid-approval → approval-instant re-verification denies"). Drill outcomes are sealed in a signed, content-addressed, *expiring* `DrillReport`; violated guarantees are recorded as failures in the signed body — the machinery cannot omit them.

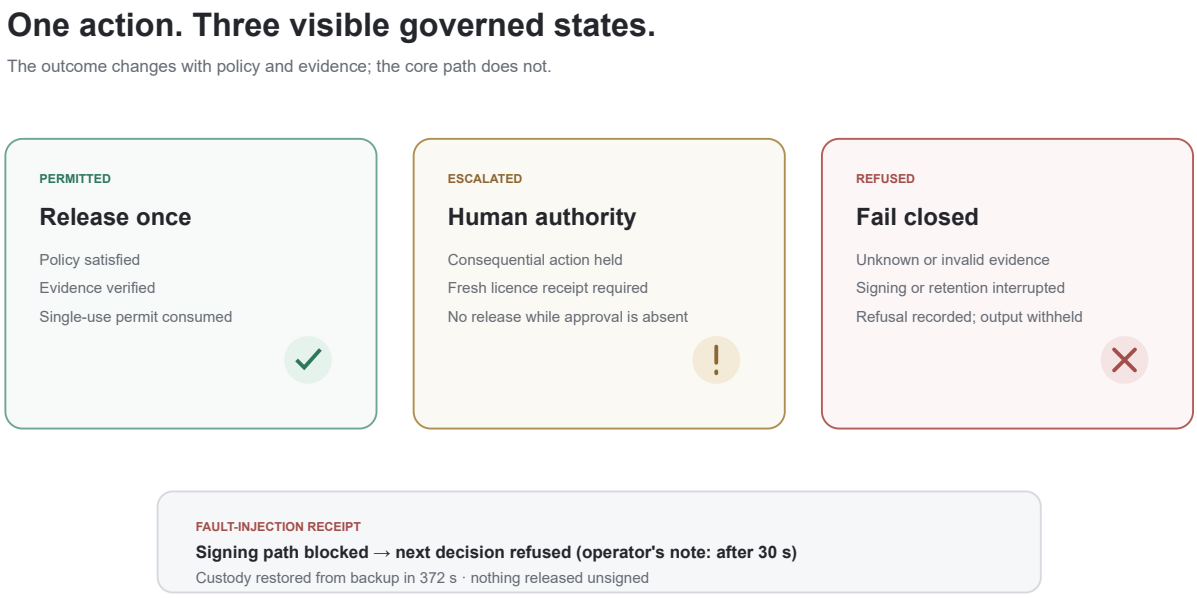


Figure 4. One action, three visible governed states. The outcome changes with policy and evidence; the core path does not.

Figure 4 in words. Three panels. Permitted: release once; policy satisfied, evidence verified, single-use permit consumed. Escalated: human authority; the consequential action is held, a fresh licence receipt is required, and nothing is released while approval is absent. Refused: fail closed; unknown or invalid evidence, or interrupted signing or retention; the refusal is recorded and the output withheld. Beneath them, a fault-injection receipt from real infrastructure: with the signing path blocked the next decision was refused (the operator's note says 'after 30 seconds'), custody was restored from backup in 372 seconds, and nothing was released unsigned.

A **production gate** evaluates a signed assurance attestation against mandatory controls and refuses release readiness unless attested controls are backed by live proof. Five controls are proof-backed in the implementation: key custody (refused while the live signer is a development key rather than KMS/HSM-backed), retention (refused without WORM receipts), authoritative identity (refused unless approvals are provider-backed), validated verifiers (refused unless every required verifier holds a current qualification), and resilience (refused unless a trusted, current drill report proves every required drill passed — a stale report is treated as a failed one). The remaining controls (calibration, incident reporting, model-risk governance, the management system) are organizational; the architecture marks them as such rather than simulating them.

4.5 Canonical evidence and projections

The evidence pack assembled per decision is canonical. Reporting is a set of *pure functions* over it — proven pure in the test suite by bit-comparison of the pack before and after all projections run:

- **Machine attestation:** an in-toto v1 Statement in a DSSE envelope, signed over the byte-exact DSSE pre-authentication encoding so standard external tooling can verify it; subjects are the evidence chain head and the emitted output digest.
- **Regulator view:** the EU AI Act Annex IV technical-documentation structure, populated from evidence (oversight ← approvals and credential receipts; performance metrics ← measured qualifications; risk management ← the digest-committed policy), with an explicit honesty boundary:

sections the runtime cannot know (harmonised standards, declaration of conformity) are marked *supplied by the management system*, never fabricated.

- **Auditor view:** the chain of custody, link by link — each artifact, its digest, and what binds it to the next.
- **Human view:** a deterministic plain-language answer to "why was this decision allowed?", each sentence derived from a pack field and anchored to the evidence chain head.

Because each projection is a total function over the fields its view requires, building the projection layer doubles as a completeness audit of the evidence itself (§6.5).

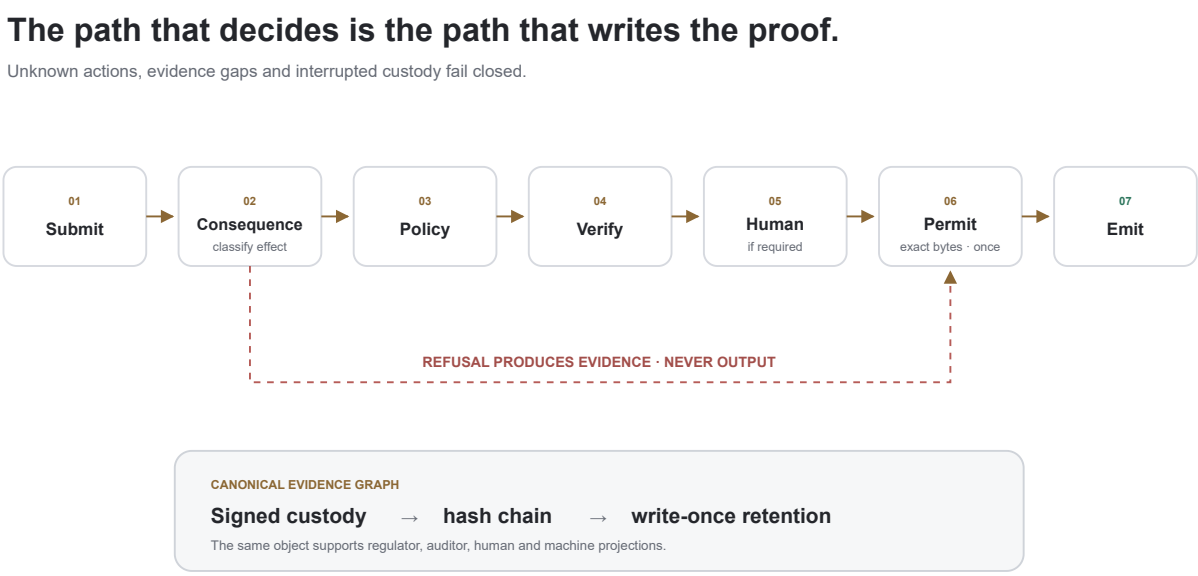


Figure 5. The path that decides is the path that writes the proof. Unknown actions, evidence gaps and interrupted custody fail closed.

Figure 5 in words. A row of stations: submit; consequence, which classifies the effect; policy; verify; human, if required; permit, for the exact bytes and once only; emit. A band beneath reads: refusal produces evidence, never output. Under the whole path lies the canonical evidence graph: signed custody, then a hash chain, then write-once retention. One and the same object supports the regulator, auditor, human and machine projections.

4.6 Domain packs: a signed, installable format

In Version 5 a domain pack was a declaration inside the repository. It is now a product mechanism. A domain pack is a **signed, versioned, multi-artifact bundle of data** that the governed runtime installs, activates and governs without importing a line of pack code (`factory/packs/`). A generic pack gate (`regulated_core/pack_gate.py`) consumes the installed rulebook as it stands, without translating or re-signing it, and drives the same release engine as every other governed decision; its activation read-lock spans the authorization, so a revocation can never slip between version validation and the issue of emission authority.

The format adopts what regulated software already trusts rather than inventing a new one: a content-indexed signed bundle whose manifest names identity, kind (base, profile or overlay), version, compatibility, dependencies and owned namespaces, after OPA bundles [26], FHIR packages [33] and

NIEM descriptions [34]. A pack carries:

- **Three separate trust facts:** the publisher's signature, the build provenance, and a domain expert's signed attestation. Each answers a different question, and none stands in for another.
- **Policy as data,** evaluated by the one default-deny engine, with jurisdiction and organization overlays that may only *tighten* the base.
- **Deterministic verification as host-qualified capabilities** that a pack parameterizes; a pack never brings its own executable checker.
- **Advisory output as cards** that a human accepts or overrides, and the **domain's own vocabulary**, locked by edition.
- **An activation lifecycle:** verify, activate atomically, keep the last known good, roll back, revoke, and pin per tenant, with every activation identifier carried into release receipts.

Packs declare **conformance by level** — structural, semantic, offline behavioural, expert validated — and a lower level never implies a higher one, so a pack author and a buyer always know exactly what has been shown.

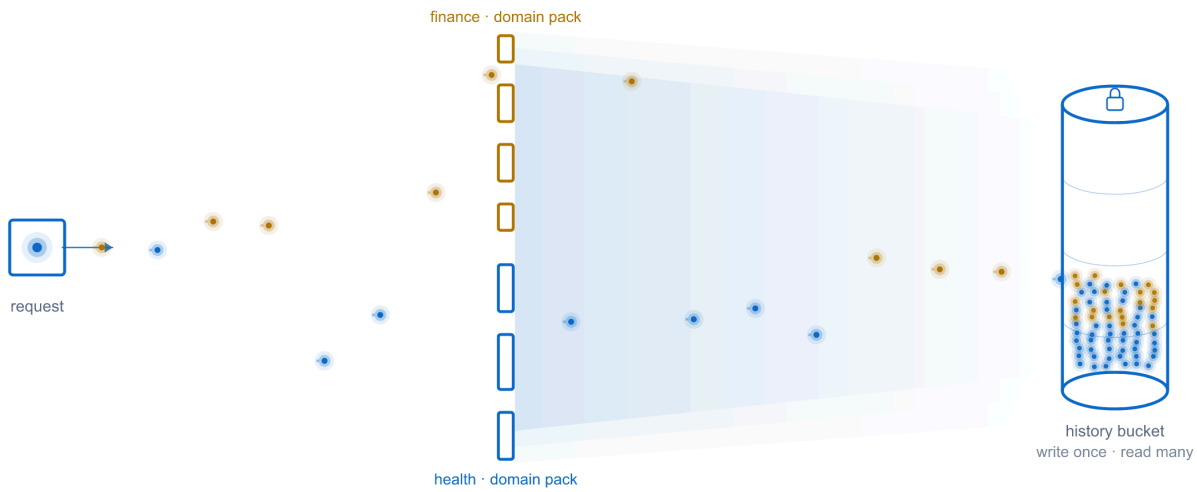


Figure 6. One governed core. Authority is declared per request. Health flows first; Finance joins beside it. Both domain packs use the same governed runtime and write permitted outcomes to one evidence history.

Figure 6 in words. On the left a request sends out a stream of small dots, blue for Health and amber for Finance. In the middle stands a gate made of two stacks of openings: the Finance domain pack above, the Health domain pack below. Dots that pass their own pack's openings travel on through the same shaded runtime path to the right, where a locked cylinder, the history bucket, written once and read many times, fills with blue and amber dots together. The two packs differ; the path and the evidence store are one.

---

5. Implementation

The implementation is a Python package, `regulated_core` (31 modules at the Version 6 source commit), living beside the host agent platform (`factory`) in the repository `agent-factory-prod`. Salient facts:

- **Real cryptography, real seams.** Signatures are Ed25519 via the `cryptography` library; the KMS seam wraps the platform's sign-by-reference `ExternalSigner`, so the production path (key outside the process) and the development path share one protocol. Durable storage schemas (append-only evidence with row-level security) exist as SQL; the reference implementation exercises in-memory equivalents.
- **Provider seams as protocol + adapter.** Every external dependency — KMS/HSM backend, WORM store, identity provider, licence registry — is expressed as a narrow protocol. The signing and retention seams were exercised against real AWS KMS, single-tenant CloudHSM, and S3 Object Lock infrastructure (§6.7). Identity and licence-registry seams remain deterministic local adapters with induced-outage controls for drills.
- **The platform was in service.** At evaluation (July 2026) the host platform ran as a managed deployment on AWS: every task family in private subnets across two availability zones, container images scanned, signed and pinned by digest, ECS-native canary releases with alarm-driven rollback proven by live fire, durable state in PostgreSQL, multi-tenant, with per-workspace encrypted credentials so that a customer's runs use only that customer's own model keys or subscriptions. The governed Education workbench was deployed on it at evaluation (July 2026).
- **Eleven regulated domains.** Education is delivered as an installable pack (§4.6, §6.8). Of the first ten, two are flagship verticals: *clinician-approved patient education* (health: four deterministic verifiers — PHI boundary, source fidelity, prohibited clinical action, readability; single clinician approval, provider-verified) and *investment research briefs* (financial services: five verifiers; dual control, research principal + compliance). Plus eight minimal *proof verticals* (legal, insurance, pharmaceuticals, aviation, industrial safety, government, human resources, critical infrastructure), each a policy pack with three domain-specific verifiers and a licence-verified approval authority (Table 2). Every domain is a policy pack plus a thin gate subclass plus a specification; all inherit the entire authority path from one base gate.
- **Provenance of the core.** The consolidated core is the outcome of an adversarial process: two independently written reference implementations, blind cross-audit with reviewers instructed to argue opposing verdicts, followed by a merge that kept one substrate, grafted three mechanisms from the other (the single enforced emission chokepoint; requirement-digest approval binding; a lean base assertion), and fixed five security defects surfaced by the audit (among them: verification anchored to keys carried inside objects, and a mutable permit-use ledger).

---

6. Evaluation

Evaluation combines construction evidence, deterministic tests, adversarial drills, independent attestation verification, and operational measurement. The accompanying reproducibility command re-derives the architecture-specific results and evidence artifacts (§6.6).

6.0 Adversary model & trust assumptions

The security-relevant evaluation assumes an adversary who can: submit arbitrary content and actions through the platform; present forged or replayed artifacts (verifications, approvals, permits, drill and qualification reports) including artifacts signed with attacker-generated keys under trusted identity *names*;

re-time, re-target, or alter content after approval; reorder or truncate the evidence log given storage access; hold valid credentials in the *wrong* role, jurisdiction, or status (expired, suspended, revoked mid-flow); and cause outages of any external provider (signing, retention, identity, licence registry) at chosen moments.

Out of scope, and stated as explicit trust assumptions: compromise of the trust directory's registration path itself; a malicious approver holding a *valid, in-scope* licence (the architecture makes such an approval attributable and non-repudiable, not impossible); a malicious platform host (the plane runs in-process with the platform); side channels; and incorrect interpretation of a regulation encoded into a policy document (the architecture proves which policy content was in force, not that the content was legally correct). Each in-scope capability maps to at least one negative test or drill in §6.3–§6.4.

6.1 Additivity

At the evaluation snapshot the platform's full suite passed at 2,031 tests with the plane and all ten domains present; the 184 tests covering pre-existing (unregulated) verticals passed unchanged, and a static check confirmed the platform contained no import of the governance plane. The platform has kept growing on that footing: at the Version 6 source commit the suite collects 14,721 tests (`pytest --collect-only`), and the engine, core, application declarations, quality gate and provider drivers still contain zero imports of the plane. We treat this collected test inventory as evidence of platform scale and architectural decoupling, not executed non-regression proof; the architecture-specific evidence is the roughly one hundred and sixty tests across the regulated plane (integration, ten domains end-to-end, all six seams, drills, projections, external attestation verification).

6.2 The ontology test, replicated across ten domains, and passed by an eleventh

The second domain (finance) was built under a pass condition declared in advance: it must require zero new abstractions in the core. It did — the vertical landed without modifying any core module, and its verifier-qualification suite subsequently landed as a single test file, including the structurally awkward case of *contextual* verifiers taking (draft, sources) rather than bare text, which the qualification contract absorbed because a labelled case's text is defined as the verifier's full serialized input.

The same experiment was then repeated for eight further domains, chosen to stress *different regulatory philosophies* rather than different subject matter, under the same pre-declared pass condition. Each landed as one new domain module plus one test file; a diff over the eight commits shows **zero modifications to any existing file** (additions only), making the claim a checkable property of the repository history rather than an assertion.

Table 2. Domain replications of the ontology test.

#	Domain (regulatory philosophy)	Vertical	Approval structure	New core abstractions	Files modified
1	Health (clinical safety)	Patient education	1 key: clinician	0 (baseline)	—
2	Financial services (financial conduct)	Investment research	2 keys: principal + compliance	0	0
3	Legal (legal liability)	Research memorandum	1 key: attorney	0	0
4	Insurance (contractual obligation)	Claims explanation	1 key: adjuster	0	0

#	Domain (regulatory philosophy)	Vertical	Approval structure	New core abstractions	Files modified
5	Pharmaceuticals (drug regulation)	Drug information	1 key: pharmacist	0	0
6	Aviation (safety engineering)	Maintenance brief	1 key: licensed engineer	0	0
7	Industrial safety (occupational safety)	Work procedure	1 key: safety officer	0	0
8	Government (administrative law)	Citizen information	1 key: authorized officer	0	0
9	Human resources (employment law)	Policy guidance	1 key: HR officer	0	0
10	Critical infrastructure (operational resilience)	Operational procedure	2 keys: operator + shift supervisor	0	0
11	Education (formative assessment)	Formative feedback on a learner's geometric construction, in mathematics and technical drawing	1 key: teacher, under a school-signed enrolment	0 domain-specific	delivered as a signed installable pack (§4.6)

Only four things vary per row of the first ten: the knowledge domain, the policy pack, the verifier set, and the approval authority — which is exactly what Finding 1 (composition without changing the abstraction) predicts and what Finding 3 (the abstraction generalizes) puts to the test. This is **architectural replication**: an abstraction applied independently ten times without core change is empirical evidence that it is robust. Two structural observations strengthen it beyond simple repetition: dual control recurs in a second philosophy (row 10: operator + shift supervisor) as pure policy data, confirming it was never finance-specific code; and every domain's approvals are provider-backed (licence-verification receipts), so each replication also re-exercises the licence-bound-approval mechanism (Finding 2b, gap G2).

The eleventh domain. Earlier editions of this paper closed this section with a challenge to ourselves: *an eleventh domain remains a fresh trial.* The trial has been run. Education arrived, and it arrived in a stronger form than the first ten: not as a module in the repository but as signed data, through a general pack mechanism that knows nothing about education. The governed core needed no education-specific code. The domain-pack thesis — one governed core, specialized by declaration rather than rebuilt — is established for the domains we have implemented.

What the replication shows, exactly. Rows 3 to 10 are *proof verticals*: compact policy packs with three deterministic verifiers each, built to a shared template to isolate one question, whether the core must change, and the answer was no, eight times in a row. Education is a working product built on the format. All eleven were built by the author's team; what they establish is the domain-independence of the architecture, and the invitation below opens the next question to everyone else.

The invitation. The installable format turns the next experiment into an open one. A twelfth and a thirteenth domain, authored by someone who has never seen the core, is now a matter of building a pack and declaring its conformance level. Every pack that installs and governs without a change to the authority path is a further independent replication of Finding 3, and the format tells its author precisely what they have shown.

6.3 Adversarial and fault-injection testing

Ten drills inject real failures against real components and assert guarantees rather than recovery: KMS outage (no unsigned evidence; pre-outage chain intact), identity-provider outage (no receipt → no grant → no certificate), licence revoked mid-approval (caught at approval instant), qualification expiry (visible; production gate refuses), WORM outage (release aborts before any byte is written), permit replay (denied, and the denial recorded), policy supersession mid-flight (divergence detectable via ledger), forged signer under a trusted name (rejected by the trust directory), evidence-chain reordering (chain verification fails), and approval clock skew (approval binds one decision instant). The drill-report machinery is itself tested for honesty: a deliberately violated guarantee is recorded as a failed outcome in the signed report, and the production gate refuses stale, partial, unproven, and wrongly-signed reports.

6.4 Security-Relevant Negative Tests & Multi-Tenant Isolation

Beyond the fault-injection drills, the test suite verifies the security kernel against adversarial failure modes, privilege escalations, and cross-tenant leakage:

- **Authority & Signature Forgery**: Forgery of each authority under its correct name is rejected via directory trust-anchor pinning; certificate replay against re-timed or content-tampered decisions is refused via requirement/record digest binding; single-person dual-control attempts are blocked; and self-approval attempts fail closed.
- **Multi-Tenancy by Construction & Negative Isolation**: The current implementation carries tenant identity through authorization, credential lookup, and persistence (`factory/tenant.py`, threaded from commit `fe495990a`, 31 May 2026). Named tests cover specific tenant-boundary refusals, including cross-tenant approval refusal (`tests/test_command_service.py`: `test_cross_tenant_approval_is_refused`), indistinguishability from missing (`tests/test_calendaring_service.py`: `test_cross_tenant_is_indistinguishable_from_missing`), cross-tenant project mutation barriers (`tests/teaching_production/test_api.py`: `test_other_tenant_cannot_read_mutate_or_export_project`), fail-closed cross-tenant run ID validation (`tests/test_coding_run_state.py`: `test_file_state_fails_closed_for_cross_tenant_run_ids`), and database row-level tenant separation in the public reference suite (`tests/test_regulated_persistence_pg.py`: `test_database_enforces_tenant_isolation_and_append_only_for_application_role`); test-file inventory is not an executed isolation-test total. The PostgreSQL application-role test requires a configured database and is skipped in the reported offline subset run; when a PostgreSQL instance is supplied, that test executes and passes (raising the executed test suite from 222 to 235 passed). The regulated PostgreSQL adapter defines tenant Row-Level Security (RLS) policies across evidence tables (`regulated_core/persistence_pg.py`).
- **Two-Tier Bring Your Own Key (BYOK) Envelopes**: Supported model/agent authentication paths use customer-bound credentials and isolated authentication directories (`factory/harness_credentials.py`), failing loud with `MissingCredentialError` rather than falling back to shared credentials. Connector grants mediate account-credential use, while authenticated access to the platform's MCP server is a separate boundary; arbitrary customer external-MCP onboarding remains unverified in this baseline. At Tier 2, the signing backend is injectable; the reported operational evaluation did not exercise per-customer signing custody, evaluating the path under operator keys.

- **Defensible Architectural Boundaries & Threat Model**: We distinguish these verified mechanisms from demonstrated public multi-tenant SaaS scale. The system has been evaluated in a single-operator deployment with multi-tenant seams intact, and has not undergone external penetration testing. Furthermore, the platform threat model explicitly excludes a malicious or compromised platform host root; in-process memory and tenant context protections defend against agent-level privilege escalation and cross-tenant confusion, but do not protect against an adversary with host root access.

6.5 The projection-completeness finding

Constructing the reporting projections exposed a latent evidence-completeness defect: the emission permit — enforced everywhere (single-use, output-bound) — was never copied into the evidence pack. Enforced, but invisible to audit. The defect was found because a projection is a total function over the fields its view requires; it was fixed before merge. We report this as qualitative evidence for the projection design working as intended: rendering every stakeholder view forces every evidence field into the light.

6.6 Reproducibility and external verification

A single command (`scripts/reproduce_regulated.py`) re-derives this section's numbers from a clean checkout: it runs the architecture-specific test surface, executes all ten drills and prints their observations, composes the everything-wired stack on local adapters, releases one governed decision, renders all four projections (writing the artifacts to disk), and checks projection purity and attestation validity. Writing the artifact immediately paid for itself: its first draft embedded a qualification dataset that had drifted from the canonical one (recall 0.6667 vs. the cited 0.7500), which the artifact now prevents by construction.

Separately, the DSSE attestation is verified by *independent* tooling — `securesystemslib` 1.4.0, the reference implementation maintained by the in-toto/TUF community — which accepts the genuine envelope and rejects both a tampered payload and a signature from an attacker key under the correct identity name. "Designed to be externally verifiable" is thereby upgraded to "externally verified," with tool and version on record.

6.7 Operational measurement against real cryptographic modules

Sections 6.1–6.6 exercise every external seam against deterministic local adapters. This section reports the first operational measurement of the authority path against **real** cloud cryptographic infrastructure — the signing seam backed by a real hardware security module and a real managed key service, and the retention seam backed by real write-once storage. It answers the question the local evaluation cannot: what does real custody and real retention cost per governed decision, and does the runtime hold its guarantees against real infrastructure under sustained load?

Two signing profiles, one path. The drop-in-provider property of the signing seam (§5) is put to the test: the identical governed-decision path is run under two production signing profiles that differ only in the injected backend, with no change to any policy, gate, evidence, or canonical-encoding code.

- **STRICT** — ECDSA P-384 signed inside a *single-tenant* AWS CloudHSM cluster operating in FIPS mode (FIPS 140-3 Level 3, CMVP certificate #4703, Marvell LS2 HSM family). The module certificate lists ECDSA validation under FIPS 186-4; P-384 also appears in ETSI TS 119 312 v1.4.3 cryptographic suites (these references do not establish eIDAS qualification of this deployment).

- **MANAGED** — Ed25519 (EdDSA, FIPS 186-5) signed in AWS KMS, whose hardware security modules are FIPS 140-3 Security Level 3 validated. This profile trades single-tenant custody for a managed, per-request key service. The two profiles differ in tenancy and algorithm; FIPS-mode operation of the tested profile is claimed here for STRICT (CloudHSM) only.

Retention in both profiles is anchored to real S3 Object Lock (governance mode). Both are exercised end to end from an in-region client, so measured latencies reflect the module round-trip, not wide-area distance.

Retention profile. Production evidence requires S3 Object Lock **COMPLIANCE** mode, verified on each retained object version. The July 2026 evaluation used **GOVERNANCE** mode on a staging test bucket; those measurements do not qualify compliance-mode retention. Governance retention permits privileged bypass and does not satisfy the production requirement.

The price of custody and retention. Table 3 reports per-stage latency and throughput for 300 complete governed decisions per profile, measured first against local deterministic adapters (the baseline) and then against the real modules. The delta between the two is the measured operational cost of real custody and real retention per governed decision — a figure seldom published.

Table 3. Operational cost per governed decision: real modules vs local baseline (p50 ms, n=300).

Stage	MANAGED local	MANAGED — KMS (Ed25519)	STRICT local	STRICT — CloudHSM (ECDSA-P384, FIPS)
signing	1.3	15.4	4.7	27.7
retention (WORM anchor)	0.3	53.3	0.3	49.3
total per decision	9.9	82.1	16.2	92.1
throughput (decisions/s)	85.8	11.5	61.2	9.9

Saved 300-decision baselines report median total latency of 82.10 ms for managed signing and 92.13 ms for STRICT, with 95th percentiles of 130.71 ms and 162.98 ms. These are recorded-workload measurements, not maximum latency guarantees.

Two observations hold across both profiles. First, the per-decision cost of real infrastructure is dominated not by signing (15–28 ms of module round-trip) but by write-once retention (~50 ms per S3 Object Lock anchor): custody is cheaper than the durable, tamper-evident record it protects. Second, the runtime's behavior is identical under both algorithms — every governed decision fails closed identically and both profiles' signatures verify under the same trust fabric — confirming that the signing algorithm is a swap at the composition root, not a property of the authority path.

Resilience: custody is disposable and recoverable, measured. With the cluster's encrypted backup as the only persistent state ($0 while no HSM exists), a torn-down HSM was restored to a working, signing-capable state in a measured **372 seconds** (backup → available), after which the restored HSM signed and verified correctly — the non-extractable key material survived the backup/restore cycle intact. This turns "custody hardware is disposable and recoverable" from an assertion into a published recovery-time number.

Sustained operation. To measure endurance rather than a single burst, the STRICT profile ran continuously for **24 hours** against the FIPS-mode HSM: **854,100 governed decisions across 2,847 batches, zero failed batches** — the 24 heartbeat receipts, at intervals of 3,222–3,633 seconds, record between 31,200 and 38,100 decisions each, averaging 35,588, from the first receipt to the last. Every

decision traversed the full authority path (preflight, consequence policy, approval binding, permit issuance, signed hash-chained evidence, write-once anchor) and produced a signed evidence record on write-once storage. Sustained operation at this scale, maintaining steady throughput across the 24 heartbeat intervals with no unsigned or unanchored evidence, is the operational counterpart to the correctness the test suite establishes. As documented in the reproducibility release notes, these July 14–15 operational run records establish hardware-backed execution against physical cloud cryptographic infrastructure, but do not bind an executed source git commit.

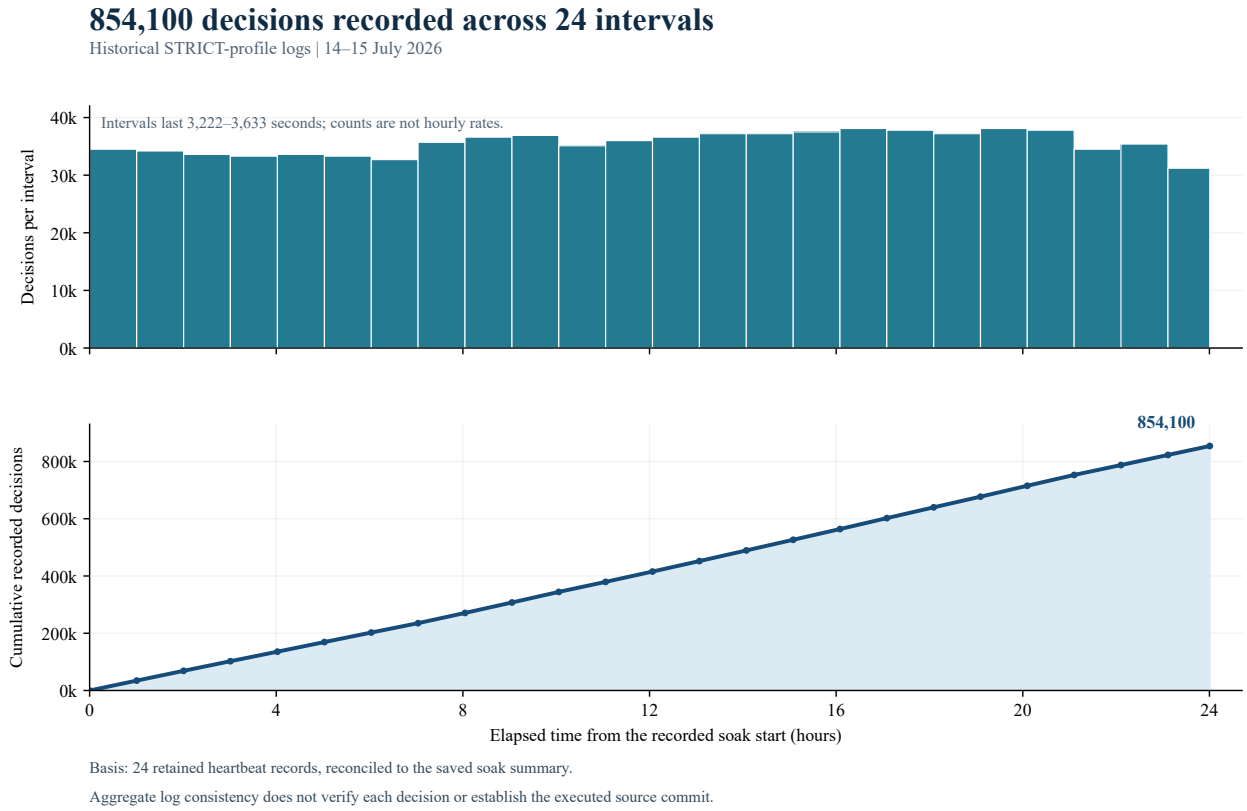


Figure 7. The 24-hour run against the FIPS-mode hardware security module: 854,100 governed decisions, 2,847 batches, none failed, throughput steady across the 24 heartbeat intervals.

Figure 7 in words. A chart of the 24-hour soak, one measured deployment. Bars for the 24 heartbeat intervals (3,222–3,633 seconds each) range from 31,200 to 38,100 governed decisions, about 35,600 on average. A line rises steadily from zero to the cumulative total: 854,100 governed decisions.

Fail-closed against real infrastructure. The fail-closed guarantee established against induced local outages (§6.3) was exercised against real infrastructure on both dependency classes. For storage: during a live run, the client's write permission to the WORM bucket was revoked, and the runtime aborted the release before any byte was emitted, surfacing the retention failure as `WormUnavailable` — no receipt, no output, no unsigned or unanchored evidence. For the signer: a live STRICT run signing through the single-tenant CloudHSM anchored 612 governed decisions, at which point the client's network path to the module was severed at the subnet boundary — a stateless network-ACL deny, which drops packets on an already-established flow rather than merely closing new ones. The operator's note records that, after 30 seconds, the next decision failed closed at the signing gate as `KmsUnavailable`, the prior hash-chain intact and no unsigned or unanchored evidence appended past the outage. Both dependency classes converge on the same single fail-closed chokepoint (§6.3): the release is refused before emission, never after.

6.8 The Education pilot: the first domain built on the installable format

Education is the first domain built end to end on the installable pack format. The signed Education pack (`lumiwright.education.formative-feedback`) carries subject vocabulary, data schemas, workflow stages, review rules, approved-source references and conformance cases. It has shipped three signed versions, 0.1.0, 0.1.1 and 0.1.2, each with build provenance and a digest in the deployment catalogue (`deploy/packs/`), which exercises the lifecycle of §4.6 on a real upgrade path; 0.1.2 is the version activated in the hosted pilot, with a readback in the release contract.

A teacher-governed workbench (`factory/education_workbench/`) lets a learner construct and explain a geometric construction, the meeting point of mathematics and technical drawing, save and submit a revision, and a teacher in the same class add separate annotation. Identity and role come from the host, never from the pack and never from the browser: a school authority signs each enrolment as a versioned policy document, expiry and revocation are checked on every request, and no client can promote a role. Every submission retains its draft digest, its geometry digest and the pack's authority pins; every annotation binds the submission it answers.

The pilot demonstrates two things. First, the **pack law in practice**: a new domain arrives as signed data with its own vocabulary and rules while the governed core stays as it was. Second, the **host-authority boundary**: the pack declares *who may approve*; the organization proves *who that is*. Six industry starter packs — Education, Health, financial onboarding, insurance complaints, ICT supplier oversight and public procurement review — are specified with primary legal sources and jurisdiction profiles (South Africa, the United Kingdom, the European Union, and a United States expansion reference), and are the natural subjects of the invitation in §6.2.

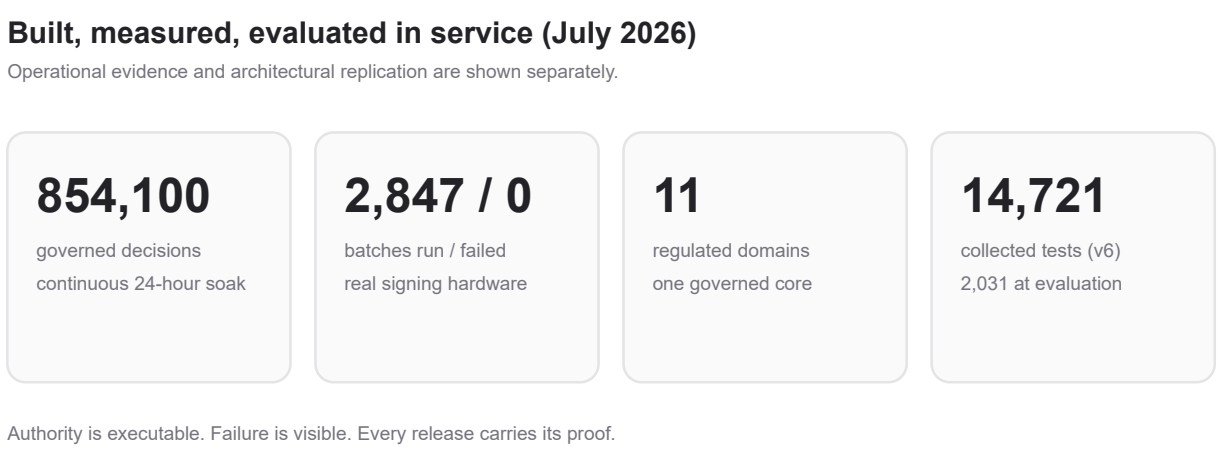


Figure 8. Built, measured and evaluated in service (July 2026): the operational evidence and the architectural replication, side by side.

Figure 8 in words. Four figures in a row. 854,100 governed decisions in a continuous 24-hour soak. 2,847 batches run and zero failed, on real signing hardware. Eleven regulated domains on one governed core. 14,721 collected test inventory at Version 6, from 2,031 passed at the evaluation snapshot. Beneath them: authority is executable, failure is visible, every release carries its proof.

6.9 Generality Across Workload Kinds: Attempted Agentic OS Abstraction

Alongside the high-assurance authority gate, the platform was developed as an attempt to define an **Agentic Operating System** and evaluate whether a single, minimal runtime abstraction could coordinate divergent workload categories. The system was evaluated across two operational planes:

1. **High-Assurance Regulated Plane**: The eleven regulated domains (§6.1–6.2) requiring fail-closed authority gating, professional credential verification, dual-control policy, and hardware cryptographic signing.
2. **Creative, Multimedia & Autonomous Delegation Plane**: High-capability workflows executing under consequence-tiered policy without regulatory friction, demonstrated across reference workload categories:

- **Deterministic Video Editing**: Automated long-form to short-form video editing, evaluated against the operational workflow pattern of OpusClip as a reference standard (without claiming algorithmic parity with proprietary commercial engines), orchestrating audio transcription, silence detection, operator review, and an automated cut-plan validation gate that strictly refuses cut lists bisecting speech boundaries or exceeding retention floors prior to ffmpeg and FCPXML timeline rendering.
- **Multi-Modal Animation & Spatial Explanation**: As part of the operating system generality effort, a separately developed explanation studio (Lumi Studio) was integrated through teaching and persistence adapters. Across the specified September 8 integration snapshots (commits `d867f76`, `93d61d2`, `393cd1f`), the Project OS engine file (`factory/projectos/engine.py`) and `regulated_core` tree were unchanged, while host wiring and workload-specific components were added. This demonstrates reuse at those specific boundaries, rather than a claim that all workload code or infrastructure remained invariant. Avatar presentation references standards such as HeyGen.
- **Autonomous Assistance & Multi-Agent Delegation**: General-purpose personal assistant orchestration (Assistant shell referencing OpenClaw, 6,921 lines) and hierarchical organizational task delegation (Paperclip and Delegation architectures, 6,268 lines).

The platform demonstrates that authority governance is not a vertical application silo, but an orthogonal runtime layer that can be applied across general-purpose agentic systems.

---

7. What Is Established, and the Road Ahead

Established. An enforceable authority path with verifiable evidence, implemented, deployed and measured. Eleven regulated domains on one governed core. A signed, installable domain-pack format with a working product built on it. Independent verification of the machine attestation by community tooling. Real cryptographic module custody under two signing profiles (managed AWS KMS Ed25519 and CloudHSM FIPS 140-3 Level 3 ECDSA P-384) evaluated across 300-decision baselines, with real write-once retention; sustained 24-hour endurance testing under the CloudHSM STRICT profile across 854,100 decisions with zero failed batches; fail-closed enforcement on real infrastructure when signing or retention was interrupted; and CloudHSM custody recovery from backup in 372 seconds.

The division of labour. The runtime enforces controls and produces evidence. The institution interprets its regulation, assesses conformity, runs its management system and incident operations, and remains accountable for human judgment. The architecture is designed around that division: it never simulates an

organizational control, and every attestation shows which controls the runtime proved and which the organization supplied. That is what makes its evidence usable in front of a regulator.

The road ahead. Each item extends a result that already stands; none is a precondition for it.

1. **Authoritative identity and licence integrations.** The interfaces, the receipts and the fail-closed behaviour are implemented and drilled against deterministic adapters, and the Education pilot already runs host-signed enrolment as its authority source. Connecting national licence registries and enterprise identity providers is adapter work behind a proven seam.
2. **Probabilistic verifiers.** Today's verifiers are deterministic and measured. The qualification schema reserves per-call confidence and calibration for them, so calibrated probabilistic verifiers with drift monitoring arrive as an addition.
3. **Larger validation corpora.** The golden sets are honest engineering sets that include known misses. The same content-addressed mechanism scales to regulator-approved corpora without a change in design.
4. **Distributed fault injection.** Custody and retention have been cut on real infrastructure and the runtime failed closed. Network partitions, duplicate delivery and key-rotation overlap extend the same drill catalogue.
5. **Independently authored packs.** The format and its conformance levels are published in the repository; §6.2's invitation stands.
6. **Certified production deployments.** The platform was deployed and the authority path was operationally measured; certification is the adopting institution's step, and the evidence it needs is what this runtime produces.

---

8. Conclusion

Agentic AI does not need another layer of governance documentation attached after execution. It needs a dependable runtime foundation that can decide whether a consequential action is authorized, refuse the action when proof is incomplete, and preserve the evidence when release is permitted.

Executable Assurance provides that foundation as an optional governance plane over a general-purpose agent platform. It combines consequence-aware policy, qualified automated verification, credentialed human approval, output-bound permits, signed evidence, write-once retention, and stakeholder-specific projections in one enforced path. Ten domain proofs exercised the same machinery without adding a new core abstraction, and an eleventh, Education, arrived as a signed installable pack with a working product built on it. Independent attestation verification, real cryptographic-module measurements and a managed cloud deployment make this an implemented, deployed and operationally measured system.

The broader implication is a platform shift. Healthcare, finance, government, aviation, and other regulated domains do not need identical policies, but they can inherit the same trustworthy machinery. Governance can be built once as infrastructure, specialized through domain packs, and improved for every application that shares the platform. That changes the question from *how do we bolt governance onto this AI application?* to **what policy and authority must this application inherit before it may act?**

The claim is deliberately precise: this architecture does not certify an AI system or remove human accountability. It makes authority executable, failure visible, and every released decision traceable to the evidence that permitted it. Where actions are consequential, it keeps human accountability exactly where regulation places it: with a named, credentialed person.

Not smarter AI. Trustworthy action, with proof.

An invitation from a thousand directions

This door will be released as open source. It is not a complete solution, and we do not claim it is the answer. Amid the fear about where AI is heading, it shows that something can be done. In James Reason's Swiss-cheese model, safety comes from many independent layers, each with its own holes, arranged so that the holes do not line up. The authority path is one such layer: the preventative gate between an autonomous proposer and a real-world effect.

The invitation is not to attack this layer, but for each reader to ask what layer their own skill can build: in systems security, formal methods, safety engineering, law, human factors, domain practice and beyond. The problem can then be attacked from a thousand directions. Each contribution should state what it prevents, what it assumes, how it can fail and how its effectiveness can be tested, so that the layers can be integrated into one multi-layered defence whose protections cover one another's gaps.

Author responsibility

The author is responsible for the architecture, claims, evidence interpretation and final manuscript.

Evidence and Reproducibility

The current system release is tagged at `v1.9.0` (commit `4e35a6811d3e38cc560473d01e7bcb3eefff00c0`). The public reproducibility repository is github.com/ernest-chiweshe/executable-assurance-reference. Walkthrough videos are at lumiwright.com/open-source/videos. The companion paper, *Capability Is Not Authority* (doi.org/10.5281/zenodo.23580974), sets out the human stakes, the Six-Ring Assurance Architecture, the authority and evidence mechanisms, the risk mapping, and the institutional path. The repository provides the standalone reference implementation of the authority gate, verification contracts, and 149 reproduced unit and fault-injection tests for independent execution. Operational evidence includes the dated 24-hour CloudHSM soak logs (sustaining 854,100 decisions across 2,847 batches with zero failed batches; `soak.log`), controlled 300-decision strict latency benchmarks (median p50 latency 92.13 ms, p95 162.98 ms), disaster recovery timing (372 seconds), exported public trust anchors, and DSSE attestation envelopes verified via `securesystemslib` 1.4.0.

Earlier October reference evidence (historical snapshot). The separate private reference exercises Education and Health workflows with synthetic identities. Signed evidence precedes permits; qualification report digests and expiry bind to the permit path. A delegated PostgreSQL receiver demonstrates shared-budget settlement, replay refusal and recovery at two receiving-worker interruption points. Version 0.1.5 passed all 58 tests, zero skips, in an independently created Windows/Python 3.12 environment installed offline. An independent reviewer also passed all 58 tests on a separate Ubuntu/Python 3.14 machine in

three new environments: source installed from a fresh clone, the reviewed wheel, and hash-locked dependencies installed without a package index. These add reproduction evidence to the existing platform. Index-free installation does not mean the host network was disconnected. The isolated logical restore drill ends in HOLD; safe authority resumption after restore remains OPEN.

References

1. SCSC Assurance Case Working Group. (2021). *GSN Community Standard Version 3*. Safety-Critical Systems Club. https://scsc.uk/r141
2. ISO/IEC/IEEE. (2022). *ISO/IEC/IEEE 15026-2:2022 Systems and software engineering — Systems and software assurance — Part 2: Assurance case*.
3. DARPA. (n.d.). *Automated Rapid Certification of Software (ARCOS)*. https://www.darpa.mil/research/programs/automated-rapid-certification-of-software
4. Wei, Ran; et al. (2024). *ACCESS: Assurance Case Centric Engineering of Safety-critical Systems*. arXiv preprint arXiv:2403.15236. https://arxiv.org/pdf/2403.15236
5. University of York, Assuring Autonomy International Programme. (2021). *Guidance on the Assurance of Machine Learning in Autonomous Systems (AMLAS)*.
6. UK AI Security Institute. (n.d.). *Safety cases at AISI*. https://www.aisi.gov.uk/blog/safety-cases-at-aisi
7. Goemans, Arthur; et al. (2024). *Safety case template for frontier AI: a cyber inability argument*. arXiv preprint arXiv:2411.08088. https://arxiv.org/abs/2411.08088
8. Porter, Zoe; Habli, Ibrahim; McDermid, John; Kaas, Marten. (2023). *A principles-based ethics assurance argument pattern for AI and autonomous systems*. AI and Ethics. https://link.springer.com/article/10.1007/s43681-023-00297-2
9. National Institute of Standards and Technology. (2024). *Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile*. NIST AI 600-1. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf
10. ISO/IEC. (2023). *ISO/IEC 42001:2023 Information technology — Artificial intelligence — Management system*.
11. European Union. (2024). *Regulation (EU) 2024/1689 (Artificial Intelligence Act)*, Article 11 and Annex IV. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689
12. RTCA. (2011). *DO-330: Software Tool Qualification Considerations*.
13. ISO. (2018). *ISO 26262-8:2018 — Road vehicles — Functional safety — Part 8: Supporting processes*.
14. U.S. Food and Drug Administration. (2025). *Marketing Submission Recommendations for a Predetermined Change Control Plan for Artificial Intelligence/Machine Learning (AI/ML)-Enabled Device Software Functions*. https://www.fda.gov/regulatory-information/search-fda-guidance-documents/marketing-submission-recommendations-predetermined-change-control-plan-artificial-intelligence

15. Mitchell, Margaret; et al. (2019). *Model Cards for Model Reporting*. https://arxiv.org/abs/1810.03993

16. Gebru, Timnit; et al. (2021). *Datasheets for Datasets*. Communications of the ACM, 64(12), 86–92. https://arxiv.org/abs/1803.09010

17. Birgisson, Arnar; et al. (2014). *Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud*. Network and Distributed System Security Symposium (NDSS). https://research.google/pubs/macaroons-cookies-with-contextual-caveats-for-decentralized-authorization-in-the-cloud/

18. biscuit. (n.d.). *Biscuit: public-key, offline-attenuable authorization tokens with Datalog policies*. https://www.biscuitsec.org/

19. SPIFFE. (n.d.). *SPIFFE: Secure Production Identity Framework For Everyone*. https://spiffe.io/

20. in-toto project. (n.d.). *in-toto Attestation Framework*. https://in-toto.io/

21. secure-systems-lab. (n.d.). *DSSE: Dead Simple Signing Envelope*. https://github.com/secure-systems-lab/dsse

22. SLSA. (n.d.). *SLSA: Supply-chain Levels for Software Artifacts, v1.2 provenance*. https://slsa.dev/spec/v1.2/distributing-provenance

23. Sigstore. (n.d.). *Sigstore: keyless signing and transparency for software artifacts*. https://www.sigstore.dev/

24. Tuesca, Facundo. (2025). *Catching Malicious Package Releases Using a Transparency Log*. OpenSSF Blog. https://openssf.org/blog/2025/12/19/catching-malicious-package-releases-using-a-transparency-log/

25. Coalition for Content Provenance and Authenticity. (2026). *C2PA Technical Specification v2.4*. https://spec.c2pa.org/specifications/specifications/2.4/specs/C2PA_Specification.html

26. OPA. (n.d.). *Open Policy Agent*. https://www.openpolicyagent.org/

27. Amazon Web Services. (n.d.). *Cedar: an open-source policy language and authorization engine*. https://www.cedarpolicy.com/

28. Hadarean, Liana; Tristan, Jean-Baptiste. (2026). *Why Policy in Amazon Bedrock AgentCore chose Cedar for securing agentic workflows*. AWS Security Blog. https://aws.amazon.com/blogs/security/why-policy-in-amazon-bedrock-agentcore-chose-cedar-for-securing-agentic-workflows/

29. LangChain. (n.d.). *Human-in-the-loop documentation*. https://docs.langchain.com/oss/python/langchain/human-in-the-loop

30. MCP. (n.d.). *Model Context Protocol Specification*. https://modelcontextprotocol.io/specification/2025-11-25

31. NVIDIA. (n.d.). *NeMo Guardrails*. https://github.com/NVIDIA-NeMo/Guardrails

32. Meta. (2024). *Llama Guard 3 8B Model Card*. https://github.com/meta-llama/PurpleLlama/blob/main/Llama-Guard3/8B/MODEL_CARD.md

33. HL7 International. (n.d.). *FHIR Package Specification*. https://hl7.org/fhir/packages.html

34. NIEMOpen, an OASIS Open Project. (n.d.). *National Information Exchange Model*. https://niemopen.org/

35. NVIDIA. (2026, October 21). *NVIDIA Launches OpenShell Reference Architecture for Safe AI Agents* [Press release]. https://nvidianews.nvidia.com/news/openshell-reference-architecture