Reliance Before Closure

Evidence-Qualified Stability for Machine Decisions Under Unresolved Effects

Moon Lee · RISU Institute · August 2026
DOI: 10.5281/zenodo.22038494

Abstract

A workflow can remain unfinished after the effects relevant to one downstream decision have become bounded. Formal methods can test stability over a stated set of futures; a live system must also justify that set for use by another machine. Reliance Before Closure specifies this evidence-to-reliance handoff. Provider-specific qualification uses operational evidence and explicit assumptions to establish a claim-relative future boundary under a named trust profile. For the frozen scalar-threshold profile, the resulting certificate carries a stability margin, while a proposed action is separately bounded by its possible movement toward the verdict's flip boundary. A provider-blind relying function then checks semantic identity and margin preservation, with unknown unbounded paths failing closed. Kubernetes commissioning exercised mutation control, action qualification, concurrency, and controller timing. An initial mechanism remained REVIEW. In a later trust-scoped execution, the relying function returned SUPPORTED before the subsequent Pod creation succeeded and before broader Deployment closure. The archived evidence and offline verifier reconstruct the boundary, margin, action bound, relying decision, commit evidence, and closure. The result is a narrow existence claim: claim-relative machine reliance can precede workflow-wide closure when the relevant future boundary and proposed action are qualified under the same evidence and trust bindings.

Keywords: runtime verification; runtime assurance; evidence qualification; operational evidence; decision stability; relying parties; interoperability; Kubernetes

1. Introduction

Machine workflows often treat broad completion as the point at which downstream systems may rely on a result. A particular downstream decision may depend on only a subset of the effects that remain unresolved. A rollout, for example, can still be stabilizing after the effects relevant to a separate resource decision have become bounded. The practical question is claim-relative: which unresolved effects can still change the decision another machine is about to make?

Formal methods already address whether a verdict remains invariant over a stated set of future states. In a live system, the set itself must be justified. An enforced bound may still be mutable, and an observed control surface may omit a relevant path. Downstream reliance therefore depends on evidence that qualifies the future boundary and keeps explicit assumptions and unresolved unknowns visible under the relying party's trust profile.

Reliance Before Closure defines an executable interface for that handoff. Provider-specific qualification uses operational evidence and explicit assumptions to produce a verified stability certificate carrying a verdict and margin. A proposed action is qualified separately by a bound on its claim-worsening effect. The relying function consumes that certificate, the action bound, and the relying context; it returns SUPPORTED only when their semantic identities match and the action remains within the certified margin. Provider-specific semantics remain upstream.

This note defines the contract and tests whether its prerequisites can be obtained and composed in a live system. The commissioning exercises mutation boundaries, action qualification, concurrency, and rollout timing before the final composition. In one trust-scoped Kubernetes execution, the relying function returned SUPPORTED and the action committed before broader Deployment closure; the frozen release permits offline reconstruction of the recorded result.

2. Why Stability Is Not Enough

Several lines of work already solve parts of the problem addressed here. Relative information completeness asks whether a query answer remains invariant across all extensions permitted by a partially closed database [1]. Finite-trace runtime verification can issue definitive verdicts before a trace ends, while assumption-based and anticipatory variants reason about unobserved or future behavior under explicit models and assumptions [2]-[4]. Distributed snapshot theory provides a classical account of stable properties whose truth persists over reachable successor states [5]. Robust optimization studies feasibility over specified uncertainty sets [6], and invariant confluence characterizes when application invariants can be preserved without coordination [7].

2.1 What existing work already establishes

These results settle much of the logic and mathematics that an early reliance decision may use. The remaining systems question is how a live system justifies the inputs to that reasoning.

Complete mediation contributes a different constraint. By analogy to its requirement that every access to every object be checked for authority [8], an effect-exclusion claim is credible only if relevant claim-worsening effects are forced through the surface being qualified. Where such a surface exists, the problem shifts from enumerating every actor to qualifying the surface and its mutation paths.

Table 1. Established results and the remaining operational handoff

Line of work	Established result	Operational handoff
Relative information completeness [1]	Query invariance over permitted extensions	Why those extension constraints are operationally justified now
Runtime verification [2]-[4]	Early verdicts over continuations constrained by models and assumptions	How the model, assumptions, and observations are qualified for use
Stable-property detection [5]	Detection of properties that remain true over reachable successor states	Which successor relation is justified for this claim at reliance time
Robust optimization [6]	Worst-case feasibility over a specified uncertainty set	How that uncertainty set is evidenced, bounded, and identified
Coordination avoidance [7]	Conditions under which invariants permit correct execution without global coordination	Reliance when the relevant future is qualified but broader workflow closure remains open
RATS [9]	Vendor-specific Evidence can be appraised into Attestation Results consumed under a Relying Party's own policy	Semantics for unresolved claim-worsening effects and the future boundary carried to the relying party
Proof-carrying code [10]	A producer supplies code plus proof of compliance with a predefined safety policy for the host to check	Qualification of live operational futures and action effects rather than code safety

Recent agent-systems work makes neighboring concerns concrete. Cordon stages and validates task-level effects before commit [11]. Proof-Carrying Agent Actions binds runtime-neutral governance evidence to portable action certificates and envelopes [12]. Outcome Finality argues that effects able to change a scored outcome must be resolved, bounded, or retained as uncertainty before a final label is justified [13]. Mission-level runtime assurance separates mission verdicts from evidence completeness so that missing evidence becomes explicit unknown rather than an unsupported all-clear [14]. These systems make effects, evidence, and portable decision artifacts explicit. The remaining handoff here is narrower: qualifying heterogeneous operational evidence into a claim-relative future boundary, binding that boundary to a proposed action and relying context, and letting a provider-blind consumer use the result without importing provider-specific semantics.

2.2 What operational evidence must establish

That handoff can fail even when the downstream stability calculation is correct. A limit can be enforced now but mutable before the relying action matters. An observed control surface can omit a bypass. Non-observation can be mistaken for evidence of absence. A provider summary can be stale or too broad for the claim. Multiple records can describe one physical effect. An action can induce later claim-worsening effects larger than its immediate delta. Two actions can each fit the same margin even though neither has reserved it.

Table 2. Failure modes at the evidence-to-reliance boundary

Failure	Unsafe inference	Required treatment
Mutable hard bound	current limit = future limit	Qualify relaxation and mutation paths; otherwise fail closed
Coverage gap	observed sources = complete effect surface	Establish claim-relative coverage or retain uncovered paths as unknown
Unsupported absence	not observed = cannot occur	Keep evidence, explicit trust assumptions, and unknown effects distinct
Stale or overbroad witness	provider status = current claim-specific fact	Qualify freshness, authority, and semantic relevance for the claim
Duplicate evidence	two records = two effects	Preserve effect identity before aggregation
Action-induced expansion	immediate delta = total action effect	Bound claim-worsening post-action futures, not only the direct delta
Concurrent reuse	one action fits = margin is owned	Treat the result as descriptive; reservation and commit discipline remain external

None of these failures is repaired by a correct stability calculation. They concern either whether the future boundary supplied to that calculation is justified or whether the resulting margin is validly applied to an action. The profile therefore keeps two claims separate: a verdict is stable over a stated future boundary, and the available evidence is sufficient to make that boundary admissible under the stated trust profile. The mathematics determines stability once the boundary is given. The systems problem is whether another machine has enough evidence to use that boundary at all.

3. Evidence-Qualified Reliance

The frozen v0.1 profile separates three operations: qualify a claim-relative future boundary, derive a stability certificate for the scalar threshold profile, and compose that certificate with a separately qualified action bound and the relying party's accepted context. The provider-blind rely() function performs only the final composition. It does not discover paths, establish authenticity, or recompute provider-specific evidence.

3.1 Qualifying the Future Boundary

Let F denote the set of claim-relevant future states consistent with the qualified boundary under the accepted trust profile. F is explanatory notation, not a set that rely() enumerates. Upstream producers may justify the boundary with enforcement, coverage, mutation controls, residual bounds, source qualification, and explicit assumptions.

For the declared claim/scope inventory, every identified claim-worsening path has exactly one disposition: INCLUDED, EVIDENCED_EXCLUDED, EXPLICIT_TRUST_ASSUMPTION, or UNKNOWN_UNBOUNDED. No inventoried path may silently disappear. An assumption remains an assumption; it can condition a positive result only when the relying context accepts the same verified trust-profile identity. UNKNOWN_UNBOUNDED prevents boundaryStatus = PROVEN.

boundaryStatus = PROVEN means that the declared inventory and the evidence or assumptions needed to justify it passed the profile's qualification rules under the named boundary and trust identities. It does not assert universal closure or discovery of every physical future.

Boundary status and verdict stability are evaluated separately. A stable arithmetic result over an UNPROVEN, STALE, CONFLICTED, or OUT_OF_MODEL boundary is not reliance-grade.

3.2 Threshold Stability and Margin

For THRESHOLD_SLACK_V0, write the normalized claim as $q \le B$, where q is the scalar claim value at a future state and B is the fixed threshold. Let U and L denote the harmful extremes for preserving SATISFIED and VIOLATED, respectively. These are upstream derivation quantities; q, B, U, and L are not inputs to rely(). Worst-case extrema over a specified uncertainty set are conventional robust reasoning [6].

$$U = \sup \{ q(x) : x \in F \} \qquad L = \inf \{ q(x) : x \in F \}$$

For a SATISFIED certificate, a reliance-grade margin is $M = B - U \ge 0$. For a VIOLATED certificate, v0.1 consumes only $M = L - B > 0$; a certificate carrying $M = 0$ is rejected by the reference consumer.

$$\text{SATISFIED: } M = B - U \ge 0 \qquad \text{VIOLATED: } M = L - B > 0$$

The reference consumer carries M as stabilityMargin, a canonical nonnegative integer string in a declared atomic unit. The producer must express M exactly in that unit before the object is consumable; arbitrary decimal arithmetic is outside the frozen profile. The upstream derivation may still use richer state, residual bounds, correlations, coverage, and provider semantics. Compressing that reasoning to a decision-relevant summary is established abstraction practice [15]; here verdict plus M is sufficient only for this threshold-preservation check, and boundary/trust identity remains attached.

3.3 Action Composition and Reliance

A QualifiedActionBound carries adverseConsumption, denoted C_a, a nonnegative bound on movement toward the certified verdict's flip boundary over POST_ACTION_ADMISSIBLE_FUTURES. For SATISFIED this is upward movement in q; for VIOLATED it is downward movement. C_a must cover the direct action effect and any attributable future expansion in the adverse direction. If the action can invalidate the boundary or escape the qualified post-action scope, the producer cannot emit a finite PROVEN bound and the decision returns to requalification.

Let U_a and L_a denote the corresponding post-action harmful extremes. Qualification of C_a establishes the directional bound:

$$\text{SATISFIED: } U_a \le U + C_a \qquad \text{VIOLATED: } L_a \ge L - C_a$$

The provider-blind rule is therefore:

$$\text{SATISFIED: SUPPORTED iff } C_a \le M \qquad \text{VIOLATED: SUPPORTED iff } C_a < M$$

The inequalities directly preserve the verdict: $U_a \le U + C_a \le B$ in the SATISFIED case, while $L_a \ge L - C_a > B$ in the VIOLATED case. The strict VIOLATED comparison is required because equality reaches $q = B$ and flips the non-strict predicate to SATISFIED. Checking a candidate action against a safety margin is established safety-filter territory [16]; the narrower contribution here is the evidence-qualified, identity-bound interface that supplies this check across machines.

Before the arithmetic is used, certificate, action bound, and relying context must match exactly on claim identity, semantic profile, unit, boundary identity, and trust-profile identity. These identities are verified semantic bindings, not self-authenticating strings: authenticity, issuer trust, and binding to unchanged definitions are upstream preconditions.

The outputs are SUPPORTED, REQUALIFY, and NOT_APPLICABLE. SUPPORTED means only that this proposed use lies within the certified threshold margin under the matching identities. It grants no authorization, reserves no margin, and does not guarantee that a commit will succeed.

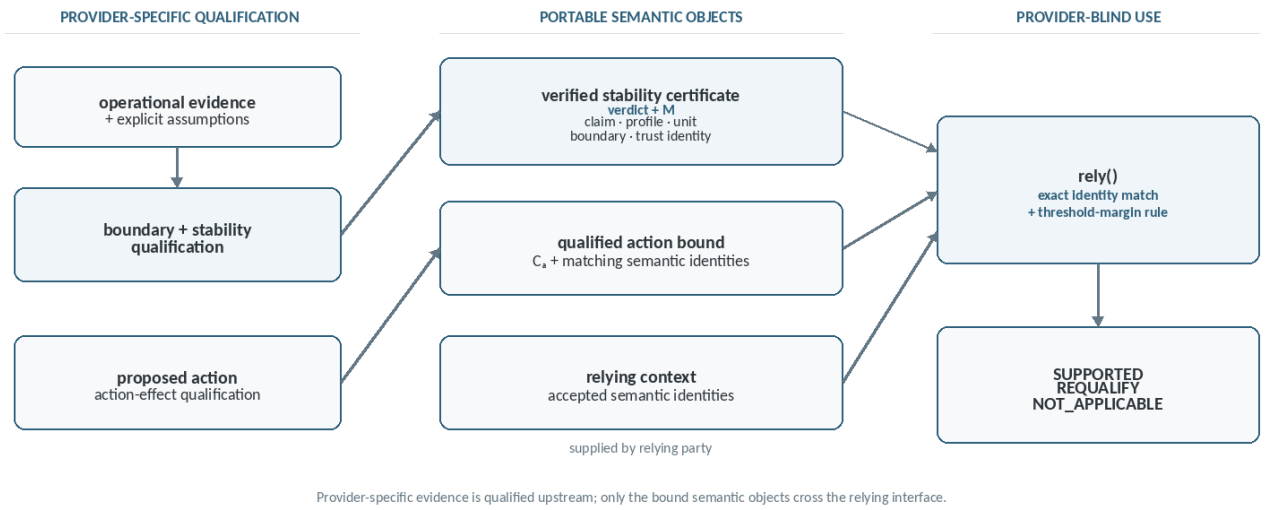

Figure 1. Evidence-qualified reliance architecture. Provider-specific qualification produces the verified stability certificate and qualified action bound; the relying party supplies the accepted context. The provider-blind rely() function performs exact identity checks and the threshold-margin rule only.

4. Empirical Commissioning

The commissioning asked whether the semantic prerequisites in Section 3 could be obtained from a live system and composed before broader workflow closure. The synthetic core had already been frozen under deterministic oracles and metamorphic laws. Kubernetes was used for properties that required an operational environment: enforcement, mutation authority, action normalization, concurrency, asynchronous controller state, and trust-scoped boundary evidence. The final commissioning run followed separate controls for these failure surfaces.

Table 3. Commissioning controls and the claims they constrain

Control	Observed result	Claim constrained
Mutable ResourceQuota	The same hard value was enforced while the evaluated identity could mutate it	A usable future bound also requires qualified mutation control
Restricted-actor boundary	Tested quota, rollout, and privilege-escalation mutation paths were unavailable to the actor	The enforced bound became usable relative to the declared actor/mutation boundary
Typed action + concurrency	Two one-slot actions separately qualified; ResourceQuota admitted only one concurrent commit	Margin reservation remains an external commit concern
Initial rollout mechanism	REVIEW; post-hoc physical separation was 0.596 s against a 10 s threshold	Controller scale-down semantics determined whether useful separation emerged

4.1 Lead-Time Mechanism Revision

The first lead-time commissioning used replicas = 4, maxSurge = 1, maxUnavailable = 1, and minReadySeconds = 20. The precommitted reliance predicate never fired. A post-hoc weaker physical predicate would have produced only 0.596 seconds of separation, well below the precommitted 10-second utility threshold. The official result remained REVIEW, and the configuration was preserved as run.

The trace showed why. With maxUnavailable = 1, the Deployment retained an old replica until enough new replicas became available, so creation-capacity release occurred almost at availability closure. The revised test changed the controller

mechanism and the witness used for the narrow claim: maxUnavailable = 4 allowed old replicas to disappear earlier, minReadySeconds = 20 preserved the availability-stabilization requirement, and direct Pod/ReplicaSet topology replaced the lagging aggregate Deployment count. The revised run retained the precommitted stop rule: a result below 10 seconds would end further Kubernetes tuning.

4.2 Single-Execution Composition

The final commissioning run used replicas = 4, namespace Pod quota hard = 5, maxSurge = 1, maxUnavailable = 4, and minReadySeconds = 20. For this Pod-count claim, ResourceQuota supplied the enforced admission bound: Kubernetes rejects requests that violate a configured quota and recommends restricting update or deletion of the quota when that enforcement must remain fixed [17]. The evaluated actor was denied the tested quota, Deployment, ReplicaSet, and RBAC mutation paths. The boundary remained TRUST_SCOPED because failure-induced replacement and undiscovered external-reconciler creation were explicit assumptions accepted by the trust profile.

At the first reliance point, the archived objects showed that the controller had observed the current Deployment generation (generation = observedGeneration = 2). They also showed four active rollout Pods, all on the new template, no old active or terminating rollout Pods, new ReplicaSet desired replicas = 4, old ReplicaSet desired replicas = 0, and broader Deployment completion = false. ResourceQuota status reported 4 of hard 5 as a consistency observation. The margin derivation used the four active claim-relevant Pods plus zero included adverse reserve under the accepted ledger, yielding M = 5 - (4 + 0) = 1 POD_SLOT.

The server-normalized candidate was a standalone, ownerless Pod with a fixed namespace and name. The typed producer derived C_a = 1 POD_SLOT. Certificate, action bound, and relying context matched on the five semantic identities required by Section 3.3, so the unchanged provider-blind rely() returned SUPPORTED. The actual CREATE then succeeded. Final state contained the four rollout Pods plus the auxiliary Pod, with ResourceQuota observed at 5 of 5 and no exceeded-quota event after the action.

Broader Deployment closure followed later. Kubernetes marks a Deployment complete when all replicas are updated and available and no old replicas remain; minReadySeconds controls how long a newly ready Pod must remain ready before it counts as available [18]. The archived closure object independently satisfied observedGeneration = generation = 2, updatedReplicas = 4, availableReplicas = 4, and Progressing=True with reason NewReplicaSetAvailable. Runner monotonic timing placed the reliance point at 0.697147 seconds and broader closure at 21.310634 seconds. minReadySeconds = 20 was deliberately used to make the separation observable. The measured 20.613487 seconds therefore serves only as a scenario-conditioned interval; the empirical result is the ordering of the two independently reconstructed frontiers.

At the reliance point, direct Pod/ReplicaSet topology already showed all four active rollout Pods on the new template and old ReplicaSets at desired zero, while Deployment.status.updatedReplicas still reported 1. The Deployment API defines status as the most recently observed state and updatedReplicas as the number of non-terminating replicas updated to the desired template [19]. The aggregate status remained a valid observation, yet direct topology was the stronger witness for this creation-closure claim at that instant. This source distinction is part of the qualification performed before the downstream margin calculation.

5. Reconstructing the Result

The frozen release provides a direct verification path from archived source evidence to the Section 4 result. The evidence bundle preserves raw provider objects, the boundary inventory and ledger, derivation records, semantic objects, timing and commit records, and an internal SHA-256 manifest. The manifest checks internal byte integrity; the offline verifier performs the semantic reconstruction. It recomputes all 27 manifest entries and reconstructs the qualified boundary, M = 1 POD_SLOT, C_a = 1 POD_SLOT, SUPPORTED, the committed action, broader Deployment closure, final quota 5/5, and the recorded 20.613487-second interval [20]. This is an offline reconstruction of the recorded execution. Re-running the live Kubernetes experiment is a separate procedure.

The relying semantics are also directly exercisable. The canonical zero-dependency Python rely() and a separate TypeScript implementation are checked against exact semantic fixtures [20]. Both consume only the verified stability certificate, qualified action bound, and relying context defined in Section 3.3. Kubernetes-specific qualification stays upstream, and the verified semantic identity bindings required by Section 3.3 remain a caller-side precondition.

6. Scope and Transfer Criteria

The empirical evidence establishes a narrow existence result for THRESHOLD_SLACK_V0: one live Kubernetes execution exhibited a claim-relative reliance point before broader Deployment closure under a named TRUST_SCOPED boundary, and the unchanged provider-blind consumer returned SUPPORTED before the action committed. The evidence reaches that

threshold profile, the commissioned standalone-Pod action profile, and that trust-scoped setting. Prevalence across workloads and portability across domains remain open questions.

In that run, boundaryStatus = PROVEN was trust-scoped: it depended on three explicit assumptions about the declared adverse-path inventory, failure-induced replacement after the reliance point, and undiscovered external-reconciler creation [20]. The accepted trust profile names those assumptions explicitly. Under the frozen rules, an UNKNOWN_UNBOUNDED path prevents boundaryStatus = PROVEN. Authenticity, issuer trust, and binding of supplied semantic objects to unchanged definitions remain upstream premises, as specified in Section 3.3.

The artifact establishes a necessary condition for portability: rely() contains no Kubernetes logic, and the commissioning used that consumer unchanged [20]. Cross-domain portability itself remains empirical. A same-domain independent rerun should reach the same consumer contract without semantic modification. A stronger transfer test would qualify evidence from a substantially different system into the same verified stability certificate, qualified action bound, and relying context. Success would require provider-specific reasoning to stay upstream while the frozen consumer contract remains unchanged. Repeated need for richer consumer semantics would indicate a narrower domain of applicability for v0.1.

7. Conclusion

This note isolates a smaller systems obligation than broad workflow completion: qualify the futures relevant to one claim, preserve the boundary and trust bindings of that qualification, and bound the proposed action against the resulting stability margin. The Kubernetes commissioning provides one trust-scoped execution in which those conditions were satisfied before broader Deployment closure. Reliance may precede closure, but only when the evidence for doing so survives inspection.

Artifact Availability

Reliance Inspector v0.4.0 [20] packages the frozen semantic consumer and evidence verifier, canonical cases, schemas, recorded evidence bundles, integration tests, validation records, and SHA-256 manifest used to inspect the result in this note.

References

[1] W. Fan and F. Geerts, “Relative Information Completeness,” ACM Transactions on Database Systems, vol. 35, no. 4, Article 27, 2010. doi:10.1145/1862919.1862924.

[2] A. Bauer, M. Leucker, and C. Schallhart, “Runtime Verification for LTL and TLTL,” ACM Transactions on Software Engineering and Methodology, vol. 20, no. 4, Article 14, 2011. doi:10.1145/2000799.2000800.

[3] A. Cimatti, C. Tian, and S. Tonetta, “Assumption-Based Runtime Verification with Partial Observability and Resets,” in Runtime Verification (RV 2019), LNCS 11757, pp. 165-184, 2019. doi:10.1007/978-3-030-32079-9_10.

[4] R. Hipler, H. Kallwies, M. Leucker, and C. Sánchez, “General Anticipatory Runtime Verification,” in Computer Aided Verification (CAV 2024), LNCS 14682, pp. 133-155, 2024. doi:10.1007/978-3-031-65630-9_7.

[5] K. M. Chandy and L. Lamport, “Distributed Snapshots: Determining Global States of Distributed Systems,” ACM Transactions on Computer Systems, vol. 3, no. 1, pp. 63-75, 1985. doi:10.1145/214451.214456.

[6] D. Bertsimas and M. Sim, “The Price of Robustness,” Operations Research, vol. 52, no. 1, pp. 35-53, 2004. doi:10.1287/opre.1030.0065.

[7] P. Bailis, A. D. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica, “Coordination Avoidance in Database Systems,” Proceedings of the VLDB Endowment, vol. 8, no. 3, pp. 185-196, 2014. doi:10.14778/2735508.2735509.

[8] J. H. Saltzer and M. D. Schroeder, “The Protection of Information in Computer Systems,” Proceedings of the IEEE, vol. 63, no. 9, pp. 1278-1308, 1975. doi:10.1109/PROC.1975.9939.

[9] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, “Remote ATtestation procedureS (RATS) Architecture,” RFC 9334, January 2023. doi: 10.17487/RFC9334.

[10] G. C. Necula, “Proof-Carrying Code,” in Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL ’97), pp. 106-119, 1997. doi:10.1145/263699.263712.

[11] Z. Chen, H. Liu, D. Xu, D. Dong, J. Li, B. Pu, and J. Zhai, “Cordon: Semantic Transactions for Tool-Using LLM Agents,” arXiv:2606.17573, 2026. Preprint.

[12] Z. Wang, “Proof-Carrying Agent Actions: Model-Agnostic Runtime Governance for Heterogeneous Agent Systems,” arXiv:2606.04104, 2026. Preprint.

[13] A. M. Casheekar, “When Is an Agent Evaluation Over? Outcome Finality and Cross-Unit Separation,” arXiv:2608.14940, 2026. Preprint.

[14] N. Kekatos, P. Katsaros, A. Lekidis, T. Nestoridis, and T. Nianios, “Mission-Level Runtime Assurance for LLM-Assisted ISR Swarms over a Verification-Aware Fabric,” arXiv:2607.23532v2, 2026. Preprint.

[15] P. Cousot and R. Cousot, “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints,” in Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL ’77), pp. 238-252, 1977. doi:10.1145/512950.512973.

[16] K.-C. Hsu, H. Hu, and J. F. Fisac, “The Safety Filter: A Unified View of Safety-Critical Control in Autonomous Systems,” Annual Review of Control, Robotics, and Autonomous Systems, vol. 7, pp. 47-72, 2024. doi:10.1146/annurev-control-071723-102940.

[17] Kubernetes Documentation, “Resource Quotas,” accessed August 20, 2026. https://kubernetes.io/docs/concepts/policy/resource-quotas/.

[18] Kubernetes Documentation, “Deployments,” accessed August 20, 2026. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

[19] Kubernetes Documentation, “Deployment,” apps/v1 API reference, accessed August 20, 2026. https://kubernetes.io/docs/reference/kubernetes-api/apps/deployment-v1/.

[20] M. Lee, “Reliance Inspector: A Local-First Research Instrument for Reliance Before Closure,” Version 0.4.0, Zenodo, August 20, 2026. doi: 10.5281/zenodo.22037607.