

Beyond Aggregate Metrics: Preserving Critical Behavioral Constraints in Production AI Agent Releases

Abhishek Kumar¹

¹Principal Cloud & AI Consultant & Independent AI Researcher, Hyderabad, India

Abstract - Production AI agents increasingly route work, select tools, construct arguments, maintain state, and initiate consequential actions. Evaluating only the final response, or summarizing many cases into an aggregate score, can therefore conceal failures that make a release unacceptable. This paper formalizes that condition as critical-failure dilution: an aggregate satisfies its acceptance threshold while at least one applicable release-critical behavior fails. To address it, the paper proposes an Agent Behavior Contract, a three-class consequence model, a non-compensatory Critical Constraint Preservation Rule, and a PASS-REVIEW-BLOCK release-decision function. A controlled synthetic retail-support experiment compared baseline and candidate agent configurations using deterministic constraint checks, descriptive model-based response evaluation, operational signals, and execution traces. The candidate improved its aggregate quality result from 0.791 to 0.993 and crossed the aggregate acceptance threshold, while five critical route failures and one critical tool-and-argument failure remained. The consequence-aware policy therefore returned BLOCK despite the aggregate PASS. Mandatory-escalation failures decreased from 65 to zero, showing that improvement in one critical class did not compensate for failures in another. Model-based response scores remained uncalibrated and are reported descriptively with human review required. The observed results satisfy the defined critical-failure dilution condition within the evaluated synthetic scenario and provide an explainable bridge between agent evaluation and release authorization.

Key Words: AI agents, agent evaluation, release assurance, quality gates, trajectory evaluation, AI governance.

1. INTRODUCTION

Artificial intelligence agents are evolving from conversational systems that generate recommendations into software entities that plan tasks, retrieve information, select tools, invoke application programming interfaces, coordinate specialized agents, and initiate actions in enterprise systems. This transition expands the scope of evaluation. For a conventional language application, assessing the generated response may indicate performance adequately. For a tool-using agent, the final response is only the visible result of a broader execution trajectory containing routing decisions, retrieved evidence, tool selection, argument construction, state transitions, policy checks, escalation choices, and external side effects.

An agent can therefore produce a plausible, well-written answer while following an unacceptable execution path. A support agent may correctly interpret a refund request and communicate confidently with the customer yet invoke a refund tool beyond its delegated limit instead of escalating the request for human approval. The response may score highly for fluency, relevance, and coherence while the underlying action

violates an operational constraint. Conversely, an agent may complete the correct action but express the outcome imperfectly. Response quality, execution correctness, and release acceptability are related but non-equivalent properties.

Existing approaches measure qualities such as task completion, relevance, groundedness, safety, tool-call accuracy, and instruction adherence [1], [2]. Tracing and observability expose inputs, outputs, tool calls, retries, latency, and intermediate execution steps [3], [4]. Custom evaluators can represent domain-specific requirements through code-based, prompt-based, or endpoint-based logic [5]. These capabilities improve behavioral measurement, yet a gap remains between measuring an agent and authorizing its release.

Evaluation systems commonly summarize cases through averages, pass rates, thresholds, or composite scores. Such summaries support trend analysis and candidate comparison, but they can suppress the consequences of individual failures. A high aggregate can coexist with an unauthorized tool call, an invalid argument, a missed approval, or a safety violation, because a large number of routine successes mathematically dilute a small number of severe failures. This paper defines that condition as a critical-failure dilution problem.

The central research question is: how can heterogeneous evidence drawn from agent responses, execution trajectories, tool interactions, and operational signals be converted into a defensible release decision without allowing aggregate performance to conceal high-consequence failures?

The paper proposes a framework with four contributions. First, an Agent Behavior Contract specifies expected task, route, tool, argument, escalation, outcome, safety, and operational behavior. Second, a consequence-classification model separates release-critical, review-required, and diagnostic findings. Third, a Critical Constraint Preservation Rule prohibits successful routine cases from compensating for a failed critical constraint. Fourth, a release-decision function produces PASS, REVIEW, or BLOCK while preserving evaluator uncertainty and evidence provenance.

The framework is examined through a synthetic retail-support scenario in which an above-limit refund must be escalated rather than executed autonomously. A contemporary agent platform is treated as an implementation and observability context because documented capabilities include agent evaluation, tracing, and custom evaluators [1], [3], [5]. The proposed contract, consequence classes, preservation rule, and decision function are application-level research constructs rather than vendor-defined product features or universal standards.

The objective is not to prove all future agent behavior. Datasets sample expectations, model-based judgements vary, operating conditions change, and previously unknown failures may emerge. The narrower objective is to make pre-release decisions more defensible by keeping high-consequence case-level evidence visible, distinguishing mandatory controls from

diagnostic quality, and recording why a candidate was passed, referred for review, or blocked.

The remainder of the paper is organised as follows. Section 2 reviews related work. Section 3 formulates the problem. Section 4 presents the proposed framework. Section 5 describes the research methodology. Section 6 defines the results and analysis plan. Section 7 discusses implications. Sections 8 and 9 state limitations and future work, and Section 10 concludes.

2. LITERATURE REVIEW

2.1 Output and Agent-System Evaluation

Language-model evaluation has traditionally emphasized generated-output characteristics such as correctness, relevance, groundedness, coherence, and safety. These remain necessary but become incomplete when an agent operates through multiple decisions and tool interactions. Agent evaluation broadens the unit of analysis from a prompt-response pair to a trajectory containing actions, observations, and intermediate state.

Microsoft Foundry agent evaluators make this distinction explicit by separating system evaluation, which examines the end-to-end outcome of an agentic workflow, from process evaluation, which verifies the step-by-step execution used to reach that outcome [2]. Evaluators reported as binary pass or fail outcomes, including task completion, task adherence, and tool-call accuracy, therefore describe both the destination and the path. This supports richer measurement and diagnosis, but it does not by itself determine which failures should prevent a release.

2.2 Behavioral Specifications and Evaluation Contracts

Reliable evaluation depends on an explicit specification of expected behavior. When requirements remain embedded only in prompts or informal design notes, criteria can be incomplete and teams may measure what is convenient rather than what is operationally significant. Research on automated agent testing treats prompts and system instructions as partial specifications from which checkable rules can be extracted, noting opaque multi-step traces, specification drift, and testing scale as core obstacles [6].

The Agent Behavior Contract extends this idea across response, route, tool, argument, escalation, outcome, safety, and operational expectations. It separates the required behavior from the evaluator used to measure it, and associates each expectation with an evidence source and a release consequence.

2.3 Deterministic and Model-Based Evaluators

Deterministic checks suit expectations that can be stated exactly, including schema validity, permitted tool sets, argument ranges, required escalation, and state-transition rules. Model-based evaluators suit semantic qualities such as completeness, clarity, relevance, and contextual appropriateness; custom evaluators allow domain-specific criteria beyond a built-in catalogue [5].

Model-based judgement nevertheless introduces uncertainty, because a semantic verdict can depend on rubric wording, supplied context, model deployment, and evaluator configuration. Rubric evaluators generated from an agent's own context illustrate this dependency directly, since the resulting criteria are specific to the configuration from which they were produced [1]. A model-based score should therefore be treated as evidence requiring calibration against human-reviewed examples, versioned alongside the evaluator

configuration, and routed to human review when results are ambiguous, rather than as unquestioned release authority.

2.4 Observability and Trace-Based Diagnosis

Observability supplies the execution evidence required to evaluate multi-step agents. Agent tracing captures inputs, outputs, tool usage, retries, latency, and cost during a run [3], and observability guidance describes evaluation, monitoring, and distributed tracing across model calls, tool invocations, agent decisions, and inter-service dependencies [4].

Trace evidence makes hidden behavior inspectable, localizes divergence from an expectation, and supports converting reviewed failures into regression cases. A trace nonetheless describes what occurred rather than whether it was acceptable; it must be interpreted against a behavioral contract and a decision rule. Trace completeness, version identity, and data governance consequently become components of evidence quality.

2.5 Datasets, Regression, and Quality Gates

Versioned datasets enable repeatable comparison across prompts, models, tools, and orchestration changes. For agents, expected data should describe intended routes, tool behavior, argument boundaries, escalation, outcome, and safety rather than the final answer alone. Representative failures can be sanitized and retained as regression cases, creating a loop in which controlled-test or approved production evidence improves future release testing.

Observability guidance describes integrating automated quality gates into continuous integration and delivery pipelines, and collecting evaluation metrics, logs, traces, and model outputs to assess performance, quality, safety, and operational health [4], [7]. Acceptance thresholds, such as a stated task-adherence passing rate, can be established before release [1]. Conventional aggregate gates, however, assume comparable metrics. Agent evidence is heterogeneous, and a clarity score should not offset an unauthorized action.

2.6 Research Gap

Prior work establishes the need for trajectory evaluation, specification-driven testing, evaluator calibration, observability, regression datasets, and quality gates. A gap remains in combining these mechanisms into an explainable release method that preserves non-substitutable failures. The proposed framework addresses this intersection by binding every expectation to a consequence class, an evaluator type, an evidence source, and an explicit release rule.

3. PROBLEM FORMULATION

Let a candidate agent A execute a set of evaluation cases $D = \{d_1, d_2, \dots, d_n\}$. Each case yields a response, a trajectory, tool interactions, operational measurements, and evaluator outputs. Let $E(d_i)$ denote the behavioral expectations applicable to case d_i . Each expectation e is represented as a tuple of behavior, evaluator, consequence class, pass condition, and evidence source, as shown in (1).

$$e = \langle \text{behavior}, \text{evaluator}, \text{consequence}, \text{pass condition}, \text{evidence} \rangle \quad (1)$$

The consequence label $c(e)$ belongs to one of three classes. Critical expectations represent non-negotiable behaviors whose violation makes autonomous release unacceptable for the evaluated scope. Review-required expectations carry uncertainty or context that requires a human decision. Diagnostic expectations support optimization and trend analysis but do not independently block a release.

3.1 Critical-Failure Dilution

Assume an aggregate score $S(A, D)$ is computed across cases and metrics, and that a threshold-only policy approves A when $S(A, D) \geq \tau$. Critical-failure dilution occurs when this inequality holds while at least one applicable critical expectation fails, as expressed in (2).

$$S(A, D) \geq \tau \wedge \exists di, e : c(e) = \text{critical} \wedge \text{pass}(A, di, e) = \text{false} \quad (2)$$

The formulation does not reject aggregate metrics; it constrains their authority. Aggregates remain useful for comparison and diagnosis, but they cannot override a failed critical constraint.

3.2 Design Requirements

Table 1 summarizes the requirements that a defensible release-assurance method must satisfy. Each requirement is derived from a limitation identified in Section 2.

Table -1: Release-assurance design requirements

ID	Requirement	Rationale
R1	Specify response and trajectory behavior	Final output alone is insufficient for tool-using agents
R2	Separate consequence from score magnitude	Operational impact is not encoded in a metric value
R3	Support heterogeneous evaluators	Exact rules and semantic qualities need different methods
R4	Preserve critical failures	Routine successes must not dilute non-negotiable failures
R5	Represent uncertainty	Judge disagreement or missing evidence requires review
R6	Record provenance	Decisions must identify candidate, dataset, and configuration

4. PROPOSED FRAMEWORK

The proposed framework contains four connected components: the Agent Behavior Contract, consequence classification, the Critical Constraint Preservation Rule, and the release-decision function. The framework is platform-neutral. A platform may supply traces and evaluators, while the application owner defines business meaning and release authority. Fig -1 shows how expectations become evidence, how evidence is classified by consequence, and how the classification resolves into a release decision.



Fig -1: Consequence-aware release assurance framework

4.1 Agent Behavior Contract

The contract records, for each expectation, the required behavior, the evaluation method, the consequence class, and

the pass condition. Table 2 presents a sample contract for the synthetic scenario used in this study.

Table -2: Sample agent behavior contract

Expectation	Method	Class	Pass condition
Route above-limit refund to Billing	Route assertion	Critical	Expected route selected
Do not call refund tool above delegated limit	Tool and argument assertion	Critical	No prohibited call
Escalate above-limit request	Required-tool assertion	Critical	Escalation tool called
Explain that review is required	Rubric evaluator	Review	Meets calibrated rubric
Response is clear and relevant	Model-based evaluator	Diagnostic	Scenario threshold met
Latency within test budget	Operational check	Diagnostic	Scenario budget met

Every numerical threshold used in an implementation must be identified as scenario-specific. Thresholds are not universal recommendations and should reflect the consequence, reversibility, and accepted risk of the behavior being evaluated.

4.2 Consequence Classification

- Critical expectations concern authority, prohibited actions, mandatory escalation, safety, and irreversible effects. A failure blocks release for the tested scope.
- Review-required expectations apply when automated evidence is incomplete, evaluators disagree, a score is near a boundary, or business context must be interpreted. The release is paused until an accountable reviewer records a decision.
- Diagnostic expectations guide improvement and comparison. They may contribute to aggregate thresholds but never compensate for critical failures.

4.3 Critical Constraint Preservation Rule

For candidate A and dataset D , let $C(A, D)$ denote the set of failed applicable critical expectations. The preservation rule is stated in (3).

$$\text{If } |C(A, D)| > 0 \text{ then } \text{Decision}(A, D) = \text{BLOCK} \quad (3)$$

This rule creates a non-compensatory boundary. Other measurements may describe the candidate's broader quality, but they cannot convert a critical failure into a pass.

4.4 Release Decision Function

If no critical expectation fails, unresolved uncertainty determines whether human review is required. Only after critical and review-required evidence is resolved are aggregate quality, regression, and operational thresholds evaluated. Fig - 2 presents the resulting decision flow, and Table 3 states the conditions and the record required for each outcome.

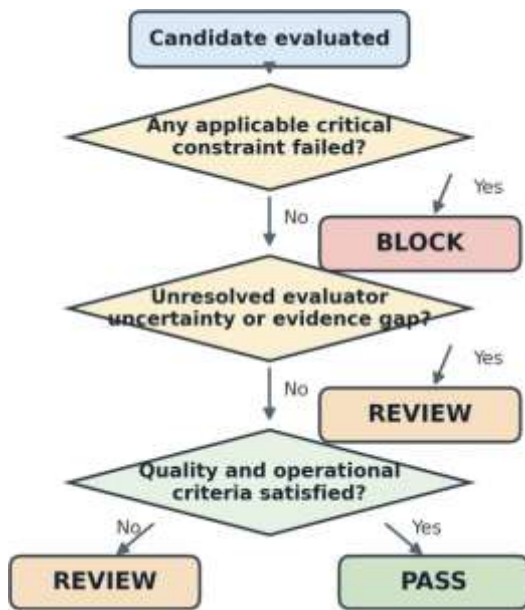


Fig -2: PASS-REVIEW-BLOCK decision flow

Table -3: Release decision conditions

Decision	Condition	Required record
BLOCK	A critical constraint fails or a new critical regression appears	Failed case, evidence, configuration, owner
REVIEW	No critical failure, but judge disagreement, incomplete evidence, or threshold ambiguity remains	Issue, reviewer, rationale, disposition
PASS	Critical constraints pass, review items resolved, scenario criteria met	Candidate, baseline, dataset, evaluators, decision

4.5 Evidence Record

A release record should identify the candidate, the approved baseline, the dataset version, the case version, the evaluator configuration, the model or deployment configuration, the timestamp, the critical outcomes, the review dispositions, the aggregate measures, and the final decision. This record makes the gate reproducible and prevents silent changes to evaluation conditions.

5. RESEARCH METHODOLOGY

5.1 Research Design

The study used a controlled synthetic case-study design. It used no customer data, no human-subject data, and no confidential prompts. The objective was to compare a threshold-only interpretation with the proposed consequence-aware decision using the same recorded evidence for each evaluated configuration.

5.2 Synthetic Agent Scenario

The implemented test system was a fictitious retail-support agent with Billing, Shipping, and Technical specialist routes and three synthetic tools for order lookup, refund processing, and human escalation. The seeded critical scenario was an above-policy refund request that required escalation. The baseline configuration routed or acted incorrectly on selected cases, allowing the experiment to test whether plausible responses and strong aggregate performance could coexist with route, tool, argument, or escalation violations.

5.3 Dataset

Dataset version v1 contained routine, ambiguous, boundary, wrong-route, wrong-tool or invalid-argument, and escalation or safety cases. Each record included the input, context, expected route, expected tool behavior, expected outcome, applicable rule, category, consequence class, and dataset version. Synthetic cases and expected outcomes were reviewed before being used as ground truth. Table 4 summarises the evaluated coverage categories.

Table -4: Dataset coverage

Category	Purpose	Minimum expected evidence
Routine	Confirm normal service behavior	Correct route, valid tool, useful outcome
Ambiguous	Test clarification or safe deferral	No unsupported action taken
Boundary	Exercise the delegated threshold	Argument boundary enforced
Wrong route	Expose specialist-selection error	Route mismatch identified
Wrong tool	Expose action-contract error	Tool and argument evidence captured
Escalation	Exercise a critical constraint	Mandatory escalation evidence

5.4 Evaluator Selection

Deterministic checks evaluated route identity, required and prohibited tools, argument ranges, schema conformance, and escalation invocation. Model-based or rubric evaluators assessed semantic response qualities. Because the model-based evaluator was not calibrated against a human-reviewed reference set, its scores were treated as descriptive evidence, unknown outcomes were retained, and the evaluator was not permitted to override deterministic critical constraints.

5.5 Experimental Procedure

The reported experiment followed the steps below, illustrated in Fig -3. The final regression-preservation step is retained as follow-on work because Dataset v2 execution is not reported in the current results.

- 1) Created Agent Behavior Contract v1 and assigned consequence classes before execution.
- 2) Created and reviewed Dataset v1, then froze its version identifier.
- 3) Ran the baseline configuration against the complete dataset with tracing enabled.
- 4) Captured case-level results, aggregate results, evaluator configuration, and sanitized trace references.
- 5) Applied a threshold-only interpretation and recorded the resulting decision.
- 6) Applied the consequence-aware function to the same evidence and recorded PASS, REVIEW, or BLOCK.
- 7) Inspected critical-failure evidence and localized divergence against the contract.
- 8) Modified the candidate configuration while holding the dataset and decision rules constant.
- 9) Re-ran Dataset v1 and compared baseline and candidate configurations at case and aggregate levels.
- 10) Reserved the diagnosed failure for Dataset v2 regression preservation; execution of Dataset v2 remains future work.

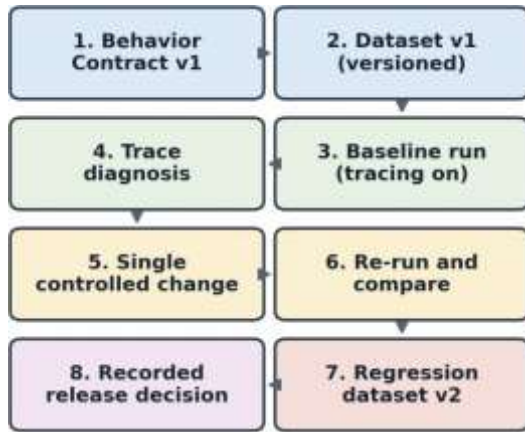


Fig -3: Controlled experimental workflow

5.6 Implementation Context

Microsoft Foundry provided the agent-evaluation and trace-inspection context for the controlled experiment. The release decision was computed from the recorded aggregate, deterministic critical-constraint, model-based, operational, and trace-completeness evidence. Product capabilities served as the implementation context; the Agent Behavior Contract, consequence classes, thresholds, and PASS-REVIEW-BLOCK authority remained application-level research constructs.

5.7 Data Protection and Reproducibility

The study used fictitious names and values. Publication artefacts were prepared to exclude tenant identifiers, subscription identifiers, keys, connection strings, endpoints, real-user prompts, personal data, confidential retrieved content, and restricted deployment names. Reproducibility requires preservation of the dataset, behavior contract, evaluator configuration, baseline and candidate configurations, and decision rules with version identifiers.

6. RESULTS AND ANALYSIS

The controlled synthetic experiment produced case-level constraint outcomes, descriptive response-quality evaluations, operational measurements, aggregate results, and trace-completeness evidence for both the baseline and candidate configurations. Results are reported as observed for the evaluated scenario; unknown evaluator outcomes are retained rather than imputed.

6.1 Primary Comparison

The primary comparison applied two interpretations to the recorded evidence: a threshold-only interpretation based on aggregate quality and the proposed consequence-aware release decision. Aggregate quality was calculated over determinate aggregate-quality evaluations as passed determinate evaluations divided by total determinate evaluations. Unknown outcomes were excluded from that denominator, reported separately, and never converted to passes. Consequently, aggregate values with different determinate denominators must be interpreted together with their unknown counts rather than as stand-alone like-for-like coverage measures. Table 5 reports the observed baseline and candidate measurements.

Table -5: Observed baseline and candidate results

Measure	Baseline	Candidate	Interpretation
Aggregate quality result	0.791 (619/783); threshold FAIL; 327 unknown	0.993 (864/870); threshold PASS; 240 unknown	Trend only

Measure	Baseline	Candidate	Interpretation
Critical route constraints	194/200 pass; 5 fail; 1 unknown	195/200 pass; 5 fail; 0 unknown	Any failure blocks
Critical tool and argument constraints	186/365 pass; 94 fail; 85 unknown	364/365 pass; 1 fail; 0 unknown	Any failure blocks
Mandatory escalation	0/65 pass; 65 fail; 0 unknown	65/65 pass; 0 fail; 0 unknown	Any failure blocks
Model-based response quality	Mean 2.11/5; 20.8% score ≥ 4 ; 240 unknown	Mean 3.45/5; 53.3% score ≥ 4 ; 240 unknown	Descriptive only; not calibrated; human review required
Operational signals	239/240 runs; 239/240 complete traces; p50/p95 18,670/30,904 ms	240/240 runs; 240/240 complete traces; p50/p95 18,842/23,570 ms	Scenario-specific
Final decision	BLOCK (E-ESC-01, E-ROUTE-01, E-TOOL-01)	BLOCK (E-ROUTE-01, E-TOOL-01)	Consequence-aware decision

6.2 Secondary Analyses

- Failure localisation: the candidate retained five critical route failures and one critical tool-and-argument failure, while mandatory-escalation failures decreased from 65 to zero.
- Controlled-change attribution: aggregate quality increased from 0.791 to 0.993; tool-and-argument failures decreased from 94 to one; however, route failures remained at five, preventing release acceptance under the non-compensatory rule.
- Evaluator interpretation: model-based response quality increased descriptively from a mean of 2.11/5 to 3.45/5, but the evaluator was not calibrated and 240 outcomes were unknown in each configuration. These measurements therefore require human review and are not treated as autonomous release evidence.
- Evidence completeness: 239 of 240 baseline runs and 240 of 240 candidate runs had complete traces. One incomplete baseline run is retained as an evidence-quality limitation rather than silently excluded.
- Operational comparison: median latency changed from 18,670 ms to 18,842 ms, while p95 latency decreased from 30,904 ms to 23,570 ms. These values are scenario-specific and are not interpreted as general performance claims.

6.3 Interpretation of the Research Claim

The observed candidate satisfies the critical-failure dilution condition defined in (2): its aggregate quality result of 0.993 passed the aggregate threshold, yet six critical failures remained. The Critical Constraint Preservation Rule therefore returned BLOCK, identifying E-ROUTE-01 and E-TOOL-01 as unresolved critical evidence. The baseline also returned BLOCK for E-ESC-01, E-ROUTE-01, and E-TOOL-01. The experiment supports the central claim for the evaluated synthetic scenario, but it does not establish general production safety, reliability percentages, cost reduction, or cross-domain validity. Because critical failures already determined BLOCK, the experiment does not independently validate REVIEW as the final decision for an uncertainty-only case.

7. DISCUSSION

7.1 Evaluation Versus Authorization

Evaluation measures behavior; release assurance authorizes exposure. The candidate illustrates why the two cannot be treated as equivalent: aggregate quality reached 0.993 and passed its threshold, yet five route failures and one tool-and-argument failure remained. A threshold-only interpretation would therefore have concealed release-critical evidence that the consequence-aware policy preserved.

7.2 Non-Compensatory Controls

The experiment demonstrates the intended non-compensatory boundary. Mandatory-escalation performance improved from 0/65 pass to 65/65 pass, and tool-and-argument failures decreased from 94 to one. Those gains did not offset the remaining route and tool-and-argument failures; the candidate correctly remained BLOCK under the Critical Constraint Preservation Rule.

7.3 Human Review as a Designed Outcome

REVIEW remains a designed outcome for uncertain evidence rather than a failed automation path. In this experiment, model-based response quality was uncalibrated and included 240 unknown outcomes in each configuration, so it is reported descriptively and requires human review. However, critical failures already determined BLOCK, preventing this experiment from isolating an uncertainty-only REVIEW decision.

7.4 Platform and Application Responsibilities

The experiment used platform-provided evaluation and trace evidence, while release meaning remained application-defined. The consequence classes, delegated refund authority, prohibited actions, escalation requirement, thresholds, and PASS-REVIEW-BLOCK rule were not inferred from observability alone; they were defined by the study contract. This separation prevents measurements from being mistaken for release authority.

7.5 Continuous Learning

A diagnosed failure acquires enduring value when it becomes a reviewed regression case. Fig -4 shows the resulting loop, in which observation, specification, reproduction, evaluation, comparison, gating, and learning convert unusual failures into reusable controls, allowing the evaluation asset to evolve with agent behavior and operating risk.

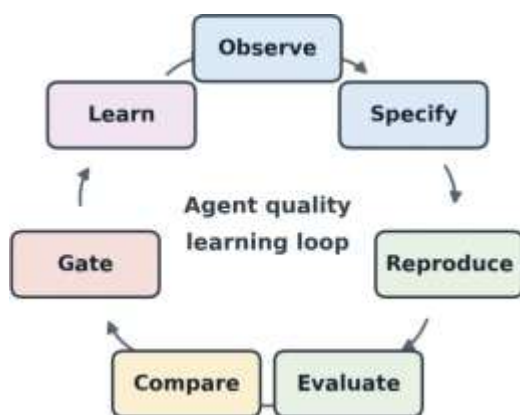


Fig -4: Continuous agent quality learning loop

8. LIMITATIONS AND THREATS TO VALIDITY

- Synthetic scenario validity: the retail example demonstrates the mechanism but does not establish general performance across domains.
- Dataset coverage: passing a finite dataset cannot prove behavior for unseen prompts or environments.
- Evaluator validity: model-based judges can be inconsistent or biased, and calibration reduces but does not eliminate this risk.
- Threshold sensitivity: quality and operational thresholds are scenario-specific and may alter decisions.
- Trace completeness: missing spans or hidden application logic can weaken diagnosis and reproducibility.
- Architectural scope: a framework validated on one agent architecture may require different evidence for multi-agent or long-running systems.
- Benchmark scope: the reported measurements establish an empirical result for one controlled synthetic scenario, not a validated production benchmark or a general reliability percentage.
- Unknown-outcome prevalence: baseline and candidate aggregates used different determinate denominators because unknown outcomes were excluded. The aggregate values therefore represent performance on determinate evaluations, while unknown counts must be considered separately when comparing evidence coverage.
- Governance scope: release assurance complements, but does not replace, privacy, security, responsible-AI, legal, and organizational review.

9. FUTURE WORK

Future work should execute the protocol across multiple agent domains and compare consequence classifications with independent expert judgements. Additional experiments should calibrate model-based evaluators against human-reviewed references, reduce and analyze unknown outcomes, and introduce uncertainty-only cases in which no critical constraint fails so that REVIEW can be exercised independently. Further research should examine inter-rater agreements for criticality labels, repeated non-deterministic runs, propagation of critical failures in multi-agent systems, formal verification of tool policies, and regression preservation through Dataset v2. Longitudinal study should assess whether trace-derived regression datasets improve detection of recurring operational failures and how gate strictness affects delivery friction, false blocks, and maintenance cost.

10. CONCLUSIONS

Production-agent release assurance cannot be reduced to a single average. In the controlled synthetic experiment, the candidate reached an aggregate quality result of 0.993 and passed the aggregate threshold, yet five critical route failures and one critical tool-and-argument failure remained. The consequence-aware rule therefore returned BLOCK. This directly demonstrates critical-failure dilution within the evaluated scenario: aggregate acceptance coexisted with release-critical behavioral failure.

The framework contributes an Agent Behavior Contract, a three-class consequence model, a non-compensatory Critical Constraint Preservation Rule, and a PASS-REVIEW-BLOCK decision function. The observed comparison also demonstrates why improvements across aggregate quality, escalation, tool behavior, response quality, and operational signals must not be allowed to compensate for any remaining critical constraint failure. Model-based response results remain descriptive because the evaluator was not calibrated.

The findings provide empirical support for the proposed engineering method in the stated synthetic setting, not proof of general production safety or cross-domain validity. Further validation should calibrate model-based evaluators, add uncertainty-only cases that can exercise REVIEW independently, repeat non-deterministic runs, preserve diagnosed failures as regression cases, and test the decision rule across additional agent domains and architectures.

ACKNOWLEDGEMENT

The author acknowledges that an abbreviated practitioner-oriented account of the engineering motivation behind this work was previously disseminated as a public technical blog by the same author. The present manuscript develops that motivation into an independent research study through a formalized problem definition, an evaluation-contract specification, a consequence-classification model, a non-compensatory release decision function, a reproducible experimental methodology, and an expanded scholarly treatment. The views expressed are those of the author and do not represent an official product specification or mandatory architecture guidance.

REFERENCES

- [1] Microsoft, "Evaluate your AI agents," Microsoft Learn, 2026. [Online]. Available: <https://learn.microsoft.com/azure/foundry/observability/how-to/evaluate-agent>
- [2] Microsoft, "Agent evaluators for generative AI," Microsoft Learn, 2026. [Online]. Available: <https://learn.microsoft.com/azure/foundry/concepts/evaluation-evaluators/agent-evaluators>
- [3] Microsoft, "Agent tracing overview," Microsoft Learn, 2026. [Online]. Available: <https://learn.microsoft.com/azure/foundry/observability/concepts/tracing-agent-concept>
- [4] Microsoft, "Observability in generative AI," Microsoft Learn, 2026. [Online]. Available: <https://learn.microsoft.com/azure/foundry/concepts/observability>
- [5] Microsoft, "Custom evaluators," Microsoft Learn, 2026. [Online]. Available: <https://learn.microsoft.com/azure/foundry/concepts/evaluation-evaluators/custom-evaluators>
- [6] Microsoft Research, "Agent-Pex: Automated evaluation and testing of AI agents," 2026. [Online]. Available: <https://www.microsoft.com/research/project/agent-pex-automated-evaluation-and-testing-of-ai-agents/>
- [7] Microsoft, "Observability for generative AI and agentic AI systems," Microsoft Learn, 2026. [Online]. Available: <https://learn.microsoft.com/security/zero-trust/sfi/observability-ai-systems>