10

WHITE PAPER · AI OPERATING GOVERNANCE

An AI Evaluation Is a Production Operating Control

Govern changing workflow behavior, not a one-time test

Marco Policani

Enterprise Portfolio · PMO · AI Operating Governance

policani.net · linkedin.com/in/marcpolicani

July 2026

EXECUTIVE SUMMARY

Evaluation continues in production because workflow behavior, context, and reliance continue to change.

A model can pass its launch evaluation and still fail the operation that adopts it, because the test measured a moment and production lives in a stream of changing models, data, cases, and reliance. This paper keeps evaluation alive as an operating control: representative cases with a stated boundary, thresholds by consequence, monitored signals, change as a test event, incidents converted to cases, and named authority over reliance.

01 Write thresholds by consequence class before results are seen; afterward they negotiate themselves toward whatever the system achieved.

02 Treat every material change — including prompt edits — as a regression event scoped to what it could have broken.

03 Run the staleness audit: the gap between the last evidence date and the last change date is how much of production runs on faith.

The argument

Benchmarking, red teaming, and production evaluation answer different questions. Production evaluation is a living control that tests representative workflow cases, monitors change, converts incidents into cases, and changes permitted reliance when evidence changes.

The failure that taught me the difference arrived eight months after a successful launch. A document-classification workflow had cleared its pre-deployment evaluation convincingly — a proper case set, honest thresholds, results everyone could stand behind. Then, over two quarters, three small things happened that nobody connected: the vendor shipped a model update, an upstream team changed how source documents were formatted, and the business expanded the workflow to a document type the case set had never contained. Each change was individually announced, individually reasonable, and individually invisible to the evaluation, which had run once, passed once, and retired to a folder. The workflow's accuracy on the new mix degraded for weeks before a downstream complaint surfaced it. The launch evaluation had not been wrong. It had been abandoned.

That is the general condition: a model can pass an evaluation and still fail the operation that adopts it, because the test measured a moment and the operation lives in a stream. Cases drift from the tested population, users rely on output in ways the design never assumed, retrieval data ages, and vendors change the engine underneath. None of this is exotic — it is what production means — and it implies that evaluation cannot be a launch artifact. It has to be an operating control: maintained, connected to change, and wired to the authority that decides what the workflow may rely on.

This paper describes that control as a production evaluation loop with seven disciplines. NIST's generative-AI profile argues for systematic, continuing measurement; Microsoft's red-team lessons show how much failure-finding depends on deliberate, resourced effort [1][2]. The loop is the machinery I ended up building around that classification workflow, generalized; the paper is written for AI product leaders, workflow owners, risk teams, and technology executives.

Three tests, three questions

The word 'evaluation' covers three different jobs, and conflating them is the root error in most AI assurance conversations. A benchmark compares capability across models on standardized tasks; it answers 'which engine is stronger, in general?' A red team searches adversarially for ways a system can be made to fail; it answers 'where are the weaknesses a motivated actor or unlucky input could find?' A production evaluation tests a specific workflow's cases under its own consequence structure; it answers 'may this operation rely on this system, for these cases, right now?'

Each test is necessary and none substitutes for the others. The characteristic misuse is stretching: a strong benchmark cited as proof a workflow is safe, a pre-launch red team cited as proof the system remains safe a year later, a passing workflow evaluation cited to authorize case types it never contained. Stretching feels efficient because tests are expensive; it is actually the most expensive move available, because the stretched result authorizes reliance the evidence does not cover — and the gap is exactly where the incidents live [2].

The discipline is documentary and cheap: every test on record states which question it answers, what population it covers, what environment it ran in, and what decision it may support. When a governance decision needs an answer that no existing test provides, the correct move is to commission the missing test — never to promote an existing result beyond the jurisdiction its own record claims for it.

The three-test separation also allocates scarce effort correctly. Benchmarks are largely free — published, maintained by others, useful for shortlisting engines. Red teams are episodic and expensive — commissioned at launch, after major change, and on a cadence proportionate to adversarial exposure [2]. Production evaluation is continuous and owned — the only one of the three the operation must run itself, because only the operation has its cases, its consequences, and its definition of fit. Budget conversations that treat the three as one line item reliably fund the first two and starve the third.

The production evaluation loop

The loop has seven disciplines. The first three build the evaluation's substance — cases, thresholds, and the separation above. The next three keep it alive — monitoring, change events, and incident learning. The last connects it to power: someone must be able to change what the workflow relies on, or the whole apparatus is measurement without consequence.

EXHIBIT 1

Seven disciplines keep the evidence as current as the system

1. **Separate three evaluation jobs**
   Benchmarks compare capability, red teams search for weaknesses, and operating evaluations test a specific workflow under its own consequences.
2. **Build representative workflow cases**
   The case set should reflect ordinary work, important subgroups, exceptions, difficult inputs, and foreseeable misuse.
3. **Define thresholds by consequence**
   Average quality can hide severe failures; thresholds should combine performance, severity, and control response.
4. **Monitor production signals**
   Evaluation continues through drift, exceptions, overrides, complaints, incidents, and changes in user behavior.
5. **Turn change into a test event**
   Model, prompt, retrieval, policy, tool, user, and workflow changes can invalidate prior evidence.
6. **Learn from incidents and near misses**
   Production failures should expand the case set and improve boundaries, not end with a local correction.
7. **Assign authority to change reliance**
   Evaluation matters only when someone can narrow, pause, restore, or retire use.

Build representative workflow cases

The case set is the evaluation's claim to relevance. Built from convenience — the examples the build team had at hand, the demos that impressed — it tests the system's comfort zone and certifies nothing beyond it. Built representatively, it samples production history with the mess left in, weights the subgroups that matter, includes the exceptions and edge conditions that consume operating judgment, and adds the foreseeable misuse a red-team mindset contributes.

Provenance is what separates the two kinds of case set, and it should be written down as part of the set itself: where each case class came from, why its frequency in the set is defensible against production reality, which reference answers are certain and which are themselves judgment calls carrying recorded uncertainty. Domain experts belong in this work — the operation's people know which cases are hard and why, and their afternoon of taxonomy is worth more than a month of synthetic generation.

The set's boundary is a governance output in its own right: permitted use extends to the population the cases represent, and no further. The expansion that broke my classification workflow — a new document type, admitted without a case-set extension — is precisely the move this discipline prohibits. New population, new cases, new results, then new reliance; the order is the control.

Size is the least important property of a case set, and the most argued about. Two hundred well-chosen cases with honest reference answers govern better than ten thousand scraped ones, because the failure analysis that follows a run has to be read by humans who will act on it. Start small and representative, and let the incident pipeline grow the set where reality says it is thin — growth by evidence, not by volume anxiety.

Define thresholds by consequence

A single accuracy number is an average, and averages are where severe failures hide. Ninety-six percent overall can contain a critical case class failing three times in ten — invisible in aggregate, intolerable in operation. Thresholds therefore attach to case classes, weighted by what a failure in that class costs: acceptance rates for routine classes, stricter floors for consequential ones, abstention and escalation expectations where the right answer is 'route to a human,' latency bounds where timing is part of correctness.

Abstention deserves explicit thresholds because it is the workflow's pressure valve. A system that answers everything is running without one; a system that abstains too readily exports its work back to the humans it was meant to relieve. Setting an expected abstention band per case class — and alarming on both edges — catches two failure modes with one instrument: overconfidence on cases the system should route away, and silent capability collapse disguised as caution.

Thresholds must be written before results are seen — afterward, they negotiate themselves toward whatever the system achieved, and the evaluation becomes a mirror instead of a bar. Writing them first forces the uncomfortable, valuable conversation about what the operation actually requires: which failures are absorbable, which are expensive, and which are the reason the workflow has a human in it at all.

The decision rule keeps the aggregate honest: a critical-class failure pauses or narrows reliance even when the average looks excellent. Review load belongs in the threshold set too — false rejections that flood human reviewers are a real cost, and a threshold structure that ignores them will be quietly overridden by the operation defending its own capacity.

Monitor production signals

Between formal evaluations, the workflow itself broadcasts evidence — if anything is listening. Input distributions shift, output patterns move, reviewer overrides cluster, exceptions rise in one case class, users develop workarounds, downstream corrections accumulate. Most organizations instrument uptime and cost and stop there, which means they can see the system running and cannot see whether it still fits the work.

The monitoring discipline instruments fitness: input characteristics against the case set's assumptions, output distributions against historical shape, review dispositions and override reasons, exception rates by class, and sampled quality checks on live cases with appropriate privacy handling. None of this needs to be elaborate at the start — a monthly sample read by someone who knows the work, plus drift flags on the obvious distributions, catches most of what matters, and the instrumentation can grow where the signals prove valuable rather than where the tooling catalog suggests.

The design question is response, not collection: each signal needs a threshold that triggers something — a focused evaluation on the moving class, a review-capacity check, a case-set extension. Signals collected without response thresholds become dashboard wallpaper, and their real function becomes retrospective: explaining, after the incident, that the evidence had been available all along.

Reviewer disagreement is a monitoring signal in its own right, and an underused one. When two qualified reviewers sampling the same live cases diverge on what counts as acceptable, the workflow has either a drifting system or a drifting standard — and both need governance attention. Tracking inter-reviewer agreement on the

monthly sample costs almost nothing and catches quality-definition drift that no automated distribution check can see.

Turn change into a test event

Production AI systems change constantly, and every change class can invalidate prior evidence: model updates, prompt edits, retrieval-source changes, tool additions, policy revisions, workflow redesigns, user-population shifts. The operating question is never whether to allow change — it is whether change reaches production through a path that re-tests what the change could have broken.

The discipline is regression scoped by change class: a vendor model update reruns the full case set; a prompt refinement reruns the affected classes; a new retrieval source reruns retrieval-dependent cases plus a contamination sample; a workflow redesign may require new cases entirely. Version history ties each production state to its evidence, and approval scales with the consequence of what the change touches. My formatting change from the opening — upstream, unannounced to the evaluation, untested — is the canonical miss: nobody owned the question 'what does this change touch?'

The rule is symmetrical with the case-set boundary: deployment waits where regression evidence is incomplete, and reliance narrows where a change degraded a class the operation depends on. Teams accept this discipline readily for code and resist it for prompts and models; the resistance is worth naming, because a prompt edit can move behavior more than most code releases.

Vendor changes deserve their own contract clause, negotiated before dependence: advance notice windows for model updates, version pinning where offered, and change logs specific enough to scope a regression. Providers vary widely on all three, and the variance is procurement-relevant information. A vendor that cannot say what changed has made its customers' regression scope 'everything' — a recurring cost that belongs in the price comparison alongside the per-token rate.

Learn from incidents and near misses

When the workflow fails in production — or nearly does — the failure is a free case, paid for at incident prices. The wasteful response repairs the instance: fix the record, apologize downstream, adjust the one prompt. The evaluative response converts it: a short causal review across the usual suspects — model, data, interface, instructions, review capacity, permissions, operating pressure — then a new case class in the set, a threshold or monitoring adjustment if the failure revealed a blind spot, and recurrence tracking to verify the fix held.

Near misses deserve equal standing, and mostly do not get it. A failure caught by an alert reviewer is simultaneously evidence that the control worked and evidence that the system produced the failure — both facts belong in the record, because the next instance of the same failure may not meet the same alert reviewer on the same unhurried day. Programs that learn only from consequential incidents have chosen, implicitly, to learn at the maximum available price per lesson.

Over quarters, the incident-to-case pipeline becomes the evaluation's growth mechanism: the set stops being the launch team's best guess about hard cases and becomes the operation's accumulated memory of actual ones. That memory is a portfolio asset — new deployments in adjacent workflows inherit case classes their own launches would have discovered expensively.

The incident review's scope discipline matters: across model, data, interface, instructions, review capacity, permissions, and operating pressure, the cause is frequently not the model at all. A misrouted case may trace to an upstream data change; an accepted bad answer may trace to a review queue running at twice its designed volume. Converting incidents into cases without the causal breadth just grows the case set while the actual defect — capacity, interface, instruction — recurs untouched.

Assign authority to change reliance

Everything above produces evidence; this discipline gives evidence somewhere to go. A named workflow owner holds the reliance decision — what the operation may depend on, for which cases. A technical owner holds the system and its versions. A risk voice holds challenge rights. A defined forum reviews evaluation results and monitoring signals on cadence, and an emergency route exists for findings that cannot wait for the cadence.

The failure this prevents is the analyst-report pattern: evaluations run, results circulate, and nothing changes because no one owns the connection between evidence and permission. The tell is a finding that everyone read and no one acted on — a threshold breached in an appendix while adoption momentum carried the workflow forward. Where that tell appears, the organization does not have an evaluation control; it has evaluation content.

The authority's evidence is decision-shaped: reliance narrowed within days of a breached threshold, restorations tied to re-tests, emergency stops exercised at least once against a rehearsed fallback, and a record connecting each change in permitted use to the evidence that drove it. Time-from-signal-to-decision is the metric that matters; evidence that changes reliance in days is a control, and evidence that changes it in quarters is history.

Cadence completes the authority design: a standing review — monthly in most operations — that reads the monitoring summary, the change log, and the incident conversions, and issues one of three findings per workflow: reliance confirmed, reliance adjusted, or evidence gap opened with an owner and a date. The three-finding format keeps the meeting short and the record decision-shaped; a review that ends without one of the three has been a briefing.

The loop as a system

The disciplines compound in a specific way: representative cases make thresholds meaningful, thresholds make monitoring actionable, monitoring makes change events detectable, change discipline keeps the case evidence current, incidents feed the case set its hardest members, and named authority makes all of it consequential. Remove any one and the others degrade predictably — most commonly, organizations build cases and thresholds, skip monitoring and change discipline, and rediscover at incident time that their evidence describes a system that no longer exists.

EXHIBIT 2

The reliance decision each discipline feeds

Separate three evaluation jobs

Commission the missing test rather than stretching one result beyond its scope.

Build representative workflow cases

Limit permitted use to the population the case set can reasonably represent.

Define thresholds by consequence

Pause or narrow reliance when a critical threshold fails even if the aggregate average looks strong.

Monitor production signals

Trigger focused evaluation when signals move beyond expected variation.

Turn change into a test event

Withhold or reduce deployment when regression evidence is incomplete.

Learn from incidents and near misses

Change permitted use when the incident reveals a class of exposure larger than the original case.

Assign authority to change reliance

Treat evaluation as an operating control, not a research report.

The loop is also the connective tissue between the other controls an AI operation needs. Authority ladders decide how much the workflow may act on the system's output; the loop supplies the evidence those grants cite. Instance-level challenge protocols catch the specific bad answer; the loop catches the drift no instance reveals. Reliance decisions at the production gate cite the launch evaluation; the loop is what keeps that citation truthful afterward. One evaluation, maintained, serves all three.

Standing the loop up

The loop installs in the order its evidence compounds:

- Create the first case set from real workflow history, not a generic benchmark.
- Write acceptance and stop thresholds before running the evaluation.
- Connect production telemetry and reviewer decisions to a maintained evaluation backlog.
- Run regression after material change and a scheduled review even when no incident occurs.

The first loop cycle doubles as archaeology. Building the case set from history surfaces how the workflow actually behaves — the exception classes nobody had counted, the case types that dominate reviewer time, the quiet workarounds users built. Teams routinely report that the evaluation's first contribution preceded any test run: the act of assembling representative cases taught them what the workflow was.

Resourcing is the honest conversation to have at installation, because the loop is standing work: case-set maintenance, monitoring review, regression runs, incident conversion. The sizing heuristic is proportionality to reliance — a workflow the operation depends on daily deserves a few hours of evaluation stewardship weekly, and a workflow that cannot justify that stewardship is implicitly declaring how much it should be relied on. Unfunded evaluation does not degrade gracefully; it stops silently, while its last passing result keeps circulating as if current.

Ownership placement matters less than ownership reality: the stewardship can sit with the product team, a platform group, or a quality function, provided the reliance authority stays with the workflow owner and the challenge right stays independent of the team whose system is being judged. The anti-pattern is evaluation owned entirely by the builders — not because builders are dishonest, but because the evaluation's job includes disappointing them.

What the record should show

A living evaluation leaves a current trail:

- A case set with provenance, subgroup coverage, and a stated population boundary.
- Thresholds by consequence class, written before results, with review load included.
- Monitoring signals with response thresholds — and the responses they triggered.
- Change events matched to regression runs and version-tied evidence.
- Incidents and near misses converted into case classes, with recurrence tracked.
- Reliance changes — narrowings, pauses, restorations — tied to evidence, with dates.

Auditors and enterprise buyers increasingly ask for exactly this trail, which gives the loop a second constituency beyond the operation itself. An organization that can produce current, versioned, consequence-weighted evaluation evidence for its production AI answers due-diligence questionnaires from its records; one that cannot answers them with meetings. The difference is measured in sales-cycle weeks and audit findings, which is often the argument that finally funds the stewardship.

The staleness check is the cheapest audit available in this field: compare the date of the last evaluation evidence with the date of the last material change to the system, the data, or the workflow. A gap means the operation is running on faith with paperwork. The loop's entire purpose is to make that gap structurally impossible — every change either ran through the evidence machinery, or it is visibly and attributably outstanding on a named owner's list.

The strongest counterargument

No evaluation can prove universal safety or future performance. It provides bounded evidence for a named use under observed conditions. Honest limits, maintained cases, and authority to change reliance are more valuable than a large one-time score.

The cost objection — that living evaluation is overhead a lean team cannot carry — inverts the comparison. The alternative to the loop is not zero evaluation cost; it is evaluation cost paid irregularly, at incident prices, with interest: the degraded quarter nobody measured, the trust collapse after the visible failure, the re-validation

scramble when an auditor asks which evidence covers the current system. The loop is the amortized version of a bill the operation pays either way. Proportionality governs the size of the payment; nothing governs whether.

The shift

The shift is from evaluation as a launch gate to evaluation as an operating control — from the closed question 'did it pass?' to the standing question 'what does the current evidence cover, and does our reliance still fit inside it?' The first question gets answered once and ages. The second gets answered continuously, and the machinery that answers it is the same machinery that makes the answer matter: cases that mirror the operation, thresholds that respect consequence, monitoring that listens, change that re-tests, incidents that teach, and an owner who can change what the workflow is allowed to lean on.

The habit the loop builds is the one that transfers: treating 'the system works' as a claim with a date on it. Everything else in AI governance — autonomy grants, challenge protocols, reliance boundaries — leans on that dated claim, and the loop is the machinery that keeps the date current.

My classification workflow runs today with that machinery around it. The model has changed four times since; the document mix has changed twice; the evaluation has changed with all six. The number that matters is the one that has not recurred: weeks of silent degradation. The loop never made the system better — it made the organization aware, every week, of exactly what the system was.

Notes and sources

[1] NIST, "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile," July 2024. https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.600-1.pdf

[2] Blake Bullwinkel et al., "Lessons From Red Teaming 100 Generative AI Products," Microsoft Research, January 2025. https://www.microsoft.com/en-us/research/publication/lessons-from-red-teaming-100-generative-ai-products/

About the author

Marco Policani is an enterprise portfolio, PMO, and AI operating-governance leader. He builds portfolio governance systems where AI carries the analytical load and named humans own every decision — an approach documented across case studies, walkthroughs, and working governance guides on his portfolio site.

Portfolio & governance library: policani.net/governance · Full portfolio: policani.net · LinkedIn: linkedin.com/in/marcpolicani