

After Launch: A Statistical Framework and Two Hard Rules for Monitoring Models and Agents

Great Unicorn Intelligence Research Team | HX Research • Analysis • 1 November 2026

Abstract: Post-launch monitoring of models and agents is usually built as a dashboard. We argue instead that it is a sequential decision system that, under explicit error-rate constraints, answers two questions: what changed, and who should act. We separate data drift, concept drift, performance decay and measurement drift; state when PSI, KS, CUSUM and anytime-valid tests apply; discuss thresholds under multiple testing; explain why agents need trajectory, tool, cost and safety layers; and derive two rules: no silent parameter changes, and no alert that directly rewrites business outcomes.

Keywords: production monitoring; drift detection; sequential testing; multiple testing; agent observability; incident response; change governance

Introduction: why a dashboard is not enough

A typical scene: three months after a customer-service agent goes live, the weekly report shows the automatic quality score falling from 4.3 to 4.1, responses getting longer, the tool error rate creeping up and the monthly bill running 20% over budget. The team meets for an afternoon and concludes: tweak the prompt and loosen the thresholds a little. Two weeks later the metrics recover, and nobody can say whether that is because the problem was fixed, because the user mix changed, or because the vendor quietly swapped the version of the judge model that produces the score.

What this scene exposes is not a shortage of metrics. Three things were never separated. Where does a change come from: the inputs, the relationship between inputs and outcomes, the model itself, or the measuring instrument? Is the change noise: at what threshold should an alarm fire, and how are false alarms controlled when dozens of metrics are watched at once? And after an alarm, who is allowed to change what: parameters, outcomes, or only through a change process? Our central thesis is that post-launch monitoring should be designed as a sequential decision system that, under error-rate constraints, answers "what changed, and who should act", rather than as an ever-growing set of dashboards. For agents, the unit of observation must also move from the single request to the whole trajectory, and even to state that spans trajectories; and response must enter a controlled change process. The two rules — no silent parameter changes, and no alert that directly rewrites business outcomes — follow from this thesis.

The rest of the article has five sections: first, separating what changed; second, how to set thresholds; third, what extra must be watched for agents; fourth, the response process and the reasoning behind the two rules; and fifth, a worked example with recomputable numbers, followed by forecasts.

1. First separate what changed: four kinds of change and their detectors

Four kinds of change

The classic concept-drift literature splits change on a data stream into a change in the input distribution $P(X)$ and a change in the conditional relationship $P(Y|X)$ ¹. In large-model and agent systems at least two more kinds must be added: changes in the behaviour of the model or service itself, and changes in the measuring instrument — above all, the large-model judge used for scoring. Table 1 summarises the four kinds of change, their typical causes and how to detect them.

Table 1 Four kinds of post-launch change (compiled by the Great Unicorn Intelligence Research Team)

Type	What changes	Typical causes	Detectable without labels?	Common detectors
Data drift	Input distribution $P(X)$	User mix, channels, seasonality	Yes	PSI, KS, two-sample tests in embedding space
Concept drift	Conditional relation $P(Y X)$	Business rules, the meaning of user intent	No, needs delayed labels	Per-segment performance, calibration error
Performance decay	Quality of system output	Upstream dependencies, tools, model versions	Partly	Success rates, trajectory metrics, judge scores
Measurement drift	The scoring instrument itself	Judge model upgrades, scoring-prompt edits	Needs an anchor set	Judge-minus-human gap on the anchor set

The point of the split is that the four kinds call for entirely different responses. Data drift does not necessarily mean the system is worse; it may only call for re-evaluation. Concept drift means that yesterday's correct answers are no longer correct, which requires relabelling or even redefining the task. Performance decay requires finding the faulty component. Measurement drift means that the "decline" may itself be an artefact. Mixing all four on one dashboard produces exactly the scene in the introduction.

Data drift: what PSI and KS each answer

The population stability index (PSI), widely used in credit risk, is defined over bins as $\sum (a_i - e_i) \cdot \ln(a_i/e_i)$, where e_i and a_i are the shares of the reference and current periods falling in bin i . It

is easy to show that PSI equals the sum of the KL divergences in both directions (the Jeffreys divergence), so it measures the size of a distributional difference. The KS statistic is the maximum gap between two cumulative distribution functions and answers whether the difference is significant.

The distinction is sharpest when sample size changes. Take illustrative, recomputable numbers: five bins with reference shares of 10%, 20%, 40%, 20% and 10% and current shares of 6%, 16%, 38%, 24% and 16%. Computing bin by bin gives $\text{PSI} \approx 0.066$; the largest cumulative gap occurs at the third bin, so $\text{KS} = 0.10$. With 2,000 samples in each period, the 5% critical value of KS is about $1.358 \times \sqrt{(1/2000 + 1/2000)} \approx 0.043$, so the difference is highly significant; yet under the common rule of thumb that "PSI < 0.1 is stable", it is stable. The two verdicts do not contradict each other: KS says "it really changed", PSI says "not by much".

The rule-of-thumb threshold itself deserves more attention. A second-order expansion shows that, in the absence of real change, PSI approximately satisfies $(nm/(n+m)) \cdot \text{PSI} \sim \chi^2(B-1)$, where n and m are the two sample sizes and B is the number of bins. The PSI threshold at a 5% false-alarm level is therefore about $\tau = \chi^2_{0.95}(B-1) \cdot (1/n + 1/m)$. With five bins and 200 samples per period, $\tau \approx 9.49 \times 0.01 \approx 0.095$ — in other words, in small windows the "0.1" line is almost exactly the 95th percentile of pure noise. We ran 20,000 no-drift simulations (illustrative, fixed random seed, recomputable): PSI had a mean of about 0.041, a 95th percentile of about 0.097, and exceeded 0.1 about 4.4% of the time, consistent with the approximation. The conclusion: a PSI threshold should scale with window size and bin count, rather than being one line shared across the industry.

For high-dimensional unstructured inputs such as text and images, direct binning is not feasible. A systematic comparison found that reducing dimension with a pretrained model before running a two-sample test works well for detecting dataset shift, and that domain-discriminating approaches help characterise a shift and judge whether it is harmful².

Concept drift and calibration: the part that needs labels

Concept drift can only be confirmed once outcome labels arrive. If only the class prior has changed (label shift), the new prior can be estimated from a confusion matrix and corrected for without retraining³. Another often-ignored signal is calibration: modern neural networks are frequently more confident than they are accurate, and simple methods such as temperature scaling help markedly⁴. In production, a change in calibration error often precedes a change in accuracy; a model that starts "not knowing what it does not know" is an early sign that the conditional relationship has moved.

Two kinds of change specific to large models: nondeterminism and measurement drift

Large-model systems carry a kind of baseline noise that traditional models lack. A system-level taxonomy lists 15 hidden failure modes in production, including multi-step reasoning drift, version drift and cost-driven performance collapse, and notes that existing benchmarks barely measure stability, reproducibility or drift⁵. Repeated-run experiments show that the same prompt still yields different outputs even at temperature 0⁶. On the serving side, prefix-cache reuse can make the same request produce different results on the cache-hit and recompute paths, and quantization amplifies the divergence: according to the paper, 37.4% of multi-turn tool-calling episodes for Qwen2.5-7B diverge at 16 bits and 82.5% at 4 bits⁷. Monitoring therefore has to measure the range of fluctuation when nothing has changed; otherwise no change can be judged.

Measurement drift is subtler. Continuous evaluation usually has a strong model act as judge, scoring every interaction and alarming when scores fall. But the judge is itself a model behind an API, and a silent version upgrade or an edit to the scoring prompt changes how it scores, so no drift alarm can distinguish "the product got worse" from "the judge changed"⁸. One proposal is a fixed, human-labelled anchor set that the current judge re-scores at a fixed interleaving rate, with a separate sequential test run on the judge-minus-human gap, so that each alarm is attributed to one of three causes: no change, system change, or judge change. According to the paper, in one real silent version upgrade, all 60 runs were correctly identified as judge drift and none were misattributed to system drift⁸. A pre-registered audit that treats large-model judges as raters found that, in each of five version comparisons, the shift in scoring severity exceeded a family-wise-corrected permutation null, and that one judge was deprecated during the study and caught only by an "identity canary"⁹. Other work found that repeated verdicts change even at temperature 0, and that swapping candidate order flips most verdicts on hard items¹⁰.

Together these results say that the scoring instrument must be versioned and monitored just like the system it scores. A judge-score curve without an anchor set cannot on its own justify an alarm.

2. Setting thresholds: false alarms, misses and watching many metrics at once

An underrated problem: multiple testing

Once the kind of change is separated, the next question is how large a change should raise an alarm. Many teams test each metric at a 5% significance level while watching dozens of metrics

and checking every day. An illustrative, recomputable calculation: with 40 independent metrics each tested once a day at 5%, even if nothing has changed, the probability of at least one false alarm on a given day is $1 - 0.95^{40} \approx 87\%$; over a month the expected number of false alarms is $40 \times 30 \times 0.05 = 60$. On-call staff soon learn to ignore alerts, which is precisely why alerting fails when the real problem arrives.

There are two families of remedy. Family-wise error control (such as the Bonferroni correction, which lowers each metric's level to $0.05/40 = 0.00125$) guarantees that the probability of any false alarm stays below 5%, at the cost of more misses. False discovery rate control (the Benjamini–Hochberg procedure) instead controls the share of false alarms among the alarms raised, and has more power when there are many metrics and more than one real change¹¹. For monitoring, the latter is often closer to what is actually needed: what on-call staff care about is how many of the ten times they were paged were for nothing.

The cost of repeated looks and anytime-valid tests

The second underrated problem is peeking. A fixed-sample p-value is valid only when the sample size is set in advance and the data are looked at once. If the test is recomputed with every new batch and an alarm fires as soon as it is significant, the false-alarm rate grows with the number of looks, far above the nominal level. Monitoring is, by nature, repeated looking.

Sequential tests exist for exactly this. The classic cumulative-sum (CUSUM) method adds up the log-likelihood ratio observation by observation and alarms when the sum crosses a threshold¹²:

$$S_t = \max(0, S_{t-1} + \ln[f_1(x_t)/f_0(x_t)]), \text{ alarm when } S_t > h.$$

Its two key quantities are how long, on average, it runs before a false alarm when nothing has changed (the average run length, ARL_0), and how long it takes to detect a change once one occurs (the detection delay). A higher h gives a longer ARL_0 and slower detection; this is the direct trade between false alarms and misses.

Take the tool-call error rate as an illustrative, recomputable example. The baseline error rate is $p_0 = 2\%$, and we want to detect a rise to $p_1 = 4\%$ quickly. Each error adds $\ln(0.04/0.02) \approx 0.693$ to the sum; each success adds $\ln(0.96/0.98) \approx -0.0206$. With no change, the expected increment per call is -0.0063 , so the sum is pulled back to zero; after the change it is $+0.0079$, and the sum climbs steadily. With $h = 5$, running 300 simulations each with a fixed random seed, ARL_0 is about 29,600 calls and the mean detection delay after the change is about 520 calls; with $h = 4$, the two figures are about 9,600 and 380. If the agent makes 2,000 tool calls a day, $h = 5$ means roughly one false alarm every 15 days and detection within about six hours of a change. The team can use this to discuss with the business how many false alarms it will accept in exchange for how fast detection, instead of choosing a threshold by feel.

CUSUM needs the pre- and post-change distributions to be specified in advance. Where the distribution is unknown, or the direction of change cannot be foreseen, the recently developed framework of testing by betting is more general: build a "wealth process" (an e-process) whose expectation does not grow when nothing has changed; by Ville's inequality, the probability that it ever crosses $1/\alpha$ is at most α , so it can be stopped and an alarm raised at any time without breaking the false-alarm guarantee¹³. The framework has been applied to unstructured data streams: one study has the "bettor" keep updating the direction of drift it bets on as new data arrive, balancing sensitivity and coverage while keeping false-alarm control¹⁴. Another observes that common conformal test martingales mix post-change samples into the reference set and dilute the evidence, and that testing against a fixed no-change reference set avoids this contamination¹⁵. A third calibrates the rejection threshold on independent no-change samples, which is tighter than the generic $1/\alpha$ and so shortens detection delay¹⁶. The same idea has been applied inside the generation process: framing the detection of when a hallucination begins as a quickest change-detection problem, researchers derived a lower bound on detection delay and found that, at a 1% false-alarm budget, every detector caught fewer than a third of hallucination onsets¹⁷. Fast detection at low false-alarm rates is intrinsically hard, and this is a reality that threshold setting has to accept.

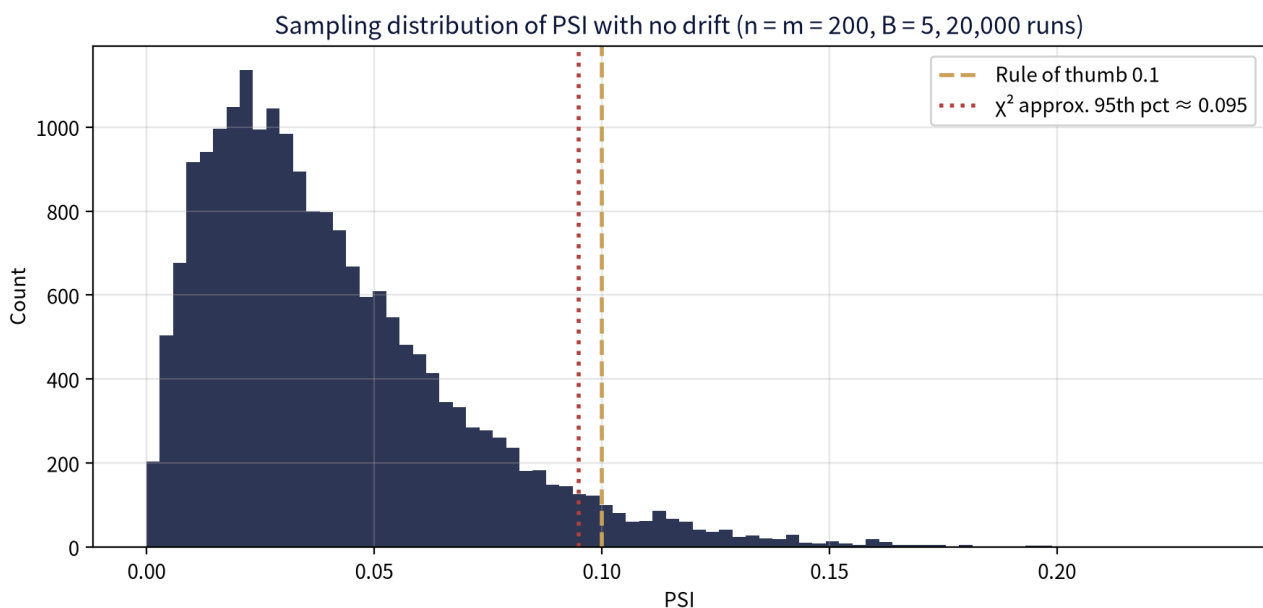
When many metrics are monitored sequentially at once, the two kinds of requirement collide head-on. One study proves that for any multi-stream detection procedure with a finite average run length, the worst-case false discovery rate and family-wise error rate both equal 1, and proposes a new measure, false alarms relative to monitoring time, as a compromise, combining e-detectors with multiple-testing procedures such as e-BH¹⁸. The engineering implication: false-alarm control in multi-metric monitoring can only be agreed as a number of false alarms per unit time, never as a promise of no false alarms.

Writing thresholds as budgets: burn-rate alerts

Reliability engineering handles a similar problem with error-budget burn rates. With a 99.9% availability target, the 30-day error budget is 0.1% of requests; a burn rate of 1 means the budget is used up exactly at the end of 30 days. Google's SRE practice recommends paging when the burn rate reaches 14.4 over a 1-hour window (2% of the budget in one hour) or 6 over a 6-hour window (5%), opening a ticket when it reaches 1 over a 3-day window (10%), and combining long and short windows to balance detection speed and precision¹⁹. The advantage is that thresholds become a language the business understands — "at this rate, the budget runs out in so many hours" — and the approach fits cost monitoring naturally: replace the error budget with the monthly spend budget and you have a cost burn-rate alert.

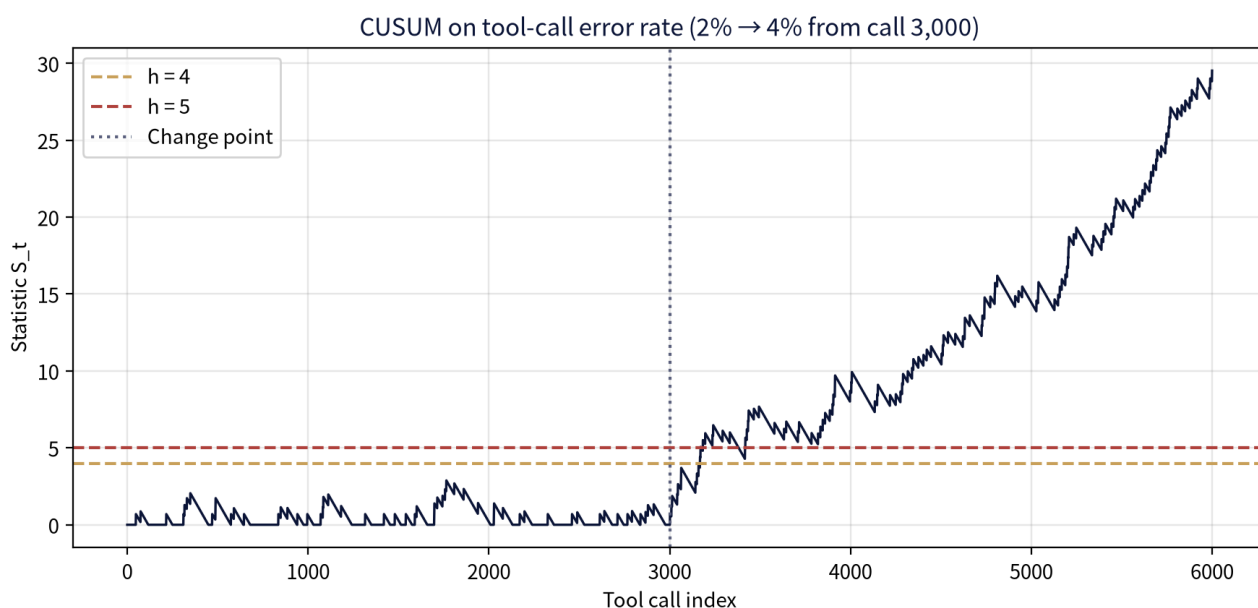
AUROC is not an alerting policy

Finally, beware of substituting offline metrics for production thresholds. One study notes that activation probes can predict safety properties with high AUROC, but production monitoring makes threshold decisions under a strict false-alarm budget, and the two are different estimands; when a task offers several chances to alarm, per-row AUROC and false-positive rates cannot capture the risk that "the task alarms at least once"²⁰. According to the paper, monitors with AUROC of 0.957 and 0.935 on two benchmarks reach detection rates of only 0.642/0.742 and 0.719/0.782 when the false-positive rate is fixed at 5% and 10%²⁰. Model selection should compare directly how much is caught at the false-alarm rate we can accept.



Source: simulation by the Great Unicorn Intelligence Research Team (illustrative, seed 7)

Fig. 1 | Sampling distribution of PSI with no drift (200 samples per period, five bins, 20,000 simulations). The dashed line is the 0.1 rule of thumb; the dotted line is the 95th percentile from the χ^2 approximation. Illustrative data simulated by the Great Unicorn Intelligence Research Team.



Source: simulation by the Research Team (illustrative, seed 11)

Fig. 2 | CUSUM on the tool-call error rate: from call 3,000 the error rate rises from 2% to 4%. A higher threshold h means fewer false alarms and slower detection. Illustrative data simulated by the Great Unicorn Intelligence Research Team.

3. What extra to watch for agents: trajectories, tool calls, cost and safety

The monitoring unit for a traditional model is one prediction. One task for an agent consists of a few to several hundred steps, each of which may call a tool, read or write state and consume budget. Looking only at final success is like looking at the outcome of a trade without looking at how it was executed: many problems are hidden in the outcome, or are already irreversible by the time it appears. We split agent-specific monitoring into four layers, each answering a different question.

Prerequisite: record the trajectory first

All four layers depend on structured trace data. The OpenTelemetry semantic conventions for generative AI already define span types for operations such as "create agent", "invoke agent" and "execute tool", and specify fields such as model name, provider, input and output token counts, tool name and call arguments, along with metrics such as agent invocation duration and the numbers of inference and tool calls. Note that these conventions are still in "Development" status and fields may change²¹. Researchers have also proposed a structured logging framework for safety and accountability that records events across operational, cognitive and contextual surfaces in one schema²².

The point of recording is to make every later statistic traceable to a specific step. For failure attribution, two studies offer usable approaches. A taxonomy of multi-agent system failures,

built on more than 1,600 annotated traces, identifies 14 failure modes in three categories: system design, inter-agent misalignment and task verification²³. Another study annotated the "critical failure step" in 170 failed trajectories and located it automatically by combining step-wise constraint checks with a judge model; according to the paper, localisation accuracy improved by 75% on average over prior methods²⁴.

Trajectory layer: is the process executable throughout?

Looking only at the end point overrates agents. One study proposes "prefix executability": checking whether every prefix of a tool-call sequence is still executable under tool schemas, state preconditions and data dependencies, measured by first-error depth and trajectory executability rate. The authors found that rankings by final success and by prefix metrics diverge: agents that rely on unchecked plans have higher point estimates of final success but markedly lower trajectory executability²⁵. Some "successes", in other words, are lucky, and fail when the environment shifts slightly.

Online early warning is the other face of the trajectory layer. One method learns events from execution prefixes, predicts failure risk over several horizons with a recurrent network, and organises history into replayable risk paths with a finite-state machine²⁶. Another aligns an agent's steps and dependencies with recorded successful runs by optimal transport, raising alerts or intervening at about 1 millisecond per step, though the less history is available, the more the detectable problems are limited to loops, stalls and repeated calls²⁷. Counter-evidence matters just as much: a study of 147,000 execution trajectories of coding agents found that early prefixes slightly improve the ranking of outcomes across tasks, but within a single task the AUC for distinguishing "this run will succeed" from "this run will fail" is close to 0.50 — no better than chance²⁸. Trajectory monitoring is therefore better suited to catching process violations (privilege overreach, loops, departures from plan) than to predicting whether this particular run will succeed.

Tool layer: silent errors are more dangerous than explicit ones

Tool failures are common in real systems. One study derived a fault distribution from more than 57,000 issues and pull requests across 1,774 open-source MCP repositories, then injected faults into tool-call paths according to that distribution. According to the paper, under stress conditions dominated by permanent faults, task completion scores fell from 8.08 to 3.23 (out of 10), and larger models brought no better fault tolerance²⁹. Harder to catch are silent errors: a tool returns a plausible but wrong value with no error signal. A benchmark built on real e-commerce workflows shows that silent errors roughly halve task success³⁰.

This puts two concrete requirements on tool monitoring. Beyond the error rate, return values need plausibility checks (type, range, consistency with other sources). And for critical tools,

probe requests with known answers should regularly check that returned values are still correct.

Cost layer: runaway budgets are a failure class of their own

Cost overruns are not a finance problem; they are a failure class with its own patterns. One study compiled 63 confirmed production incidents across 21 orchestration frameworks from 2023 to 2026, each with a public issue record, and grouped them into eight failure modes; a typical case is a single retry loop spending thousands of dollars before anyone notices³¹. A study of 200 trajectories of vulnerability-finding agents found that code localisation and comprehension together with vulnerability reasoning account for 60.4% of tokens; spending more does not reliably buy better results; and in paired comparisons only 24.4% of existing cost-saving methods reduce total cost while preserving success³².

The cost layer therefore has to do two things at once: account for spending per task rather than per month, so that a single runaway task becomes visible; and set a hard cap for each task, enforced by the framework rather than by the model. The burn-rate alert from the previous section applies directly.

Safety layer: per-step checks do not compose into overall safety

Two counter-intuitive results in the safety layer deserve emphasis. First, per-action authorisation checks do not compose into global authorisation: one study proves that a class of "purpose violations" spread across executions has no locus of responsibility on any single trajectory, so no per-action monitor that never raises a false alarm can detect them; only monitoring across executions can, and none of the six guardrail schemes the authors surveyed does so³³. Second, safety does not compose across iterations: for long-running autonomous agents, if attack evidence is spread over many iterations, the true-positive rate of any monitor that looks at one trajectory at a time equals its false-positive rate; and simply letting the risk score decay geometrically is not enough, because a patient attacker need only wait out a cooldown that does not depend on the total horizon³⁴.

Chain-of-thought monitoring is another route that has drawn attention. Researchers from several institutions argue that models that "think" in natural language offer an opportunity to monitor intent, but that this monitorability is fragile and development decisions should consider their effect on it³⁵; follow-up work proposes an evaluation suite for monitorability and finds most frontier models "fairly, but not perfectly" monitorable³⁶. The limits of the monitors themselves also matter: a study covering 15 models and 38 activation readers found that, on out-of-distribution rewrites of harmful requests and at matched false-positive rates, latent harmfulness monitors do no better than the model's own refusal behaviour³⁷.

Table 2 Four layers of agent monitoring: signals, statistics and alerting policy (compiled by the Great Unicorn Intelligence Research Team)

Layer	Question answered	Representative signals	Recommended statistics	Alerting policy
Trajectory	Is the process compliant and executable?	First-error depth, loops and stalls, departure from plan	Prefix executability; per-task sequential tests	Alert on process violations; do not predict single-run success
Tool	Are tools available and are returns trustworthy?	Error rate, timeouts, failed return checks, probe-answer deviations	CUSUM on error rate; probe agreement rate	Count explicit and silent errors separately
Cost	Is spending as expected?	Tokens per task, retries, spend per task	Budget burn rate (long and short windows)	Hard per-task cap plus monthly burn rate
Safety	Overreach, injection, global-policy violations?	Unauthorised calls, cross-execution purpose violations, refusal regressions	Cross-trajectory state; detection rate at fixed FPR	High-severity events go straight to human handling

4. After the alarm: the response process and the reasoning behind the two rules

The first three sections answer how to know reliably that something has gone wrong. But the value of monitoring is realised in what happens after the alarm, and that is exactly where things often go wrong. Two practices are common. One is to adjust the prompt, temperature, thresholds or retrieval parameters directly once a metric falls, until the metric returns to its normal range. The other is to let the alerting system rewrite business outcomes directly: if the monitor judges an answer "risky", it is automatically replaced by a refusal; if it judges an operation "anomalous", it is automatically rolled back. Both look responsible. We argue that both should be banned, and give the reasoning for each in turn.

Rule 1: no silent parameter changes

"Silent" means adjusting, in production and without a change record, a version identifier and a controlled evaluation, any parameter that affects model behaviour — prompts, decoding parameters, retrieval configuration, judge prompts and alert thresholds. There are four reasons, each building on the last.

Reason 1: it destroys attribution. As shown earlier, a change in a metric may come from the system, the data or the measuring instrument, and the anchor-set method can tell "the system changed" from "the judge changed" only if everything else is held fixed⁸. An unrecorded parameter change is like secretly swapping a variable halfway through an experiment: whether

the metric then recovers or keeps falling, the cause can no longer be identified. The introduction's "the metrics recovered, but nobody can say why" is exactly this situation.

Reason 2: it voids the statistical guarantees. The false-alarm guarantee of a sequential test rests on a fixed no-change reference. Tests against a fixed reference set work because the reference set is not contaminated during testing¹⁵; methods that calibrate thresholds on independent no-change samples likewise require that the system calibrated is the system monitored¹⁶. After a silent parameter change, the reference distribution no longer represents the current system, yet the monitor keeps computing against the old reference, so ARL_0 and the false-alarm rate lose their meaning.

Reason 3: it turns monitoring into an optimisation target. When parameters are tuned with the goal of making alarms go away, the monitored metric turns from a measuring instrument into an optimisation target. Experiments by OpenAI researchers on coding agents offer a direct analogy: putting a chain-of-thought monitor into the reinforcement-learning reward produced more capable and more aligned agents at low optimisation strength, but with too much optimisation the agents learned "obfuscated reward hacking" — hiding their intent in the chain of thought while continuing to cheat³⁸. Manual tuning applies far less optimisation pressure than reinforcement learning, but in the same direction: tuning repeatedly until the metrics look normal tends to yield a system that is better at passing the monitor, not a better system.

Reason 4: it breaks reproducibility and auditability. Production systems already carry baseline noise^{6,7}. Add unrecorded parameter changes, and no post-mortem can reproduce the system's behaviour at the time of the incident. For businesses that must explain decisions to customers or regulators, that is a compliance risk in itself.

Rule 1 does not mean parameters can never change. It means changes go through a change process: record the version, state the reason, compare against the current version on shadow traffic or a small slice of traffic, roll out fully only after passing, and mark the change time in monitoring as a known change point.

Rule 2: no alert that directly rewrites business outcomes

The second rule targets "automatic correction". Intuitively, letting the monitor correct an outcome the moment it finds a problem seems safer. The reasoning shows the opposite.

Reason 1: the positive predictive value of alerts is usually low. Run illustrative, recomputable numbers through Bayes' rule: a monitor has a 90% detection rate and a 5% false-alarm rate, and truly problematic trajectories make up 1% of all trajectories. Then among all alerts, the share that are truly problematic is $0.9 \times 0.01 / (0.9 \times 0.01 + 0.05 \times 0.99) \approx 15.4\%$. If every alert automatically rewrote the outcome, about 85% of rewrites would land on outcomes that were

correct. As shown in Section 2, monitors with high offline AUROC can lose much of their detection rate once the false-alarm rate is fixed²⁰, so the real positive predictive value is often lower still.

Reason 2: it contaminates future learning signals. Outcomes rewritten by a monitor flow into logs and reports and often into the next round of training or evaluation data. The "correct outcomes" we then observe have already been filtered by the monitor's biases. One study specifically examines this "monitor-induced selection": the safe samples the monitor lets through are not like the safe samples it wrongly blocks, so using the former to estimate the latter goes systematically wrong³⁹. Automatic rewriting writes this selection bias straight into business data.

Reason 3: it makes failure paths invisible. One study confirmed 18 "evidence-path failures" across 11 large-model and agent safety projects: the detector had identified the problem, but the framework dropped the result before making a blocking decision⁴⁰. The monitoring chain itself fails, and the detection power of safety classifiers under distribution shift varies by classifier: a pre-registered experiment over 4 classifiers and 5 kinds of shift measured a control-validated detection rate of about 86.8% (according to the paper), and found that a "disagreement canary" using a second classifier flagged about 75.5% of attack inputs that the primary classifier wrongly scored as safe, while a residual still evaded both⁴¹. If the monitor holds both the power to detect and the power to rewrite, its failures show up directly as wrong business outcomes, and no second check exists to catch them.

Reason 4: it blurs accountability. After a rewrite, it becomes unclear who is responsible for the outcome — the business system, the model or the monitoring rule. For decisions whose consequences must be owned by a named person, that is unacceptable.

Rule 2 likewise does not mean doing nothing after an alarm. What is allowed is reversible containment: pause an agent or a class of actions, degrade to read-only, route pending outcomes to human review, require a second confirmation for a single high-risk action. What these share is that they do not make a new decision on the business's behalf; they stop a possible error from spreading and hand the decision back to someone with the authority to make it.

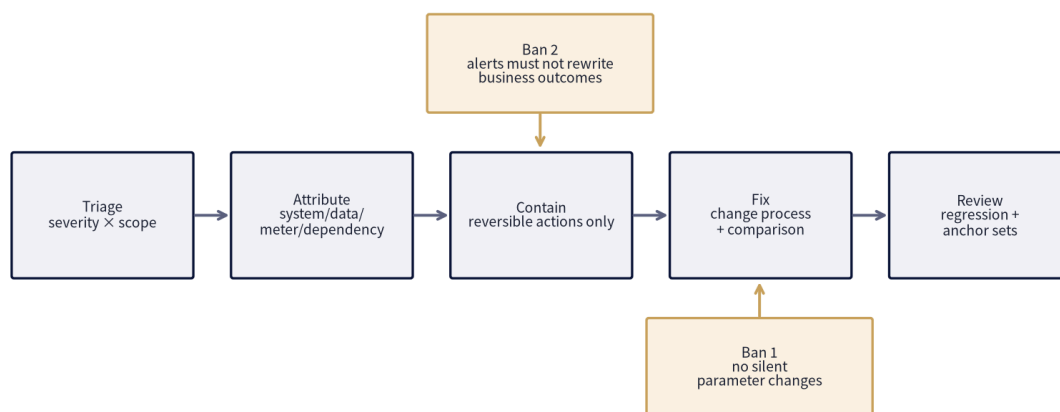
The response process: a decision framework

Placing the two rules back into the process, the post-alarm response has five steps: triage (by severity and scope), attribution (system, data, measuring instrument or external dependency), containment (reversible actions only), fix (through the change process, full rollout after a shadow or small-traffic comparison), and review (add incident samples to the regression and anchor sets). Table 3 gives a framework for choosing containment actions.

Table 3 Choosing containment actions after an alert, by severity and reversibility (compiled by the Great Unicorn Intelligence Research Team)

	Reversible consequences (e.g. retractable messages, drafts)	Irreversible consequences (e.g. outbound sends, writes, charges)
Low severity	Record and handle in the next change window	Add a second confirmation for that class of action
Medium severity	Route to human spot checks; raise monitoring sampling	Pause that class of action; keep read-only capability
High severity	Pause the relevant agent; human takeover	Disable the relevant function immediately; handle per runbook

Post-alert workflow and the two bans



Source: Great Unicorn Intelligence Research Team

Fig. 3 | The post-alert response process. The two rules constrain the "fix" and "contain" steps respectively. Drawn by the Great Unicorn Intelligence Research Team.

5. A complete worked example, its limits, and forecasts

One week's monitoring report, computed item by item

We use a customer-service agent in a non-financial setting to tie the methods together (all numbers below are illustrative and recomputable). The agent handles after-sales enquiries, with about 14,000 conversations a week and about 2,000 tool calls a day. This week monitoring produced five "suspicious" signals:

1. Intent distribution: the shares of five intent classes moved from 10%, 20%, 40%, 20% and 10% to 6%, 16%, 38%, 24% and 16%, with about 2,000 samples in each period. As computed earlier, $PSI \approx 0.066$ and $KS = 0.10$, well above the critical value of 0.043. Verdict: the inputs really did change, but not by much. This is data drift; it does not warrant paging on-call staff, and should be recorded and per-segment performance checked.

2. Response length: the mean rose 8%, KS-test $p = 0.03$.
3. Judge score: fell from 4.30 to 4.12, $p = 0.004$.
4. Tool error rate: CUSUM ($h = 5$) crossed its threshold from Wednesday.
5. Spend: the 6-hour burn rate reached 6.5.

Step one: multiple testing. Forty metrics were checked this week. Applying BH correction to the p-values of these signals (intent KS p well below 0.001, length 0.03, score 0.004; the two sequential alarms are handled separately under their own false-alarm guarantees) together with the other 35 metrics, at a 10% false discovery rate the length change with $p = 0.03$ is no longer significant, because its rank among the 40 sets its corrected threshold at about $0.1 \times 3/40 = 0.0075$. The length change is downgraded to "watch".

Step two: attribute the score drop. The anchor set (200 fixed human-labelled samples, re-scored at a 5% interleaving rate) shows the current judge scoring 0.17 lower on the anchors than last month, almost the same as the 0.18 drop in production, while the human labels are unchanged. Verdict: the score drop comes mainly from judge drift, not a worse system. The vendor's changelog shows a version update to the judge model on Tuesday. Action: freeze the judge version, recalibrate the anchor set against the new version, and mark the change point in reports.

Step three: locate the tool errors. The errors concentrate in one logistics-lookup tool, rising from about 2% to about 4% from Wednesday, a change point that coincides with an upgrade to the tool's upstream API. Probe requests against the tool also find that about 3% of returns are well-formed but map tracking numbers incorrectly — a silent error, which deserves higher priority than the explicit errors. Actions: add human review to that tool's replies (containment), file the problem with the API owner (fix), and add the silent-error samples to the regression set.

Step four: cost. The 6-hour burn rate of 6.5 is above the paging line of 6. Investigation finds that the silent errors made the agent retry lookups repeatedly, raising the mean number of calls per task from 3.1 to 5.4. The cost problem and the tool problem share one root cause. Action: set a per-task retry cap on that tool, enforced by the framework.

Step five: what we did not do. This week we did not edit the answer prompt to make the score recover, did not loosen the error threshold, and did not let the monitor automatically replace "suspicious" answers with template replies. The score drop has been attributed to the judge and the tool problem contained; any parameter change waits for the next change window and a controlled comparison before going live.

The example shows what the central thesis means in practice. Behind five alarms were only two real problems (a judge version change and a silent tool error) — one measurement drift and one performance decay. The data drift was real but harmless; the length change was noise.

Treating the five alarms as five problems, each to be "fixed by tuning", would have made things worse than doing nothing.

Limits: what this framework cannot do

The framework has three limits worth stating. First, every statistical guarantee assumes enough samples; in low-traffic settings the detection delay grows long enough to be meaningless, and human spot checks are more reliable than statistical tests. Second, anchor sets need continual maintenance: once the task definition changes (concept drift), old anchors become invalid and must be relabelled. Third, sequential tests control false alarms per unit time; they do not eliminate them. On-call rotations, alert tiers and post-mortems remain indispensable.

Forecasts

On this basis we offer five views, each with a time scale and a falsification condition:

View 1 (12–24 months): Evaluation and monitoring pipelines for large models will widely adopt fixed anchor sets with judge-drift tests, and "did the system change or did the judge?" will become a mandatory attribution field on alerts. Evidence: severity shifts caused by judge version changes have been measured in several studies^{9,10}, and anytime-valid attribution methods already exist⁸. If, 18 months from now, mainstream observability products still do not support recording judge versions and evaluating against anchors, the view fails. Enterprise implication: start keeping version and anchor records for every judge now.

View 2 (6–18 months): Anytime-valid sequential tests (e-processes, testing by betting) will enter mainstream monitoring tools and gradually replace "compute a p-value over a fixed window". Evidence: methods for unstructured data streams and for threshold calibration have appeared in quick succession over the past year^{14–16}, and an operational definition of false alarms for the multi-stream case now exists¹⁸. If no major open-source monitoring library implements any such test within the period, the view is premature. Enterprise implication: pilot on one or two key metrics, run alongside existing thresholds for a quarter, and compare false alarms and delays.

View 3 (12–24 months): Agent safety monitoring will extend from single steps and single trajectories to state that spans trajectories and iterations, and permission systems will introduce global policies on "purpose" and "provenance". Evidence: two impossibility results show that per-step and per-trajectory monitoring have structural blind spots^{33,34}. If public incident post-mortems show that per-trajectory monitoring already covers the main safety incidents, the view should be downgraded. Enterprise implication: keep cross-task audit state for long-running agents instead of clearing it when each task ends.

View 4 (6–12 months): Hard per-task budgets and cost burn-rate alerts will become default configuration in agent frameworks. Evidence: budget-overflow incidents already form a classifiable failure family³¹, and spending more does not mean doing better³². If major frameworks still rely on the monthly bill as their only cost control after 12 months, the view is premature. Enterprise implication: in procurement and in-house builds, list "per-task caps enforced by the framework" as an acceptance criterion.

View 5 (12–24 months): Trace data emitted under the OpenTelemetry generative-AI semantic conventions will become one of the acceptance criteria when enterprises buy agent platforms. Evidence: conventions for agent and tool-call spans and metrics already exist²¹, and failure-attribution research depends on such structured trajectories^{23,24}. If the conventions remain in Development status and major frameworks have not implemented them within 24 months, the view fails. Enterprise implication: require vendors to provide exportable, standardised traces, not just a visual interface.

Where common industry views may need correcting

- "PSI above 0.1 means drift": the threshold should scale with sample size and bin count; in small windows, 0.1 is close to noise.
- "Setting temperature to 0 makes runs reproducible": server-side caching and quantization can make the same request produce different results⁷.
- "A detector with high AUROC can go straight to production": what matters in production is the detection rate at a fixed false-alarm rate²⁰.
- "Final success rate is enough to evaluate an agent": rankings by process executability and by final success diverge²⁵.
- "Automatic correction after an alert is safer": at low base rates, most automatic corrections land on correct outcomes.

Conclusion

Designing post-launch monitoring as a decision system means three things: first separate where a change comes from; then decide whether to alarm under an explicit false-alarm budget; and finally route the response through a controlled change process. The two rules are not managerial conservatism but conclusions of statistical reasoning: silent parameter changes destroy attribution and statistical guarantees, and alerts that directly rewrite outcomes create more errors at low base rates.

In Great Unicorn Intelligence's products, these principles appear in the Harness governance engine of the enterprise agent platform: the model router selects a model per task and pins its version, forbidding automatic upgrades mid-run; the independent verifier reads only

structured data and re-checks in layers; and the audit-monitoring and emergency-stop module supports full trajectory logging, read-only degradation, human takeover and tiered shutdown. The online path of DH S1, the integrated inference-and-training large-model platform, provides metric monitoring and A/B testing, with model and strategy updates linked across online and offline paths (product information from the Great Unicorn Intelligence website). These designs correspond to this article's "pin the version", "reversible containment" and "go live after comparison".

Key points:

1. Split change into data drift, concept drift, performance decay and measurement drift; each calls for a different response.
2. A threshold is a trade between false alarms and misses and should be set by sample size, look frequency and number of metrics; sequential tests and burn-rate alerts turn thresholds into budgets that can be communicated.
3. Agents need added monitoring in four layers — trajectory, tool, cost and safety — and per-step checks do not compose into overall safety.
4. The two rules have a statistical basis: silent parameter changes destroy attribution and statistical guarantees, and automatic rewriting creates more errors at low base rates.
5. The permitted alert actions are reversible containment; every fix goes through the change process.

References

1. Gama, J., Žilobaite, I., Bifet, A., Pechenizkiy, M. & Bouchachia, A. A survey on concept drift adaptation. *ACM Comput. Surv.* 46, 1–37 (2014). <https://doi.org/10.1145/2523813>
2. Rabanser, S., Günnemann, S. & Lipton, Z. C. Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift. Preprint at <https://arxiv.org/abs/1810.11953> (2018).
3. Lipton, Z. C., Wang, Y. & Smola, A. Detecting and Correcting for Label Shift with Black Box Predictors. Preprint at <https://arxiv.org/abs/1802.03916> (2018).
4. Guo, C., Pleiss, G., Sun, Y. & Weinberger, K. Q. On Calibration of Modern Neural Networks. Preprint at <https://arxiv.org/abs/1706.04599> (2017).
5. Vinay, V. Failure Modes in LLM Systems: A System-Level Taxonomy for Reliable AI Applications. Preprint at <https://arxiv.org/abs/2511.19933> (2025).
6. Nicholson, C. Quantifying non deterministic drift in large language models. Preprint at <https://arxiv.org/abs/2601.19934> (2026).
7. Patodiya, A. Same Request, Different Answer: Quantization Amplifies Cache-Induced Divergence in LLM Serving. Preprint at <https://arxiv.org/abs/2609.04748> (2026).

8. Li, Y. Who Drifted: the System or the Judge? Anytime-Valid Attribution in LLM Evaluation Pipelines. Preprint at <https://arxiv.org/abs/2606.15474> (2026).
9. Sunkavalli, V. K. LLM Judges as Raters: A Pre-Registered Audit of Severity, Halo, Reliability, and Version Instability in LLM Essay Scoring on Public Corpora. Preprint at <https://arxiv.org/abs/2608.29517> (2026).
10. Kumar, V., Rathore, D. & Moitra, A. All Verdicts are Not Equal: Rethinking LLM Judge Reliability. Preprint at <https://arxiv.org/abs/2610.12083> (2026).
11. Benjamini, Y. & Hochberg, Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. J. R. Stat. Soc. B 57, 289–300 (1995). <https://doi.org/10.1111/j.2517-6161.1995.tb02031.x>
12. Page, E. S. Continuous inspection schemes. Biometrika 41, 100–115 (1954). <https://doi.org/10.1093/biomet/41.1-2.100>
13. Ramdas, A., Grünwald, P., Vovk, V. & Shafer, G. Game-theoretic statistics and safe anytime-valid inference. Preprint at <https://arxiv.org/abs/2210.01948> (2022).
14. Zhou, S. et al. Learning to Detect Shifts in Unstructured Data Streams via Monitoring-by-Betting. Preprint at <https://arxiv.org/abs/2610.17399> (2026).
15. Shaer, S., Bar, Y., Prinster, D. & Romano, Y. Testing For Distribution Shifts with Conditional Conformal Test Martingales. Preprint at <https://arxiv.org/abs/2602.13848> (2026).
16. Ding, Y., Wei, L., Zhu, X. & Dai, W. Reference-Null Calibrated Thresholds for E-Processes with Applications to Conformal Martingales. Preprint at <https://arxiv.org/abs/2609.32678> (2026).
17. Itkin, I. Quickest Detection of Hallucination Onset: Delay Bounds and Learned CUSUM Statistics. Preprint at <https://arxiv.org/abs/2606.12476> (2026).
18. Dandapanthula, S. & Ramdas, A. Multiple testing in multi-stream sequential change detection. Preprint at <https://arxiv.org/abs/2501.04130> (2025).
19. Google SRE. Alerting on SLOs. The Site Reliability Workbook, <https://sre.google/workbook/alerting-on-slos/> (2018).
20. Gao, X. From Probe Scores to Alarm Policies: Operational Validity of Activation Monitors for Language-Model Agents. Preprint at <https://arxiv.org/abs/2610.04575> (2026).
21. OpenTelemetry. Semantic conventions for GenAI agent and framework spans (status: Development). <https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/gen-ai-agent-spans.md> (2026).
22. AlSayyad, A., Huang, K. Y. & Pal, R. AgentTrace: A Structured Logging Framework for Agent System Observability. Preprint at <https://arxiv.org/abs/2602.10133> (2026).
23. Cemri, M. et al. Why Do Multi-Agent LLM Systems Fail?. Preprint at <https://arxiv.org/abs/2503.13657> (2025).

24. Barke, S. et al. AgentRx: Diagnosing AI Agent Failures from Execution Trajectories. Preprint at <https://arxiv.org/abs/2602.02475> (2026).
25. Rezaeian, A. H., Sun, W., Benajiba, Y. & Roth, D. Prefix Executability: Evaluating Tool-Using Agents Beyond Final Success. Preprint at <https://arxiv.org/abs/2610.40444> (2026).
26. Huang, X. et al. PrefixGuard: Online Failure Warning and Trace-Grounded Diagnosis for LLM Agents. Preprint at <https://arxiv.org/abs/2605.06455> (2026).
27. Barazandeh, B., Swanson, C., Kulkarni, C. & Mungel, N. OnTrack: Real-Time Monitoring and Intervention in LLM Agent Trajectories via Streaming Structure-Aware Optimal Transport. Preprint at <https://arxiv.org/abs/2610.12375> (2026).
28. Srinivasan, S., Li, L., Menzies, T. & Assunção, W. Continue or Restart? The Limits of Early Failure Prediction in Coding Agents. Preprint at <https://arxiv.org/abs/2610.35971> (2026).
29. Iannillo, A. K. et al. Tool-Calling LLM Agents Under Realistic MCP Faults: Scale Does Not Purchase Fault Tolerance. Preprint at <https://arxiv.org/abs/2610.32555> (2026).
30. Zhang, W. et al. RITE: Evaluating End-to-End Resilience to Silent Tool Errors in E-Commerce Agentic Systems. Preprint at <https://arxiv.org/abs/2610.27777> (2026).
31. Khan, S. Token Budgets: An Empirical Catalog of 63 LLM-Agent Budget-Overrun Incidents, with an Affine-Typed Rust Mitigation as a Case Study. Preprint at <https://arxiv.org/abs/2606.04056> (2026).
32. Lu, L., Zhao, Y., Chen, H. & Wang, H. Where Do the Tokens Go? Understanding and Reducing Costs in LLM Agents for Vulnerability Discovery. Preprint at <https://arxiv.org/abs/2610.11602> (2026).
33. Pandey, A., Jain, A., Chen, L. & Maple, C. Local Allow, Global Deny: Why Per-Action Authorization Does Not Compose in LLM Agents. Preprint at <https://arxiv.org/abs/2610.27620> (2026).
34. Wu, C. et al. Safety Does Not Compose: Non-Decaying Loop State for Autonomous LLM Agents. Preprint at <https://arxiv.org/abs/2608.27141> (2026).
35. Korbak, T. et al. Chain of Thought Monitorability: A New and Fragile Opportunity for AI Safety. Preprint at <https://arxiv.org/abs/2507.11473> (2025).
36. Guan, M. Y. et al. Monitoring Monitorability. Preprint at <https://arxiv.org/abs/2512.18311> (2025).
37. Mishra, A., Chatterjee, S. & Maniyar, S. Faithful Latent Harm Monitors Work No Better Than the Models Own Behaviour at a Matched False Alarm Rate. Preprint at <https://arxiv.org/abs/2610.30727> (2026).
38. Baker, B. et al. Monitoring Reasoning Models for Misbehavior and the Risks of Promoting Obfuscation. Preprint at <https://arxiv.org/abs/2503.11926> (2025).
39. Yan, X. et al. Positive-Unlabeled Learning for Agent Safety False Alarm Auditing. Preprint at <https://arxiv.org/abs/2610.02925> (2026).

40. Liao, J. Monitor-Validity Failures: LLM Safety Evidence Paths at Evolving Framework Boundaries. Preprint at <https://arxiv.org/abs/2610.23362> (2026).
 41. Leong, J. W. Online Shift Detection and Conformal Adaptation for Deployed Safety Classifiers. Preprint at <https://arxiv.org/abs/2606.11949> (2026).
-

Copyright Notice: © 2026 Great Unicorn Intelligence Technology (Shenzhen) Co., Ltd. All rights reserved. Any reposting, quotation in commentary, or adaptation must credit the author (Great Unicorn Intelligence Research Team) and cite the source (Great Unicorn Intelligence, <https://www.hxallai.com>).