LaunchDarkly

5 KPIs for Controlling Agents at Runtime

Fewer blind spots, faster containment, and agents that stay in control.

Table of Contents

Everything is moving faster than teams can control.

The LaunchDarkly 2026 AI Control Gap Report found that 94% of engineering leaders say AI has increased the pace of code generation. At the same time, 91% of respondents said their teams have become more cautious about pushing changes to production.

The same report shows that 69% of teams roll back or hotfix at least once per week, and only 15% can deploy daily and keep incidents monthly or less.

94%

of engineering and DevOps leaders say AI has increased the pace of code generation.

91%

say their teams have become more cautious about pushing changes to production.

That was the first wave: AI-generated code outrunning the release process. Now we're entering the second.

Agents are being put to work in production at scale—from customer-facing experiences to backend operations—making decisions, taking action, and evolving over time.

Agents don't fail like traditional software. They drift. Models update, context shifts, and behavior changes without a single line of code changing.

When agents behave, they're incredibly powerful. When they misbehave, they create unacceptable risk. You can't always catch this before production. You have to control it in production.

Why existing dashboards don’t see agents

Knowing is not the same as doing.

Unlike traditional software, where behavior is expected to remain stable after deployment, agents have no equivalent moment of "done." Agents and models are inherently unpredictable and indeterminate.

Behavior can degrade without a code change as model checkpoints update, environments shift, and something that worked reliably can drift without anyone on the team touching it. When something goes wrong, customers can feel it before anyone on the team does, and the standard response—find it, fix it, redeploy—is often too slow for a system that never stops running.

Most teams running agents in production have invested in observability and generally know when something is wrong. But visibility into a problem and the ability to act on it fast enough to matter are different things.

An alert tells you something has degraded, but it doesn't act on it. The controls needed to respond aren't in the same place as the data that surfaced the problem.

What follows are the five KPIs that close that gap: four to measure, one to act.

The five dimensions of AgentControl

Four signals. One lever.

A control dashboard for agents in production has four signals and one lever:

01 Quality (eval score)

Is the agent doing the right thing?

02 Latency

Is it fast enough for the user it's serving?

03 Cost per run

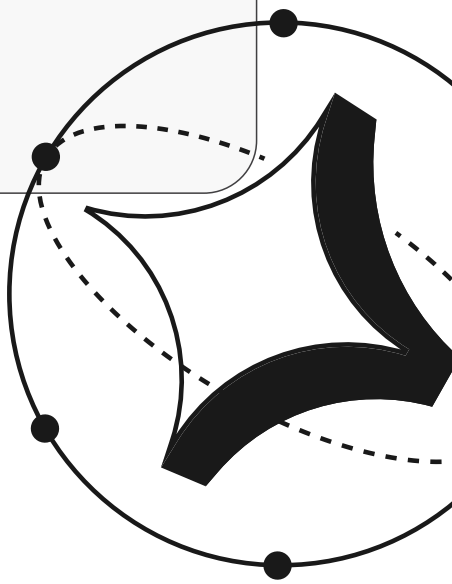
Is it economically viable at this volume?

04 Behavioral drift

Is it still the agent we shipped?

05 Mean time to contain (MTTC)

When something goes wrong, how fast can we stop it?



KPIs 1–4 tell you the state of the system. KPI 5 is what makes the dashboard useful. Without it, you have observability. With it, you have control.

1

KPI

Quality

When teams build AI systems, they're orchestrating prompts, parameters, and models that must work together. Each change—a prompt tweak, a model update, a new routing rule—can subtly shift how an AI system behaves when it meets real users. The goal is rarely to find a single "right" answer, but rather to find the best answer for the moment: accurate enough, grounded in context, and delivered in a tone that feels natural.

You can't always rely on unit tests or static datasets to measure how an AI workflow performs. Quality spans accuracy, tone, safety, and completeness, and continues to shift as inputs, models, and user expectations evolve. Most evaluations still happen offline, long before a system sees live traffic.

Online evals close that gap by measuring quality continuously, not after the fact.

Quality as a control signal

LLM-as-a-Judge uses large language models (LLMs) to automatically score the complete output of an agent. Three out-of-the-box evaluators cover most use cases:

Accuracy	Does the answer match what the user asked?
Relevancy	Is the response on-topic and grounded?
Toxicity	Is the output safe to send?

Each score is generated in real time and stored as a metric. That means quality can flow into the same place teams already manage rollouts, experiments, and guardrails, turning evaluation into a live part of the workflow.

Make it actionable

Because judge scores exist as metrics, they plug directly into existing guardrail logic. The same way you'd set a rollout rule based on latency or cost, you can set one based on quality (for example, routing to a fallback configuration if responses score below a 0.75 accuracy threshold). LLM judge scores are directional signals, not ground truth: They're most valuable when tracked as trends over time instead of hard cutoffs in isolation.

With quality scored continuously, teams get:

Safer rollouts	Accuracy thresholds trigger automatic rollbacks if a new configuration degrades quality.
Confident experimentation	Compare configuration variants using relevance scores, not just engagement metrics.
Early detection	Spot toxicity spikes or accuracy drift within minutes.
Ongoing observability	Sample quality continuously, even after experiments end.

2

KPI

Latency

Latency measures the delay in generating the response to each request. Of all the agent KPIs, it’s the one users feel most.

Latency increases due to larger numbers of user requests or longer input. Monitoring at multiple levels (request, span, model call) helps determine where the bottleneck lies. Measuring latency is essential for determining an agent's responsiveness and usability. In real-time interactions, high latency directly degrades the user experience.

Throughput—the number of requests an agent can handle per unit of time—sits

alongside latency as the capacity question. Throughput tracking helps determine whether to scale up or down required resources; poor throughput indicates an implementation problem rather than a model problem.

Latency as a control lever

Track latency percentiles to detect lag or capacity issues. The p95 is what your power users feel; the p50 is what your average user feels. The gap between them is where your bad days hide.

Latency is also where prompt design pays off. Removing unnecessary instructions and avoiding excessive context reduces input token counts, which tightens both the request and the response.

Once latency is a measured metric in the same control plane as your rollouts, it becomes a threshold you can act on:

- Pause a rollout if p95 latency exceeds the prior version by a set margin.
- Route traffic to a faster model when a latency budget is breached.
- Escalate complex queries to higher-capacity models only when needed.

When performance thresholds are breached or error rates spike, the system can automatically stop a rollout or disable a feature, reducing the time between detection and response.

3

KPI

Cost per run

LLM costs fluctuate based on input size, output length, and system settings such as temperature or context window size. One user might ask a simple question that consumes 20 tokens, while another might ask for a detailed report with a long context that consumes 2,000.

Multiply these differences by thousands of users, and forecasting monthly usage becomes almost impossible. This effect is magnified in chatbots, agents, and conversational assistants, where response length cannot be reliably predicted.

Model behavior itself can change. If an update to a frontier model results in 20% longer answers, your costs will also increase by 20%. Only by monitoring token usage can you catch such shifts early.

20x One test showed 20x higher cost from the premium model, but only a 4% accuracy improvement from a side-by-side test on 105 customer sentiment classifications.

Cost as a tunable

Cost is a controllable KPI—if the model choice is decoupled from the deploy. When you can route traffic between models at runtime, cost stops being a fixed line item and starts being a tunable. You can:

Route by user tier	Premium users get the higher-cost, higher-accuracy model. Free users get the faster, cheaper one.
Route by query complexity	Simple queries go to the smaller model; complex ones are escalated.

Route by region	Support with data residency requirements without rewriting code.
A/B at the model level	Run a faster, more cost-efficient model against a more capable one on real traffic. Let the data decide which one ships.

4

KPI

Behavioral drift

Model drift is when an agent's behavior changes over time in ways that degrade quality. There are several types:

Data drift

Incoming data shifts from what the model was originally trained or evaluated on. New slang, new product names, new topic distributions. The model is unchanged; its performance still declines.

Concept drift

The meaning or relationships in the data change. A new routing rule, API format, or market convention appears, and the "correct" output for the same input evolves.

Embedding drift

Upstream embedding models, tokenization, or vector normalization change, subtly shifting how inputs are represented before the model ever sees them. Unlike prompt drift, this type is rarely visible in your code history: It shows up as degraded output from a system no one touched.

Prompt drift

Small changes made by engineers to prompt text, parameters, context, or retrieval logic shift behavior, tone, or factual accuracy over time.

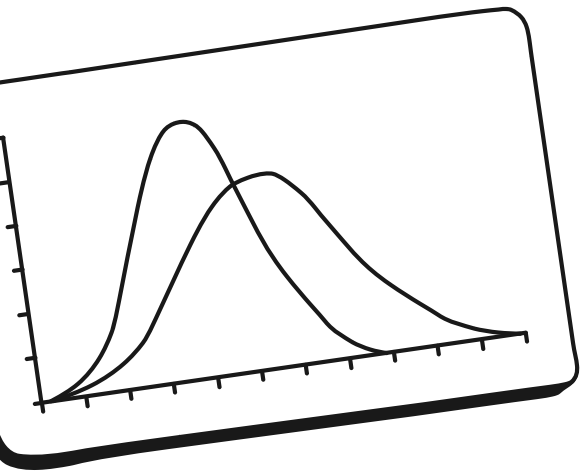
Making drift visible (and reversible)

Drift analysis becomes actionable when it's coupled with an audit trail that records who changed what configuration when, making it easy to correlate a performance degradation with a specific change. This is what the Trends Explorer does:

Visualized insights	Line and bar charts for time to first token, generation time, and other metrics over time.
Filter and compare	Slice by environment to separate staging from production; compare different agents, chatbots, or completion configs side by side.
Cost-spike detection	Catch shifts before they become big surprises.

And drift doesn't have to be retrospective. Built-in approvals can prevent unintended changes from going live in the first place. Whether the change is a prompt variation or a targeting rule update, approvals require sign-off before any update is applied, turning every change into a reviewed, accountable event.

LLMs are extremely sensitive to the phrasing of their input, the length of their context, and the quality of their retrieval.



5

KPI

MTTC

We propose a new metric alongside the ones you already track: Mean time to contain (MTTC), the time from detecting a problem to fully stopping its blast radius.

It's the agent-era version of MTTR, but where MTTR includes everything up through full resolution, MTTC is the narrower, more urgent question: How fast can we make the bleeding stop?

A kill switch is the simplest and most important control mechanism: a feature flag designed specifically for rapidly disabling functionality that may introduce risk. At the code level, a kill switch is usually implemented as a Boolean flag that guards a new or sensitive code path. The application checks the flag before executing the logic.

If an agent starts spamming logs, generating unstable outputs, or drifting from expected behavior, a kill switch lets you shut it down instantly—without reverting unrelated code.

For a kill switch to work reliably, three conditions:

- The risky functionality must be completely wrapped by the flag.
- A stable fallback path must exist and be tested.
- Monitoring and alerts must be tied to the flagged feature.

SDKs maintain a streaming connection and receive updates in seconds, so behavior changes without a redeployment. This matters during incidents, when each passing minute directly increases customer impact.

From manual kill switch to automated containment

Manual kill switches work. But the most effective MTTC isn't measured in minutes-to-toggle—it's measured in milliseconds-to-decision, and a human is not always in the loop.

By connecting feature flags to observability data, teams can configure automated safeguards that disable a feature when predefined limits are exceeded. This turns a kill switch from a manual emergency tool into an automated safety mechanism. This is the role of Adaptive Triggers:

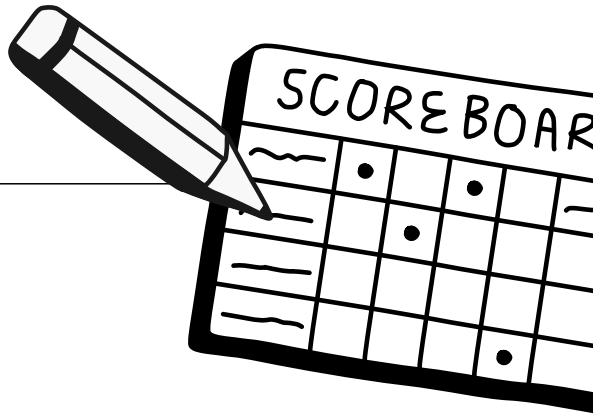
If error rates from a provider breach a threshold, switch to another. If quality scores drop, escalate to a more capable agent. The team decides what to do, and the system handles it automatically, helping contain issues before a bad response reaches the user.

70% reduction in change failure rate, once the team had the ability to cleanly roll back.

Leonid Igolnik
EVP Engineering, Clari

Without this layer, teams either slow down (adding gates and extending reviews) or speed up unsafely (hoping nothing breaks).

Scorecard



Where are you on the dashboard?

Most surveyed teams running agents in production fall into one of three patterns.

Fast but fragile	Ship often and hope nothing breaks	Fix often	Rollback means lost hours, large team needed
Slow but safe	Long, time-consuming review cycles	Rare incidents	Rollback takes hours of structured process
Fast and safe	Deploy often and confidently	Control built into every release	Effective tools can help stop problems instantly

15%

of teams reach the "fast and safe" cohort: deploying daily or more, with incidents kept to monthly or fewer.

2.2x

more likely for LaunchDarkly users to reach this elite cohort

What separates them isn't tooling adoption, as most teams now have feature flags, monitoring, and progressive rollouts. The differentiator is whether they've turned those capabilities into measured KPIs with action tied to thresholds.

How LaunchDarkly maps to these 5 KPIs.

KPI	LaunchDarkly capability
Quality	Online Evals: LLM-as-Judge scoring of every agent output against accuracy, relevancy, and toxicity. Offline Evals for pre-launch validation.
Latency	AI Insights: p95 latency tracked per config variant, with thresholds that pause rollouts or route to faster models.
Cost per run	AI Insights: Token usage and $ per invocation tracked by model/version. Runtime model routing without redeploy.
Behavioral drift	Trends Explorer, versioned configs, and audit trails to attribute drift. Approvals prevent unsanctioned drift before it happens.
Mean time to contain	Guarded Rollouts with automatic rollback. Adaptive Triggers act on the condition before a bad response reaches the user—no manual intervention required.

Models and prompts can be changed in under 200ms, targeted to specific users, and governed from a single place across every team in the organization.

Take control

Set goals. Set guardrails. Ship continuously.

In this new era, the best teams will stay in control—setting goals and guardrails while using agents that ship continuously, learn instantly, and adapt in real time.