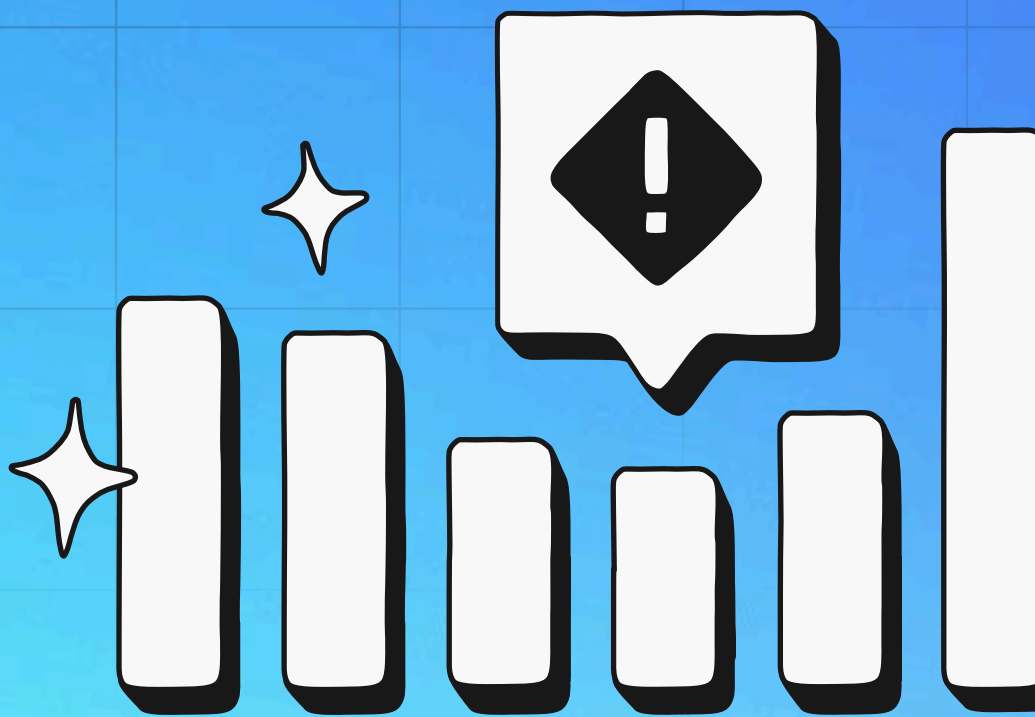




The Practical Guide to AI Observability

Why observability is essential in the AI era—and why it's not enough.

Table of Contents

Why observability in AI applications matters	2
--	---

What great AI observability looks like	4
--	---

Why many teams struggle with AI observability	7
---	---

The limitations of traditional observability in the AI era	10
--	----

What AI-native teams do differently: runtime control	11
--	----

Conclusion	13
------------	----

Introduction

The promise of artificial intelligence has never been greater. From customer service [chatbots that can handle up to 80% of routine inquiries and tasks](#) to predictive models that optimize supply chains in real time, AI features are transforming how businesses operate and compete—and AI agents are increasingly writing the code that powers them. Yet despite massive investments and high expectations, a sobering reality emerges when we examine what happens to AI projects in production.



42%	of surveyed businesses are scrapping AI initiatives.
------------	--

80%	of AI projects fail (estimated).
------------	----------------------------------

Only 20%	of AI projects reach pilot stage.
-----------------	-----------------------------------

Just 5%	of custom AI tools are successfully implemented.
----------------	--

Recent industry research reveals an alarming trend: [42% of surveyed businesses are scrapping most of their AI initiatives](#), up from 17% last year. Even more concerning, it's estimated that [more than 80% of AI projects fail](#)—twice the rate of failure for traditional IT projects that don't involve AI. MIT's latest research shows that while 60% of organizations investigated embedded or task-specific GenAI systems, [only 20% of these organizations reached the pilot stage and just 5% implemented in production](#).

What's driving this crisis? While many factors contribute to AI project failures—from unclear business objectives to poor data quality—the lack of AI observability stands out as a critical root cause. AI is changing not just what teams ship, but how fast they ship it. Teams using AI to generate code are pushing more changes toward production than ever before, making it harder to know which change caused an incident and harder to

move fast enough to contain the damage. And for teams building AI features directly into their products, the challenge runs even deeper.

Unlike traditional software applications where behavior is predictable and deterministic, AI systems operate in a world of probabilities, evolving data patterns, and nondeterministic outputs. Without proper tools, monitoring and analyzing AI systems can account for a significant amount of a team's daily effort. This creates a perfect storm where teams lack visibility into their AI systems precisely when they need it most: in production, where real users, real data, and real business impact converge.

This ebook explores why AI observability is essential—and why it's only the beginning. We'll discuss what great AI observability looks like, why DIY approaches consistently fall short, and what separates the teams that are successfully shipping AI features and AI-generated code in production from the ones still fighting fires.

Why observability in AI applications matters

The transition from AI prototype to production system marks the difference between just an exciting demo and true business transformation. Yet this critical phase is where most AI projects encounter their greatest challenges.

Identifying breaking points

AI systems fail differently than traditional software. While a web application might crash with a clear error message, AI models degrade silently, producing plausible-seeming but incorrect outputs that can persist undetected for weeks or months. When data patterns differ from training data, performance degrades gradually rather than catastrophically. Without proper observability, teams operate blind to these degradation patterns, discovering problems only after significant business impact has occurred.



Avoiding wasted development cycles

Poor observability creates information gaps that lead to misguided optimization efforts. Teams spend weeks investigating the wrong issues—focusing on model accuracy when the real problem is latency, cost, or user experience. Without comprehensive observability, debugging becomes guesswork and A/B testing may lead to incorrect conclusions about what actually drives performance improvements.

Aligning customer experience with key business metrics

AI systems blur the line between technical performance and business outcomes. A content recommendation system might maintain high technical accuracy while actually decreasing user engagement because it's optimizing for the wrong business outcome. Observability helps teams understand the connection between model behavior and business KPIs, enabling correlation of model updates with conversion rates, satisfaction scores, and retention.

Keeping up with the fast-evolving world of AI

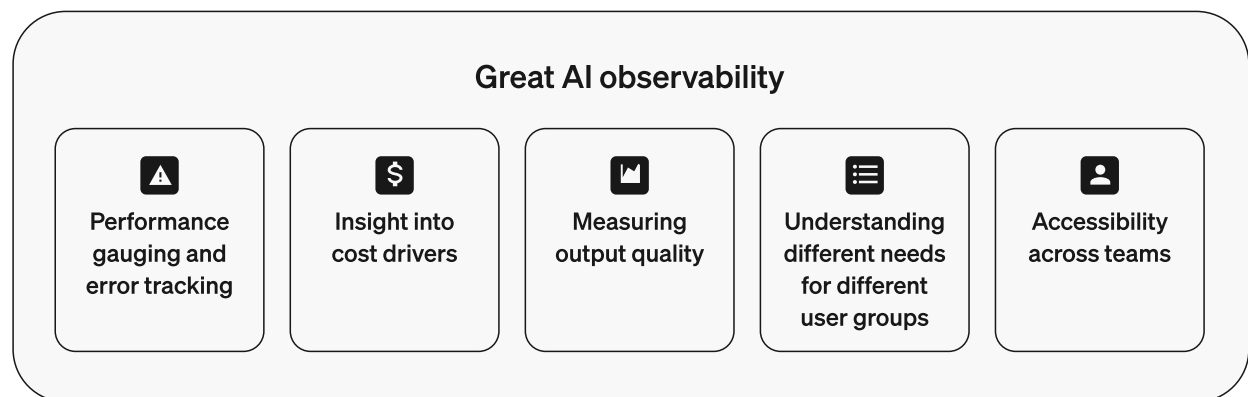
The AI landscape evolves at unprecedented speed. You can't benchmark models once and be done. You need continuous, comparative data in production to keep up. As teams consider new models or different AI providers, they need objective data to evaluate trade-offs between cost, performance, and reliability. Organizations with superior AI observability can iterate faster, deploy more confidently, and optimize more effectively—creating sustainable competitive advantages in AI-driven markets.



What great AI observability looks like

Effective AI observability isn't just about collecting more data. It's about creating a comprehensive visibility framework that enables proactive management, rapid iteration, and confident decision-making.

Great AI observability has five essential characteristics that distinguish it from traditional monitoring approaches.



Performance gauging and error tracking

Great AI observability provides comprehensive performance monitoring that accounts for the probabilistic nature of AI systems. This means tracking multidimensional performance metrics simultaneously—accuracy, latency, throughput, token usage, cost, and error rates—while enabling real-time configuration changes to address issues instantly.

For agent-based systems, this monitoring has to reach a level of detail that traditional APM was never built for. A single user request might fan out into a multi-step workflow: The agent reasons, calls a tool, interprets the result, and calls a model again before responding. Great AI observability captures that full execution as a trace, with each model call and tool invocation recorded as a span carrying the prompt, the parameters, the response, and the latency, token usage, and cost of that specific step. It also records which model and provider version produced each completion, which helps account for

third-party model updates the team isn't aware of. When something goes wrong, this level of visibility is the difference between knowing the agent failed and knowing which step failed, what it was asked, and how it answered.

Unlike traditional monitoring that waits for system failures, effective AI observability detects gradual performance degradation and model drift before they impact users. The best systems provide automated anomaly detection that understands normal variance in AI behavior, preventing both false alarms and missed issues.

Insight into cost drivers

AI systems introduce unique cost dynamics that require specialized tracking beyond infrastructure monitoring. Great AI observability provides granular visibility into token consumption per request, user, and feature, enabling teams to optimize both performance and economics simultaneously.

The scaling challenge becomes especially acute as teams move from pilot to production. What starts as manageable costs during development can quickly spiral when you're serving thousands of users across multiple features. Teams need predictive cost modeling that can forecast expenses based on usage patterns and user behavior—before the bill arrives.

The most effective platforms correlate cost metrics with business outcomes, helping teams understand the ROI of different models and configurations. This includes predictive cost modeling based on usage patterns and the ability to make runtime adjustments that immediately impact cost efficiency.

Q Great AI observability provides **granular visibility into token consumption per request, user, and feature**, enabling teams to optimize both performance and economics simultaneously.

Measuring output quality

Unlike traditional software, where the "correct" behavior is deterministic and testable, AI systems produce outputs that can be technically functional yet still wrong, unhelpful, or off-brand. For instance, a support agent might return a low-latency, error-free response that confidently states the wrong refund policy. Quality is the dimension that's unique to AI, and it's foundational to what agent builders and AI engineers care about most.

Great AI observability depends on a clear definition of what “good” looks like. After that’s in place, it must be turned into a trackable metric. Techniques such as LLM-as-a-judge evaluation, human review sampling, and task-specific scoring rubrics let teams quantify dimensions like hallucination rate, relevance, tone compliance, or task completion—and track them over time and across model or prompt versions.

This matters most when it comes to acting on signals. Latency spikes and error rates are easy to catch and fix, but quality regressions can slip by unnoticed. A model can look perfectly healthy on every infrastructure metric while quietly getting worse at its actual job, which makes quality-related issues especially problematic.

Understanding different needs for different user groups

AI systems often serve diverse user populations with varying needs and behaviors. Great observability enables segmented performance tracking across user groups, geographic regions, device types, and subscription tiers—helping to ensure equitable experiences while enabling personalized optimization.

This segmentation capability should support sophisticated targeting rules that allow teams to serve different AI configurations to different user groups based on context, then measure performance across each segment to optimize experiences without requiring code changes.

Accessibility across teams

The most powerful AI observability democratizes access to insights across technical and nontechnical stakeholders. Rather than requiring engineering expertise for every



configuration change, great platforms enable product managers, data scientists, and other team members to iterate on AI systems directly.

This accessibility includes role-based dashboards, collaborative investigation tools, and the ability to make prompt adjustments, model switches, and parameter changes through intuitive interfaces—enabling rapid iteration without deployment bottlenecks or cross-team dependencies.



Key takeaway

Great observability lets you move from guesswork to guided iteration at scale—and in production.

Instead of hoping your AI changes work, you know they do. You get the data and controls you need to optimize confidently.

Why many teams struggle with AI observability

Many organizations initially attempt to build AI monitoring solutions using existing tools, but this DIY approach consistently leads to incomplete solutions that fail to deliver the visibility needed for AI to be successful in production.

Scattered internal tools

Teams typically end up with fragmented monitoring across multiple systems: infrastructure monitoring for server metrics, application performance monitoring for API latency, custom scripts for model accuracy, spreadsheets for cost analysis, and separate dashboards for different AI providers. This fragmentation makes it nearly impossible to correlate events and understand root causes when issues arise.



You end up writing token-counting scripts at 11 PM, watching five different dashboards after every release, and hoping your makeshift alert logic actually catches something before your users do. When something breaks, you're playing detective across multiple tools, trying to piece together whether the spike in latency is related to the new prompt you deployed or the model switch you made last week.

Limited and inconsistent metrics

Internal solutions focus on obvious technical metrics while missing critical business and operational indicators. Teams often define metrics differently across projects, making it extremely difficult to compare performance. More importantly, they often lack the specialized metrics needed for AI systems (token usage tracking, model drift detection, and cost optimization insights) that traditional monitoring tools weren't designed to handle.



Nothing feels worse than confidently shipping what you think is an improvement, only to discover weeks later that your change is actually performing worse—because you didn't have the right metrics to catch it. Every team defines "accuracy" differently, every project tracks costs in spreadsheets, and when stakeholders ask, "Which model is actually better?" you're left shrugging and saying, "It depends on how you measure it."

Risk-laden experimentation

AI systems require extensive experimentation, but internal solutions tend to lack the safety nets needed for confident testing. Teams struggle with statistical significance, proper A/B test design, and automated rollbacks when experiments degrade performance. The complexity of tracking multiple variables—model versions, prompts, parameters—across experiments makes it difficult to understand what actually causes performance changes.



Every experiment feels like a gamble. You're never quite sure if your sample size is big enough, if the improvement you're seeing is real or just noise, and you definitely don't sleep well knowing that your "experiment" is running on real users without proper guardrails. When things go sideways, you're frantically reverting commits and praying the rollback works faster than angry user feedback rolls in.

Slow and arduous process of code changes and redeployment

The most significant limitation is the tight coupling between monitoring capabilities and code deployment cycles. Without specific tooling, change requires a full deployment cycle, simple threshold adjustments need code reviews, and teams generally struggle to iterate on observability without affecting production systems. This friction slows the rapid iteration that AI optimization requires.

For teams also shipping AI-generated code, that same deploy friction compounds the problem: The faster AI writes code, the more changes are heading toward production—and the harder it becomes to trace which one caused an issue.



Want to adjust a prompt? Deploy. Want to try a different model? Deploy. Want to change an alert threshold? Deploy, deploy, deploy. Meanwhile, your product manager is asking why the "simple prompt tweak" is taking three sprints, and you're explaining for the hundredth time that changing AI configurations isn't like updating a config file—it's a full engineering effort that touches production systems.



The limitations of traditional observability in the AI era

Even teams that have invested seriously in AI observability are discovering that visibility is necessary, but not sufficient. Traditional observability tools can only tell you what happened. They don't tell you which change caused the problem, and they don't proactively act on what they see.

This creates a gap between the moment you know something is wrong and the moment you're able to fix it. An engineer gets paged, investigates the dashboards, identifies the culprit, and deploys a fix. That model is straining under the pressure of AI-driven development, and it breaks down entirely in AI production systems.

The time between "We know something is wrong" and "We've fixed it" is where real damage happens.

AI agents are nondeterministic by design

The nondeterminism of agentic systems isn't a flaw; it's the whole point. AI agents reason and adapt dynamically, which means their behavior can't be reliably predicted—even by the teams that built them. Additionally, the models powering these agents are constantly and quietly being updated by providers, and users interacting with them are endlessly variable in how they ask questions and what they expect.

Even after teams know there's a problem with a production agent, the path to remediation is almost never straightforward. Evals live in one tool, behavior control lives elsewhere, and implementing a tested fix still requires a redeploy. This is a critical problem because a misbehaving agent doesn't stop running while teams figure out how to handle it. Customers may continue to be exposed to bad responses for as long as the deployment cycle takes.

AI has changed how software is built

Agents in production aren't the only source of this pressure. Teams that are using AI to write code are finding that the increased velocity comes with a corresponding increase in production incidents.

According to the [LaunchDarkly Control Gap Report](#), 94% of engineering teams reported that AI has accelerated their code velocity, but nearly as many (91%) say they are more cautious about pushing AI-written code live. The survey data also underscores the cost of the problem: 24% of teams report having to roll back or hotfix production issues daily, and 14% get caught in this cycle multiple times a day.

What AI-native teams do differently: Runtime control

There's a better way to operate, and it's called runtime control. Runtime control doesn't replace observability; it extends it. While observability tools provide visibility into what's happening in production, they're not designed to intervene. Runtime control bridges that gap, giving teams the ability to detect and respond to concerning, change-based signals before users can feel the impact—without waiting for a redeploy.

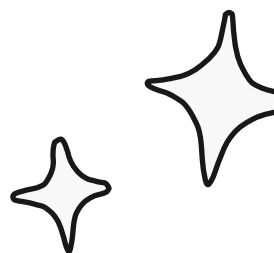
With this approach, the incident that used to take hours to diagnose and resolve can be handled in seconds. Engineers wake up to "Something happened, and it's been handled" instead of a 2 AM page.

What runtime control looks like in practice

At its core, runtime control means:

Acting at the change level, not the application level.

When behavior degrades, teams can isolate the specific change that caused it, contain the blast radius, and act—without triaging an entire application.



**Closing the loop
between
observation and
remediation.**

Rather than requiring a human to investigate every alert and deploy a fix, the system can detect a threshold breach and respond automatically. This often involves pausing a rollout, restoring a previous configuration, or rerouting traffic.

**Decoupling
deployment from
exposure.**

Code ships continuously, but what users actually see is controlled separately—progressively, precisely, and reversibly. The volume of AI-generated change stops being the variable that determines risk.

This is not just about moving faster. It's about making speed safer.

Checklist: Is your team ready?

As you evaluate your current approach to AI in production, consider whether your team can:

- Control and govern AI agent behavior in production—not just monitor it.
- Release AI-generated changes progressively, limiting exposure while observing real-world impact.
- Halt or roll back within seconds when performance falls outside acceptable thresholds, without a redeploy.
- Trace an incident to the specific change that caused it, automatically.
- Act on concerning health and performance signals before users feel the impact.

If the answer to most of these is "no" or "not yet," you're not alone—but the gap between where you are and where AI-native teams operate is worth closing.

The teams who are succeeding in the AI era aren't just better at observability. They've changed what they can do when something goes wrong.



Conclusion

The statistics are sobering: 42% of surveyed businesses are scrapping most of their AI initiatives, and it's estimated that 80% of AI projects fail—with just 20% of organizations that evaluated tools like ChatGPT or Copilot reaching the pilot stage and a mere 5% being implemented in production.

The difference between success and failure increasingly comes down to two capabilities working in tandem: comprehensive AI observability and the ability to act on what that observability reveals.

As we've explored throughout this ebook, AI systems introduce unique challenges that traditional monitoring approaches cannot address. But equally important is what happens after an issue is surfaced. In a world where AI-written code ships faster than any review process can keep up with, and where AI agents behave in ways that can't be fully predicted, the teams that win are those who have closed the gap between detection and remediation.



Key takeaways

The production crisis is real—and it's accelerating.

AI-generated code and nondeterministic agents are introducing a new category of production risk that traditional observability wasn't designed to handle.

Visibility is necessary, but not sufficient.

The teams outperforming their peers aren't just better at monitoring. They've built systems that can act on what they see — automatically, at the change level, in real time.

The competitive advantage is compounding.

While many organizations struggle with basic AI deployment challenges, those who invest in the right infrastructure now will be best positioned to capitalize on every new model, framework, and agentic capability that emerges.

Getting it right early is much easier than fixing a mess later.



The choice facing organizations today isn't whether to invest in AI—that decision has largely been made. **The critical choice is whether to invest in the infrastructure that makes AI investments successful.**

Observability is a prerequisite. But the teams compounding their advantage in AI production have figured out that it's the floor, not the ceiling. The question isn't whether AI observability matters—it does, deeply—it's whether your organization will build the full foundation before your competitors do.

Where to start

You don't need to overhaul your stack or spin up a new team to make progress. Here are a few good places to begin:

Audit what you can (and can't) see today.

Can you trace which prompts, models, or agents are powering live experiences? Can you measure how they're performing—or when they fail? If not, start mapping where your blind spots are.

Pick one high-impact config to observe.

Choose a prompt, model, or system message that's critical to the user experience—something with risk, complexity, or customer exposure. This becomes your observability pilot.

Define what “bad” looks like before it happens.

Is it latency above 2 seconds? A spike in hallucinations? A drop in feedback scores?

Ask whether you can act, not just observe.

For every alert your system can fire, ask: What happens next? If the answer is always "A human investigates and deploys a fix," that's the gap worth closing next.

The goal is to shift from reactive to proactive—from "That broke and we didn't know why" to "We saw it coming, and we handled it automatically before users felt it."



Ready to close the gap between detection and action?

Runtime control →

Try it free →