Weave Intelligence

MARKET GUIDE

# Observability Trends in Platform Engineering

# Market Guide for Observability Trends in Platform Engineering

## Table of contents


| | |
|---|---|
| 1 - Executive overview | 4 |
| Key findings | 5 |
| Recommendations for platform leaders | 6 |
| 2 - Introduction | 7 |
| Scope and problem statement | 7 |
| Why the traditional approach fails | 10 |
| The convergence of FinOps and observability | 11 |
| Agentic AI as an operational forcing function | 12 |
| OTel standardization as the foundation for AI readiness | 12 |
| 3 - Market definition: The Observability Plane in platform engineering | 14 |
| Definition and position in the stack | 14 |
| Must-have capabilities | 17 |
| Standard capabilities | 18 |
| Optional and advanced capabilities | 20 |
| 4 - Market direction and trends | 21 |
| The shift-down philosophy: Instrumentation as platform infrastructure | 21 |
| Agentic AI: From dashboard consumer to autonomous operator | 24 |
| FinOps convergence: Cost as a first-class observability signal | 26 |
| OTel standardization as AI readiness | 27 |
| 5 - Market analysis: Adoption dynamics | 28 |
| Benefits for the platform team and the developer | 29 |
| Challenges and friction points | 30 |
| Vendor landscape categorization | 32 |

| | |
|---|---|
| 6 - Representative vendors | 34 |
| Vendor selection criteria | 35 |
| Representative vendor landscape | 35 |
| OTel-native platforms | 35 |
| Pipeline and context specialists | 36 |
| Full-stack incumbents | 37 |
| AI/LLM native | 38 |
| Runtime intelligence | 38 |
| Structural dynamics shaping the landscape | 39 |
| 7 - Strategic Recommendations for Platform Teams | 40 |
| Implement pipeline-layer cost controls before agentic workloads reach production scale | 41 |
| Enforce OTel semantic conventions at the collector layer as a platform policy | 42 |
| Give developers direct access to live production observability — stop proxying through tickets | 43 |
| Treat telemetry as a data engineering problem, not just an ops tool | 44 |
| Extend the Observability Plane to treat LLM and agentic workloads as a first-class concern | 45 |
| Appendix | 46 |
| Authors | 46 |
| About the survey | 47 |
| Disclaimer | 48 |
| Imprint | 49 |

# 1

# Executive overview

Platform engineering teams now own a shared infrastructure cost problem that most organizations have not yet formally named. Telemetry data volumes are growing by 28–40% annually, while IT budgets remain flat. Agentic AI workloads are compounding that gap faster than incremental optimization can close it. Each LLM query generates approximately 2–5x more telemetry than a standard application log event, and AI SRE tools pursuing parallel diagnostic hypotheses can generate 10–100x the query load against observability endpoints that existing architectures were never designed to handle. For platform teams that have begun deploying agentic tooling even in test environments, this is not a future risk.

# Key findings

## The Observability Plane is the data substrate for autonomous operations

The shift from human-facing dashboards to agent-driven investigation is already underway. Platform teams that architect their Observability Plane for agentic consumption now will absorb the next wave of operational complexity. Teams that do not will face a compounding cost and reliability problem that becomes structurally harder to unwind as agentic workloads scale.

## OTel standardization is an AI readiness strategy, not a housekeeping task

LLMs are trained on OpenTelemetry documentation and semantic conventions. OTel-structured telemetry is inherently more interpretable by AI agents, requiring less prompt engineering and producing more reliable root cause identification. Every investment in OTel posture simultaneously improves observability quality for human operators and agentic investigation effectiveness.

## FinOps and observability are the same operational discipline

The telemetry pipeline that routes logs and metrics also determines what data is ingested, stored, and billed. Separating these concerns organizationally results in the reactive cost management that platform teams are trying to avoid.

## Incumbent platforms face a structural constraint on data reduction

A platform whose revenue model depends on ingestion volume cannot build aggressive data reduction capabilities without cannibalizing its own revenue. Engineering investment does not close this gap - it is a business model conflict that creates a durable competitive moat for OTel-native and pipeline-specialist vendors.

# Recommendations for platform leaders

- Implement pipeline-layer cost controls such as tail sampling, null field removal and log metricization before agentic workloads reach production scale, not as a parallel workstream.
- Enforce OTel semantic conventions at the collector layer as a platform policy, not a developer responsibility.
- Give developers direct, scoped access to live production observability. Stop proxying investigation through tickets and SRE availability.
- Treat telemetry as a data asset with defined ownership, tiered storage policies, and a versioned schema. Do not treat it as an operational byproduct managed by vendor defaults.
- Extend the Observability Plane to cover LLM and agentic workloads as a first-class concern, with session-level evaluation and continuous improvement loops in place before production deployment, not as a retrofit.

# 2

# Introduction

## Scope and problem statement

Platform engineering teams are being asked to do something structurally contradictory. Application teams want full telemetry fidelity for debugging. Finance teams want cost reduction. SRE teams want real-time signal quality for incident response. When the underlying data architecture treats all telemetry as equally valuable, satisfying any one of these demands degrades the others.

Two forces have converged to make this tension existential rather than merely inconvenient.

The first is economic. Telemetry data volumes are growing at 28 - 40% annually while IT budgets remain flat. That gap has been widening for years, but it has now reached a point where standard responses such as tiering data, tuning retention policies, and negotiating vendor contracts cannot close it. Platform teams are making triage decisions about which signals to keep and which to discard, and cost rather than operational value increasingly drives those decisions.

The second force is architectural. Agentic AI workloads do not consume observability data the way human operators do. A human investigates incidents serially, one hypothesis at a time. An AI SRE tool pursues 10-12 simultaneous hypotheses, each hitting observability endpoints independently.
The query load this generates is 10-100x what existing tools were designed to handle. For platform teams that have begun deploying agentic tooling even in test environments, this amplification effect is already visible in telemetry volumes and pipeline latency.

When platform teams cannot provide fast, reliable, cost-effective telemetry, the organizational consequence is predictable and documented.

Developers route around them. They stand up shadow observability environments, instrument services inconsistently, and produce the fragmented signal quality that makes automated remediation impossible. The platform team's mandate, to provide observability as a shared, self-service capability, collapses into a reactive support function.

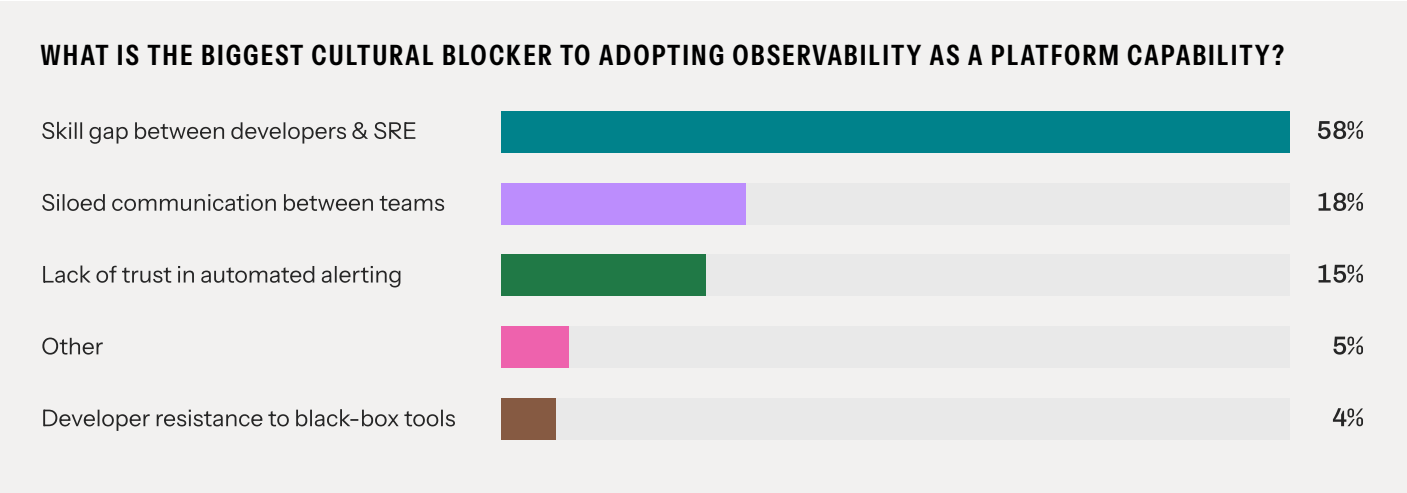


**Figure 1**: Cultural blockers to adopting observability as a platform capability

Among 105 surveyed platform engineers, SREs, and operations leaders, 57% said their observability setup produces too much noise or does not help them identify root cause. Another 58% cited instrumentation skill gaps as the primary cultural blocker, while 39% said instrumentation adds more than 24 hours to their feature-to-production cycle.

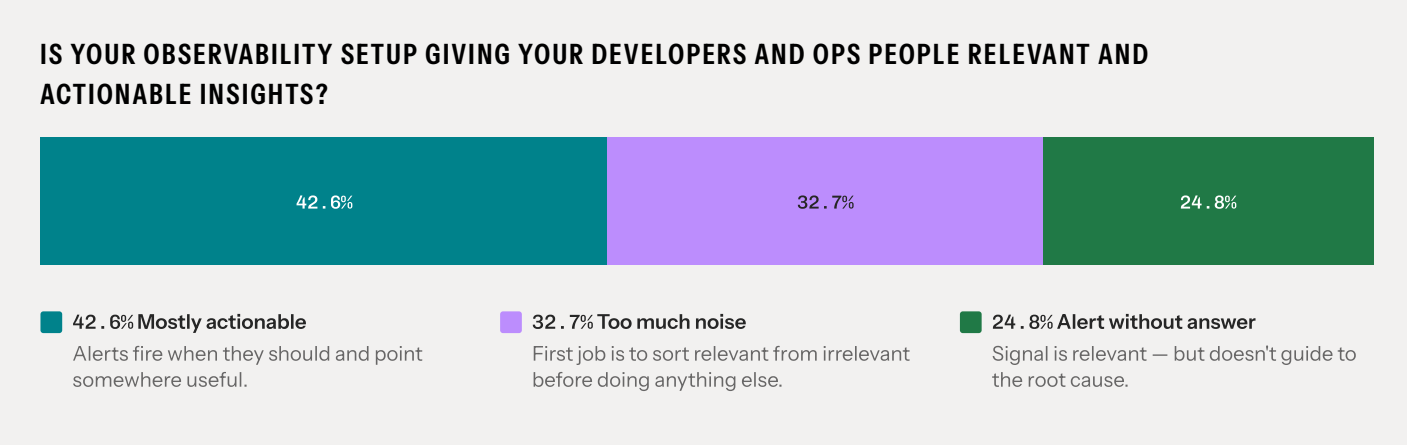


**Figure 2**: Quality of insights provided by the observability setup

## Four specific challenges structure the problem:

- **Inconsistent data** across applications and environments, where different teams use different field names, different cardinality conventions, and different signal types for equivalent concepts.
- **Lack of telemetry practice**, where different teams monitor different, incomplete aspects of their services and cannot agree on a shared view of system state during incidents.
- **Lack of impact analysis**, where telemetry signals cannot be tied back to user-facing impact, making it impossible to prioritize remediation by business consequence.
- **Tooling drift**, where custom dashboards and exporters no longer match the reality of the systems they were built to observe.

Each challenge maps to a specific capability gap. Inconsistent data maps to the absence of enforced semantic conventions. Lack of practice maps to the absence of golden paths for instrumentation. Lack of impact analysis maps to the absence of SLO-driven signal prioritization. Tooling drift maps to the absence of dashboards-as-code lifecycle management.

The architectural resolution to all four is the same: stop pushing observability responsibilities onto individual development teams and start embedding instrumentation, sampling, and context enrichment natively into the platform layer. Practitioners call this shift-down observability. The platform absorbs the instrumentation burden; developers receive telemetry as a default capability rather than a responsibility.

This guide covers the structural position of the Observability Plane within the IDP stack, the capability taxonomy that distinguishes mature implementations from immature ones, the vendor landscape organized by architectural approach rather than market share, and the strategic recommendations that follow from both.

# Why the traditional approach fails

The traditional observability model was built for a bounded number of services, a human operator reviewing dashboards, and a telemetry pipeline sized for the data volumes of the previous year. Each of those assumptions has broken down.

At engineering organizations operating at scale, the number of services and deployments is increasing faster than any team-by-team instrumentation approach can keep pace with, and agentic coding tools are accelerating that rate further. Organizations that have adopted agentic coding heavily are already observing measurable increases in incident rates, a direct consequence of shipping more code faster without a corresponding increase in observability coverage. The platform's role has shifted: it is no longer sufficient to provide observability tooling and expect development teams to use it correctly. The platform must absorb the instrumentation burden itself.

The economic model of traditional observability platforms compounds this problem. Platforms whose revenue depends on data ingestion volume face a structural challenge: they cannot build aggressive data reduction capabilities without cannibalizing their own revenue. This is the innovator's dilemma applied to the observability market. A major European retail player implemented edge-side tail sampling within its own environment - before any data crossed a network boundary - reducing spans per second by more than 20x and saving $2.6 million per year in egress costs alone.

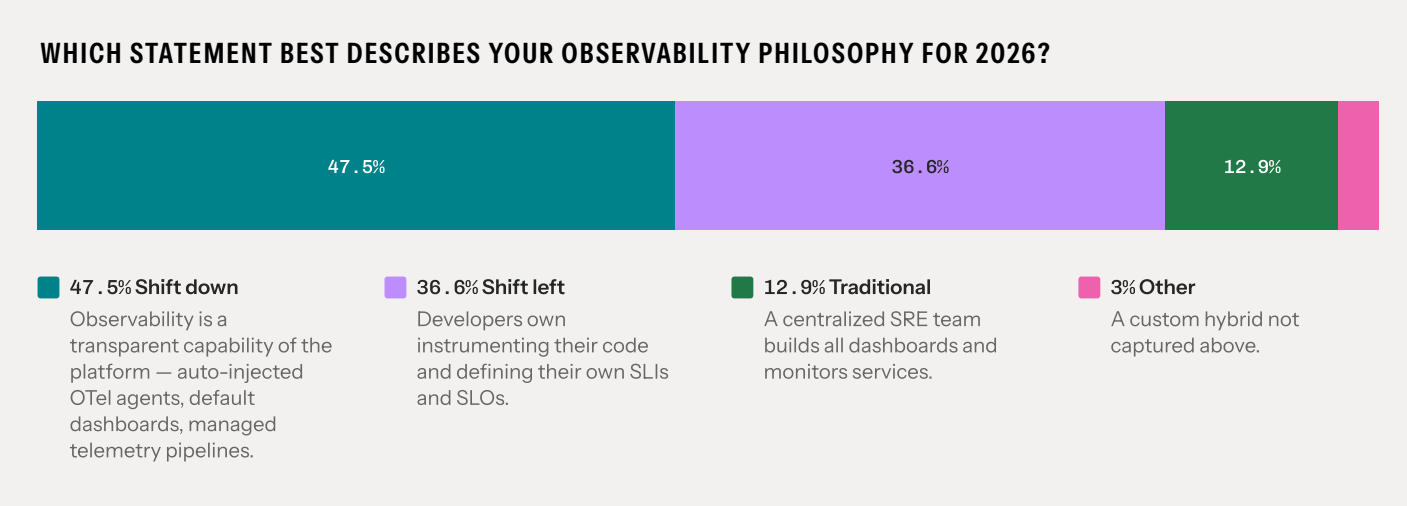


**Figure 3**: Observability approaches

Survey data reveals both aspirations and existing gaps: 47.5% describe their 2026 strategy as shift down, focusing on platform auto-injection of OTel, default dashboards, and managed pipelines. Meanwhile, 12.9% remain in reactive mode, concentrating on maintaining current monitoring systems.

The paradigm shift underway mirrors an earlier one. Just as infrastructure-as-code moved infrastructure management out of proprietary vendor UIs and into version-controlled, auditable workflows, agentic observability is moving incident investigation out of proprietary dashboards and into embedded, automated workflows. The Observability Plane is evolving from a human-facing visualization layer into the data substrate for autonomous operations.

# The convergence of FinOps and observability

Cost visibility and telemetry visibility are converging into a single operational discipline.

The cardinality explosion mechanism illustrates why this convergence is operationally necessary. A single developer adding a high-cardinality tag, such as a user ID, to a metric can trigger a cost event that is invisible until the billing cycle. The number of unique time series created by that single instrumentation change can multiply by orders of magnitude depending on the cardinality of the tag value. Platform teams that lack real-time cardinality visibility cannot prevent these events; they can only respond after the cost has been incurred. Pipeline-layer controls - null field removal, log metricization, tail sampling, edge-side processing - address this structurally rather than reactively.

# Agentic AI as an operational forcing function

The Observability Plane was designed for human operators. Agentic AI tools are a fundamentally different consumer of observability data, and the architectural implications of that difference are not yet fully reflected in most platform teams' infrastructure decisions.

Each LLM query generates approximately 2 - 5x more telemetry than a standard application log event. AI SRE tools that pursue multiple simultaneous diagnostic hypotheses multiply this effect further, generating query loads that existing observability endpoints were never designed to handle. The cost event this creates at production scale is an order of magnitude larger than what platform teams are currently observing in test environments.

The market is converging on a two-way door maturity model for agentic remediation: AI-assisted investigation first, then automated execution of reversible actions (feature flag toggles, pod restarts, rollbacks) with human approval gates at each stage before expanding to more complex infrastructure changes. Human approval gates are not a limitation on this model; they are the trust-building mechanism that makes progressive automation organizationally viable. The platform teams that will benefit most from agentic remediation are those that have already established the observability data quality and pipeline latency required to support it.

# OTel standardization as the foundation for AI readiness

OpenTelemetry standardization has moved beyond avoiding vendor lock-in. It is now an AI readiness strategy with compounding returns.

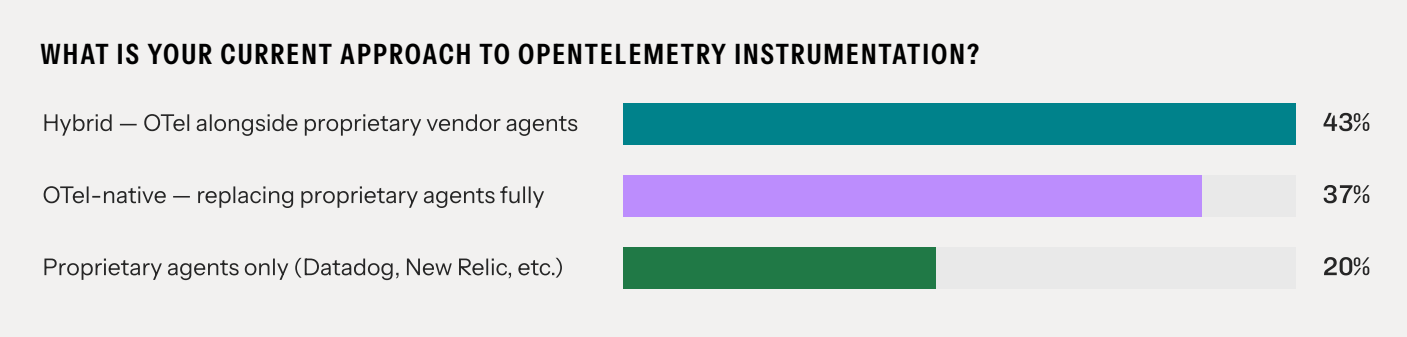


**Figure 4**: Approaches to OpenTelemetry instrumentation

> 37% of survey respondents have fully replaced proprietary agents with OTel, 43% are in a hybrid model, and 20% still rely exclusively on proprietary instrumentation which is a concrete, quantifiable segment accumulating AI-readiness debt that compounds as agentic tooling becomes more prevalent.

LLMs are trained on public OTel documentation, semantic conventions, and sample applications. OTel-structured telemetry is inherently more interpretable by AI agents. When semantic conventions are enforced, an agent investigating an incident knows exactly where to find `service.name` , `cloud.region` , and `http.response.status_code` . Without enforced conventions, the agent lacks the structural context to navigate telemetry reliably, and the quality of agentic investigation degrades proportionally.

Organizations that allow teams to instrument arbitrarily - using custom field names, inconsistent label schemas, and proprietary agent formats - are accumulating AI readiness debt that compounds as agentic tooling becomes more prevalent. Every investment in OTel posture simultaneously improves observability quality for human operators and agentic investigation effectiveness. That compounding return makes the standardization argument straightforward to justify to engineering leadership.

FinOps convergence, agentic AI adoption, and OTel standardization define the structural context within which the Observability Plane must now be evaluated. The market definition that follows establishes what the Observability Plane is, where it sits within the IDP stack, and which capabilities distinguish implementations that are positioned for this operational reality from those that are not.

# 3

# Market definition: The Observability Plane in platform engineering

## Definition and position in the stack

The Observability Plane is the self-service capability layer of an Internal Developer Platform responsible for collecting, processing, routing, and surfacing telemetry signals, e.g. – metrics, logs, traces, and increasingly AI/ML-specific signals, – in a form that enables both human operators and autonomous agents to understand and act on system state.

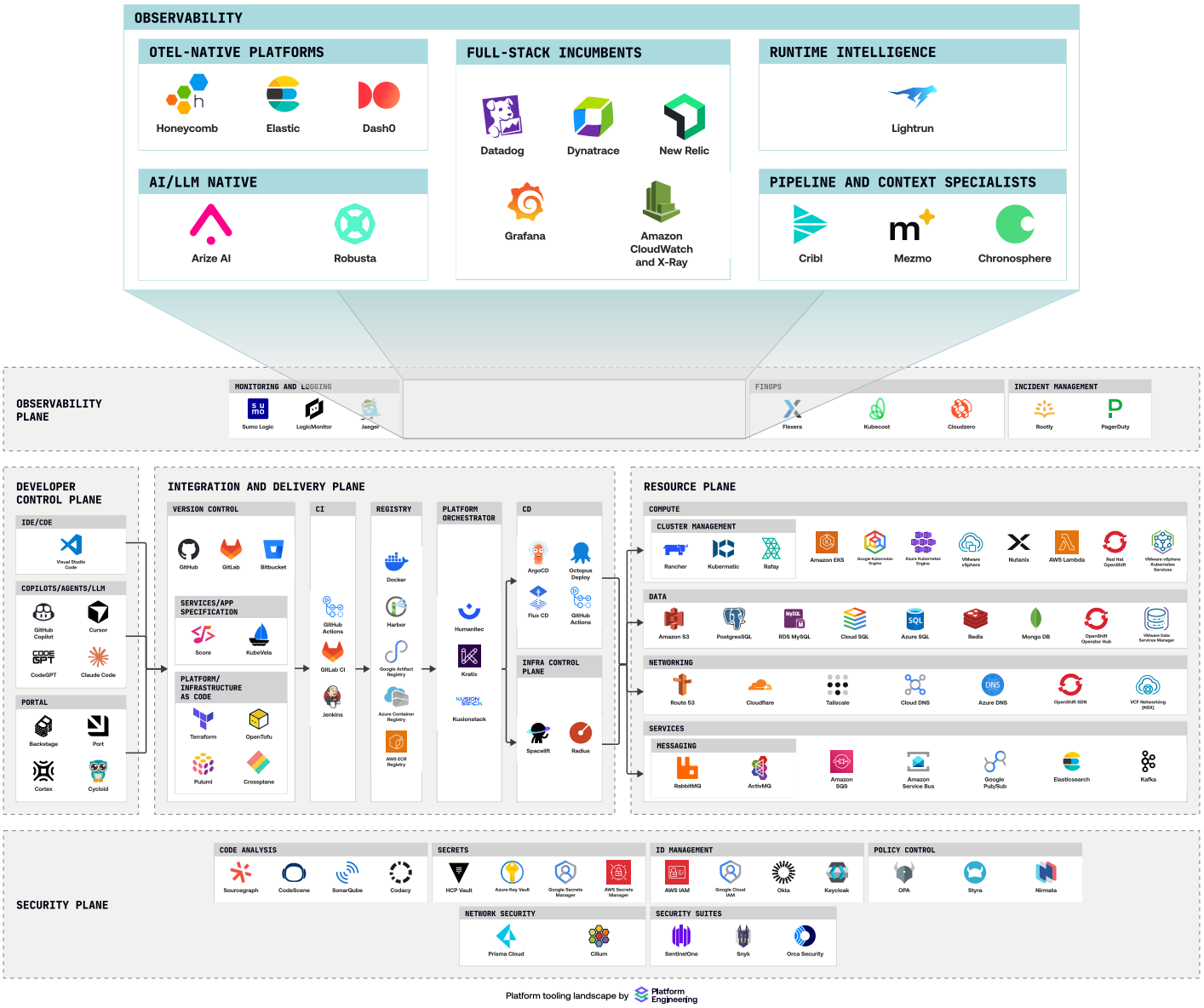


**Figure 5**: Observability plane of the Internal Developer Platform

This definition distinguishes the Observability Plane from monitoring. Monitoring is the practice of collecting telemetry. Observability is the property of a system that allows you to understand why it is in a given state by asking questions from the outside. The Observability Plane includes the tools, paths, and code that enable this property at scale across a multi-team, multi-service engineering organization. If monitoring tells you something is wrong, observability tells you why.

Within the IDP stack, the Observability Plane sits above the Resource Plane - compute, networking, storage, i.e., the sources of raw telemetry - and the Integration and Delivery Plane, which includes CI/CD pipelines and telemetry pipeline processors. It feeds upward into the Developer Control Plane, through which developers and agents consume observability signals, and that handles scorecarding, operational maturity tracking, and engineering intelligence.

The telemetry pipeline layer (including OTel Collectors, edge processors, context engineering tools) sits at the boundary between the Integration and Delivery Plane and the Observability Plane. This boundary is where the most significant architectural innovation is currently occurring.

Platform teams bear two distinct observability responsibilities. First, observing the platform itself: Kubernetes control planes, CI/CD pipelines, shared databases, load balancers. Second, enabling application teams to observe their own services through self-service instrumentation, pre-configured dashboards, and standardized telemetry conventions. A mature Observability Plane serves both mandates without requiring application teams to understand the underlying infrastructure.

Production IDP reference architectures across GCP, Azure, and AWS and the Platform Engineering Tooling Landscape show a consistent four-component structure for the Observability Plane:

1. **Monitoring and Logging** - collection and visualization of metrics, logs, and system health signals.
2. **Observability** - exploratory analysis, distributed tracing, and high-cardinality querying.
3. **FinOps** - cost visibility, chargeback, and telemetry spend optimization.
4. **Incident Management** - alerting, escalation, and remediation workflow coordination

The consistent inclusion of FinOps across all three cloud provider reference architectures reflects a practitioner reality: cost control and signal quality are the same problem viewed from different angles, and separating them organizationally produces the reactive cost management that platform teams are trying to escape.

# Must-have capabilities

These are capabilities without which a platform team cannot fulfill the dual mandate at scale.

## OpenTelemetry-native ingestion and semantic convention enforcement

OTel is not merely a data format - it is the vendor-neutral contract that makes telemetry portable, composable, and AI-ready. Platforms that enforce OTel semantic conventions produce telemetry that is queryable across teams, compatible with any downstream tool, and structured in a way that LLMs are trained to understand. Because LLMs are trained on public OTel documentation, semantic conventions, and sample applications, OTel-structured telemetry is inherently more interpretable by AI agents than proprietary or inconsistently structured data. Platforms that allow teams to instrument arbitrarily produce telemetry that is voluminous but contextually incoherent. Semantic convention enforcement is the difference between data and signal.

## Real-time telemetry pipeline with policy enforcement

The OTel Collector, - or an equivalent pipeline processor deployed at the platform layer, - must function as a policy engine, not merely a data router. From this central control point, platform engineers must be able to reduce ingestion costs by sampling high-volume traces without touching application code, enforce compliance by redacting sensitive fields (PII, API keys in plaintext) before data reaches any downstream store, and reduce noise by dropping debug logs that add volume without adding signal. Latency in the telemetry pipeline directly limits the viability of automated remediation, because you cannot automate what you cannot observe in time to act.

## No-touch instrumentation via Kubernetes Operator

Auto-instrumentation (no code modification required) is table stakes. No-touch instrumentation, in which instrumentation occurs entirely at the platform layer via Kubernetes CRDs and annotations with zero developer involvement, is a must-have capability for organizations operating at scale. The OpenTelemetry Operator is the current reference implementation of this pattern. This is the practical definition of shift-down observability: the platform absorbs the instrumentation burden, allowing developers to focus on business logic.

### FinOps integration within the Observability Plane

Cost visibility must be a first-class capability of the Observability Plane, not a separate function managed by a different team. Platform teams need real-time visibility into telemetry cost drivers - cardinality explosions, unexpected volume spikes from a single developer instrumentation change, egress charges from cross-region data flows - at the same granularity as they have visibility into system health. Without this, cost management is reactive (discovered at billing time) rather than proactive (detected and addressed in real time).

### Observability pipeline latency sufficient for automated remediation

This is a binary capability gate. If the observability pipeline introduces latency exceeding the response-time requirements of automated remediation workflows, the entire agentic remediation use case is blocked. You must explicitly test pipeline latency under production-representative load conditions, not just under synthetic benchmarks.

## Standard capabilities

Mature implementations include these capabilities, and you can expect them from established vendors, but their absence does not immediately disqualify a solution.

### SLO and error budget tracking with automated alerting

Service Level Objectives, defined as code and enforced through the platform, provide the back-pressure mechanism that prevents agentic coding velocity from silently degrading reliability. SLOs are the organizational signal that tells platform teams when to slow down and when to invest in remediation. Platforms that do not surface SLO health as a first-class observability signal are missing a critical organizational feedback loop.

### Dashboards-as-code with GitOps lifecycle management

Dashboards and alert configurations need to be version-controlled, peer-reviewed, and deployed through CI/CD pipelines - not manually edited through vendor UIs. Using tools like Perses for dashboard definition in YAML ensures that observability configurations are auditable, portable across tools, and recoverable after incidents. This approach aligns observability management with the broader platform engineering ethos of treating everything as code.

## Multi-signal correlation (logs, metrics, traces)

The ability to navigate from a metric anomaly to the correlated trace to the relevant log line, without switching tools or manually constructing queries, is the operational definition of observability as distinct from monitoring. Platforms that treat logs, metrics, and traces as separate silos with distinct query semantics impose cognitive load on the operators who most need to move quickly during incidents.

## Role-based access control and multi-tenant data isolation

In a shared platform serving multiple application teams, telemetry data must be partitioned by team, service, and environment. RBAC for telemetry is not a security nice-to-have - it is an operational requirement for any platform serving more than a handful of teams.

# Optional and advanced capabilities

These capabilities represent significant competitive differentiation today and are likely to become standard within 12 - 24 months.

## Agentic AI for automated root cause analysis and remediation

Connecting observability data to an autonomous investigation agent, one that pursues multiple diagnostic hypotheses simultaneously, presents ranked root cause candidates with supporting evidence, and executes approved remediation actions, represents the most significant capability evolution in this market. The maturity progression follows a two-way door model: start with reversible, low-risk actions (feature flag toggles, pod restarts, rollbacks) before expanding to more complex remediations. Human approval gates at each stage are the trust-building mechanism that enables progressive automation.

## Edge-side tail sampling and context engineering

Performing tail sampling within the customer's environment, before data is transmitted to any downstream store, eliminates egress costs entirely for the data that is dropped, rather than merely reducing storage costs after ingestion. Context engineering extends this pattern further: preprocessing telemetry to store only anomalous or derived data maximizes LLM attention model efficiency and optimizes for agentic rather than human consumption. This architectural pattern is structurally unavailable to platforms whose revenue model depends on ingestion volume.

## AI/ML-specific observability

As organizations deploy agentic workloads into production, the Observability Plane must extend beyond infrastructure metrics to cover the behavior of AI systems themselves: session-level evaluation (measuring agent outcomes across entire user sessions, not individual LLM calls), drift detection, and hallucination detection. These capabilities require purpose-built data stores and evaluation frameworks that traditional infrastructure observability platforms do not provide.

# 4

# Market direction and trends

## The shift-down philosophy: Instrumentation as platform infrastructure

Observability is no longer only a capability that platform teams provide to developers. It has become a capability that platform teams embed into the infrastructure so that developers no longer have to proactively take care of it, but get it by design.

The OpenTelemetry Operator pattern enables platform teams to instrument any service running on Kubernetes by adding an annotation to the deployment manifest - no application code changes required. Instrumentation configuration, sampling logic, semantic convention enforcement, and collector routing are all managed at the platform layer through CRDs and GitOps pipelines. Developers receive pre-wired logs, metrics, and traces correlated out of the box.

AI coding assistants are accelerating code velocity, and the volume of new services and deployments is increasing faster than any team-by-team instrumentation approach can keep pace with. Survey data confirms both the aspiration and the implementation gap: 47.5% of respondents describe their platform philosophy as shift-down.

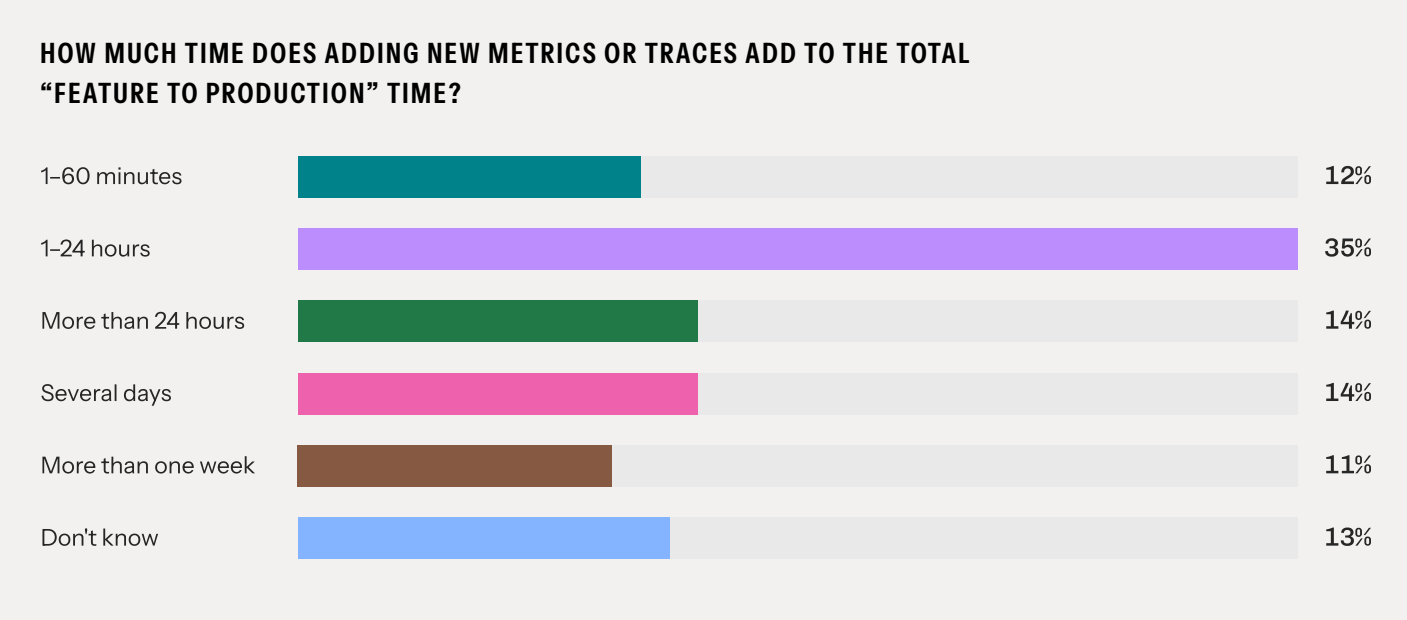


**Figure 6**: Increase of "Feature to Production" time due to adding new metrics or traces

Yet 39% report instrumentation still adds more than 24 hours to their feature-to-production cycle, and only 18% have automated more than 50% of observability tasks. The vision is widely shared; the execution is not yet widespread.

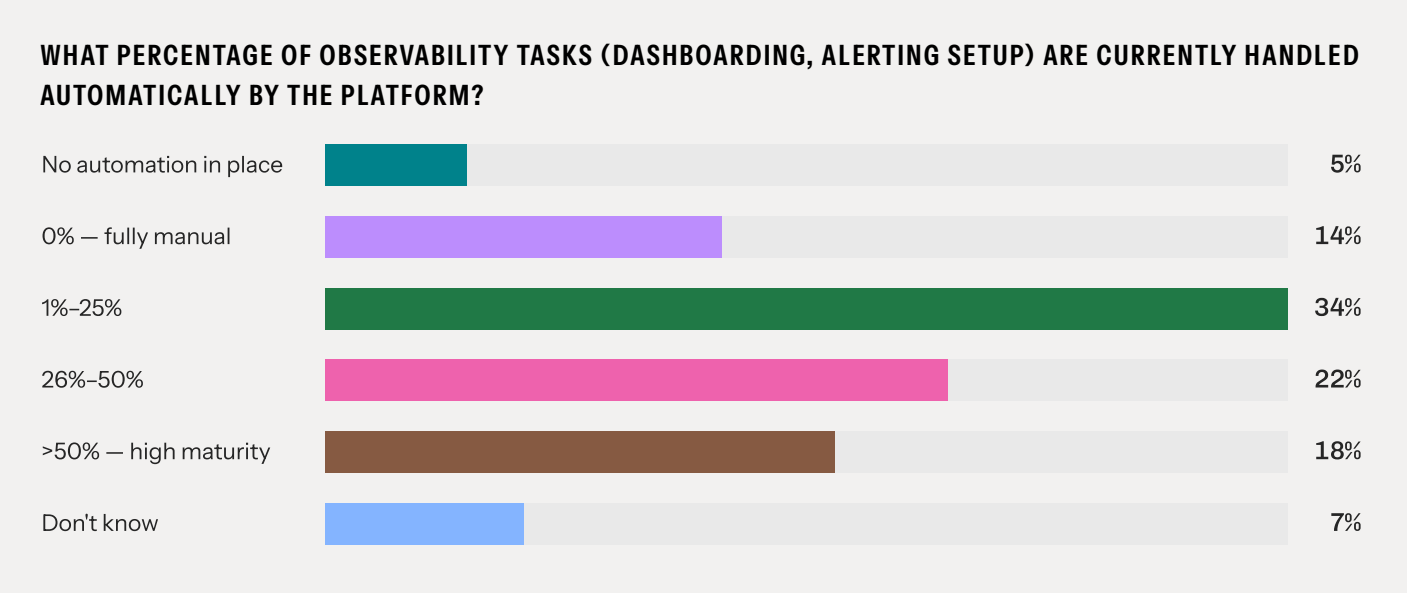


**Figure 7**: Observability tasks handled by the platform

Organizations that have adopted agentic coding heavily are already observing measurable increases in incident rates, which is a direct consequence of shipping more code faster without a corresponding increase in observability coverage. Shift-down instrumentation is the platform-layer response to this quality degradation signal. It is the only approach that can maintain observability coverage as agentic coding scales.

eBPF-based instrumentation, which captures telemetry at the kernel level without any application-layer involvement, extends the shift-down pattern to workloads that cannot be instrumented through the OTel Operator: legacy applications, third-party services, and network-layer events. Security teams currently position this capability primarily as a network policy enforcement and runtime threat-detection mechanism, while platform teams are increasingly evaluating it for observability use cases. Treat it as an emerging capability rather than a current standard.

# Agentic AI: From dashboard consumer to autonomous operator

Human operators investigate incidents serially - one hypothesis at a time, navigating dashboards and query interfaces. Agentic AI tools investigate incidents in parallel, pursuing 10-12 simultaneous hypotheses, each hitting observability endpoints independently. This is a fundamentally different consumption pattern that existing architectures were not designed to support.

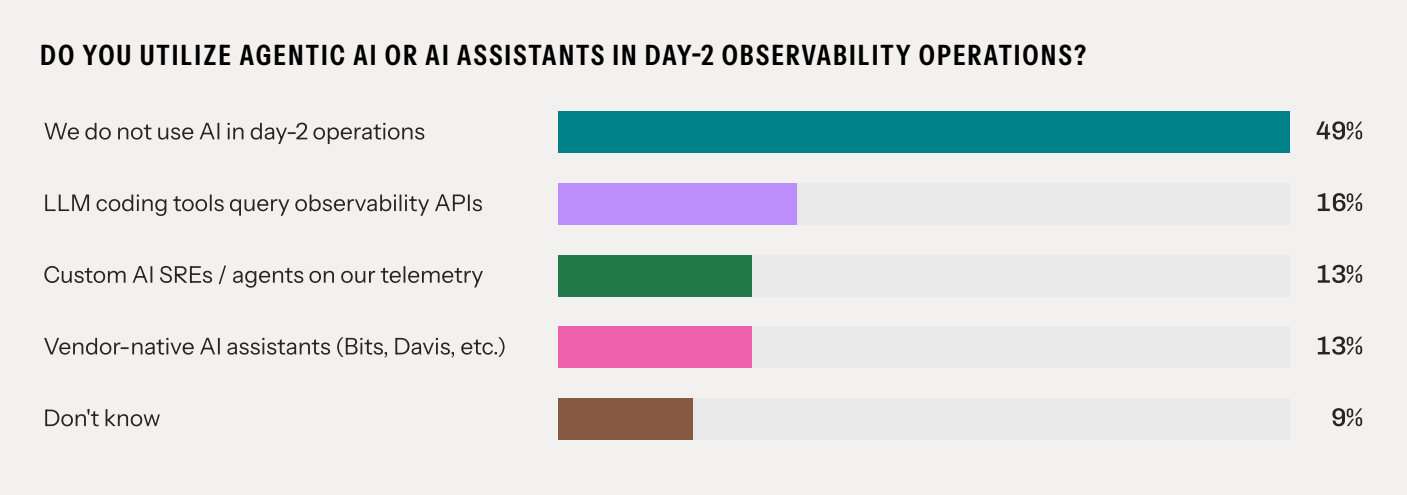


**Figure 8**: Utilization of AI in day-2 opertations

13% of survey respondents are building custom AI SREs on open-source frameworks; 16% are using LLMs and coding assistants to query observability APIs; 13% rely on vendor-native AI assistants; and 49% are not using AI for day-2 observability operations at all. The adoption curve is real but still early but teams investing in the Observability Plane architecture now are building a structural advantage before the curve steepens.

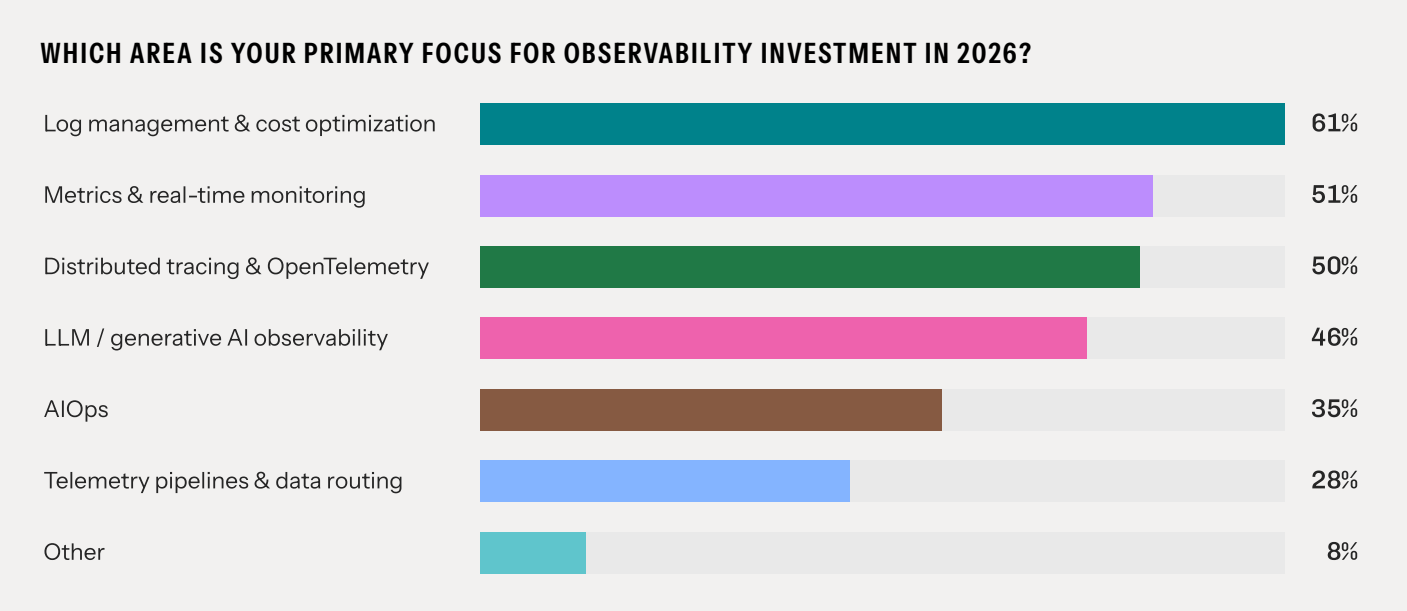


**Figure 9**: Primary focus areas for observability investment in 2026

Depending on the stage of the AI journey you are at, different use cases are seen. Early-stage implementations focus on AI-assisted investigation, presenting ranked root cause hypotheses with supporting evidence, reducing the time from alert to diagnosis. Mid-stage implementations add automated execution of reversible actions: feature flag toggles, pod restarts, and rollback to previous deployment. Advanced implementations extend to more complex infrastructure changes, with human approval gates that progressively widen as trust is established through demonstrated reliability.

The next architectural evolution moves beyond storing all telemetry for human consumption toward storing only the context that agents need for effective investigation. Pre-processing telemetry to identify anomalous events, storing derivative summaries rather than raw data, and pre-embedding context for efficient LLM consumption. These techniques define what “optimizing for agentic consumption” means in practice. A useful three-tier hierarchy frames the data reduction opportunity: raw telemetry, human-relevant signals, and agent-relevant context. Humans need about a magnitude less telemetry than they generate, and LLMs need about a magnitude less than what humans need.

A credible emerging signal suggests that the primary interface through which developers and agents interact with observability data will shift from dedicated observability UIs toward embedded agentic interfaces - IDE integrations, Slack and Teams bots, and MCP server connections. This transition is estimated to take 2-4 years from mainstream adoption, but you can architect your data layer now to support multiple consumption interfaces rather than optimizing exclusively for dashboard-based human interaction.

# FinOps convergence: Cost as a first-class observability signal

Production IDP reference architectures across multiple cloud providers consistently place cost management tooling within the Observability Plane. The same telemetry pipeline that routes logs and metrics to monitoring tools also determines what data is ingested, stored, and billed. Cost control and signal quality are the same problem viewed from different angles.

The cardinality explosion mechanism illustrates why cost management cannot be delegated to individual development teams. A single developer adding a high-cardinality tag, such as a user ID, to a metric can trigger a cost explosion that is invisible until the billing cycle. The number of unique time series created by that single instrumentation change can multiply by orders of magnitude, depending on the cardinality of the tag value. This is a recurring operational event at organizations running Kubernetes at scale. Platform teams that do not have real-time cardinality visibility cannot prevent these events - they can only respond to them after the cost has been incurred.

Pipeline-layer cost reduction mechanisms address this structurally (based on a vendor example):

- **Null field removal** reduces payload size by up to 30% without any signal loss.
- **Log metricization** converts high-volume repetitive log events into a single metric with a count, rather than storing each event individually.
- **Tail sampling** retains only the traces that carry diagnostic value (errors, slow requests, anomalous patterns) and drops the rest.
- **Edge-side processing** performs all of the preceding steps within the customer's environment before data is transmitted, eliminating egress costs for dropped data entirely.

Agentic AI workloads will dramatically accelerate the telemetry cost problem. Each LLM query generates significantly more telemetry than a standard application log event, and AI SRE tools that fan out to multiple simultaneous hypotheses multiply this effect further.

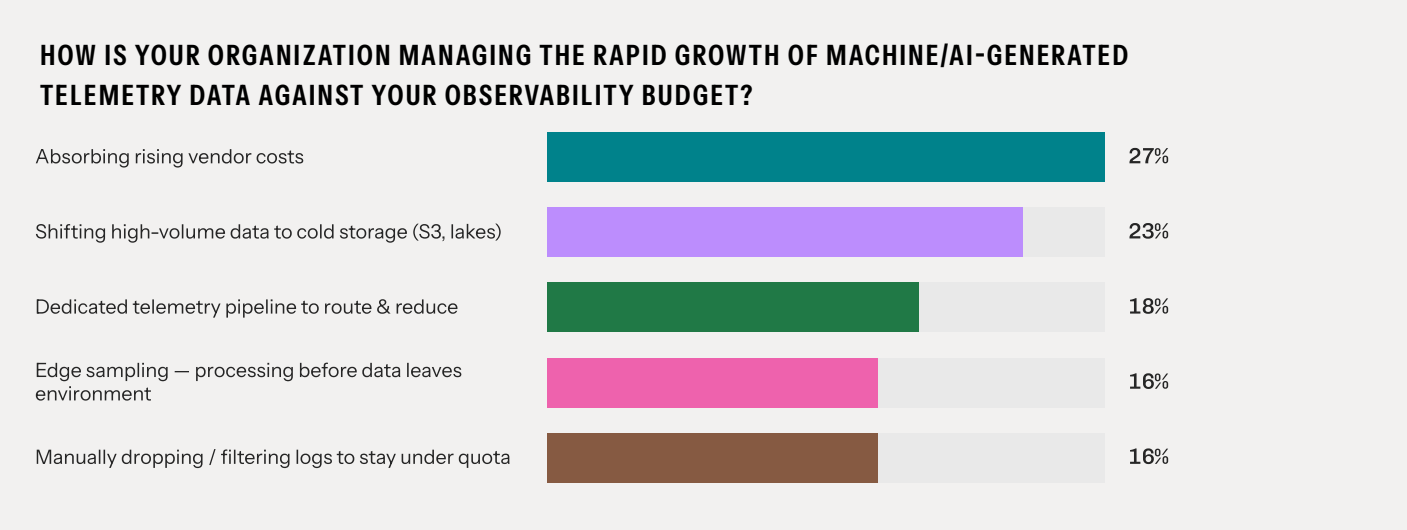


**Figure 10**: Strategies for managing AI-caused data growth against observability budget

Platform teams that have not implemented pipeline-layer cost controls before deploying agentic tooling at scale will face a cost event that is structurally difficult to reverse.

# OTel standardization as AI readiness

OpenTelemetry standardization is no longer primarily a strategy to avoid vendor lock-in. LLMs are trained on public OTel documentation, semantic conventions, and sample applications. OTel-structured telemetry data is inherently more interpretable by LLMs than proprietary or inconsistently structured data. Platforms that enforce OTel semantic conventions produce telemetry that AI agents can reason about more effectively, with less prompt engineering overhead and more reliable root cause identification.

Every investment in OTel posture, i.e. adding semantic conventions, enriching context propagation, standardizing field names, simultaneously improves observability quality for human operators and AI readiness for agentic tools. This compounding return is a meaningful strategic argument for platform teams that need to justify standardization work to engineering leadership.

Incumbent observability platforms face a structural constraint in responding to the edge-side data reduction trend. A platform whose revenue model is based on data ingestion volume cannot build a capability that reduces ingestion volume by 50-80% without cannibalizing its own revenue. This creates a structural competitive moat for OTel-native platforms built from the ground up - they can implement aggressive data reduction without conflicting business model incentives. Factor this structural dynamic into your vendor evaluation criteria.

# 5

# Market analysis: Adoption dynamics

# Benefits for the platform team and the developer

## Reduced cognitive load through shift-down instrumentation

When instrumentation, sampling, and semantic convention enforcement are managed at the platform layer, developers receive observability as a default capability rather than a responsibility. This eliminates the instrumentation toil that currently consumes significant engineering time at organizations without mature platform observability, and ensures consistent telemetry quality across all services regardless of individual team practices.

## Faster incident resolution through AI-assisted investigation

AI-assisted root cause analysis reduces the time from alert to diagnosis by automating the hypothesis generation and evidence gathering steps that currently require experienced SRE judgment. For organizations where downtime carries significant revenue impact, as in high-transaction environments, event-driven businesses, or regulated services, the ROI of even modest MTTR reduction is substantial enough to justify the platform investment in a single incident.

## Cost control through pipeline-layer intelligence

Organizations that implement telemetry pipeline controls, like sampling, metricization, null field removal, and edge-side processing, can reduce observability tool spend substantially without reducing signal quality for the use cases that matter. This is a structural cost reduction that frees budget for platform investment in higher-value capabilities.

## AI readiness as a compounding return

Organizations that standardize on OTel and enforce semantic conventions are simultaneously improving their observability quality and their AI readiness. Every investment in telemetry hygiene compounds over time as agentic tooling becomes more prevalent.

# Challenges and friction points

## Opaque and unpredictable pricing models

The most consistently cited adoption barrier of customers is the inability to predict observability tool costs before they arrive on the invoice. Cardinality explosions, unexpected volume spikes, and cross-region egress charges create billing events that are invisible until after the fact. When evaluating vendors, a relevant factor is consumption-based pricing models with real-time cost visibility over capacity-based models that require purchasing at peak.

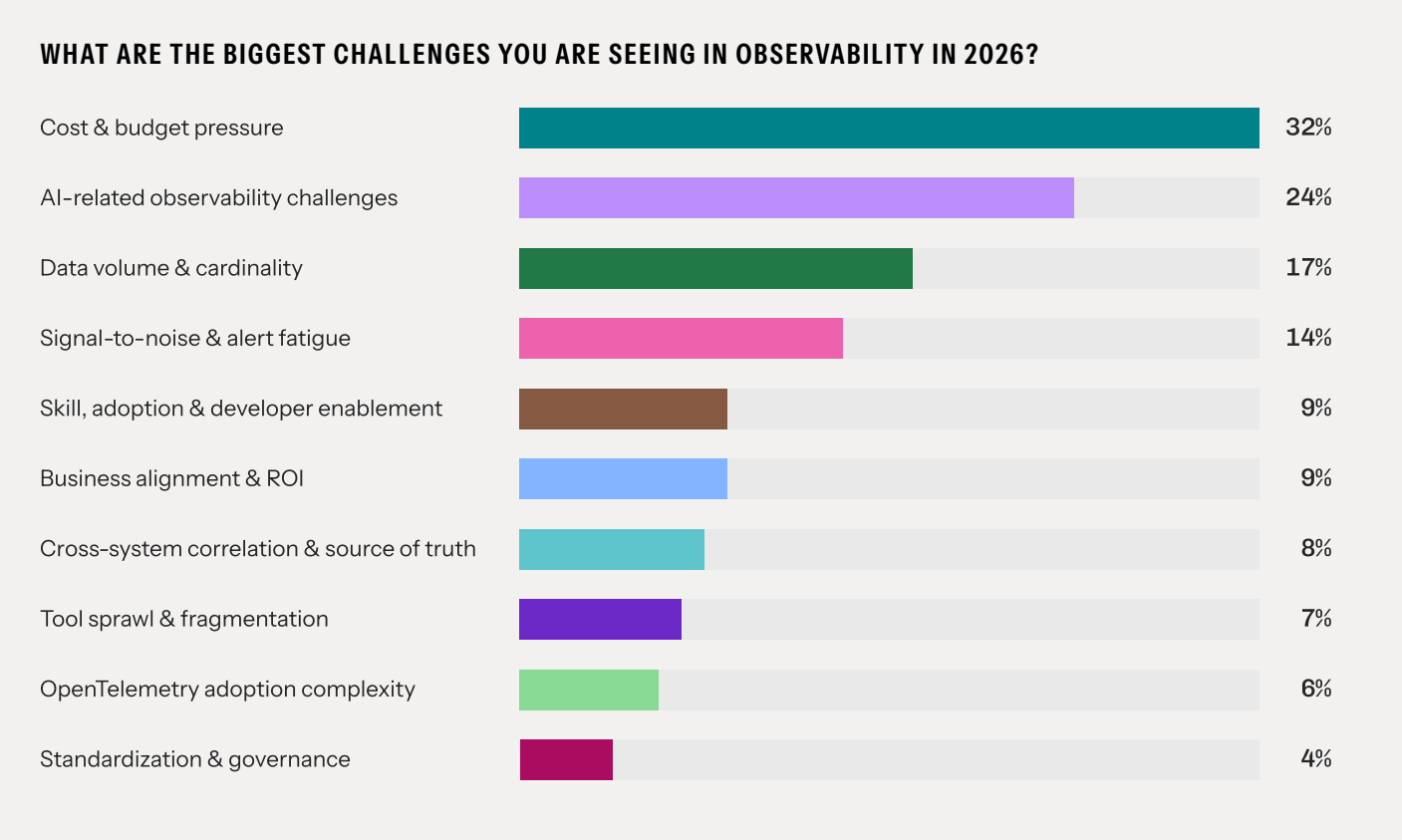


**Figure 11**: Biggest observability challenges in 2026

## The build-vs-buy decision for pipeline tooling

The primary competitive dynamic in the telemetry pipeline market is not vendor vs. vendor, it is build vs. buy. Platform teams with sufficient engineering capacity often attempt to build their own pipeline tooling using Python scripts, regex-based processors, and open-source streaming frameworks. These solutions work until they encounter scale, backpressure events (when error rates spike and log volume overwhelms the pipeline), or the need for sophisticated sampling logic. The hidden cost of maintaining custom pipeline infrastructure is rarely accounted for in the build-vs-buy analysis.

## Cultural resistance to agentic remediation

The human-in-the-loop requirement for automated remediation is not merely a technical constraint: it is an organizational trust-building process.

SURVEY SIGNAL

15% of respondents specifically cite resistance to automation and lack of trust in automated alerts as a cultural blocker, consistent with the SRE trust-building pattern described above.

SRE teams that have spent years developing intuition for incident response are understandably skeptical of autonomous systems making infrastructure changes. Adoption requires a progressive approach: start with AI-assisted investigation (no automated actions), then introduce reversible automated actions with explicit human approval, then progressively widen the automation scope as trust is established through demonstrated reliability.

## Integration complexity with legacy IT Ops tooling

Organizations with significant investments in legacy monitoring infrastructure face integration complexity when adopting modern observability platforms. The telemetry pipeline layer is the primary integration mechanism - it can receive data from legacy agents and proprietary formats, normalize and enrich it, and route it to modern destinations, but this requires platform engineering investment that is often underestimated.

### Latency requirements for automated remediation

Not all observability pipelines are architecturally capable of supporting automated remediation. Pipeline latency that is acceptable for human-reviewed dashboards may be too high for the response time requirements of automated incident response. You must explicitly evaluate pipeline latency under production-representative conditions before committing to an agentic remediation architecture.

## Vendor landscape categorization

### Full-stack incumbents

Comprehensive platforms offering metrics, logs, traces, APM, RUM, synthetics, and increasingly AI-assisted investigation within a single proprietary data store and UI. Their primary value proposition is breadth and integration depth. Their primary limitations are pricing opacity at scale, structural inability to implement aggressive data reduction without revenue impact, and proprietary data formats that create lock-in. Best suited for organizations that prioritize operational simplicity over cost optimization and architectural flexibility.

### OTel-native challengers

Platforms built from the ground up on OpenTelemetry standards, with no proprietary data format layer. Their primary value proposition is cost efficiency (enabled by aggressive data reduction without business model conflict), AI readiness (OTel-structured data is inherently more LLM-interpretable), and vendor lock-in avoidance. Their primary limitation is breadth - they are typically earlier in building out the full feature surface of incumbent platforms. Best suited for born-in-the-cloud organizations with high data volumes and strong OTel adoption.

### Pipeline and context specialists

Vendors focused exclusively on the telemetry pipeline layer - receiving data from any source, processing and enriching it, and routing it to any destination. Their primary value proposition is vendor neutrality (they work with any upstream source and any downstream destination), cost reduction (through sampling, metricization, and null field removal), and migration facilitation (enabling organizations to move between observability platforms without rip-and-replace agent changes). Their primary limitation is that they do not provide the analysis and visualization layer - they are infrastructure for the Observability Plane, not the plane itself.

## AI/LLM native platforms

Purpose-built platforms for observing AI and agentic workloads - LLM tracing, session-level evaluation, continuous improvement loops, drift detection, and hallucination detection. These capabilities do not exist in traditional infrastructure observability platforms and require purpose-built data stores, evaluation frameworks, and feedback mechanisms. As organizations deploy more agentic workloads into production, this category will become a required component of the Observability Plane rather than an optional add-on.

## Runtime intelligence

Solutions that capture live production state on demand rather than through continuous telemetry collection. Instead of instrumenting everything upfront, they give developers and AI coding assistants direct access to runtime behavior when investigation is needed, for example, from within IDEs or agentic workflows, without the overhead of always-on APM agents. Their primary value proposition is targeted, low-overhead inspection that surfaces what is actually happening in production code without requiring pipeline investment or instrumentation changes. These tools are optimized for precise on-demand debugging rather than trend analysis, SLO tracking, or historical signal correlation.

# 6

# Representative vendors

This section features a selection of vendor profiles that highlight the landscape of observability solutions. The aim is to showcase the diversity and variety of observability vendors while emphasizing key trends, features, and criteria discussed in this report. It is important to note that vendors do not need to meet all criteria to be included. Additionally, these examples are meant for illustrative purposes only, and Weave Intelligence does not endorse any specific vendors.

# Vendor selection criteria

- **Telemetry pipeline or edge processing with demonstrable ingestion volume reduction** - addresses the FinOps convergence imperative and the structural gap between telemetry data growth (28 - 40% annually) and flat IT budgets.
- **Agentic AI capabilities for automated root cause analysis, incident investigation, or remediation workflows** - addresses the shift from human-driven to agent-driven observability and the agentic fan-out problem, where AI SRE tools generate 10 - 100x the query load of human operators against observability endpoints.
- **Native OTel support or framework-agnostic integration** - addresses both vendor lock-in avoidance and AI readiness, given that LLMs are trained on OTel documentation and perform more reliably on OTel-structured telemetry than on proprietary or inconsistently structured data.
- **Integration of observability data into platform engineering workflows, IDP interfaces, or chat interfaces** - addresses the cognitive load reduction mandate and the shift-down instrumentation philosophy, where the platform absorbs instrumentation burden rather than delegating it to individual development teams.

# Representative vendor landscape

## OTel-native platforms

Vendors in this category have built their core architecture around OpenTelemetry semantic conventions from the ground up - not as a retrofit or integration layer, but as the foundational data model.

**Honeycomb** ingests high-cardinality event data natively through OTel and exposes it for exploratory analysis without pre-aggregation, preserving the full dimensionality that AI-assisted root cause analysis requires.

**Elastic** provides a schema-agnostic observability backend that accepts OTel signals across logs, metrics, and traces while integrating with the broader Elastic ecosystem for search and security signal correlation.

**Dash0** enforces OTel semantic conventions at ingestion, performs in-environment tail sampling before egress - reducing transmitted data volumes from tens of millions of spans per second to hundreds of thousands at production scale - and embeds agentic AI capabilities throughout the platform rather than surfacing them as a separate interface.

## Pipeline and context specialists

Vendors in this category operate as a processing layer between telemetry sources and observability backends, focused on data reduction, routing, normalization, PII redaction, and - at the frontier - context engineering for agentic consumption.

**Cribl** routes telemetry from any source to any destination through a vendor-neutral pipeline that applies compression, null-field removal, format normalization, and PII redaction at scale, with composable RBAC and pipeline-as-code capabilities designed for platform engineering governance models.

**Mezmo** pre-processes telemetry in real time using machine learning to identify anomalous signals, pre-embed log data, and generate real-time service dependency maps from trace data - structuring telemetry specifically to maximize LLM attention model effectiveness rather than simply reducing volume.

**Chronosphere** provides a cost-governance-first observability backend with a Control Plane that surfaces cardinality spikes in real time, enables aggregation rule creation against unused tags, and prices on retained data rather than peak ingestion capacity - making cost reduction a structural property of the platform rather than an operational discipline.

## Full-stack incumbents

Vendors in this category provide comprehensive observability suites with broad feature coverage, established enterprise install bases, and SaaS-first deliverv models.

**Datadog** unifies metrics, traces, logs, and user-experience data across infrastructure and applications in a single platform with extensive integrations, AI-assisted analysis features, and a broad enterprise support model.

**Dynatrace** provides full-stack observability with automated topology discovery, AI-driven root cause analysis through its Davis engine, and broad coverage across cloud, on-premises, and hybrid environments.

**New Relic** delivers unified telemetry ingestion with OTel compatibility, AI-assisted alerting, and a consumption-based pricing model that has evolved from its legacy per-host structure.

**Grafana** provides a source-agnostic observability platform that connects to existing data sources - Prometheus, Loki, Tempo, and third-party backends - without requiring data migration, with deployment flexibility spanning OSS, enterprise, and SaaS models.

**Amazon CloudWatch and X-Ray** deliver monitoring and distributed tracing natively within the AWS ecosystem, with consolidated billing through AWS accounts and deep integration with AWS-managed services that makes them the default observability layer for AWS-native architectures.

## AI/LLM native

Vendors in this category are purpose-built for observability of AI agents and LLM workloads, or for using AI agents to observe and remediate traditional infrastructure.

**Arize AI** provides session-level evaluation of LLM applications and multi-agent systems - measuring whether an agent accomplished a user's goal across a full multi-turn interaction, not just whether individual calls succeeded - with a purpose-built database designed for the query patterns and evaluation workloads that LLM observability generates at scale.

**Robusta** deploys HolmesGPT, an open-source AI agent for Kubernetes incident triage that connects to existing observability backends including Datadog, New Relic, and AWS to investigate alerts, generate root cause hypotheses, and surface remediation recommendations without requiring a backend replacement.

## Runtime intelligence

**Lightrun** is a developer-native observability platform that captures live runtime state on demand, directly from within the IDE and without continuous telemetry collection. It gives developers and AI coding assistants direct access to production system behavior for root cause analysis and autonomous debugging, without the overhead of always-on APM agents.

# Structural dynamics shaping the landscape

Two structural dynamics shape how you read this landscape, and neither is visible from individual vendor feature comparisons.

Full-stack incumbents whose revenue model depends on ingestion volume face a structural constraint when building aggressive data-reduction capabilities. A platform billing on data ingested cannot build a capability that reduces ingestion by 50-80% without cannibalizing its own revenue. Engineering investment does not close this gap - it is a business model conflict. OTel-native platforms and pipeline and context specialists do not face this constraint, which creates a durable value proposition regardless of feature parity on other dimensions.

A convergence signal is also worth naming explicitly. Pipeline and context specialists and AI/LLM native platforms are both optimizing for agent-relevant context - the third tier of the data reduction hierarchy - but approaching it from opposite directions. Mezmo and Cribl arrive from the data reduction side, asking what can be dropped before transmission. Arize AI arrives from the AI evaluation side, asking what must be retained and structured for continuous improvement loops. As these two optimization pressures meet in the middle, the boundary between these categories will blur over the next 12 - 24 months, and vendors in both categories will face pressure to extend into each other’s territory.

# 7

# Strategic recommendations for Platform Teams

Agentic workloads are already amplifying telemetry volumes in test environments. Simultaneously, the instrumentation gap between platform capability and developer workflow is widening as agentic coding accelerates service creation faster than team-by-team observability adoption can keep pace. The five recommendations below address the architectural and organizational decisions that determine whether your Observability Plane absorbs the next wave of operational complexity or is forced to retrofit under production pressure.

# Implement pipeline-layer cost controls before agentic workloads reach production scale

Each LLM query generates significantly more telemetry than a standard application log event. AI SRE tools pursuing parallel hypotheses can generate 10–100x the query load on observability endpoints. Organizations that have begun deploying agentic tooling in test environments are already observing this amplification effect, and the cost event at production scale will be an order of magnitude larger. Implementing pipeline-layer controls after agentic workloads are in production means the cost event arrives before the mitigation does.

SURVEY SIGNAL

Only 34% of teams have implemented a pipeline or edge-sampling approach to cost reduction. 27% are absorbing rising vendor costs with no structural mitigation in place.

The production evidence for what these controls can achieve is concrete. Edge-side tail sampling reduced one major European retail company's telemetry spans by 20x, saving $2.6 million per year in egress costs alone. Null field removal reduces payload by approximately 30% without any signal loss. Log metricization converts high-volume repetitive events into single metrics, eliminating the storage cost of individual event records. Before deploying any AI SRE tooling in production, audit your current pipeline for three specific controls: tail sampling that retains errors and anomalous traces while dropping routine successful requests; null field removal applied to all log streams; and real-time cardinality monitoring that surfaces cost events before they reach the billing cycle. Treat these as prerequisites to agentic deployment, not a parallel workstream.

# Enforce OTel semantic conventions at the collector layer as a platform policy

OTel standardization is an AI readiness strategy with compounding returns. LLMs are trained on OpenTelemetry documentation, semantic conventions, and sample applications. OTel-structured telemetry is inherently more interpretable by AI agents, requiring less prompt engineering overhead and producing more reliable root cause identification. Organizations that allow teams to instrument arbitrarily — using custom field names, inconsistent label schemas, and proprietary agent formats — are accumulating AI readiness debt that compounds with every service and team that diverges from conventions.

SURVEY SIGNAL

20% of teams still rely exclusively on proprietary agents. Each month of divergence from OTel conventions is a month of AI-readiness debt that compounds as agentic tooling scales.

Semantic convention enforcement belongs at the OTel Collector layer, not in developer documentation. Configure the Collector to enrich incoming telemetry with missing standard fields — service name, environment, region, deployment ID — and to reject or flag telemetry that violates naming conventions before it reaches any downstream store. This is the practical definition of shift-down observability applied to data quality: the platform enforces the standard, and developers receive compliant telemetry as a default rather than a responsibility. Every investment in OTel posture simultaneously improves observability quality for human operators and agentic investigation effectiveness. That compounding return makes the standardization argument straightforward to justify to engineering leadership.

# Give developers direct access to live production observability — stop proxying through tickets

A consistent and underexamined gap in most Observability Plane implementations is the distance between the signals that exist and the developers who need them. Observability tooling built for Ops and SRE teams is oriented around dashboards, alert channels, and incident management workflows — not toward the developer who needs to understand why their service is behaving unexpectedly in production. The result is a proxy model: developers raise tickets, SREs investigate on their behalf, and resolution cycles expand accordingly.

As agentic coding tools accelerate code velocity and increase the volume of new services and deployments, this proxy model becomes untenable. Developers need safe, scoped, self-service access to production telemetry, within the governance boundaries the platform defines, but without opening a ticket or waiting for an SRE to be available. The platform's role is to make this access viable through RBAC, data isolation, and default instrumentation that ensures there is always something useful to see. Removing the access proxy is not a security compromise; it is the same shift-down philosophy applied to the investigation workflow rather than to the instrumentation layer.

# Treat telemetry as a data engineering problem, not just an ops tool

The operating assumption embedded in most observability architectures is that telemetry is generated by systems, consumed by operators, and discarded when its retention window expires. That assumption is no longer sufficient. Telemetry is increasingly the input to automated investigation, the training signal for evaluation models, the audit trail for compliance, and the cost driver that finance teams are starting to treat as a line item. Managing it as a byproduct of operations rather than as a data asset produces the reactive cost management, fragmented signal quality, and AI readiness debt described throughout this guide.

Applying data engineering discipline to telemetry means defining what gets collected and why, establishing tiered storage policies based on query frequency and retention requirements. Instead of relying on vendor defaults, we should

treat the telemetry schema as a versioned contract. It's important to manage the pipeline as platform infrastructure with defined SLAs, rather than viewing it as a configuration detail delegated to individual teams. Platform teams that have made this shift report structural cost reductions, more reliable agentic investigation outcomes, and an organizational credibility with finance and leadership that purely ops-oriented observability programs rarely achieve.

## Extend the Observability Plane to treat LLM and agentic workloads as a first-class concern

Traditional infrastructure observability platforms, including those with AI-assisted investigation features, do not provide the session-level evaluation, continuous improvement loops, drift detection, and hallucination detection that agentic workloads require. Platform teams that attempt to observe these workloads using infrastructure tooling will have visibility into latency and error rates but not into the quality, coherence, or reliability of agent outputs. That is a fundamental capability gap, not a configuration gap. The longer it persists after agentic workloads reach production, the harder it becomes to close.

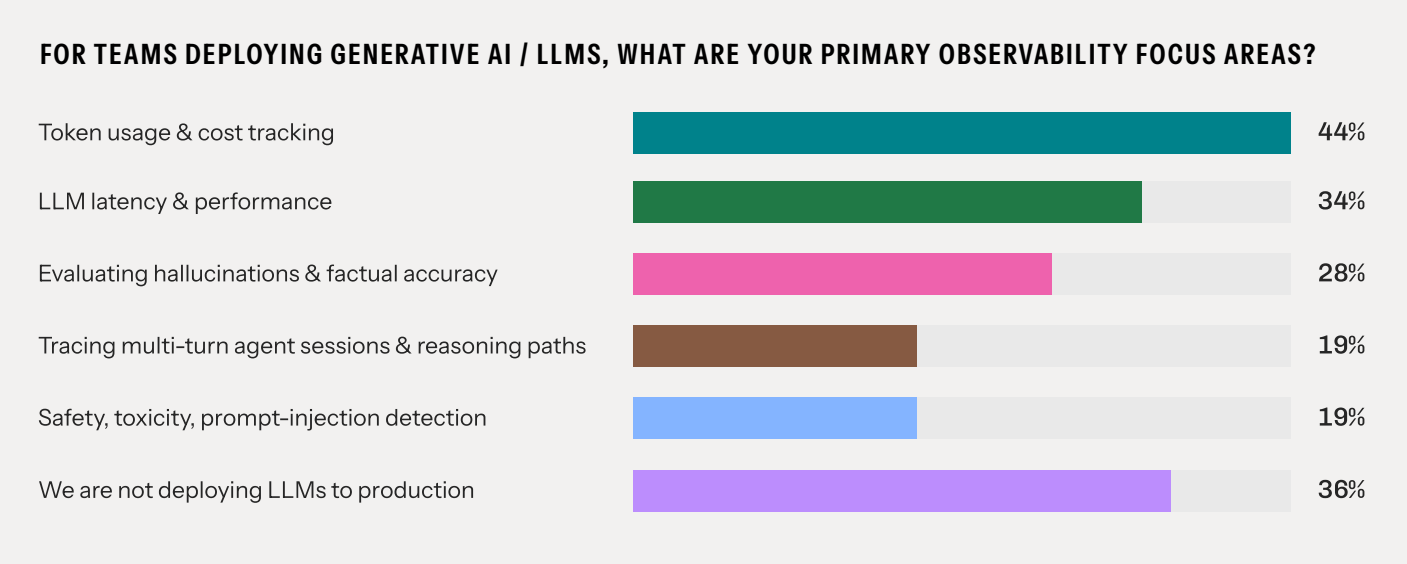


**Figure 12**: Focus areas of teams deploying generative AI/LLMs

SURVEY SIGNAL

Among teams deploying LLMs today, only 19% are tracing multi-turn agent sessions, and 28% are evaluating hallucinations and factual accuracy — significant gaps even among early AI adopters.

Session-level evaluation is the clearest marker of what this capability tier requires: agent outcomes must be measured across entire user sessions, not individual LLM calls. A customer support agent that resolves a query in session is a success; one that generates individually coherent responses but fails to resolve the underlying issue is a failure. Infrastructure observability cannot distinguish between these outcomes. Before deploying any agentic workload into production, define the evaluation criteria for that workload, implement session-level tracing from day one, and establish a baseline evaluation set from the first 30 days of production data. Treat AI observability as a platform product feature with the same ownership, SLAs, and enforced defaults as infrastructure observability — not as a capability the AI team figures out independently.

The vendor landscape will continue to evolve — categories will converge, new entrants will emerge, and the boundary between infrastructure observability and AI observability will blur as agentic workloads become the operational norm. The foundational principle will not change: observability, treated as a platform product with clear ownership, enforced standards, and automated defaults, is the prerequisite for every other platform capability. The architectural decisions you make in 2026 will determine whether your platform is positioned ahead of or behind the next wave of operational complexity.

# Appendix

## Authors

### Florian Lipp

Senior Analyst @ Weave Intelligence

Dr. Florian Lipp is an analyst at Weave Intelligence, where he researches how platform engineering teams design, operate, and evolve their platforms. His current work spans topics including cloud economics and observability, helping teams move beyond dashboards and cost reports toward feedback systems that actually inform decisions. He co-organizes PlatformCon, curates its content program, and serves as editor in chief at platformengineering.org, two of the community's most important gathering points.

### Dilek Altin

Senior Analyst @ Weave Intelligence

Dilek Altin is an analyst at Weave Intelligence with a deep technical background in platform architecture and engineering. His research sits at the intersection of artificial intelligence (AI) and platforms, covering two distinct but related challenges: building agentic development platforms that make platforms themselves AI-enabled and support agentic coding workflows, and designing platforms purpose-built for artificial intelligence and machine learning (AI/ML) workloads. Alongside this, he covers platform security, examining how teams build and govern platforms that can withstand the threat landscape of modern software delivery.

### Luca Galante

Managing Director and Lead Analyst @ Weave Intelligence

Luca Galante is an analyst and community leader in platform engineering. As managing director at Weave Intelligence, the leading analyst firm in the field, he benchmarks how the discipline is practiced across hundreds of enterprise organizations. He is the co-author of Thinking in Platforms, hosts PlatformCon, the largest annual platform engineering conference, and writes Platform Weekly, read by more than 100,000 platform engineers every Friday. He has chronicledthe discipline as much as he has shaped the community around it.

# About the survey

The findings in this guide draw on a Weave Intelligence survey of 105 platform engineers, site reliability engineers, and operations leaders, fielded online in May and June 2026. The survey combined a scored self-assessment on observability maturity with open-ended questions on tooling, investment priorities, and cultural blockers to platform-native observability.

This research was conducted independently by Weave Intelligence. No vendor commissioned or sponsored the survey, and no vendor had input into the questions or influence over the findings.

As with any practitioner survey of this size, results should be read as directional rather than statistically representative of the platform engineering population as a whole. Percentages reflect the share of respondents who answered a given question; not all questions applied to all respondents, for example, questions on LLM/agentic AI observability practices were shown only to teams already deploying generative AI in production.

# Disclaimer

Weave Intelligence does not endorse any specific vendor, product, or service. The information contained in this report has been obtained from sources believed to be reliable. Weave Intelligence disclaims all warranties as to the accuracy, completeness, or adequacy of such information. This publication is provided on an “as-is” basis without warranty of any kind, either express or implied. Weave Intelligence shall have no liability for errors, omissions, or inadequacies in the information contained herein or for interpretations thereof.

# About Weave Intelligence

## Weave Intelligence

Weave Intelligence is a leading analyst firm specializing in platform engineering. By uniting a team of senior analysts with industry experts and enterprise leaders, we deliver the rigorous research that defines the field. We enable organizations to leverage the #1 trend in IT as the modern framework for operational excellence and innovation.

Weave Intelligence GmbH
Wöhlertstraße 12-13
10115 Berlin