Machine Learning Observability in Practice

Joran Leest
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
j.g.leest@vu.nl

Ilias Gerostathopoulos
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
i.g.gerostathopoulos@vu.nl

Patricia Lago
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
p.lago@vu.nl

Claudia Raibulet
Universita' degli Studi di Milano-Bicocca
Milan, Italy
claudia.raibulet@unimib.it

Abstract

Production machine learning (ML) systems fail silently – not with crashes, but through wrong decisions. While observability is recognized as critical for ML operations, there is a lack empirical evidence of what practitioners actually capture. This study presents empirical results on ML observability in practice through seven focus group sessions in several domains. We catalog the information practitioners systematically capture across ML systems and their environment and map how they use it to validate models, detect and diagnose faults, and explain observed degradations. Finally, we identify gaps in current practice and outline implications for tooling design and research to establish ML observability practices.

Keywords

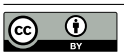
machine learning, monitoring, observability

ACM Reference Format:
Joran Leest, Ilias Gerostathopoulos, Patricia Lago, and Claudia Raibulet. 2026. Machine Learning Observability in Practice. In *2026 IEEE/ACM 5th International Conference on AI Engineering - Software Engineering for AI (CAIN '26), April 12–13, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3793653.3793771

1 Introduction

Software monitoring has matured through decades of practice and research, enabling engineers to track system health, validate correctness, and detect degradations using established reliability practices [9]. At its core, monitoring is enabled through observability [14, 40, 41, 46, 60] – the ability to measure and understand a system's behavior during operation [29, 60]. Practically, this means capturing the system context as logs, metrics, and traces [25]. As systems increasingly incorporate data pipelines and warehouses as core infrastructure, the notion of observability has extended its scope to include data quality metrics and lineage, enabling monitoring of these components [11, 50, 58].

With machine learning (ML), monitoring requires observability that extends beyond code and data to also capture learned behaviors, such as how models decide which customers receive loans, which content gets promoted, or which transactions trigger fraud alerts. Unlike traditional software, failures often do not cause crashes or logs; the system simply makes wrong decisions [8, 12, 52, 54]. These silent failures make ML monitoring especially challenging and call for new observability approaches, as we discuss below.



This work is licensed under a Creative Commons Attribution 4.0 International License.
CAIN '26, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2475-6/26/04
https://doi.org/10.1145/3793653.3793771

Monitoring ML systems requires distinguishing real failures from benign changes, which proves to be inherently ambiguous. When faults occur, they typically surface as distributional changes – changes in the statistical properties of the data that can degrade model performance, also known as data drift. [24, 61]. These drifts can come from internal faults (e.g., failed pipeline runs) or external factors (e.g., trends or events) [32, 53]. But neither origin nor magnitude reliably indicates whether action is needed – i.e., pipeline instrumentation alerts range from benign to critical, and external shifts from benign fluctuation to changes that break model assumptions. Moreover, teams often lack timely label feedback to assess whether data drift actually degrades model performance [26, 32, 53, 55, 65], leaving practitioners to observe drifts without reliably determining their cause or impact.

This ambiguity arises because ML systems function as computational systems and as models of external processes, requiring observability that captures both views. Synthesizing the literature, the authors of [37] conclude that effective monitoring indeed requires capturing information across both views of the system context: the ML system (dependency graphs and metadata showing how data, models, and applications interact) and the external environment (tracking exogenous drivers like sales campaigns).

Yet existing tools do not capture this unified system context. Tools provide capabilities for data lineage [17, 43], data validation [3, 30], and drift detection [6, 23], yet ML teams rarely receive actionable signals [51, 53, 55]. We argue that the problem is fragmented context. Tools capture pieces of information in isolation rather than connecting them, requiring practitioners to correlate relevant signals manually [10, 51, 53, 55]. A recent survey on ML operations (MLOps) confirms the tooling gap: monitoring and observability rank as the top challenge, with 77% of the respondents reporting either no monitoring tools or custom-built solutions [59].

We lack empirical evidence on how practitioners actually achieve ML observability in practice. Despite growing interest, there is not yet a commonly accepted framework for practice [47, 48, 54]. The descriptive model proposed in [37] offers initial understanding, but frameworks derived from literature may diverge from actual practice. This motivates our research question:

> *What information of the system context do practitioners use when monitoring an ML system?*

We present an empirical study of ML observability across organizations and domains. Informed by an a priori descriptive model [37] (Section 2), we ran focus groups and analyzed them using framework analysis (Section 3). We report what teams capture about system context, how they use it for monitoring, and remaining gaps and opportunities (Section 4). We provide the first empirical characterization of ML observability and, second, evaluate and refine the model (Section 5).

2 Background: Descriptive Model of System Context Information

We use the descriptive model from [37] to organize our findings, characterizing information relevant to ML monitoring by *system context* and *information type*. This enables us to catalog what practitioners capture – from core signals like feature and prediction statistics to broader contextual information.

2.1 System Context

The *system* dimension specifies the context of which information is captured. Whereas traditional observability centers on computational artefacts (e.g., code or data artefacts), ML systems require a broader perspective that captures both the technical ML system itself and the natural environment that generates the data it processes. In [37], the authors distinguish two views of system context: the natural *environment* and the technical *ML system*.

The *environment* represents the data-generating processes external to the ML system – the view a domain expert adopts when diagnosing faults in the model (e.g., "did an external event cause the performance drop?"). It includes three subsystems:

- *Reference domain* – the real-world process the model aims to capture, expressed as unobserved random variables over *features* (e.g., customer age) and a *target*. The target is the predicted variable, a domain property invariant to the ML system's operation, capturing either a realised event (e.g., a fraudulent transaction) or a propensity towards a future event (e.g., a defaulted loan).
- *Exogenous influences* – observable *exogenous variables* that drive the modeled domain (e.g., weather conditions);
- *Latent influences* – unobserved *latent variables* that can only be inferred from statistical patterns (e.g., customer trends).

The *ML system* covers the managed technical artefacts relevant to running the model – the engineer's view when interpreting an observed drift (e.g., "was it a failed job, corrupted data, or an infrastructure failure?"). It includes the following subsystems:

- *Processing pipeline* – *data transformations* from *source tables* to *feature tables* (e.g., clicks and sessions aggregated into a weekly activity feature);
- *Inference pipeline* – the serving layer where an ML model consumes features to produce predictions (e.g., a recommendation model producing ranked product preferences);
- *Application* – downstream decision policies converting predictions into actions (e.g., selecting a personalized promotion from ranked preferences).

These subsystems are causally ordered: exogenous and latent influences drive the reference domain; behavior is observed and engineered into features; the model maps features to predictions; and application policies turn predictions into actions.

2.2 Information Type

The type of information that can be captured of the system context varies by *aspect* (states, structures, or properties) asserted about that element, and the format in which this is *represented* (formal or informal) [37]. For brevity, we list the unique combinations between these dimensions observed in our study as *information types*:

- *Attributes* – numerical measures of the conditions under which a system element operates (e.g., inference latency).
- *Metadata* – semi-structured descriptions of a system element's conditions (e.g., a table schema).
- *Metrics* – numerical measures that evaluate a system element (e.g., distribution divergence).
- *Identifiers* – numerical key-value measures that identify records as belonging to an entity (e.g., customer geography).
- *Slices* – logical definitions of entities to partition records (e.g., adult customers are aged 18-65).
- *Graphs* – graphical representations of relations between system elements (e.g., causal diagram of feature variables).
- *Assertions* – logical predicates that encode the properties of a system element (e.g., customer age is in range [0, 100]).

3 Study Design and Execution

To address our main research question (Section 1), we conducted focus group sessions exploring two sub-questions: *"What information of the system context do practitioners capture and use in monitoring their ML system?"* (RQ1), and *"What information is used, but not systematically captured (or only informally managed)?"* (RQ2). We elicit current ML observability practices – the system information practitioners systematically capture and rely on, organizing items according to the descriptive model reused from [37], and identify gaps in the captured information.

3.1 Participant Recruiting

We recruited participants through LinkedIn[1], Slack MLOps communities[2], and our professional network. The recruitment message targeted anyone involved in monitoring ML systems, explicitly welcoming diverse roles including but not limited to data scientists, ML/data/software engineers, platform engineers, and business stakeholders, mitigating role representation bias by capturing holistic perspectives rather than preconceived notions about roles. We mitigated selection bias by recruiting through multiple channels and including participants at any experience level.

3.2 Sessions

We conducted seven focus group sessions in six organizations that span the banking, financial technology, industrial technology, and e-commerce domains. Sessions lasted 54-82 minutes, with 4-9 participants in each with a total of 37 participants.

Participants represented four self-declared roles: (1) *data scientists* (including ML scientists) focusing on model development; (2) *ML engineers* (including MLOps engineers) handling pipeline implementations and production operations; (3) *platform engineers* (including infrastructure engineers) managing ML tooling and infrastructure; and (4) *business stakeholders* (project managers, product owners) providing organizational context.

We summarize the sessions information in Table 1. Notice that sessions [S1] and [S2] were from the same organization but involved different teams, non-overlapping participants, and distinct subdomains. We therefore treat these as separate sessions.

[1] https://www.linkedin.com/feed/
[2] https://mlops.community/

Table 1: Anonymised descriptions of focus group sessions

ID	Domain	Participants†	Org. Size	Maturity
S1*	Banking	5 MLE	Large	Early
S2*	Banking	2 DS, 2 MLE	Large	Early
S3	Financial Tech	4 DS, 2 MLE	Medium	Advanced
S4	Industrial Tech	3 DS, 1 MLE	Small	Early
S5	E-commerce	4 DS, 1 MLE	Large	Early
S6	E-commerce	1 DS, 3 PE	Medium	Intermediate
S7	Banking	3 DS, 4 MLE, 2 BS	Large	Intermediate

Notes. * S1 and S2 represent separate sessions conducted with the same organization.
† DS = Data Scientist; MLE = ML Eng.; PE = Platform Eng.; BS = Business Stakeholder.

We defined organization size by employee count: small (<1,000), medium (1,000-10,000), and large (>10,000). We assessed monitoring maturity based on participants' self-reported experience and extent of tooling in place. *Early stage maturity* indicates primary reliance on manual processes with basic tracking that teams rarely act upon. *Intermediate stage maturity* involves active tracking and alerting but maintains largely manual processes with significant coverage gaps. *Advanced stage maturity* includes organizations with established monitoring platforms, though may still rely on manual processes.

We mitigated threats to generalizability by continuing recruitment until diversity was reached across roles, domains, and maturity levels, as shown in Table 1.

3.3 Procedure

Sessions used a printed poster displaying the six subsystems from the descriptive model reused from [37] (reference domain, exogenous influences, and latent influences for the external environment; processing pipeline, inference pipeline, and application for the ML system). The facilitators used post-it notes to document *general information* of the case, *where* incidents occurred, and *what* information was consulted throughout the session – using brief in-session member checks to verify interpretation [13].

Each session followed four phases. **First**, we elicited monitoring challenges and priorities to understand what practitioners find problematic. **Second**, we mapped participants' ML case onto the poster – e.g., the complexity of their data processing, the model domain, their application – for a shared understanding. **Third**, we elicited incident scenarios' deep-dives where participants walked through specific monitoring incidents, describing how issues surfaced, how they investigated, and how they diagnosed root causes. **Finally**, we conducted a wrap-up with member checking.

We mitigated researcher bias in how we introduced the descriptive model. We explained the six subsystems shown on the poster as a structure for discussion – using it to organize post-it notes consolidating their case – but did not explain the types of information that could be captured. We elicited information through neutral questions centered around monitoring practice – e.g., "how did the incident surface?", "how did you find the root-cause?" – rather than directly prompting information usage to avoid bias. This approach enabled validation of the reused framework's dimensions: The system context dimension was validated through participants recognizing their system in the provided structure, while the information type dimension was validated through inductive coding of naturally described information. We mitigated construct validity threats by including an open field on the poster for unmappable information, and ensured balanced role representation through round-robins and inclusive prompting of participants.

3.4 Data Collection and Analysis

We recorded and transcribed all focus group sessions. The transcripts served as our primary data source for analysis, with collected post-it notes as supplementary materials.

Again, to address RQ1, we employed framework analysis [27], using the descriptive model from [37] as our a priori analytical frame. We deductively coded *information items* – self-contained statements of explicitly captured information relevant to ML monitoring. While coding was performed at the full granularity of the model, we report simplified information-type categories (Section 2.2) for clarity. Applying the model as our coding frame provides evidence for its construct validity and face validity in operational settings. To mitigate the risk of forcing data into predefined categories, we treated unmappable content inductively. Recurrent unmappable content indicated systematic gaps in the original model; we formalized these as extensions to the descriptive model (Section 5.1) and integrated them into our results.

For RQ2, we inductively coded *uncaptured information items* – items that participants reported using or needing in practice but that are not systematically captured or available to the monitoring workflow (e.g., inaccessible sources, tacit knowledge).

We annotated each information item (RQ1, RQ2) with its participant-described monitoring use and grouped items by similar uses (e.g., fault detection, localization) into six monitoring activities.

The replication package of this study is provided separately[3].

4 Results

The resulting catalog of captured information is presented in Table 2, organized by system element and information type (as defined in Section 2). Table 3 shows how this information supports specific monitoring activities. We structure this section around six monitoring activities that emerged from our analysis. For each activity, we address both research questions: (RQ1) what information practitioners systematically capture and how they use it, and (RQ2) what information remains uncaptured despite being recognized as critical for the activity.

Two elements emerge from practice that the descriptive model in [37] did not capture. Following our framework analysis approach (Section 3.4), we integrate these as extensions. In particular, *Feature vectors* are a system element within inference pipeline representing the formatted inputs fed to models at inference time. *Outcome variables* are a system element within the *endogenous outcome* subsystem – a new subsystem capturing observable outcomes (such as business metrics) generated after the ML system's actions influence the environment. We discuss these extensions to the descriptive model in Section 5.1.

[3]Available at https://doi.org/10.5281/zenodo.17419114. It includes the invitation letter, focus group script, poster, codebook, and a mapping table aligning the model's classifications with the simplified information type categories (Section 2.2). Transcripts are not made publicly available to protect participants' privacy and sensitive details.

Table 2: Catalog of system information captured in ML monitoring practice, elicited from practitioner sessions. Columns show: *System Elements* (the context of which information is captured), *Information* (type of information captured), a description of each, concrete examples, a count of the number of unique sessions in which each appeared, and the corresponding session ID.

System Element	Information	Description	*Example*	Count	ID
Exogenous Var.	Identifiers	Key-value mappings that partition data into subgroups	*Device type: mobile/desktop*	3	[S3;S5–6]
Feature Var.	Slices	Expressions over features that partition data into subgroups.	*Power users (sessions>20/week)*	1	[S7]
	Assertions	Declarative domain constraints over feature variables.	*User age in range [18, 100]*	2	[S2;S6]
Target Var.	Assertions	Declarative domain constraints over target variables.	*Positive class rate: 4%-6%*	2	[S2;S6]
Source Table	Metadata	Descriptive information of source data.	*Table schema definitions*	2	[S2–3]
	Metrics	Summary measures computed from source data.	*Row counts, column nulls*	2	[S2;S6]
Transformation	Attributes	Operational measures of a data operation's execution.	*ETL job runtime*	2	[S2;S7]
	Metadata	Descriptive information of data operations.	*Mapping config file*	2	[S6–7]
	Graphs	Directed graph of data operations tracing the data lineage.	*Data dependency DAG*	2	[S6–7]
Feature Table	Attributes	Operational measures of stored feature columns.	*Table freshness*	1	[S2]
	Metrics	Summary measures over feature columns.	*Column consistency*	3	[S1–3;S6]
Feature Vector*	Metrics	Summary measures from served input vectors per feature.	*Null rates per feature*	5	[S2–3;S5–7]
Model	Attributes	Serving measures of the deployed model.	*Inference latency*	3	[S4–5;S7]
	Metadata	Descriptive information of the model artefact.	*Model card*	1	[S4]
Predictions	Metrics	Summary measures computed from model outputs.	*Predicted positive class rate*	6	[S1–3;S5–7]
Policy	Metadata	Descriptive information of the decision policy.	*Decision config version*	1	[S3]
Actions	Metrics	Summary measures of executed actions.	*Intervention rate*	4	[S1;S3–4;S7]
Outcome Var.*	Metrics	Summary measures of observed outcomes.	*Conversion rates*	4	[S3–6]

Legend System elements are grouped in colored subsystem boxes (= Exogenous Influences; = Reference Domain; = Processing Pipeline; = Inference Pipeline; = Application; = Endogenous Outcomes) and outlines indicate the system view (= environment; = ML system). (⟶) illustrate relations between subsystems, ordered by runtime sequence: environment generates data, ML system processes and acts, producing outcomes.

* This category extends the descriptive model of [37]; see Section 5.1 for suggested refinements to the model.

4.1 Validate through Outcome Signals

Practitioners track outcomes to validate whether the ML systems do have the desired impact. Yet, systematically capturing outcomes proves challenging and faces organizational barriers.

4.1.1 Business metrics and experiments to estimate real-world impact. Teams track **outcome metrics** to validate whether their ML systems achieve intended business impact (4/7 sessions; [S3–6]) – measuring customer satisfaction, conversion rates, or prevented losses that ultimately determine system success.

The most straightforward approach is tracking *direct outcomes*, where teams track business metrics during model rollouts to test whether offline improvements translate to measurable gains [S5–6]. But even this seemingly simple validation hits a wall. Attribution becomes nearly impossible because the internal system and external environment evolve continuously while effect sizes remain small. One practitioner captured the frustration [S6]:

> *"It's hard to see that back in online metrics... to attribute the change of a model... to the business KPI that are only observed all the way at the end."*

In some cases, a more fundamental measurement problem emerges. The system's own interventions censor the outcomes that the teams need to observe to validate the model [S3–4;S7]. Alternative outcomes effectively never occur and thus cannot be assessed – e.g, content recommendations hide user responses to alternative content, or early interventions with at-risk patients prevent natural

Table 3: Results organized by monitoring activity, mapped to captured *information* (gray) of the *system context* (colored).

	Section	
§4.1	**VALIDATE** Through Outcome Signals	Metrics
§4.2	**DETECT** Faults Through Instrumentation	Metrics; Attributes*
§4.3	**LOCALIZE** a Fault Through DataFlows	Graphs
§4.4	**IDENTIFY** Root Causes With Provenance	Metadata; Attributes*
§4.5	**VERIFY** Correctness with Domain Knowledge	Assertions
§4.6	**INTERPRET** Shifts Through Domain Context	Identifiers; Slices

Legend = Exogenous Influences; = Reference Domain; = Processing Pipeline; = Inference Pipeline; = Application; = Endogenous Outcomes.
* *Attributes* apply to only a subset of subsystems, but are grouped for visual simplicity.

readmission patterns. Teams address this issue through two strategies. Some collect structured customer feedback from affected users [S4], asking directly whether interventions proved to be appropriate given the observed context. Others track *potential outcomes* by deliberately withholding action from small holdout populations, enabling observation of what would happen without the model's intervention [S3;S5]. As one practitioner explained [S3]:

> *"We have control traffic, so there is a portion of the traffic that we do produce [predictions] for but we don't act on them."*

This interventional monitoring strategy actively elicits counterfactual outcomes that censoring would otherwise hide; the ability to observe outcomes thus requires controlled interventions that generate otherwise inaccessible signals.

4.1.2 Barriers to systematic outcome capture. Despite recognizing the need to track **outcome metrics**, many teams reported that systematically capturing **outcome metrics** remains challenging due to organizational constraints (4/7 sessions; [S1;S4;S6–7]). Some teams wanted to expand their capture of *direct outcomes* but faced barriers. One team wanted to systematically capture customer feedback but lacked the necessary product integration; the capability simply does not exist in the user-facing interface [S7]. Feedback delays further complicate validation, preventing teams from tracking validation results in reasonable timeframes [S1;S6]. Tracking *potential outcomes* through holdouts faces another barrier. While teams recognized feedback censoring as a problem, regulatory constraints and business risk prevented them from withholding treatment [S4;S7]. The intervention needed to generate the monitoring signal violated compliance requirements or threatened unacceptable business losses. As one practitioner explained the asymmetric risk [S4]:

> *"[False negatives] are costly... [false positives are] perceived as not as bad. But of course, it's better for us to have a high recall."*

Furthermore, continuous experimentation to systematically track (potential) *outcome metrics* often loses out to other priorities. Teams at earlier maturity stages prioritized developing models and tracking model performance over establishing online experimentation [S6]. Expensive experiments with small effect sizes reduced the priority level of outcome tracking – teams chose velocity over validation infrastructure.

4.2 Detect Faults through Instrumentation

All teams instrument their pipeline to emit metrics throughout the ML system – from data ingestion through feature computation to model predictions – aiming to detect failures close to where they originate. By tracking signals at each stage where faults might occur "to make sure technical metrics catch things before the business metrics" [S6]. Yet as we discuss in Section 4.2.2, the reality proves more complex – with gaps at system boundaries complicating fault detection.

4.2.1 Signals captured across the internal system. Practitioners prioritize monitoring for internal system faults – failures in data processing, model serving, and application logic. Among these, data integrity issues emerged as the dominant failure mode across sessions (6/7; [S1–3;S5–7]).

To catch these faults early, teams track data quality metrics across data processing stages. These metrics include a range of summary statistics: distribution divergence [S2], uniqueness [S6], min/max values [S1–2;S6], and number of NULL values [S1–6]. Teams reported a clear division of responsibility: tracking these metrics typically falls under data engineering teams rather than ML teams directly [S1–3;S6], with one practitioner noting [S6]:

> *"The data warehouse team is testing whatever they provide to us as ML engineers."*

Source table metrics track data quality on raw data sources after ingestion, catching corrupted data transmitted from third parties or customers. Two sessions reported systematically tracking these metrics [S2;S6], to catch problems "early in the process" [S2].

Feature table metrics track the processed data that models actually consume, detecting when tables become stale (not updated) [S1;S6], contain incorrect values [S1;S6], or are not populated at all [S1;S3;S6] – any of which would degrade model outputs. Three sessions reported tracking data quality metrics over feature table columns [S1;S3;S6], serving as an early warning system before corrupted features reach production models. While these metrics target issues within feature tables themselves, they also surface upstream faults that cascade through table dependencies. As one practitioner described [S6]:

> *"The data didn't get refreshed. But that was a table that we based another table on... but we were just missing a part."*

Once data is processed through the data pipelines, ML teams add instrumentation to track the serving layer, as discussed below.

Feature vector metrics capture data quality at the moment of model inference, computing summary statistics like distribution divergence, uniqueness, max and min values, and NULL counts on the formatted inputs fed to the model (5/7 sessions; [S2–3;S5–7]). These signals catch local serving issues – e.g., failed API calls to feature stores, or schema mismatches. However, practitioners predominantly tracked distribution metrics to monitor data state broadly – capturing upstream faults and natural environmental changes propagating through the pipeline.

Model attributes capture operational measures of the serving infrastructure itself, including inference latency, throughput, and service health (3/7 sessions; [S4–5;S7]). Teams set alerts directly on these attributes, where drops in throughput or anomalies in volume automatically trigger alarms – primarily intended to catch service failures in the serving infrastructure. Yet these signals also surface upstream faults cascading from an upstream source. As one practitioner discovered through a drop in model throughput [S5]:

> *"We have a monitor for [throughput], and it decreased significantly... I discovered that somewhere upstream, one of the features wasn't sent to us, it was empty."*

Prediction metrics emerged as the primary signal ML teams actively monitor (6/7 sessions; [S1–3;S5–7]), serving as the main, catch-all indicator of upstream issues and model health. Most teams track aggregate metrics that capture changes in the distribution of predictions over time [S1–3;S5–7], with some also actively tracking accuracy metrics when feedback is available – a metric computed of prediction and outcome metrics jointly [S3–4;S6].

Downstream from the model, teams track **action metrics** – the decisions actually executed by application policies that consume model predictions (4/7 sessions; [S1;S3–4;S7]). These outputs are produced by decision policies that take various forms – rule-based systems where model scores inform conditions [S1;S3;S6], workflows that aggregate multiple model outputs [S2;S7], or simple thresholds over model scores [S4–5;S7]. Practitioners report tracking both distribution divergence of executed actions [S3–4;S7] and the correctness of individual decision rules [S1]. Teams set alerts on action metrics to catch local application faults: outdated thresholds that fail to adapt to model improvements [S4;S7], or misconfigured rules that create technically valid but logically flawed decisions [S3]. As one practitioner explained [S3]:

> *"If the configuration doesn't make sense... or if there is a mistake, from technical point of view, it's a valid change, the system behaves normally, but logically the action wasn't correct."*

Yet as the final output point, action metrics effectively serve as a sink for issues originating anywhere in the system: data processing faults, model degradation, environmental shifts, and application logic errors all ultimately manifest in changed actions.

Tracking the correctness of actions involves cross-team collaboration, with business stakeholders who configure these policies actively monitoring alongside ML teams [S1;S3;S6]. As one described, "analysts are constantly eyeballing the time-series" of the final decisions [S3]. Practitioners viewed tracking actions and managing decision policies as integral to ML practice, with one noting [S1]:

> *"However many models we have, we probably have two orders of magnitude more rules, and I view a rule as a simple machine learning model."*

4.2.2 Missing instrumentation at system boundaries. Despite extensive instrumentation across the ML pipeline, practitioners reported that signals must comprehensively cover subsystems to be actionable – otherwise gaps in coverage prevent effective triage (4/7 sessions; [S1–3;S6]). When downstream signals fire but upstream signals are missing, teams cannot determine the cause – whether the alert indicates a critical failure or benign change. When upstream signals fire without corresponding downstream impact signals, teams cannot assess the impact – whether the detected issue actually matters for system operation.

The previous subsection illustrated this challenge: prediction metrics and action metrics frequently surfaced issues originating in data processing or the environment, yet without comprehensive upstream instrumentation, practitioners faced manual investigation to understand what triggered the downstream alert. As one practitioner emphasized the need for complete boundary coverage rather than selective instrumentation [S1]:

> *"We'd like to have [data quality tracked] at each interface, before going to the next."*

This gap proves especially challenging at organizational boundaries where different teams own different stages of the ML system. ML teams repeatedly encountered downstream anomalies caused by a fault in a subsystem managed by different teams – including corrupted source data from third parties, stale feature tables from failed data engineering jobs, or outdated configurations [S1–3;S6].

However, broad instrumentation coverage presents its own challenges. Teams reported that extensive monitoring produces many false positives [S3;S6] – detecting inconsequential changes that do not impact downstream predictions or actions, making it harder to identify critical issues. One team addresses this by composing signals and only treats an observed shift as operationally relevant when multiple signals shift jointly – e.g., the NULL rate of a feature jointly with prediction divergence, as explained in [S3]:

> *"To get your signaling in order is pretty hard because you've lots of false positives, but if you combine a couple of signals, you get a much cleaner signal."*

4.3 Localize a Fault Through DataFlows

When faults are detected, practitioners need to trace them back through their systems to identify where problems originated. This tracing depends on understanding how data and dependencies flow through the ML pipeline – connecting downstream effects to upstream causes. Yet most teams lack systematic representations of these flows, instead relying on manual investigation through complex, undocumented systems.

4.3.1 Graphical representations to systematically navigate dependencies. A common challenge across teams was navigating complex dependencies – especially pipelines with many intermediate steps and data loaded from different sources – when tracing a fault to its root-cause (5/7; [S1–3;S5–6]). As one practitioner noted [S1]:

> *"It's not obvious... how that final feature store came to be. There's all sorts of intermediate transformations which are dependencies*

on the final things. Many steps of transformations, many steps to create features, and features used to create other features."

To address this, some teams maintain **transformation graphs** – directed graphs showing how data flows from source tables through transformations to feature tables – which they use to systematically trace faults back to their origins (2/7 sessions; [S6–7]). When available, these graphs dramatically reduce diagnostic time. One team described quickly identifying which transformation caused a sample size drop [S7]:

"It was obvious from the graph, we saw a decline in the samples we're using... we look at the individual filters we apply... and then we saw that's the culprit."

The participants mentioned that these graphs are automatically derived using orchestration frameworks [S6–7] such as dbt [19] and Luigi [57].

4.3.2 Missing structural overviews force manual fault tracing. Despite the clear value of graphical representations, most teams reported lacking systematic visual overviews of their systems (5/7 sessions; [S1–5]). Instead, they rely on mental models and manual investigation – tracing system elements upstream from detected faults without structured guidance. This manual approach proves time-consuming and error-prone. As one participant described having *"a pretty busy two weeks figuring out what happened"* [S3], tracking down a fault that originated from a configuration change in source data ingestion and propagated through multiple pipeline stages before surfacing as an increase in false positive predictions.

The challenge extends beyond data processing alone. Teams reported complex interdependencies spanning multiple system layers [S1–3;S7]: models feeding into other models [S3;S7], decision policies aggregating outputs of multiple models [S2;S7], and cascading chains where policies consume model outputs, feed decisions into subsequent models, which then inform further policies [S3]. One practitioner captured the overwhelming scope of decision policies managed by different teams [S3]:

"There are rules here, everywhere, rules before us, rules afterwards, rules between us... I think we're at the stage where everyone realizes that we cannot move anymore like this."

While teams recognize the value of capturing structural overviews, it may not the first priority when immediate monitoring gaps demand attention [S1]. Furthermore, connecting subsystems managed by different data, modeling, and infrastructure teams across organizational boundaries complicates the aspiration of building a unified dependency view spanning the entire ML system [S3].

4.4 Identify Root Causes With Provenance

Upon receiving a fault signal, practitioners trace backwards through their systems to identify what changed upstream. Code deployments, configuration updates, or schema changes that coincide with detected anomalies become candidates for the root-cause. This backward tracing depends on provenance information – the metadata and operational attributes that record what underlies a fault.

4.4.1 Diagnostic information accessed during root-cause analysis. Teams reported investigating faults by examining artifacts across different system layers. **Transformation metadata** captures the code and configurations that transform data through the pipeline [S6–7]. Teams examine SQL scripts or Python transformations to identify logical faults – filters that exclude too much data, joins with incorrect conditions, or transformations that introduce errors [S6–7]. These errors often remain undetected for extended periods. As one practitioner discovered [S6]:

"We did a join, and we did some filters, and I think after a few months, we noticed that one of our filters excluded a quarter of our [data]."

Teams capture metadata artifacts to trace faults back to their origins. **Source table metadata** is examined to identify schema issues [S2–3]. **Model metadata** – like training scripts – were consulted to identify faults introduced during training that only surfaced in production [S4;S6]. **Policy metadata** is captured covering decision rule configurations enabling tracing faults to logical inconsistencies made by stakeholders [S3].

Beyond metadata, practitioners use operational measures to pinpoint fault location. **Transformation attributes** like throughput help identify which pipeline stages experienced issues [S6–7], while **feature table attributes** like staleness indicators help trace prediction faults to tables that failed to update [S2].

When tooling makes provenance information accessible within individual systems, investigation becomes straightforward. Databricks [16] exposed feature table staleness [S2], while Luigi [57] logs captured throughput of individual operations – enabling precise fault isolation [S7]. A gap emerges when tooling doesn't automatically surface these connections in response to downstream alerts.

4.4.2 Provenance disconnected from monitoring workflows. Like most software systems, provenance information is often captured by default. Version information exists and is tracked, but automatic correlation with monitoring signals does not. As one practitioner explained [S3]:

"You see the version, of course we're tracking it, but no one goes ahead and automatically correlates it, like there was a configuration before and then there's a configuration after, and immediately you see the change, this sharp distributional change."

The gap is not that provenance metadata does not exist – code is versioned, configurations are stored, schemas are tracked. The gap is that this metadata remains isolated from the monitoring signals that would make it actionable for root-cause analysis.

4.5 Verify Correctness with Domain Knowledge

Without observing the ground truth, practitioners must rely on domain knowledge to evaluate whether the data processing produces correct data and their models produce correct outputs. This knowledge exists along a spectrum – from formalized assertions that enable automated verification to tacit knowledge existing only in stakeholders' judgment.

4.5.1 Supervision through domain assertions. Some teams encode domain knowledge as explicit assertions – logical rules that define valid ranges and expected behaviors for feature and target variables – enabling automated detection of domain-violating outputs [S2;S6]. These assertions encode properties of the modeled domain – the external real-world process – rather than the internal system artifacts where they are applied.

Feature assertions test whether observed statistics align with known properties of feature variables in the modeled domain [S2;S6]. For example, an assertion that customer age must fall between 0 and 120 years encodes domain knowledge about real-world customers. This assertion serves as a verification rule against which observed metrics (such as min/max values) can be checked – both in feature tables after data processing and in feature vectors at serving time; they verify that processed data conform to expected real-world patterns and are indeed correctly processed and served.

Target assertions monitor whether prediction frequencies align with known properties of the target variable in the modeled domain [S2;S6]. One team noted they leverage their understanding that the positive class rate should remain stable (e.g, at 5%) to set alert thresholds [S6]. When predictions suddenly jump (e.g., to 20%), they recognize this as *"a clear signal that either something is wrong with our population or something is wrong with the model"* [S6].

4.5.2 Correctness criteria often remain tacit, existing only in stakeholder judgment. Many teams rely on human-in-the-loop verification through ad hoc consultation or *"an angry phone call"* [S3;S7], leaving domain knowledge tacit rather than codified as systematic checks (5/7 sessions; [S1;S3–4;S6–7]). Product teams and analysts informally monitor outputs, assessing correctness based on this domain expertise. As one practitioner noted [S6]:

> *"Our colleagues are our monitoring tool... whenever they see some unexpected output, we get notified."*

Some teams aspire to capture this tacit knowledge as more expressive assertions – moving beyond simple range checks to conditional rules that encode expected model behavior in specific scenarios [S4;S6]. As one team described their plans [S4]:

> *"You have a bunch of scenarios and from domain knowledge, you know what you want from a model, how a model should behave... that really tests if the model that is currently in production doing what I expect from it."*

The challenge is to systematically elicit this knowledge from stakeholders and to determine which assertions and scenarios warrant formalization – recognizing that exhaustive coverage of all possible cases remains fundamentally infeasible.

Beyond these practical challenges, correctness verification itself proved to be contested in all roles. Although domain expertise on target variables can be encoded as assertions to monitor correctness, practitioners reported tensions between technical and organizational definitions of correctness [S6–7]. Data scientists often assess whether predictions align with population-level probabilities – treating the target as a random variable with an underlying distribution [S7]. If a model's predictions reflect that distribution (e.g., a 10% base rate), they consider it correct. Business stakeholders, however, judge correctness against organizational requirements and expectations – such as maintaining a desired rate of actions triggered by a threshold – which may not align with the model's predictions.

4.6 Interpret Shifts Through Domain Context

Distribution shifts happen continuously, but distinguishing real problems from benign changes requires understanding the domain context. Practitioners slice metrics by customer segments to localize issues, but ultimately rely on manual consultation with analysts and domain experts to explain what events – seasonality, campaigns, disruptions – are driving the changes.

4.6.1 Domain context to expose subgroup behavior. Practitioners structure their monitoring around domain context by defining customer segments and tracking metrics separately for each subgroup – using **exogenous identifiers** and **feature slices** to partition their data (4/7 sessions; [S3;S5–7]).

Teams structure their understanding of the domain through two forms of segmentation. **Exogenous identifiers** assign observations to business-defined categories such as geography, customer maturity, or market segment (3/7 sessions; [S3;S5–6]). These key-value fields partition observations by external context. **Feature slices**, by contrast, partition the feature space by behavioral patterns within the data (1/7 sessions; [S7]). For example, "customer age" might be sliced into young (0-18), adult (18-65), and senior (65+) subgroups.

Both enable segment-level monitoring of instrumented metrics or business outcomes across countries, customer cohorts, or product categories [S3;S5–7]. This contextual understanding helps explain metric variations; different groups exhibit distinct behaviors and respond differently to external events in the environment – distribution shifts are often localized, requiring monitoring that captures these differences across subgroups. For example, a national holiday might affect sales in one country but not others; a sales campaign might impact new customers differently than established ones.

By partitioning data streams into semantically consistent groups, practitioners can explain issues that would otherwise be obscured in aggregate metrics. One practitioner explained the first step after detecting a shift in predictions [S3]:

> *"The output probabilities shifted as well. For some [subgroups], not so much for some... more false positives, and for others [more false negatives]. So there was like a bit of a mix of problems across [subgroups]."*

The fact that all subgroups exhibited changes – though with different symptoms – provided an early indication of an internal fault. Indeed, it turned out to be an outage of an upstream dependency.

4.6.2 Reliance on a human-in-the-loop instead of systematic contextualization. Practitioners easily detect distribution shifts but struggle to distinguish technical failures from benign distribution shifts [S1–4;S6]. As one practitioner described this ambiguity [S3]:

> *"It's not so hard to monitor when [metrics] go up or down. No. Tricky thing is, they naturally go up or down."*

Teams actively consult domain experts – typically analysts with real-time understanding of the domain – to explain why distributions shift [S1;S6]. These experts provide context about trends or events – like, marketing campaigns, or supply chain disruptions that *"get flagged by a business analyst"* [S6] – that only become visible as distribution shifts in the observed data. As one practitioner explained their dependency on analysts [S1]:

> *"Because [analysts] see the data live, they also pick up on new trends quickly. And we have mechanisms in place [to communicate this] back to us."*

This reliance on human judgment creates two critical challenges. First, without systematic capture of external context, teams cannot

automate their response – each alert requires manual consultation [S1;S6]. Second, without understanding whether shifts reflect transient or persistent changes, automated retraining risks overfitting to temporary patterns. As one practitioner described [S3]:

> *"What we often have is like [a transient event that] impacts our training data quite a bit, so the model is retrained, adapts to it, but then during inference, this pattern disappears. So performance goes down, because we're overfitting to some non causal pattern."*

Teams reported they lacked mechanisms to record this information; they viewed the external environment as *"a complex black-box system"* [S3] with no means to track events in real-time [S3–4]. One team recognized this limitation and aspired to capture the exogenous context in their monitoring workflow [S7]:

> *"I think in our monitoring we need to be aware on social cues and social news that would affect and help us explain to our stakeholders, why are the [metrics] high during those periods."*

5 Discussion

We characterized ML observability practices across organisations in various domains. Below, we answer the two research questions defined in our study design (Section 3) referring to current observability practices (RQ1) and identified gaps in observability (RQ2).

Answer to RQ1. Teams consistently capture information about both the ML system and external environment. They instrument data processing, model serving, and application layers to detect faults, and track outcome metrics when feasible. For fault investigation, teams capture artefact metadata and attributes for tracing, with some maintaining dependency graphs for systematic root-cause analysis. Teams represent domain understanding by defining meaningful subgroups to read signals in context and by implementing rule-based checks to confirm correctness.

Answer to RQ2. Instrumentation to detect faults remains incomplete, particularly across elements owned by different teams, while outcome metrics are often infeasible due to organizational constraints. With few documented dependencies beyond those inferred by orchestration, and provenance metadata detached from monitoring, root-cause analysis remains mostly manual. Interpretation depends predominantly on tacit knowledge – including external events that explain distribution shifts and domain expertise to verify correctness – which is often not systematically captured and therefore relies heavily on human-in-the-loop practices.

Taken together, these results motivate refinements to the descriptive model (Section 5.1) and implications for practice (Section 5.2); we also note limitations (Section 5.3), related work (Section 5.4), and future work (Section 5.5).

5.1 Reflection on Descriptive Model

The literature-derived model [37] effectively categorized most practitioners' monitoring information (Section 2), demonstrating adequate coverage and discriminant validity. Recurrent unmappable content showed systematic gaps in the original model. Following our framework analysis methodology (Section 3.4), we propose the refinements below based on the empirical results of our study.

Consolidate to simplified information types. As noted in Section 2.2, the descriptive model characterizes information through aspect (states, structures, or properties) and representation (formal or informal) dimensions. While these can combine in different ways – e.g., properties of a system element could be captured as logical assertions, probabilistic models, or textual descriptions – practitioners consistently used a constrained set of combinations. We recommend adopting these simplified categories (identifiers, slices, assertions, metadata, attributes, metrics, graphs) for operationalizing the model in practice – e.g., as a domain-specific language (DSL) – while preserving the full framework for academic purposes.

Add an endogenous outcome subsystem. As reported in Section 4.1, teams extensively track business outcomes and impact metrics – but the current descriptive model cannot represent the feedback loops where ML system actions influence the environment and generate these outcomes. We propose adding an endogenous outcome subsystem – as defined in Section 4 – within the environment view to capture how actions generate observable outcomes.

Add a feature vector system element. As discussed in Section 4.2, teams systematically monitor feature vectors at serving time separately from stored feature tables. The distinction matters operationally – serving-infrastructure failures surfaced only in vectors, not tables – and it mirrors typical ownership (ML engineers monitor served features; data engineers monitor feature tables. We propose adding a separate feature vector element to the inference pipeline subsystem, while representing produced features as a feature table element in the processing pipeline, for consistency with practice.

Remove the latent influences subsystem. When explaining the *system context* dimension, we noted persistent ambiguity between *latent* and *exogenous influences*. Both describe external drivers in the environment, and their only difference – being measured directly or inferred – is already handled by the *representation* dimension. Maintaining a separate subsystem therefore adds redundant complexity without improving discriminative value.

5.2 Implications

Our results have some critical implications for both tool development and future research directions, which we discuss here.

5.2.1 Implication for practice and tooling support. Current ML monitoring tools address different subsets of the system information catalogued in Table 2, creating fragmentation that likely contributes to the gaps we observed. ML observability platforms (Arize [6], Fiddler [23], WhyLabs [62]) capture model-centric signals – prediction metrics, feature statistics, and sliced performance analysis. Data observability platforms (Monte Carlo [43], Datadog [17]) capture pipeline information – table metrics, job attributes, and auto-derived dependency graphs for fault tracing. Cloud ML services (Vertex AI [28], SageMaker [2]) provide integrated serving metrics and drift detection over model inputs and predictions. Assertion frameworks (Great Expectations [30], deequ [3]) enable encoding domain constraints for automated verification across the pipeline.

This fragmentation likely contributes to the reported gaps in this study (Section 4). Diagnosing issues requires instrumentation and tracing across subsystems, yet tool support remains narrow. As argued in [37], the descriptive model could provide a conceptual foundation for modern ML monitoring tooling by organizing information in a graph structure connecting information to system elements and elements to each other. This study defines the scope ML monitoring tools need to cover for completeness: systematically

capturing information from the processing pipeline through the inference pipeline to downstream applications and the external environment – far broader than current tools address.

5.2.2 Implications for research. The descriptive model synthesized from prior literature places significant emphasis on capturing information about the external environment [37]. Practice shows limited reliance on this information and a narrower range of information types than the literature emphasizes. This does not invalidate the theoretical framework or prior research, but it shows that proposed capabilities remain largely aspirational. Understanding this gap offers important directions for future research.

For example, prior research has developed sophisticated methods for capturing domain knowledge. Some approaches model scenarios probabilistically, describing how known external events affect the data-generating process to enable proactive detection [36, 56]. Others propose textual playbooks and documentation frameworks that capture scenarios of external events and failure modes to support monitoring governance [20, 21, 31, 49]. Others advocate utilizing causal graphs of the feature and target variables – capturing the structure of the data-generating process, enabling systematic attribution of distribution shifts [22, 34, 64].

None of our participants implemented these capabilities in production, despite their demonstrated theoretical merit, suggesting a gap between research capabilities and practical adoption.

5.3 Limitations

Our study design involves several inherent constraints. First, focus groups capture reported practices rather than observing monitoring practice as conducted – some tacit knowledge or ad hoc practices may be underrepresented. Second, we did not triangulate focus group data with analysis of actual monitoring artifacts such as dashboards, alert configurations, or incident tickets – examining these artifacts might reveal practices not articulated during sessions. Finally, resource constraints limited engagement depth with each team – extended case studies might provide a more complete characterization of observability not surfaced in our sessions.

5.4 Related Work

Software observability practices are well-established – interview studies confirm traditional monitoring frameworks effectively support practitioners despite remaining challenges [18, 38, 46]. ML observability, however, is less mature, with exploratory and visionary attempting to establish the practice [47, 48, 54].

Several empirical studies have examined ML monitoring from different perspectives. Interview studies in the broader MLOps space consistently surface monitoring as a significant challenge [5, 44, 53], while more focused interview work specifically examines ML monitoring practices [51, 55]. In similar, work, a questionnaire-based study confirms that tool support remains limited and no established framework for ML observability has emerged [67]. Multivocal literature reviews synthesizing practitioner discussions identify key guidelines and challenges, including the need to track comprehensive metrics and understand the causes behind observed distributional changes [45, 51]. Related surveys on trustworthy AI highlight monitoring challenges around fairness and explainability [7, 15].

Experience reports and case studies shaped most of our understanding of ML monitoring and observability practice. Initial experiences from practice identified monitoring and observability challenges unique to ML that informed subsequent research [52]. Case studies document operational ML systems [1, 4], highlighting both challenges and workarounds practitioners employ. Maturity assessments show that less mature organizations rely predominantly on manual monitoring, with automated practices emerging as organizations advance [33]. Case studies specific to ML monitoring yielded results on particular aspects like continuous data validation [39], and drift detection frameworks in practice [42, 63, 66].

Our work addresses a gap in this body of research. Although prior studies have classified monitoring challenges and documented practitioner pain points [44, 45, 51, 53, 55, 67], none has systematically characterized what information practitioners capture for ML monitoring. We provide the first holistic empirical characterization of ML observability practices – beyond documenting isolated challenges and best practices.

5.5 Future Work

Our findings need validation across broader contexts. This study provides an initial empirical characterization of ML observability practice through focus groups across organizations. In-depth case studies are needed to validate and extend our findings – particularly to examine the cross-role nature of ML monitoring we observed. Our results showed involvement across stakeholders: data engineers tracking pipeline quality, ML teams monitoring predictions, analysts verifying decisions, and domain experts interpreting shifts. How these roles monitor ML systems and capture information to enable observability warrants future research.

Beyond validation, future research should characterize patterns of system context information needed to detect and diagnose specific failure modes. While we catalogued what practitioners capture broadly, understanding which information combinations enable effective detection of particular faults – data corruption, model degradation, policy misconfiguration – remains unknown. Empirical studies could identify these patterns and evaluate whether implementing the corresponding capture mechanisms (automated provenance correlation, dependency graphs, formalized domain knowledge) improves monitoring effectiveness in practice.

To translate validated practices into practical tooling, an avenue for future work is to develop observability tooling that operationalizes the information surfaced here (e.g., through a DSL [35]).

6 Conclusion

This study presents the first empirical characterization of ML observability practices through focus groups with 37 practitioners in 7 organizations. We classified the information teams systematically capture across both the ML system – signals spanning processing pipelines, inference pipelines, and applications – and the external environment – customer segments, domain constraints, and outcome metrics. Our results validate and refine a descriptive model for ML observability, confirming that effective monitoring requires capturing information about both system views. As such, we contribute a sound basis for developing practical ML observability.

References

[1] Klaus Ackermann, Joe Walsh, Adolfo De Unánue, Hareem Naveed, Andrea Navarrete Rivera, Sun-Joo Lee, Jason Bennett, Michael Defoe, Crystal Cody, Lauren Haynes, et al. 2018. Deploying machine learning models for public policy: A framework. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 15–22.

[2] Amazon Web Services. 2025. Amazon SageMaker. https://aws.amazon.com/sagemaker. Accessed 2025-10-09.

[3] Amazon Web Services. 2025. Deequ: Unit Tests for Data. https://github.com/awslabs/deequ. Accessed 2025-10-09.

[4] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[5] Chintan Amrit and Ashwini Kolar Narayanappa. 2025. An analysis of the challenges in the adoption of MLOps. *Journal of Innovation & Knowledge* 10, 1 (2025), 100637.

[6] Arize AI. 2025. Arize AI ML Observability Platform. https://arize.com Accessed 9 Oct 2025.

[7] Firas Bayram and Bestoun S Ahmed. 2025. Towards trustworthy machine learning in production: An overview of the robustness in mlops approach. *Comput. Surveys* 57, 5 (2025), 1–35.

[8] Michele Bennett, Jaya Balusu, Karin Hayes, and Ewa J Kleczyk. 2022. The Silent Problem–Machine Learning Model Failure–How to Diagnose and Fix Ailing Machine Learning Models. *arXiv preprint arXiv:2204.10227* (2022).

[9] Betsy Beyer, Niall Richard Murphy, David K Rensin, Kent Kawahara, and Stephen Thorne. 2018. *The site reliability workbook: practical ways to implement SRE.* " O'Reilly Media, Inc.".

[10] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D Sculley. 2017. The ML test score: A rubric for ML production readiness and technical debt reduction. In *2017 IEEE international conference on big data (big data)*. IEEE.

[11] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Whang, and Martin Zinkevich. 2019. Data Validation for Machine Learning.. In *MLSys*.

[12] Till J Bungert, Levin Kobelke, and Paul F Jaeger. 2023. Understanding silent failures in medical image classification. In *International conference on medical image computing and computer-assisted intervention*. Springer, 400–410.

[13] Amber G Candela. 2019. Exploring the function of member checking. *The qualitative report* 24, 3 (2019), 619–628.

[14] Bryan Cantrill. 2006. Hidden in Plain Sight: Improvements in the observability of software can help you diagnose your most crippling performance problems. *Queue* 4, 1 (2006), 26–36.

[15] Zhenpeng Chen, Jie M Zhang, Max Hort, Mark Harman, and Federica Sarro. 2024. Fairness testing: A comprehensive survey and analysis of trends. *ACM Transactions on Software Engineering and Methodology* 33, 5 (2024), 1–59.

[16] Databricks. 2013. Databricks Lakehouse Platform. https://www.databricks.com/.

[17] Datadog. 2025. Datadog Monitoring and Analytics Platform. https://www.datadoghq.com Accessed 9 Oct 2025.

[18] Thomas Davidson, Emily Wall, and Jonathan Mace. 2023. A qualitative interview study of distributed tracing visualisation: A characterisation of challenges and opportunities. *IEEE Transactions on Visualization and Computer Graphics* 30, 7 (2023), 3828–3840.

[19] dbt Labs. 2016. dbt (data build tool). https://www.getdbt.com/.

[20] Paul-Arthur Dreyfus, Antoine Pélissier, Foivos Psarommatis, and Dimitris Kiritsis. 2022. Data-based model maintenance in the era of industry 4.0: A methodology. *Journal of Manufacturing Systems* 63 (2022).

[21] Jean Feng, Rachael V Phillips, Ivana Malenica, Andrew Bishara, Alan E Hubbard, Leo A Celi, and Romain Pirracchio. 2022. Clinical artificial intelligence quality improvement: towards continual monitoring and updating of AI algorithms in healthcare. *NPJ digital medicine* 5, 1 (2022), 66.

[22] Jean Feng, Adarsh Subbaswamy, Alexej Gossmann, Harvineet Singh, Berkman Sahiner, Mi-Ok Kim, Gene Anthony Pennello, Nicholas Petrick, Romain Pirracchio, and Fan Xia. 2024. Designing monitoring strategies for deployed machine learning algorithms: navigating performativity through a causal lens. In *Causal Learning and Reasoning*. PMLR, 587–608.

[23] Fiddler AI. 2025. Fiddler AI Model Performance Management. https://www.fiddler.ai Accessed 9 Oct 2025.

[24] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

[25] Radoslav Gatev. 2021. Observability: Logs, metrics, and traces. In *Introducing distributed application runtime (dapr) simplifying microservices applications development through proven and reusable patterns and practices*. Springer, 233–252.

[26] Tony Ginart, Martin Jinye Zhang, and James Zou. 2022. Mldemon: Deployment monitoring for machine learning systems. In *International conference on artificial intelligence and statistics*. PMLR, 3962–3997.

[27] Laurie J Goldsmith. 2021. Using Framework Analysis in Applied Qualitative Research. *The Qualitative Report* 26, 6 (2021), 2061–2076.

[28] Google Cloud. 2025. Vertex AI. https://cloud.google.com/vertex-ai Accessed 9 Oct 2025.

[29] Ian Gorton, Liz Fong-Jones, and Alf Larsson. 2023. Observability q&a. *IEEE Software* 41, 1 (2023), 50–54.

[30] Great Expectations. 2025. Great Expectations Data Validation Framework. https://greatexpectations.io Accessed 9 Oct 2025.

[31] Tom Haider, Felippe Schmoeller Roza, Dirk Eilers, Karsten Roscher, and Stephan Günnemann. 2021. Domain Shifts in Reinforcement Learning: Identifying Disturbances in Environments.. In *AISafety@ IJCAI*.

[32] Chip Huyen. 2022. *Designing machine learning systems.* " O'Reilly Media, Inc.".

[33] Meenu Mary John, Helena Holmström Olsson, and Jan Bosch. 2021. Towards mlops: A framework and maturity model. In *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 1–8.

[34] Budhathoki Kailash, Janzing Dominik, Blöbaum Patrick, and Ng Hoiyi. 2021. Why did the distribution change?. In *Proceedings of Machine Learning Research*.

[35] Panagiotis Kourouklidis, Dimitris Kolovos, Joost Noppen, and Nicholas Matragkas. 2023. A domain-specific language for monitoring ML model performance. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 266–275.

[36] Joran Leest, Claudia Raibulet, Ilias Gerostathopoulos, and Patricia Lago. 2024. Expert Monitoring: Human-Centered Concept Drift Detection in Machine Learning Operations. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 1–5.

[37] Joran Leest, Claudia Raibulet, Patricia Lago, and Ilias Gerostathopoulos. 2025. From Tea Leaves to System Maps: A Survey and Framework on Context-aware Machine Learning Monitoring. *IEEE Transactions on Software Engineering* (2025).

[38] Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. 2022. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering* 27, 1 (2022), 25.

[39] Lucy Ellen Lwakatare, Ellinor Rånge, Ivica Crnkovic, and Jan Bosch. 2021. On the experiences of adopting automated data validation in an industrial machine learning project. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE.

[40] Michael R Lyu. 2007. Software reliability engineering: A roadmap. In *Future of Software Engineering (FOSE'07)*. IEEE, 153–170.

[41] Ankur Mahida. 2023. Enhancing Observability in Distributed Systems-A Comprehensive Review. *Journal Of Mathematical & Computer Applications. Src/Jmca-166. Doi: Doi. Org/10.47363/Jmca/2023 (2)* 135 (2023), 2–4.

[42] Ankur Mallick, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. 2022. Matchmaker: Data drift mitigation in machine learning for large-scale systems. *Proceedings of Machine Learning and Systems* 4 (2022), 77–94.

[43] Monte Carlo Data. 2025. Monte Carlo Data Observability Platform. https://www.montecarlodata.com Accessed 9 Oct 2025.

[44] Dennis Muiruri, Lucy Ellen Lwakatare, Jukka K Nurminen, and Tommi Mikkonen. 2022. Practices and infrastructures for machine learning systems: An interview study in finnish organizations. *Computer* 55, 6 (2022).

[45] Hira Naveed, Scott Barnett, Chetan Arora, John Grundy, Hourieh Khalajzadeh, and Omar Haggag. 2025. Monitoring Machine Learning Systems: A Multivocal Literature Review. *arXiv preprint arXiv:2509.14294* (2025).

[46] Sina Niedermaier, Falko Koetter, Andreas Freymann, and Stefan Wagner. 2019. On observability and monitoring of distributed systems–an industry interview study. In *International Conference on Service-Oriented Computing*. Springer, 36–52.

[47] Diego Nogare, Ismar Frango Silveira, Pedro Pinheiro Cabral, Rafael Jorge Hauy, and Veronica Neves. 2024. Machine Learning Model: Perspectives for quality, observability, risk and continuous monitoring. In *Congresso Latino-Americano de Software Livre e Tecnologias Abertas (Latinoware)*. SBC, 181–187.

[48] Guilherme Palumbo, Davide Carneiro, and Victor Alves. 2023. Observability: Towards Ethical Artificial Intelligence. In *International Conference on Disruptive Technologies, Tech Ethics and Artificial Intelligence*. Springer, 233–236.

[49] Abhishek Paul, Roderick Y Son, Shiv A Balodi, and Kenney Crooks. 2024. MLOps FMEA: A proactive & structured approach to mitigate failures and ensure success for machine learning operations. In *2024 Annual Reliability and Maintainability Symposium (RAMS)*. IEEE.

[50] Andy Petrella. 2023. *Fundamentals of Data Observability.* " O'Reilly Media, Inc.".

[51] Dominik Protschky, Luis Lämmermann, Peter Hofmann, and Nils Urbach. 2025. What Gets Measured Gets Improved: Monitoring Machine Learning Applications in Their Production Environments. *IEEE Access* (2025).

[52] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *Advances in neural information processing systems* 28 (2015).

[53] Shreya Shankar, Rolando Garcia, Joseph M Hellerstein, and Aditya G Parameswaran. 2024. " We Have No Idea How Models will Behave in Production until Production": How Engineers Operationalize Machine Learning. *Proceedings of the ACM on Human-Computer Interaction* 8, CSCW1, 1–34.

[54] Shreya Shankar and Aditya G Parameswaran. 2022. Towards Observability for Machine Learning Pipelines.. In *CIDR*.
[55] Murtuza N Shergadwala, Himabindu Lakkaraju, and Krishnaram Kenthapadi. 2022. A human-centric perspective on model monitoring. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, Vol. 10. 173–183.
[56] Piotr Sobolewski and Michał Woźniak. 2017. SCR: simulated concept recurrence–a non-supervised tool for dealing with shifting concept. *Expert Systems* 34, 5 (2017).
[57] Spotify. 2012. Luigi: A Python Framework for Building Data Pipelines. https://luigi.readthedocs.io/. Documentation.
[58] Arun Swami, Sriram Vasudevan, and Joojay Huyn. 2020. Data sentinel: A declarative production-scale data validation platform. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1579–1590.
[59] The Institute for Ethical AI & Machine Learning. 2024. The State of Production ML in 2024. https://ethical.institute/state-of-ml-2024. Accessed 2025-10-09.
[60] Muhammad Usman, Simone Ferlin, Anna Brunstrom, and Javid Taheri. 2022. A survey on observability of distributed edge & container-based microservices. *IEEE Access* 10 (2022), 86904–86919.
[61] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. Characterizing concept drift. *Data Mining and Knowledge Discovery* 30, 4 (2016), 964–994.
[62] WhyLabs. 2025. WhyLabs AI Observability Platform. https://whylabs.ai Accessed 9 Oct 2025.
[63] Zhentao Xu, Ruoying Wang, Girish Balaji, Manas Bundele, Xiaofei Liu, Leo Liu, and Tie Wang. 2023. Alertiger: Deep learning for ai model health monitoring at linkedin. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.
[64] Haoran Zhang, Harvineet Singh, Marzyeh Ghassemi, and Shalmali Joshi. 2023. "Why did the Model Fail?": Attributing Model Performance Changes to Distribution Shifts. In *Proceedings of Machine Learning Research*.
[65] Shihao Zheng, Simon B van der Zon, Mykola Pechenizkiy, Cassio P de Campos, Werner van Ipenburg, Hennie de Harder, and Rabobank Nederland. 2019. Labelless concept drift detection and explanation. In *NeurIPS 2019 Workshop on Robust AI in Financial Services: Data, Fairness, Explainability, Trustworthiness, and Privacy*.
[66] Xianzhe Zhou, Wally Lo Faro, Xiaoying Zhang, and Ravi Santosh Arvapally. 2019. A framework to monitor machine learning systems using concept drift detection. (2019), 218–231.
[67] Eduardo Zimelewicz, Marcos Kalinowski, Daniel Mendez, Görkem Giray, Antonio Pedro Santos Alves, Niklas Lavesson, Kelly Azevedo, Hugo Villamizar, Tatiana Escovedo, Helio Lopes, et al. 2024. Ml-enabled systems model deployment and monitoring: Status quo and problems. In *International Conference on Software Quality*. Springer, 112–131.