Explaining Shifts in Machine Learning Systems with Causal Maps

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

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

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

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

Abstract

Monitoring machine learning (ML) systems is hard, with standard practice focusing on detecting distribution shifts rather than their causes. Root-cause analysis often relies on manual tracing to determine whether a shift is caused by software faults, data-quality issues, or natural change. We propose *ML System Maps* – causal maps that, through layered views, make explicit the propagation paths between the environment and the ML system's internals, enabling systematic attribution of distribution shifts. We outline the approach and a research agenda for its development and evaluation.

Keywords

machine learning, root-cause analysis, distribution shift, causality

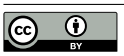
ACM Reference Format:
Joran Leest, Ilias Gerostathopoulos, Claudia Raibulet, and Patricia Lago. 2026. Explaining Shifts in Machine Learning Systems with Causal Maps. In *2026 IEEE/ACM 5th International Conference on AI Engineering - Software Engineering for AI (CAIN '26), April 12–13, 2026, Rio de Janeiro, Brazil*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3793653.3793782

1 Introduction

Machine learning (ML) systems contain data-driven components whose logic cannot be fully inspected or understood [29]. Their behavior depends on many upstream factors – both internal data processing (feature engineering, transformations) and external data-generating processes [2, 14, 21, 33]. Distribution shifts in any of these dependencies can cause the model to stop generalizing, deteriorating the decisions it drives [2, 23].

In operational ML systems, distributional shifts serve as the primary monitoring signal, indicating that *something* changed but not *why* [14, 31, 33, 36, 43]. The root cause might be data quality issues [3, 10], deployment bugs [14, 33], changing user behavior [18, 48], or even the system's own actions [11, 35]. Each requires different responses – e.g., reverting a pipeline change, or model retraining – yet shifts appear as undifferentiated symptoms.

Effective monitoring thus requires solving an *attribution* problem: tracing an observed shift from symptom to source – determining whether a shift originates locally, is inherited from upstream dependencies, or reflects a natural change [27, 32, 34]. This often requires extensive manual investigation [20]; a process that can be understood as answering three attribution questions (AQs).



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/10.1145/3793653.3793782

AQ1: ***Which subsystem is responsible for the shift, and which team should it be escalated to?*** ML systems span organizational boundaries – data teams maintain pipelines, ML teams manage models, business teams consume predictions [17]. Tight coupling causes shifts to propagate downstream, creating symptoms in multiple subsystems [2]; attribution first requires determining whether a shift originates locally or was inherited upstream.

AQ2: ***Which component within the identified subsystem could have introduced the shift?*** Once the responsible subsystem is identified, the owning team must first confirm whether the shift originated locally and, if so, isolate potential faults (e.g., a stale table, code change, or faulty configuration). This is difficult because subsystems comprise many interdependent components [34].

AQ3: ***If the shift is not caused internally, could a natural change in the underlying process explain it?*** Not every shift indicates a problem – ML systems model external processes that continuously change [7, 18]. If a shift stems from changes in external processes, we can rule out internal faults (i.e., "explain it away" [41]) – e.g., a shift might simply match a recurring pattern. Even in this case, the exact source needs to be understood to separate benign shifts from those that invalidate modeling assumptions [14, 40].

We envision a systematic approach to answering these questions automatically through *ML System Maps* – a hierarchically structured representation of the cause-and-effect relationships within an ML system and its environment. ML System Maps build on the conceptual framework of [21] and draw on causal inference methods [16, 26, 28, 37]. They enable systematic attribution through three coordinated views, each answering one AQ: the *ML System View* (AQ1), *Subsystem View* (AQ2), and *Environment View* (AQ3). We develop this vision by first characterizing operational ML systems and their structure (Section 2), then we introduce our approach (Section 3), illustrating its attribution capabilities (Section 4), and discuss practical considerations and a research agenda (Section 5).

2 The Structure of an ML system

To analyze how changes propagate through an ML system, we first need to understand how ML systems are structured. In this section, we characterize the structure of ML systems and establish key concepts for systematic representation. We distinguish the **internal ML system** (managed subsystems) from the **environment** (data-generating processes) [21]. Throughout the paper, we use a *running example* to illustrate the concepts and our approach;

> **Running example.** *A company uses an ML system to reduce customer churn through targeted promotions. The system processes user features through a churn risk model and a promotion-ranking model. These predictions inform two sequential decisions: identifying at-risk customers, and selecting promotional offers.*

2.1 Internal ML System

The internal ML system comprises three subsystems through which data flows, each with a canonical responsible stakeholder, and each can introduce distribution shifts [21]:

Data Pipeline. Maintained by *data engineers*, this subsystem involves the extract-transform-load (ETL) processes that ingest source data, apply transformations through aggregations and joins, and load the transformed data into feature tables. Tools such as dbt [9] represent pipelines as directed acyclic graphs (DAGs), with transformations and artefacts as nodes and dependencies as edges. In processing the data, distribution shifts can be introduced through scheduled updates (e.g., schema changes) or faults (e.g., failed jobs) – causing data quality issues [33].
Example: *The pipeline parses raw user activity logs into daily activity counts, then computes rolling averages and other aggregations to produce features for the churn model. Parsing errors in the raw logs lead to under-represented activity, while failed jobs produce stale features – both introducing data quality issues.*

Model Serving. Managed by *ML engineers*, this subsystem involves the serving pipeline that loads features from tables, applies pre-processing, runs inference through ML models, and produces predictions. Frameworks like TensorFlow Extended [13] represent these serving flows as pipeline graphs, tracking model versions and serving configurations. Distribution shifts can arise from intentional changes or faults, potentially compromising predictions and propagating to dependent models (often managed by different teams) [2, 31].
Example: *The serving pipeline loads demographic and activity features, runs inference through the churn risk model to generate predictions, which are then consumed by the promotion-ranking model. Routing traffic to different model versions (e.g., during experiments) or timeouts in fetching features can shift churn predictions and propagate to the dependent model.*

Application. Managed by *business stakeholders*, applications combine predictions with business rules to produce business decisions. These range from simple rule-based logic (e.g., threshold-based decisions) to complex workflows (e.g., Salesforce [30] or Zapier [44] automations). Applications can cause distribution shift via selection effects, where decisions drive actions that determine which cases downstream models observe – and the data they are trained on [12, 35].
Example: *The retention team uses churn predictions combined with business rules (budget constraints) to make outreach decisions. A separate promotions team then selects which offer to send, based on ranked promotions. A change in outreach policy risks exposing the promotion-ranking model to a shifted user population.*

Having described these subsystems as data flows, we leverage the insight of [26] that such flows can be translated into a cause-and-effect diagram (or causal graph) by representing the outputs of processes as causally connected nodes. The principle is straightforward: if a downstream variable is derived from upstream variables, it is causally determined by them.

2.2 Environment

Since ML systems are data-driven, their behavior is fundamentally *conditioned on* the environment they observe and act upon, not just internally managed processes. Without accounting for these changing external data-generating processes, shift attribution remains incomplete. Following [21], we treat the environment as an integral system to consider jointly with the internal ML system.

Environment. The real world that generates data for the ML system and is affected by its decisions. Changes in the environment arise from trends and seasonality (e.g., holiday patterns) or from unexpected events (i.e., exogenous shocks) like logistics disruptions or service outages [7, 40]. Since these changes occur before measurement, they propagate through the ML system (from data processing to application) and can resemble internal faults [34, 43]. While teams do not control the environment, *data scientists* typically track trends and events to ensure modeling assumptions remain valid [45]. **Example**: *The environment represents the customer churn process – how behavior leads to churn outcomes – which the ML system observes and influences through promotions.*

Like the internal ML system, the environment has a cause-and-effect structure, such as user demographics shaping behavior. While reasoning over this structure is common (though often left implicit) in model development [19], explicitly formalizing this structure enables tracing shifts to their source – e.g., a shift in observed user demographics explaining changes in their activity patterns [16].

3 ML System Maps

Building on this causal framing, we introduce our approach *ML System Maps*: a method for modeling the causal structure of a system, and using it for explaining distribution shifts. The causal structure is specified by a *notation* (Section 3.1) and a set of *views* (Section 3.2) organized to systematically answer the three attribution questions mentioned in Section 1. Figure 1 shows an *ML System Map* for the customer retention example, expressed in the notation and views.

3.1 Notation

The notation defines variable types for system quantities and their relations, forming the vocabulary used in our method:

- **Data variables** are populated within the system through managed processes. They represent outputs produced by each step of the data flow – e.g., features to predictions.
- **Modulator variables** condition how output data are produced from their input data – including configurations (e.g., model version), and operational status (e.g., data freshness).
- **Random variables** denote real-world phenomena that the system observes and/or influences. They remain conceptually distinct from data variables and are not directly observed, but accessed through proxies – e.g., user activity is observed through activity features.
- **Causal relations** denote functional determination between variables. They occur as *data to data* (e.g., features to predictions), *modulator to data* (e.g., model version to predictions), and *random to random* (e.g., user demographics to activity).

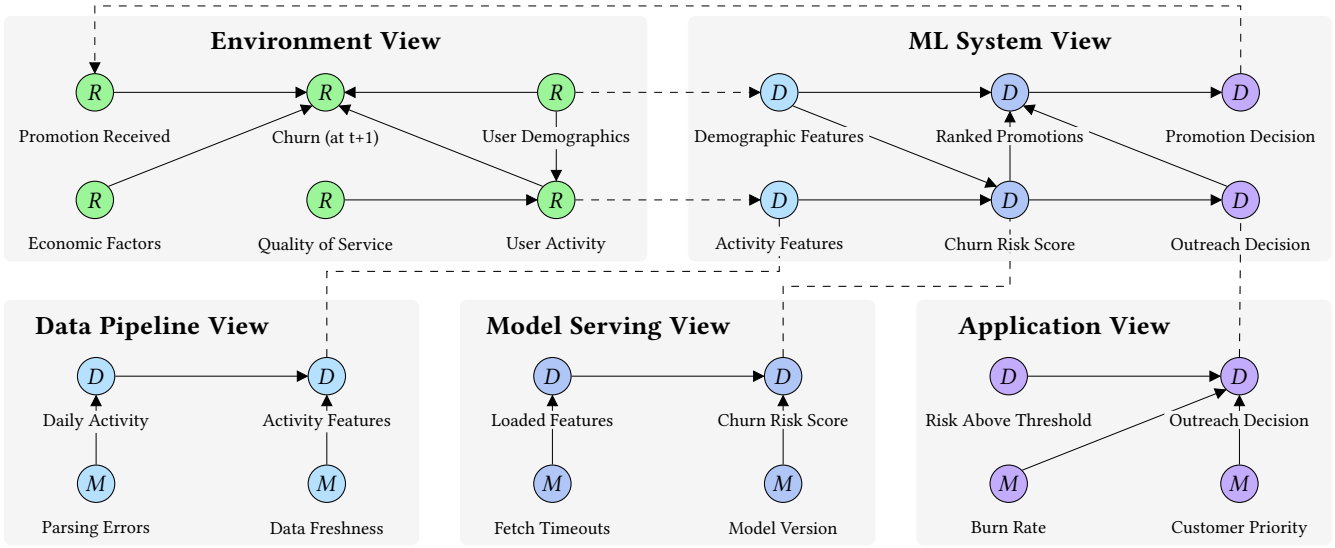


Figure 1: Illustrative ML System Map composed of five views: *Environment*, *ML System*, and three *Subsystem views* (*data pipeline* for activity features, *model serving* for churn, *application* for outreach). Nodes represent variables: data (D), modulators (M), and random (R). Edges denote relations: causal (solid arrows), mapping (dashed lines), and causal mapping (dashed arrows).

- **Mapping relations** denote that two data variables represent the same quantity across views (explained in Section 3.2) – e.g., the churn risk score (ML System View) maps to the terminal data variable (Model Serving View)
- **Causal mapping relations** connect what the environment produces (random variables) with what the internal system computes (data variables). Unlike mapping relations – which align quantities across views – they assert a cross-view causal link and treat the data variable as a proxy for a random variable. In the *random to data* direction, an environmental quantity is captured and processed into an observed data variable that serves as its proxy (e.g., user activity to activity features). In the *data to random* direction, a system data variable brings about an environmental outcome (e.g., outreach decision to promotion received). The relation is not identity – the two nodes are distinct, linked by measurement or actuation – and it does not assert independence – any upstream dependence on the source side (e.g., user demographics influencing activity) remains.

3.2 Views

We compose variables and relations into three types of views that systematically address the three attribution questions: the *ML System View* (AQ1), *Subsystem views* (AQ2), and the *Environment View* (AQ3). Each data variable in the ML System View represents the output of a subsystem that could be expanded into its own Subsystem view (Figure 1 shows three such views). The views follow two principles: *sparsity* – each view shows only the variables and relations necessary to answer its specific attribution question, focusing attention on relevant factors and also making causal inference tractable [16, 38]; and *causal consistency* – causal relationships are preserved both within each view and when tracing across views.

The **ML System View** contains only the terminal data variables from each internal subsystem – the contribution of a subsystem, and thus team, to the ML system's function. This view abstracts away all intermediate data and modulator variables within the subsystems, capturing only the high-level dependencies between them. When we observe drift in a downstream variable, such as final outreach decisions, this sparse structure allows us to identify which upstream subsystem may be responsible (**AQ1**).

A **Subsystem View** exposes the internal structure of a specific subsystem. Each such view includes all intermediate data variables produced along the subsystem's processing pipeline, from initial inputs to the terminal data variables. For example, the Data Pipeline View shows how raw activity counts are transformed into final activity features, with modulator variables for parsing quality and data freshness governing the transformation steps. The terminal data variable is connected via direct mapping to the ML System View above. This structure enables locating which specific component within a subsystem introduced a fault (**AQ2**).

The **Environment View** structures random variables according to causal relationships in the underlying data-generating process – e.g., how user demographics affect activity. Data variables connect to random variables through causal mappings and enable reasoning about environmental quantities when the proxy is valid. For example, activity features reflect actual user activity only if data quality is maintained; outreach decisions translate to promotions received only if delivery succeeds. The view may also include external variables outside the feature set – such as those omitted during feature selection [19] – to explain shifts that features cannot account for (e.g., quality of service or economic factors) [21]. When proxies are valid and the causal structure is modeled, we can determine whether shifts originate from natural external changes (**AQ3**).

4 Explaining Distribution Shifts

We now turn to the use of *ML System Maps* for attributing shifts. We first define core concepts and then describe view traversal.

4.1 Attribution and Shift Propagation

A distribution shift is a change in the properties of data between current and reference observation windows (e.g., this week vs. last week), measured through divergence metrics [14]. The attribution problem is to identify the source of the shift [16]. This requires understanding which variable(s)' mechanisms changed.

The intuition behind attribution follows from the causal graph, where each variable is generated through a *mechanism*. **Marginal mechanisms** are the distributions $P(X_j)$ of root variables (those with no parents). For example, $P(\text{Demographics})$ captures how demographic characteristics vary in the user population. **Causal mechanisms** are the conditional distributions $P(X_j|PA_j)$ that define how a variable depends on its direct causes PA_j. For example, $P(\text{Activity}|\text{Demographics})$ captures how user activity depends on demographics through some underlying mechanism.

When a variable shifts, it must be explained by a change in one of these mechanisms. A **change in marginal mechanism** occurs when a root variable's distribution changes and propagates through stable causal mechanisms. For example, if activity shifts while demographics also shift, and $P(\text{Activity}|\text{Demographics})$ remains stable, the activity shift is explained by the demographic change (e.g., we are just observing more young users) propagating through an unchanged mechanism, tracing back to $P(\text{Demographics})$.

A **change in causal mechanism** explains a shift when the conditional $P(X_j|PA_j)$ itself has changed – intuitively, this occurs when a variable changes despite its direct causes remaining stable. For example, activity features shifting but demographics remain stable, indicating that some hidden confounding factor caused the shift [48]. Similarly, computational changes are causal mechanism changes – e.g., if an aggregation job fails and produces NULLs, the mechanism transforming raw data into features has changed [26].

While ML System Maps remain agnostic to the attribution algorithm used, various attribution methods [7, 16, 26, 37] can identify which mechanism is responsible for a shift. For example, attribution can be performed with Shapley values, as in [16]. Their method models the mechanisms – marginal and causal, approximated using empirical and conditional models, respectively. It then swaps mechanisms between windows: for a subset of variables, it uses their current mechanisms while keeping others from the reference window, calculating each mechanism's contribution to the shift. High scores on a variable indicate that its mechanism changed – marginal (for root variables) or causal (for variables with parents).

4.2 Attribution over ML System Maps

When a shift alert fires, we traverse the ML System Map to trace the shift from symptom to source, answering one AQ per view. Within each view, we identify attribution patterns (APs) – outcomes where attribution mass (e.g., Shapley values) concentrates on specific variables. Figure 2 illustrates this process with scenarios where a promotion ranking shift traces to different explanations (for brevity, only the Data Pipeline View is shown for AQ2; other subsystem views follow analogously). We proceed as follows:

AQ1 – Route. Alerts are defined over *data variables* in *the ML System View*. Routing attributes to the subsystem whose *terminal* data variable best explains a shift; it answers *where*, not *why*. For example, when the ML team managing the promotion-ranking model observes a shift in the rankings, routing on the *ML System View* potentially assigns responsibility upstream. Two APs apply:

AP1.1 Subsystem Isolated. Mass concentrates on a *conditional data variable*. Upstream data feeding this variable is stable, so the shift is local to its subsystem. Route to the corresponding subsystem (*AQ2*) – e.g., mass on outreach decisions isolates the outreach application, a policy change might have exposed the model to a different user population.

AP1.2 Isolated at Boundary. Mass concentrates on a *root data variable* (i.e., features), not a conditional. We first route to the corresponding subsystem that produced the data (*AQ2*). However, the cause may also be in the environment (*AQ3*) – e.g., mass on activity features can reflect pipeline issues, schema or logic changes, or a natural behavioral shift.

AQ2 – Localize. We open the implicated *Subsystem View* to localize the possible cause of the shift. For example, if mass concentrated on activity features (*AP1.2*), we open the *Data Pipeline View* and attribute to potential faults. The following APs apply:

AP2.1 Root Cause Localized. Mass concentrating on a *modulator* indicates the shift was caused by a change in how a function within the subsystem produces data – e.g., attribution pointing to parsing errors that affected the activity events prior to computing user activity features. The root cause is identified and the procedure terminates.

AP2.2 Component Localized. Mass concentrates on an internal *data variable* while no modulator carries mass. The shift is internal to the subsystem – e.g., mass on activity features indicates failed jobs or logic changes, but manual investigation is required to identify the root cause.

AP2.3 Localized at Boundary. Mass concentrates on the *first data variable* at the *data pipeline's* input boundary (e.g., daily activity counts). This boundary is ambiguous between internal issues and genuine external changes. Proceed to *AQ3* to test if external factors explain the shift.

AQ3 – Externalize. When attribution mass concentrate at the data pipeline's input boundary (AP2.3), we turn to the *Environment View*. Having ruled out internal causes, we return to the feature-level variable that routes to this pipeline (e.g., user activity features), treating it as a proxy for its corresponding random variable to test whether external factors explain the shift. The following APs apply:

AP3.1 Explained Externally. Mass concentrates on an *upstream random variable* (e.g., quality of service) consistent with a known causal relation. The shift reflects external change; assess whether it's transient (e.g., a service outage that is already resolved) or requires model retraining if persistent.

AP3.2 Cannot Determine. Mass concentrates on the implicated random variable itself (e.g., user activity) rather than its parents, indicating the shift cannot be explained by known external factors. This reflects unresolved ambiguity – the cause may still be internal or hidden confounders in the environment (e.g., recent product feature releases).

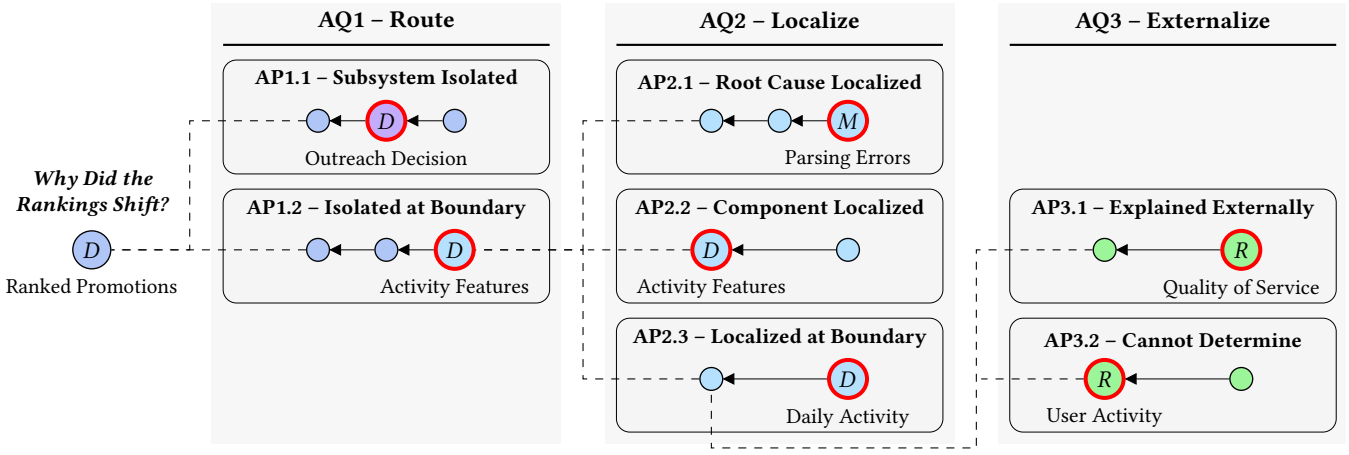


Figure 2: Illustrative scenarios showing how a shift in the promotion rankings traces by sequentially traversing views (attribution questions AQ1-3) via dashed mapping lines (same quantity across views). Branching paths show scenarios based on where mass concentrates (red outlines), each an attribution pattern (AP). Unlabeled nodes show variables around the implicated variable.

Note on Attribution. These patterns assume clear mass concentration. Scores may be distributed across variables – e.g., due to noise, or concurrent causes. When distributed, scores guide parallel investigations rather than committing to a single path. The approach applies to any data variable in the ML System View, including adding ground truth (e.g., churn labels) for performance change attribution (prediction-label joint distribution shifts).

5 Discussion

Organizational Considerations. While we focused on technical aspects, organizational considerations are equally central. **First**, we believe system maps can establish a collaborative diagnostic capability by providing a shared structure for attributing shifts across organizational boundaries. **Second**, *ML system maps* could create natural incentives for participation. Downstream teams facing the most direct exposure to shifts benefit from instrumenting their dependencies to attribute non-local faults; upstream teams similarly benefit from instrumenting their subsystems when shifts are escalated to them. **Third**, the approach can consolidate knowledge in two ways: understanding of environmental data-generating processes can be shared across modeling teams, while investigations consolidate findings by extending the map where incomplete. While maintaining a *ML System Map* requires coordination (upfront and continuous, since the system may change), we expect this to be offset by improved traceability and faster incident resolution.

Implementation Considerations. Our approach assumes extensive observability – instrumentation and lineage traceability throughout the ML system. It requires decision-level lineage that extends traceability in both **scope** and **granularity**. Current lineage tools often [9, 25] focus on data pipelines, whereas we require end-to-end coverage from data ingestion through model serving to decisions. They also operate at table or column level, whereas observation-level lineage remains an open problem [5, 39]. Additionally, such traceability presents substantial overhead through complex temporal alignment across asynchronous subsystems. When such traceability is infeasible, alternative attribution strategies can be considered – e.g., tracking summary statistics at each node and applying anomaly-based attribution over these [4]. While the attribution as illustrated might not be feasible without sufficient observability, we believe the notation and view structure provide value both as theoretical framework and implementation target.

Related Work. We situate our work closely to [21], building on their conceptual framework for describing ML systems, the insight of [26] that flow-based architectures enable fault localization through shift attribution, and the attribution algorithm of [16]. Our work integrates their insights into a holistic approach.

More broadly, causal modeling has been applied to various aspects of ML and software systems. In software quality assurance, causal graphs have been used for fault localization in microservices – modeling service dependencies and metrics [6, 8, 22, 42, 47]. Within ML systems specifically, causal analysis has addressed isolated concerns: analyzing fairness implications of pipeline and model changes [1, 15, 24], and attributing performance degradations to external changes [46]. Our approach differs by providing a holistic, system-wide model of operational ML systems that captures both internal computation and external processes, enabling causal attribution of distribution shifts across the entire system.

Research Agenda. Significant work remains to operationalize *ML System Maps* in practice. **First**, controlled experiments on simulated systems are needed to validate that the view structure enables systematic attribution under different levels of observability and map completeness. **Second**, case studies in production systems are needed to establish practical and organizational feasibility by identifying strategies for incremental map construction, sustained cross-team coordination, and addressing technical challenges. **Third**, across both experimental and production settings, comparative evaluation of causal attribution algorithms [7, 16, 26, 37] is needed to assess performance tradeoffs between attribution accuracy, computational cost, and observability requirements.

References

[1] Sumon Biswas and Hridesh Rajan. 2021. Fair preprocessing: towards understanding compositional fairness of data transformers in machine learning pipeline. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 981–993.

[2] 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, 1123–1132.

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

[4] Kailash Budhathoki, Lenon Minorics, Patrick Blöbaum, and Dominik Janzing. 2022. Causal structure-based root cause analysis of outliers. In *International conference on machine learning*. PMLR, 2357–2369.

[5] Xiwen Cai, Xiaodong Ge, Kai Xiong, Shuainan Ye, Di Weng, Ke Xu, Datong Wei, Jiang Long, and Yingcai Wu. 2025. HYPNOS: Interactive Data Lineage Tracing for Data Transformation Scripts. *IEEE Transactions on Visualization and Computer Graphics* (2025).

[6] Pengfei Chen, Yong Qi, and Di Hou. 2016. CauseInfer: Automated end-to-end performance diagnosis with hierarchical causality graph in cloud environment. *IEEE transactions on services computing* 12, 2 (2016), 214–230.

[7] Oliver Cobb and Arnaud Van Looveren. 2022. Context-aware drift detection. In *International conference on machine learning*. PMLR, 4087–4111.

[8] Cesar Couto, Christofer Silva, Marco Tulio Valente, Roberto Bigonha, and Nicolas Anquetil. 2012. Uncovering causal relationships between software metrics and bugs. In *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, 223–232.

[9] dbt Labs. 2025. dbt Developer Hub. https://docs.getdbt.com/. Documentation site. Accessed 2025-10-17.

[10] Lisa Ehrlinger, Verena Haunschmid, Davide Palazzini, and Christian Lettner. 2019. A DaQL to monitor data quality in machine learning applications. (2019), 227–237.

[11] Jean Feng, Alexej Gossmann, Gene A Pennello, Nicholas Petrick, Berkman Sahiner, and Romain Pirracchio. 2024. Monitoring machine learning-based risk prediction algorithms in the presence of performativity. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 919–927.

[12] 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.

[13] Google. 2024. TensorFlow Extended (TFX). https://www.tensorflow.org/tfx

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

[15] Zhenlan Ji, Pingchuan Ma, Shuai Wang, and Yanhui Li. 2023. Causality-aided trade-off analysis for machine learning fairness. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 371–383.

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

[17] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. 2023. Machine learning operations (mlops): Overview, definition, and architecture. *IEEE Access* (2023).

[18] Cloudera Fast Forward Labs. 2021. Inferring Concept Drift Without Labeled Data. (2021).

[19] Asmae Lamsaf, Rui Carrilho, João C Neves, and Hugo Proença. 2025. Causality, machine learning, and feature selection: a survey. *Sensors* 25, 8 (2025), 2373.

[20] Joran Leest, Ilias Gerostathopoulos, Patricia Lago, and Claudia Raibulet. 2025. Monitoring and Observability of Machine Learning Systems: Current Practices and Gaps. *arXiv preprint arXiv:2510.24142* (2025).

[21] 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).

[22] Lixian Li, Jin Liu, Zhangbing Zhou, Haoyu Luo, Wenrui Liu, and Juan Li. 2014. Causal inference based service dependency graph for statistical service fault localization. In *2014 10th International Conference on Semantics, Knowledge and Grids*. IEEE, 41–48.

[23] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2019. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2019), 2346–2363. doi:10.1109/TKDE.2018.2876857

[24] Verya Monjezi, Ashish Kumar, Gang Tan, Ashutosh Trivedi, and Saeid Tizpaz-Niari. 2024. Causal graph fuzzing for fair ML sofware development. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*.

[25] Mohammad Hossein Namaki, Avrilia Floratou, Fotis Psallidas, Subru Krishnan, Ashvin Agrawal, Yinghui Wu, Yiwen Zhu, and Markus Weimer. 2020. Vamsa: Automated provenance tracking in data science scripts. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 1542–1551.

[26] A Paleyes, Siyuan Guo, Bernhard Schölkopf, and ND Lawrence. 2023. Dataflow graphs as complete causal graphs. In *IEEE/ACM 2nd International Conference on AI Engineering–Software Engineering for AI (CAIN 2023)*. IEEE, 7–12.

[27] 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).

[28] Victor Quintas-Martinez, Mohammad Taha Bahadori, Eduardo Santiago, Jeff Mu, Dominik Janzing, and David Heckerman. 2024. Multiply-robust causal change attribution. *arXiv preprint arXiv:2404.08839* (2024).

[29] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. 2022. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistic Surveys* 16 (2022), 1–85.

[30] Salesforce. [n. d.]. Marketing Automation Software | Salesforce. https://www.salesforce.com/eu/marketing/automation/. Accessed: 2025-10-16.

[31] 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).

[32] Shreya Shankar, Labib Fawaz, Karl Gyllstrom, and Aditya Parameswaran. 2023. Automatic and Precise Data Validation for Machine Learning. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 2198–2207.

[33] 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.

[34] Shreya Shankar and Aditya G Parameswaran. 2022. Towards Observability for Machine Learning Pipelines.. In *CIDR*.

[35] Yining She, Sumon Biswas, Christian Kästner, and Eunsuk Kang. 2025. FairSense: Long-Term Fairness Analysis of ML-Enabled Systems. *arXiv preprint arXiv:2501.01665* (2025).

[36] 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.

[37] Raghav Singal, George Michailidis, and Hoiyi Ng. 2021. Flow-based attribution in graphical models: A recursive shapley approach. In *International Conference on Machine Learning*. PMLR, 9733–9743.

[38] Peter Spirtes. 2010. Introduction to causal inference. *Journal of Machine Learning Research* 11, 5 (2010).

[39] Kalle Tomingas, Priit Järv, and Tanel Tammet. 2016. Discovering Data Lineage from Data Warehouse Procedures. In *International Conference on Knowledge Discovery and Information Retrieval*, Vol. 2. SCITEPRESS, 101–110.

[40] 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.

[41] Michael P Wellman and Max Henrion. 1993. Explaining'explaining away'. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15, 3 (1993), 287–292.

[42] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. 2021. Causal inference techniques for microservice performance diagnosis: Evaluation and guiding recommendations. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 21–30.

[43] 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*.

[44] Zapier. [n. d.]. Build automated workflows with Zapier. https://zapier.com/workflows. Accessed: 2025-10-16.

[45] Amy X Zhang, Michael Muller, and Dakuo Wang. 2020. How do data science workers collaborate? roles, workflows, and tools. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW1 (2020), 1–23.

[46] 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*.

[47] Qixun Zhang, Tong Jia, Zhonghai Wu, Qingxin Wu, Lichun Jia, Donglei Li, Yuqing Tao, and Yutong Xiao. 2022. Fault localization for microservice applications with system logs and monitoring metrics. In *2022 7th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. IEEE, 149–154.

[48] Indre Zliobaite, Mykola Pechenizkiy, and Joao Gama. 2015. An overview of concept drift applications. *Big data analysis: new algorithms for a new society* (2015), 91–114.