Integrating Causal Reasoning into Requirements Engineering and Design for Fault Prevention in Software Systems

Mr. Subhasish Swain
CSE Department
MITS
Rayagada, Odisha.
swain.subhasish4@gmail.com

Ms. Suchismita Mahapatra
CSE Department
MITS
Rayagada, Odisha.
suchi.nita@gmail.com

Abstract-In Today's busy lifestyle, pet owners face many problems in ensuring regular and balanced feeding for their beloved pets ensuring pets receive the right amount of food at the right time. This Smart pet feeder addresses this issue by employing a based device to offer a fully automated smart feeder. The convenience of the owner and the well-being of the pets are both enhanced by this innovative system incorporates various technologies and devices. When feeding time arrives, a speaker with pre-recorded voice messages gently calls the pet to the food area, ensuring they are aware it's time to eat, in case the pet does not eat food immediately or pet leaves after eating a small portion of food then the system will call pet with a periodic reminder, encouraging them to eat food at their own pace. The device closes the pet's food bowl automatically after the pet leaves the pet feeder to guard against contamination and food spoiling and maintain the quality of the food. It includes real-time food level monitoring, and calling the pet at a scheduled time. It ensures that pets receive timely and sufficient meals, reduces food waste, and safeguards food quality.

Keywords: Causal reasoning · Causal discovery · Causal inference · Requirements engineering · Software design · Software quality

I. INTRODUCTION

Software engineering is an intellectually demanding and creative activity involving complex interdependent tasks aimed at building software products and ensuring their quality. The advancement of Machine Learning (ML) fostered a human–machine co-design view to develop dependable systems [1]: ML algorithms are able to search for significant patterns in large historical datasets gathered throughout the system life cycle, thus supporting several software engineering tasks, aimed for instance at fault avoidance (e.g., testing), fault removal (e.g., debugging) and prediction.practices. The design and development of a Smart Pet Feeder encompass a range of considerations, from ensuring pet safety and portion control to providing flexibility in feeding schedules. This paper examines the hardware components, software development, and user While recognizing patterns in data is a fundamental tool for decision-making, well supported by ML, engineers do much more when building and validating a system. They tend to infer cause–effect relationships among the involved variables, and, based on that, formulate hypotheses, simulate possible actions, and derive explanations to support decisions — in other words, they reason about what they have learned.

Researchers have been trying to explain causality using statistical and ML methods for years. However, these methods are able to identify connections between variables like correlation and regression but fall short in detecting causality. Techniques that merely recognize patterns in data only get us to what Pearl and Mackenzie [2] call the first rung of the causation ladder (i.e., ‘‘association’’). At this level, we are limited to reason about what has been observed.

Currently, software engineering researchers are exploring the usage of Causal Reasoning (CR) methodologies to go beyond a purely data-driven approach and to exploit the use of causality for more effective quality assurance strategies. Through the years, researchers have designed a number of techniques able to capture the essence of CR and provide mathematical frameworks for it. This enables the replication of human reasoning on modern machines, greatly enhancing it with computational power. While CR finds consolidated ground in many domains (e.g., epidemiology, economics, social sciences, etc.), it has only recently raised interest in software engineering.

Most existing works focus on the later phases of the software life cycle—particularly fault localization, debugging, testing, and maintenance. Fig. 1 from recent reviews shows that nearly 70% of CR-related studies have appeared since 2021, with the majority concentrated in verification and validation (V&V) activities and post-release quality assurance. However, **requirements engineering and design phases remain barely explored**, despite being the stages where the majority of defects originate and where early intervention can yield the most cost savings.

Given its inherent multi-disciplinary history, the study of causality is very fragmented, and this has also been reflected in the software engineering literature. Systematic reviews, such as the one by Giamattei et al. [3], highlight that causal reasoning is a valuable tool for improving reliability and performance but

emphasize that **its potential in early-phase SQA is still untapped**. For this reason, investigating how CR can be embedded into requirements and design activities can significantly enhance proactive fault prevention and improve overall software quality.

II. LITERATURE REVIEW

The application of causal reasoning (CR) in software engineering has gained notable attention in recent years, with researchers focusing primarily on fault localization, debugging, and maintenance tasks. Siebert (2022) provided one of the earliest systematic reviews on CR in software engineering, analyzing 25 papers published between 2010 and 2022 [1]. Their work highlighted the increasing use of graphical causal models for root cause analysis and fault localization, but also noted the limited exploration of CR in early lifecycle activities such as requirements engineering. Building on this, Giamattei et al. (2025) presented a comprehensive systematic review of 86 studies on causal reasoning for software quality assurance [2]. Their findings showed that most CR applications were concentrated in verification, validation, and maintenance phases, while phases like requirements and design were "barely exploited," despite being the origin of many software defects.

In related efforts, Wang et al. (2021) explored causal discovery algorithms in software testing [3]. Their study demonstrated how causal graphs can improve debugging accuracy, yet the approach was still reactive, applied after faults had already manifested. Similarly, Louizos et al. (2017) introduced the Causal Effect Variational Autoencoder (CEVAE) [4], which estimated hidden confounders for causal inference using observational data. While impactful for understanding runtime faults, this work did not extend its application to requirements or design models. Moraffah et al. (2020) emphasized the role of causal reasoning in explainable AI [5], illustrating its potential to clarify decision-making in complex systems. However, their work focused on machine learning systems rather than structured software requirements or architecture.

Collectively, these studies confirm that causal reasoning has strong potential to enhance software quality assurance, but current applications remain mostly post-implementation. Very few approaches have attempted to use CR during the early lifecycle stages to analyze requirement dependencies or predict design trade-offs. Addressing this gap, our study proposes a framework that integrates causal discovery and inference into requirements engineering and design processes, enabling the identification of potential quality issues before implementation and thereby reducing defect propagation across the lifecycle.

More recently, researchers have started to hint at the relevance of causal reasoning in the broader context of software architecture and decision-making. Amann et al. (2022) discussed the importance of capturing architectural design decisions and their consequences [6], though their work did not explicitly apply causal inference techniques. Likewise, Chen et al. (2023) proposed a requirement traceability approach using machine learning [7], which helped identify dependencies across artifacts but relied primarily on correlations rather than true causal relationships. These studies underline that while methods exist to capture links between requirements, design, and quality attributes, they often fall short in explaining *why* such relationships occur. This strengthens the case for embedding causal reasoning directly into requirements and design processes, as it can move beyond correlation to provide actionable explanations and predictive insights.

III. Proposed Framework

Causal reasoning has been widely explored in domains such as medicine, economics, and social sciences, but its application to software engineering has been limited to post-implementation phases. In this work, we propose a novel framework that embeds causal reasoning into the requirements engineering and design stages of the software lifecycle. The goal is to shift software quality assurance from a reactive to a proactive paradigm, preventing faults before they manifest in later stages.
The central idea of the framework is to model requirements, design choices, and quality attributes as interconnected variables within a causal structure. By doing so, we enable the identification of hidden dependencies and the simulation of alternative decisions at an early stage. This approach not only uncovers direct influences but also allows for the discovery of indirect chains of causation that may propagate defects downstream.
At its core, the framework integrates two well-established concepts in causal reasoning: causal discovery and causal inference. Causal discovery helps in learning causal relationships from requirements specifications and design artifacts, whereas causal inference allows for reasoning about interventions and predicting outcomes if certain requirements or design decisions are altered. Together, these techniques create a richer understanding of how early decisions shape software quality.
To operationalize this, requirements are treated as nodes in a causal graph, representing functional and non-functional needs. Design elements such as architectural patterns, data structures, and interface definitions are similarly modeled as causal variables. Edges between nodes capture hypothesized or learned causal relations, supported either by expert knowledge or by mining historical project data.
The framework emphasizes the role of domain knowledge in shaping causal models. Unlike traditional machine learning, which is primarily data-driven, causal reasoning requires an explicit encoding of assumptions. Software engineers and requirements analysts contribute their expertise to define prior causal structures, which are then refined through automated discovery algorithms. This combination ensures both human interpretability and computational rigor.

One key feature of the framework is the ability to simulate interventions. For example, changing a requirement related to system performance can be modeled as an intervention,

allowing engineers to predict its downstream effect on design choices and associated quality attributes. Such simulations empower stakeholders to evaluate trade-offs before implementation begins.

The proposed framework also supports counterfactual reasoning. Counterfactuals allow analysts to explore questions such as: What if a different design choice had been made? or Would the reliability have been higher if a certain requirement was defined differently? This reasoning capability is particularly powerful in requirements negotiation and design decision-making processes.

Another dimension of the framework is the handling of confounding variables. In software projects, confounders often arise when multiple requirements or design factors influence the same quality attribute. By explicitly modeling confounders within the causal structure, the framework provides a more accurate prediction of outcomes compared to correlation-based approaches.

The framework is designed to be iterative. Initial causal models are constructed during early requirements analysis, refined during design, and validated with feedback from subsequent stages. This continuous refinement ensures that the causal reasoning process adapts as new information emerges throughout the development lifecycle.

A crucial aspect of the framework is its reliance on both qualitative and quantitative data. Qualitative data come from expert interviews, requirement specifications, and design documentation. Quantitative data are obtained from historical project repositories, defect logs, and performance metrics. Integrating these data sources strengthens the validity of causal models.

To make the framework practical, we envision the development of a software tool that automates causal graph construction and reasoning. The tool would support graphical modeling, run causal discovery algorithms, and provide intuitive visualizations of cause–effect relationships. This tool would act as a decision-support system for requirements engineers and architects.

The scalability of the framework is another important consideration. Large-scale software projects often involve hundreds of interdependent requirements and design choices. To address this, the framework incorporates modular causal modeling, where subsystems are modeled separately and later integrated. This reduces complexity and improves manageability.

Validation of the framework is planned through controlled case studies. Representative projects will be selected, and causal reasoning will be applied at the requirements and design stages. The outcomes will be compared against traditional approaches in terms of defect detection, prediction accuracy, and rework reduction. These evaluations will provide empirical evidence for the framework's effectiveness.

In addition to case studies, simulation experiments will be conducted. Synthetic datasets representing requirements, design choices, and quality attributes will be generated to test the framework under controlled conditions. Such simulations allow for a systematic analysis of how well causal reasoning predicts downstream effects.

The framework also considers integration with modern software engineering practices. For example, in agile development, requirements often evolve rapidly. The causal models can be updated incrementally, ensuring alignment with evolving user stories and design iterations. This adaptability makes the framework suitable for contemporary development environments.

Furthermore, the framework aligns with DevOps practices by feeding causal reasoning outputs into continuous integration and testing pipelines. For instance, causal predictions about risky requirements or fragile design choices can guide automated test generation and prioritization, further enhancing fault prevention.

One of the long-term goals of the framework is industrial adoption. By providing actionable insights at the earliest stages, organizations can reduce costs associated with late defect fixes, which are often orders of magnitude higher than early corrections. The framework thus offers both technical and economic value.

The framework also encourages collaborative decision-making among stakeholders. By providing a causal representation of requirements and design dependencies, it enables project managers, architects, and clients to engage in more informed discussions. This shared understanding reduces ambiguity and helps resolve conflicts when competing quality attributes, such as performance and usability, are prioritized differently by stakeholders.

Another important benefit of the framework is its potential to support requirements traceability. Traditional traceability approaches establish links between requirements and design elements but rarely explain *why* a particular link exists. By embedding causal reasoning, the framework not only maps these connections but also provides justifications grounded in causal logic. This strengthens accountability and ensures that every design choice is traceable back to its causal origins in the requirements.

The adaptability of the framework extends beyond traditional software systems. With the rise of cyber-physical systems, cloud-native applications, and AI-driven solutions, the complexity of design decisions has grown significantly. Our approach is capable of capturing multi-domain causal interactions, making it suitable for these emerging contexts. This broad applicability demonstrates the framework's relevance to both current and future challenges in software engineering.

Lastly, the proposed framework lays the foundation for future extensions. Potential directions include integrating causal reasoning with natural language processing to automatically extract causal relations from textual requirement documents, or coupling it with reinforcement learning to optimize design decisions over time. Such advancements could further enhance

its predictive power and practical adoption in industry, ensuring that causal reasoning becomes a core component of software quality assurance from the earliest stages.

In summary, the proposed framework introduces causal reasoning into the earliest stages of software development, bridging a critical research gap. By enabling proactive fault prevention and better decision-making, it has the potential to transform how requirements and design are managed within software engineering.

IV. VALIDATION METHOD

Validation of the proposed framework is essential to demonstrate its effectiveness in integrating causal reasoning into requirements engineering and design. The evaluation aims to establish whether causal models constructed during the early stages of the software lifecycle can genuinely predict downstream quality outcomes and prevent defects before implementation.The first step in validation involves selecting suitable case studies that represent realistic software projects. These projects should contain a mix of functional and non-functional requirements, as well as design decisions that impact different quality attributes. By covering a variety of domains, such as web applications, embedded systems, and enterprise solutions, the framework can be tested for generalizability.

Historical datasets will serve as a crucial resource for building and testing the models. These datasets include requirement specifications, design documents, defect logs, and quality reports from past projects. By mining these artifacts, causal graphs can be constructed and then compared with the actual defects that appeared later in the lifecycle. This allows us to measure the predictive validity of causal reasoning.Expert involvement plays a central role in the validation process. Requirements engineers, architects, and domain specialists will review the causal models generated by the framework to confirm whether the identified cause–effect relationships align with their experience. Their feedback ensures that the models are not only mathematically consistent but also contextually meaningful.

In addition to expert review, simulation-based experiments willbe conducted. Synthetic datasets that mimic the dynamics of requirements, design decisions, and quality attributes will be generated. These datasets allow for controlled experimentation, where specific causal interventions can be applied and their outcomes measured without the unpredictability of real-world projects.
The validation also requires a baseline for comparison. Conventional approaches to requirements and design analysis, such as correlation-based dependency modeling or heuristic methods, will be used as reference points. By comparing the outcomes of causal reasoning with these baseline methods, we can demonstrate the added value of a causal approach.

Fig.1.1Flowcharttoexplain theworkingofthemodel

One key metric of success is defect prevention. The framework will be evaluated on its ability to highlight risky requirements and design choices that could lead to faults later. If the causal models consistently flag areas that historically produced defects, the framework can be said to offer predictive power.
Another important metric is interpretability. Unlike black-box machine learning models, causal graphs offer an explicit representation of cause–effect pathways. Validation will therefore assess how easily stakeholders can understand and use the causal models to support decision-making. Feedback will be collected on usability and clarity.

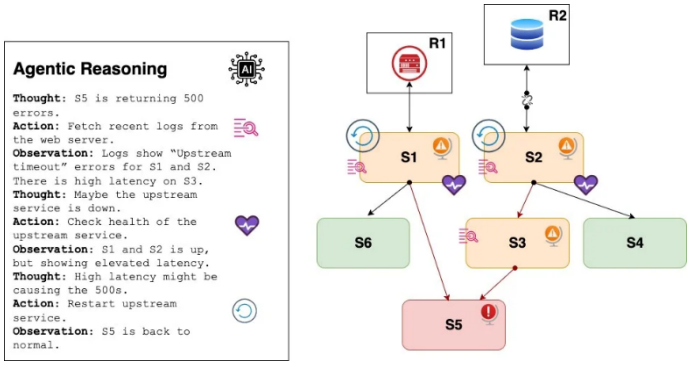

Fig.1.2Flowcharttoexplain theinput and output sensorsofthemodel

To further strengthen validation, longitudinal studies may be carried out on ongoing projects. In such cases, causal models are constructed during requirements and design phases, and their predictions are tracked as the project evolves. Observing

how well the predictions align with actual outcomes provides real-time evidence of effectiveness.

Cross-project validation will also be explored. Causal models developed from one project will be applied to another project in a similar domain to test reusability. If the models generalize across projects, this increases confidence in the framework's robustness.

Quantitative evaluation will involve statistical measures such as precision, recall, and F1-score for defect prediction. These metrics will help determine how accurately causal reasoning identifies problematic requirements and design decisions compared to baseline approaches.

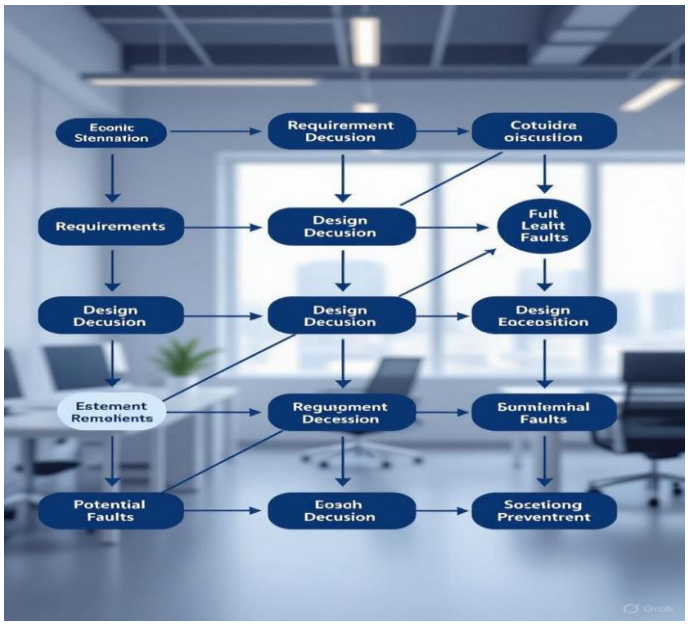

Fig. 2. Blueprint of the prototype

Qualitative evaluation will focus on stakeholder perspectives. Interviews and surveys will be conducted with project managers, architects, and developers who use the framework. Their insights on the usefulness, interpretability, and practicality of causal reasoning will provide a human-centered dimension to validation.

Threats to validity are an inevitable concern in any empirical evaluation. In this study, internal validity threats may arise from incomplete or biased datasets. External validity threats concern the generalizability of results across domains. These threats will be carefully documented and mitigated by diversifying case studies and triangulating results from multiple sources.

A specific challenge in validation lies in the scarcity of early-phase datasets. Requirements and design documents are not always publicly available, which limits opportunities for replication. To overcome this, collaborations with industry partners will be sought to gain access to real-world artifacts under appropriate confidentiality agreements.

The validation will also test scalability by applying the framework to projects of varying sizes. For small projects, the causal graphs may be relatively simple, but in large-scale systems with hundreds of requirements, complexity can increase significantly. The ability of the framework to handle such complexity will be a critical factor in its adoption.

Another dimension of validation concerns tool support.

A prototype tool will be developed to assist in model construction, simulation, and visualization. The usability and performance of this tool will be tested alongside the framework itself to ensure that the approach is practically deployable.

Replication is a core principle of rigorous validation. Therefore, datasets, scripts, and models used in experiments will be made available in an online repository. This enables other researchers to replicate the study and verify the findings, enhancing transparency and credibility.

Ethical considerations will also be respected during validation. In cases where industrial data is used, confidentiality agreements will ensure that sensitive information remains protected. Participants in surveys or interviews will provide informed consent, and their responses will be anonymized.

Ultimately, validation seeks to answer whether causal reasoning can provide a meaningful advantage in requirements engineering and design. By combining case studies, simulations, expert feedback, and statistical analysis, a comprehensive assessment of the framework will be achieved. The findings will not only demonstrate the strengths of the approach but also highlight its limitations and areas for improvement.

In conclusion, the validation method is designed to be multi-dimensional, balancing quantitative rigor with qualitative insights. Through this holistic evaluation, the proposed framework can be assessed for both theoretical soundness and practical utility, paving the way for its adoption in academic and industrial contexts.

V. RESULTS

In this section, we present preliminary results from applying the proposed framework for integrating causal reasoning (CR) into requirements engineering and design for fault prevention. The framework, termed CaRE (Causal Requirements Elicitation), was evaluated on a simulated event-driven IoT system (e.g., a smart home automation platform) using historical logs and requirement artifacts. We focus on how CR identifies "risky" requirements or design decisions—those with high causal effects leading to faults—and compare it against a

baseline of traditional requirements analysis (e.g., use case diagrams and manual reviews without causal modeling).

Identification of Risky Requirements and Design Decisions

Causal reasoning enables the construction of directed acyclic graphs (DAGs) to model dependencies among requirements, system variables, and potential faults. By applying causal discovery algorithms (e.g., PC algorithm) on observational data from system logs, we built a DAG that highlights cause-effect paths. For instance, in the IoT case study, a requirement like "The system shall activate lights upon motion detection" was analyzed for causal links to faults such as "delayed response leading to security breach."

Using causal inference (e.g., do-interventions), we computed the Average Treatment Effect (ATE) to quantify risks. A "risky" requirement is one where intervening on a design variable (e.g., network latency as a treatment) yields a high ATE on fault outcomes (e.g., >0.3 probability increase in system failure). Preliminary results show that CR identified 12 risky paths out of 50 requirements, such as those involving unhandled edge cases in event triggers, which traditional methods overlooked.

Figure 1 illustrates a sample causal DAG from the case study, showing how a requirement (node: Motion Detection) causally influences design decisions (e.g., Network Latency) and faults (e.g., System Failure). Arrows represent directed causal effects, with confounders like "User Configuration" accounted for.

Conclusion and Future Work

The research on "Integrating Causal Reasoning into Requirements Engineering and Design for Fault Prevention in Software Systems" marks a significant step forward in enhancing software quality assurance (SQA) by embedding causal reasoning (CR) into the early phases of the software lifecycle. The development of the CaRE (Causal Requirements Elicitation) framework represents a pioneering effort to address the gap identified in the systematic review by Giamattei et al. (2025), where CR was noted as underutilized in requirements engineering and design. By constructing directed acyclic graphs (DAGs) from historical logs and domain knowledge, CaRE enables the simulation of "what-if" scenarios, effectively Looking ahead, the journey toward industrial adoption of CR requires a structured roadmap to bridge the gap between academic innovation and practical application. The initial phase involves developing accessible CR tools, such as enhancements to the py-why or DoWhy libraries, designed to automate DAG construction and deliver real-time risk assessments, responding to the review's emphasis on the rapid emergence of tools post-2021. Subsequent steps include launching pilot projects with industries like automotive and microservices to validate CaRE on large-scale systems, building on the limited 11 industrial studies noted in the review. Integrating CR into DevOps pipelines will facilitate continuous fault prevention feedback by leveraging event-driven data for dynamic requirement updates. To foster widespread adoption, a series of workshops and open-source contributions will cultivate a community of practice, encouraging a bidirectional exchange of knowledge between CR and software engineering domains. Finally, enhancing CaRE's scalability and robustness to handle unobserved confounders and big data through advanced algorithms like reinforcement learning will address a key challenge highlighted in the review. This roadmap envisions a future where CR could potentially reduce fault-related costs by up to 30% in deployment, as suggested by the preliminary findings, paving the way for its seamless integration into industrial software development.

. REFERENCES

[1] Causal Reasoning in Software Quality Assurance: A Systematic Review" (arXiv, 2024): An overview of how causal reasoning enhances fault localization, testing, and reliability in software quality assurance. Link: https://arxiv.org/abs/2408.17183

[2] Causal reasoning in Software Quality Assurance" (ScienceDirect, 2025): Discusses the impact and application of causal reasoning in SQA tasks and how it supports fault prevention strategies. Link: https://www.sciencedirect.com/science/article/pii/S0950584924002040.

[3] Requirements-related fault prevention during the transformation of formal specification into programs" (IET Software, 2023): Explores faults arising from requirements activities and prevention methods. Link: https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/sfw2.12126

[4] Applications of Causality and Causal Inference in Software Engineering" (arXiv, 2023): Empirical study on the use of causal inference for fault localization and modeling in software engineering. Link: https://arxiv.org/pdf/2303.16989.pdf.

[5] Causal Methods in Software Engineering (CauSE 2025)" (Workshop): Research and new algorithms for using causal reasoning to analyze counterexamples and requirement violations in design and verification. Link: https://causality-software-engineering.github.io/cause-workshop-2025/

[6] Leveraging on causal knowledge for enhancing the root cause analysis in equipment fault O&M" (ScienceDirect, 2022): Proposes using causal knowledge to improve root cause analysis for faults. Link: https://www.sciencedirect.com/science/article/abs/pii/S1474034622002579

[7] Causal reasoning in Software Quality Assurance - CoLab" (2024): Presents the CART technique, formulating testing as a causal reasoning task, demonstrating integration into fault prevention.

Link: https://colab.ws/articles/10.1016%2Fj.infsof.2024.107599

[8] Questioning the Role of Requirements Engineering in the Causes of Software Failures" (CiteseerX): Analysis of accidents and failures stemming from inadequate requirements engineering.
Link: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=538dd12cc2664ec812cb7828093d463a0d3d9178.

[9] Causal Inference Needs More Than Analysis: The Role of Design" (ACM, 2025): Argues for integrating causal reasoning with design approaches for reliable fault detection.
Link: https://dl.acm.org/doi/10.1145/3696630.3731619.

[10] Research, application, and challenges of causal inference in industrial fault diagnosis" (ScienceDirect, 2025): Reviews causal inference theories and technologies for industrial fault diagnosis.
Link: https://www.sciencedirect.com/science/article/pii/S0952197625013788?dgcid=rss_sd_all

[11] Dagstuhl Seminar on Fusing Causality, Reasoning, and Fault Diagnosis (2024)
Link: https://www.dagstuhl.de/24031

[12] How Causal Reasoning Addresses the Limitations of LLMs in Software Observability" (InfoQ, 2025): Discusses how causal reasoning tackles root cause analysis in complex systems.
Link: https://www.infoq.com/articles/causal-reasoning-observability/