Interventional Root Cause Analysis of Failures in Multi-Sensor Fusion Perception Systems

Shuguang Wang*, Qian Zhou*, Kui Wu†, Jinghuai Deng*, Dapeng Wu*, Wei-Bin Lee‡, Jianping Wang*
*City University of Hong Kong, Email: {sgwang6-c, jinghdeng2-c}@my.cityu.edu.hk,
{qiazhou, jianwang}@cityu.edu.hk, dpwu@ieee.org
†University of Victoria, Email: wkui@uvic.ca
‡Information Security Center, Hon Hai Research Institute, Email: wei-bin.lee@foxconn.com

***Abstract*—Autonomous driving systems (ADS) heavily depend on multi-sensor fusion (MSF) perception systems to process sensor data and improve the accuracy of environmental perception. However, MSF cannot completely eliminate uncertainties, and faults in multiple modules will lead to perception failures. Thus, identifying the root causes of these perception failures is crucial to ensure the reliability of MSF perception systems. Traditional methods for identifying perception failures, such as anomaly detection and runtime monitoring, are limited because they do not account for causal relationships between faults in multiple modules and overall system failure. To overcome these limitations, we propose a novel approach called interventional root cause analysis (IRCA). IRCA leverages the directed acyclic graph (DAG) structure of MSF to develop a hierarchical structural causal model (H-SCM), which effectively addresses the complexities of causal relationships. Our approach uses a divide-and-conquer pruning algorithm to encompass multiple causal modules within a causal path and to pinpoint intervention targets. We implement IRCA and evaluate its performance using real fault scenarios and synthetic scenarios with injected faults in the ADS Autoware. The average F1-score of IRCA in real fault scenarios is over 95%. We also illustrate the effectiveness of IRCA on an autonomous vehicle testbed equipped with Autoware, as well as a cross-platform evaluation using Apollo. The results show that IRCA can efficiently identify the causal paths leading to failures and significantly enhance the safety of ADS.**

I. Introduction

In recent years, autonomous vehicles have transformed the transportation industry with significant advantages, such as decreased traffic accidents and congestion, enhanced urban management, and improved fuel efficiency. Services like Waymo One's driverless rides [1], sanctioned by the California Public Utilities Commission, represent a giant leap in autonomous driving technology. An autonomous driving system (ADS) comprises multiple components, such as localization, perception, prediction, planning, and control. Among them, the perception system stands as the foundation upon which subsequent systems like prediction and planning are built. To achieve a high accuracy in perception tasks like object detection and tracking, multi-sensor fusion (MSF) perception systems have been widely used in ADS, such as Autoware [2] and Apollo [3]. By harnessing the collective capabilities of various sensors, MSF perception systems offer a holistic view of the vehicle's surroundings. Despite these advances, the intricate interactions between the various perception modules, as illustrated in Figure 1, introduce complexity and vulnerabilities. Even minor errors or inaccuracies in a single sensor can cascade, potentially leading to critical failures of the entire perception system [4], [5].

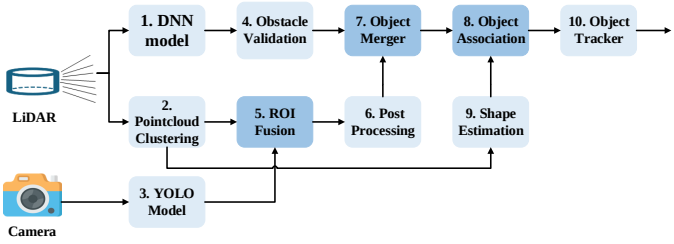


Fig. 1: The architecture of a typical MSF perception system.

Conducting a root cause analysis (RCA) of perception failures is essential for discovering causal modules and aiding the system's recovery. The manual extraction of causal relationships for RCA proves to be a daunting task, especially within complex perception systems characterized by numerous interdependent components and extensive data. Thus, automating the RCA process becomes critical. This process aims to achieve two primary objectives: (1) locating the responsible perception modules for the failure; and (2) identifying the specific fault modes[1] within a perception module that caused the failure. The above goals pose three unique challenges:

- (**Challenge 1**) Causal modeling of MSF systems: In MSF systems, each module's output contains multidimensional information and dynamically changes with varying environmental inputs, making it challenging to identify effective causal variables.
- (**Challenge 2**) Complex dependency relations between modules: MSF perception systems have intricate depen-

[1]A perception module may have multiple fault modes, e.g., missing objects and misclassifying objects are different fault modes.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.240036
www.ndss-symposium.org

dencies as shown in Figure 1. These complex relationships further complicate the identification of module-level faults when system-level failures occur [4].
- (**Challenge 3**) Multiple root causes: A specific failure in the perception system could have multiple root causes. The causal effects of multiple faults may obscure each other, making it challenging to identify all root causes.

Existing RCA approaches cannot meet the unique challenges of our application context. Firstly, RCA methodologies in cloud applications typically construct causal graphs based on a predefined set of performance metrics to identify the root causes [6], [7]. However, these methodologies rely on a specific set of selected metrics, which cannot provide effective indicators for identifying module faults in the MSF perception system. Secondly, RCA has also been explored in the ADS domain. For instance, Sun et al. [8] and Wan et al. [9] have applied causality in scenario-based testing to uncover causal factors behind driving violations in ADS. However, their methods only perform well when module dependencies are simple, but struggle to handle the complex dependency relations between modules in the MSF perception system. Thirdly, the adaptive interventional debugging (AID) approach [10] localizes the root cause of intermittent failures by combining statistical debugging with causal analysis. Nonetheless, AID assumes a single root cause, and is ill-equipped to analyze the multiple concurrent root causes that can impact MSF perception systems. Overall, the intricate nature of the MSF perception system demands a new solution that focuses on the specific causal relationships between modules.

To answer the call, we propose a novel *interventional root causal analysis (IRCA)* approach that tackles the unique challenges faced in analyzing failures within MSF perception systems in autonomous vehicles. To be specific, IRCA consists of several key features that directly address the three main challenges mentioned above.

1) Hierarchical structural causal model (H-SCM): IRCA incorporates hierarchical structural causal models enriched with MSF domain-specific insights. By leveraging these insights, the approach effectively captures the dynamic and multi-dimensional relationships within MSF systems. H-SCM enables precise identification of effective causal variables and systematically links module faults to system failures. (**Addressing Challenge 1**)
2) Counterfactual interventions: Unlike traditional causality approaches that might only identify correlations, IRCA utilizes counterfactual reasoning to establish definitive cause-and-effect relationships. It involves altering outputs of suspected modules to hypothesized fault-free states and observing whether these changes resolve the system-level failures. This approach addresses intricate dependencies by directly testing the impact of specific module outputs on the overall system behavior. (**Addressing Challenge 2**)
3) Hierarchical pruning algorithm: IRCA employs a hierarchical intervention algorithm. This algorithm uses a divide-and-conquer strategy, systematically applying interventions across different modules in the system. By doing so, it isolates and identifies the specific contributions of each fault to the overall failure, thereby managing the complexity introduced by multiple concurrent root causes. (**Addressing Challenge 3**)

In summary, we make the following contributions:

- We are the first to study the root cause analysis of perception failure within the domain of ADS. We propose interventional root cause analysis (IRCA), a novel causality-driven testing approach that localizes the root causes of perception failures. IRCA leverages a unique combination of runtime monitoring, causal analysis, and scenario testing to identify the root causes systematically.
- We propose a novel solution to identify multiple causes contributing to perception failures using a hierarchical pruning algorithm. Our algorithm not only handles a single root cause but also handles concurrent root causes.
- Our approach is evaluated in both a simulated environment and on a real-world autonomous vehicle. We implemented and assessed our method using Autoware, a leading open-source ADS, leveraging real issues documented on GitHub. The high F1-score of 95.39% highlights the effectiveness of IRCA in the actual system. To further ensure a comprehensive evaluation, we also conducted experiments with various injected synthetic faults. The average recall of 89.83% indicates that our approach effectively identifies faults from diverse modules.

II. Background

We introduce the MSF perception system and define failures and fault modes in this section.

A. MSF Perception System

MSF perception system is used in many industry-grade ADS to reduce the risks associated with single sensor failures, thus helping to prevent potential accidents [3], [11]. MSF often integrates multiple outputs from the camera, radar, and LiDAR. When used for ADS, MSF mainly has three types based on the level of data abstraction and integration [12]: high-level fusion (HLF), mid-level fusion (MLF), and low-level fusion (LLF). In HLF, each sensor processes its data independently to perform object detection or tracking tasks. The output from each sensor is typically at the object level, meaning that each sensor independently identifies and tracks objects within its field of view. MLF works at the feature level by integrating features extracted from the data provided by different sensors [13]. LLF operates at the lowest level by directly fusing raw data from multiple sensors [14]. Our work focuses on HLF, which is widely used in open-source ADS [11], [15].

Now we introduce the structure of module outputs. An MSF perception system S consists of n modules. These modules are denoted as a set $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$, including various object detection, fusion, and tracking components. For each time step t within the duration T of interest, each module

$M_i \in \mathcal{M}$ outputs a set of detected objects $\mathcal{O}_i^t$. Each detected object $O_{ij}^t \in \mathcal{O}_i^t$ is characterized by a tuple (X_{ij}^t, C_{ij}^t), where:

- **Parameters** (X_{ij}^t): The parameters that define the kinematic and shape of the object O_{ij}^t. Kinematic parameters contain position, rotation, and velocity, the shape is defined by dimensions.
- **Class** (C_{ij}^t): The classification category, such as 'Vehicle', 'Pedestrian', and 'Truck'.

On the other hand, the ground truth set of objects surrounding the system S at time t is denoted as $\mathcal{G}^t$. Each ground truth object $G_k^t \in \mathcal{G}^t$ is characterized by a tuple (X_k^t, C_k^t).

B. Failures, Faults, and Fault Modes

In the MSF perception system, a failure represents the *final* erroneous output, whereas a fault is an erroneous output from an individual module.

Fault modes at time t for i-th module M_i are defined based on inconsistency of the object O_{ij}^t with the ground truth G_k^t, as below:

1) **Missing Obstacle** (MO): A binary indicator for the fault mode where a true object is not detected, i.e.,
$$\text{MO}_{ik}^t = \mathbb{1}(G_k^t \notin \mathcal{O}_i^t)), \tag{1}$$
where $\mathbb{1}(x)$ is a function that returns 1 is the condition x is true and 0 otherwise.
2) **Ghost Obstacle** (GO): A binary indicator for the fault mode where a non-existent object is detected, i.e.,
$$\text{GO}_{ij}^t = \mathbb{1}(O_{ij}^t \notin \mathcal{G}^t). \tag{2}$$
3) **Misclassification** (MC): A binary indicator for the fault mode where an object's class is incorrectly identified, i.e.,
$$\text{MC}_{ij}^t = \mathbb{1}(C_{ij}^t \neq C_k^t). \tag{3}$$
4) **Parameter Errors** (PE): An indicator representing the fault mode where the parameters of a detected object differ from the ground truth by more than a predefined threshold, i.e.,
$$\text{PE}_{ij}^t = \mathbb{1}(\left|X_{ij}^t - X_k^t\right| > \text{threshold}). \tag{4}$$

Fault Mode Vector. The fault mode vector for module M_i at time t is defined as:
$$V_i^t = (\text{MO}_i^t, \text{GO}_i^t, \text{MC}_i^t, \text{PE}_i^t) \tag{5}$$
where $\text{MO}_i^t, \text{GO}_i^t, \text{MC}_i^t, \text{PE}_i^t$ each takes a value in $\{0,1\}$. These indicators are set to 1 if their respective fault conditions are met for module M_i. For example, MO_i^t is 1 if any MO_{ik}^t is 1, indicating at least one missing obstacle detection. The vector V_i^t thus collects these indicators to comprehensively represent each module's fault status at time t.

Security Implications. "MO" can increase the risk of collisions. If the ADS fails to detect an obstacle, it may continue to drive at unsafe speeds, thereby increasing the likelihood of a collision with the obstacle [16]. "GO" may lead the ADS to execute unnecessary or sudden maneuvers, which could result in collisions with other vehicles or improper parking. "MC" can lead to inappropriate safety measures taken by ADS because different types of obstacles may require varying deceleration and avoidance maneuvers. "PE", such as incorrect localization, can impact the decision-making process due to wrongly determining the relative position of obstacles, posing a risk of triggering improper driving decisions [17]. Identifying the causes of these perception failures is essential for improving ADS's overall safety and reliability.

III. Problem formulation

In an MSF perception system, detecting the root causes of failures involves identifying not just isolated faulty modules but also understanding how these modules interact along multiple sensor fusion paths. The system is represented as a directed acyclic graph (DAG), where each node corresponds to a module, and edges represent dependencies between these modules. We build the Hierarchical Structural Causal Model (H-SCM) that combines the DAG and the set of potential abnormal outputs, which are identified as fault mode vectors and are then treated as causal variables in the H-SCM. The DAG defines the potential causal relationships between these causal variables. Therefore, we frame the RCA problem of MSF failures as a causal path discovery problem through H-SCM, which aims to identify an ordered subset of causal modules that form a complete causal path leading to the failure. The formal definition is as follows:

Definition 1. ***Causal Path Discovery:*** *Given a Hierarchical Structural Causal Model (H-SCM) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a failure indicator F indicating a specific failure, $\mathcal{V}$ includes all potential variables, $\mathcal{E}$ represents the causal relationships between these variables. The causal path discovery problem aims to identify a path $\mathbb{P} = \langle V_0, V_1, \ldots, V_n \rangle$ such that the following conditions are met:*

1) *V_i represents the fault mode vector of the i-th module.*
2) *For all $0 \leq i \leq n$, $V_i \in \mathcal{V}$ and for all $0 \leq i < n$, $(V_i, V_{i+1}) \in \mathcal{E}$.*
3) *Each V_i causally influences V_{i+1}.*
4) *Each V_i is a sufficient condition to trigger F; F remains unless all V_i ($0 \leq i \leq n$) are fixed.*

Counterfactual Causality. This analysis focuses on the counterfactual causes of failure, utilizing the concept of counterfactual causality [18], [19] to understand the causal relationship between the causal path $\mathbb{P}$ and a failure F in the MSF perception system S. We articulate this relationship as follows: "If $\mathbb{P}$ does not exist, F will not exist."

Collaborative Faults. As noted in condition 4), this paper addresses multi-fault situations characterized by an OR relationship: the presence of *any* V_i in $\mathbb{P}$ is sufficient to cause F. Theoretically, there could also be scenarios where multiple faults have an AND relationship, with *all* of them collaborating to cause F. While we acknowledge the possibility of such situations in other systems, we have not identified any in our context yet. This absence is consistent with the real faults reported to the Autoware Foundation [11] and related

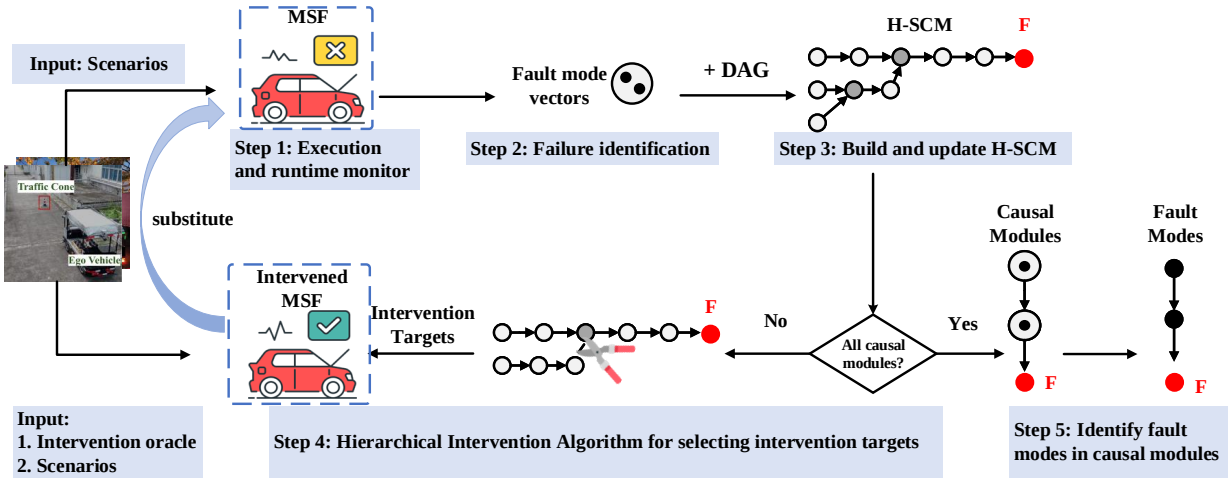


Fig. 2: The framework of interventional root cause analysis.

academic publications [17]. Hence, we focus on resolving real issues that impede autonomous vehicles and do not consider fictional collaborative faults. Handling collaborative faults, if any, remains an open challenge for future research.

Mode-Level Cause Attribution. After identifying module-level causes, IRCA also discovers specific fault modes in the identified modules that lead to failures in object detection and tracking tasks. We use a fault mode vector based on data from object messages to pinpoint the modules responsible for system failure. This helps us determine which messages are associated with specific faults, such as classification errors, shape recognition inaccuracies, and distance measurement errors. By analyzing the fault mode vector, we can address these faults more effectively.

IV. Root Cause Analysis Approach

In this section, we introduce the design of IRCA. IRCA has two main goals: (1) Finding the perception modules responsible for the failure; and (2) Identifying the specific fault modes at the message level that caused the failure.

A. Overview

IRCA is a methodical approach to identify system failures' root causes. It combines runtime monitoring and intervention algorithms to uncover the underlying causal relationship behind system failures. Figure 2 illustrates the IRCA workflow. It is fully automated and designed to be model-agnostic. In this framework, identified failure scenarios are the initial inputs that drive the subsequent steps of the root cause analysis.

Step 1: Runtime monitoring. The framework first employs runtime monitoring to track the outputs of each module in the MSF perception system during execution.

Step 2: Failure identification. When a failure occurs, this step involves identifying abnormal outputs from various modules, which are suspected as potential causes of the failure. As illustrated in Figure 2, the fault mode vectors are depicted as black solid circles within each module (represented by grey circles). A binary encoding operation is applied to these vectors to aggregate the elements of these vectors into a single scalar causal variable. For instance, consider the vector $V_i^t = (0, 0, 1, 1)$. By treating each element as a binary digit, the vector can be encoded as the binary number 0011, and represented by 3 in decimal. This operation simplifies the initial identification process by providing a consolidated view of potential fault contributions from each module.

Step 3: Build the hierarchical structural causal model (H-SCM). After identifying the potential causal variables $\mathcal{V}$, where each variable represents a fault mode vector V_i of a module M_i, we apply structural constraints in the directed acyclic graph (DAG) of the MSF perception system to build and update the H-SCM. There are three structural constraints in the DAG defined by the configuration of the MSF [11], [15]:

1) **Required connection**: A direct causal link must exist from V_i to V_j.
2) **Forbidden connection**: A direct causal link from V_i to V_j is not allowed.
3) **Temporal order**: For any V_i and V_j representing the fault modes of modules M_i and M_j, respectively, if M_i precedes M_j, then V_i precedes V_j.

These structural constraints ensure that the H-SCM reflects the real interactions and causal relationships among MSF modules, grounded in both expert knowledge and empirical findings.

Step 4: Hierarchical intervention algorithm. While the H-SCM captures temporal precedence, it alone is not sufficient to establish causality. Some modules are identified as fault modules because their outputs are influenced by faulty inputs. Their fault mode vectors are not true root causes of the final failure. To differentiate between mere correlations and actual causal relationships, IRCA employs interventions [19] detailed in Section IV-B, which involve deliberately altering the outputs of modules which are suspected to be faulty. The system's response to the interventions helps verify or refute the causality of modules.

The algorithm improves efficiency by selectively pruning the search space for potential intervention targets, utilizing

Branch Pruning with intervention (Section V-A) and Node Pruning with intervention(Section V-B). As depicted in Figure 2, the iterative process replaces the original MSF with an intervened version in subsequent iterations. Our method tests various combinations of modules and interventions, progressively refining the identification of discriminative faults from **Step 1** to **Step 4**. This iterative pruning and testing process removes non-causal connections from the H-SCM until the true causal paths are discovered. Upon successful identification of causal modules, the process goes to **Step 5** to further isolate specific fault modes within these modules.

Step 5: Identify fault modes in causal modules. After the encoded causal variable has been established in **Step 2** and the causal modules have been identified in **Step 4**, this step focuses on decoding the causal variables from scalars to raw fault vectors to analyze individual fault modes. For instance, scalar 3 converts back to binary 0011, which maps the fault mode vector $V_i^t = (0, 0, 1, 1)$ defined by Equation (5). We identify specific fault modes by analyzing their frequencies across various frames. Each element of the decomposed vector represents a specific fault mode. The fault modes are further examined to understand their impact and the specific messages associated with each, providing deeper insights into the root causes. The causal chain of these fault modes is represented as $\mathbb{P} = \langle V_0, V_1, \ldots, V_n \rangle$, which outlines the sequence of modules associated with fault modes.

B. Intervention Mechanism

We implement a counterfactual causality-based intervention mechanism in our IRCA approach. This mechanism involves the use of an "intervention oracle" to replace the abnormal output from a system module with a normal output. "Oracle" comes from the software engineering term "test oracle", used to check the correctness of the program's outputs for test cases [20]. In our context, intervention oracle represents the expected output of each module for the MSF perception system to work correctly. The intervention oracle contains the actual object information such as position and classification. In simulations, the object information is provided by the simulators, while in physical implementations, we utilize recorded event logs. We verify the output type from the module, determining if they are tracked or detected objects. The intervention oracle is then serialized into the format according to the unique message structure for each object type. Following this, we replace the target module requiring intervention with the intervention oracle.

C. Automation of Implementation

The implementation benefits from the publish-subscribe architecture that facilitates module communication in real-time systems. For example, Autoware [11] and Apollo [15], which are two prominent open-source autonomous driving systems (ADS) platforms, utilize the ROS2 and CyberRT frameworks, respectively. Both frameworks employ a publish-subscribe architecture to manage communications [21], [22]. This allows IRCA to be integrated efficiently, monitoring and analyzing data streams from various perception modules.

Consider the implementation in Autoware: IRCA is adeptly configured to monitor Autoware's ROS2 topics, specifically from perception modules like the LiDAR detection module. This module, structured as a ROS2 node, publishes obstacle detection data to topics such as /perception/object_recognition/objects. Downstream modules, such as those responsible for tracking, subscribe to these data streams to continue the processing pipeline. Similarly, in the Apollo platform, the CyberRT framework utilizes Cyber Channels, which function akin to ROS2 topics, to enable runtime data stream monitoring.

When intervention is necessary, IRCA can dynamically instantiate a new ROS2 node to publish modified topics for targeted modules. To ensure a deterministic effect of interventions, we ensure that the replacement message maintains the same publish frequency as the original module's message in real time. Furthermore, we configure the communication to synchronous mode to guarantee synchronous communication between the simulator and the ADS. Lastly, by verifying the reproduction of the failures, we ensure that no other uncontrollable variables impact the perception results post-intervention. The original topics of these intervened modules are then dynamically replaced, and the configuration files automatically manage these changes. These files manage the launch parameters of each module, ensuring that updates are non-intrusive and lightweight yet effective in performing automated causal analysis.

V. Hierarchical Intervention Algorithm

This section introduces the Hierarchical Intervention Algorithm, which is the core method of IRCA.

Algorithm 1: Hierarchical Intervention Algorithm

Input: H-SCM $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Failure indicators F, F_c
Output: $\mathbb{P}$

if $\mathcal{G}$ *is a single chain* **then**
  $\mathcal{C} \leftarrow$ `NPI(`$\mathcal{G}, F$`)`
else
  $\mathcal{B} \leftarrow$ `BPI(`$\mathcal{G}, F_c$`) ;  /* Prune branches */`
  $\mathcal{C} \leftarrow$ `NPI(`$\mathcal{B}, F$`) ;  /* Prune nodes */`
end
$\mathbb{P} \leftarrow$ `Identify(`$\mathcal{C}, F$`) ;  /* fault modes */`

As shown in Algorithm1, the Hierarchical Intervention Algorithm is divided into two parts to address different structural complexities of H-SCM:

1) **Branch Pruning with Intervention (BPI):** BPI is employed to prune unnecessary branches and extract the primary causal path.
2) **Node Pruning with Intervention (NPI):** NPI is applied to further refine the chain by discarding non-causal modules.

Algorithm 2: Branch Pruning with Intervention BPI($\mathcal{G}, F_c$)

Input: H-SCM $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Fault indicator of the collider node c: F_c
Output: $\mathcal{B}$, $\mathcal{R}$

1 $\mathcal{R} \leftarrow \emptyset$; /* certain causal variables */
2 $\mathcal{U} \leftarrow \emptyset$; /* variables to be pruned */
3 $\mathcal{B} \leftarrow \emptyset$; /* causal path variables */
4 **for** *each collider node* c *in topological order* **do**
5 $P_a \leftarrow$ Parents(c) ; /* parent nodes */
6 $p^* \leftarrow \text{argmin}_{p \in Pa} |\text{Ancestors}(p, \mathcal{G})|$
7 $D_p^* \leftarrow$ Intervene(p^*)
8 $P_a \leftarrow P_a \setminus \{p^*\}$
9 **if** $F_c \notin D_p^*$ **then**
10 $\mathcal{U} \leftarrow \mathcal{U} \cup c$
11 $\mathcal{U} \leftarrow \mathcal{U} \cup \{P_a\} \cup \text{Ancestors}(P_a, \mathcal{V})$
12 **if** *Ancestors*$(p^*, \mathcal{V})$*is empty* **then**
13 $\mathcal{R} \leftarrow \mathcal{R} \cup \{p^*\}$
14 **else**
15 $\mathcal{B} \leftarrow \mathcal{B} \cup \{p^*\} \cup \text{Ancestors}(p^*, \mathcal{V})$
16 **end**
17 **else**
18 $\mathcal{R} \leftarrow \mathcal{R} \cup c$
19 $\mathcal{U} \leftarrow \mathcal{U} \cup \{p^*\} \cup \text{Ancestors}(p^*, \mathcal{V})$
20 $\mathcal{B} \leftarrow \mathcal{B} \cup \{P_a\} \cup \text{Ancestors}(P_a, \mathcal{V})$
21 **end**
22 $\mathcal{V} \leftarrow \mathcal{V} \setminus (\mathcal{U} \cup \mathcal{B} \cup \mathcal{R})$
23 **end**
24 $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{V}$
25 **return** $\mathcal{B}$, $\mathcal{R}$; /* No further intervention needed for $\mathcal{R}$ */

In cases where the H-SCM is comprised solely of a linear chain, the NPI algorithm is directly applied without the preliminary step of branch pruning.

A. Branch Pruning with Intervention (BPI)

The BPI algorithm is specifically designed to efficiently prune branches from the H-SCM and extract potential causal paths. It operates on the principle that a single causal path can be responsible for a failure. Algorithm 2 describes the process of BPI that involves pruning branches at colliders [19], following the topological order of the H-SCM.

1) Identifying Collider Nodes. In the H-SCM, colliders are nodes where multiple causal paths converge. They correspond to fusion nodes in the MSF, where outcomes from different branches are integrated, and play a crucial role in branch pruning. A collider is denoted as c, with its parent variables represented as Pa (line 5). In the example illustrated in Figure 3, there are two colliders: $C1$ (with parents $P1$ and $P2$) and $C2$ (with parent $P4$).

2) Prioritizing Intervention on the Shortest Branch. Subsequently, an intervention is performed on the parent node with the fewest ancestors in the H-SCM, which is denoted as p^* (line 6). Prioritizing p^* minimizes the complexity of the analyzed causal path, as it is closer to the final failure in the MSF. This intervention triggers a re-execution of the MSF, resulting in the return of a set of causal variables. In Figure 3, $P1$ and $P2$ have no ancestors, while $P4$ has one; therefore, $P1$ is selected as p^*, and an intervention is conducted on it.

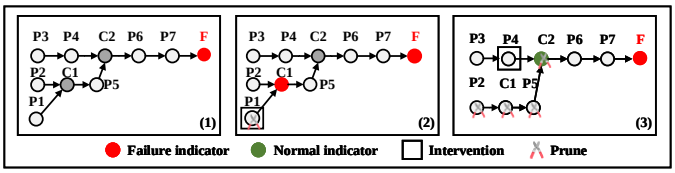


Fig. 3: A step-by-step illustration of BPI.

3) Analyzing Counterfactual Effects. After an intervention is performed, there are two possible outcomes:

Without Counterfactual Effects. If the counterfactual causal effect is not observed on the collider following an intervention on p^*, this indicates that p^* is not a cause of failure. Besides, any ancestors of p^* that are connected to the collider solely through p^* are also not causes. Therefore, the branch of p^* can be excluded from the H-SCM (line 19). The other branches are retained (line 20) for further processing by BPI. In Figure 3, $C1$ remains faulty after the intervention on $P1$, indicating that $P1$ is not a cause and will be pruned. Next, $P4$ will be selected as p^* for intervention.

With Counterfactual Effects. Conversely, if the counterfactual effect is observed on the collider, the branch of p^* can be identified as containing the causes. In this case, other branches can be pruned from the H-SCM (line 11). If the branch of p^* has only one node, it is labeled as causal (line 13). Otherwise, this branch is designated as a causal path (line 15) for further processing by NPI. In Figure 3, $C2$ becomes normal after the intervention on $P4$; thus, the branch of $P5$ is pruned, and the branch of $P4$ will be analyzed further using NPI.

4) Analyzing Collider Nodes. Given that a collider functions as a fusion node, its role is to select the outcome that most closely aligns with the truth. If an intervention with idealized substitution is applied to a parent node, but the collider node is still in the same state, this suggests that the fusion node is not working correctly and should be marked as causal (line 18). After processing all the branches connected to the colliders, the remaining variables are included to maintain them in the causal chain (line 24).

5) Finalizing Causal Chains. Finally, BPI returns candidate causal chains for the next process and some confirmed causal modules that do not require further processing. It ensures that the pruning process is efficient by focusing on intervention at nodes with the fewest ancestors and updating nodes for future operations based on the current state of the graph, thus avoiding redundant operations on nodes already decided upon.

B. Node Pruning with with Intervention (NPI)

The NPI algorithm is designed to identify and eliminate non-causal nodes within a single causal path by combining adaptive group testing with targeted interventions. It is robust in managing systems with multiple faults.

Algorithm 3: Node Pruning with Intervention NPI $(\mathcal{B}, F)$

Input: Branch $\mathcal{B} = (\mathcal{V}, \mathcal{E})$, Failure indicator F
Output: The set of causal modules $\mathcal{C}$

1 $\mathcal{C} \leftarrow \emptyset$; /* Causal variables set */
2 $\mathcal{U} \leftarrow \emptyset$; /* Non-causal variables set */
3 **while** $\mathcal{V} \neq \emptyset$ **do**
4 $\mathcal{V}_1 \leftarrow$ first half of $\mathcal{V}$ in topological order
$D^*_{v_1} \leftarrow$ Intervene$(\mathcal{V}_1)$; /* Intervention on the first half of $\mathcal{V}$ */
5 **if** $\exists v \in D^*_{v_1}$, *s.t.* F **then**
6 $\mathcal{C}, \mathcal{U} \leftarrow$ NPI$(\mathcal{V} \setminus \mathcal{V}_1, \mathcal{G}, F)$
7 $F \leftarrow (\mathcal{V} \setminus \mathcal{V}_1)[0]$
8 $\mathcal{C}', \mathcal{U}' \leftarrow$ NPI$(\mathcal{V}_1, \mathcal{G}, F)$
9 $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}', \mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{U}'$
10 **end**
11 **if** $\nexists v \in D^*_{v_1}$, *s.t.* F **then**
12 **if** $\mathcal{V}_1$ *contains only one causal variables* **then**
13 $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{V}_1$
14 **else**
15 $\mathcal{C}', \mathcal{U}' \leftarrow$ NPI$(\mathcal{V}_1, \mathcal{G}, F)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}', \mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{U}'$
16 **end**
17 **end**
18 **for** *each* $v' \in \mathcal{V} \setminus \mathcal{V}_1$ **do**
19 **if** $\exists v^* \in \mathcal{D}^*_{v_1}$, *such that* $(\neg v'_{v*} \wedge F) \vee (v'_{v^*} \wedge \neg F)$ **then**
20 $\mathcal{U} \leftarrow \mathcal{U} \cup \{v'\}$
21 **end**
22 **end**
23 $\mathcal{V} \leftarrow \mathcal{V} \setminus (\mathcal{C} \cup \mathcal{U})$
24 **end**
25 **return** $\mathcal{C}$

1) Dividing in Topological Order. NPI intervenes strategically in nodes based on their topological order. The original chain is divided into two smaller chains to facilitate focused interventions using a divide-and-conquer strategy. In the example illustrated in Figure 4, the chain is split into two: $(P3, P4)$ and $(P6, P7)$.

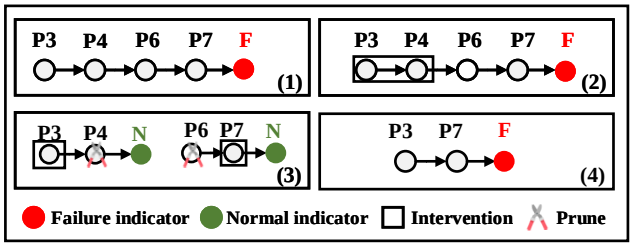


Fig. 4: A step-by-step illustration of NPI.

2) Analyzing the Right Chain. In each round, the procedure first applies interventions to all nodes in the left chain to restore their states to normal, thereby isolating the influence of the left chain and focusing on analyzing the right one. If the final outcome of the right chain turns normal, it indicates that the right chain contains no causes of failure and requires no further analysis. Conversely, if the failure persists, it suggests that some nodes in the right chain are causes, prompting NPI to be applied recursively to the right chain. This process systematically addresses components of the causal chain by breaking it down and intervening step by step (lines 5, 6). In Figure 4, the left chain $(P3, P4)$ is fully intervened, yet the failure in the right chain persists. Therefore, the right chain $(P6, P7)$ requires further analysis.

3) Analyzing the Left Chain. In the previous step, since all nodes in the left chain have been intervened to restore normal states, it remains unclear whether there are any causes of failure within that chain. Therefore, the left chain always requires further analysis by NPI. When analyzing the left chain, relying solely on the MSF's failure indicator (at the end of the right chain) is insufficient to determine the causal impact of the failure due to interactions with factors from the right chain. The existing solution, as outlined in [10], prunes the left chain without considering the concurrent impacts of both halves.

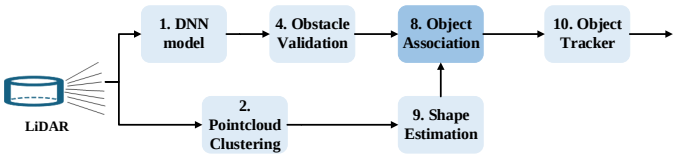
To address this limitation, we propose a modification by adding another failure indicator at the end of the left chain (line 7). Subsequently, NPI is applied to analyze the left chain (lines 8, 9). This approach ensures a comprehensive evaluation of the combined effects of multiple causal modules. In Figure 4, the left chain $(P3, P4)$ requires further analysis.

4) Conducting Recursive Analysis. As mentioned, further analysis will be conducted on the left chain and conditionally on the right chain (lines 11-15). After each round of intervention, NPI refines the causal path by removing nodes that have been definitively identified as either causal or spurious. It assesses each node that does not precede the intervened node, checking for any interventions that reveal a counterfactual violation linking the module to the failure. When such a violation is identified, the corresponding node is pruned (lines 18-23). The algorithm then proceeds to the next round of intervention, repeating this process until all potential causal nodes have been classified as either causal or spurious. Once all nodes have been evaluated, NPI produces a causal path composed of causal nodes.

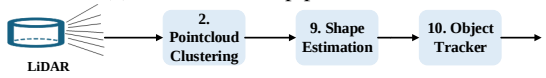
In Figure 4, the left chain is divided into $P3$ and $P4$; after intervening on $P3$, the output of the left chain becomes normal, identifying $P3$ as a cause and leading to the pruning of $P4$. The right chain is split into $P6$ and $P7$ for further analysis, with $P7$ eventually identified as a cause. Ultimately, NPI identifies $(P3, P7)$ as the causal path.

VI. Simulation Evaluation

We evaluate our IRCA by answering three research questions: **RQ1:** How effective is IRCA? **RQ2:** How efficient is IRCA? **RQ3:** Case study: What are the representative causes of the failures found?



(a) LiDAR-fusion pipeline DAG.



(b) Cluster pipeline DAG.

Fig. 5: DAGs illustrating the distinct pipeline configurations.

A. Experimental Setup

We employ Carla 0.9.14 as the simulation platform [23] to create the driving scenarios for the ADS. We use the Autoware Universe galactic as the ADS. Autoware is built on the Robot Operating System (ROS 2) and aims to enable the commercial deployment of autonomous vehicles across various platforms and applications [11]. The experiments are executed on two Precision 3660 Tower Workstations running Ubuntu 20.04, NVIDIA GeForce RTX 3080, and 64GB memory.

MSF configurations. Autoware's perception modules are divided into three main functional components: object segmentation, traffic light recognition, and object recognition. Our work focuses on the core functionality of object recognition within the MSF perception system. The object recognition component in Autoware is designed to identify dynamic objects in the surrounding environment of the ego vehicle in real time. It follows a tracking-by-detection pipeline architecture, with modules designed to be extensible and reusable, forming what is referred to as the micro autonomy architecture.

Our experiments contain three types of Autoware perception pipelines, as illustrated in Figure 1 and Figure 5. The "Camera-LiDAR-fusion" pipeline (Figure 1), combines data from the camera and LiDAR sensors to improve object detection capabilities. Initially, the pipeline integrates detections from the image-based YOLO model with point cloud segmented using the Euclidean Cluster (Pointcloud Clustering method), focusing on regions of interest (ROI). Subsequently, integrated data is further enhanced by merging it with results from the LiDAR CenterPoint (DNN model) detector. Finally, detections are merged again with clustering outputs. The "LiDAR-fusion" pipeline (Figure 5a), which is the default version, comprises two detectors based on LiDAR point cloud data. One detector is based on the LiDAR CenterPoint, and the other is on Euclidean Cluster. The "Cluster" pipeline (Figure 5b) solely relies on the Euclidean clustering branch, which is useful in replicating some failure scenarios.

B. Scenario Generation

Given the absence of an existing dataset to assess the underlying causes of failures in the MSF perception system, we are creating our own dataset, which will be made openly accessible. This dataset incorporates two sources of fault scenarios, ensuring both realism and completeness in testing.

Dataset 1: Real Fault Scenarios. We begin by gathering real fault scenarios reported to the Autoware Foundation [11] by other developers. The selection process involves two key steps: 1. Relevance: We filter issues related to our problem using the keyword "component: perception" and retain only those that have been officially acknowledged. 2. Reproducibility: We manually review each official issue-fix record to identify issues that are reproducible and cause perception failures. This process results in nine diverse groups of real faults, which we include in Dataset 1 without any subjective selection criteria, ensuring high realism in our testing cases.

In the MSF perception system, the fusion process consists of multiple paths that contribute to the final output. These causal paths are represented by different sets of modules, each exhibiting a specific fault mode (e.g., Missing Obstacle (MO)). We label these causal paths distinctly; for instance, the sets {(A, MO), (B, MO), (C, MO)} and {(X, MO), (Y, MO)} represent two causal paths, each contributing to an MSF failure. Detailed information about the real fault scenarios is provided in Table I. In this table, some groups correspond to multiple related issue IDs based on GitHub records.

Prevalence of "Missing Obstacle". Notably, all module fault modes in Groups 1 to 7 are MO, which may suggest a bias in the data. However, this accurately reflects the real-world distribution of faults: MO faults, which impair the system's ability to detect obstacles, are particularly critical as they can lead to serious hazards such as collisions. Consequently, these issues receive heightened attention and are prioritized for urgent resolution on GitHub. Our Dataset 1 aligns with this reality to ensure testing realism, while Dataset 2 (synthetic fault scenarios) will be used to enhance testing completeness.

Dataset 2: Synthetic Fault Scenarios. While real module faults reported to the Autoware Foundation focus primarily on MO, research [17] indicates that a broader range of faults also exists, including Ghost Obstacle (GO), Misclassification (MC), and Mislocalization (ML, a type of Parameter Error (PE)). To cover these faults and enhance testing completeness, we have created Dataset 2, which comprises synthetic fault scenarios.

These synthetic scenarios are generated by simulating various failure conditions through fault injection into the system. Specifically, we inject different faults at the object-level output of each module, as detailed in Section II-B. The injection process is automated, with failure-triggering cases undergoing manual review for validation. Scenarios are categorized into single-fault and multi-fault groups based on the number of injected faults. Details of the injected faults can be found in Table II.

For each group, we generate 100 failure scenarios by mutating scenario parameters. We define a specific object generation zone as outlined by Piazzoni et al. [17] to create scenarios that induce perception failures. The scenario parameters include both object-specific and environmental factors. Object-specific parameters cover details such as size, distance from the ego vehicle, and number of objects, while environmental parameters encompass weather and lighting conditions. By varying these parameters, we have generated a diverse array

TABLE I: Scenarios generated with real faults. Issue IDs are from the GitHub repo. Module IDs in curly braces {} for sets of modules in a causal path. Asterisk (*) signifies faults identified by our approach and confirmed by developers on GitHub.

Group ID	Issue ID	Module with Fault	Pipeline	Scenario Symptom	Failure Mode
1	#7106	{(1, MO)},{(2,ML), (8,MO)}	LiDAR-fusion	Not recognize large buses.	MO
2	#7563*	{(1,MO)},{(2,MO)}	LiDAR-fusion	Not recognize cyclists/pedestrians near a car.	MO
3	#4680, #5751	{(1,MO) (4*,MO)},{(2,MO), (9,MO)}, {(3,MO), (5,MO)}	Camera-LiDAR-fusion	Not recognize construction warning obstacles.	MO
4	#4681, #6938	{(1,MO) (4*,MO)},{(2,MO), (9,MO)}, {(3,MO), (5,MO)}	Camera-LiDAR-fusion	Not recognize traffic cones on road.	MO
5	#5148, #4949	{(1,MO) (4*,MO)}, {(2,MO), (9,MO)}, {(3,MO), (5,MO)}	Camera-LiDAR-fusion	Not recognize sparse objects on road.	MO
6	#4948	{(2,ML) (8,MO)}	Cluster	Not recognize trucks.	MO
7	#7860	{(8,MO)}	Cluster	Unidentifiable results for bus.	MO
*	#6936*, #6962	Planning	Camera-LiDAR-fusion	Driving hindered by mistaking uphill or downhill as an obstacle	GO
*	#6940	Planning	Camera-LiDAR-fusion	Emergency stop due to mistaking a roadside tree for an obstacle	GO

Module ID. 1: LiDAR CenterPoint Detector, 2: Euclidean Cluster, 3: YOLO Detector, 4: Obstacle Validation, 5: Roi Fusion, 6: Post Processing, 7: Object Merger, 8: Shape Estimation, 9: Object Association, 10: Object Tracker.

*. Groups 8 and 9 are not used for the evaluation experiments. Their root causes belong to the Planning component and will be introduced in the case study, where we use exclusion from the MSF component to determine them. We identified these issues through our scenario simulation and subsequently reported to GitHub.

Fault Mode. MO: Missing Obstacle, GO: Ghost Obstacle MC: Misclassification, ML: Mislocalization(a type of parameter error(PE), in terms of position parameter)

TABLE II: Synthetic scenarios generated with injected faults.

Type	Group ID	Module with fault	Pipeline	Failure Mode
Single Fault	8	(9, MO)	LiDAR-fusion	MO
	9	(9, GO)	LiDAR-fusion	GO
	10	(10, MO)	LiDAR-fusion	MO
	11	(10, GO)	LiDAR-fusion	GO
	12	(10, ML)	LiDAR-fusion	ML
	13	(10, MC)	LiDAR-fusion	MC
Multiple Faults	14	{(8, MO), (9, MO)}	LiDAR-fusion	MO
	15	{(8, ML), (9, MO)}	LiDAR-fusion	MO
	16	{(4, MO), (9, MO)}	LiDAR-fusion	MO
	17	{(1, MO), (9, MO)}	LiDAR-fusion	MO
	18	{(6, MO), (7, MO)}	Camera-LiDAR-fusion	MO
	19	{(6, GO), (7, GO)}	Camera-LiDAR-fusion	GO

Module ID and Fault Modes are the same as in Table I.

of scenarios, allowing for a comprehensive evaluation of the perception system under various challenging conditions.

Based on these datasets, we will conduct an extensive and thorough evaluation, including: 1) Autoware simulation in this section; 2) real vehicle tests with Autoware in Section VII; and 3) Apollo simulation in Section VIII.

C. Baselines

We compare our IRCA against the following baselines:

1) **DVCA** [9]: Driving violation cause analysis (DVCA) is the most recent method used in the ADS domain for causal analysis of driving violations. It utilizes binary search to pinpoint the faulty modules.
2) **AID** [10]: Adaptive interventional debugging (AID) is a method that combines statistical debugging and causal analysis to identify causal relationships.
3) **RCD** [6]: Root cause discovery (RCD) is not directly comparable to IRCA because RCD uses the Ψ-PC algorithm to learn the causal structure from data while IRCA constructs a causal graph with domain knowledge. For a fair comparison, we make RCD use the same causal graph constructed with domain knowledge as in IRCA. RCD returns the top-k causal factors, and we determine the value of k based on the output length of our method.

D. Evaluation Metric

We employ three commonly used metrics: Precision (P), Recall (R), and F1-score (F1), which are calculated by $P = TP/(TP + FP)$, $R = TP/(TP + FN)$, and $F1 = (2 \times P \times R)/(P + R)$, respectively, where TP represents True Positives, FP represents False Positives, and FN represents False Negatives. TP is the number of causal modules correctly identified as being in the ground truth causal path. FP is the number of non-causal modules incorrectly identified as in the causal path. FN is the number of unidentified causal modules in the ground truth.

Given that multiple causal paths can lead to failures, and addressing any of these paths may resolve the issue, we need to appropriately measure the solution's performance. To this end, we propose three distinct evaluation strategies for a comprehensive assessment:

1) **Best Match**: The solution's output is compared with each ground truth causal path to identify the one with the highest overlap or similarity. This strategy aims to find the most accurate alignment between the solution's output and the correct options in the ground truth, highlighting the best case in terms of prediction accuracy.

2) **Union**: This strategy considers all elements across the different ground truth causal paths, forming a union set. It provides insight into how well the solution performs in capturing all possible causal elements without missing any.

3) **Average**: This strategy calculates Precision, Recall, and F1-score for each solution's output and each ground truth path. The final scores are the averages of these individual scores. It offers a balanced view by assessing the solution's consistency across multiple correct answers.

E. *RQ1: Effectiveness on Real Fault Scenarios*

Best Match strategy. The performance results of four methods—IRCA (Ours), DVCA, AID, and RCD—for the Best Match strategy are listed in Table III. IRCA demonstrates superior performance across all metrics in real fault scenarios, achieving an average precision of 95.77%, recall of 95.07%, and an F1-score of 95.39%. These results demonstrate the effectiveness of IRCA in accurately identifying a complete causal path. In contrast, while AID and DVCA demonstrate commendable precision, their recall metrics are significantly

TABLE III: Evaluation results for Best Match on the real fault scenarios. Precision (P), Recall (R), and F1-score (F1) in Percentage (%). The best performance value is highlighted in **bold**.

Group ID	IRCA (Ours)			DVCA			AID			RCD		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
1	**97.94**	**95.00**	**96.45**	97.00	50.52	66.44	92.00	85.98	88.89	45.50	59.48	51.56
2	80.56	**87.00**	83.65	56.00	56.00	56.00	**87.00**	**87.00**	**87.00**	28.00	56.00	37.33
3	98.50	**98.50**	**98.50**	95.00	47.50	63.33	**100.00**	45.50	62.54	51.76	51.50	51.63
4	**99.48**	**95.00**	**97.19**	94.00	47.00	62.67	95.00	47.50	63.33	38.50	38.50	38.50
5	**97.46**	**96.00**	**96.73**	91.00	45.50	60.67	92.00	46.00	61.33	30.30	30.00	30.15
6	**97.44**	**95.00**	**96.20**	93.00	46.50	62.00	95.00	47.50	63.33	66.67	60.00	63.16
7	**99.00**	**99.00**	**99.00**	**99.00**	**99.00**	**99.00**	98.99	98.00	98.49	58.59	58.00	58.29
Avg.	**95.77**	**95.07**	**95.39**	89.29	56.00	67.16	94.28	65.35	74.99	45.62	50.50	47.23

lower. This suggests that although AID and DVCA can correctly identify elements of the causal path, they frequently overlook several causal modules, resulting in incomplete path detection. RCD shows the weakest performance, due to its lack of utilization of the branch relationships in the causal graph when clustering. These branch relationships provide structural information that forms a complete causal path.

AID achieves a relatively balanced performance in Groups 1 and 2. It is because these scenarios involve fewer causal modules, which simplifies the matching process and reduces the likelihood of missing causal modules. In Groups 3 through 6, the increased number of causal modules poses challenges, particularly for AID and DVCA. The complexity of these tasks affects the ability of AID and DVCA to maintain high recall, indicating difficulties in fully capturing the causal path.

Comparison under different evaluation strategies. Figure 6 presents the F1-scores of the four methods evaluated under different strategies (Best Match, Union, and Average) across three different DAG configurations: Camera-LiDAR-fusion, LiDAR-fusion, and Clustering. The reported F1-scores are the averages across all the real fault scenarios.

In the Camera-LiDAR-fusion DAG (Figure 6(a)), the IRCA method demonstrates the highest F1-scores across all strategies, with 97.47% for Best Match, 78.32% for Union, and 65.43% for Average. The drop from Best Match to Union and Average strategies indicates that while IRCA is highly effective at identifying complete causal paths, its performance diminishes as the evaluation strategy incorporates the need for broader causal path identification. DVCA, AID, and RCD follow a similar trend. For example, AID's performance also drops from 62.40% (Best Match) to 30.70% (Union) and 24.41% (Average). We can draw similar conclusions for the LiDAR-fusion DAG (Figure 6(b)). In the Cluster DAG (Figure 6(c)), all three evaluation strategies yield consistent results because this DAG only contains a single causal path. IRCA maintains consistent performance with an F1-score of 97.60% across all strategies. DVCA and AID also show stable performance, with F1-scores of 80.50% and 80.91%, respectively, while RCD remains consistent at 60.73%.

Overall, Figure 6 reveals the following insights:

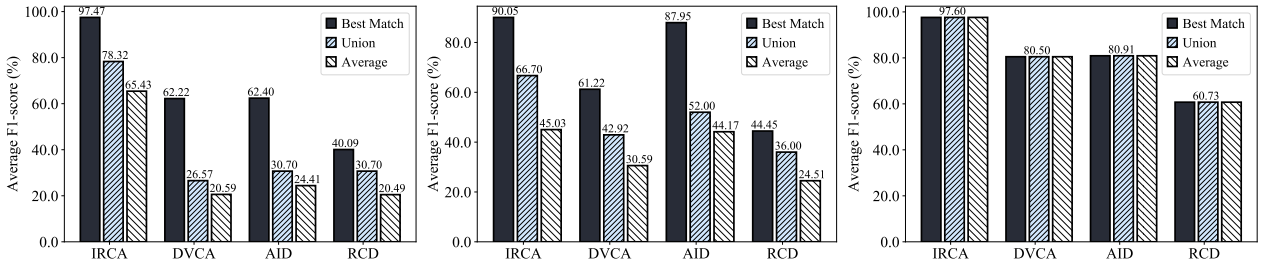
- All methods achieve their highest F1-scores under the Best Match strategy. This indicates that the methods are effective at identifying complete causal paths when the evaluation focuses on optimal path identification.
- The Union strategy introduces complexity by amalgamating modules from different causal paths, resulting in notable performance declines for all methods. The broader and more varied ground truth poses a greater challenge for methods initially designed to identify a single optimal path.
- The Average strategy consistently yields the lowest performance across all methods, highlighting the increased difficulty in assessing effectiveness across all potential causal paths. This strategy requires a more comprehensive identification process beyond the primary focus of single-path identification.
- **IRCA consistently outperforms other methods across all DAGs and evaluation strategies.**

*F. **RQ1:** Effectiveness on Synthetic Fault Scenarios*

Single injected fault. Table IV presents the performance metrics of four methods—IRCA (Ours), DVCA, AID, and RCD—evaluated on a single injected fault. IRCA demonstrates outstanding performance, showing the highest average recall (97.50%) and F1-score (96.73%) among the methods. A high F1-score suggests IRCA's strong capability in accurately identifying and confirming actual root causes. AID records the highest average precision (97.72%), indicating its conservative nature in root cause identification. AID tends to identify fewer potential causes than the actual number present, which ensures minimal false positives. DVCA features equal precision and recall values across all groups, which is a direct result of its binary search method for determining intervention targets. This method balances the identification of non-causal modules as causal (false positives) with the omission of actual causal modules (false negatives), resulting in consistent precision and recall. While RCD generally shows lower average metrics, it achieves the best precision in Group 13.

The results indicate that all methods are generally effective in identifying the singular root cause of a failure, which can be attributed to their utilization of counterfactual interventions.

Multiple injected faults. Table V presents performance metrics for the four methods evaluated on the scenarios with multiple injected faults. IRCA achieves the best results in recall and F1-score, with the F1-score being 12.97% higher than the second (AID). Although AID achieves the highest precision at 95.33%, it has notable limitations in recall. The



(a) Camera-LiDAR-fusion DAG (b) LiDAR-fusion DAG (c) Cluster DAG

Fig. 6: Comparison of IRCA (Ours), DVCA, AID and RCD on real fault scenarios using three evaluation strategies: Best Match, Union, and Average.

TABLE IV: Evaluation results on the synthetic fault scenarios (single fault).

Group ID	IRCA (Ours)			DVCA			AID			RCD		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
8	**97.09**	**100.00**	**98.52**	92.00	92.00	92.00	92.00	92.00	92.00	67.35	66.00	66.67
9	96.70	88.00	**92.15**	77.00	77.00	77.00	**97.33**	73.00	83.43	90.14	**90.14**	90.14
10	99.01	**100.00**	**99.50**	95.00	95.00	95.00	**100.00**	97.00	98.48	92.13	82.00	86.77
11	95.10	**97.00**	96.04	95.00	95.00	95.00	**97.00**	**97.00**	**97.00**	92.93	92.00	92.46
12	95.24	**100.00**	97.56	98.00	98.00	98.00	**100.00**	96.00	**97.96**	91.92	91.00	91.46
13	93.46	**100.00**	**96.62**	90.00	90.00	90.00	**100.00**	93.00	96.37	**100.00**	91.00	95.29
Avg.	96.10	**97.50**	**96.73**	91.17	91.17	91.17	**97.72**	91.33	94.21	89.08	85.36	87.13

TABLE V: Evaluation results on the synthetic fault scenarios (multiple faults).

Group ID	IRCA (Ours)			DVCA			AID			RCD		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
14	88.53	**96.50**	**92.34**	91.00	45.50	60.67	**100.00**	50.00	66.67	98.04	50.00	66.23
15	69.66	**81.50**	**75.12**	97.00	48.50	64.67	**100.00**	50.00	66.67	83.87	65.00	73.24
16	96.04	**97.00**	**96.52**	**100.00**	50.00	66.67	76.00	38.00	50.67	50.52	48.50	49.49
17	98.34	**89.00**	**93.44**	90.00	45.00	60.00	**100.00**	50.00	66.67	46.51	30.00	36.47
18	94.29	**99.00**	**96.59**	71.00	35.50	47.33	**98.00**	49.00	65.33	78.67	59.00	67.43
19	79.17	**76.00**	**77.55**	67.00	33.50	44.67	**98.00**	49.00	65.33	96.08	49.00	64.90
Avg.	87.67	**89.83**	**88.59**	86.00	43.00	57.34	**95.33**	47.67	63.56	75.62	50.25	59.63

overall performance trend is consistent with that observed in the single-fault scenarios, although the recall for AID and DVCA becomes significantly lower in the multi-fault scenarios.

DVCA's approach involves a binary group search, which stops as soon as it identifies what it perceives as the ground truth. This explains its lower recall, as it terminate the search without discovering all pertinent faults. AID, designed to identify every causal module along the causal path, suffers from an overly aggressive pruning rule that fails to account for situations where multiple causal modules contribute to a failure, leading to an excessive exclusion of relevant modules. RCD's performance varies widely across groups. For instance, its F1-score in Group 17 is 36.47% lower than in Group 15, indicating inconsistent effectiveness across different scenarios.

Overall, the performance for the multi-fault scenarios is lower than that of the single-fault scenarios. Nevertheless, the performance of our method, IRCA, remains robust with an F1-score of 88.59%, demonstrating its effectiveness in more complex fault scenarios.

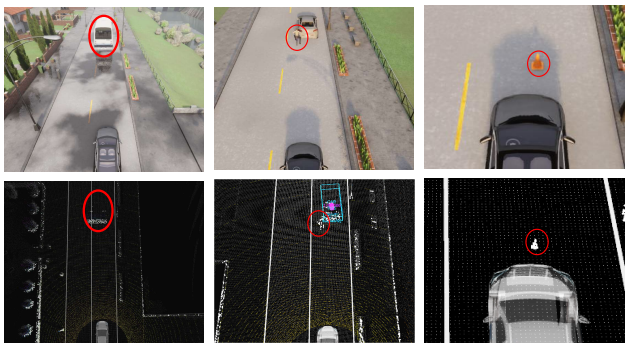
G. ***RQ2:*** *Efficiency*

TABLE VI: Results of the average intervention numbers and execution time(minutes).

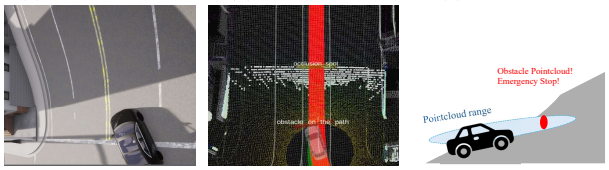
Method	Real Faults		Single Fault		Multiple Faults	
	Intervention	Time	Intervention	Time	Intervention	Time
IRCA	2.99	4.67	**0.96**	**1.58**	2.51	3.91
DVCA	**2.29**	**3.60**	1.03	1.62	**1.79**	**2.72**
AID	3.34	5.07	1.03	1.62	1.84	2.79
RCD	3.03	4.82	1.04	1.64	2.55	3.97

In assessing the efficiency of the IRCA, we utilize two primary metrics: the number of interventions and the execution time. As detailed in Table VI, we compare these metrics across various scenarios—including real faults, and both single and multiple injected faults—across four different methods.

For scenarios involving a single fault, IRCA demonstrates superior efficiency, requiring the fewest interventions (an average of 0.96) and the shortest execution time (1.58 minutes). For real-fault scenarios, although IRCA does not achieve the lowest intervention count or fastest execution time—recording



(a) Issue #7106 (b) Issue #7563 (c) Issue #6938



(d) Issue #6936. The illustration briefly describes the situation.

Fig. 7: Notable cases of perception failure scenarios: Subfigures (a)-(c) (top): snapshots from the Carla simulator illustrating different failure scenarios. Subfigures (a)-(c) (bottom): corresponding snapshots perceived by the ADS. Subfigure (d): snapshots of Issue #6936: snapshot from Carla (left), corresponding snapshot perceived by the ADS (middle), and an illustration of the situation (right).

2.99 interventions over 4.67 minutes, slightly higher than DVCA's 2.29 interventions in 3.6 minutes—it demonstrated greater overall effectiveness. This indicates that IRCA while taking marginally longer, provides more comprehensive diagnostics. In multiple fault scenarios, IRCA's capabilities are particularly notable. Despite a modest increase in execution time compared to DVCA (3.91 minutes for IRCA vs. 2.72 minutes for DVCA), IRCA supports concurrently diagnosing multiple root causes. This functionality significantly advances over other methods, such as DVCA, AID, and RCD, which exhibit limitations in handling complex diagnostic challenges.

The experimental results support IRCA's ability to swiftly and accurately identify the causal modules responsible for system failures, underscoring its practical utility in multi-sensor fusion perception systems.

H. *RQ3: Case Study*

We illustrate several notable cases of perception failures in real fault scenarios. We also discuss the generality of our approach on Apollo.

Issue #7106 (shown in Figure 7a) This issue arises from two causal paths. In the LiDAR CenterPoint detector branch, the model fails to detect a truck. Concurrently, in the cluster branch, an initial wrong clustering of the point cloud by the Euclidean cluster, followed by errors in the shape estimation module, leads to incorrect vehicle shape refinement. As a result, a truck that is around 4 meters in length in front of the ego vehicle is not detected. The corresponding pull request (PR) is that the shape estimation module needs a fix since the maximum size filter for trucks was previously set too low.

Issue #7563 (shown in Figure 7b) There are two causal paths that contribute to this failure. The LiDAR CenterPoint detector does not recognize a cyclist due to its overlap with a vehicle's point cloud. Meanwhile, the Euclidean clustering method on another branch inaccurately merges the cyclist and a nearby vehicle into a single obstacle. Thus, the cyclist near the vehicle is incorrectly identified as part of the vehicle.

Issue #6938 (shown in Figure 7c) This failure is caused through three causal paths within the Camera-LiDAR-fusion DAG. (1) LiDAR CenterPoint Detector Branch: the detector occasionally fails to identify traffic cones due to sparse LiDAR point clouds. Meanwhile, the obstacle validation module mistakenly filters out the detected traffic cone. This filtering issue has been identified using our IRCA and addressed in a fix submitted to GitHub. (2) Cluster Branch: the Euclidean Cluster detector has difficulty clustering the traffic cone with fewer point clouds. Furthermore, the objects identified by the Euclidean Cluster detector are assigned a lower priority during object association. This can lead to inaccurate results, even when the Euclidean Cluster correctly identifies the obstacle. (3) Region of Interest (ROI) Fusion Branch: The ROI cluster fusion criteria require both point cloud data and ROIs detected by the camera for successful operation. If the obstacle is detected solely by the camera, the fusion process fails. This issue was addressed by introducing a pseudo dataset for the YOLO detector specifically designed to improve traffic cone detection. Additionally, the refinement for ROI fusion now considers only the distance to the camera to enhance the accuracy of this process.

Issue #6936 (shown in Figure 7d) This issue involves the vehicle treating the uphill ramp as an obstacle and becoming stuck in front of the ramp. IRCA can find that object recognition in the perception system is not the root cause of this issue. Further analysis reveals that the planning component is inadvertently subscribing to both the final output from the perception component and the unfiltered point cloud message. This subscription to the unfiltered point cloud introduces ghost obstacles into the planning component, precipitating the issues. Notably, **this issue was first identified through our scenario simulation and reported to GitHub. Subsequently, the issue was confirmed by developers on GitHub**.

VII. Real-world Experiments

A. Experimental Settings

Testbed. In real-world experiments, a testbed car, as illustrated in Figure 8, serves as the ego vehicle. Its MSF perception system contains a LiDAR and a camera. A Robosense Helios-32 LiDAR, featuring 32 beams and a 10 Hz frame rate, is mounted at the top front center at a height of 1.9 meters from the ground. A high-resolution camera equipped with a Sony IMX335 image sensor is positioned at the front center of the car, 1.6 meters above the ground. This camera captures images at a resolution of 1920 × 1080. The ADS installed on the car is Autoware.Universe, which runs on Ubuntu 22.04

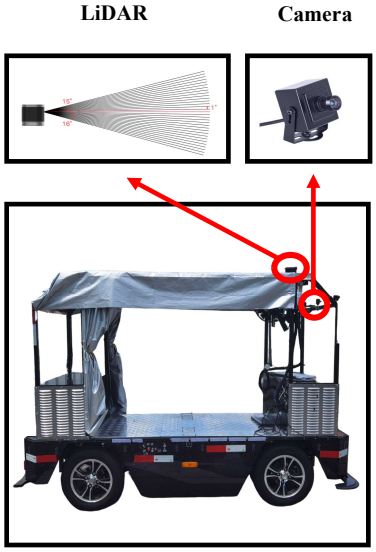
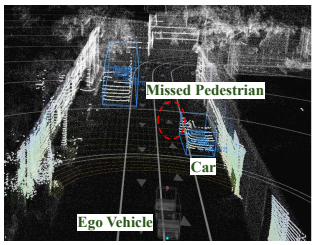
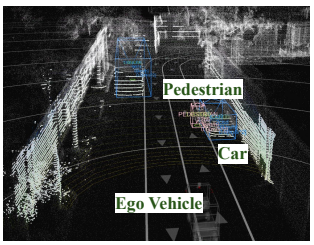


Fig. 8: A testbed car equipped with a LiDAR (32-Beam even distribution) and a Camera.

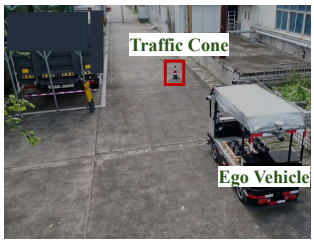
(a) Physical scenario: a pedestrian close to a car.



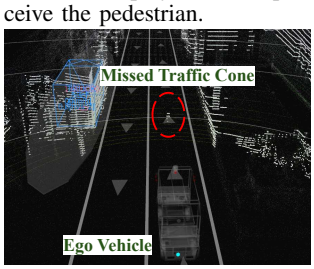
(b) ADS display: fails to perceive the pedestrian.



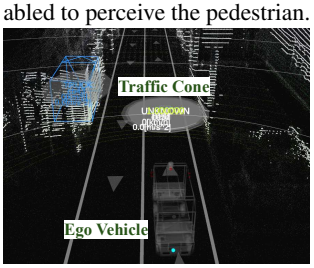
(c) Counterfactual effect: enabled to perceive the pedestrian.



(d) Physical scenario: a traffic cone ahead of the ego vehicle.



(e) ADS display: fails to perceive the traffic cone.



(f) Counterfactual effect: enables to perceive the traffic cone.

Fig. 9: Two real-world scenarios.

with ROS2 Humble as the middleware. The modules of MSF perception system utilized in these physical tests are consistent with those used in simulation-based experiments.

B. Field Test Scenarios

We let the car drive autonomously in a small community and capture perception failures by comparing the objects detected by the MSF perception system with the ground truth observed by the human onsite. Two perception failures were recorded during the field test. The failures were similar to those reported in Issues #7563 and #6938 in Table I, respectively.

Missed pedestrian adjacent to a car. This scenario, shown in Figure 9a, involved a pedestrian who was in the same lane as the ego vehicle, approximately 13 meters ahead, and very close to a stationary black Nissan. A truck was stationary 25 meters ahead on the left side of the lane, but it is not relevant to the targeted failure being analyzed. In this scenario, the MSF perception system employs a LiDAR-fusion pipeline, which integrates the detection results from the Euclidean clustering detector and the LiDAR CenterPoint detector.

Missed traffic cone. In this scenario, shown in Figure 9d, the system failed to detect a traffic cone on the road. The cone, measuring 0.7 meters in height and 0.3 meters in base width, was directly positioned in the lane, 10 meters ahead of the ego vehicle. A truck is present in the same lane, positioned 12 meters to the left of the ego vehicle, but it did not impact this specific failure. The MSF perception system utilizes a Camera-LiDAR-fusion pipeline that combines the detector results from the Euclidean Cluster, the LiDAR CenterPoint detector, and the YOLO detector.

C. Root Cause Analysis with IRCA

Missed pedestrian adjacent to a car. This scenario illustrates a perception failure where a pedestrian adjacent to a car is not detected (see Figure 9b). With IRCA, we pinpointed one causal path contributing to this failure, which involves the Euclidean cluster module suffering from a missing obstacle fault mode. After localizing this root cause module, we conducted an intervention on the Euclidean clustering module. As shown in Figure 9c, the counterfactual effect demonstrates that the pedestrian is successfully detected and tracked, indicating that correcting the clustering-based module can effectively remove the perception failure.

Missed traffic cone. Figure 9e illustrates the perception failure of a traffic cone. Using IRCA, we determined two causal paths: The first causal path included the LiDAR CenterPoint module and the obstacle validation module, both suffering from a missing obstacle fault. We then performed a counterfactual intervention on the LiDAR Centerpoint and Obstacle validation modules, and the final output of MSF (tracked module) perceived the traffic cone as shown in Figure 9f. Note that the traffic cone label was “unknown” in the Autoware [24]. The result indicates that rectifying issues within the LiDAR CenterPoint and the obstacle validation modules can eliminate the perception failure.

These real-world experiments demonstrate that IRCA can effectively identify the root causes of MSF perception failures and provide valid causal paths for resolving the failures.

VIII. Cross-Platform Evaluation

This evaluation aims to validate that our IRCA solution can operate on various ADS beyond Autoware. Specifically, we expand our assessment to include Apollo, another mainstream

(a) Missed Pedestrian (Camera-LiDAR-Fusion) (b) Missed Shopping Cart (Camera-Tracking) (c) Misclassified Car (LiDAR-Tracking) (d) Ghost Object (LiDAR-Tracking)

Fig. 10: Apollo scenarios: Subfigures (a)-(d) (top): snapshots from Apollo DreamView (Visualization Application) illustrating different failure scenarios (left) and counterfactual effects (right). Subfigures (a)-(d) (bottom): images from the camera.

platform. Since our solution has been thoroughly tested and compared with baselines in Section VI, this section conducts smaller-scale experiments, which are sufficient to demonstrate cross-platform usability.

A. Experimental Setup

The experiments are conducted using Apollo version 9.0 on the same workstation as in Section VI. Our experimental configurations involve three Apollo perception pipelines for obstacle detection to evaluate the adaptability of IRCA: the default Camera-LiDAR fusion pipeline [25], the LiDAR pipeline, and the Camera pipeline.

B. Test Scenarios

TABLE VII: Evaluation Scenarios for IRCA on Apollo

ID	Fault Type	Pipeline	Module with fault	Scenario Symptom	Failure Mode
1	real	Camera-Lidar-Fusion	{(4, MO), (7: MO)}	No shopping cart detected	MO
2	real	Camera-Lidar-Fusion	{(4, MO), (7: MO)}	Pedestrian not recognized	MO
3	real	Camera-Lidar-Fusion	{(7:MO)}	Cars not perceived	MO
4	real	Camera-Track	{(4, MO)}	No shopping cart detected	MO
5	Injected	Camera-Track	{(5, GO)}	Ghost car on the road	GO
6	Injected	Camera-Track	{(6, ML)}	Car mislocated	ML
7	real	Lidar-Track	{(2, MC)}	Car misclassified	MC
8	real	Lidar-Track	{(1, GO)}	Ghost car beside the road	GO
9	Injected	Lidar-Track	{(2, MO)}	Car not recognized	MO
10	Injected	Lidar-Track	{(3, ML)}	Car mislocated	ML

Module ID: 1:LiDAR Detector 2: LiDAR Detection Filter 3:LiDAR Tracking 4:YOLOX 3D Detector 5:Camera Location Refinement 6: Camera Tracking 7:Multi-Sensor Fusion

The driving scenarios selected for our study are based on the recorded data package provided by Apollo. As summarized in Table VII, these scenarios are crucial for evaluating the effectiveness and cross-platform usability of our IRCA. First, our verification focuses on identifying actual fault types within the modules, as documented in the Apollo GitHub repository. We find that the module faults reported by Apollo are consistent with those in Autoware. Second, we inject additional fault cases not originally recorded to enhance the diversity of the test scenarios and ensure a comprehensive evaluation of potential faults. Specifically, we inject missing obstacles (MO) into the LiDAR detection filter module, mislocalization (ML) faults in both LiDAR and camera tracking, and ghost obstacles (GO) in camera location refinement.

C. Root Cause Analysis with IRCA

Our IRCA can effectively identify all faulty modules as listed in Table VII. This demonstrates the applicability of our method to Apollo. To further discuss the performance of IRCA on Apollo, Figure 10 presents scenarios from the Apollo DreamView visualization application. These images show typical failure scenarios on the left and their counterfactual interventions on the right, with corresponding views captured directly from the camera.

Missed pedestrian. In this scenario, the perception configuration of Apollo uses a Camera-LiDAR-Fusion pipeline. Figure 10a (top left) displays the Apollo DreamView of perception results, including a clear LiDAR point cloud of two pedestrians. Although the camera image captures the pedestrians, the system does not detect them, illustrating a missed obstacle failure scenario. The right side of Figure 10a demonstrates the counterfactual effect after interventions on the YOLOX 3D detector and multi-sensor fusion modules, resulting in successful detection and tracking of the pedestrians. It is important to note that intervening only on the YOLOX 3D detector module does not resolve the failure due to a Missing Obstacle fault in the multi-sensor fusion module, disregarding detection results from upstream modules. This fault is also documented in a GitHub issue.

Missed shopping cart. The perception configuration employs a Camera-Tracking pipeline. Figure 10b (top left) shows the perception results and the camera image below it, where a shopping cart in front of the ego vehicle is visible but not detected in Apollo DreamView. Figure 10b also illustrates the counterfactual effect of an intervention on the YOLOX 3D detector module, pinpointing the root cause of this missed detection.

Misclassified car. In this scenario, Apollo uses a LiDAR-Tracking configuration, Figure 10c (top left) depicts a misclassified car failure. The car on the right side ahead of the ego vehicle is incorrectly recognized as an unknown object, a result of a fault in the LiDAR Detection Filter module, which mislabeled the data received from the LiDAR Detector module. Figure 10c (top right) illustrates the counterfactual

effects after intervening on the LiDAR Detection Filter, where the car is correctly recognized.

Ghost object. Also under a LiDAR-Tracking setup, Figure 10d shows a ghost object failure scenario. The image shows an empty road ahead of the ego vehicle, yet the perception result on Apollo DreamView reports unknown obstacles. The right side of Figure 10d shows the counterfactual effects following interventions on the LiDAR Detection module, effectively eliminating the ghost obstacle.

These experiments demonstrate that IRCA can effectively identify the root causes of various perception failures in the Apollo system. From the above cross-platform evaluation, it is evident that IRCA can conduct root cause analysis across different systems, highlighting its broad applicability.

IX. Related Work

Perception failure identification. Most existing research is centered around testing methods to detect the failure in MSF perception systems by creating diverse and challenging test scenarios [26]–[29]. For example, MultiTest [16] is a fitness-guided metamorphic testing method that synthesizes and positions realistic multi-modal object instances in background images and point clouds to create diverse test cases. Another related research area is fault detection with anomaly detection or runtime verification. Researchers are increasingly turning to machine learning-based anomaly detection methods to pinpoint the aberrant modules potentially responsible for perception failures [30]–[33]. For example, Rahman et al. proposed an FSNet model [34] to identify misclassifications in semantic segmentation. The combination of runtime verification and model-checking techniques has been deployed to trace module anomalies that might be linked to broader system malfunctions [35], [36]. Specifically, probabilistic model checking leverages transition probabilities from confusion matrices to ensure modules adhere to the system's formal specifications defined in temporal logic [37]–[40]. Antonante et al. [41] formalized runtime fault detection and identification in perception systems, using a probabilistic diagnostic graph. Compared to our technique, testing and fault detection methods are less effective in explaining failures, since they produce a ranked candidate set [42] but do not dive into the underlying causes of these faults. Therefore, identifying the root causes of perception failures in ADS is still a problem yet to be solved.

Causal analysis for systems. RCA has been extensively applied across various domains to identify underlying causes of system failures, particularly in microservices and configurable robotic systems. In the domain of microservices, RCA is used to diagnose service failures by analyzing data collected through various monitoring tools [43]–[45]. These methods leverage key performance indicators (KPIs), traces, and logs to detect anomalies and pinpoint their root causes. These methods are integral for maintaining system reliability and performance, as they allow for timely identification and resolution of issues that could potentially lead to system downtime or degraded service. Specifically targeting system configurations, the Unicorn framework designed by Iqbal et al. [46] and the CARE proposed by Hossen et al. [47] offer insights into how different settings impact system abnormal performance and behavior. Unicorn applies causal reasoning to determine an optimal system configuration under various operational scenarios, while CARE uses causal learning to correlate configuration parameters with abnormal behaviors in configurable robots. However, both approaches are not suitable for RCA of perception failures because they are developed for handling multi-dimensional system metrics data, which differ significantly from MSF perception object data.

Causality testing in ADS. In the field of ADS, recent studies have integrated causal analysis with system testing to detect driving violations [8], [9]. For example, CART [48] introduces a novel framework that treats testing as a causal reasoning process. This approach aims to uncover the causal relationships between test inputs and outputs, focusing on how different driving scenarios affect component-level performance. Zhong et al. [49] apply counterfactual causality analysis to verify fusion errors in MSF. However, their research is limited to fusion errors associated with driving violations and does not consider other causal modules within the MSF perception system. None of the aforementioned work has addressed the causal relations between modular faults and perception failures in ADS. When considering causal modules in MSF, the complexity increases due to a larger DAG and specific structures like colliders. Moreover, these studies typically consider only a single causal factor without accounting for the potential interactions among multiple concurrent causal modules. IRCA fills this gap.

X. Conclusion

We designed and implemented a novel methodology, interventional root cause analysis (IRCA), which utilizes a strategic combination of runtime monitoring, causal analysis, and scenario testing to identify the root causes of failures within the MSF perception of autonomous driving systems (ADS). IRCA has three distinctive features: (1) it can track multiple concurrent root causes, (2) it can pinpoint both faulty modules and the specific fault modes within these modules, and (3) it can reveal complex causal paths and the interactions among multiple faults. Collectively, these capabilities provide a comprehensive view of the root causes of perception failures, enabling developers to isolate and resolve the issues rapidly.

We assessed the effectiveness of IRCA using the open-source ADS platform, Autoware. Our evaluation incorporated both real-world issues documented on GitHub and synthetic scenarios with injected faults across various modules. The results indicate that IRCA consistently surpasses baselines and achieves superior performance across different root causal modules. Furthermore, we demonstrated IRCA's real-world applicability by testing it on an operational testbed. Additionally, we showcased the generality of our approach through a cross-platform evaluation on Apollo. This research significantly advances the safety of ADS through the novel application of IRCA and its proven effectiveness in practical settings.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their valuable comments and suggestions. The project is supported in part by C1042-23G, a project from Hong Kong Research Grant Council under Collaborative Research Fund scheme.

REFERENCES

[1] Waymo Team, "Waymo's next chapter in san francisco," https://waymo.com/blog/2023/08/waymos-next-chapter-in-san-francisco/, 2023.

[2] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 287–296.

[3] S. Kochanthara, T. Singh, A. Forrai, and L. Cleophas, "Safety of perception systems for automated driving: A case study on apollo," *ACM Trans. Softw. Eng. Methodol.*, nov 2023. [Online]. Available: https://doi.org/10.1145/3631969

[4] X. Gao, Z. Wang, Y. Feng, L. Ma, Z. Chen, and B. Xu, "Benchmarking robustness of ai-enabled multi-sensor fusion systems: Challenges and opportunities," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 871–882. [Online]. Available: https://doi.org/10.1145/3611643.3616278

[5] J. Tu, H. Li, X. Yan, M. Ren, Y. Chen, M. Liang, E. Bitar, E. Yumer, and R. Urtasun, "Exploring adversarial robustness of multi-sensor perception systems in self driving," in *Proceedings of the 5th Conference on Robot Learning*, vol. 164, 08–11 Nov 2022, pp. 1013–1024. [Online]. Available: https://proceedings.mlr.press/v164/tu22a.html

[6] A. Ikram, S. Chakraborty, S. Mitra, S. Saini, S. Bagchi, and M. Kocaoglu, "Root cause analysis of failures in microservices through causal discovery," in *Advances in Neural Information Processing Systems*, vol. 35. Curran Associates, Inc., 2022, pp. 31 158–31 170. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/c9fcd02e6445c7dfbad6986abee53d0d-Paper-Conference.pdf

[7] Y. Zhang, Z. Guan, H. Qian, L. Xu, H. Liu, Q. Wen, L. Sun, J. Jiang, L. Fan, and M. Ke, "Cloudrca: A root cause analysis framework for cloud computing platforms," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, ser. CIKM '21, 2021, p. 4373–4382. [Online]. Available: https://doi.org/10.1145/3459637.3481903

[8] H. Sun, C. M. Poskitt, Y. Sun, J. Sun, and Y. Chen, "Acav: A framework for automatic causality analysis in autonomous vehicle accident recordings," *arXiv preprint arXiv:2401.07063*, 2024.

[9] Z. Wan, Y. Huai, Y. Chen, J. Garcia, and Q. A. Chen, "Towards automated driving violation cause analysis in scenario-based testing for autonomous driving systems," *arXiv preprint arXiv:2401.10443*, 2024.

[10] A. Fariha, S. Nath, and A. Meliou, "Causality-guided adaptive interventional debugging," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 431–446.

[11] Autoware Foundation, "Autoware: The world's leading open-source software project for autonomous driving." https://github.com/autowarefoundation/autoware, 2024.

[12] X. Zhang, Y. Gong, J. Lu, J. Wu, Z. Li, D. Jin, and J. Li, "Multimodal fusion technology based on vehicle information: A survey," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 6, pp. 3605–3619, 2023.

[13] Y. Li, D. K. Jha, A. Ray, and T. A. Wettergren, "Feature level sensor fusion for target detection in dynamic environments," in *2015 American Control Conference (ACC)*. IEEE, 2015, pp. 2433–2438.

[14] J. H. Yoo, Y. Kim, J. Kim, and J. W. Choi, "3d-cvf: Generating joint camera and lidar features using cross-view spatial feature fusion for 3d object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[15] Baidu Apollo, "Apollo: Open source autonomous driving." https://github.com/ApolloAuto/apollo, 2024.

[16] X. Gao, Z. Wang, Y. Feng, L. Ma, Z. Chen, and B. Xu, "Multitest: Physical-aware object insertion for testing multi-sensor fusion perception systems," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639191

[17] A. Piazzoni, J. Cherian, J. Dauwels, and L.-P. Chau, "Pem: Perception error model for virtual testing of autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 1, pp. 670–681, 2024.

[18] J. Pearl, "Causal inference," *Causality: objectives and assessment*, pp. 39–58, 2010.

[19] J. Pearl, M. Glymour, and N. P. Jewell, *Causal inference in statistics: A primer*. John Wiley & Sons, 2016.

[20] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 05, pp. 507–525, may 2015.

[21] "Ros2 topics," https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html.

[22] "Cyber rt documentation," https://cyber-rt.readthedocs.io/en/latest/.

[23] "CARLA Simulator: Open Source Simulator for Autonomous Driving Research," https://carla.readthedocs.io/en/latest/.

[24] Autoware Foundation, "ObjectClassification Message in autoware_msgs/autoware_perception_msgs," https://github.com/autowarefoundation/autoware_msgs/blob/main/autoware_perception_msgs/msg/ObjectClassification.msg.

[25] "Apollo perception," https://github.com/ApolloAuto/apollo.

[26] G. Christian, T. Woodlief, and S. Elbaum, "Generating realistic and diverse tests for lidar-based perception systems," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 2604–2616.

[27] Y. Cao, N. Wang, C. Xiao, D. Yang, J. Fang, R. Yang, Q. A. Chen, M. Liu, and B. Li, "Invisible for both Camera and LiDAR: Security of Multi-Sensor Fusion based Perception in Autonomous Driving Under Physical World Attacks," in *Proceedings of the 42nd IEEE Symposium on Security and Privacy (IEEE S&P 2021)*, May 2021.

[28] J. Shen, J. Y. Won, Z. Chen, and Q. A. Chen, "Drift with devil: Security of Multi-Sensor fusion based localization in High-Level autonomous driving under GPS spoofing," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 931–948. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/shen

[29] T. Laurent, S. Klikovits, P. Arcaini, F. Ishikawa, and A. Ventresque, "Parameter coverage for testing of autonomous driving systems under uncertainty," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, apr 2023. [Online]. Available: https://doi.org/10.1145/3550270

[30] V. Besnier, A. Bursuc, D. Picard, and A. Briot, "Triggering failures: Out-of-distribution detection by learning from local adversarial attacks in semantic segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 15 701–15 710.

[31] Z. Li, Y. Zhu, and M. Van Leeuwen, "A survey on explainable anomaly detection," *ACM Trans. Knowl. Discov. Data*, vol. 18, no. 1, sep 2023. [Online]. Available: https://doi.org/10.1145/3609333

[32] E. Khalastchi and M. Kalech, "On fault detection and diagnosis in robotic systems," *ACM Comput. Surv.*, vol. 51, no. 1, jan 2018. [Online]. Available: https://doi.org/10.1145/3146389

[33] T. Ji, A. N. Sivakumar, G. Chowdhary, and K. Driggs-Campbell, "Proactive anomaly detection for robot navigation with multi-sensor fusion," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4975–4982, 2022.

[34] Q. M. Rahman, N. Sünderhauf, P. Corke, and F. Dayoub, "Fsnet: A failure detection framework for semantic segmentation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3030–3037, 2022.

[35] D. Wang, W. Dou, Y. Gao, C. Wu, J. Wei, and T. Huang, "Model checking guided testing for distributed systems," in *Proceedings of the Eighteenth European Conference on Computer Systems*, ser. EuroSys '23, 2023, p. 127–143. [Online]. Available: https://doi.org/10.1145/3552326.3587442

[36] E. Zapridou, E. Bartocci, and P. Katsaros, "Runtime verification of autonomous driving systems in carla," in *International Conference on Runtime Verification*. Springer, 2020, pp. 172–183.

[37] A. Balakrishnan, J. Deshmukh, B. Hoxha, T. Yamaguchi, and G. Fainekos, "Percemon: online monitoring for perception systems," in *Runtime Verification: 21st International Conference, RV 2021, Virtual Event, October 11–14, 2021, Proceedings 21*. Springer, 2021, pp. 297–308.

[38] A. Badithela, T. Wongpiromsarn, and R. M. Murray, "Evaluation metrics of object detection for quantitative system-level analysis of safety-critical

autonomous systems," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 8651–8658.
[39] P. Antonante, S. Veer, K. Leung, X. Weng, L. Carlone, and M. Pavone, "Task-aware risk estimation of perception failures for autonomous vehicles," *arXiv preprint arXiv:2305.01870*, 2023.
[40] M. S. Ramanagopal, C. Anderson, R. Vasudevan, and M. Johnson-Roberson, "Failing to learn: Autonomously identifying perception failures for self-driving cars," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3860–3867, 2018.
[41] P. Antonante, H. G. Nilsen, and L. Carlone, "Monitoring of perception systems: Deterministic, probabilistic, and learning-based fault detection and identification," *Artificial Intelligence*, vol. 325, p. 103998, 2023.
[42] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. U. Jain, and M. Stumm, "Simple testing can prevent most critical failures: An analysis of production failures in distributed Data-Intensive systems," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Oct. 2014, pp. 249–265. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan
[43] M. Ma, W. Lin, D. Pan, and P. Wang, "Self-adaptive root cause diagnosis for large-scale microservice architecture," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1399–1410, 2020.
[44] M. Li, Z. Li, K. Yin, X. Nie, W. Zhang, K. Sui, and D. Pei, "Causal inference-based root cause analysis for online service systems with intervention recognition," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 3230–3240.
[45] R. Xin, P. Chen, and Z. Zhao, "Causalrca: causal inference based precise fine-grained root cause localization for microservice applications," *Journal of Systems and Software*, vol. 203, p. 111724, 2023.
[46] M. S. Iqbal, R. Krishna, M. A. Javidian, B. Ray, and P. Jamshidi, "Unicorn: reasoning about configurable system performance through the lens of causality," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 199–217.
[47] M. A. Hossen, S. Kharade, B. Schmerl, J. Cámara, J. M. O'Kane, E. C. Czaplinski, K. A. Dzurilla, D. Garlan, and P. Jamshidi, "Care: Finding root causes of configuration issues in highly-configurable robots," *IEEE Robotics and Automation Letters*, vol. 8, no. 7, pp. 4115–4122, 2023.
[48] L. Giamattei, A. Guerriero, R. Pietrantuono, and S. Russo, "Causality-driven testing of autonomous driving systems," *ACM Trans. Softw. Eng. Methodol.*, dec 2023. [Online]. Available: https://doi.org/10.1145/3635709
[49] Z. Zhong, Z. Hu, S. Guo, X. Zhang, Z. Zhong, and B. Ray, "Detecting multi-sensor fusion errors in advanced driver-assistance systems," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2022, 2022, p. 493–505. [Online]. Available: https://doi.org/10.1145/3533767.3534223

APPENDIX A
ARTIFACT APPENDIX

A. Description & Requirements

We provide IRCA, an automatic root cause analysis tool designed specifically for addressing perception failures in Autonomous Driving Systems (ADS). IRCA integrates three core functionalities: runtime monitoring, failure identification, intervention, and a hierarchical intervention algorithm. To evaluate the effectiveness of IRCA, we have developed a comprehensive platform that incorporates co-simulation of Autoware.Universe Galactic with CARLA simulator. This platform is further enriched with fault injection capabilities and upgraded packages that ensure simulation synchronization with CARLA simulator.

1) How to access: Our implementation is available on Zenodo with DOI: https://doi.org/10.5281/zenodo.14250642. Alternatively, we also make the artifact available on GitHub:https://github.com/sgNicola/ACsim. Please get the map data from https://bitbucket.org/carla-simulator/autoware-contents/src/master/.

2) Hardware dependencies: GPU: 8 GB, CPU: 8 cores, RAM: 16GB and Disk space: 100 GB of space

3) Software dependencies:

- Operating System: Ubuntu 20.04
- ROS 2: ROS 2 Galactic. For ROS 2 system dependencies, refer to REP-2000.
- Install ROS 2 Dev Tools, the RMW Implementation, pacmod, Autoware Core dependencies, Autoware Universe dependencies, pre-commit dependencies, Nvidia CUDA, Nvidia cuDNN and TensorRT

4) Benchmarks: None

B. Artifact Installation & Configuration

(1) Artifact Installation

- **Create Workspace**: Download the code repository, and name as `~/ACsim` workspace.
- **Source Installation of Autoware.Universe**
  1) **Installing Dependencies Using Ansible:** Navigate to `~/ACsim/autoware` and run the setup script to configure the development environment:
     ```
     ./setup-dev-env.sh
     ```
  2) **Install Dependent ROS 2 Packages:** Install all ROS packages required by Autoware.Universe.
  3) **Build Autoware with Colcon:** Use colcon to build the Autoware.Universe from source:
     ```
     colcon build --symlink-install \
     --cmake-args \
     -DCMAKE_BUILD_TYPE=Release
     ```
     Upon successful build, the output should indicate: 247 packages finished
- **Package Installation of CARLA 0.9.14**
  - Please navigate back to the `~/ACsim` root directory and prepare for CARLA installation in `~/ACsim/CARLA_0.9.14`.
  - Please refer to Carla: Quick Start Package Installation to download and install CARLA 0.9.14.
- **Prepare Map Data**
  - Please download the autoware maps and put them into the folders. Each folder under `~/ACsim/map_data/TownXX` should contain two types of maps: `lanelet2_map.osm`, `pointcloud_map.pcd`

(2) Configuration

1) **Network settings for ROS 2 and Autoware**:
   - **Enable localhost-only communication:** Please Enable multicast for lo.
   - **Tune system-wide network settings:** Set the config file path and enlarge the Linux kernel maximum buffer size before launching Autoware. Increase the maximum receive buffer size to 2 GiB for network packets.

2) **Workspace Path Configuration:** Please ensure that your workspace path matches the environment variable settings. Replace the default path `/home/anonymous/ACsim/` with the actual directory of your workspace in the files listed `~/ACsim/carla_autoware/path_list.md`.
3) **Environment Variables Configuration:** Please configure the environment variables in `~/ACsim/readme.md` and add them to your `~/.bashrc` file.
4) **Create folders for saving data:** Please ensure you have the read/write access of ObjectData_Directory and Rosbag_Directory.

C. Major Claims

- (C1): SYSTEM successfully performs automatic root cause analysis with runtime monitoring, failure identification, and intervention functionalities, as demonstrated by [Figure 2]. This is proven by the experiment (E1)
- (C2): IRCA achieves the F1-scores above 95%. This is proven by the experiment (E1, E2, E3) whose results are reported in [TABLE III].
- (C3): SYSTEM consistently achieves an average of less than three interventions. This performance is validated by experiments (E1,E2,E3), with results in [Table VI].
- (C4): SYSTEM can automatically generate syntheic scenarios with injected faults. This performance is validated by experiments (E4), with results in [Table IV].

D. Evaluation

*1) **Experiment E1**:* **[Camera-LiDAR-fusion configuration with real fault scenarios]** about [20 human-minutes + 100 compute-hours]: This experiment aims to:

- Assess the functionality of the automated IRCA.
- Evaluate IRCA's effectiveness in real-world fault scenarios, as Table III details.
- Evaluate IRCA's efficiency as detailed in Table VI.

[Preparation]

1) **Multi-Sensor Fusion Configurations (MSF) in Autoware:** The default configuration Camera-LiDAR-fusion Configuration is used in this experiment.
2) **Scenario Parameters Configuration:** The configuration of each group is set up according to the specific scenario arguments. The group IDs in TABLE III and their corresponding scenario arguments are{1:Huge_truck, 2: Truck_walker 3-4:Cone (with different object types), 5:Sparse, 6:Truck, 7:Vans}.
3) **Failure Modes Configuration:** The failure mode arguments in `param.py` are configured based on the definitions provided in TABLE I. Mappings include {false_negative: Missing Obstacle, false_positive: Ghost Obstacle, wrong_localization: Mislocalization, wrong_classification: Misclassification}.

[Execution]

- **Main Script:** Execute `python3 fusion_hira.py`.
- **Experiment Workflow:**
  - **Initialization:** This framework will start the CARLA server to simulate driving scenarios based on the configured scenario parameters.
  - **Start Co-simulation** The Autoware system is launched along with the generated scenarios.
  - **Data Recording:** The perception outputs of each module are recorded with ROS2 bags in the specified `ROSbag_directory`. Then the ROS2 bag messages are processed and saved in the `Object_Directory`.
  - **Fault Mode Identification:** The fault modes of the modules are identified and recorded in `Experiment/(Execution_id)_(object_id)_(failure_mode).csv`. If the final output in the perception system matches the pre-configured failure mode, it will trigger the subsequent iterations for root cause analysis.
  - **Root Cause Analysis:** The main difference in the subsequent iterations from the first one is the **intervention**. IRCA will start intervention nodes in a new terminal session and display intervention details in the terminal. Other steps, such as data recording and analysis, remain consistent throughout all iterations. Interventions are adjusted based on ongoing analysis until root causes are identified.
  - **Completion:** Conclude the experiment cycles once root cause analysis is completed and save the final results in `/ACsim/carla_autoware/results.csv`.

[Results]

The results from a group are saved in `/ACsim/carla_autoware/results.csv`. The columns in the CSV contain experiment ID, iterations, and identified modules. After completing a series of experiments, rename the results.csv file with the group ID and method used, for example, 01_aid.csv. Note that the name of IRCA method is hira.

- **Effectiveness:** The performance metrics, including recall, F1-score, and precision are calculated with `evaluation.ipynb`.
- **Efficiency:** The number of interventions in each experiment are determined by calculating the file count minus one in the directory `/CausalAnalysis/data/(Experiment_id)`. This adjustment is made because the first iteration does not involve an intervention.

*2) **Experiment E2**:* **[LiDAR-fusion configuration with real fault scenarios]** [about 20 human-minutes + 100 compute-hours]: This experiment aims to evaluate the effectiveness of groups 1-2, as detailed in Table III.

[Preparation]

LiDAR-fusion Configuration: For the LiDAR-fusion setup, please update the launch arguments in `autoware.launch.xml` and

`detection.launch.xml` according to preparation in `ACsim/readme.md`.Additionally, modify line 6 and line 10 in the `utils.py` file to import the specific modules run_lidar and parse_lidar respectively, which are needed for the LiDAR configuration.

[Execution]&[Results]

Execute `python3 lidar_hira.py`, with other steps the same as E1.

*3) **Experiment E3**:* **[Real fault scenarios]**[about 20 human-minutes + 100 compute-hours]: This experiment aims to evaluate the effectiveness and efficiency in real-world fault scenarios of groups 6-7, as detailed in Table III.

[Preparation]

Cluster configuration: Modify line 4 in `lidar_hira.py` to import the appropriate module run_cluster for cluster-based processing.

[Execution]&[Results]

These steps are the same as in E2.

*4) **Experiment E4**:* **[Syntheic Scenarios]**[about 10 human-minutes + 200 compute-hours]: This experiment aims to evaluate the effectiveness and efficiency in synthetic fault scenarios of groups 8-19, as detailed in Table IV and Table V.

Fault Injection: To inject faults as listed in Table II, please update the arguments for each module in the script `ACsim/carla_autoware/op_agent/start_ros2.sh` according to module IDs and corresponding arguments in `readme.md`. To inject an MO fault to the Merger Module, update the argument to: `merger_faulty_mode:=1`.

[Execution]&[Results]

These steps are the same as in E1-E3.