IoT Anomaly Detection Via Device Interaction Graph

Jincheng Wang
The Chinese University of Hong Kong
Hong Kong, China
wjcuhk@gmail.com

Zhuohua Li
The Chinese University of Hong Kong
Hong Kong, China
zhli@cse.cuhk.edu.hk

Mingshen Sun
Independent Researcher
San Jose, USA
bob@mssun.me

Bin Yuan[*]
Huazhong University of Science and Technology
Wuhan, China
yuanbin@hust.edu.cn

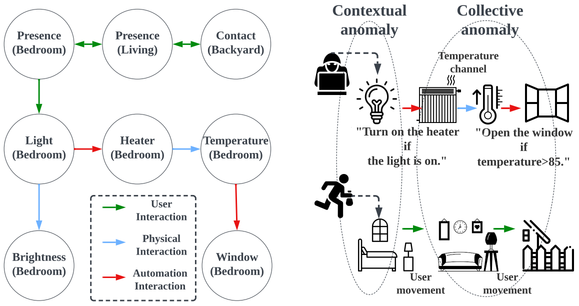
John C.S. Lui
The Chinese University of Hong Kong
Hong Kong, China
cslui@cse.cuhk.edu.hk

***Abstract*—With diverse functionalities and advanced platform applications, Internet of Things (IoT) devices extensively interact with each other, and these interactions govern the legitimate device state transitions. At the same time, attackers can easily manipulate these devices, and it is difficult to detect covert device control. In this work, we propose the *device interaction graph*, which uses device interactions to profile normal device behavior. We also formalize two types of device anomalies, and present an anomaly detection system CausalIoT. It can automatically construct the graph and validate runtime device events. For any violation of interaction executions, CausalIoT further checks whether it can trigger unexpected interaction executions and tracks the affected devices.1 Compared with existing methods, CausalIoT achieves the highest detection accuracy for abnormal device state transitions (95.2% precision and 96.8% recall). Moreover, we are the first to detect unexpected interaction executions, and CausalIoT successfully reports 91.9% anomaly chains on real-world testbeds.**

***Index Terms*—IoT, anomaly detection, causal discovery, device interaction graph**

I. Introduction

With ubiquitous sensing, actuating, and communication capabilities, the IoT service has been playing a critical role in human lives [1]. Smart home, one of the most popular IoT applications, has proliferated significantly in the past few years. As predicted by Statista [2], more than 400 million IoT devices will be deployed at smart homes by 2025. Besides the large-scale deployment, these devices also interact with each other: The state change of one device can directly affect others' states. Figure 1a shows an example of an interaction network. Each directed edge represents a device interaction, and the color denotes its source. For instance, user activities usually involve sequential operations on a set of devices, which create interactions among them (e.g., sensors to detect user movement). Moreover, since IoT devices can change and sense the physical environment, they also interact with each other if they work on the same physical channel (e.g., temperature). Finally, with the emergence of IoT platforms, users can install automation rules for easy device management. The automation logic specifies a device interaction between the triggering device and the action device. In Figure 1a, users install a rule *Activate the heater if the light is on* [3], which creates an interaction between the light switch and the heater.



(a) Device interactions

(b) Security threats

Fig. 1: Device interactions and security threats at smart homes

Despite the convenience provided by the smart home, there are growing concerns about its security, and Figure 1b presents two examples. First, attackers can easily manipulate the deployed devices for their broad attack surface [4]–[11]. As a result, covert device state transitions frequently happen, which threaten the user and environment (e.g., mysterious light activation [4], [12] and ghost presence [13], [14]). Second, since IoT devices extensively interact with each other, any device misbehavior can further trigger unexpected interaction executions to affect other devices. For example, the light activation can trigger the execution of an automation chain [10], which consequently activates the heater and opens the window.

[*] Bin Yuan is the corresponding author.

The severity of smart home security promotes the need for device monitoring and anomaly detection. Unfortunately, current IoT platforms do not provide such support. As a result, users often complain about their devices' misbehavior [12], [14]–[18]. Specifically, users want to be notified at once if their devices are covertly operated, so they can initiate device recovery and risk evaluation. Moreover, once some interactions are maliciously executed and a group of devices collectively misbehaves, it is critical to report the anomaly chain for later device recovery, forensic study, and root cause localization. For example, detecting the light activation and its consequences can help users to pinpoint the affected devices and re-evaluate the risk of the deployed automation rules. As another example, detecting a suspicious presence event at the bedroom and its consequences can help to identify burglar intrusion and reconstruct his traces.

In order to address users' concerns, many anomaly detection systems have been proposed [19]–[25]. However, few consider the nature of the prevalent device interactions, and the limitation is as follows. (1) There is a lack of understanding on how devices normally work. Since the device interactions govern the legitimate device state transitions, without the knowledge of these interactions (e.g., Presence → Light), it is challenging to justify runtime device behavior (e.g., light activation), and this dramatically reduces the detection accuracy. (2) It is almost impossible to track the unsolicited execution of device interactions. As a result, when anomalies (e.g., a high-temperature reading and then open the window) propagate along the interaction chain, existing solutions cannot pinpoint all the affected devices. (3) It is difficult to interpret the detected anomaly and provide hints for root cause localization. Instead, the interaction can provide additional information about the anomaly. For instance, the context of no presence in the bedroom can help users to reason about the detection result. Moreover, it can help the security analyst to exclude the possibility that the light is physically compromised, and focus more on the investigation of the remote control.

The above discussion highlights the importance of using device interactions for smart home anomaly detection. In this work, we propose the *device interaction graph*, which profiles the complex interaction network at end users' smart homes (Section III). Each node in the graph represents an IoT device, and each directed edge represents an interaction from the parent device to the child device (e.g., Presence → Light). Moreover, the graph maintains a set of conditional probability tables, e.g., $P(\text{Light}|\text{Presence})$, such that given the states of parent devices, the graph can output the likelihood of the child device's state. Based on the graph, we formalize two types of device anomalies: the *contextual anomaly* and *collective anomaly* (Section IV). The former captures a device event that violates the interaction execution (i.e., with low likelihood), while the latter captures a sequence of device events that follows the maliciously triggered interaction execution. In the previous example, the light activation event is a contextual anomaly since it is seldom activated when there was no presence detected. The sequence of events from the heater, temperature sensor, and window is a collective anomaly as they follow the unsolicited execution of the Heater → Temperature → Window interaction chain.

Finally, we present CausalIoT, a security system for automated interaction mining and anomaly detection (Section V). It first constructs an interaction graph from the logged device events. In particular, CausalIoT initiates the causal discovery process [26], [27], and it can reduce the graph complexity and increase its interpretation capability (See Section II for more discussions). With the interaction graph, CausalIoT checks if the runtime event violates the interaction execution (contextual anomaly detection). If that is the case, CausalIoT further tracks its propagation and eventually reports the anomaly chain to users (collective anomaly detection). In summary, we make the following contributions.

- We provide a new perspective toward understanding normal device behaviors at smart homes, i.e., the device interaction graph. Moreover, we are the first to initiate the *causal discovery process* for studying device interactions, which requires no user intervention or background knowledge.
- We formalize two types of smart home anomalies that imply severe security threats: Contextual and collective anomalies. We also design an algorithm that utilizes the interaction graph for anomaly detection.
- We implement a prototype CausalIoT and evaluate it on real-world testbeds. The evaluation result shows that it can identify most device interactions. For anomaly detection, compared with existing methods, CausalIoT achieves the best performance for contextual anomaly detection (95.2% precision and 96.8% recall). Moreover, we make the first step toward collective anomaly detection, and successfully report 91.9% anomaly chains.

II. Background

A. Smart Home

At the core of smart homes are ubiquitous IoT devices and a centralized IoT platform (e.g., openHAB [28] and SmartThings [29]). Specifically, IoT devices are deployed with a large number of sensors and actuators, which allow them to sense the environment and execute predefined operations. The platform is responsible for device management. After the platform binds with the device, it first abstracts a set of virtual *device attributes* (e.g., TemperatureSensor) to describe the device state [30]. Once the user or the physical environment changes the device state, the device will immediately send a device event to the platform. As a result, the platform collects these runtime events to track the latest device states. Usually, the event is in the format of (*timestamp, device name, installation location, device state*), and the value types of device states are diverse (e.g., the enum on/off state and the numeric illuminance measurement). Besides state tracking, the IoT platform allows users to design automation rules for easy device control. These rules follow the trigger-action programming paradigm [31], and users need to specify the triggering device and the action device. By doing so, when a condition is met, the platform will operate the action device. For instance, the window will be opened if the temperature exceeds 85°F.

An essential feature of a smart home is the widespread device interactions, which govern legitimate device state transitions. Nevertheless, few works discuss how to infer the complex interaction network. Some efforts proposed static analysis to study the automation logic [10], [32]–[38]. However, they only focus on automation interactions and require users to provide the source codes of their installed automation rules [39]. In order to capture the interaction from diverse sources while reducing user intervention, a potential solution is to use logged device events and learn the statistical association relationship among device states [23]. Unfortunately, while the interaction implies the association, the reverse does not hold. We find that there are many "spurious interactions" which stem from the *intermediate factor* and the *common precedence* [40]. Going back to Figure 1a, since the heater is an intermediate device in the interaction chain Light → Heater → Temperature Sensor, the states of the light and the temperature sensor are associated. However, they have no direct interaction as the change of the light state does not directly affect the temperature sensor's state. As another example, since the light is common precedence, i.e., Heater ← Light → Brightness Sensor, the states of the brightness sensor and the heater are also associated. Similarly, turning on/off the heater has nothing to do with the brightness sensor's state. These spurious interactions increase the complexity of the interaction network and reduce the interpretation capability for the detection result. For instance, it makes no sense to explain to users that "a high brightness sensor reading is abnormal because the heater was not turned on". Therefore, a more stringent statistical language instead of pure association is needed to describe the interaction, and for this, we resort to the causal primitive.

B. Causality

The questions that motivate many studies (e.g., biomedical, social, and behavioral sciences [41]–[43]) are not about association but causality in nature. For example, researchers are interested in the question "*whether the obesity* (S_i) *causes high COVID-19 susceptibility* (S_j) [44]". In order to make sure that there is no other factor that affects the justification, one usually fixes all other variables, e.g., age, at some values. In this case, if intervening on S_i can affect the value of S_j, we say that S_i causes S_j. Usually, a directed edge is used to represent the causal relationship, which points from the cause to the outcome ($S_i \rightarrow S_j$).

For a complex system that has a large number of variables, the causal network may be too complex. As a result, prior work [44] based on the Bayesian network and proposed a *causal graph*. Specifically, it is a directed acyclic graph with the following components: (1) a set of nodes (variables), (2) a set of directed edges between pairs of nodes, and (3) a joint probability distribution over the possible values of all the variables. In particular, the second component encodes the causal relationship. It states that *two variables with an edge in between are associated if one fixed all other variables at some values.* Such a requirement guarantees that each edge encodes not only statistical association, but also a direct effect on the outcome when the cause is intervened.

In order to identify causal relationships, a traditional approach is to design a controlled experiment. However, in many cases, they are too expensive or infeasible. Therefore, causal discovery from purely observational data has drawn much attention [45], [46]. In particular, one of the most well-known methods is called the *PC algorithm* [47], [48]. Roughly speaking, the algorithm aims to identify a set of directed edges among variables and construct the skeleton of the causal graph. It starts with a fully-connected undirected graph. Then for each edge $S_i - S_j$, the algorithm provides a conditional independence test framework to find a conditioning set C, such that it makes the variables conditionally independent, i.e., $S_1 \perp\!\!\!\perp S_2 | C$. Once the set is identified, no causal relationship exists between these two variables, and the edge is removed. Otherwise, the algorithm preserves the edge and uses a set of empirical rules [49] to decide its orientation.

III. Device Interaction Graph

The causal primitive precisely follows the definition of device interactions that the operation on the parent device (cause) can directly affect the child device (outcome) [1]. As a result, a naive approach to modeling an interaction network at smart homes is to use a causal graph, where each node represents a device state, and each edge represents an interaction. Unfortunately, this does not work. The reason is that a smart home is in fact a temporal system: The device state and the interaction are time-dependent. By checking device states at different timestamps, one can find that the states of the devices at different timestamps can be causally related. However, the causal graph was not designed to model a temporal system, and it does not encode any temporal information. To address the modeling issue, we extend the causal graph and propose a *device interaction graph* (DIG). An enhanced PC algorithm for DIG construction is further introduced in Section V-B. Before diving into the details, we first introduce some symbols used throughout this paper.

[1] Since each interaction is interpreted as a causal relationship, in this paper, we use the terms "parent device" and "cause" interchangeably. The same condition holds for the terms "child device" and "outcome"

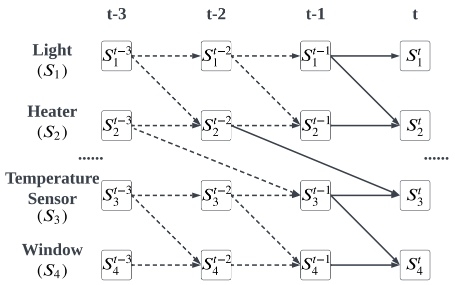


Fig. 2: An example of the device interaction graph

Suppose there are n devices deployed at a smart home. For the ith device, its state at timestamp t is modeled as a random variable S_i^t. The timestamp is discrete and based on the order of the device event. By default, we call the timestamp t as "present". As we introduced in Section II-A, whenever there is a device state transition, the device reports its new state by sending a device event. For simplicity, we use $e^t : \{S_i^t = s_i^t\}$ to denote an event at timestamp t which reports the state of the ith device, and s_i^t is its state value. Since only one device reports its state at each timestamp, we omit the subscript i for clarity. Given a sequence of device events $(e^1, ..., e^m)$ and an initial system state $S^0 = (s_1^0, ..., s_n^0)$, one can derive the system state at each timestamp. Specifically, the system state S^j at timestamp $j \in \{1, ..., m\}$ is $(s_1^{j-1}, ..., s_i^j, ..., s_n^{j-1})$, where s_i^j is recorded in the event e^j, and s_k^{j-1} $(k \neq i)$ is recorded in S^{j-1}. As a result, the sequence of system states $(S^0, ..., S^m)$ forms a *time series*. We define the device interaction graph as follows.

Definition *1:* A device interaction graph (DIG) is an extended causal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{P})$, where $\mathcal{V} = \{S_i^j | i \in \{1, ..., n\}, j \in \mathbb{Z}\}$ is a set of nodes that represents time-dependent device states, $\mathcal{E}$ is a set of directed edges that represents the causal relationship, and $\mathcal{P}$ is a set of conditional probability tables (CPT) for nodes in $\mathcal{V}$.

Figure 2 depicts an example. Each solid edge denotes an interaction from the cause to the outcome. The dashed edge represents a repeated interaction that will be introduced later. Note that each node in DIG represents a temporal variable. One benefit is that it is easy to orient the interaction since the cause always comes before its effect in time [47]. Another advantage is that the interaction can now describe the time-dependent effect. Specifically, for each state S_i^j, let $Ca(S_i^j) \subseteq \{S_k^{j-l} | k \in \{1, ..., n\}, l \in \{1, ..., j\}\}$ be the set of its causes, and $ca(S_i^j)$ be the values of $Ca(S_i^j)$. The DIG maintains a conditional probability table $P(S_i^j = s_i^j | Ca(S_i^j) = ca(S_i^j))$, which describes the state distribution of S_i^j under the interaction execution. For instance, in Figure 2, $P(S_3^t = 1 | S_2^{t-2} = 1, S_3^{t-1} = 0)$ quantifies the influence of the heater on the temperature sensor through the physical channel. As another example, a high value of $P(S_4^t = 1 | S_3^{t-1} = 1, S_4^{t-1} = 0)$ describes the logic of the deployed automation rule, i.e., open the window if the temperature is high.

Note that the number of nodes and edges in the DIG can be extremely large. To address the complexity issue, we make two mild assumptions that are common in the temporal setting [50]–[52]. The first is the *τth-order Markov Assumption*. Specifically, given a maximum time lag $\tau > 0$, for each device state S_i^j, the time lag of its causes is smaller than τ, that is, $Ca(S_i^j) \subset \{S_k^{j-l} | k \in \{1, ..., n\}, l \in \{1, ..., \tau\}\}$. In Figure 2, the maximal time lag $\tau = 2$, and so the farthest cause for S_3^t is S_2^{t-2}. The second assumption is the *Stationarity Assumption*: The interaction between $t - \tau$ to t holds for every $t' \in \mathbb{Z}$. In Figure 2, each dashed edge encodes the same interaction as the solid edge. With these two assumptions, one only needs to focus on the nodes during the period from $t - \tau$ to t, i.e., $\mathcal{V} = \{S_i^j | i \in \{1, ..., n\}, j \in \{t - \tau, ..., t\}\}$. Moreover, one can use the interactions during the period to explain the logged events e^j with timestamp $j < t$, and forecast the future device event with $j > t$. Define $G^j = (S^{j-\tau}, ..., S^j)$ as the graph snapshot at timestamp j. For each S_i^j, one can determine the values of its causes $ca(S_i^j)$ from G^j.

The DIG provides an efficient way to study device behavior. Specifically, given an event e^j and the snapshot G^j, one can look up the conditional probability table and get the likelihood of the event. The likelihood can be used to quantify the anomalous level of the event and determine whether it violates the interaction execution. For example, when the heater was deactivated and the platform received a high-temperature reading, a table lookup shows that it is unlikely to receive the event (i.e., a low $P(S_3^t = 1 | S_2^{t-2} = 0, S_3^{t-1} = 0)$ value), and therefore it violates the execution of the physical interaction. Moreover, the set of causes can provide additional hints for the event, which helps users to reason about the detection results. Finally, one can use the DIG to keep track of the interaction execution. When an interaction chain is abnormally executed, the graph can help to track the affected devices.

IV. Threat Model and Goals

In this work, we focus on the covert device state transitions, which are widely reported by smart home users [14]–[17], [53]–[56]. For example, users complain about the smart plug events for being mysteriously turned on at midnight [17], or ghost presence events [54], [55]. These anomalies are harmful as they usually imply severe consequences (e.g., overheating or burglar intrusion). We first survey the IoT attack vectors which can cause ghost transitions, and categorize two security threats based on the attacker's target. Then we define two types of device anomalies and show that the DIG can properly address them. Finally, we present our goals and some mild assumptions.

A. Smart Home Anomaly

Malicious device control. The attacker seeks to covertly control the devices by exploiting IoT vulnerabilities. Specifically, the attacker can physically manipulate the device [13], exploit device firmware flaws [5], [57], leverage communication protocol vulnerabilities [8], [58], or abuse physical channels [11]. As a result, the attacker can force a device state transition without users' consent and awareness. Moreover, the attacker can initiate advanced rule-level attacks [32], [59], such that he can tamper with the device state when certain conditions hold (e.g., malicious rule insertion). Since these anomalies usually violate the interaction execution, we define them as *contextual anomalies.*

Definition *2 (Contextual Anomaly):* Given a DIG $\mathcal{G}$, the graph snapshot G^j at timestamp j, an anomaly score function f, and a score threshold c, a contextual anomaly is a device event $e^j : \{S_i^j = s_i^j\}$ which deviates from the interaction execution, i.e., $f(e^j, G^j, \mathcal{G}) \geq c$.

In particular, the anomaly score is calculated as follows.

$$f(e^j, G^j, \mathcal{G}) = 1 - P(S_i^j = s_i^j | Ca(S_i^j) = ca(S_i^j)). \qquad (1)$$

The larger the score is, the more unlikely the event will be observed in the interaction execution, and therefore the event will be more anomalous. The anomaly is contextual because it is rare under the context specified by $Ca(S_i^t)$. For instance, when no presence was detected in the kitchen, a plug activation event is highly suspicious as users seldom operate the plug when not in the kitchen.

Unsolicited interaction execution. The attacker targets device interactions and aims to trigger unsolicited interaction executions. To achieve the goal, the attacker needs to first compromise an IoT device and promote its state transition (i.e., create a contextual anomaly). Consequently, a sequence of device events collectively happens due to the interaction execution. For example, after breaking into the house, an attacker can initiate unsolicited user activities, which create a sequence of events about contact and presence sensors [13] (e.g., search for belongings). Moreover, the attacker can control IoT devices to abuse the physical channel, which affects the environment and causes unexpected sensor readings [4] (e.g., maliciously turn on the light and increase the brightness). Finally, the attacker can compromise the triggering device specified in an automation rule, which triggers the rule execution [10], [32], [36], [59]. As a result, we define the above anomalous event sequence as the *collective anomaly.*

Definition *3 (Collective Anomaly):* Given a DIG $\mathcal{G}$, a sequence of graph snapshots $(G^j, ..., G^k)$, an anomaly score function f, and a score threshold c, a collective anomaly is a sequence of events $(e^j, e^{j+1}, ..., e^k)$ created by unexpected interaction executions, such that $f(e^{j-1}, G^{j-1}, \mathcal{G}) \geq c$ and $f(e^{j+\ell}, G^{j+\ell}, \mathcal{G}) < c$ for $\ell \in \{0, ..., k\}$.

The collective anomaly is difficult to discover because the involved event has a low anomaly score, which follows the interaction execution and seems legitimate. Consider Figure 1a as an example. After the attacker maliciously turns on the light, the consecutive brightness event encodes a normal reading from the physical channel. Moreover, the consecutive events from the heater and the window follow the execution of user-installed automation rules. However, since the contextual anomaly e^{j-1} has polluted the system states in $G^{j+\ell}$, the low anomaly score in fact measures the interaction execution under a malicious context. As a result, the involved events should be anomalous in the view of users.

While this work mainly discusses the smart home application and its security issues, we envision that our formalism of DIG and device anomalies can also apply to other IoT applications. For example, in industry IoT [60], a smart warehouse system may have the following business logic. A sensor first detects a low inventory level for an item, and the platform asks a programmed robot to put the item on an autonomous truck. Then the truck delivers the item to the warehouse. The above logic specifies an interaction chain Sensor → Robot → Truck, and our DIG is well-suited for describing it. With the DIG, one can further formalize various industrial IoT anomalies according to Definition 2 and Definition 3 (e.g., robot misbehavior caused by command injection [60] and unsolicited truck movement triggered by robot [61]). As another example, in a water system, multiple sensors are deployed in different places to monitor the water quality [62]. Depending on their locations, sensors interact with each other, and their readings are naturally affected by the upstream sensor readings. A DIG that profiles the sensor network can help to detect water pollution (reflected in abnormal sensor readings) by checking the upstream sensor readings. Moreover, it supports tracking the flow of the polluted water, which is reflected in a sequence of abnormal readings.

B. Security Goals and Assumptions

We aim to design an anomaly detection system that achieves the following goals. (1) *DIG construction.* Given the logged device events at smart homes, the system can identify the underlying device interactions and construct the DIG. Moreover, the process should be automated without any background knowledge or user involvement (e.g., source codes of automation rules). (2) *Runtime monitoring.* The system can validate runtime device events, such that it can detect contextual and collective anomalies. Moreover, the detection process should be precise with a small number of false and missing alarms.

Besides the τth-order Markov assumption and the Stationarity assumption for the DIG (Section III), we also make the following assumptions for the attackers and the smart home. (1) The IoT platform is not compromised, such that it can collect runtime device events and keep track of the device state. The structure of device events can be simple which only contains the timestamp and device state. Jamming attacks that block communication are not in our scope, as one can easily detect them due to session timeout or missing sequence numbers [63]. Since there are several existing solutions for event collection in IoT platforms [64]–[66], the design of the collection module

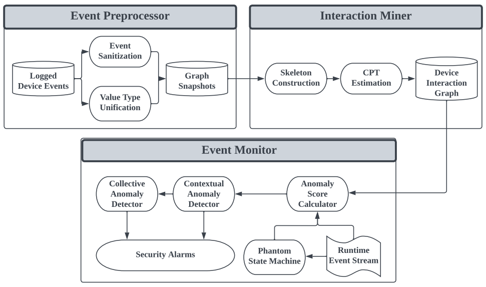


Fig. 3: The architecture of CausalIoT

is beyond our scope. Our anomaly detection system can be easily integrated into existing platforms and leverage their collected device events. (2) We assume that there is no hidden device that is not tracked by the platform. Otherwise, it may affect other devices and introduce spurious interactions (e.g., acting as a common cause). (3) Like other anomaly detection work [22], [23], we assume a *semi-supervised* setting, where there are no or few anomalies in the logged device events.

V. CausalIoT

We present our anomaly detection system CausalIoT, and Figure 3 shows the architecture. (1) *Event Preprocessor.* The module is responsible for preprocessing logged device events and generates graph snapshots. (2) *Interaction Miner.* It takes the snapshots as inputs and initiates the mining algorithm. The output is a DIG that models the interaction network at end users' smart homes. (3) *Event Monitor.* The module takes the runtime device event as input, and keeps track of the latest graph snapshot. Moreover, it uses the DIG to check whether the event is a contextual anomaly. For any detected contextual anomaly, the module further initiates collective anomaly detection.

A. Event Preprocessor

Preprocessing the device event is necessary for the following reasons. (1) The raw events are usually noisy with duplicated state reports [23], [39]. For example, the brightness sensor periodically reports its readings, even though the environment remains unchanged. (2) The diverse value types of device states make it difficult to mine device interactions. Specifically, the mining algorithm involves a large number of conditional independence tests. However, the issue of mixed value types can increase the time complexity and reduce the accuracy of the test result [67]. (3) The raw events are in the structural format, which the mining algorithm cannot directly use. As a result, it is necessary to transform them into a set of graph snapshots. We thus design the Event Preprocessor that initiates the following procedures.

Event sanitation. The preprocessor first sanitizes the logged device events by checking whether they imply duplicated state reports. If that is the case, the preprocessor filters them. For the device state of the numeric type (e.g., the brightness reading), the preprocessor further estimates its mean μ and the standard deviation σ. Readings that violate the *three-sigma rule* [68] (i.e., fall outside the range $[\mu - 3\sigma, \mu + 3\sigma]$) are considered as extreme values and are filtered out.

Type unification. After checking the list of device attributes from the SmartThings' developer website [30], we categorize three types of device states according to their value types: *binary state*, *responsive numeric state*, and *ambient numeric state.* The binary state value usually implies the device's ON/OFF state, e.g., the switch. The responsive numeric state value is zero when the device is idle, and the value becomes positive when the device is in use. For example, the water meter readings are non-zero only when the user is using the sink. As a result, one can set the threshold to zero and transform the responsive numeric state to a binary state, which implies the Idle/Working state of the device. Finally, the ambient numeric state value is usually positive, because these devices are designed for continuous measurement of environmental factors. Since most device states are of binary and responsive numeric types, we unify the types of device states to the binary state. To achieve the goal, we use the *Jenks natural breaks algorithm* [69] and discretize the value of the ambient numeric state, which implies the Low/High state of the device.

Snapshot generation. With the set of preprocessed events, we first generate the time series $(S^0, ..., S^m)$ as specified in Section III. Given the maximum time lag τ, we further concatenate every neighboring τ+1 system states, and eventually generate the graph snapshots $G = \{G^j : (S^{j-\tau}, ..., S^j) | j \in \{\tau, ..., m\}\}$. Note that the value of the time lag τ affect both the graph complexity and the capability of interaction mining. Specifically, a significant τ incurs more nodes in the DIG, which further increases the time complexity of DIG construction (See Section V-B and Section V-D). However, a smaller τ may miss some causes of an outcome state whose effects usually take a long time, and these missing causes can further affect the capability of anomaly detection. To determine the value of τ, we first calculate the average time interval v (in seconds) between two neighboring events. Then we follow the previous work [23] and set a maximum duration d = 60s, which is long enough to wait for any feedback (e.g., automation execution) given a device operation. Finally, we set $\tau = d/v$, such that the DIG can encode the underlying device interactions.

B. Interaction Miner

The module aims to construct the DIG from the graph snapshot. Specifically, we first design a variant of the PC algorithm to identify the graph skeleton, i.e., the set of edges in the graph. Then an estimation procedure is initiated to infer the conditional probability table for each node in the graph.

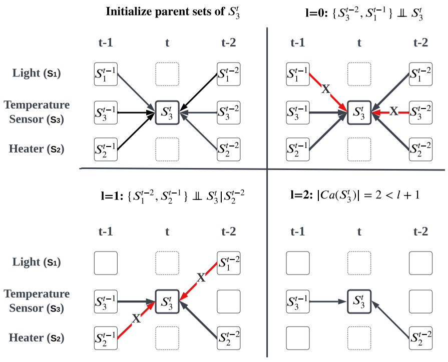

Initialize parent sets of S_3^t
l=0: $\{S_3^{t-2}, S_1^{t-1}\} \perp\!\!\!\perp S_3^t$
l=1: $\{S_1^{t-2}, S_2^{t-1}\} \perp\!\!\!\perp S_3^t | S_2^{t-2}$
l=2: $|Ca(S_3^t)| = 2 < l + 1$
t-1
t
t-2
Light (S1)
Temperature Sensor (S3)
Heater (S2)
S_1^{t-1}
S_3^{t-1}
S_2^{t-1}
S_3^t
S_1^{t-2}
S_3^{t-2}
S_2^{t-2}
X

Fig. 4: The workflow of TemporalPC

Skeleton construction. Given the graph snapshots, the PC algorithm provides a conditional independence test framework (Section II-B) to discover the edges among the set of device states $\{S_i^j | i \in \{1, ..., n\}, j \in \{t - \tau, ..., t\}\}$. However, the algorithm was not designed to handle a temporal system. In particular, it uses a set of heuristic rules for edge orientation (e.g., the Meek's rule [49]). Unfortunately, these rules are not guaranteed to orient each edge, which significantly reduces the accuracy of skeleton construction. Instead, incorporating temporal information can help to precisely orient each edge. As a result, we propose our enhanced *TemporalPC* algorithm.

Algorithm 1 shows the pseudocode. It takes a device state S_i^t as the input and outputs its causes $Ca(S_i^t)$. To construct the skeleton of the DIG (e.g., Figure 2), one can initiate the algorithm for all S_i^t ($i \in \{1, ..., n\}$), and unify the identified edges. Note that TemporalPC requires to specify a conditional independence test method (Line 1 to Line 4). Since each S_i^j is a binary state (Section V-A), we use the *G square test* [70]. Specifically, to determine whether two variables X and Y are conditionally independent given a conditioning set Z, the test first sets a null hypothesis: $X \perp\!\!\!\perp Y|Z$ is true. Then it uses the graph snapshots to calculate a G^2 statistic. The smaller the statistic is, the more likely the null hypothesis holds (measured by the p-value). Therefore, TemporalPC takes an additional parameter α as input, which acts as the significance threshold for the p-value. The null hypothesis holds if the calculated p-value is more significant than α. Otherwise, the algorithm will reject it and regard these two variables as dependent variables.

To better explain the workflow of TemporalPC, we present an example with $n = 3$, $S_i^t = S_3^t$, and $\tau = 2$ (Figure 4). TemporalPC starts with a fully-connected graph, where each time-lagged device state (i.e., the state with a smaller timestamp) is a parent of S_3^t (Line 5). We here exploit the temporal knowledge that *the cause always comes before its effect in time*. As a result, each edge is automatically oriented from the time-lagged device state to the present state S_3^t. Then TemporalPC iterates over each dimensionality l, enumerates the subset of parents with size l, and initiates a series of conditional independence tests for each edge (Line 12 to Line 20). For example, to test the edge $S_1^{t-1} \rightarrow S_3^t$ with $l = 0$, TemporalPC sets an empty conditioning set and checks whether $S_1^{t-1} \perp\!\!\!\perp S_3^t$. Since the result shows that they are independent, the corresponding edge is removed (Line 14 to Line 18). The same condition holds for the state S_3^{t-2}.

After TemporalPC finishes the test with $l = 0$, there are four remaining parents. Then it increases the dimensionality with $l = 1$, and generates the conditioning set with size one from the remaining parents. For example, TemporalPC enumerates three conditioning sets ($\{S_3^{t-1}\}$, $\{S_2^{t-1}\}$, and $\{S_2^{t-2}\}$) for testing the edge $S_1^{t-2} \rightarrow S_3^t$. As a result, it identifies that $S_1^{t-2} \perp\!\!\!\perp S_3^t | \{S_2^{t-2}\}$, and therefore the edge is removed. Similarly, it also identifies that $S_2^{t-1} \perp\!\!\!\perp S_3^t | \{S_2^{t-2}\}$, and removes the edge $S_2^{t-1} \rightarrow S_3^t$. After TemporalPC finishes the test with $l = 1$, it further increases l. The algorithm terminates when the number of the remaining parents is smaller than $l + 1$ (Line 9 to Line 11), because in this case, the algorithm cannot generate any conditioning set for the test. In Figure 4, the algorithm terminates when $l = 2$. As a result, the preserved edges are the identified device interactions, and the set of causes for S_3^t is identified (i.e., $\{S_3^{t-1}, S_2^{t-2}\}$).

One can check from Figure 2 that the removed edge $S_1^{t-2} \rightarrow S_3^t$ is in fact a spurious interaction created by the intermediate state S_2^{t-2}. Specifically, the association first goes from S_1^{t-2} to S_1^{t-3}, then it goes to S_3^t along the chain $S_1^{t-3} \rightarrow S_2^{t-2} \rightarrow S_3^t$. As another example, the removed edge $S_2^{t-1} \rightarrow S_3^t$ is also a spurious interaction created by the common cause S_2^{t-2}, i.e., $S_2^{t-1} \leftarrow S_2^{t-2} \rightarrow S_3^t$. With the stringent conditional independence test, TemporalPC can successfully remove these spurious interactions.

CPT estimation. After TemporalPC identifies the causes of each device state S_i^t for $i \in \{1, ..., n\}$, these interactions form the skeleton of the DIG (Figure 2). The next step is to estimate the conditional probability table for each S_i^t. To this end, we use the *maximum likelihood estimation* [71]. Specifically, let $P(S_i^t = s_i^t | Ca(S_i^t) = ca(S_i^t))$ be the parameter that we aim to estimate. The main idea is to determine the value $\hat{P}(s_i^t | ca(S_i^t))$ which maximizes the likelihood of observing the collected graph snapshots. As a result, the value is calculated as $\hat{P}(s_i^t | ca(S_i^t)) = \frac{\hat{P}(s_i^t, ca(S_i^t))}{\hat{P}(ca(S_i^t))} = \frac{\#(s_i^t, ca(S_i^t))}{\#(ca(S_i^t))}$, where $\#(*)$ is the number of the snapshots satisfying the condition. For example, suppose there are 100 snapshots with $S_2^{t-2} = 1$ and $S_3^{t-1} = 0$, among which 80 snapshots are with $S_3^t = 1$. The maximum likelihood estimation will calculate as $P(S_3^t = 1 | S_2^{t-2} = 1, S_3^{t-1} = 0) = 0.8$ and $P(S_3^t = 0 | S_2^{t-2} = 1, S_3^{t-1} = 0) = 0.2$, respectively.

C. Event Monitor

Finally, the Event Monitor module takes the DIG and runtime device events as inputs, and initiates anomaly detection. The main idea is that for each incoming event, it first leverages Eq.(1) to calculate the anomaly score. Then

Algorithm 1 TemporalPC

```
Input: The graph snapshots $G$, the maximum time lag $\tau$,
    the outcome $S_i^t$, and threshold $\alpha$
Output: The set of causes $Ca(X_i^t)$
Function CI(X,Y,Z)
    Test $X \perp\!\!\!\perp Y|Z$ using test statistic measure $I$
    return p-value
Skeleton Construction
Initialize the preliminary set of causes $Ca(S_i^t) = \{S_j^{t-k} : j \in \{1,...,n\}, k \in \{1,...,\tau\}\}$
Let $l = -1$
while $l \leq n$ do
    $l = l + 1$
    if $|Ca(S_i^t)| - 1 < l$ then
        Break
    end if
    for all parent $S_j^{t-k} \in Ca(S_i^t)$ do
        for all subset $C_s \subseteq Ca(S_i^t) \backslash \{S_j^{t-k}\}$ with $|C_s| = l$ do
            p-value ← $CI(S_j^{t-k}, S_i^t, C_s)$
            if p-value $> \alpha$ then
                Remove $S_j^{t-k}$ from $Ca(S_i^t)$
                Break
            end if
        end for
    end for
end while
return $Ca(C_i^t)$
```

it uses Definition 2 and Definition 3 to determine whether the event is anomalous.

Phantom state machine. To calculate the anomaly score, the module dynamically tracks the latest graph snapshot, which is achieved by the phantom state machine. Specifically, the machine maintains a memory for storing the recent $\tau + 1$ system states. When an event e^t comes, the phantom state machine first preprocesses the event and derives the current system state S^t. Then the machine records it and slides the stored system states to remove the one with the oldest timestamp. By doing so, the machine continuously tracks the latest snapshot of the graph, i.e., $G^t = (S^{t-\tau}, ..., S^t)$. The phantom state machine also supports the query of device states. For example, given a runtime event $\{S_i^t = s_i^t\}$, the state machine can fetch the values of its causes $ca(S_i^t)$ from G^t.

Score threshold calculator. According to Definition 2, a score threshold is needed to distinguish the contextual anomaly and the legitimate event. To this end, we resort to the logged events. We first calculate the anomaly score for each logged event. Then we rank these scores and get a score distribution. Finally, we select the qth percentile as the score threshold [72]. The parameter q can be interpreted as the confidence level about the logged events' normality. Since there are few anomalies in the logged events (Section IV-B), one can set a significant value, e.g., 99.

Anomaly detection procedure. Algorithm 2 shows the pseudocode of the detection procedure. Specifically, it takes an event stream, the constructed DIG, the calculated score threshold, and an integer $k_{\max} \geq 1$ as inputs. The outputs are the alarms about the detected contextual and collective anomalies. In particular, the parameter $k_{\max}$ specifies the maximum length of the collective anomaly which users want to detect. When $k_{\max} = 1$, the algorithm seeks to detect the contextual anomaly. When $k_{\max} > 1$, besides the detection of the contextual anomaly, the algorithm further tracks the collective anomaly with a maximum length of $k_{\max} - 1$. The setting of $k_{\max}$ depends on the specific smart home usage. A possible way is that users can use existing automation analysis techniques [32], [37], [59] to identify the maximum length of the automation chain and let $k_{\max}$ be the maximum length. By doing so, when the longest automation chain is maliciously triggered, the algorithm can reconstruct the whole chain.

The detection procedure starts with initializing a phantom state machine PM (Line 1). Moreover, it initializes an empty list W for recording the detected anomalies. Whenever an event $\{S_i^t = s_i^t\}$ comes, PM first updates the graph snapshot and fetches the values of $Ca(S_i^t)$ (Line 4). Then the procedure looks up the conditional probability table in the DIG, and calculates the anomaly score using Eq.(1) (Line 5). Depending on the length of the event list $|W|$, the procedure interprets the score differently. When the list is empty, it checks whether the score is larger than the threshold c. If that is the case, the event will be reported as a contextual anomaly and appended to the

Algorithm 2 k-Sequence Anomaly Detection

```
Input: Runtime event stream $E$, the device interaction
    graph $\mathcal{G}$, the score threshold $c$, and the maximum
    length $k_{\max}$
1:  Initialize a phantom state machine $PM$, and an empty anomaly list $W \leftarrow []$
2:  for all $e^t : \{S_i^t = s_i^t\} \in E$ do
3:      $PM$.Update($e^t$)
4:      $ca(S_i^t) \leftarrow PM$.Get($Ca(S_i^t)$)
5:      $score \leftarrow 1 - P(S_i^t = s_i^t | Ca(S_i^t) = ca(S_i^t))$
6:      if ($|W| > 0$ and $score < c$) or ($|W| = 0$ and $score \geq c$) then
7:          $W$.append(($e^t, ca(S_i^t)$))
8:      end if
9:      if $|W| = k_{\max}$ or ($0 < |W| < k_{\max}$ and $score \geq c$) then
10:         Raise an alarm and report $W$ to users for amendment
11:         $W \leftarrow []$
12:     end if
13: end for
```

list. Moreover, additional information for later anomaly interpretation is recorded (i.e., the value of its causes). However, if the list is not empty, the procedure checks whether the score is smaller than the score threshold. If it holds, the event will be determined as a member of the collective anomaly, and will also be appended to W. When the list size $|W|$ is equal to $k_{\max}$ or an abrupt event with a significant anomaly score happens during the collective anomaly detection, the tracking process ends, and the anomalous event list is reported to users for device recovery.

D. Computational complexity analysis

We here analyze the computational complexity of the Interaction Miner and Event Monitor modules. Interaction Miner involves the skeleton construction process and the CPT estimation process, and the former plays a dominant role in determining the computational complexity. Specifically, since our TemporalPC algorithm follows PC's conditional independence test framework, its computational complexity is of the same order as PC. Let n be the number of nodes in DIG, and k be the maximum degree of any node. The maximum number of conditional independence tests required by TemporalPC is $O(n^k)$ [73]. For Event Monitor, the computational complexity for validating a runtime event is $O(1)$, as it only involves a table lookup for calculating an anomaly score and a comparison operation with a constant threshold. Considering the realistic sources of device interactions (Section I), the value of k is usually limited, and the time complexity of our approach can be polynomial. Moreover, there have been attempts [74]–[76] which aim to improve the scalability of PC. Their solutions can be easily integrated into our TemporalPC algorithm. As a result, besides the smart home, our system can also be applied in other large-scale smart applications where the number of devices is significant, e.g., industrial IoT.

VI. Evaluation

A. Experimental Setup

We use open-source smart home datasets for the evaluation. Specifically, we find two real-world testbeds which invite a participant to live in and collect the generated device events: CASAS [77] and ContextAct [78]. Table I shows an overview of the deployed devices. CASAS collected 32,388 device events during a 30-day living experience, while ContextAct collected 54,748 events for seven days. Since the purpose of these testbeds is to study users' daily-living activities, there are few anomalies in the logged events. Moreover, as ContextAct is an advanced testbed with diverse types of devices, we mainly present its evaluation results in this section. The results of CASAS are further presented in our technical report [79].

TABLE I: Overview of device information

Abbr.	Attributes	# devices (CASAS)	# devices (ContextAct)	Value type	Description
S	Switch	0	2	Discrete	Change of actuators
PE	Presence Sensor	7	5	Discrete	Movement detection
C	Contact Sensor	1	2	Discrete	Door/window state
D	Dimmer	0	2	Responsive Numeric	Change of lights
W	Water Meter	0	1	Responsive Numeric	Water usage
P	Power Sensor	0	6	Responsive Numeric	Appliance usage (e.g., stove and fridge)
B	Brightness Sensor	0	4	Ambient Numeric	Luminosity level

TABLE II: Automation rules in ContextAct

Rule ID	Description
R1	If anyone reaches the living room, activates dishwasher
R2	If anyone leaves bathroom, activate stove
R3	If the heater is on, activate bedroom player
R4	If anyone opens the fridge door, turn on the kitchen light
R5	If the kitchen is bright, turn on the bathroom light
R6	If bedroom player is deactivated, activate electric curtain
R7	If the electric curtain is activated, turn on the dining room light

Automation rule generation. Note that both testbeds do not install any automation rule. As a result, it is infeasible to evaluate CausalIoT's performance for automation mining and detection of automation-related anomalies (e.g., chained automation execution). To address it, we inject device events that simulate a set of automation rules and their functionalities. By doing so, we aim to generate a synthetic dataset that contains the original traces and the execution of the installed automation rules. We first identify the devices which are suitable for being the triggering and action devices, respectively. In particular, the brightness and the presence sensors are not suitable for the action device, as they are not bound to any actuator. Then we randomly select the triggering and action devices to generate automation rules. Eventually, we generate 12 automation rules, and Table II shows several examples. Note that some rules are chained together. For instance, $R6$ and $R7$ are chained together for the event of the electric curtain activation. As another example, $R4$ and $R5$ are chained together for the common physical channel, i.e., the high brightness in the kitchen.

Finally, we traverse the original dataset and identify a set of candidate positions where the last event matches the trigger of the installed automation rules. At each candidate position, an event that simulates the action device's behavior is supposed to be injected. However, some candidate positions are not suitable for event injection. Specifically, we check the real-world automation execution and find that the rule will not be executed if the action device's state has already followed the automation rule. For instance, given that the stove has already been activated, if the presence sensor in the bathroom was just deactivated, rule R_2 would not be executed. Therefore, we further check the action device's state at each position and inject the event of the action device if its state does not follow the automation execution. Totally 5,004 events are injected into the dataset.

Event preprocessing. The raw events are preprocessed as specified in Section V-A. We construct the IoT time series and split them for training and testing purposes. In particular, 80% time series are used as the training data for DIG construction, while the remaining 20% time series are used for the evaluation of anomaly detection.

Ground truth construction. Finally, the evaluation requires a set of ground-truth device interactions. We traverse all the neighboring events and extract the device pairs as the candidate interaction. Then for each candidate interaction, we manually examine the involved devices by asking the following questions. (1) Is there any daily-life activity in which users operate these two devices sequentially? (2) Do they work on the same physical channel? (3) Do they form the logic of any automation rule? If the candidate interaction passes any test, it will be accepted as a ground-truth interaction. As a result, we identified 196 ground-truth interactions.

B. Interaction Mining Evaluation

With the training data, CausalIoT initiates the TemporalPC algorithm for DIG construction (Section V-B). It takes two parameters as inputs: The maximum time lag τ and the significance threshold α. In particular, we set $\tau = 2$ as it is large enough to cover the period for the interaction execution. We set $\alpha = 0.001$ since it is a common practice for stringent conditional independence tests [45], [80]. The evaluation metric we used is the *precision* and *recall*. Specifically, for each ground-truth interaction, if the graph contains an interaction that matches the cause and outcome, it is counted as a *true positive* (TP). Otherwise, a *false negative* (FN) is counted. On the other hand, for each interaction encoded in the graph, if it is not a ground-truth interaction, we will count it as a *false positive* (FP). Given the above definition, the precision is calculated as $\frac{TP}{TP+FP}$, and the recall is $\frac{TP}{TP+FN}$.

The evaluation result shows that CausalIoT successfully identifies 190 interactions with 95.9% precision and 97.0% recall. In particular, for all the 12 automation rules, CausalIoT successfully identified them. We further categorize the identified interactions according to their sources, which are listed in Table III. (1) The user interaction encodes the device usage pattern for different user activities. For example, $P(\mathrm{S}^{t}_{\text{player}} = 0|\mathrm{P}^{t-2}_{\text{curtain}} = 1) = 0.836$ captures the user preference when she is going to sleep, i.e., turning off the music. (2) The physical interaction describes how the devices affect and sense the brightness channel. For instance, the light activation increases the brightness, and the brightness sensor senses the change. (3) The automation interaction encodes the functionality of the installed automation rule. For example, $P(\mathrm{P}^{t}_{\text{stove}} = 1|\mathrm{PE}^{t-1}_{\text{bathroom}} = 0) = 0.946$ shows the influence of the triggering device (presence sensor) on the action device (stove), which follows the logic of rule $R2$. (4) Finally, autocorrelation usually contains information about device usage time. For instance, $P(\mathrm{W}^{t}_{\text{sink}} = 0|\mathrm{W}^{t-2}_{\text{sink}} = 1) = 0.792$ shows that the user usually finishes the usage of the sink within a short period. On the contrary, $P(\mathrm{P}^{t}_{\text{stove}} = 0|\mathrm{P}^{t-2}_{\text{stove}} = 1) = 0.116$ implies that the usage of the stove usually lasts for a long time.

CausalIoT also successfully rejects 280 candidate interactions. Specifically, it rejects 87 of them because the involved device states are independent, which are reflected by the high p-values. For example, it rejects the candidate interaction $\mathrm{B}_{\text{living}} \rightarrow \mathrm{W}_{\text{kitchen}}$ because the p-value for testing $B_{\text{living}} \perp\!\!\!\perp W_{\text{kitchen}}$ is 0.953, which is far more significant than the threshold 0.001. The remaining 193 interactions are shown to be spurious because CausalIoT identifies some conditioning sets which make the involved devices conditionally independent. Most spurious interactions are attributed to the intermediate factor. For instance, the interaction $\mathrm{S}_{\text{player}} \rightarrow \mathrm{D}_{\text{bathroom}}$ is rejected because the device $\mathrm{P}_{\text{heater}}$ acts as an intermediate device in the chain $S_{\text{player}} \rightarrow P_{\text{heater}} \rightarrow D_{\text{bathroom}}$. On the other hand, some spurious interactions are attributed to a common cause. For example, CausalIoT rejects the interaction $\mathrm{B}_{\text{living}} \rightarrow \mathrm{PE}_{\text{dining}}$, because they share a common cause $\mathrm{B}_{\text{living}} \leftarrow \mathrm{D}_{\text{living}} \rightarrow \mathrm{PE}_{\text{dining}}$.

Finally, CausalIoT reports eight false positives and misses six ground-truth interactions, respectively. We find that most false positives are related to the brightness sensor. For instance, CausalIoT reports a spurious interaction $\mathrm{B}_{\text{living}} \rightarrow \mathrm{B}_{\text{bedroom}}$, while the change of the living room's brightness does not necessarily affect that of the bedroom. A further investigation shows that these spurious interactions are attributed to unmeasured environmental factors (e.g., sunrise and weather). These factors can be the common cause of the brightness sensors in different rooms. However, the testbed did not measure them, and the interaction graph did not consider them. We will discuss potential solutions in our technical report [79]. For missing interactions, they are mainly due to the low occurrence of the interaction execution. For example, there are only two instances of the $\mathrm{P}_{\text{fridge}} \rightarrow \mathrm{W}_{\text{sink}}$ execution, which makes it difficult to detect the state dependency between these two devices.

C. Contextual Anomaly Detection Evaluation

Anomaly generation. We survey the previously reported security threats and categorize four malicious cases as listed in Table IV: (1) sensor fault [21], [22], [81], (2) burglar intrusion [13], (3) remote control [4], [18], [58], and (4) malicious automation rule [59], [82], [83]. To simulate the first three cases, we first traverse the testing data and randomly select 5,000 candidate positions. At each position, we randomly generate a spoofed device event, which promotes the unsolicited device state transition. These events imply anomalous sensor readings (e.g., fluctuating brightness levels), unexpected presence (e.g., presence-on and contact-open events), and unsolicited actuator operations (e.g., switch on and off), respectively. Then we inject the corresponding anomalous system state into the time series. As a result, we create three testing time series, each of which contains 16,950 system states. For the case of malicious automation rules, we randomly generate a set of malicious rules that aim to control the devices under some conditions (e.g., if users leave the kitchen, activate the stove). Then we follow the procedure as specified in Section VI-A and simulate their executions. Eventually,

TABLE III: Identified device interactions in ContextAct

Sources	Category	Number	Example	Description
User Activity	Use-after-Use	42	$\mathrm{P_{heater}} \rightarrow \mathrm{D_{bathroom}}$, $\mathrm{P_{curtain}} \rightarrow \mathrm{S_{player}}$	Sequential operations over devices
	Use-after-Move	24	$\mathrm{C_{kitchen}} \rightarrow \mathrm{P_{oven}}$, $\mathrm{PE_{bathroom}} \rightarrow \mathrm{D_{bathroom}}$	Move to room and operate devices
	Move-after-Use	45	$\mathrm{D_{bathroom}} \rightarrow \mathrm{PE_{living}}$, $\mathrm{P_{dishwasher}} \rightarrow \mathrm{PE_{dining}}$	Operate devices then move
	Move-after-Move	27	$\mathrm{PE_{kitchen}} \rightarrow \mathrm{PE_{dining}}$, $\mathrm{PE_{bedroom}} \rightarrow \mathrm{PE_{living}}$	Traces of user movements
Physical Channel	Brightness	18	$\mathrm{D_{living}} \rightarrow \mathrm{B_{living}}$, $\mathrm{P_{stove}} \rightarrow \mathrm{B_{kitchen}}$	Change and sense the brightness level
Automation	N/A	12	$\mathrm{PE_{bathroom}} \rightarrow \mathrm{P_{stove}}$, $\mathrm{P_{heater}} \rightarrow \mathrm{S_{player}}$	Logics of installed automation rules
Autocorrelation	N/A	22	$\mathrm{W_{sink}} \rightarrow \mathrm{W_{sink}}$, $\mathrm{P_{oven}} \rightarrow \mathrm{P_{oven}}$	Devices' state flipping

TABLE IV: Contextual anomaly and detection accuracy

ID	Case	Anomaly Description	Creation Method	Accuracy	Precision	Recall	F1
1	Sensor Fault	Fluctuating brightness level	Insert anomalous sensor readings	0.978	0.964	0.960	0.962
2	Burglar Intrusion	Suspicious presence report	Insert unexpected presence events	0.978	0.957	0.968	0.962
3	Remote Control	Ghost actuator operation	Insert flipped state events	0.972	0.945	0.962	0.953
4	Malicious Rule	Execution of hidden rules	Insert conditional transition events	0.989	0.943	0.984	0.963

we generate 2,000 malicious events and create a testing time series with 13,950 system states.

Evaluation results. With the testing data, CausalIoT initiates 1-sequence algorithm for the detection of contextual anomalies, and the results are shown in Table IV. For each anomaly case, we first compare the injected positions and the alarming positions. Then we check the number of true, false, and missing alarms. The results show that CausalIoT achieves a high detection accuracy with an average of 95.2% precision and 96.8% recall. Below we describe several examples showing how CausalIoT works to pinpoint the anomaly.

(1) *Fluctuating brightness report.* CausalIoT raises an alarm for a high brightness reading after checking the time-lagged states of the dimmer, oven, and stove. Given the deactivation of these causes, CausalIoT determines that the event violates the physical interactions (in terms of a low likelihood of 0.02%). Moreover, the reported context (i.e., values of causes) implies the potential fault of the sensor.

(2) *Covert door control.* A kitchen-door event is reported after CausalIoT checks the states of the presence sensors deployed in neighboring rooms. Given that no presence was detected, CausalIoT determines that the door event violates the users' normal traces. Similarly, it reports a mysterious dimmer event after checking the presence sensors deployed in the bedroom and study room. The context shows that these devices are likely to suffer from the remote control.

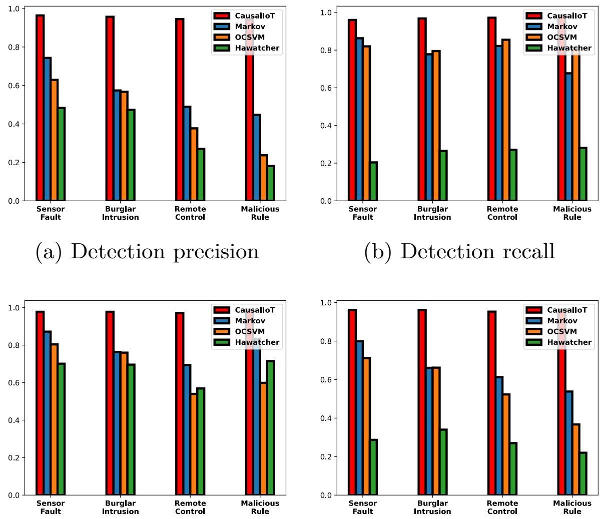
(3) *Malicious automation execution.* CausalIoT finds that every time the presence sensor in the kitchen is deactivated, the stove will be turned on. It further checks the state of the fridge (as users usually use them sequentially), and its deactivation state confirms that the stove event is anomalous.

We also examine the false and missing alarms. Specifically, the false alarms are mainly due to user behavioral deviations. For example, the training data shows that users rarely (0.26%) move to the living room after operating the stove. However, the behavior becomes frequent (7.61%) in the testing data. Since the $\mathrm{P_{stove}} \rightarrow \mathrm{PE_{bedroom}}$ interaction has changed its execution, the stove event is regarded as an anomaly by the outdated interaction graph. While it is arguable whether false alarms due to behavioral deviations should be counted [23], we still count them because they do not stem from our anomaly injection scheme. We also discuss potential solutions in our technical report [79]. On the other hand, the missing alarm is mainly caused by spurious brightness interactions. For example, a spurious $\mathrm{B_{dining}} \rightarrow \mathrm{C_{bedroom}}$ interaction can mitigate the abnormal level of a malicious door-unlocking event that happens in the daytime.

Baseline comparison. We select three popular anomaly detection methods as the baseline and make comparisons with CausalIoT. These baselines show potentials for detecting the contextual anomaly [21]–[23], [84], and we categorize them according to the taxonomy of the detection technique [85].

- *(Stochastic Learning) kth-order Markov chain* [21], [22]. It first uses the training data to estimate the likelihood of the current system state given the preceding k system states. If a runtime event implies a system state transition that never happens, it is reported as an anomaly. In particular, we set $k = \tau$ in our evaluation.
- *(Classic Machine Learning) One class support vector machine* [84]. It takes the training system states as inputs and learns a classification boundary. Then for each runtime event, OCSVM performs a classification task and determines whether the system state is anomalous or not.
- *(Data Mining) HAWatcher* [23]. The rule-based method uses a set of inferred rules to detect device anomalies. It first utilizes background knowledge (e.g., the device location) and association tests to learn a set of *event-to-state* rules (e.g., $E^{plug}_{high} \rightsquigarrow S^{heater}_{on}$). Then for each runtime event, HAWatcher identifies the related rules and validates the corresponding state. If any rule is violated, the event is marked as anomalous.

The comparison result is shown in Figure 5. CausalIoT achieves the best performance for all the anomaly cases. For the baselines, the Markov model achieves a relatively good recall. However, it heavily relies on the temporal order among events, and many disordered IoT events (e.g., the periodic brightness report) can cause unexpected system state transitions. As a result, a large number of false alarms are generated. Our method, on the contrary, carefully selects device interactions based on the persistent state dependencies. As a result, it is more resilient to disordered events. OCSVM also achieves a good recall. However, the number of false positives is large (56.2% in average), which makes the detection system useless. Moreover, it cannot provide any information for anomaly interpretation. Finally, HAWATCHER achieves the lowest accuracy, and the main reason is the flawed rule construction process. Specifically, it rejects a large number of device interactions for rule construction because they do not follow the spatial constraint (e.g., $\mathrm{PE}_{\mathrm{kitchen}} \rightarrow \mathrm{PE}_{\mathrm{dining}}$) or the functionality dependency (e.g., $\mathrm{P}_{\mathrm{stove}} \rightarrow \mathrm{B}_{\mathrm{kitchen}}$). However, these interactions are beneficial for profiling normal device behavior. The result shows that the smart home is a complex system, and there exists no reliable background knowledge for understanding the inter-device relationship. On the contrary, our work presents a powerful method for mining device interactions without background knowledge.



(a) Detection precision

(b) Detection recall

(c) Detection accuracy

(d) Detection F1 score

Fig. 5: Comparisons for contextual anomaly detection

D. Collective Anomaly Detection

Finally, we initiate the evaluation for collective anomaly detection. We simulate three malicious cases as follows. (1) burglar wandering. After the burglar breaks into the house, he opens the doors and searches for each room, which creates a sequence of contact/presence events [13]. (2) actuator manipulation. The attacker initiates a sequence of unsolicited actuator operations for some malicious task. For instance, he can turn the light on and off and use the light as a covert communication channel to transfer information [4]. As another example, he activates a set of devices used in an activity of daily life (e.g., the kitchen light, stove, and fridge for cooking), such that he can camouflage himself as the normal user and cause severe consequences (e.g., fire accident). (3) chained automation rules. With the knowledge of the deployed automation rules, the attacker can selectively target a triggering device and cause the chained automation execution [32], [36]. For example, after he hijacks the fridge and covertly activates it, the fridge misbehavior triggers the automation $R4$ and turns on the light. The high brightness further triggers the execution of $R5$, which eventually turns on the bathroom light.

TABLE V: Collective anomaly and detection accuracy

ID	Case Description	$k_{\max}$	Avg. anomaly length	% detected anomalies	% tracked anomalies	Avg. detection length
1	Burglar Wandering	2	2.000	93.2%	93.2%	2.000
		3	2.481	92.6%	92.3%	2.474
		4	3.003	88.6%	87.6%	2.968
2	Illegal Actuator Operations	2	2.000	89.3%	89.3%	2.000
		3	2.509	88.1%	82.7%	2.452
		4	2.982	84.3%	80.5%	2.912
3	Chained Automation Rules	2	2.000	98.7%	98.7%	2.000
		3	2.454	97.0%	97.0%	2.449
		4	3.001	95.5%	95.5%	3.001

For each malicious case, we first randomly select 1,000 positions in testing data to generate contextual anomalies. Based on the ground truth, we propagate each contextual anomaly and generate a sequence of events that follow the interaction execution at smart homes. For instance, given that a bedroom presence is injected as the contextual anomaly, we further simulate the attacker's trace and inject presence events in the study room and the living room. Note that the length of these sequences is bounded by a parameter $k_{\max}$, which quantifies the maximum number of interaction executions and affected devices. For instance, given the case of chained automation rules and $k_{\max} = 3$, we simulate the scenario where at most two automation rules are chained together, and at most three devices are affected. In this evaluation, $k_{\max}$ ranges from two to four, and the average length of the anomaly chain is listed in Table V.

Finally, CAUSALIOT initiates the k-sequence anomaly detection with $k = k_{max}$. Since we are the first to report abnormal interaction executions and initiate collective anomaly detection, no baselines are selected for comparison. In particular, we are interested in the following two questions about the performance of CAUSALIOT. (1) Can it detect the existence of abnormal interaction executions, i.e., any subsequences of the collective anomalies? (2) Can it track the whole sequence and identify all the affected devices? Table V shows the evaluation result. CAUSALIOT achieves an average of 91.9% detection accuracy for the above three malicious cases. Moreover, CAUSALIOT successfully reconstructs 90.8% of the detected anomalies. For the remaining 9.2% anomalies, while not fully tracked, CAUSALIOT can still collect most of the involved events. One can check that the average detection length is close to the average anomaly length with an average of only 0.17 missing events.

We further investigate the root cause of the detection result for missing anomalies. Specifically, some missing alarms are attributed to the missing detection of the contextual anomaly. As a result, CAUSALIOT mistakenly regards the whole sequence as a normal interaction execution and does not initiate the collective anomaly tracking. The remaining missing alarms are due to the inability of collective event tracking. That is, after CausalIoT detects the contextual anomaly, the anomaly score of the consecutive event is still high. As a result, the consecutive event acts as an abrupt event and stops anomaly tracking. In summary, based on the accurate contextual anomaly detection and interaction profiling, CausalIoT performs very well for collective anomaly detection.

VII. Related Work

A. Causality

The understanding of the causal relationship has evolved from the deterministic functional equations [86]–[88] to the stochastic functional equations [44], [89] in the last century. Specifically, [44] proposed a *structural causal model* which uses structural equations to describe the causal relationship among variables. Researchers also base on the bayesian network [70] and propose a graphical representation of the model, i.e., the causal bayesian network (or causal graph for simplicity). Compared with the bayesian network, the edges in the causal graph describe more complex relationships among variables: They encode both the conditional dependence relationship and causal relationship in terms of intervention [90]. Due to its transparent and interpretable representation of causal relationships, the structural causal model has been widely used in economics [91], [92], epidemiology [93], [94], and social sciences [95], [96]. Our work makes a first step towards utilizing the causal model in the IoT ecosystem to interpret the prevalent device interactions.

Prior works propose various methods to infer the causal model [47], [48], [73], [97]–[100]. Specifically, constraint-based methods aim to efficiently search for a causal graph [47], [48], [73], and they can encode various independence test methods to handle different types of data/dependence formats. Equation-based methods leverage the noise term in the structural equations and test asymmetry between the causal and effect [97], [98]. However, they cannot identify the causal relationship without any assumption on the function [101]. Recently, representation learning methods have been proposed [99], [100]. Generally speaking, any methods described above can be extended to handle the temporal setting. Considering their benefits and popularity, we extend one of the most famous constraint-based methods, the PC algorithm, and use it for causal discovery for device interaction mining.

B. Detection of Smart Home Security Threats

Many prior trials focus on detecting threats in specific IoT applications and functionalities. In particular, [10], [32], [36]–[38], [59], [102]–[104] extensively study the security issues of the automation rules. However, their methods largely depend on the accessibility of the source codes, and cannot handle the case where the automation rules come from multiple platforms and each platform uses different programming languages. Instead, CausalIoT uses a data-driven approach to adequately address the challenge of automation inference. Moreover, our work can detect and track unsolicited automation executions at runtime, which these works cannot address.

[39], [65], [105] leverage the idea of "contextual integrity" and design a set of security policies for device access control. Specifically, they highlight that one can use external conditions (e.g., user location or the state of other devices) to justify a device operation. Unfortunately, they require significant human efforts (e.g., user survey or runtime application prompt) to infer the user preference for device operations. Our work focuses on studying the inter-device relationship for profiling normal device behavior. We use the causal primitive to describe the device interaction, and we design a TemporalPC algorithm for automatic DIG construction. It can also handle more sophisticated security threats which are not considered by prior work (e.g., unsolicited interaction execution).

Finally, many anomaly detection systems utilize data mining techniques to profile the normal system behavior [19]–[22], [106]. The main difference between these detectors and our work is that we exploit the nature of the widespread device interactions and use them to profile legitimate smart home usage. As a result, our work achieves better detection accuracy. Moreover, the knowledge of device interactions can provide helpful information for explaining the detection result, which was unsupported by prior work.

VIII. Conclusion

One unique feature of smart homes is the high volume of device interactions. Since they are well-suited for profiling normal device behavior, a promising direction is to utilize them for device anomaly detection. In this work, we propose the device interaction graph and take the first step toward automated graph construction from the logged device events. Moreover, we formally categorize two types of smart home anomalies and utilize the interaction graph for runtime anomaly detection. We built a prototype CausalIoT and evaluated it on the real-world testbed against seven malicious cases. The result shows that it outperforms existing methods and can handle new types of anomalies that were not considered by prior work.

Acknowledgment

The work of John C.S. Lui is supported in part by the RGC R4032-18. The work of Bin Yuan is supported by National Key Research and Development Plan of China under Grant 2022YFB3103403. We would also like to express our gratitude to anonymous reviewers for their constructive comments.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE communications surveys & tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] Statista, "Number of smart homes forecast in the world from 2017 to 2025," https://www.statista.com/forecasts/887613/.

[3] "Programmable thermostat scheduler." https://community.smartthings.com/t/release-5-2-day-programmable-thermostat-scheduler-weekday-weekend-with-remote-temperature-sensor-for-each-schedule/6780.

[4] E. Ronen and A. Shamir, "Extended functionality attacks on iot devices: The case of smart lights," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 3–12.

[5] M. Kim, D. Kim, E. Kim, S. Kim, Y. Jang, and Y. Kim, "Firmae: Towards large-scale emulation of iot firmware for dynamic analysis," in *Annual Computer Security Applications Conference*, 2020, pp. 733–745.

[6] N. Redini, A. Machiry, R. Wang, C. Spensky, A. Continella, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Karonte: Detecting insecure multi-binary interactions in embedded firmware," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1544–1561.

[7] J. Wang, Z. Li, M. Sun, and J. C. Lui, "Zigbee's network rejoin procedure for iot systems: Vulnerabilities and implications," 2018.

[8] D.-G. Akestoridis, M. Harishankar, M. Weber, and P. Tague, "Zigator: analyzing the security of zigbee-enabled smart homes," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 77–88.

[9] J. Margulies, "Garage door openers: An internet of things case study," *IEEE Security & Privacy*, vol. 13, no. 4, pp. 80–83, 2015.

[10] W. Ding and H. Hu, "On the safety of iot device physical interaction control," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 832–846.

[11] T. Sugawara, B. Cyr, S. Rampazzi, D. Genkin, and K. Fu, "Light commands: Laser-based audio injection attacks on voice-controllable systems," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2631–2648.

[12] "Undesired poltergeist lighting effect," https://community.smartthings.com/t/undesired-poltergeist-lighting-effect/24132, 2017.

[13] L. Babun, A. K. Sikder, A. Acar, and A. S. Uluagac, "Iotdots: A digital forensics framework for smart environments," *arXiv preprint arXiv:1809.00745*, 2018.

[14] "Smartthings freaking out," https://community.smartthings.com/t/smartthings-freaking-out/101358, 2017.

[15] S. community, "Is there a simple way to trace source of commands?" https://community.smartthings.com/t/is-there-a-simple-way-to-trace-source-of-commands-which-account-or-integration/101334.

[16] "Mysterious "lock was unlocked electronically," https://community.smartthings.com/t/mysterious-lock-was-unlocked-electronically/100942, 2017.

[17] "When st glitches become major safety/fire hazard," https://community.smartthings.com/t/when-st-glitches-become-major-safety-fire-hazard/43109, 2017.

[18] "Ghost switch - how to find reason for it turning on by itself?" https://community.openhab.org/t/ghost-switch-how-to-find-reason-for-it-turning-on-by-itself/136770.

[19] S. Munir and J. A. Stankovic, "Failuresense: Detecting sensor failure using electrical appliances in the home," in *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE, 2014, pp. 73–81.

[20] P. A. Kodeswaran, R. Kokku, S. Sen, and M. Srivatsa, "Idea: A system for efficient failure management in smart iot environments," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 2016, pp. 43–56.

[21] A. K. Sikder, H. Aksu, and A. S. Uluagac, "6thsense: A context-aware sensor-based attack detector for smart devices," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 397–414.

[22] J. Choi, H. Jeoung, J. Kim, Y. Ko, W. Jung, H. Kim, and J. Kim, "Detecting and identifying faulty iot devices in smart home with context extraction," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 610–621.

[23] C. Fu, Q. Zeng, and X. Du, "Hawatcher: Semantics-aware anomaly detection for appified smart homes," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[24] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No. 99CB36344)*. IEEE, 1999, pp. 120–132.

[25] K. Yang, S. Kpotufe, and N. Feamster, "An efficient one-class svm for anomaly detection in the internet of things," *arXiv preprint arXiv:2104.11146*, 2021.

[26] P. Spirtes, "Conditional independence in directed cyclic graphical models for feedback," 1994.

[27] J. Pearl, *Causality*. Cambridge university press, 2009.

[28] "openhab," https://www.openhab.org/, 2022.

[29] SmartThings, "Smartthings," https://www.smartthings.com/.

[30] "Smartthings device capability reference," https://developer-preview.smartthings.com/docs/devices/capabilities/capabilities-reference, 2018.

[31] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decentralized action integrity for trigger-action iot platforms," in *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.

[32] W. Ding, H. Hu, and L. Cheng, "Iotsafe: Enforcing safety and security policy with real iot physical interaction discovery," in *the 28th Network and Distributed System Security Symposium (NDSS 2021)*, 2021.

[33] Z. B. Celik, P. McDaniel, G. Tan, L. Babun, and A. S. Uluagac, "Verifying internet of things safety and security in physical spaces," *IEEE Security & Privacy*, vol. 17, no. 5, pp. 30–37, 2019.

[34] Z. B. Celik, E. Fernandes, E. Pauley, G. Tan, and P. McDaniel, "Program analysis of commodity iot applications for security and privacy: Challenges and opportunities," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–30, 2019.

[35] M. Balliu, M. Merro, M. Pasqua, and M. Shcherbakov, "Friendly fire: cross-app interactions in iot platforms," *ACM Transactions on Privacy and Security (TOPS)*, vol. 24, no. 3, pp. 1–40, 2021.

[36] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 411–423.

[37] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 147–158.

[38] M. O. Ozmen, X. Li, A. Chu, Z. B. Celik, B. Hoxha, and X. Zhang, "Discovering iot physical channel vulnerabilities," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[39] S. Manandhar, K. Moran, K. Kafle, R. Tang, D. Poshyvanyk, and A. Nadkarni, "Towards a natural perspective of smart homes for practical security and safety analyses," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 482–499.

[40] P. Spirtes and K. Zhang, "Causal discovery and inference: concepts and recent methodological advances," in *Applied informatics*, vol. 3, no. 1. SpringerOpen, 2016, pp. 1–28.

[41] N. Aung, M. Y. Khanji, P. B. Munroe, and S. E. Petersen, "Causal inference for genetic obesity, cardiometabolic profile and covid-19 susceptibility: a mendelian randomization study," *Frontiers in genetics*, p. 1417, 2020.

[42] J. M. Cordero, V. Cristóbal, and D. Santín, "Causal inference on education policies: A survey of empirical studies using pisa, timss and pirls," *Journal of Economic Surveys*, vol. 32, no. 3, pp. 878–915, 2018.

[43] H. R. Varian, "Causal inference in economics and marketing," *Proceedings of the National Academy of Sciences*, vol. 113, no. 27, pp. 7310–7315, 2016.

[44] J. Pearl, "Causal inference in statistics: An overview," *Statistics surveys*, vol. 3, pp. 96–146, 2009.

[45] C. Glymour, K. Zhang, and P. Spirtes, "Review of causal discovery methods based on graphical models," *Frontiers in genetics*, vol. 10, p. 524, 2019.

[46] D. Malinsky and D. Danks, "Causal discovery algorithms: A practical guide," *Philosophy Compass*, vol. 13, no. 1, p. e12470, 2018.

[47] P. Spirtes and C. Glymour, "An algorithm for fast recovery of sparse causal graphs," *Social science computer review*, vol. 9, no. 1, pp. 62–72, 1991.

[48] D. Colombo, M. H. Maathuis *et al.*, "Order-independent constraint-based causal structure learning." *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3741–3782, 2014.

[49] C. Meek, "Causal inference and causal explanation with background knowledge," *arXiv preprint arXiv:1302.4972*, 2013.

[50] S. M. Ross, J. J. Kelly, R. J. Sullivan, W. J. Perry, D. Mercer, R. M. Davis, T. D. Washburn, E. V. Sager, J. B. Boyce, and V. L. Bristow, *Stochastic processes*. Wiley New York, 1996, vol. 2.

[51] D. Entner and P. O. Hoyer, "On causal discovery from time series data using fci," *Probabilistic graphical models*, pp. 121–128, 2010.

[52] J. Runge, P. Nowack, M. Kretschmer, S. Flaxman, and D. Sejdinovic, "Detecting and quantifying causal associations in large nonlinear time series datasets," *Science advances*, vol. 5, no. 11, p. eaau4996, 2019.

[53] "Door knocker going crazy," https://community.smartthings.com/t/door-knocker-going-crazy/55570.

[54] "Motion detection false positive," https://community.smartthings.com/t/motion-detection-false-positive/119816.

[55] "False positives on motion sensors," https://community.smartthings.com/t/faq-false-positives-on-motion-sensors/19876.

[56] S. community, "Are the poltergeists back?" https://community.smartthings.com/t/october-2017-are-the-poltergeists-back-devices-randomly-turning-on/101402, 2017.

[57] L. Chen, Y. Wang, Q. Cai, Y. Zhan, H. Hu, J. Linghu, Q. Hou, C. Zhang, H. Duan, and Z. Xue, "Sharing more and checking less: Leveraging common input keywords to detect bugs in embedded systems," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 303–319.

[58] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang, "Burglars' iot paradise: Understanding and mitigating security risks of general messaging protocols on iot clouds," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 465–481.

[59] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, and C. A. Gunter, "Charting the attack surface of trigger-action iot platforms," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1439–1453.

[60] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero, "An experimental security analysis of an industrial robot controller," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 268–286.

[61] H. P. Breivold and K. Sandström, "Internet of things for industrial automation–challenges and technical solutions," in *2015 IEEE International Conference on Data Science and Data Intensive Systems*. IEEE, 2015, pp. 532–539.

[62] F. Giannoni, M. Mancini, and F. Marinelli, "Anomaly detection models for iot time series data," *arXiv preprint arXiv:1812.00890*, 2018.

[63] N. Sufyan, N. A. Saqib, and M. Zia, "Detection of jamming attacks in 802.11 b wireless networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, no. 1, pp. 1–18, 2013.

[64] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan, "Casas: A smart home in a box," *Computer*, vol. 46, no. 7, pp. 62–69, 2012.

[65] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. Unviersity, "Contexlot: Towards providing contextual integrity to appified iot platforms." in *NDSS*, vol. 2, no. 2, 2017, pp. 2–2.

[66] M. A. B. Ahmadon, S. Yamaguchi, S. Saon *et al.*, "On service security analysis for event log of iot system based on data petri net," in *2017 IEEE International Symposium on Consumer Electronics (ISCE)*. IEEE, 2017, pp. 4–8.

[67] V. K. Raghu, A. Poon, and P. V. Benos, "Evaluation of causal structure learning methods on mixed data types," in *Proceedings of 2018 ACM SIGKDD Workshop on Causal Discovery*. PMLR, 2018, pp. 48–65.

[68] F. Pukelsheim, "The three sigma rule," *The American Statistician*, vol. 48, no. 2, pp. 88–91, 1994.

[69] G. F. Jenks, "The data model concept in statistical mapping," *International yearbook of cartography*, vol. 7, pp. 186–190, 1967.

[70] R. E. Neapolitan *et al.*, *Learning bayesian networks*. Pearson Prentice Hall Upper Saddle River, NJ, 2004, vol. 38.

[71] I. J. Myung, "Tutorial on maximum likelihood estimation," *Journal of mathematical Psychology*, vol. 47, no. 1, pp. 90–100, 2003.

[72] A. Ogbechie, J. Díaz-Rozo, P. Larrañaga, and C. Bielza, "Dynamic bayesian network-based anomaly detection for in-process visual inspection of laser surface heat treatment," in *Machine learning for cyber physical systems*. Springer, 2017, pp. 17–24.

[73] P. Spirtes, C. N. Glymour, R. Scheines, and D. Heckerman, *Causation, prediction, and search*. MIT press, 2000.

[74] P. Bühlmann, M. Kalisch, and M. H. Maathuis, "Variable selection in high-dimensional linear models: partially faithful distributions and the pc-simple algorithm," *Biometrika*, vol. 97, no. 2, pp. 261–278, 2010.

[75] J. Ramsey, J. Zhang, and P. L. Spirtes, "Adjacency-faithfulness and conservative causal inference," *arXiv preprint arXiv:1206.6843*, 2012.

[76] M. Kalisch and P. Bühlman, "Estimating high-dimensional directed acyclic graphs with the pc-algorithm." *Journal of Machine Learning Research*, vol. 8, no. 3, 2007.

[77] D. J. Cook, N. C. Krishnan, and P. Rashidi, "Activity discovery and activity recognition: A new partnership," *IEEE transactions on cybernetics*, vol. 43, no. 3, pp. 820–828, 2013.

[78] P. Lago, F. Lang, C. Roncancio, C. Jiménez-Guarín, R. Mateescu, and N. Bonnefond, "The contextact@ a4h real-life dataset of daily-living activities," in *International and interdisciplinary conference on modeling and using context*. Springer, 2017, pp. 175–188.

[79] "Technical report." https://github.com/anonymous-dsn23/dsn23-technical-report.

[80] K. Marazopoulou, R. Ghosh, P. Lade, and D. Jensen, "Causal discovery for manufacturing domains," *arXiv preprint arXiv:1605.04056*, 2016.

[81] J. Ye, G. Stevenson, and S. Dobson, "Fault detection for binary sensors in smart home environments," in *2015 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2015, pp. 20–28.

[82] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 636–654.

[83] Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. McDaniel, and M. Smith, "Sok: Lessons learned from android security research for appified software platforms," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 433–451.

[84] J. Inoue, Y. Yamagata, Y. Chen, C. M. Poskitt, and J. Sun, "Anomaly detection for a water treatment system using unsupervised machine learning," in *2017 IEEE international conference on data mining workshops (ICDMW)*. IEEE, 2017, pp. 1058–1065.

[85] S. Schmidl, P. Wenig, and T. Papenbrock, "Anomaly detection in time series: a comprehensive evaluation," *Proceedings of the VLDB Endowment*, vol. 15, no. 9, pp. 1779–1797, 2022.

[86] S. Wright, “Systems of mating. i. the biometric relations between parent and offspring,” *Genetics*, vol. 6, no. 2, p. 111, 1921.

[87] T. Haavelmo, “The statistical implications of a system of simultaneous equations,” *Econometrica, Journal of the Econometric Society*, pp. 1–12, 1943.

[88] O. D. Duncan, *Introduction to structural equation models*. Elsevier, 2014.

[89] J. Pearl, “Causal diagrams for empirical research,” *Biometrika*, vol. 82, no. 4, pp. 669–688, 1995.

[90] ——, “Theoretical impediments to machine learning with seven sparks from the causal revolution,” *arXiv preprint arXiv:1801.04016*, 2018.

[91] A. Abadie, A. Diamond, and J. Hainmueller, “Synthetic control methods for comparative case studies: Estimating the effect of california’s tobacco control program,” *Journal of the American statistical Association*, vol. 105, no. 490, pp. 493–505, 2010.

[92] J. D. Angrist and J.-S. Pischke, “The credibility revolution in empirical economics: How better research design is taking the con out of econometrics,” *Journal of economic perspectives*, vol. 24, no. 2, pp. 3–30, 2010.

[93] M. A. Hernán and J. M. Robins, “Using big data to emulate a target trial when a randomized trial is not available,” *American journal of epidemiology*, vol. 183, no. 8, pp. 758–764, 2016.

[94] T. J. VanderWeele and P. Ding, “Sensitivity analysis in observational research: introducing the e-value,” *Annals of internal medicine*, vol. 167, no. 4, pp. 268–274, 2017.

[95] G. Imbens, “Instrumental variables: an econometrician’s perspective,” National Bureau of Economic Research, Tech. Rep., 2014.

[96] C. F. Manski, “Identification problems in the social sciences,” *Sociological Methodology*, pp. 1–56, 1993.

[97] S. Shimizu, P. O. Hoyer, A. Hyvärinen, A. Kerminen, and M. Jordan, “A linear non-gaussian acyclic model for causal discovery.” *Journal of Machine Learning Research*, vol. 7, no. 10, 2006.

[98] P. Hoyer, D. Janzing, J. M. Mooij, J. Peters, and B. Schölkopf, “Nonlinear causal discovery with additive noise models,” *Advances in neural information processing systems*, vol. 21, 2008.

[99] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio, “Toward causal representation learning,” *Proceedings of the IEEE*, vol. 109, no. 5, pp. 612–634, 2021.

[100] H. Mao, H. Liu, J. X. Dou, and P. V. Benos, “Towards cross-modal causal structure and representation learning,” in *Machine Learning for Health*. PMLR, 2022, pp. 120–140.

[101] K. Zhang, Z. Wang, J. Zhang, and B. Schölkopf, “On estimation of functional causal models: general results and application to the post-nonlinear causal model,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 7, no. 2, pp. 1–22, 2015.

[102] Z. B. Celik, G. Tan, and P. D. McDaniel, “Iotguard: Dynamic enforcement of security and safety policy in commodity iot.” in *NDSS*, 2019.

[103] M. Surbatovich, J. Aljuraidan, L. Bauer, A. Das, and L. Jia, “Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of ifttt recipes,” in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 1501–1510.

[104] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, and P. McDaniel, “Iotsan: Fortifying the safety of iot systems,” in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, 2018, pp. 191–203.

[105] W. He, M. Golla, R. Padhi, J. Ofek, M. Dürmuth, E. Fernandes, and B. Ur, “Rethinking access control and authentication for the home internet of things (iot),” in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 255–272.

[106] K. Kapitanova, E. Hoque, J. A. Stankovic, K. Whitehouse, and S. H. Son, “Being smart about failures: assessing repairs in smart homes,” in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, 2012, pp. 51–60.