

# Federated IoT Interaction Vulnerability Analysis

Guangjing Wang<sup>†</sup>, Hanqing Guo<sup>†</sup>, Anran Li<sup>\*</sup>, Xiaorui Liu<sup>‡</sup>, Qiben Yan<sup>†</sup>

<sup>†</sup>Michigan State University, <sup>\*</sup>Nanyang Technological University, <sup>‡</sup>North Carolina State University  
wanggu22@msu.edu, guohanqi@msu.edu, anran.li@ntu.edu.sg, xiaorui.liu@ncsu.edu, qyan@msu.edu

**Abstract**—IoT devices provide users with great convenience in smart homes. However, the interdependent behaviors across devices may yield unexpected interactions. To analyze the potential IoT interaction vulnerabilities, in this paper, we propose a federated and explicable IoT interaction data management system FexIoT. To address the lack of information in the closed-source platforms, FexIoT captures causality information by fusing multi-domain data, including the descriptions of apps and real-time event logs, into interaction graphs. The interaction graph representation is encoded by graph neural networks (GNNs). To collaboratively train the GNN model without sharing the raw data, we design a layer-wise clustering-based federated GNN framework for learning intrinsic clustering relationships among GNN model weights, which copes with the statistical heterogeneity and the concept drift problem of graph data. In addition, we propose the Monte Carlo beam search with the SHAP method to search and measure the risk of subgraphs, in order to explain the potential vulnerability causes. We evaluate our prototype on datasets collected from five IoT automation platforms. The results show that FexIoT achieves more than 90% average accuracy for interaction vulnerability detection, outperforming the existing methods. Moreover, FexIoT offers an explainable result for the detected vulnerabilities.

**Index Terms**—Federated graph learning, model explainability, vulnerability analysis, IoT, smart home

## I. INTRODUCTION

A growing number of smart devices are deployed in modern homes to achieve home automation. The devices are controlled by rules that follow the trigger-action paradigm. For example, a SmartThings [1] app has the automation rule (R1) “*If smoke is detected (trigger), turn on the water valve and start alarm beeping (action)*”. According to a recent survey [2], 82.4% of smart homes have multiple rules for controlling a single device. These rules allow devices to interact with each other, referred to as IoT interaction.

However, unexpected IoT interactions could lead to vulnerabilities such as action conflict and action revert, which result in severe security and privacy risks. For instance, suppose a user has set up the above SmartThings rule R1. Unexpectedly, as soon as the water valve was turned on and a water leak was detected in the kitchen, the following SmartThings rule R2 “*Close the water valve when a water leak is detected*” would close the water valve. In consequence, the two rules compose a vulnerable interaction, which exposes the vulnerability “*the water valve fails to turn on when smoke is detected*” because of the action conflict “*water valve opening and closing*”. Such vulnerable interactions can be caused by user errors [3]–[5] and physical attacks [6]–[9].

To avoid interaction vulnerabilities, it is crucial to manage the IoT automation rules and track the trigger-action infor-

mation flow. Naturally, the IoT automation rules follow the general trigger-action paradigm, and the interactions among rules can compose an interaction graph. Automation rules can be represented by nodes, and the edges are “trigger-action” connections among different rules. A naive idea is to detect the sensitive event based on predefined security policies, and then trace it back to find root causes. One may consider querying a graph database that stores interaction graphs.

However, the query or search-based methods have limited coverage of the security policies or vulnerability patterns. Most methods [10]–[14] pre-define security policies and vulnerability patterns within a single platform [2]. For example, ProVThings [11] requires users to define and input policies describing sequences of causal interactions with a graph database backend. Yet, the predefined policies can hardly cover potential new vulnerabilities across heterogeneous platforms. As a result, it may yield significant false positive and false negative errors. Moreover, federated query-based methods [15]–[18] are not applicable. Even though common security policies could be extracted and shared among multiple houses, device interactions in different houses do not intersect. Each interaction graph is a complete data sample. Therefore, the query outcome for vulnerability detection in a single house cannot necessarily be computed over the union of source databases of multiple houses.

Another approach profiles the behaviors of systems by analyzing event logs [19]–[21]. However, they cannot fully expose the interaction correlations between different events. Their limitations are three-fold. **First**, considering the complex causal dependencies among multiple automation rules, it is hard to accurately mine the cross-app interaction logic from event log sequences, which are within individual devices [13]. **Second**, sharing smart home usage data with a third party might cause privacy leakage issues such as the leakage of living habits and routines, while a dataset from a single house is insufficient for training a model with high generalization ability. **Third**, the mining results are difficult to interpret. When a threat alarm is generated, the users can hardly learn the exact segment of device interactions that caused the issue.


To address these problems and limitations, we propose **FexIoT**, a **Federated** and **explicable** GNN-based approach for automatic **IoT** interaction vulnerability analysis. For an interaction graph, the node features can be represented by semantic-aware word or sentence embeddings. Compared with benign graphs, vulnerable interaction graphs could have different graph patterns that express abnormal behaviors. FexIoT utilizes the GNN model to learn the interaction patterns.

Meanwhile, as the event logs belong to different platforms and households, by leveraging federated learning, we can collaboratively develop a more generalized model while retaining users’ data locally. Moreover, the interaction graph allows to search and extract a special subgraph that contains execution flow paths related to abnormal behaviors. Based on the extracted flow path, we can explain the model’s predictions and identify the causes of vulnerable interactions. There are three key technical challenges for IoT interaction analysis across multiple closed-source platforms.

(i) *How to extract and represent real-time interaction causality information from heterogeneous closed-source platforms?* Existing data mining methods take event logs as input [19]–[21]. They ignore semantic information such as automation logic and device relations. Moreover, most event logs are coarse-grained data as they only contain a timestamp, object, and attribute status. As a result, it is hard to extract fine-grained trigger-action logic. Given that the app descriptions contain essential trigger-action causality information, we propose to fuse multi-domain sequence information into interaction graphs using natural language processing (NLP) techniques. With the fused IoT interaction graphs, we can apply GNN models for interaction analysis in a unified format.

(ii) *How to build the vulnerability detection model with high heterogeneity in datasets while avoiding sharing raw data?* One household has a limited number of devices, which makes it infeasible to train a custom and robust GNN model. Federated learning (FL) allows for collaboratively training a model without exposing raw data. However, the data heterogeneity leads to poor modeling performance with slow convergence and low accuracy in FL training paradigms [22]–[24], especially when the data distribution is time-varying or not well balanced among different clients. To overcome this challenge, we design a layer-wise clustering-based federated graph contrastive learning framework. We train a shared vulnerability detection model by differentiating normal and vulnerable graphs over *non-i.i.d.* datasets with high generalization ability without sharing users’ data.

(iii) *How to adapt to new vulnerability patterns and automatically identify potential causes of vulnerable interactions?* The interactions in the IoT world are complex due to the different types of devices and threats. Even though we apply FL to train on various graph data from different home sources, we acknowledge that there could be drifting samples that evolve from existing vulnerabilities or are novel kinds of vulnerabilities. We propose to filter out drifting samples based on federated contrastive graph representations and the corresponding median absolute deviation. Then, we design an efficient cause analysis method by exploring different subgraphs of an interaction graph with Monte Carlo beam search (MCBS) [25], which is a heuristic game tree search algorithm. Based on the potential correlation features of IoT devices, we propose to use the optimal SHAP values [26] to measure the risk of subgraphs of a complete interaction graph. Thus, we can trace the most possible information flow chain relevant to the vulnerable interactions.

<b>Rule R1 from SmartThings App</b>	
Turn lights on if motion is detected	2022-05-08 17:08:05 - INFO: Motion in living room is active >
<b>Rule R2 from Alexa Skill</b>	2022-05-08 17:08:45 - INFO: Lamp 4 in living room is on >
Lock front door when living room lights are on	2022-05-08 17:08:50 - INFO: Front door is locked >
<b>Rule R3 from Home Automation</b>	2022-05-08 20:48:10 - INFO: Smoking alarm starts beeping >
Turn on water valve and unlock doors if smoke is detected	2022-05-08 20:48:25 - INFO: Water valve is turned on >
<b>Rule R4 from IFTTT Applet</b>	2022-05-08 20:49:15 - INFO: Front door is unlocked >
Turn off water valve when water leak is detected	2022-05-08 20:50:15 - INFO: Water leak is detected in Kitchen >
	2022-05-08 20:50:45 - INFO: Close water valve in kitchen >

(a) Rules in a smart home. (b) An event log file example.

Fig. 1: The smart home rules and event logs.

In summary, we make the following contributions:

- We propose a cross-modality data fusion method, which fuses sequence data from multiple sources into graph data to capture the causality of IoT interaction information.
- We design a layer-wise clustering-based federated contrastive GNN model for *non-i.i.d.* graph datasets to learn contrastive graph representations, and it achieves more than 90% average accuracy in vulnerability detection.
- We propose an efficient cause analysis method based on Monte Carlo beam search and the SHAP value to evaluate the risk of subgraphs, which automatically provides explanations for any complex vulnerable interactions.

## II. PRELIMINARIES

We first introduce the IoT interaction graph, and then formally define the problem of interaction vulnerability analysis.

**Definition 1 (Interaction Graph).** Let  $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{X}\}$  represent an interaction graph, which consists of a set of nodes  $\mathcal{V}$ , edges  $\mathcal{E}$  and node features  $\mathcal{X}$ . Each node  $v \in \mathcal{V}$  is an automation rule from an IoT automation control app (e.g. a SmartThings app). For example, the rule R1 in Figure 1a is represented as a node. Each edge  $e \in \mathcal{E}$  represents the correlation between two rules. For instance, rules R1 and R2 in Figure 1a compose the “trigger-action” correlation edge, which means R1 triggers the execution of R2. Each node  $v \in \mathcal{V}$  is associated with feature information  $\mathcal{X}_v$ , which is a word or sentence embedding of a rule. Each graph  $\mathcal{G}$  will have a graph label  $y$ , which denotes whether the interaction graph is vulnerable or not.

We define two types of interaction graphs corresponding to static analysis and dynamic analysis. The first is the offline interaction graph, which is constructed purely from app rule descriptions for static analysis. The advantage of rule descriptions is that it provides “trigger-action” interaction logic, and the possible interactions among rules can be modeled by the offline interaction graph. But rule descriptions cannot reveal the actual device status under customized settings. For example, it cannot show the specific device involved (e.g., the location of lights in Figure 1a R1 and the device execution time). Moreover, rule descriptions cannot represent the real-time device status such as whether the water valve is on or off at a specific time. The exact device status affects vulnerability detection results. Furthermore, many platforms such as Home Assistant [27] allow users to customize automation rules, so different houses might have different rules.

Therefore, for dynamic analysis with heterogeneous closed-source platforms such as IFTTT [28], Google Assistant [29], and Amazon Alexa [30] during runtime, we define the online interaction graph, which is constructed from event logs and app descriptions. Figure 1b shows an example of event logs. One benefit is that it records time, the specific device, and the device status. However, such event logs are coarse-grained and lack the key “trigger-action” logic information. The status changes of a device can be triggered by different possible events. For example, it can be hard to determine which previous event triggers “Front door is locked” in Figure 1b. Therefore, to derive the actual trigger-action interaction graph, we need to identify the “trigger-action” information from the deployed rules. This motivates us to fuse app rule descriptions and event logs. To summarize, the main difference between online and offline graphs is that online graphs reflect real-time device interaction information.

**Definition 2 (Interaction Vulnerability).** Interaction vulnerability refers to the vulnerability coming from interactions between devices, and the environment. Specifically, we label the graph as vulnerable if it satisfies at least one of the 6 types of vulnerabilities identified by existing work [31]: *condition bypass, condition block, action revert, action loop, action conflict, and action duplicate*. Internal graph vulnerability refers to the vulnerabilities inherently from interaction graphs, while external graph vulnerability refers to the vulnerabilities caused by external attacks. Instead of enumerating all possible interaction vulnerabilities, we design a federated GNN model to learn the vulnerability patterns. Given the interaction graph  $G$ , our task is to learn the graph embedding  $Z^t$  at each time  $t$ , and  $Z^t$  is used for vulnerable interaction prediction. Considering a graph can contain multiple types of vulnerabilities, multi-class classification is not suitable. Thus, the prediction problem is formulated as a binary classification problem  $f(Z^t) \rightarrow y^t$  that maps the interaction graph embedding  $Z^t$  to the binary label  $y^t$  of interaction incident at a given time  $t$ . Finally, we identify a subgraph  $G_{sub}$  with the highest risk score by exploring various subgraphs with the SHAP-based Monte Carlo beam search, which explains the prediction result  $y^t$ .

### III. SYSTEM DESIGN

We design FexIoT for IoT interaction analysis as shown in Figure 2. A client represents a house where data is collected and a GNN model is trained using federated learning. Each client can run FexIoT on devices such as a Raspberry Pi or NVIDIA Jetson Nano. Each client implements three main components: data fusion from event logs and apps’ descriptions, vulnerable interaction detection with federated GNN, and vulnerability explanation with the Monte-Carlo beam search-based method. A server can perform the clustering and aggregation in FexIoT, which could be served by a security solution provider.

#### A. Cross-modality Data Fusion for Graph Construction

We first propose a cross-modality data fusion method, which fuses sequence data from apps’ descriptions and event

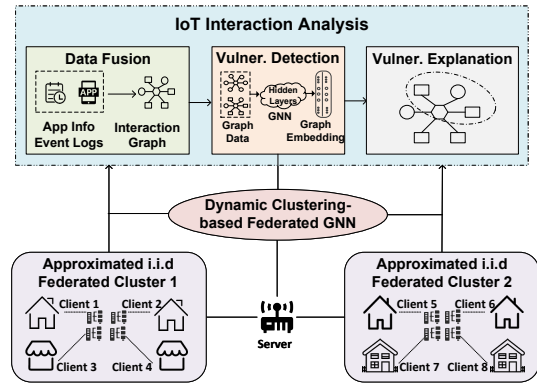


Fig. 2: System architecture of FexIoT.

logs into graphs. Based on NLP techniques, we propose to extract multi-grained semantic features. Creatively, we design classifiers to aid in identifying the “action-trigger” connections among devices and the environment in rules. Finally, we chain different connections with event logs information to construct the interaction graphs in heterogeneous platforms.

1) *Processing App Description:* We aim to obtain “trigger-action” causal logic information from app rule descriptions. With existing tools such as SpaCy [32], we can extract linguistic elements. Specifically, we apply part-of-speech (POS) tagging to categorize words in correspondence with a particular POS. For example, for an automation rule “Close the water valve if a water leak is detected”, the root verb *close* can be recognized as the main task. We are interested in direct objects, nominal subjects and clausal complements to extract the device objects, properties, and the main task. Additionally, we eliminate the named entity because the same entity might modify two distinct objects.

We then compute the rule correlation features based on the above linguistic components as follows. (i) We compute similarity features, which include the verb elements similarity and object elements similarity. But a pair of sentences can have a different number of verb or object elements. In such a case, we calculate the dynamic time warping distance [33] of word embeddings as the similarity score. (ii) We calculate the causal relation features. Specifically, we check whether two sentences have synonym, hypernym, meronym, or holonym relations. We use one-hot vectors to represent these causal relation features. (iii) We calculate sentence-level feature embeddings by using sentence encoder [34] to express word sequences in sentence-level embedding space. The trigger-action pair embedding  $A$  is computed as follows:

$$A = \frac{1}{m} \sum A_{w_i} + \frac{1}{n} \sum A_{w_j}, \quad (1)$$

where  $m$  and  $n$  are the numbers of words in the trigger and action sentences, respectively.  $A_{w_i}$  is the embedding of word  $w_i$ , and  $A_{w_j}$  is the embedding of word  $w_j$ .

2) *Processing Event Log:* We obtain real-time device status information from event logs. We utilize the event logs generated by the apps, which contain timestamps, devices, and

their status. We keep the key information while removing noise from event logs. There are different types of noise in raw event logs. For example, device execution failures can cause execution errors to be recorded. The periodic reporting of sensor readings also brings the noise in event logs. Thus, we remove the repetitive readings and execution errors that do not change the device status. Moreover, some sensor readings are numerical values, while the corresponding app functions are logical values such as “low” or “high”. For example, “*humidity in the living room is 32*” is recorded in event logs when the SmartThings app description contains the “*humidity is low*” trigger. The Jenks natural breaks algorithm [35] is applied to convert numerical values into logical ones. Thus, we clean the log data, which is used for interaction graph construction.

3) *Interaction Graph Construction*: We first build a complete offline interaction graph from all rules deployed in a home. In the first phase, we construct “action-trigger” correlation (the action of R1 triggers the execution of R2) pairs among different rules, which is called interaction correlation discovery. If two rule sentences have an “action-trigger” correlation, we label it as true, otherwise, false. We use the manually labeled IFTTT interaction flow dataset from [31] and our new crawled IFTTT and SmartThings interaction flow dataset as the training dataset. We extract features described in Section III-A1, and train a binary classification model. Finally, we apply the well-trained model to classify the unlabeled rule sentence pairs. In the second phase, we randomly choose and chain the “trigger-action” and “action-trigger” pairs into interaction graphs. This makes the generated interaction graph samples more representative and less prone to bias.

However, offline interaction graphs cannot reflect the exact interactions in real-time device interactions. Moreover, the devices have different types of “trigger-action” interactions, which can compose different interaction graphs. Thus, we need to match the current device status in event logs to the possible interaction graphs. To achieve that, we extract the device name, status, and event time from the event logs. With the “trigger-action” logic in existing interaction graphs, we can match devices and their status with explicit connections in interaction graphs. Meanwhile, the timestamp assists in determining the sequence of events. Thus, we integrate event logs and app descriptions into a fine-grained real-time interaction graph.

### B. Clustering-based Federated Contrastive Graph Learning

After the graph construction, we propose a fine-grained clustering-based federated contrastive graph representation learning model considering data heterogeneity and drifting samples in FL. The learned graph representations can be used for vulnerability detection. In our designed FL paradigm, each client reserves two models. The first is the graph representation learning model, which participates in the FL process in Algorithm 1. The second is a linear classification model such as an SGDClassifier, which locally learns to classify the learned graph representation to detect vulnerable or normal graphs.

1) *Contrastive Graph Learning Loss Function*: In the graph representation learning model, we design the contrastive learn-

---

### Algorithm 1: Dynamic clustering-based federated GNN

---

**Input:** Graph dataset  $G_c$  of each client  $c$ , GNN model weight  $W_c$ , GNN layer  $l < L$ , client cluster  $C$ , global update round  $T$ , thresholds  $\epsilon_1, \epsilon_2$

```

1 for  $t < T$  do
2   for  $c \in C$  do
3      $W_c \leftarrow$  local training process
4      $W_c =$  RecursiveClusteringAgg( $l, C$ )
5   end
6 end
7 Procedure RecursiveClusteringAgg( $l, C$ ):
8   if  $l > L$  then
9     Return;
10  end
11  Receive  $l$ -th layer’s weights  $W_{c_i}^l$  from each client  $c_i$ ;
12  if  $\epsilon_1 > \|\sum_{i \in [n]} \frac{|G_{c_i}|}{|G|} \Delta W_{c_i}^l\|$  &&
    $\epsilon_2 < \max(\|\Delta W_{c_i}^l\|)$  then
13     $M_{i,j} \leftarrow$  CosineSimilarity( $W_{c_i}^l, W_{c_j}^l$ ) for  $i, j \in C$ 
14     $cluster_1, cluster_2 \leftarrow$  BinaryClustering( $M_{i,j}$ )
15     $W_{cluster_1}^l \leftarrow$  FedAvg( $W_c^l$ ) for each  $c \in cluster_1$ 
16     $W_{cluster_2}^l \leftarrow$  FedAvg( $W_c^l$ ) for each  $c \in cluster_2$ 
17  end
18  else
19     $W_C^l \leftarrow$  FedAvg( $W_c^l$ ) for each  $c \in C$ 
20  end
21  Send  $W_{cluster}^l$  back to each client  $c$ 
22  RecursiveClusteringAgg( $l + 1, cluster_1$ )
23  RecursiveClusteringAgg( $l + 1, cluster_2$ )
24 End Procedure

```

---

ing [36] loss function in Eq. (2) for learning a distance function to measure the dissimilarity of samples:

$$L_c = d_{ij}^2(1 - y_{ij}) + \max(0, k - d_{ij}^2)y_{ij}, \quad (2)$$

where  $d_{ij}$  is the Euclidean distance between two graph embeddings,  $y_{ij}$  is 1 if graph  $G_i$  and  $G_j$  are from different classes, and  $y_{ij}$  is 0 if graph  $G_i$  and  $G_j$  are from the same class. We set a threshold  $k$  to restrict the unusual distance contribution of graphs from different classes. Note that we apply existing well-developed GNN models [37]–[39] in our federated learning framework. Finally, the learned representations are used for training a linear classification model to classify normal or vulnerable graphs in each client.

2) *Dynamic Clustering-based Federated Learning*: We design a fine-grained dynamic clustering-based federated learning algorithm, which is based on the following observations.

There could be domain shifts when the knowledge transfers across *non-i.i.d.* datasets in FL framework [40]. The heterogeneity is two-fold. **First**, the interaction graph can be either homogeneous or heterogeneous. Different households deploy heterogeneous devices and different users have their own usage habitats, which can cause graph heterogeneity. The heterogeneous graph data will cause negative transfer among users and decrease the model performance. **Second**, the graph dataset is unbalanced and *non-i.i.d.*. The *non-i.i.d.* graph datasets from different households will cause biased stochastic gradients, which will impede the convergence of FL models.

Another key observation is that despite the heterogeneity, there exist similar data distributions in smart homes because different users can have common automation rules. We assume that there exist several clusters of households, where the graph datasets from each cluster satisfy the *i.i.d.* property following [41]. There will be some graph datasets that share common feature information, which could be grouped into several clusters. As a result, a new challenge ensues on how to effectively aggregate clients into different clusters.

Consider the clustering-based FL setting with a central server and a set of  $n$  clients  $\{c_1, c_2, \dots, c_n\}$ , which can be dynamically clustered into different clusters  $\{cluster_1, cluster_2, \dots\}$ . Each client  $c_i$  has a set of interaction graphs  $G = \{G_1, G_2, \dots\}$ , and conducts the graph classification  $y = h_k^*(G_i)$ , where  $h_k^*$  is the optimal graph classification model for cluster set  $cluster_k$ . The graph feature information can be reflected by the model parameters and their gradients [41]. Existing federated graph classification over *non-i.i.d.* graphs [42] is coarse-grained since it only considers the similarity of parameters of a whole model. However, from the bottom up, the degree of similarity among deep models decreases [43]–[45]. Therefore, to learn the fine-grained clustering structure, we design the bottom-up layer-wise dynamic clustering algorithm to obtain the similarity among weights of clients as shown in Algorithm 1.

Specifically, suppose  $n$  is the number of clients in FL training, each client  $c$  performs local GNN training (lines 2-4), which follows the traditional FL training paradigm. The server receives  $n$  local models  $W_c^l$  (line 12). The dynamic clustering on the server starts from the bottom layer  $l_1$ . We define two thresholds  $\epsilon_1$  and  $\epsilon_2$  to determine the conditions of clustering of different clients (line 13):

$$\begin{aligned} \epsilon_1 &> \left\| \sum_{i \in [n]} \frac{|G_{c_i}|}{|G|} \Delta W_{c_i} \right\|, \\ \epsilon_2 &< \max(\|\Delta W_{c_i}\|), \end{aligned} \quad (3)$$

where  $|G_i|$  represents the number of graphs owned by client  $i$ ,  $|G|$  is the total number of graphs owned by all clients, and  $\Delta W_{c_i}$  is the local update of model weights of client  $c_i$ .  $\epsilon_1$  measures the degree of fluctuation of FL training, and it bounds the relatively stationary point of the global model before initiating the clustering. Meanwhile, if there are large norms of weight update that are greater than  $\epsilon_2$ , it means high heterogeneity occurs among different clients, and the clustering starts to avoid performance degradation among clients. The two thresholds can be determined via initial experiments on the validation sets [41]. If the conditions in Eq. (3) are satisfied, the server further divides the clients from the same cluster into two sub-clusters and performs model aggregation within each sub-cluster (lines 14-17). This process continues to the next layer recursively. Thus, with more layers of client models being clustered and aggregated, each cluster of clients has reduced divergence, which achieves model converging in fewer training rounds.

3) *Drifting Interaction Pattern Detection*: Due to dynamic changes in smart device settings and the presence of different attacks, the testing interaction graph distribution may diverge from that of the training dataset in actual deployment scenarios. The performance of the detection may be impacted by the drifting samples, which are new interaction vulnerability patterns that are different from already known vulnerabilities. Considering the high false-positive or false-negative rate caused by drifting samples, we design the interaction graph drifting pattern analysis method based on the above federated graph representation learning and median absolute deviation (MAD) [46] in the model application stage.

**First**, in each client, we use the well-trained model in FL to map all the training graph data into latent space. We can calculate the centroid of each class by computing the mean value for each dimension of the latent embedding. Similarly, for the incoming graph data, we also generate latent embeddings. In this way, we can compute the Euclidean distance  $d_i^k$  between the centroids of each class in the training dataset and the test sample  $x^k$ . **Second**, based on MAD, we can estimate the graph data distribution within each class  $i$  by computing  $MAD_i$  in the training dataset, which is the median of the absolute deviation from the median  $\bar{d}_i$  of distance  $d_i^j$ . The  $d_i^j$  is the distance between the latent embedding of each training sample to its centroid in class  $i$ . **Third**, for each testing sample we check if  $d_i^k$  is large enough to make  $x^k$  an outlier of class  $i$ . Specifically, we denote the “normal” label as 0 and the “vulnerable” label as 1. Then  $A_0^k = \frac{|d_i^k - \bar{d}_i|}{MAD_0}$ ,  $A_1^k = \frac{|d_i^k - \bar{d}_i|}{MAD_1}$  and  $A^k = \min(A_0^k, A_1^k)$ . If  $A^k$  is greater than a threshold  $T_M$  that is set as 3 empirically following existing practices [46], then  $x_t^k$  is a potential drifting sample. The MAD method allows each class to decide outliers based on its in-class distribution. In the model application stage, given a set of testing interaction graphs  $\{G\}$ , we first check each graph  $G_i$  to see if it is a drifting sample. If a new sample has a larger distance from all existing classes, then it is a potential drifting sample. In this way, we can filter out and manually inspect drifting samples that are outside of the training space.

### C. Vulnerable Interaction Analysis

Given a detected vulnerable graph, we design the SHAP-based Monte Carlo beam search (MCBS) algorithm to discover the causes of vulnerable interactions considering the IoT dependency relationship. We first formalize the vulnerability explanation problem.  $G$  represents the interaction graph that is examined by a GNN model followed by a linear classifier  $h(\cdot)$  in the model application stage, and the prediction result is  $y$ . The vulnerable interactions are caused by a connected subgraph  $G_{sub}$  in an interaction graph. We further define a cooperative game following [47] to measure the contributions of different parts (players) of a graph. Suppose  $G_{sub}$  is one player, where  $G_{sub}$  contains nodes  $\{v_1, v_2, \dots, v_s\}$ . Other nodes in  $G \setminus G_{sub}$  are other players  $\{\{v_{s+1}\}, \dots, \{v_m\}\}$ . Our goal is to find the most possible subgraph  $G_{sub}$  that is responsible for the prediction  $y$ .

---

**Algorithm 2:** SHAP based Monte Carlo Beam Search

---

**Input:** GNN model  $h(\cdot)$ , interaction graph  $G$ , MCBS iteration number  $I$ , kernel SHAP samples  $K$ , the least node number  $N_{min}$ , the root of search tree  $N_0$

**Output:** subgraph  $G_{sub}$

```
1 for  $i \leftarrow I$  do
2    $S_i = N_0$ ,  $curPath = \{N_0\}$ 
3   while  $|h(S_i)| > N_{min}$  do
4     for subgraph  $G_i$  in  $B_{level}(S_i)$  do
5       for  $k < K$  do
6          $g(z') = h(S_i)$ 
7          $h(T_x^{-1}(z')) = h(S_i \cup G_i)$ 
8       end
9        $Score(h(\cdot, G, G_i)) = W$ 
10    end
11    Select  $N_{next}$  following Eq. 7
12     $S_i = N_{next}$ 
13     $curPath = curPath \cup N_{next}$ 
14  end
15   $S_l = S_i \cup S_i$ 
16 end
17 return subgraph  $G_{sub}$  with the highest score from  $S_l$ 
```

---

However, the device’s abnormal behaviors are hardly noticeable until certain consequences occur. Moreover, an app can trigger unexpected events that are subscribed by another device, or different apps could introduce contradictory changes to a device attribute. Directly using the prediction scores to measure the risk of subgraphs is problematic, since it cannot capture connections among different graph structures. Suppose there is a subgraph  $G_{sub}$  that contains the exact interaction vulnerability chain, any subgraph  $G_{con}$  containing  $G_{sub}$  will also cause the vulnerable interaction. Meanwhile, a subgraph  $G_{inc}$  that is included in  $G_{sub}$  may also trigger the same threat, but  $G_{inc}$  can only reveal part of the causes instead of the complete causal route. Another method SubgraphX [48] applies the Shapley value to measure the risk of subgraphs, where they assume the players are independent. However, in an interaction graph, the existence and actions of nodes (players) are not independent because the deployment of devices is related to each other. Therefore, the Shapley value could neglect the dependent relations among different nodes.

We propose to calculate the SHapley Additive exPlanations [26] (SHAP) value to evaluate the risk score of subgraphs and apply MCBS to improve the efficiency of the search process as shown in Algorithm 2. In the MCBS tree, following the notation in SubgraphX [48], the root node  $N_0$  is the input graph  $G$ , and each node  $N_i$  in the search tree is a connected subgraph  $G_{sub}$ , the edge in the search tree is the pruning action  $a$ . A subgraph  $G_j$  is acquired by the action  $a_j$  from  $G_i$ , which is represented by  $(N_i, a_j)$ . We combine beam search with Monte Carlo tree search to store a set of best playouts, which is called *beam* (lines 2-4 in Algorithm 2). The size of a beam  $B_{level}$  is fixed for each level, which shows that  $B_{level}$  best nodes are kept at each level. From the model perspective, the target node feature is aggregated from a limited number of neighbor nodes. From the smart home perspective, devices

are directly related to limited neighboring devices. Therefore, it is reasonable that only neighboring nodes in a beam  $B_{level}$  are employed for information aggregation.

The SHAP approach can rate the relevance of each feature for a specific prediction. SHAP values are the Shapley values of a conditional expectation function [26], which can better measure the feature importance. Here, we use  $\{G_1, \dots, G_i, \dots, G_n\}$  to represent  $n$  subgraphs of an interaction graph  $G$ . The analysis result can be written as:

$$G^* = \arg \max_{|G_i| \leq N_{min}} \text{SHAP}(h(\cdot), G, G_i), \quad (4)$$

where  $N_{min}$  is a hyperparameter that limits the number of nodes in a subgraph. Following SHAP framework, the SHAP value  $\phi_i$  can be computed as follows:

$$\phi_i(h, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [h_x(z') - h_x(z' \setminus i)], \quad (5)$$

where  $x'$  is the set of total non-zero entries,  $z'$  is a subset of  $x'$ ,  $|z'|$  is the number of non-zero entries in  $z'$ ,  $M$  is the number of set of all entries,  $z' \setminus i$  denotes setting  $z'_i = 0$ . In our proposed Algorithm 2, non-zero entries mean the nodes (players) of a graph that are included in the Monte Carlo beam search process.  $h_x(z') = E[h(z)|z_S]$  means it calculates the expectation across all possible node combinations, where  $S$  is the non-zero indexes in  $z'$ . Solving Eq. (5) is challenging. Thus, we apply the kernel SHAP to approximate it (lines 5-9 in Algorithm 2).

$$L(h, g) = \sum_{z' \in Z} [h(T_z^{-1}(z')) - g(z')]^2 \cdot C, \quad (6)$$
$$C = \frac{M - 1}{\binom{M}{|z'|} |z'| (M - |z'|)}$$

where we assume  $g(z') = Wz'$  is the explanation model following a linear form.  $g(z')$  matches the original model  $h(z)$  when  $z = T_z(z')$ .  $T_z(\cdot)$  is a mapping function that converts the inputs to the original input space. We use the weighted linear regression to solve the equation since  $L(h, g)$  is a squared loss and  $g(z')$  is the linear function. The intuition is that Eq. (5) is calculating the difference of means, which has a connection with linear regression, and the mean value is the best least-squares point for a set of data points. Thus, noted by [26], we calculate  $\phi_i(h, x) = w_j(x_j - E(x_j))$  given  $h(x) = \sum_{j=1}^M w_j x_j + b$ , which is a linear classification model such as SGDClassifier as introduced in Section III-B.

During the MCBS process, the node  $N_{next}$  selection criterion is computed as follows (lines 11-13 in Algorithm 2):

$$\arg \max_{a_j} Q(N_{next}, a_j) + \lambda R(N_{next}, a_j), \quad (7)$$

where  $Q(N_{next}, a_j)$  is the average reward score over several visits,  $\lambda$  is the hyperparameter that balances the exploration and exploitation.  $R(N_{next}, a_j)$  is the immediate reward for choosing  $N_{next}$ , which is the SHAP( $h(\cdot), G, G_i$ ) score. Finally, Algorithm 2 will output the subgraph  $G_{sub}$  with the highest risk score for cause analysis.

TABLE I: Statistics of interaction graphs.

Type	Label	Total Graph Num.	Vulnerable Graph Num.
Homo. (IFTTT)	labeled	6,000	1,473
	unlabeled	10,000	*
Hetero. (5 Platforms)	labeled	12,758	3,828
	unlabeled	19,440	*

#### IV. EVALUATION

##### A. Experimental Setting and Dataset

To simulate the FL training, we evaluate the dynamic clustering-based federated GNN on a high-performance computing cluster, which is equipped with Intel(R) Xeon(R) Gold 6148 2.4GHz CPUs running on CentOS 7. For collecting app description data, we use Scrapy [49] to crawl smart home app rules from the following 5 platforms:

- For SmartThings [1], we crawl rule descriptions from 185 open-source apps.
- For Home Assistant [27], we crawl rule descriptions from 574 blueprints.
- For IFTTT [28], we integrate 315,393 applets from [31] and newly crawled 1,535 applets.
- For Google Assistant [29], we crawl 480 services and 4,812 action commands.
- For Amazon Alexa [30], we crawl 2,232 services and 3,274 skill commands.

For collecting event log data, a volunteer deploys the off-the-shelf smart devices in a house and collects event logs for a whole week. We remove and replace potentially private information, e.g., we modify a rule from “*Alexa, turn on HEATER\_NAME*” to “*Alexa, turn on heater*”. Besides the inherent vulnerabilities in randomly generated graphs, we also consider the external graph vulnerabilities. Thus, following HAWatcher [13], we simulate 5 types of attacks to generate external graph vulnerabilities by modifying event logs: fake events, fake commands, stealthy commands, command failure, and event losses. We match the SmartThings “trigger-action” pairs in our constructed heterogeneous interaction graphs. We construct 600 online interaction graphs, of which 300 are vulnerable interaction graphs.

Eventually, we build an IFTTT graph dataset with 6,000 labeled graphs and a heterogeneous graph dataset with 12,758 labeled graphs from the five platforms as shown in Table I. The number of nodes in each graph is from 2 to 50. We apply DGL [50] to implement graph construction, and use text embeddings to represent node features in interaction graphs. Specifically, we use spaCy [32] *en\_core\_web\_lg* model to transform extracted key phrases into word embeddings, and each embedding has 300 dimensions. App descriptions are verbose, and encoding key phrases can better model interaction logic in descriptions. We then use the Universal Sentence Encoder [34] to transform rules from Google Assistant and Amazon Alexa skills into sentence embeddings, and each embedding has 512 dimensions. As voice assistant commands

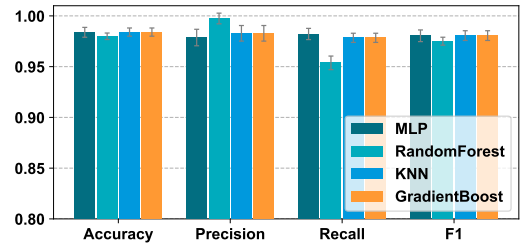


Fig. 3: Different classifiers for correlation analysis.

are concise, directly using the sentence encoder can better model the meaning of word sequences. Therefore, the nodes of graphs for different platforms have different feature spaces, which jointly compose the heterogeneous graphs.

There are two labeling phases during the dataset construction, which include “action-trigger” correlation labeling and “graph vulnerability” labeling. The “action-trigger” correlation labeling is introduced in Section III-A3 and will be evaluated in Section IV-B. Here we detail the “internal graph vulnerability” labeling process, where we check whether the interaction vulnerability inherently exists in an interaction graph. Two volunteers who are studying cyber-physical and IoT security first summarize the security procedures from the literature such as [11]–[14]. Then, they search for six types of vulnerabilities [31] as mentioned in Definition 2. They manually label a graph as vulnerable if any vulnerability is found and then cross-check the labeling results.

##### B. Interaction Correlation Discovery Evaluation

There could exist an exponential number of correlations based on the number of rules. To efficiently remove the unrealistic correlations and save manual efforts, we use classification models to discover whether there is a “action-trigger” correlation between two rules. Specifically, we manually label 5,600 “action-trigger” interaction pairs and 8,000 unrelated pairs. Then, we extract interaction pair features as described in Section III-A.

We evaluate four classifiers implemented with Scikit-learn [51]. Multi-layer Perceptron (MLP), RandomForest, K-nearest neighbors (KNN), and GradientBoost classifiers are trained on the same extracted features. We obtain the best hyperparameters with the grid search method. The results in Figure 3 are reported according to the 10-fold cross-validation. The MLP achieves the best recall rate of 99.8%, while KNN achieves the best precision rate of 99.7%. The RandomForest achieves the best average accuracy of 98.4% and an F1 value of 98%. MLP can better define the feature space with more layers. The RandomForest model avoids overfitting by using a random subspace technique. In summary, all four models achieve excellent performance, which proves the effectiveness of our extracted features. Considering the highest scores in precision, recall, and F1 value, we apply the MLP, KNN, and RandomForest to predict the remaining unlabeled sentence pair correlations. If the three models have different predictions, we inspect and discuss the labeling results to ensure the

accuracy of the labeling. Finally, we chain the IoT automation rule pairs into interaction graphs.

### C. Clustering-based Federated GNN Evaluation

After graph construction, FexIoT proceeds to conduct fine-grained clustering-based federated contrastive graph representation learning for vulnerability detection. Note that as it is hard to build large-scale smart homes in real life, following the template of deployed smart homes, we leverage the offline graphs to produce a set of online interaction graphs for federated GNN evaluation. The only difference is that online interaction graphs reflect real-time IoT interactions.

We compare FexIoT with two GNN models, four baselines, and five different Data distributions. The two homogeneous GNN models are: (i) GCN [37] is a convolutional network operating on graphs, and we adopt three graph convolutional layers. (ii) GIN [38] is a graph isomorphism network model, and we adopt the original model architecture. For the heterogeneous graph classification model, we choose MAGNN model [39], which learns heterogeneous graph embeddings based on metapath aggregation.

The four federated learning framework baselines are: (i) Federated multi-task learning (FMTL) [41], which groups the clients into clusters by using geometric aspects of the loss surface. (ii) Graph clustered federated learning (GCFL+) [42], which is a gradient sequence-based clustering method for graph classification. (iii) Federated averaging (FedAvg) [52], which aggregates locally-computed updates in federated learning. (iv) Self-training in clients (Client), which trains the GNN locally without any communications.

**Data Distribution Configuration.** We study the impacts of different data distributions on the federated GNN models. We split the IFTTT graph dataset according to Dirichlet distribution to simulate the *non-i.i.d.* dataset in each client, and we set the number of clients as 10. We draw class marginal distribution from Dirichlet distribution with the probability density function  $p(x) \propto \prod_{i=1}^k x_i^{\alpha_i-1}$ .  $\alpha$  is the positive concentration parameter, which we set to 0.5, 1, 2, 5, and 10. When  $\alpha$  is close to 0, most of the generated values drawn from the Dirichlet distribution will be close to 0. Thus, we can simulate clients with unbalanced and *non-i.i.d.* datasets.

Following the above Dirichlet distribution, different clients will have different numbers of graphs. We split 80% of the IFTTT dataset as the client training dataset and 20% of the dataset as the testing dataset for each trial. During the training process, we empirically set the two thresholds  $\epsilon_1$  and  $\epsilon_2$  in Eq. (3), which are related to the size of model weights, to start and end the layer-wise clustering process. We found that the local clients achieve decent performance when the two values are set between 0.5 and 2. The value setting also depends on the initialization settings such as the learning rate with the corresponding optimizer. We set the learning rate as 0.001 in the Adam optimizer,  $\epsilon_1$  is 1.2 and  $\epsilon_2$  is 0.8 if they are not specified in the following experiments. We apply the weighted cross-entropy loss function to handle class imbalance for interaction graph classification. We set up the weight given

to each class in the cross-entropy loss according to the inverse ratio to class frequencies. Figure 4 shows the average accuracy, precision, recall, and F1 value of models.

**GNN Models Evaluation.** We test the influence of different GNN models in the federated learning process. Note that the GNN models are used for graph representation learning in the FL process. We use SGDCClassifier from Scikit-learn [51] to implement the normal or vulnerable binary classification. As shown in Figure 4, the GIN model achieves better performance than the GCN model, which is consistent with the previous study [38]. Moreover, we found that when the dataset is more evenly distributed ( $\alpha$  is larger), both the GCN and GIN achieve better performance. However, the data distribution indeed has a great impact on the model performance. For example, the F1 values of the GIN model in FedAvg are 0.735 and 0.748 when  $\alpha$  is 0.1 and 10, respectively. The GIN achieves better performance (average F1 is 0.92) when  $\alpha$  is 10 than the performance when  $\alpha$  is 0.1 (average F1 is 0.89). Our proposed dynamic clustering-based federated graph learning method achieves the best performance with different GNN models.

**Efficacy Evaluation.** Compared to baselines, our FexIoT achieves the best performance because of the design consideration of fine-grained homogeneous data features in heterogeneous data distribution. For example, for FexIoT, the accuracy is 0.891 and 0.919 when  $\alpha$  is 0.1 and 10, respectively. For GCFL+, the accuracy is 0.852 and 0.889 when  $\alpha$  is 0.1 and 10, respectively. The accuracy of FedAvg is 0.717 and 0.768 when  $\alpha$  is 0.1 and 10, respectively. The average client accuracy is 0.542 and 0.622 when  $\alpha$  is 0.1 and 10, respectively. FexIoT improves the accuracy by 17.4% compared with the FedAvg. FedAvg is deeply affected by the data heterogeneity, which leads to poor modeling performance with low accuracy in FL training paradigms. By contrast, FexIoT can mostly reduce the data heterogeneity effect among different datasets by training on clusters with high homogeneity. We also compare the performance of federated learning compared with centralized training. We test the GIN model in centralized training, which collects all data from clients to train a GIN model. The centralized training achieves 0.944 F1 value, while our federated GIN model achieves 0.91 average F1 value over five data distributions. Overall, our proposed FexIoT can better depict the homogeneity of the GNN models learned from different data distributions than the coarse-grained ones.

**Stability Evaluation.** The clustering-based methods (FexIoT, GCFL+, and FMTL) can achieve better and more stable performance than FedAvg and single-client training. For example, with our proposed algorithm, the average accuracy of GIN under five different data distributions is 0.907, and the standard deviation (STD) is 0.01. For FedAvg, the average accuracy is 0.738, and the STD is 0.017. The performance of FedAvg is more divergent according to the data distribution of each client. Similarly, the performance of self-training in each client is also affected by different data distributions. Besides, our FexIoT outperforms the state-of-the-art method GCFL+, which has an average accuracy of 0.87 and STD is 0.012. Facing the heterogeneity in datasets, the GCFL+ considers the similarity

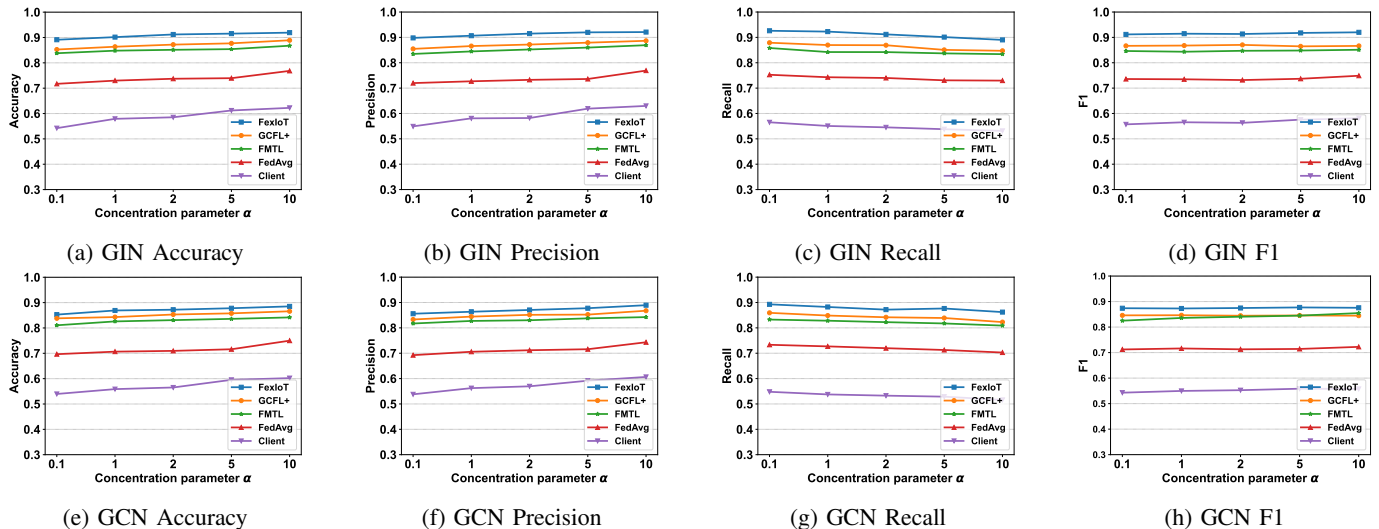


Fig. 4: The performance of two GNN models under five different Dirichlet distribution parameters  $\alpha$ .

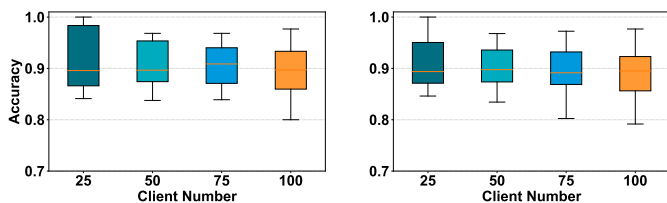


Fig. 5: The test accuracy of client models with different numbers of clients participating in training with the IFTTT dataset (left) and heterogeneous graph dataset (right).

of parameters of a whole model, which can be fluctuating due to the changes in parameters of different layers of client models. By contrast, our FexIoT implements clustering based on fine-grained bottom-up layer-wise parameters, which is less affected by fluctuating layers. Overall, our FexIoT is more stable under different data distributions than other FL methods.

**Scalability Evaluation.** We simulate different numbers of clients (25, 50, 75, 100) joining the FexIoT on two datasets. We use GIN on the IFTTT dataset and MAGNN on the heterogeneous graph dataset if not mentioned. The concentration parameter  $\alpha$  of the Dirichlet distribution is 1. Figure 5 is a box plot, in which minimum, maximum, median, first quartile, and third quartile accuracy are reported. On the IFTTT dataset, the third quartile accuracies of 25, 50, 75, 100 clients are 0.869, 0.879, 0.882, and 0.873, respectively. Therefore, even though the number of clients increases, 75% of clients can achieve more than 86% accuracy, which shows the high scalability of FexIoT. When the number of clients is larger (e.g., 100), the results show more divergence as the minimum accuracy is 0.8, and the maximum accuracy is 0.977. When the number of clients increases, due to the fixed size of the entire dataset, the size of the dataset for each client decreases, which affects the evaluation results. Nevertheless, the results show that FexIoT can effectively aggregate clients that have similar data distribution to boost the model performance.

TABLE II: Comparison of different systems with testbed data.

Method	Accuracy	Precision	Recall	F1
HAWatcher [13]	0.82	0.83	0.87	0.85
DeepLog [20]	0.74	0.78	0.79	0.78
IsolationForest [53]	0.63	0.74	0.61	0.67
FexIoT	0.9	0.9	0.93	0.91

**Drifting Interaction Pattern Evaluation.** The methods that rely on known patterns will become fruitless when there are drifting samples. Therefore, we need to identify the possible drifting graph samples with unknown interaction patterns. We evaluate drifting interaction pattern detection on the unlabeled dataset, to check if the randomly generated graph dataset contains drifting samples. We show the clustering results in Figure 6 on 1500 randomly sampled graph feature representations learning with federated contrastive graph learning. The data is processed with TSNE dimension reduction. The centroid of each class is the white cross, and the possible drifting samples are in the red circle. The six types of known interaction vulnerabilities are clustered in gray dotted boxes. Through our federated contrastive learning, the model can learn well the features of six types of vulnerability patterns and the normal graph pattern, which are separable in hidden space shown in Figure 6. Based on the drifting interaction pattern analysis in Section III-C, we found 63 and 104 potential drifting samples in the IFTTT dataset and heterogeneous graph dataset, respectively. After manually checking, we found three new types of vulnerability patterns: (i) Automation action is reverted over time. (ii) Another action can generate fake automation conditions. (iii) Non-automation settings can block the existing actions of smart devices. In this way, FexIoT can reduce the false-alarming rate while the security rule-based querying methods in the traditional database cannot discover new patterns and even incur high false-positive or false-negative rates because of the drifting samples.

**System Comparison.** We compare FexIoT (trained with

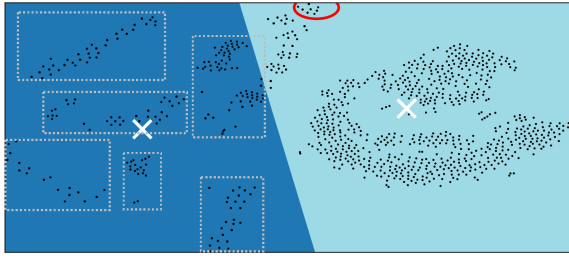


Fig. 6: K-means clustering on 1500 randomly sampled graph representations with TSNE dimension reduction.

50 clients) with existing vulnerability detection systems: (i) HAWatcher [13] is a search-based system that extracts secure rule interaction templates and evaluates device interaction in real-time. We use the extracted templates to discover vulnerabilities in the test dataset. (ii) DeepLog [20] models event logs as a language sequence and uses Long Short-Term Memory (LSTM) to learn log patterns. We train LSTM on normal logs and find anomalies that deviate from the training data. (iii) IsolationForest [53] is a density-based anomaly detection method with a tree structure, which is implemented by Scikit-learn [51], and the input is a data vector that includes device status. We use 600 online interaction graphs in Section IV-A that are integrated with event logs and descriptions as the test dataset. The system performance is shown in Table II. From the results, we find that all methods fail to differentiate between normal user interruptions (e.g. manually turning on a light) and malicious attacks, which cause false positives. Feedback from users should be provided to update such correlations and reduce false alarms. FexIoT outperforms the existing methods due to the following reasons: (i) HAWatcher only extracts binary rule templates, which can hardly cover long-term complex correlations such as the correlation between an air conditioner and temperature events, and (ii) DeepLog and IsolationForest cannot effectively mine interaction correlations across different events.

**Communication Cost Evaluation.** We measure the communication cost during the training process by computing the total data transferred (download and upload) between the server and clients. As shown in Figure 7, the total transferred data is less than 40 GB in 60 rounds with 100 clients, which is acceptable for wired network bandwidth. FexIoT saves 40.2% communication overhead compared with FedAvg [52]. FedAvg needs to aggregate the whole model during the FL process, and FMTL [41] and GCFL+ [42] also share the whole model but within different clusters. The low cost of FexIoT is attributed to our proposed layer-wise clustering-based FL method. At the initial stage, only the parameters of the first layer are uploaded to a server for clustering. Then, based on the previous clustering result, the upper layer parameters of models are uploaded to the server, and the server sends parameters of different layers back to the corresponding client clusters. Clients in the same cluster share more layers than clients that are in different clusters. Thus, it reduces the number of parameters transmitted between the server and clients.

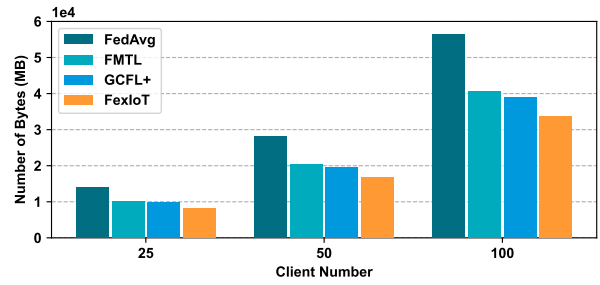


Fig. 7: Communication cost with different numbers of clients.

#### D. Vulnerable Interaction Cause Analysis Evaluation

After vulnerability detection, we implement qualitative and quantitative evaluations to show the efficacy and efficiency of our vulnerable interaction explanation method. We randomly choose 100 interaction graphs that contain vulnerable interactions, which are reported by the GCN model on the unlabeled IFTTT dataset. Among the 100 graphs, we identify four graphs that are benign after manually checking, which are false-positive results from the GCN model.

We compare our method FexIoT with two baselines of graph explanation methods: SubgraphX [48] and MCTS\_GNN. SubgraphX is a general graph explanation method that applies Monte Carlo tree search with shapely value to explore various subgraphs, while MCTS\_GNN applies Monte Carlo tree search with the prediction score of the GNN model to explore subgraphs. Other graph explanation methods such as PGExplainer [54] and GNNExplainer [55] can only identify discrete edges or nodes, which cannot be used for the interaction vulnerability explanation among nodes. The SubgraphX and MCTS\_GNN are implemented by the DIG [56] library, and we apply GCN as the vulnerability detection model and perform the vulnerability analysis for both right and wrong predictions.

**Example Illustration.** To visually demonstrate the superiority of our explanation method, we give two intuitive examples in Figure 8. The first row illustrates the example of the false positive of the GCN model prediction, while the second row illustrates the correct prediction. For the first row in Figure 8, the interaction graph is predicted to contain vulnerable interactions, but it is deemed benign after manual inspection. The interaction graph describes that when the door opens, the water flow will run with the switch turned on with a notification sent to users. The user can notify the app to turn on the camera. Then, if the smoke is detected, the door unlocks and the exhaust ventilation fan starts. This interaction graph does not contain interaction vulnerabilities, as shown in Figure 8, our FexIoT offers a minor misleading explanation subgraph, as it only identifies the concise subgraph to explain the prediction. By contrast, other methods try to find a larger subgraph to explain the prediction result, which causes more confusion for the inspector.

For the second example in Figure 8, the GCN model prediction is correct. It describes that when the user taps to turn off the camera, the camera will be turned off and the action will be recorded in a Google spreadsheet. When a new

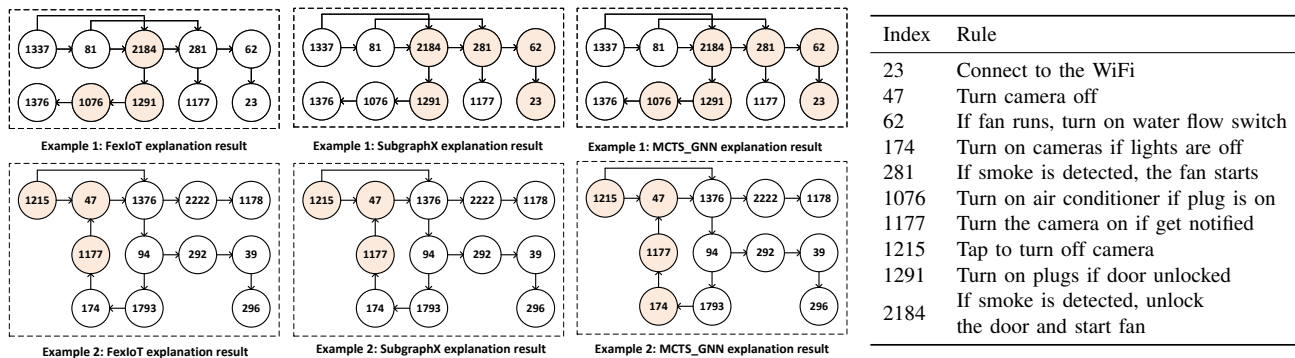


Fig. 8: Vulnerability explanation results comparison and the text description for indexes.

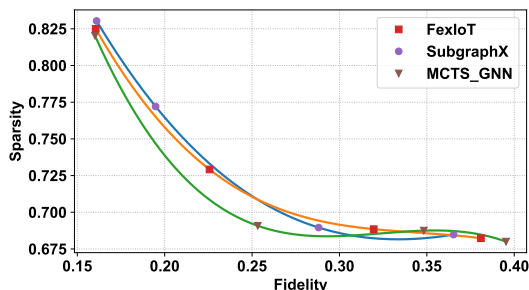


Fig. 9: The comparison of Fidelity and Sparsity scores.

spreadsheet subscriber is added, it will send a notification to turn on the camera. FexIoT, SubgraphX, and MCTS\_GNN all can recognize the key subgraph that causes the interaction vulnerability that turns off the camera within a loop. **Overall**, for both cases, our FexIoT can provide a reasonable vulnerability explanation with better visual explanations.

**Fidelity and Sparsity.** Besides the visualization to evaluate the vulnerability explanation methods, we use the *Fidelity* and *Sparsity* [57] to quantitatively measure the explanation results. The Fidelity metric refers to the difference between two prediction scores before and after removing a part structure of a graph. The Sparsity metric refers to the percentage of the structure that is recognized as important by the explanation model. There is a trade-off between Fidelity and Sparsity. A high Sparsity score means a smaller structure, which tends to be low Fidelity (less important). For real-life cases that lack the ground truth, the Fidelity and Sparsity scores can help measure the explanation quality for the security inspector.

We test the Fidelity and Sparsity scores of 50 randomly-picked interaction graphs. The scores depend on the structure of interaction graphs, which fluctuate among different cases. Half of the tested cases have relatively high Fidelity which is greater than 0.3, and low Sparsity which is smaller than 0.7. For better visualization, we plot the curve of Sparsity with respect to Fidelity for four cases as shown in Figure 9. The four cases have higher Fidelity and lower Sparsity, or lower Fidelity and higher Sparsity. Our method FexIoT can strike a decent balance between Sparsity and Fidelity scores, which means FexIoT can accurately identify the concise subgraphs that are the possible causes of the vulnerabilities. **Overall**,

TABLE III: Runtime efficiency with different datasets.

	Graph Construction Time (s)	Prediction Time (s)	Vulnerability Analysis Time (s)	Model Size (MB)
IFTTT	17.19	0.52	2.18	5.48
Hetero.	976.99	0.61	3.64	6.13

the results indicate that our explanation method can find the important yet concise subgraph for prediction explanation, which is more faithful for interaction analysis.

**Vulnerability Explanation Efficiency.** As shown in Table III, the 12,758 heterogeneous graphs generation takes 980s. The size of the MAGNN model for heterogeneous graphs is only 6.13MB, which makes it feasible to train the model on devices such as a Raspberry Pi. The prediction time is 0.61s on average, which is related to the graph size. For vulnerability explanation, the average cause discovery time is 3.64s for an interaction graph. It takes 182s for 50 graphs with 18 nodes on average. Note that the time is related to the algorithm parameters such as the number of MCTS iterations, and the kernel SHAP samples. **Overall**, the FexIoT achieves vulnerability prediction and explanation with high efficiency.

## V. RELATED WORK

**Code-based Vulnerability Analysis.** IoT interaction vulnerability has drawn much attention in recent years. Table IV summarizes the main features of FexIoT in comparison with five interaction vulnerability analysis systems. Many IoT interaction vulnerability analysis systems [10]–[13] rely on the source code analysis. For example, Celik *et al.* [10] presents SainT, which performs static analysis to identify sensitive data flows. ProvThings [11] and IoTGuard [12] apply code instrumentation to generate fine-grained event logs for IoT auditing. However, the existing studies neglect the closed-source nature of most platforms, which limits the application of code analysis-based methods. Besides, some approaches [14] are based on dynamic testing, which can hardly cover all testing cases and cause security issues during real-life testing. Furthermore, most existing interaction analysis systems assume all rules run on a single platform, which is not necessarily true. 82.4% smart home deployments have multiple automation rules to control a device, and 62.4% users deployed more than one platform according to an online survey [2]. Thus, these methods that rely on source code analysis cannot analyze the

TABLE IV: Comparing to existing IoT interaction vulnerability analysis systems

System	Source Code	Dynamic Test	Long-term Correlation	Cross-platform Interaction	Method
IoTGuard [12]	Yes	Yes	No	No	Code instrumentation
HAWatcher [13]	Yes	No	No	No	Code analysis & Log analysis
IoTSafe [14]	Yes	Yes	No	No	Static analysis & Dynamic testing
iRuler [31]	No	No	Yes	No	SMT solver & Model checker
IoTMon [58]	Yes	No	Yes	No	Code analysis & Text analysis
<b>FexIoT (Ours)</b>	<b>No</b>	<b>No</b>	<b>Yes</b>	<b>Yes</b>	<b>NLP &amp; Federated GNN models</b>

interactions across multiple closed-source platforms. Finally, FexIoT considers the long-term correlations by encoding time information in the node features, so the GNN model can mine the temporal correlation among different rules.

**Learning-based Vulnerability Analysis.** Many researchers adopt machine learning methods [19]–[21] to detect vulnerabilities in various systems. For instance, Zhou *et al.* [21] propose to use LSTM neural networks to learn the historical correlation of log patterns and then detect anomalies. Yang *et al.* [59] propose to add knowledge on historical anomalies via probabilistic label estimation. But existing methods can hardly mine the complex interaction logic of apps [13] from only event log sequences. Moreover, compared with semi-supervised learning or unsupervised learning in anomaly detection, supervised learning can be more accurate because it can optimize performance using labeled data. It is hard to accurately analyze the specific threat causes with unsupervised learning on interaction graphs. With graph representation learned from supervised learning, FexIoT provides a reliable way to automatically pinpoint the possible causes of vulnerable interaction in large and complex graphs.

**Federated Graph Learning.** FL is to collaboratively train a shared prediction model without storing the data centrally. Recent studies [60]–[65] have explored the system and data heterogeneity in FL. FedProx adds weights to aggregate different clients [60], but choosing weights for different applications is challenging. Another method is to share local device data or server-side proxy data [61] to deal with data heterogeneity, but it requires knowledge of local data distributions. Some researchers propose to bound gradients [66] or add additional noise [67] to guarantee convergence, which will increase training time and decrease model accuracy. GNN models [68]–[71] have demonstrated exceptional performance in a variety of tasks including graph or node classification. The GNN model can mine the dependencies among nodes in a graph, and map graph information into a graph embedding. Many recent works [72]–[76] apply the FL on GNN models. For instance, GraphFL [72] is an FL framework based on meta-learning for semi-supervised node classification. However, besides the feature and label heterogeneity, the graph data could contain *non-i.i.d.* structural information, which will degrade the learning performance [42]. By contrast, our proposed layer-wise clustering-based federated graph learning method can mitigate the heterogeneity among different clients.

## VI. DISCUSSION AND FUTURE WORK

**Security and Privacy.** We acknowledge that FexIoT faces security and privacy challenges. Particularly, it might suf-

fer from Sybil attacks [77]–[80] when an attacker controls multiple clients to attack the system. One possible solution is to enhance the software and network security (e.g., using firewalls, CAPTCHAs, or device-specific asymmetric keys) in smart homes to protect the internal FL model. Moreover, there are several defense methods that can identify Sybil attacks based on the diversity of client updates [78], training loss from randomly selected clients [79], and feature importance [80], which could be integrated into FexIoT. To enhance the data privacy, we will add differential privacy [81]–[83] and secure aggregation mechanisms [84]–[86] to FexIoT in the future.

**Data Labeling.** Providing ground truths is necessary for scientific evaluation. Existing works [87]–[89] have proposed methods for efficient data annotation. In this work, we manually labeled the dataset with cross-validation. In the future, we plan to combine experts, amateurs, and ML models for cross-validation to further enhance the quality of labels for IoT data. For example, experts can label a few samples and develop few-shot learning-based models. ML models and amateurs can collaboratively label new data. Iteratively, the newly labeled data could be used to re-train the model.

## VII. CONCLUSION

In this paper, we investigate how IoT rule configuration data could expose interaction vulnerabilities across heterogeneous closed-source platforms. We present and implement a novel IoT data management solution, FexIoT, that leverages federated and explicable graph learning to analyze large-scale IoT interaction data. We propose a cross-modality data fusion method to cope with the information deficiency problem in closed-source platforms. We design the fine-grained dynamic clustering-based federated contrastive graph representation learning algorithm to discover interaction vulnerabilities and tackle the concept drift problem of smart home data. We propose the MCBS-based search method with SHAP value to search and measure the risk of each subgraph. With the data collected from 5 real-world IoT platforms, we demonstrated the superior performance of FexIoT in detecting and explaining the interaction vulnerabilities.

## ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their insightful comments on our work. This work was supported in part through National Science Foundation grants CNS-1950171, and Michigan State University’s Institute for Cyber-Enabled Research Cloud Computing Fellowship, with computational resources and services provided by Information Technology Services and the Office of Research and Innovation at Michigan State University.

## REFERENCES

- [1] SmartThings, “Smarthings developer,” <https://smarthings.developer.samsung.com/docs/api-ref/capabilities.html>, 2022.
- [2] H. Chi, C. Fu, Q. Zeng, and X. Du, “Delay wrecks havoc on your smart home: Delay-based: Automation interference attacks,” in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 1575–1575.
- [3] J. Huang and M. Cakmak, “Supporting mental model accuracy in trigger-action programming,” in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015, pp. 215–225.
- [4] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman, “Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 3227–3231.
- [5] C. Nandi and M. D. Ernst, “Automatic trigger generation for rule-based smart homes,” in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, 2016, pp. 97–102.
- [6] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, “Dolphinattack: Inaudible voice commands,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 103–117.
- [7] Q. Yan, K. Liu, Q. Zhou, H. Guo, and N. Zhang, “Surfingattack: Interactive hidden attack on voice assistants using ultrasonic guided waves,” in *Network and Distributed Systems Security (NDSS) Symposium*, 2020.
- [8] Y. Wang, H. Guo, and Q. Yan, “Ghosttalk: Interactive attack on smartphone voice system through power line,” *arXiv preprint arXiv:2202.02585*, 2022.
- [9] H. Guo, C. Li, L. Li, Z. Cao, Q. Yan, and L. Xiao, “Nec: Speaker selective cancellation via neural enhanced ultrasound shadowing,” in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2022, pp. 355–366.
- [10] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, “Sensitive information tracking in commodity iot,” in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1687–1704.
- [11] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, “Fear and logging in the internet of things,” in *Network and Distributed Systems Symposium*, 2018.
- [12] Z. B. Celik, G. Tan, and P. D. McDaniel, “Iotguard: Dynamic enforcement of security and safety policy in commodity iot.” in *NDSS*, 2019.
- [13] 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.
- [14] W. Ding, H. Hu, and L. Cheng, “Iotsafe: Enforcing safety and security policy with real iot physical interaction discovery,” in *Proceedings of the 2021 Network and Distributed Systems Security (NDSS) Symposium*, 2021.
- [15] J. Kim, H. Ha, B.-G. Chun, S. Yoon, and S. K. Cha, “Collaborative analytics for data silos,” in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 2016, pp. 743–754.
- [16] J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers, “Smcql: Secure query processing for private data networks.” *Proc. VLDB Endow.*, vol. 10, no. 6, pp. 673–684, 2017.
- [17] Y. Tong, X. Pan, Y. Zeng, Y. Shi, C. Xue, Z. Zhou, X. Zhang, L. Chen, Y. Xu, K. Xu *et al.*, “Hu-fu: efficient and secure spatial queries over data federation,” *Proceedings of the VLDB Endowment*, vol. 15, no. 6, p. 1159, 2022.
- [18] F. Han, L. Zhang, H. Feng, W. Liu, and X. Li, “Scape: Scalable collaborative analytics system on private database with malicious security,” in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 1740–1753.
- [19] T. Li, Y. Jiang, C. Zeng, B. Xia, Z. Liu, W. Zhou, X. Zhu, W. Wang, L. Zhang, J. Wu *et al.*, “Flap: An end-to-end event log analysis platform for system management,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1547–1556.
- [20] M. Du, F. Li, G. Zheng, and V. Srikumar, “Deeplog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298.
- [21] P. Zhou, Y. Wang, Z. Li, X. Wang, G. Tyson, and G. Xie, “Logsayer: Log pattern-driven cloud component anomaly diagnosis with machine learning,” in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 2020, pp. 1–10.
- [22] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, “Federated learning with personalization layers,” *arXiv preprint arXiv:1912.00818*, 2019.
- [23] C. T. Dinh, N. Tran, and J. Nguyen, “Personalized federated learning with moreau envelopes,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 394–21 405, 2020.
- [24] Y. Deng, M. M. Kamani, and M. Mahdavi, “Adaptive personalized federated learning,” *arXiv preprint arXiv:2003.13461*, 2020.
- [25] T. Cazenave, “Monte carlo beam search,” *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 68–72, 2012.
- [26] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Proceedings of the 31st international conference on neural information processing systems*, 2017, pp. 4768–4777.
- [27] H. Assistant, “Home assistant,” <https://www.home-assistant.io/>, 2022.
- [28] IFTTT, “ifttt applets,” <https://ifttt.com/explore/>, 2022.
- [29] Google, “Google assistant,” <https://assistant.google.com/>, 2022.
- [30] Amazon, “Amazon alexa skills,” <https://www.amazon.com/alexa-skills/b?ie=UTF8&node=13727921011>, 2022.
- [31] 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.
- [32] spacy, “Industrial-strength natural language processing,” <https://spacy.io/>, 2022.
- [33] T. Prätzlich, J. Driedger, and M. Müller, “Memory-restricted multiscale dynamic time warping,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 569–573.
- [34] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar *et al.*, “Universal sentence encoder,” *arXiv preprint arXiv:1803.11175*, 2018.
- [35] G. F. Jenks, “The data model concept in statistical mapping,” *International yearbook of cartography*, vol. 7, pp. 186–190, 1967.
- [36] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.
- [37] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
- [38] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” *arXiv preprint arXiv:1810.00826*, 2018.
- [39] X. Fu, J. Zhang, Z. Meng, and I. King, “Maggn: Metapath aggregated graph neural network for heterogeneous graph embedding,” in *Proceedings of The Web Conference 2020*, 2020, pp. 2331–2341.
- [40] B. Csaba, X. Qi, A. Chaudhry, P. Dokania, and P. Torr, “Multilevel knowledge transfer for cross-domain object detection,” *arXiv preprint arXiv:2108.00977*, 2021.
- [41] F. Sattler, K.-R. Müller, and W. Samek, “Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints,” *IEEE transactions on neural networks and learning systems*, 2020.
- [42] H. Xie, J. Ma, L. Xiong, and C. Yang, “Federated graph classification over non-iid graphs,” *Advances in Neural Information Processing Systems*, vol. 34, 2021.
- [43] M. Long, Y. Cao, Z. Cao, J. Wang, and M. I. Jordan, “Transferable representation learning with deep adaptation networks,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 12, pp. 3071–3085, 2018.
- [44] L. Mou, Z. Meng, R. Yan, G. Li, Y. Xu, L. Zhang, and Z. Jin, “How transferable are neural networks in nlp applications?” *arXiv preprint arXiv:1603.06111*, 2016.
- [45] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” *Advances in neural information processing systems*, vol. 27, 2014.
- [46] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, “Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median,” *Journal of experimental social psychology*, vol. 49, no. 4, pp. 764–766, 2013.
- [47] H. W. Kuhn and A. W. Tucker, *Contributions to the Theory of Games*. Princeton University Press, 1953, vol. 2.

- [48] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji, "On explainability of graph neural networks via subgraph explorations," *arXiv preprint arXiv:2102.05152*, 2021.
- [49] Scrapy, "Scrapy is a web crawling and web scraping framework," <https://scrapy.org/>, 2022.
- [50] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.
- [51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [52] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.
- [53] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth IEEE international conference on data mining*. IEEE, 2008, pp. 413–422.
- [54] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," *arXiv preprint arXiv:2011.04573*, 2020.
- [55] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," *Advances in neural information processing systems*, vol. 32, p. 9240, 2019.
- [56] M. Liu, Y. Luo, L. Wang, Y. Xie, H. Yuan, S. Gui, H. Yu, Z. Xu, J. Zhang, Y. Liu, K. Yan, H. Liu, C. Fu, B. M. Oztekin, X. Zhang, and S. Ji, "DIG: A turnkey library for diving into graph deep learning research," *Journal of Machine Learning Research*, vol. 22, no. 240, pp. 1–9, 2021. [Online]. Available: <http://jmlr.org/papers/v22/li-0343.html>
- [57] P. E. Pope, S. Kourou, M. Rostami, C. E. Martin, and H. Hoffmann, "Explainability methods for graph convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10772–10781.
- [58] 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.
- [59] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1448–1460.
- [60] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.
- [61] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.
- [62] A. Li, L. Zhang, J. Wang, J. Tan, F. Han, Y. Qin, N. M. Freris, and X.-Y. Li, "Efficient federated-learning model debugging," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 372–383.
- [63] A. Li, L. Zhang, J. Wang, F. Han, and X. Li, "Privacy-preserving efficient federated-learning model debugging," *IEEE Transactions on Parallel and Distributed Systems*, 2021.
- [64] C. Li, X. Zeng, M. Zhang, and Z. Cao, "Pyramidfl: A fine-grained client selection framework for efficient federated learning," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2022.
- [65] J. Wang, L. Zhang, A. Li, X. You, and H. Cheng, "Efficient participant contribution evaluation for horizontal and vertical federated learning," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 911–923.
- [66] H. Yu, S. Yang, and S. Zhu, "Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5693–5700.
- [67] A. Khaled, K. Mishchenko, and P. Richtárik, "Tighter theory for local sgd on identical and heterogeneous data," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 4519–4529.
- [68] M. Li, S. Chen, Y. Zhang, and I. Tsang, "Graph cross networks with vertex infomax pooling," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14093–14105, 2020.
- [69] Q. Huang, H. He, A. Singh, S.-N. Lim, and A. R. Benson, "Combining label propagation and simple models out-performs graph neural networks," *arXiv preprint arXiv:2010.13993*, 2020.
- [70] S. Ivanov and L. Prokhorenkova, "Boost then convolve: Gradient boosting meets graph neural networks," *arXiv preprint arXiv:2101.08543*, 2021.
- [71] J. Zhao, X. Wang, C. Shi, B. Hu, G. Song, and Y. Ye, "Heterogeneous graph structure learning for graph neural networks," in *35th AAAI Conference on Artificial Intelligence (AAAI)*, 2021.
- [72] B. Wang, A. Li, H. Li, and Y. Chen, "Graphfl: A federated learning framework for semi-supervised node classification on graphs," *arXiv preprint arXiv:2012.04187*, 2020.
- [73] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-iid data silos: An experimental study," *arXiv preprint arXiv:2102.02079*, 2021.
- [74] D. Caldarola, M. Mancini, F. Galasso, M. Ciccone, E. Rodolà, and B. Caputo, "Cluster-driven graph federated learning over multiple domains," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2749–2758.
- [75] C. Meng, S. Rambhatla, and Y. Liu, "Cross-node federated graph neural network for spatio-temporal data modeling," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1202–1211.
- [76] H. Peng, H. Li, Y. Song, V. Zheng, and J. Li, "Differentially private federated knowledge graphs embedding," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 1416–1425.
- [77] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.
- [78] C. Fung, C. J. Yoon, and I. Beschastnikh, "The limitations of federated learning in sybil settings," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020, pp. 301–316.
- [79] Y. Jiang, Y. Li, Y. Zhou, and X. Zheng, "Sybil attacks and defense on differential privacy based federated learning," in *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2021, pp. 355–362.
- [80] C. Zhou, Y. Sun, D. Wang, and Q. Gao, "Fed-fi: Federated learning malicious model detection method based on feature importance," *Security and Communication Networks*, vol. 2022, 2022.
- [81] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.
- [82] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.
- [83] W. Liu, J. Cheng, X. Wang, X. Lu, and J. Yin, "Hybrid differential privacy based federated learning for internet of things," *Journal of Systems Architecture*, vol. 124, p. 102418, 2022.
- [84] R. Cramer, I. B. Damgård *et al.*, *Secure multiparty computation*. Cambridge University Press, 2015.
- [85] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.
- [86] Y. Li, Y. Zhou, A. Jolfaei, D. Yu, G. Xu, and X. Zheng, "Privacy-preserving federated learning framework based on chained secure multiparty computing," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6178–6186, 2020.
- [87] A. Drutsa, V. Fedorova, D. Ustalov, O. Megorskaya, E. Zermirnova, and D. Baidakova, "Crowdsourcing practice for efficient data labeling: Aggregation, incremental relabeling, and pricing," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2623–2627.
- [88] M. Yuan, L. Zhang, X.-Y. Li, and H. Xiong, "Comprehensive and efficient data labeling via adaptive model scheduling," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1858–1861.
- [89] K. Li, G. Li, Y. Wang, Y. Huang, Z. Liu, and Z. Wu, "Crowdrl: an end-to-end reinforcement learning framework for data labelling," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 289–300.