ADCL: Towards An Adaptive Network Intrusion Detection System Using Collaborative Learning in IoT Networks

Ma, Zuchao; Liu, Liang; Meng, Weizhi; Luo, Xiapu; Wang, Lisong; Li, Wenjuan

Published in:
IEEE Internet of Things Journal

Link to article, DOI:
10.1109/JIOT.2023.3248259

Publication date:
2023

Document Version
Peer reviewed version

ADCL: Towards An Adaptive Network Intrusion Detection System Using Collaborative Learning in IoT Networks

Zuchao Ma, Liang Liu, Weizhi Meng, Xiapu Luo, Lisong Wang and Wenjuan Li

Abstract—With the widespread of cyber attacks, network intrusion detection system (NIDS) is becoming an important and essential tool to protect Internet of Things (IoT) environments. However, it is well-known that the NIDS performance depends heavily on the effectiveness of detection model, which can be influenced significantly by the learning mechanism and the available training data. Many existing studies try to mitigate the above challenges, but few of them consider the adaptability and the cost of deploying an NIDS, the integrity of learning process, the capacity of model based on concrete traffic samples at the same time. To fill this gap and improve the detection performance, we propose a collaborative learning based detection framework called ADCL, which can mitigate the limitations on the knowledge of a single model by leveraging multiple models trained in similar environments and detecting intrusions in a collaborative manner. Our evaluation results indicate that ADCL can provide better performance compared with a single model on detecting various attacks in IoT networks. Specifically, ADCL improves F-score by up to 80% for adaptability, 42% in mitigating the reliance on learning integrity, 85% for model capacity. Furthermore, the detection results of ADCL guide those single models to update and increase the F-score by 15%.

Index Terms—Intrusion Detection, System Adaptability, Collaborative Learning, Multiple Model, Internet of Things

1 Introduction

Internet of Things (IoT) has already become a popular infrastructure to support many modern applications and services, such as smart homes, smart healthcare, public security, industrial monitoring and environment protection. It allows devices to collect information from surroundings, i.e., control units can gather information from other devices and make better strategies [2].

However, threatening by various network attacks, the security of IoT networks becomes more important than ever. To mitigate it, network intrusion detection system (NIDS) has been widely adopted to secure network assets. Factors considered by NIDS include but are not limited to network authentication, network routing and transferred data, which can be mined from network traffic. Thus, research involving network traffic has become an active topic in the NIDS domain. Over the past few years, with the development of machine learning techniques, various machine learning based schemes are provided to analyse network traffic superiorly by training specific models. Essentially, machine learning based detection relies on learning 'normal' behaviour or 'abnormal' behaviour of network events to generate corresponding models that can guide judging intrusions. Existing schemes can work under either the supervised way or the unsupervised way, for detecting known attacks and previously unseen attacks to a certain extent.

Motivation. Existing works focus on exploring the application of novel models (e.g., deep learning [3], [4]), feature engineering (e.g, [5], [6]), and efficiency optimization (e.g., [7], [8]) of NIDS. While we identify there are three crucial issues remained as below.

- The adaptability and the cost of deploying one NIDS. In most cases, constructing a classification model in a practical environment (network) is the main challenge for deploying NIDSs. This is because *training model requires time* - if the network administrator (user) wants to deploy the NIDS, there are two normal options: 1) using a model trained in advance with external data (e.g., from a lab dataset, manufacturer dataset), which guarantees the NIDS can work upon but could result in poor adaptability, due to the difference between prepared dataset and practical dataset; and 2) costing some time to enable the classification model to learn the practical traffic where it is deployed, which can result in better classification performance but need the learning process to be finished. More specifically, for option 1), the network where the classification model is trained may have a different topology including the number, the type and the position of network devices compared to the network where the classification model is actually deployed. For option 2), the time consumption of learning could be a heavy cost due to the efforts of collecting data, labeling data and training

- *A preliminary version of this paper appears in Proc. of the 23rd Information Security Conference (ISC), pp. 255-273, 2020 [1].*
- *Zuchao Ma is with Nanjing University of Aeronautics and Astronautics (NUAA), China, and the Department of Computing, The Hong Kong Polytechnic University, Hong Kong.*
- *Weizhi Meng is with the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Denmark. E-mail: weme@dtu.dk*
- *Xiapu Luo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong.*
- *Liang Liu and Lisong Wang are with the Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics (NUAA), China.*
- *Wenjuan Li is with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong.*

model, e.g., the time for learning process could be as long as four months [9]. Thus, keeping a balance between the adaptability and the cost of deploying is important.

- The integrity of learning process in the NIDS. The efficiency of machine learning is based on the integrity of the learning process - whether the model has been trained by enough data (samples) from a network, which is usually hard to determine. *There is no ideal model* - different models trained in various environments may have their own limited view as they are cultivated with limited training data. On one hand, learning from different networks can obtain different knowledge coming from dynamic traffic, whereas each model only has limited knowledge; on the other hand, even learning in the similar networks (e.g., with the same topology), the collected datasets could be different caused by various collection mechanisms or analysis techniques, especially in IoT networks where the traffic comes from different devices and activities. For example, if the functions of some devices are not activated during the traffic collection period, then the collection process may miss such traffic samples, leading to a negative impact on the integrity of learning. Therefore, it is important to improve the efficiency of classification model especially when it is not trained with integrated samples.

- The difference in the model capacity. As we know, different models could have diverse performance even if they are trained with the same dataset, because the classification principles and the model parameters are distinct. In more detail, the adaptability of each model towards the dataset could be influenced by the distribution of samples and the training process, e.g., the distribution of training samples has a great impact on the determination about the model performance. Therefore, it makes sense to leverage the strength of different models cultivated in different environments, including different training samples, different training processes and different model types, in order to improve the detection performance of NIDS.

Motivated by the above issues, in this work, our primary goal is to design an adaptive NIDS that can leverage the knowledge from different models and enhance the detection performance. In particular, our work aims to 1) improve the adaptability towards the practical environment as well as reduce the cost; 2) enhance the NIDS performance even when the integrity of learning process cannot be guaranteed; and 3) combine the capacity of multiple models to improve the detection performance.

Contribution. In this work, we advocate that *for each model constructed under a specific environment, it may have the limitation on its knowledge, but this limitation could be mitigated by leveraging multiple models trained in similar environments to detect intrusion collaboratively* [1]. Motivated by this idea, we design ADCL, an adaptive collaborative learning based detection system with the following key features:

1) *Model fingerprint.* When traffic samples are uploaded to train a model, ADCL collects the topology information about the network where these samples are gathered, and generates the fingerprint of the trained model.
2) *Similarity-based Model Selector (SMS).* When ADCL executes traffic detection towards a specific network, it requires Model Repository to provide multiple models trained under similar networks, through selecting those models with similar fingerprints.
3) *Detector with multiple models.* ADCL ensembles multiple models selected from SMS to construct a wise detector, leveraging the model collaboration to improve the detection adaptability and performance.
4) *Model update.* The detection result of ADCL can be used to update the models stored in Model Repository, and optimize the performance of each single model with the knowledge obtained from others automatically.

In the evaluation, we build an environment with different IoT virtual devices based on IoTID20 dataset [56]. We also consider both single-hop centralized and multiple-hop network structures. As compared with similar studies (e.g., autoencoder based detection), our results indicate that ADCL can provide better performance, i.e., by improving F-score by up to 80% for adaptability, up to 42% in mitigating reliance, up to 85% in improving capacity and up to 15% in model update. To the best of our knowledge, this is an early study with the aim of building a similarity-based collaborative detection in the current literature.

Organization. The remaining parts are organized as follows. Section 2 introduces related work on machine learning-based NIDS, including supervised learning, unsupervised learning, ensemble learning, and its use in IoT environments. Section 3 gives an overview of ADCL framework and its design goals. Section 4 describes ADCL in detail, including the design, machine learning mechanism, network topology, model repository, similarity-based model selector, and model update. Section 5 discusses the experimental settings and evaluation results. Finally, Section 6 concludes our work.

2 Related Work

2.1 Supervised Learning

Supervised learning, one of the most common learning mechanisms, needs to collect labelled data (indicating the traffic is malicious or benign explicitly) in advance to help train its model. While with the dynamic change and the evolution of attacks overtime, it is not always feasible to collect real attacks as training data in the practical networks. Thus, the trained model cannot guarantee its performance when identifying real attacks in a concrete environment. However, supervised learning is still the most popular solution in intrusion detection, such as Support Vector Machine (SVM), Gaussion Naive Bayes (NB), Linear Discriminant Analysis (LDA), Logistic Regression, Decision Tree, Random Forest and so on.

Gu *et al.* [10] proposed an IDS framework based on SVM ensemble with feature augmentation, in which they transformed the original data to enable SVM obtaining better and robust performance. They also used the naive Bayes feature embedding to optimize the performance of SVM in traffic detection [11]. Hadem *et al.* [12] introduced a Software Defined Networking (SDN) based IDS that uses SVM along with Selective Logging for IP Traceback, which achieved a high accuracy on the NSL-KDD dataset. Bhosale [13] presented Modified Naive Bayes Intrusion Detection System (MNBIDS) based on the existing Naive Bayes, which

could improve the system accuracy and execution performance on the KDD dataset. He *et al.* [14] adopted Naive Bayes classifier to reduce the false positive rate of NIDS towards Integrated Electronic Systems (IES) and achieve a lightweight detection system with 95.84 percent accuracy. Yuan *et al.* [15] proposed a Two-Layer Multi-class Detection (TLMD) method, which combined the C5.0 method and the Naive Bayes algorithm for adaptive network attack detection. It could improve the detection rate as well as the false alarm rate. Elkhadir *et al.* [16] proposed a new robust Median NN-LDA based on Linear Discriminant Analysis to improve the detection performance of NIDS. Mahmudul Hasan *et al.* studied multiple machine learning mechanisms to execute the anomaly detection in IoT sensors of IoT sites, where decision tree and random forest could achieve 99.4% accuracy [17]. Zhang *et al.* [18] proposed an IDS framework based on a distributed random forest capable of handling high-speed traffic data. Besides, it could provide better efficiency and accuracy compared to existing systems. The observation was also verified in a similar study [1].

2.2 Unsupervised learning

Compared to supervised learning, unsupervised learning can provide the following advantages: 1) mitigating the reliance when preparing a labelled dataset as unsupervised learning can cultivate its model by learning the normal traffic even without attacks included; and 2) the learning process does not need to consider concrete attacks while the trained model has the capability of detecting unknown attacks even not being collected by the used datasets. In this case, unsupervised learning can enhance the adaptability and the universality of detection model towards practical attacks in practice.

Recently, AutoEncoder has been a rising unsupervised detection model in NIDS domain. Vu *et al.* [19] developed a Multi-distributed Variational AutoEncoder (MVAE) for the network intrusion detection, aiming to make the traffic more distinguishable, where MVAE introduced the label information of data samples to force/partition network data samples into different classes with different regions in the latent feature space. Li *et al.* [20] proposed an effective deep learning method, namely AE-IDS (AutoEncoder Intrusion Detection System) based on random forest algorithm, aiming to improve the classification accuracy and reduce the training time. Luo *et al.* [9] used AutoEncoder to detect malicious events of wireless sensor networks (WSN) as the light AutoEncoder has a very low computational load and could not be deployed in an IoT network. Mirsky *et al.* [21] then adopted the ensemble of AutoEncoders for online network intrusion detection, which achieved a stream-speed detection under IoT networks.

2.3 Ensemble learning

Each single model may have 'preference' on the specific scenario, i.e., the performance of a model is good on classifying some specific samples but is weak on others. Therefore, ensemble learning is proposed aiming to use multiple models to finish a classification task, while the classification result depends on the different outputs of individual models. In fact, the idea of ensemble learning is similar to the core idea of our ADCL, but there is an essential difference. That is, ensemble learning focuses on training process by cultivating different models with various parameters in the training process, so that different models would obtain different 'preference' and adjust the corresponding weight based on their prediction scores. However, our ADCL considers the model training process as a black box (where users choose mechanisms themselves and finish the training process), and does not have the authority to record or modify the dataset from users, which is appropriate to keep the user privacy and maintain the scalability (i.e., can leverage any models coming from users).

Moustafa *et al.* [22] developed an AdaBoost ensemble learning method by using three machine learning techniques, namely decision tree, Naive Bayes (NB), and artificial neural network, to detect attacks in IoT networks, which could provide a higher detection rate and a lower false positive rate compared with some state-of-the-art techniques. Zhou *et al.* [23] proposed an intrusion detection framework based on the feature selection and ensemble learning techniques called CFS-BA, which could achieve better performance than some similar approaches. Zhong *et al.* [24] then proposed an anomaly detection framework based on the organic integration of multiple deep learning techniques.

2.4 Intrusion Detection in IoT

In the literature, emerging intrusion detection schemes under IoT networks focus mostly on the following topics.

Deploying NIDS in an IoT network. Traditional anomaly detection systems cannot be deployed well in IoT networks due to the limited resources of IoT devices. Based on the deployment structure, current anomaly detection systems can be classified into two types [25]: central deployment (detection is deployed in central points, e.g., base station) and distributed deployment (detection is performed in end-devices). The former can support powerful anomaly detection but it usually cannot percept the communications among end-devices. Thus, some attacks on end-devices might be under estimated. On the other hand, the latter cannot execute powerful anomaly detection on end-devices and this may degrade the detection efficiency.

Therefore, many works explore improving the training/detection efficiency of NIDS to satisfy the practical deployment. Roy *et al.* [7] leverage sampling and dimensionality reduction to extract the most important features, which helps to detect intrusions using less training time. Hu *et al.* [8] proposed several channel and spatial attention mechanisms to accelerate learning traffic features. Shahid *et al.* [26] introduced a lightweight dense random neural network working in IoT node. Zhao *et al.* [27] used a PCA algorithm and compression structure to achieve effective feature extraction with low computational cost. Wu *et al.* [28] proposed a fuzzy rough set-based algorithm to perform feature selection. Dao *et al.* [29] removed a set of ineffective neurons in the AutoEncoder based on their importance to reduce training parameters. TONTA [30] used a statistical light-weight Trend Change Detection (TCD) method to remove the cost caused by training. Yilmaz *et al.* [31] used transfer learning to reduce the training cost. Eric *et al.* also

proposed two light detection models [32]. In this work, the detection of ADCL can be deployed to a central server that has enough capacity to execute the detection task.

Improving the detection performance. Some studies focus on improving the performance of NIDS by optimizing the features adopted in machine learning. For example, Alghanam *et al.* [5] proposed a PIO algorithm to select features, and Li *et al.* [6] incorporated a full Bayesian possibilistic clustering module for feature processing, while Liu *et al.* [33] used empirical mode decomposition and PCA to extract features. Similarly, Kumar *et al.* [34] introduced an HGS-ROA algorithm for detection improvement.

Other works may focus on optimizing their detection strategies, such as Graph neural network [35], [36], deep learning [3], [4] and few-shot learning [37]. Lai *et al.* [38] considered the time correlation and spatial correlation of data for detection. Telikani *et al.* optimized the loss function of models [39]. Chen *et al.* optimized the parameters of neural network with multi-objective evolutionary [40]. Concerning the dynamic change of attacks, Omar [41] leveraged online learning to update model, and Krishnan *et al.* proposed a manufacturer usage description (MUD) to update the monitor of NIDS in real time [42]. Since federated learning trains model with more data and does not leak training parameters, it has been adopted gradually in NIDS [43], [44]. Zhao *et al.* [45] enhanced the performance of federated learning with knowledge distillation. Aouedi *et al.* [46] applied AutoEncoder to help label samples in federated learning. Lian *et al.* [47] decentralized federated learning to avoid single point of failure of NIDS. Federated learning and transfer learning are combined in detection [48]. Some other works also consider a more advanced attacker, such as adversarial attacks [49].

More relevant work regarding intrusion detection in IoT can refer to recent studies [25], [50], [51], [52], [53]. Compared to the literature, we select a different and novel direction, that is, enabling multiple models trained in similar environments to work collaboratively.

3 Framework Overview

Key idea and efficiency proof. The core idea of designing ADCL is based on the observation: *for each model constructed under a specific environment, it will have the limitation on its knowledge, but this limitation could be mitigated by leveraging multiple models trained in a similar environment to detect intrusion collaboratively* [1].

We have proved that for each single model m_0 adopted by NIDS, if there is another model m_x whose *sample predict range* is wider than m_0, then the detection performance can be enhanced by leveraging the m_x to detect attacks with m_0 collaboratively. Here, *sample predict range* means that the set size contains all samples that can be classified correctly by corresponding model in the sample space of m_0. Sample space of m_0 contains all traffic samples that could be generated in the network n_0 where m_0 is trained. Note that usually NIDS will adopt m_0 to detect attacks in n_0.

Theorem 1. *For a single model m_i, when the sample predict range of another model m_j is wider than m_i, then the strategy that m_i and m_j detect attacks collaboratively following the rule of this article will improve the performance of the detection using only m_i.* The sample predict range of m_i is the size of a set that contains all samples that can be classified correctly by model m_i. The sample predict range of m_j is the size of a set that contains all samples that can be classified correctly by model m_j in the sample space of m_i.

Proof 3.1. First, we define the S_i is the sample space containing all traffic samples that could be generated in the network n_i. For a model m_i, it is trained by the samples collected from n_i (a part of S_i), then it can classify all the samples of the set PS_i correctly ($PS_i \in S_i$). Then we define, the gain of model m_i towards the detection system is

$$Gain(m_i) = \frac{PS_i}{S_i} * Weight(m_i) - \frac{S_i - PS_i}{S_i} * Weight(m_i)$$

where $\frac{PS_i}{S_i}$ is the probability that a random sample from S_i will be classified correctly by m_i and $\frac{S_i - PS_i}{S_i}$ is the probability that a random sample from S_i will be classified wrongly; $Weight(m_i)$ is the influence from a model on the voting based detection (we can regard it as a coefficient here). To express the equation more lightly, we use w_i to represent $Weight(m_i)$.

Then if the detection mechanism adopts only m_i, $w_i = 1$, then the gain of this detection system is

$$\begin{aligned} Gain(m_i) &= \frac{PS_i}{S_i} - \frac{S_i - PS_i}{S_i} \\ &= \frac{2PS_i}{S_i} - 1 \end{aligned}$$

If the detection system adopts another model m_j whose PS_j contains all the samples that it can classify correctly, then the gain of the detection system will be

$$\begin{aligned} Gain(m_i, m_j) =& \frac{PS_i}{S_i} * w_i - \frac{S_i - PS_i}{S_i} * w_i \\ &+ \frac{PS_j \cap S_i}{S_i} * w_j - \frac{S_i - PS_j \cap S_i}{S_i} * w_j \\ =& \frac{2PS_i}{S_i} * w_i - w_i + \frac{2(PS_j \cap S_i)}{S_i} * w_j - w_j \\ =& \frac{2PS_i}{S_i} * w_i + \frac{2(PS_j \cap S_i)}{S_i} * w_j - 1 \end{aligned}$$

We have the assumption *the sample predict range of another model m_j is wider than m_i*, i.e., the size of $PS_j \cap S_i$ is bigger than PS_i. Therefore, we have

$$\begin{aligned} Gain(m_i, m_j) \geq& \frac{2PS_i}{S_i} * w_i + \frac{2PS_i}{S_i} * w_j - 1 \\ =& \frac{2PS_i}{S_i} - 1 \\ =& Gain(m_i) \end{aligned}$$

If the detection assembles models $m_i, m_j, m_k, m_l...$, we can regard $m_j, m_k, m_l...$ as a model m_x that will have corresponding PS_x. Then we can use the above process to prove that $Gain(m_i, m_j, m_k, m_l...)$ is bigger than $Gain(m_i)$. That is, the performance of detection will be improved by collaborative learning.

From this proof, we set the target of ADCL as constructing the model m_x that can have the wider sample predict range, so that we can improve the detection performance. To reach this target, an intuitive method is to ensemble multiple different models to extend the sample predict range but we must guarantee the sample predict range has the overlap with the sample space of m_0 (the traffic of n_0). If other models are trained in the networks that are similar to n_0, then the sample predict range of them will have a higher probability to contain the overlap with the sample space of m_0. Then this target agrees with the 'leveraging multiple models trained in a similar environment' of our core idea.

Challenges and solution. There are two crucial problems raised: 1) how to get the models trained in a similar environment, and 2) how to make different models work collaboratively in detection. Besides, we also wonder 3) whether we can use the result of collaborative detection to optimize the performance of single model.

To solve the problem 1), we decide to build a Model Repository (MR) for storing models, and these models will be saved with their *fingerprint* that can be used to estimate the similarity between the environments where different models are trained. Based on the above design, we can search the MR to get the models trained in a similar environment. To solve the problem 2), we decide to use voting-based detection mechanism and each model of collaborative detection should be assigned an appropriate weight, so that we can control its impact on the detection. To solve the problem 3), we decide to use the result of collaborative detection to re-label the samples and use these samples to train the old model incrementally, so that the performance of old model can be improved. To achieve these goals, we propose the following system design.

The ADCL framework consists of two processes: model training and traffic detection. In the first process, users upload the training traffic and the network topology information to ADCL, then user can choose a specific machine learning mechanism (e.g., supervised learning mechanisms like Artificial Neural Networks (ANN) or unsupervised learning mechanisms like AutoEncoder) to train the corresponding model according to the uploaded traffic. Then the trained model will be saved in the Model Repository (MR) with its unique identifier and its fingerprint.

Fingerprint. The fingerprint of a model is a tuple to evaluate the topology information of the traffic that can be used to initially train this model. Essentially, the topology information of a network is a tree whose structure represents the topology of network and each tree node represents the type of corresponding device. We consider a typical IoT network has two main types: 1) single-hop centralized network, which is the form of smart home IoT that all IoT devices (e.g., light bulb, smart lock) are connected to an IoT hub or gateway (e.g., SmartThing Hub); 2) multiple-hop distributed network, which is the form of industrial IoT (e.g., Wireless Sensor Network) that all IoT devices aim to build a self-organized network and send their data to the sink or base station within the network.

In the multiple-hop IoT network, the routing topology is usually a tree-like structure so that the data transferred from child-nodes can be aggregated in the parent-node to improve the efficiency of data transmission. Therefore, the topology of IoT network can be represented with a multi-branch tree (the tree node may have more than two child-nodes). To encode the topology of a multi-branch tree more efficiently, we convert the multi-branch tree into a binary tree and then use the pre-order traversal and in-order traversal to generate two expression about the binary tree. Then they can be organized together to generate a tuple, called *fingerprint of the network*, for evaluating the topology information about the network. Based on the fingerprints of two IoT networks, we can estimate the similarity of them.

During the traffic detection process, the traffic and the topology information of the network will be uploaded to ADCL for performing the intrusion detection. First, the network topology information has to be input to the SMS where the information will be translated to the form of fingerprint, denoted *f*, to be used to search models trained with similar networks. The similarity of the fingerprint *f* and the fingerprints stored in MR can guide ADCL to select 'suitable' models from MR to perform the intrusion detection collaboratively. Here, if a model in MR is more similar to the fingerprint of the network, then this model is considered as more 'suitable' to be selected as a member of the detector. That is, SMS first sends a model request to MR, and then MR returns some models to form the detector. Note that each model in the detector will be assigned a weight according to their fingerprint (more 'suitable' means a bigger weight). When the traffic is input to the detector, all models in the detector will execute their classification about the uploaded traffic. All classification results will be considered together based on the weights of models, called *Similarity-based Voting (SV)*. Finally, the output of SV can determine whether the traffic contains an intrusion.

The output of SV is used to update the model of MR. Let *m* denote a model of SMS, and there are four cases that should be considered: 1) *m* is a supervised model, the traffic is identified as benign by *m* but as malicious by SV; 2) *m* is an unsupervised model, the traffic is identified as benign by *m* but as malicious by SV; 3) *m* is a supervised model, the traffic is identified as malicious by *m* but as benign by SV; 4) *m* is an unsupervised model, the traffic is identified as malicious by *m* but as benign by SV.

For case 1), ADCL can use the uploaded traffic with label 'malicious' to train *m* incrementally and use the retrained *m* to replace the old version in MR. For case 2), ADCL can train a new model to detect the malicious traffic sample. For case 3), ADCL can use the uploaded traffic with label 'benign' to train *m* incrementally and use the retrained *m* to replace the old version in MR. For case 4), ADCL can use the traffic samples to train *m* incrementally and use the retrained *m* to replace the old version in MR. The detail of model update will be introduced in Section 4.5.

Improving the adaptability and saving the cost. ADCL does not require users to spend the time collecting traffic from practical network for model training, because this process usually requires a long time to complete. Instead, ADCL allows users to train models before the deployment of NIDS with other traffic datasets. While users can definitely train a more precise model with traffic samples collected from the practical network afterwards. In this case, though ADCL adopts the models trained in advance, it can leverage the knowledge of multiple models trained in similar networks

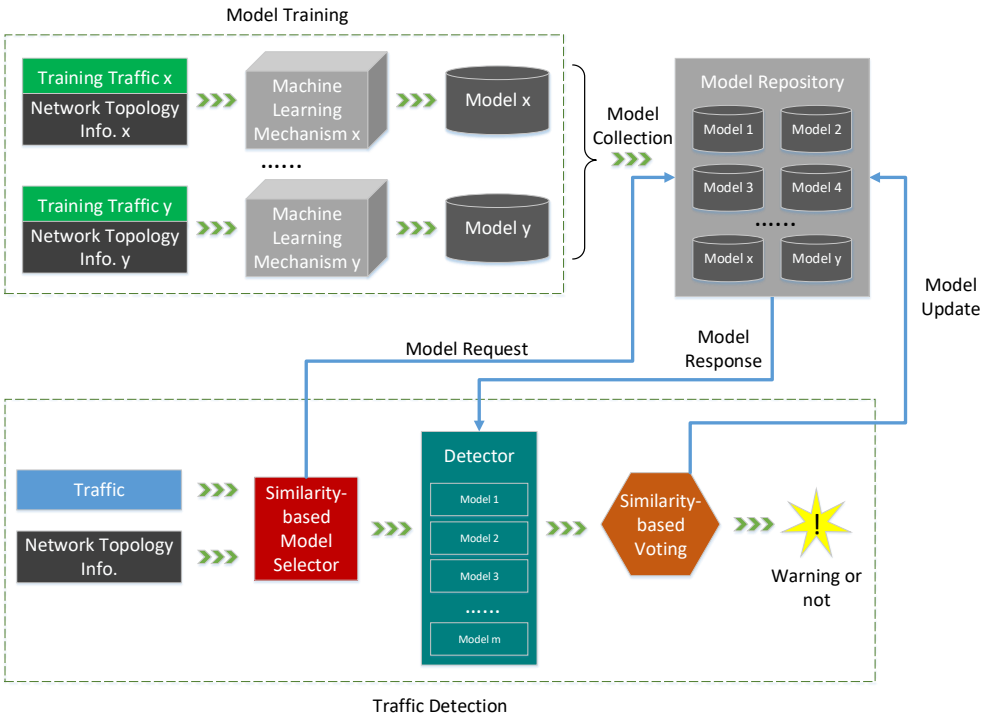

Fig. 1: ADCL Framework Overview

to help make a wiser detection decision, which can improve the adaptability towards the practical environment.

Mitigating the dependence on learning integrity. If the model is trained with the incomplete dataset, it is very likely to get a low detection rate. ADCL ensembles different models trained in similar networks, which can essentially improve the detection coverage in the sample space and optimize the detection capacity.

Combing the capacity of different models. Different models may have diverse performance based on concrete samples. If the traffic samples are going to be detected by a model that is weak in handling the sample distribution, then the detection performance may be affected largely. Hence ADCL ensembles different kinds of models to essentially extend the detection capacity by combining their advantages.

4 Design of ADCL

In this section, we introduce the design of ADCL including the major components. We use the network topology that provides model samples to formalize the fingerprint, in order to estimate the similarity between models (see Section 4.1). Section 4.2 explains the models that are stored in a repository offering collaborative models for detection. When ADCL performs the detection, it should select models from repository (see Section 4.3). The detection process is summarized in Section 4.4, and we introduce the model update mechanism in Section 4.5.

4.1 Network Topology Information

There are two major types of a typical IoT network: 1) single-hop centralized network, such as the form of smart home IoT that all IoT devices are connected to an IoT hub or gateway; 2) multiple-hop distributed network, such as the form of industrial IoT (e.g., Wireless Sensor Network) that all IoT devices build a self-organized network to send their data to the sink or base station.

An example of IoT network is shown in Fig. 2, where 1) smart devices send their data to the network hub in the smart home application, and 2) IoT devices send their data to the base station in Wireless Sensor Network (WSN) with a multi-hop path (e.g., *<Sensor 1, Sensor 3, Sensor 4, Base-station>*). In the multiple-hop IoT network, the routing topology is usually a tree-like structure where the tree root is base station. When the base station collects data from IoT nodes, the data transferred from child-nodes can be aggregated in the parent-node to improve the efficiency of data transmission.

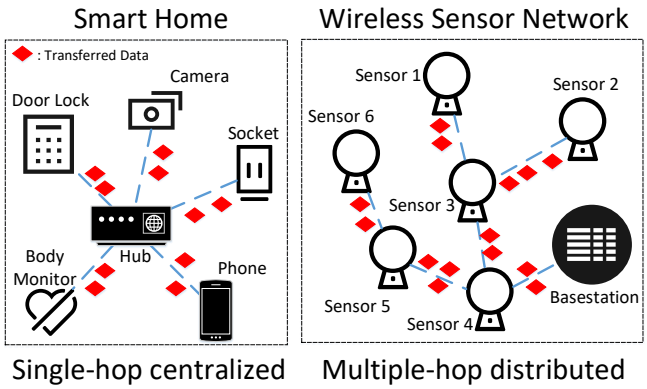

Fig. 2: IoT Networks

Therefore, the topology of IoT networks can be represented with a multi-branch tree. Based on the two cases in Fig. 2, we can generate the corresponding topology trees in Fig. 3. In particular, the topology of the smart home can be

expressed as below:

$\{value : Root, child : [\{value : DoorLock, child : NULL\}, \{value : BodyMoni., child : NULL\}, \{value : Camera, child : NULL\}, \{value : Phone, child : NULL\}, \{value : Socket, child : NULL\}]\}$.

Then the topology of the WSN case can be expressed as below:

$\{value : Root, child : [\{value : Sensor\ 4, child : [\{value : Sensor\ 5, child : [\{value : Sensor\ 6, child : NULL\}]\}, \{value : Sensor\ 3, child : [\{value : Sensor\ 1, child : NULL\}, \{value : Sensor\ 2, child : NULL\}]\}]\}]\}$.

It is worth noting that the multi-branch tree (topology info.) has to be encoded and stored in the Model Repository (MR) with the corresponding model. To encode the multi-branch tree efficiently, we convert it to a unique binary tree that can be encoded by the pre-order traversal and the in-order traversal. The convert algorithm is shown in Algorithm 1. The pre-order traversal and the in-order traversal are shown in Algorithm 2 and Algorithm 3 individually. As a result, the fingerprint of a network topology can be formalized as $Fin =< POF, IOF >$ where POF is the output of Algorithm 2 and IOF is the output of Algorithm 3. For example, the fingerprint of the WSN case, as shown in Fig. 3, can be expressed as: $Fin =<$ *POF:<Sensor 6, Sensor 5, Sensor 4, Sensor 1, Sensor 3, Sensor 2, Root>*, *IOF:<Root, Sensor 4, Sensor 5, Sensor 6, Sensor 3, Sensor 1, Sensor 2>>*.

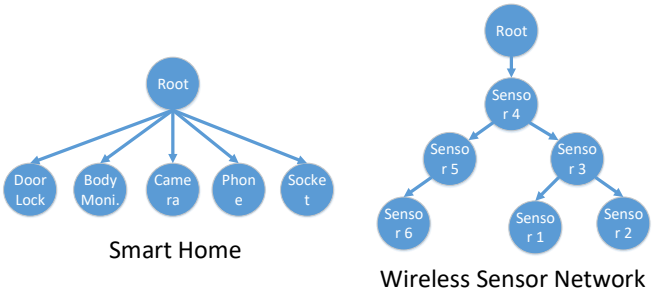

Fig. 3: Multi-branch Tree Topology

Algorithm 1 ConvertToBinaryTree: CTBT (MT)

Input: MT (The multi-branch tree);
Output: BT (The binary tree that the multi-branch tree converted to);

if MT is NULL **then**
  return NULL;
end if
$BT \leftarrow$ generate a binary tree node;
$BT.value \leftarrow MT.value$;
$BT.leftChild \leftarrow$ CTBT ($MT.child[0]$);
$brother \leftarrow BT.leftChild$;
$index \leftarrow 1$;
while $index <$ the count of the children of MT **do**
  $brother.rightChild \leftarrow$ CTBT ($MT.child[index]$);
  $brother \leftarrow brother.rightChild$;
  $index \leftarrow index + 1$;
end while
return BT;

Algorithm 2 Pre-Order Traversal: POT (BT)

Input: BT (Binary tree);
Output: POF (Pre-order fingerprint);

if BT is NULL **then**
  return NULL;
end if
$POF \leftarrow$ POT ($BT.leftChild$);
$POF \leftarrow POF +$ ','$+ BT.value$;
$POF \leftarrow POF +$ ','+ POT ($BT.rightChild$);
return POF;

Algorithm 3 In-Order Traversal: IOT (BT)

Input: BT (Binary tree);
Output: IOF (In-order fingerprint);

if BT is NULL **then**
  return NULL;
end if
$IOF \leftarrow BT.value$;
$IOF \leftarrow IOF +$ ','+ IOT ($BT.leftChild$);
$IOF \leftarrow IOF +$ ','+ IOT ($BT.rightChild$);
return IOF;

4.2 Model Repository (MR)

In ADCL, users upload their traffic samples to the platform and choose appropriate machine learning mechanisms to train models, then these models will be stored with their fingerprints in Model Repository (MR). Note that, the models stored in MR may come from different environments but can work collaboratively in detection. Thus, the record can be formalized as $Record =< ModelID, Fin >$ where $ModelID$ is the model identifier and Fin is the corresponding fingerprint. When ADCL performs the traffic detection process, the SMS will send the model request to MR. The MR provides a fingerprint list containing all the fingerprints to SMS, then SMS selects 'suitable' models based on its rules and informs MR to adopt which model(s) to construct the detector. When the detection result is available, ADCL will update the model in MR to improve its performance.

4.3 Similarity-based Model Selector (SMS)

During traffic detection process, when the traffic samples are uploaded to ADCL with the topology information, the data will be firstly input to the Similarity-based Model Selector (SMS), which aims at selecting the models trained with the traffic samples coming from the environment (network topology) that is similar to the target traffic environment. Let fm denote the fingerprint of searched models and fp denote the fingerprint of the target network. SMS selects models from MR and evaluates the similarity between fm and fp. The larger the similarity, the better the model.

In this work, we use the length of Longest Common Subsequence to estimate the similarity between fm and fp, as shown in Equation (1), where $LLCS$ can refer to Algorithm 4, which calculates the length of Longest Common Subsequence between two sequences.

$$Similarity(fm, fp) = \frac{LLCS(fm.POF, fp.POF) + LLCS(fm.IOF, fp.IOF)}{2 * Length(fp)} \quad (1)$$

Users can send the parameter mc (model count) to SMS, then SMS has to choose mc models from MR that have a bigger similarity compared with other models. It is worth noting that the similarity between fm and fp should be larger than zero. However, if MR cannot provide enough models (the count does not meet mc), then SMS has to select only those models that meet the requirements to construct the detector.

Algorithm 4 Length of Longest Common Subsequence: LLCS ($f1$, $f2$)

Input: $f1$ (fingerprint 1), $f2$ (fingerprint 2);
Output: $length$ (The length of the Longest Common Subsequence between fingerprint 1 and fingerprint 2);

$m \leftarrow$ the length of $f1$;
$n \leftarrow$ the length of $f2$;
$array[0...m][0...n] \leftarrow$ generate a 2-dimensions table;
$i \leftarrow 1$;
while $i <= m$ **do**
  $array[i, 0] \leftarrow 0$;
  $i \leftarrow i + 1$;
end while
$j \leftarrow 1$;
while $j <= n$ **do**
  $array[0, j] \leftarrow 0$;
  $j \leftarrow j + 1$;
end while
$i \leftarrow 1, j \leftarrow 1$;
while $i <= m$ **do**
  while $j <= n$ **do**
    if $f1[i] == f2[j]$ **then**
      $array[i][j] \leftarrow array[i-1][j-1] + 1$;
    else
      $array[i][j] \leftarrow max\{array[i-1][j], array[i][j-1]\}$;
    end if
    $j \leftarrow j + 1$;
  end while
  $i \leftarrow i + 1$;
end while
$length \leftarrow array[m][n]$;
return $length$;

4.4 Detector and Voting

In this section, we introduce the construction of detector and its voting based detection mechanism. Assume that models $m_1, m_2, ..., m_x$ are selected by SMS from MR to construct the detector, then each model will be assigned a corresponding weight that determines its influence towards the detection result, which is called Similarity-based Voting (SV). The weight of m_x can be evaluated by Equation (2) where fm_x is the fingerprint of m_x and fp is the fingerprint of target environment.

$$Weight(m_x) = \frac{Similarity(fm_x, fp)}{\sum_{i=1}^{x} Similarity(fm_x, fp)} \quad (2)$$

By considering that various models may have different detection capabilities, i.e., detecting a particular attack, the detection result of ADCL contains two main parts: one part for answering whether there is an attack; while the other part for answering what type of the attack is. It is worth noting that a traffic sample may contain different types of attacks at the same time so that different models may give a distinct result towards the same traffic sample. First, for answering whether there is an attack, the detector of ADCL makes a decision based on Equation (3), where $label_i$ indicates the label (or result) provided by m_i and $IsAttack(label_i)$ means whether the label represents an attack. If yes, then it should be '1'; else, it should be '0'. If $AttackScore$ is higher than 0.5, then the detector can identify there is an attack.

$$AttackScore = \sum_{i=1}^{x} IsAttack(label_i) * Weight(m_i) \quad (3)$$

To find what is the attack type, $AttackType(label_i)$ is used to represent the attack type identified by m_i (note that the result of $AttackType(label_i)$ is a set instead of only one attack type). Besides, we use $Capability(m_i)$ to represent the attack types that can be identified by model m_i and the result of $Capability(m_i)$ is also a set of attacks. For example, $Capability(m_1) = \{DoS\ attack, Mirai\ attack\}$. ADCL has the following steps:

1) selecting m_i from the detector with the following condition: $IsAttack(label_i)$ must satisfy the result of detector (i.e., if a detector determines there is an attack, $IsAttack(label_i)$ must be equal to 1).
2) creating a set S consisting of the selected model m_i. Here we use $w(m_i, type)$ to represent the weight of model m_i of S when m_i identifies a specific attack type: $type$, as shown in Equation (4). We also define $Confidence(type)$, the result confidence about identifying a specific attack, as shown in Equation (5), where $Cnt()$ is the function to calculate the count of a set.
3) creating an empty set AS, and an empty set $Result$;
4) with regard to each model m_i in S, adding each item of $AttackType(label_i)$ to AS;
5) with regard to each item t in AS, adding a tuple {type:t, confidence:$Confidence(t)$} to $Result$.

$$W(m_i, type) = \frac{Weight(m_i)}{\sum_{\substack{m_x \in S \\ type \in Capability(m_i)}} Weight(m_x)} \quad (4)$$

$$Confidence(type) = \sum_{\substack{m_i \in S \\ type \in Capability(m_i)}} Cnt(AttackType(label_i) \cap type) * W(m_i, type) \quad (5)$$

Then $Result$ provides the attack types identified by the detector and the corresponding confidence. It is worth emphasizing that we do not deny the risk that some models may suffer the false rate about the attack type. For example, assume the traffic sample only contains DoS attack, if model m_1 decides it is a DoS attack but m_2 decides it is a Mirai attack, then the detector will report the traffic sample with both a DoS attack and a Mirai attack. While this risk cannot be mitigated temporarily as ADCL considers the collaboration between different models and gives more attention to whether all attacks can be detected.

4.5 Model Update

In this section, we introduce the mechanism adopted by ADCL to update the detector models based on the detection result. The basic idea is to update those models having different results compared with the detector (i.e., SV). However, as a detector consists of multiple models, different models may provide distinct results towards even the same traffic sample. This might be caused by two reasons: (1) models give different answers because they are constructed in different training environments (though there exists a similarity between the environments), so they have different classification decisions; and (2) the learning of some models is not ideal due to that the training is offline or the poor integrity of training samples (in advance); thus, the model adaptability of the target network environment is weak.

To address this issue, an intuition is re-labeling the samples that are classified wrongly and retraining the model. Therefore, there are two questions needed to answer: which model needs to be updated and how to update the model. For the first question, considering the possibility of case (1), we only need to update the models with the highest similarity compared with the target network topology while having an opposite detection result compared with the detector. For the second question, we can consider two kinds of models: supervised model and unsupervised model. The supervised model is able to adjust the direction of training by changing the label of training samples. For example, if a model missed a malicious traffic sample during detection, then we can input this sample with a 'malicious' label to train the model incrementally for detection improvement. If a model mis-classifies a benign traffic sample, then this sample can be input with a 'benign' label to train the model for detection improvement.

While for unsupervised model, it is different from the supervised model as the training sample towards unsupervised model does not contain an explicit label. For example, autoencoder, a famous unsupervised model adopted widely in traffic detection, trains its model by reconstructing input samples without any label. This model detects malicious traffic only when it cannot reconstruct the traffic, thus changing label is not suitable for updating this model. That is, if the model misses the malicious traffic sample, then we have to train a new autoencoder whose training data must include this malicious traffic sample.

It is important to note that introducing the update mechanism of autoencoder is just an example. In practice, many unsupervised models may have totally different update mechanisms. Based on the above analysis, our ADCL can update models according to Algorithm 5.

Algorithm 5 Model Update: MU ($model$, $sample$, SV)

Input: $model$ (The model waiting for updating), $sample$ (The sample causing opposite result between $model$ and the detector), SV (Detector using Similarity-based Voting);

Output: $umodel$ (The model after updating);

1: **if** $model$ is supervised and $model.result$ == 'benign' and $SV.result$ == 'malicious' **then**
2: $umodel \leftarrow$ Label the $sample$ to be malicious and use the sample to train $model$ incrementally;
3: Replace $model$ in MR with $umodel$;
4: **else if** $model$ is unsupervised(autoencoder) and $model.result$ == 'benign' and $SV.result$ == 'malicious' **then**
5: **if** $model$ has not been updated before **then**
6: $umodel \leftarrow$ train a new autoencoder with $sample$;
7: Add $umodel$ to MR and will be denoted as $model.umodel$;
8: Set $model$ has been updated;
9: **else**
10: $umodel \leftarrow$ use the sample to train $model.umodel$ incrementally;
11: Replace $model.umodel$ in MR with $umodel$;
12: **end if**
13: **else if** $model$ is supervised and $model.result$ == 'malicious' and $SV.result$ == 'benign' **then**
14: $umodel \leftarrow$ Label the $sample$ to be benign and use the sample to train $model$ incrementally;
15: Replace $model$ in MR with $umodel$;
16: **else if** $model$ is unsupervised(autoencoder) and $model.result$ == 'malicious' and $SV.result$ == 'benign' **then**
17: $umodel \leftarrow$ Use the sample to train $model$ incrementally;
18: Replace $model$ in MR with $umodel$;
19: **end if**
20: **return** $umodel$;

TABLE 1: Labels and sample number of IoTID20 Dataset

Binary	Category	Subcategory
Normal(37861)	Normal(37861)	Normal(37861)
Anomaly(552222)	DoS(56093)	Syn Flooding(56093)
	Mirai(391823)	Brute Force(114416), HTTP Flooding(52640), UDP Flooding(172665), ACK Flooding(52102)
	MITM(33385)	ARP Spoofing(33385)
	Scan(70921)	Host Port(20927), OS(49994)

5 Evaluation

In this section, we provide an evaluation of ADCL in terms of its detection performance and the efficiency with model collaboration. In particular, we introduce the used dataset, describe the experimental setup, and analyze the results.

5.1 Dataset

There are several public datasets in the domain of intrusion detection, such as KDD-99 [54] and UNSW-NB15 [55], but they were not constructed based on IoT networks. In the evaluation, we thus used the IoTID20 dataset [56] as the base to train our models. The IoTID20 testbed consisted of many IoT devices and interconnecting structures. For instance, the devices include SKT NGU and EZVIZ WiFi camera that were connected to a smart home WiFi router. There are also laptops, tablets, smartphones connected to the router with attacks performed. The traffic in the network was captured and stored as PCAP files by CICflowmeter [57]. The IoTID20 dataset contains up to 80 network features and three label features including binary, category and subcategory, as shown in Table 1.

5.2 Experimental Setup

Traffic Generation of IoT Device. In the experiment, we created different IoT virtual devices that can generate specific traffic to the network and stimulate the device activities. For a specific type of IoT virtual device, we created a *traffic box* (TB) including a certain number of traffic samples from IoTID20 dataset, where each IoT virtual device can choose traffic samples from TB randomly. It is worth noting that traffic samples of TB were selected randomly from IoTID20 dataset, while different virtual devices with the same type would have the same TB. We guarantee that each TB could contain all types of traffic samples of IoTID20 dataset but not all traffic samples. Thus, virtual devices of different types can release different traffic.

In the evaluation, we created 20 types of IoT virtual devices and the traffic box of each device contained 1894 normal samples, 2805 syn-flooding attack samples, 5721 brute-force attack samples, 2632 http-flooding attack samples, 8634 udp-flooding attack samples, 2606 ack-flooding attack samples, 1670 arp-spoofing attack samples, 1047 hostport-scan attack samples and 2500 os-scan attack samples.

Network Structure and Running. In the experiment, we consider two kinds of network structures - single-hop centralized and multiple-hop distributed. For the former, we created multiple virtual devices and all of them were connected to a central node (hub), where we collected the network traffic. For the latter, we created multiple virtual devices connected with a multi-hop network in the form of a multi-branch tree.

Compared to the traffic generation of single-hop centralized network, in a multiple-hop distributed network, the traffic that comes from the child-nodes will be aggregated in their parent-node, which is very common in the application of WSN to save energy of IoT devices. That is, the network structure can influence the traffic generation. Therefore, our traffic generation should be related to the tree structure. To achieve this, we used Algorithm 6 to control the process of selecting samples that are related to the network topology. When we executed SS($root$) ($root$ is the network gateway), we could collect a batch of traffic samples.

Attack Traffic Injection. If we add traffic samples (with attacks) to the TB of a virtual device, then the attack traffic can be injected to the network. It is worth emphasizing that we selected traffic samples randomly from the IoTID20 dataset to each TB, indicating that different virtual devices of the same type can have the same attack capacity.

Training and Detection. In the training of supervised learning, we added traffic samples including normal samples and attack samples to the TB of each type of IoT virtual devices. Then in the network stimulation, the network hub or gateway could collect the traffic including both normal samples and attack samples, and these samples would be used to train the model. Note that we will limit the number of collected traffic so that some samples would not be learned by the model, similar to a practical situation. After training, we started running the network stimulation again, and this time the traffic collected by hub or gateway would be used for detection and verifying the model performance.

On the other hand, in the training of unsupervised learning (autoencoder), we only added normal traffic samples to the TB of each type of IoT virtual devices, and then the model could finish its training after the network stimulation. After the training process, attack samples would be added to the TB of each device and the network stimulation would run again for the detection. We used *training rate* to measure the samples selected from TB to train the model, and *test rate* to measure the samples selected from TB to test the model.

Hardware and Software. We launched the experiment with a x86 computer consisting of Intel Core i7-4700MQ, 16GB memory, and Ubuntu 18.04 LTS. The experiment was implemented by Python with machine learning algorithms running with scikit-learn 0.20 and tensorflow 1.13.0-rc1.

Algorithm 6 SelectSamples: SS (MT)

Input: MT (The multi-branch tree);
Output: $Samples$ (The selected traffic samples); $index$ (The index of selected sample)

if MT has children **then**
  $indexSet \leftarrow 0$;
  $Samples \leftarrow \{\}$;
  $i \leftarrow 0$;
  while $i <$ the count of the children of MT **do**
    $Samples\_t, index \leftarrow$ SS ($MT.child[i]$);
    $Samples \leftarrow Samples$ + $Samples\_t$;
    $indexSet \leftarrow indexSet$ + $index$;
    $i \leftarrow i$ + 1;
  end while
  $index \leftarrow indexSet$ mod the size of MT.TB;
  Add MT.TB[$index$] to $Samples$;
else
  $index \leftarrow$ generate a random number mod the size of MT.TB;
  $Samples \leftarrow \{MT$.TB[$index$]$\}$;
end if
return $Samples$, $index$;

5.3 Evaluation Metrics

As shown in Table 2, we used the following common metrics to evaluate the detection performance: *Accuracy = (TP+TN)/(TP+TN+FP+FN)*, *Precision = TP/(TP+FP)*, *Recall = TP/(TP+FN)*, *F-measure = (2Precision*Recall)/(Precision+Recall)*.

Model Selection and Settings. Due to the popularity and features, we adopted six supervised learning algorithms: Support Vector Machine (SVM), GaussianNB, LDA, Logic Regression, Decision Tree, Random Forest, and a widely used unsupervised learning algorithm: Autoencoder as our compared objects in the evaluation. For supervised learning, only when the sample is classified as its real type, this sample will be regarded as a True Positive; however, for unsupervised learning (autoencoder), as the mechanism does not have the capacity to identify a concrete attack type, we regard a sample as a True Positive if this sample is identified as malicious.

Since models may provide different detection results, ADCL ensembles multiple models based on the rules mentioned in Section 4.4. For calculating the accuracy, precision, recall and F-measure, ADCL selected one attack type at most for each sample and then selected the attack type with the biggest confidence (refer to Equation (5)). If the confidence between different attack-types is the same, we

TABLE 2: EXPERIMENT EVALUATION

		Detection Result		
		Malicious	**Benign**	**Total**
Reality	**Malicious**	True Positive (TP)	True Positive (FN)	P(Real Malicious)
	Benign	False Positive (FP)	True Negative (TN)	N(Real Benign)
	Total	P'(Detect Malicious)	N'(Detect Benign)	P+N

could pick up one attack type randomly. The model settings are summarized in Table 3.

Model Comparison. In the evaluation, the model used for comparison also represents all the detection mechanisms that adopt this model. For example, we list the autoencoder as our compared and analyzed object. Essentially it can represent all work that adopted autoencoder as detection mechanism, e.g., [9], [21]. The main goal is to figure out how much ADCL can obtain via collaborative learning compared with a single model.

TABLE 3: Model Settings

Model	Parameters setting
SVM	C=1, kernel='linear', decision_function_shape='ovr', max_iter=1000, default for others
Gaussion NB	Default setting
LDA	n_components=None, priors=None, shrinkage=None, solver='svd', store_covariance=False, tol=0.0001, default for others
Logistic Regression	penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, solver='sag', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, l1_ratio=None, default for others
Decision Tree	criterion='gini', splitter='best', max_depth=100, min_samples_split=2, min_samples_leaf=1, default for others
Random Forest	n_estimators=10, criterion='gini', max_depth=100, default for others
AutoEncoder	One hidden layer with 40 dimensions

5.4 Adaptability Improvement

To explore the adaptability improvement by ADCL in practical networks, we have the following settings. For each kind of model m_0 (e.g., SVM), we assume it will be deployed to network n_0, then we will use the traffic samples of n_0 to train m_0. We regard m_0 as a perfect-adaptive model because it is trained by the traffic samples from the deployed network. On the other hand, ADCL can select 3 models m_1, m_2, m_3 trained by the samples from networks n_1, n_2, n_3 individually. To avoid the impact by the model types, we set the type of $m_i (i = 1, 2, 3)$ as the same as the type of m_0.

The count of devices is 6 for all n_0, n_1, n_2, n_3, but for n_0 and $n_i (i = 1, 2, 3)$, only 3 devices of n_0 and n_i are the same. This setting guarantees that only a part (3 devices) of n_i is the same as a part (3 devices) of n_0 (i.e., n_i is similar to n_0, but not totally the same as n_0). Therefore, m_i does not learn the traffic samples of n_0 in advance, and it is trained under the similar environment n_i. Then we used the traffic samples of n_0 to examine the performance of m_0 and ADCL. By comparing the performance between m_0 and ADCL, we can figure out how much ADCL can obtain about the adaptability towards n_0 without learning any traffic samples from n_0 in advance. The training rate here is set as 0.7, and test rate is set as 0.3.

Fig. 4 describes the adaptability improvement. In most cases, the F-score of ADCL outperformed the single model. Especially, in the DoS-Syn Flooding case, ADCL obtains an obvious improvement (more than 85%) compared with the Gaussian NB model. For all cases, ADCL outperformed a single autoencoder obviously, which means for this kind of unsupervised model, the collaborative learning of ADCL could provide a significant efficiency. This is because, compared with other supervised learning mechanisms, autoencoder only learns normal traffic samples to obtain the capacity and reconstruct them successfully. The rate of normal samples is low (only 37861 samples in the whole dataset), which means that the learning of a single autoencoder is not so integrated. In this case, although ADCL ensembles autoencoders trained in other networks, the performance of ADCL could be well improved with more knowledge about normal samples, so as to obtain a better adaptability than a single model trained in the native network.

In DoS-Syn Flooding case and Mirai-UDP Flooding case, most models worked well except for Gaussian NB. Also, in most cases, if a single model has great performance, then ADCL ensembles the models of the same type, which may not bring a very high improvement and sometimes may even cause a light reduction. For example, for decision tree and random forest, even they are single models, they performed well in most cases compared with other types. While ADCL does not gain a great improvement by adopting decision tree and random forest. This is because the learning ability of these models is strong - they can classify samples well even without integrated training set. That is, their adaptability towards the practical environment would be strong too. While for some weak models, ADCL cannot bring the improvement because the learning ability of them is not good towards some attacks. For example, in the Scan-Host Port case and Scan-OS case, LDA and Logistic Regression worked not well. The main reason might be the features about these attacks are not appropriate for LDA and Logistics Regression.

Overall, ADCL has good adaptability towards n_0 without learning any traffic samples from n_0 in advance, and sometimes the adaptability is better than a single model trained in the native network. Hence we believe that ADCL can improve the detection adaptability.

5.5 Learning Integrity Improvement

To explore the improvement by ADCL towards mitigating the reliance on the learning integrity about practical networks, we have the following settings. For each kind of

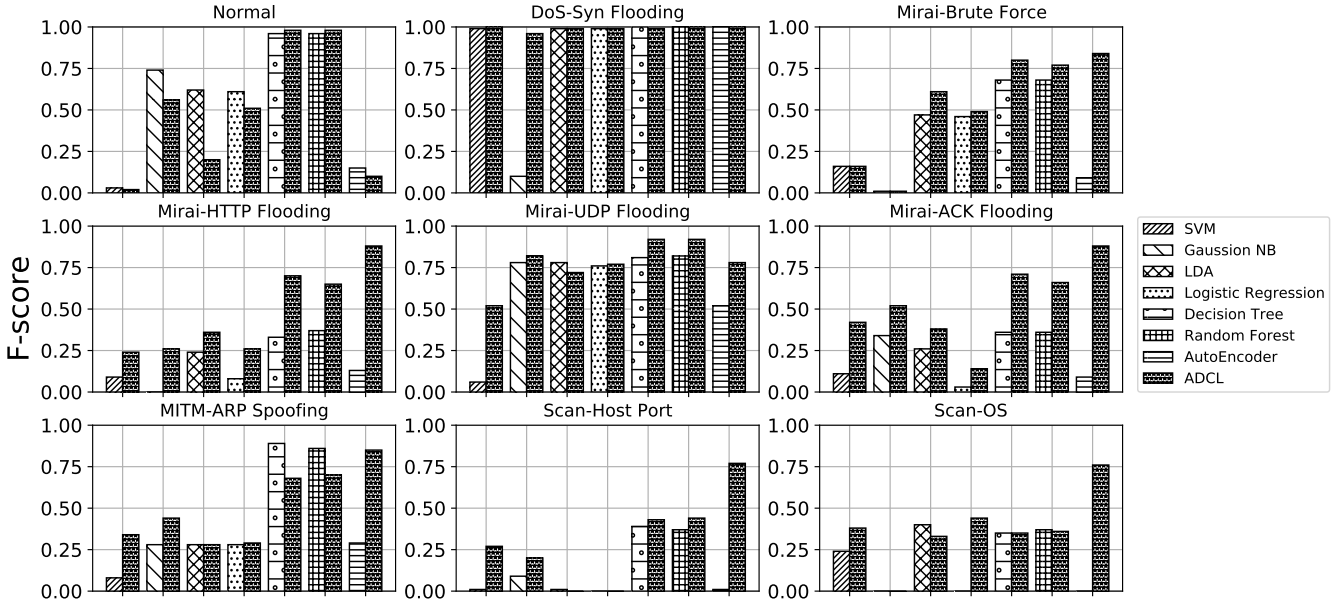

Fig. 4: Adaptability Improvement

model m_0 (e.g., SVM), we assume it will be deployed in the network n_0, then we will use the traffic samples of n_0 to train m_0 and we regard m_0 as a learning-limited model. This is because it is trained by part of traffic samples of n_0. On the other hand, ADCL will select 3 models m_1, m_2, m_3 trained by the samples from the network n_0. It is worth noting that the samples used to train m_1, m_2, m_3 are randomly selected. To avoid the impact caused by the model type, we set the type of $m_i(i = 1, 2, 3)$ as the same as the type of m_0. Then we used the traffic samples of n_0 to examine the performance of m_0 and ADCL. By comparing the performance between m_0 and ADCL, we can figure out how much ADCL can enhance the performance. The training rate of this case was set as 0.2 (low value), and the test rate was set as 0.3.

The learning integrity improvement is shown in Fig. 5. We found there is an interesting phenomenon - for normal case, ADCL could reduce the F-score of most models; however, for attack cases, ADCL could improve the F-score in the comparison with the single model. This is because the rate of normal samples is low (e.g., only 37861 samples in whole dataset). Also in this experiment, the training rate was set as 0.2, which means that the knowledge about normal traffic is very limited. Thus, multiple models with the limited knowledge may cause ADCL to suffer more false alarms. If the Model Repository (MR) has very limited knowledge about a sample, then there could be a negative impact on sample classification in ADCL. In most cases, when a single model has the worse learning integrity, ADCL would be very helpful to increase the recall of attack samples. That is, we can reduce the false rates about normal samples by providing more normal samples.

5.6 Model Capacity Improvement

To explore the improvement by ADCL towards combing the capacity of different models, we have the following settings. For each kind of model $m_i(i = 0 - 6)$ (i.e., SVM, Gaussion NB, LDA, Logistic Regression, Decision Tree, Random Forest and Autoencoder individually), we assume they will be deployed to the network n, then we can use the traffic samples of n to train m_i. We regard m_i as a capacity-limited model because it is trained by a specific mechanism. On the other hand, ADCL would select 7 models m_0 - m_6 trained by the samples from the networks n. It is worth noting that the samples used to train $m_i(i = 0 - 6)$ are the same to avoid the influence caused by the training dataset. Then we use the traffic samples of n to measure the performance of m_i and ADCL. By comparing the performance between m_i and ADCL, we can figure out how much ADCL can enhance the performance through combing different models with the same training samples. The training rate of this case was set as 0.7, and the test rate was set as 0.3.

The capacity improvement is shown in Fig. 6. It is found that decision tree and random forest have the best performance compared with the other models. Instead, both SVM and autoencoder did not work well. As we know, the classification by decision tree does not rely on specific domain knowledge and parameter assumptions, so it usually has a better performance if there is no appropriate configuration. Random forest ensembles multiple decision trees so it inherits the advantage of decision tree. However, for SVM, its performance relies heavily on the parameter settings, which is not good at handling multi-classification problem. For autoencoder, as our evaluation was based on the settings from related work [9], [21] - a simple Artificial Neural Network (ANN) with only a hidden layer, the learning capacity is relatively limited.

It is also found that when the single model has limited capacity towards specific attacks, combining multiple models collaboratively could bring a significant improvement of detection. For example, in the case of Mirai-Brute Force, Mirai-HTTP Flooding, Mirai-ACK Flooding, MITM-ARP Spoofing and Scan-OS, ADCL outperformed other single models obviously, ranged from 37% up to 88%.

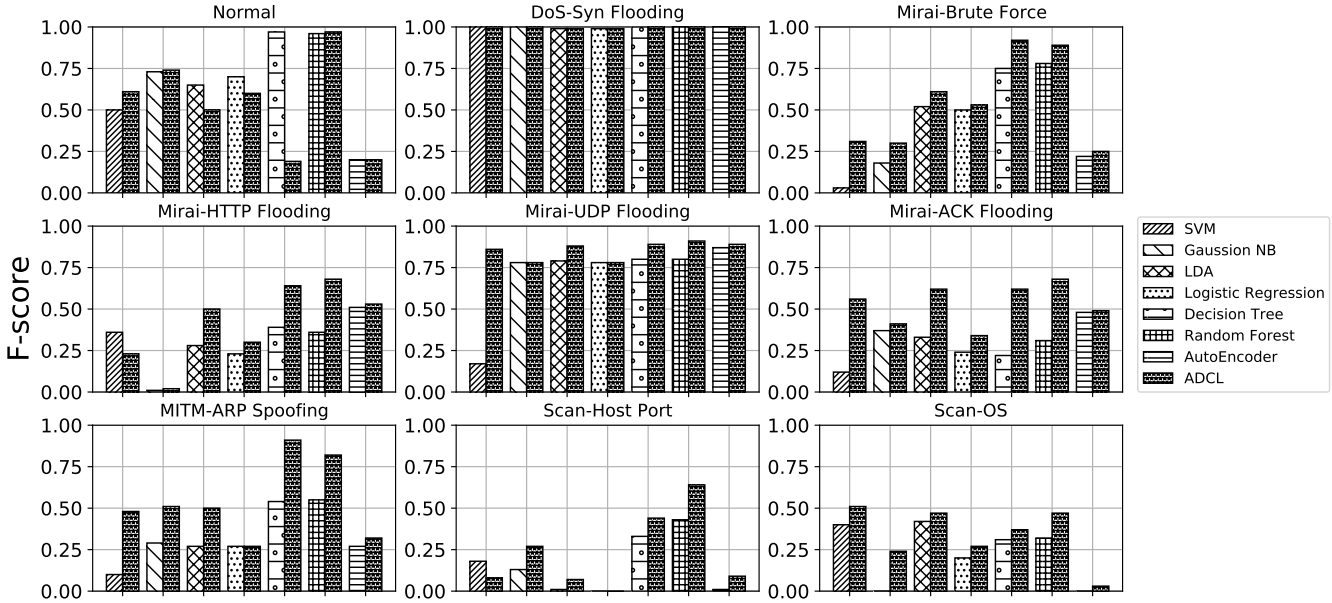

Fig. 5: Learning Integrity Improvement

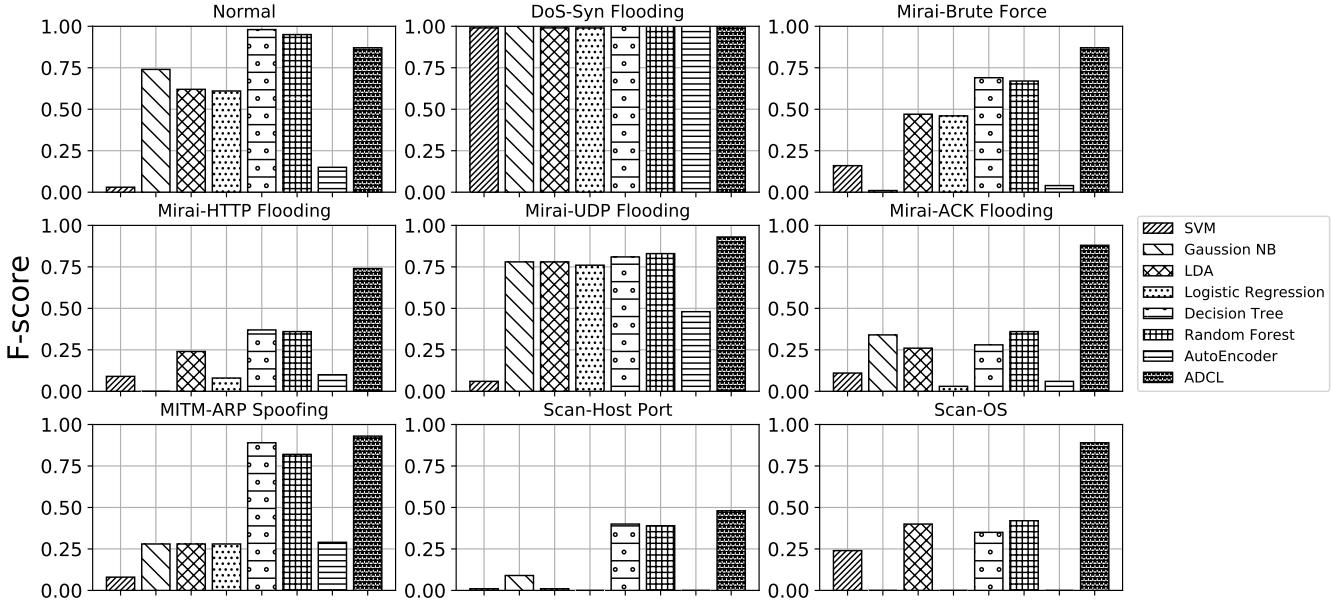

Fig. 6: Capacity Improvement

5.7 Model Update Improvement

Under ADCL, to explore the improvement of each model after the model update, we have the following settings. For each kind of model $m_i(i = 0 - 6)$ (i.e., SVM, Gaussion NB, LDA, Logistic Regression, Decision Tree, Random Forest and Autoencoder individually), we assume they will be deployed in the network n, then we will use the traffic samples of n to train m_i. On the other hand, ADCL will select 7 models m_0 - m_6 trained by the samples from the network n. The samples used to train $m_i(i = 0 - 6)$ are the same to avoid the influence caused by the training dataset. Then we use the traffic samples of n to examine the performance of m_i and ADCL. After the detection, we use the result to update model m_i (the update process can refer to Section 4.5). It is worth noting that we adopted an incremental training process, that is, for models that support online learning (e.g., training model without previous training set), we can just input those samples re-labeled by ADCL; while for those models that do not support online learning (e.g., the dataset used to train a model must contain previous training set), we can input the detection result of ADCL and the previous training set of the model. In this work, as SVM, Gaussion NB, LDA, Logistic Regression, Decision Tree, Random Forest and Autoencoder do not support online learning, we adopted the second option. The training rate of this case was set as 0.7, and the test rate was set as 0.3. When the model update is complete, we will use the samples from the last detection round to examine the detection improvement.

The update improvement is shown in Fig. 7. It is found

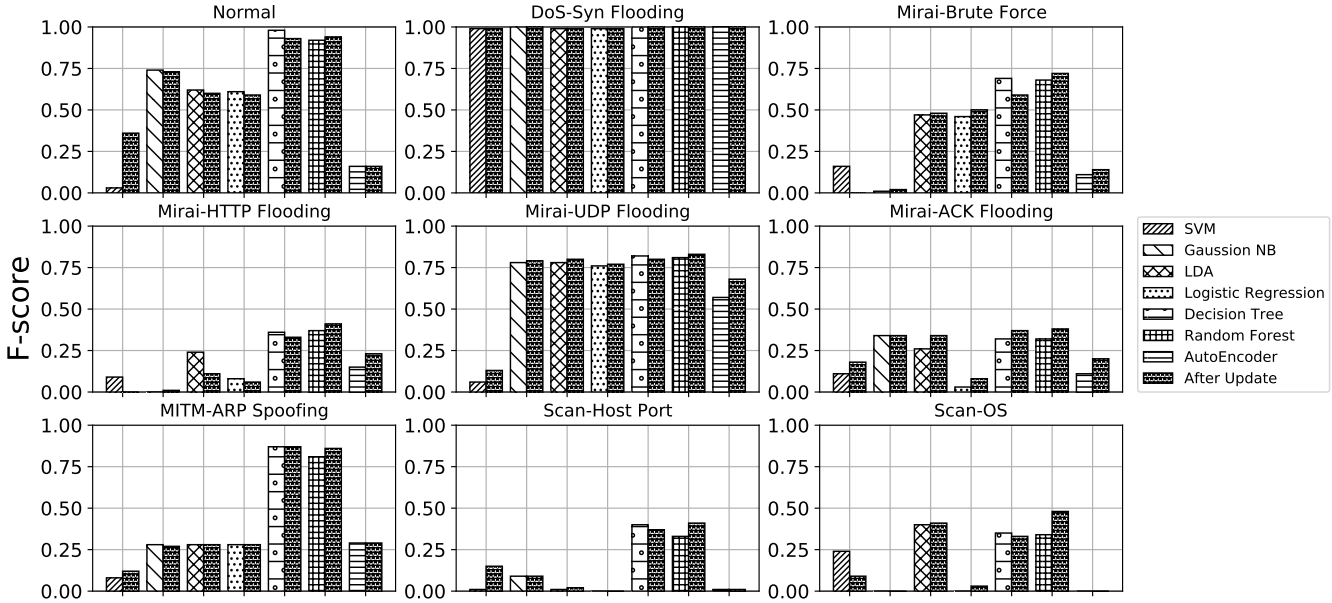

Fig. 7: Update Improvement

that the improvement raised by the model update was not so obvious in most cases, but it could indeed increase the F-score. This interesting phenomenon is mainly caused by the fact that ADCL updates a model 'incrementally' instead of abandoning its old knowledge cultivated by previous training samples. We regard the knowledge of a model obtained from previous training samples as *old knowledge*, and the knowledge obtained from the detection of ADCL as *new knowledge*. As there is a potential conflict between old knowledge and new knowledge, the model update tries to teach the old model to accept the new knowledge. Since the count of the samples of new knowledge (the samples for detection) is relatively small compared with the previous samples of old knowledge, the weight of new knowledge is small compared with old knowledge. This is because the model always tries to achieve the best performance towards 'most of samples', causing the learning improvement to be less significant. However, with the detection iteration running for a longer time, ADCL could obtain more detection results, and then the weight of new knowledge would increase to improve the detection performance. Overall, the improvement of model update is progressive.

5.8 Discussion and Limitations

In the evaluation, we investigated the improvement provided by ADCL in the aspects of adaptability, learning integrity, model capacity, and model update. Overall, it is found that ADCL can achieve better detection performance as compared with the mechanisms with a single model, i.e., ADCL can provide a better F-score by up to 80% in improving adaptability, up to 42% in mitigating reliance, up to 85% in improving model capacity and up to 15% in improving model update.

On the other hand, theoretically, in two cases, ADCL may cause a negative impact on detection performance: (1) Extremely unique predicted samples of different models; (2) The capacity of a model outperforms the other models in MR for a specific scenario. For case (1), it means the sample space (the set of all samples that can be generated) of a single model has a very small overlap with the predicted range (the set of all samples that can be classified correctly) of models in MR. Thus, it does not meet the condition in Theorem 1 - 'when the sample predict range of another model is wider'. In this case, ADCL usually cannot offer the improvement because extra models from MR cannot provide enough detection capacity. However, since the extra models selected by ADCL have a similar training environment and the user can set the similarity threshold to select models from MR, these can guarantee the overlap and avoid this problem to some extent. For case (2), it means the capacity of a single model is strong enough but the models provided by MR are too weak compared with a single model. This also disobeys the condition in Theorem 1 - 'when the sample predict range of another model is wider'. Thus, the improvement of ADCL cannot be guaranteed. While the single model has strong capacity and is trained by enough training samples, the model could be very satisfied in practice.

6 Conclusion

In a distributed environment such as IoT, network intrusion detection system is facing many new challenges caused by the learning mechanism and the available training data. In this work, we developed ADCL, a collaborative learning based detection framework, which can improve the detection in the aspects of adaptability, learning integrity, model capacity and model update. It could leverage the knowledge of different models trained in a similar environment to build a smarter detector. In particular, ADCL made the following contributions: 1) proposing a concept of model fingerprint to record the network topology information where the model is trained; 2) designing a similarity-based model selector to choose the models trained in a similar environment; 3) developing a detector that ensembles multiple models

and leverages the collaboration of models to improve the detection performance; 4) devising a model update mechanism to optimize the performance of a single model automatically, by feeding the detection result from ADCL. The experimental results indicated that ADCL can reach better performance by up to 80% in improving adaptability, up to 42% in mitigating learning integrity, up to 85% in improving model capacity and up to 15% in model update.

ACKNOWLEDGMENT

This work is supported by the National Key R&D Program of China under No. 2021YFB2700500 and 2021YFB2700502, the Open Fund of Key Laboratory of Civil Aviation Smart Airport Theory and System, Civil Aviation University of China under No. SATS202206, the National Natural Science Foundation of China under No. U20B2050, Public Service Platform for Basic Software and Hardware Supply Chain Guarantee under No. TC210804A.

REFERENCES

[1] Z. Ma, L. Liu, and W. Meng, "Eld: Adaptive detection of malicious nodes under mix-energy-depleting-attacks using edge learning in iot networks," in *International Conference on Information Security*. Springer, 2020, pp. 255–273.

[2] C. Cai, R. Zheng, and J. Luo, "Ubiquitous acoustic sensing on commodity iot devices: A survey," *IEEE Commun. Surv. Tutorials*, vol. 24, no. 1, pp. 432–454, 2022.

[3] Y. Zhang and Q. Liu, "On iot intrusion detection based on data augmentation for enhancing learning on unbalanced samples," *Future Generation Computer Systems*, vol. 133, pp. 213–227, 2022.

[4] M. Abdel-Basset, H. Hawash, R. K. Chakrabortty, and M. J. Ryan, "Semi-supervised spatiotemporal deep learning for intrusions detection in iot networks," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12 251–12 265, 2021.

[5] O. A. Alghanam, W. Almobaideen, M. Saadeh, and O. Adwan, "An improved pio feature selection algorithm for iot network intrusion detection system based on ensemble learning," *Expert Systems with Applications*, vol. 213, p. 118745, 2023.

[6] F.-Q. Li, R.-J. Zhao, S.-L. Wang, L.-B. Chen, A. W.-C. Liew, and W. Ding, "Online intrusion detection for internet of things systems with full bayesian possibilistic clustering and ensembled fuzzy classifiers," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 11, pp. 4605–4617, 2022.

[7] S. Roy, J. Li, B. Choi, and Y. Bai, "A lightweight supervised intrusion detection mechanism for iot networks," *Future Gener. Comput. Syst.*, vol. 127, pp. 276–285, 2022.

[8] X. Hu, C. Zhu, G. Cheng, R. Li, H. Wu, and J. Gong, "A deep subdomain adaptation network with attention mechanism for malware variant traffic identification at an iot edge gateway," *IEEE Internet of Things Journal*, 2022.

[9] T. Luo and S. G. Nagarajan, "Distributed anomaly detection using autoencoder neural networks in wsn for iot," in *2018 ieee international conference on communications (icc)*. IEEE, 2018, pp. 1–6.

[10] J. Gu, L. Wang, H. Wang, and S. Wang, "A novel approach to intrusion detection using svm ensemble with feature augmentation," *Computers & Security*, vol. 86, pp. 53–62, 2019.

[11] J. Gu and S. Lu, "An effective intrusion detection approach using svm with naïve bayes feature embedding," *Computers & Security*, vol. 103, p. 102158, 2021.

[12] P. Hadem, D. K. Saikia, and S. Moulik, "An sdn-based intrusion detection system using svm with selective logging for ip traceback," *Computer Networks*, vol. 191, p. 108015, 2021.

[13] K. S. Bhosale, M. Nenova, and G. Iliev, "Modified naive bayes intrusion detection system (mnbids)," in *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*. IEEE, 2018, pp. 291–296.

[14] D. He, X. Liu, J. Zheng, S. Chan, S. Zhu, W. Min, and N. Guizani, "A lightweight and intelligent intrusion detection system for integrated electronic systems," *IEEE Network*, vol. 34, no. 4, pp. 173–179, 2020.

[15] Y. Yuan, L. Huo, and D. Hogrefe, "Two layers multi-class detection method for network intrusion detection system," in *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2017, pp. 767–772.

[16] Z. Elkhadir and B. Mohammed, "A cyber network attack detection based on gm median nearest neighbors lda," *Computers & Security*, vol. 86, pp. 63–74, 2019.

[17] M. Hasan, M. M. Islam, M. I. I. Zarif, and M. Hashem, "Attack and anomaly detection in iot sensors in iot sites using machine learning approaches," *Internet of Things*, vol. 7, p. 100059, 2019.

[18] H. Zhang, S. Dai, Y. Li, and W. Zhang, "Real-time distributed-random-forest-based network intrusion detection system using apache spark," in *2018 IEEE 37th international performance computing and communications conference (IPCCC)*. IEEE, 2018, pp. 1–7.

[19] L. Vu, Q. U. Nguyen, D. N. Nguyen, D. T. Hoang, E. Dutkiewicz *et al.*, "Learning latent distribution for distinguishing network traffic in intrusion detection system," in *ICC*. IEEE, 2019, pp. 1–6.

[20] X. Li, W. Chen, Q. Zhang, and L. Wu, "Building auto-encoder intrusion detection system based on random forest feature selection," *Computers & Security*, vol. 95, p. 101851, 2020.

[21] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," in *Network and Distributed System Security Symposium (NDSS)*, 2018.

[22] N. Moustafa, B. Turnbull, and K.-K. R. Choo, "An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4815–4830, 2018.

[23] Y. Zhou, G. Cheng, S. Jiang, and M. Dai, "Building an efficient intrusion detection system based on feature selection and ensemble classifier," *Computer Networks*, vol. 174, p. 107247, 2020.

[24] Y. Zhong, W. Chen, Z. Wang, Y. Chen, K. Wang, Y. Li, X. Yin, X. Shi, J. Yang, and K. Li, "Helad: A novel network anomaly detection model based on heterogeneous ensemble learning," *Computer Networks*, vol. 169, p. 107049, 2020.

[25] Z. Ma, L. Liu, and W. Meng, "Towards multiple-mix-attack detection via consensus-based trust management in iot networks," *Computers & Security*, vol. 96, p. 101898, 2020.

[26] S. Latif, Z. e Huma, S. S. Jamal, F. Ahmed, J. Ahmad, A. Zahid, K. Dashtipour, M. U. Aftab, M. Ahmad, and Q. H. Abbasi, "Intrusion detection framework for the internet of things using a dense random neural network," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 9, pp. 6435–6444, 2021.

[27] R. Zhao, G. Gui, Z. Xue, J. Yin, T. Ohtsuki, B. Adebisi, and H. Gacanin, "A novel intrusion detection method based on lightweight neural network for internet of things," *IEEE Internet of Things Journal*, 2021.

[28] Y. Wu, L. Nie, S. Wang, Z. Ning, and S. Li, "Intelligent intrusion detection for internet of things security: A deep convolutional generative adversarial network-enabled approach," *IEEE Internet of Things Journal*, 2021.

[29] T.-N. Dao and H. Lee, "Stacked autoencoder-based probabilistic feature extraction for on-device network intrusion detection," *IEEE Internet of Things Journal*, 2021.

[30] A. Shahraki, A. Taherkordi, and Ø. Haugen, "Tonta: Trend-based online network traffic analysis in ad-hoc iot networks," *Computer Networks*, vol. 194, p. 108125, 2021.

[31] S. Yılmaz, E. Aydogan, and S. Sen, "A transfer learning approach for securing resource-constrained iot devices," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4405–4418, 2021.

[32] E. Gyamfi and A. D. Jurcut, "Novel online network intrusion detection system for industrial iot based on oi-svdd and as-elm," *IEEE Internet of Things Journal*, 2022.

[33] L. Zhiqiang, G. Mohiuddin, Z. Jiangbin, M. Asim, and W. Sifei, "Intrusion detection in wireless sensor network using enhanced empirical based component analysis," *Future Generation Computer Systems*, vol. 135, pp. 181–193, 2022.

[34] R. Kumar, A. Malik, and V. Ranga, "An intellectual intrusion detection system using hybrid hunger games search and remora optimization algorithm for iot wireless networks," *Knowledge-Based Systems*, vol. 256, p. 109762, 2022.

[35] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, "E-graphsage: A graph neural network based intrusion detection system for iot," in *IEEE/IFIP NOMS*. IEEE, 2022, pp. 1–9.

[36] Y. Zhang, C. Yang, K. Huang, and Y. Li, "Intrusion detection of industrial internet-of-things based on reconstructed graph neural

networks," *IEEE Transactions on Network Science and Engineering*, 2022.

[37] W. Liang, Y. Hu, X. Zhou, Y. Pan, I. Kevin, and K. Wang, "Variational few-shot learning for microservice-oriented intrusion detection in distributed industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 8, pp. 5087–5095, 2021.

[38] Y. Lai, L. Tong, J. Liu, Y. Wang, T. Tang, Z. Zhao, and H. Qin, "Identifying malicious nodes in wireless sensor networks based on correlation detection," *Computers & Security*, vol. 113, p. 102540, 2022.

[39] A. Telikani, J. Shen, J. Yang, and P. Wang, "Industrial iot intrusion detection via evolutionary cost-sensitive learning and fog computing," *IEEE Internet of Things Journal*, vol. 9, no. 22, pp. 23 260–23 271, 2022.

[40] Y. Chen, Q. Lin, W. Wei, J. Ji, K.-C. Wong, and C. A. C. Coello, "Intrusion detection using multi-objective evolutionary convolutional neural network for internet of things in fog computing," *Knowledge-Based Systems*, vol. 244, p. 108505, 2022.

[41] O. A. Wahab, "Intrusion detection in the iot under data and concept drifts: Online deep learning approach," *IEEE Internet of Things Journal*, 2022.

[42] P. Krishnan, K. Jain, R. Buyya, P. Vijayakumar, A. Nayyar, M. Bilal, and H. Song, "Mud-based behavioral profiling security framework for software-defined iot networks," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6611–6622, 2021.

[43] P. Ruzafa-Alcazar, P. Fernandez-Saura, E. Marmol-Campos, A. Gonzalez-Vidal, J. L. H. Ramos, J. Bernal, and A. F. Skarmeta, "Intrusion detection based on privacy-preserving federated learning for the industrial iot," *IEEE Transactions on Industrial Informatics*, 2021.

[44] V. Mothukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, "Federated-learning-based anomaly detection for iot security attacks," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2545–2554, 2021.

[45] R. Zhao, Y. Wang, Z. Xue, T. Ohtsuki, B. Adebisi, and G. Gui, "Semi-supervised federated learning based intrusion detection method for internet of things," *IEEE Internet of Things Journal*, 2022.

[46] O. Aouedi, K. Piamrat, G. Muller, and K. Singh, "Federated semi-supervised learning for attack detection in industrial internet of things," *IEEE Transactions on Industrial Informatics*, 2022.

[47] Z. Lian and C. Su, "Decentralized federated learning for internet of things anomaly detection," in *ACM AsiaCCS*, 2022, pp. 1249–1251.

[48] T. V. Khoa, D. T. Hoang, N. L. Trung, C. T. Nguyen, T. T. T. Quynh, D. N. Nguyen, N. V. Ha, and E. Dutkiewicz, "Deep transfer learning: A novel collaborative learning model for cyberattack detection systems in iot networks," *IEEE Internet of Things Journal*, 2022.

[49] H. Jiang, J. Lin, and H. Kang, "Fgmd: A robust detector against adversarial attacks in the iot network," *Future Generation Computer Systems*, vol. 132, pp. 194–210, 2022.

[50] S. T. Mehedi, A. Anwar, Z. Rahman, K. Ahmed, and R. Islam, "Dependable intrusion detection system for iot: A deep transfer learning based approach," *IEEE Trans. Ind. Informatics*, vol. 19, no. 1, pp. 1006–1017, 2023.

[51] Z. Ma, L. Liu, and W. Meng, "DCONST: detection of multiple-mix-attack malicious nodes using consensus-based trust in iot networks," in *ACISP*, vol. 12248. Springer, 2020, pp. 247–267.

[52] X. Sun, W. Meng, W.-Y. Chiu, and B. Lampe, "TDL-IDS: Towards a transfer deep learning based intrusion detection system." in *IEEE GLOBECOM*, 2022, pp. 1–6.

[53] W. Meng, Y. Cai, L. T. Yang, and W. Chiu, "Hybrid emotion-aware monitoring system based on brainwaves for internet of medical things," *IEEE Internet Things J.*, vol. 8, no. 21, pp. 16 014–16 022, 2021.

[54] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE symposium on computational intelligence for security and defense applications*. IEEE, 2009, pp. 1–6.

[55] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS)*. IEEE, 2015, pp. 1–6.

[56] I. Ullah and Q. H. Mahmoud, "A scheme for generating a dataset for anomalous activity detection in iot networks," in *Canadian Conference on Artificial Intelligence*. Springer, 2020, pp. 508–520.

[57] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Charaterization of tor traffic using time based features." in *ICISSP*, 2017, pp. 253–262.

BIOGRAPHY

Zuchao Ma received his Bachelor's degree in 2018, from the Nanjing University of Aeronautics and Astronautics, China. He is currently a master student in College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His research interests include Cloud Security, System Security and IoT Security.

Liang Liu is currently an Associate Professor in the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu Province, China. His research interests include distributed computing, big data and system security. He received the B.S. degree in computer science from Northwestern Polytechnical University, Xi'an, Shanxi Province, China in 2005, and the Ph.D. degree in computer science from Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu Province, China in 2012.

Weizhi Meng is currently an Associate Professor in the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Denmark. He obtained his Ph.D. degree in Computer Science from the City University of Hong Kong (CityU). He is a recipient of the Hong Kong Institution of Engineers (HKIE) Outstanding Paper Award for Young Engineers/Researchers in both 2014 and 2017. His primary research interests are cyber security, artificial intelligence and blockchain technology, including intrusion detection, smartphone security and biometric authentication. He is a senior member of IEEE.

Xiapu Luo is currently an Associate Professor with the Department of Computing, The Hong Kong Polytechnic University. His current research interests include mobile security and privacy, blockchain and smart contracts, network security and privacy, and software engineering. His work appeared in top venues in the areas of security, software engineering, and networking. He is a senior member of IEEE.

Lisong Wang is an associated professor at the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. He receives his Ph.D degree in 2010 form Nanjing University of Aeronautics and Astronautics. His research interests include wireless sensor network, data management in distributed environment, and formal methods.

Wenjuan Li is currently a Research Assistant Professor with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University. She received the Research Tuition Scholarships and Outstanding Academic Performance Award during her doctorate studies. Her research interests include network management and security, intrusion detection, blockchain technology, and E-commerce security. She is a senior member of IEEE.