High-Confidence Computing 5 (2025) 100230

Contents lists available at ScienceDirect

High-Confidence Computing

journal homepage: www.sciencedirect.com/journal/high-confidence-computing

Research article

Modular neural network for edge-based detection of early-stage IoT botnet

Duaa Alqattan [a,b,*], Varun Ojha [a], Fawzy Habib [a,c], Ayman Noor [d], Graham Morgan [a], Rajiv Ranjan [a]

[a] School of Computing, Newcastle University, Newcastle Upon Tyne NE1 7RU, United Kingdom
[b] Alahsa Technical College, Technical and Vocational Training Corporation, Alahsa 11472, Saudi Arabia
[c] Computer Science College, University of Jeddah, Jeddah 21959, Saudi Arabia
[d] Department of Computer Science, College of Computer Science and Engineering, Taibah University, Madinah 41477, Saudi Arabia

ARTICLE INFO

Article history:
Received 22 December 2023
Revised 8 January 2024
Accepted 8 March 2024
Available online 16 April 2024

Keywords:
Modular neural network
IoT botnet
Edge computing
Botnet detection

ABSTRACT

The Internet of Things (IoT) has led to rapid growth in smart cities. However, IoT botnet-based attacks against smart city systems are becoming more prevalent. Detection methods for IoT botnet-based attacks have been the subject of extensive research, but the identification of early-stage behaviour of the IoT botnet prior to any attack remains a largely unexplored area that could prevent any attack before it is launched. Few studies have addressed the early stages of IoT botnet detection using monolithic deep learning algorithms that could require more time for training and detection. We, however, propose an edge-based deep learning system for the detection of the early stages of IoT botnets in smart cities. The proposed system, which we call EDIT (Edge-based Detection of early-stage IoT Botnet), aims to detect abnormalities in network communication traffic caused by early-stage IoT botnets based on the modular neural network (MNN) method at multi-access edge computing (MEC) servers. MNN can improve detection accuracy and efficiency by leveraging parallel computing on MEC. According to the findings, EDIT has a lower false-negative rate compared to a monolithic approach and other studies. At the MEC server, EDIT takes as little as 16 ms for the detection of an IoT botnet.

1. Introduction

Smart cities integrated with sensors are the peak of urban innovation in the digital era. They are important for improving the quality of public services and people's lives. A smart city is dependent on the Internet of Things (IoT) and edge/cloud computing for operation. The term "Internet of Things" refers to the ability of devices to talk to each other over the Internet and other networks using sensors built into the devices. Each IoT device in smart cities keeps track of how it is being used and what is going on around it and shares this information with its network. This includes information gathered from residents, buildings, public transportation vehicles, and healthcare facilities, among other domains. The huge quantities of data collected by IoT devices in smart cities are sent to servers in either an edge or cloud environment, where they are analysed to improve the management of facilities, equipment, and services. As smart cities proliferate, the attack surface expands due to the expansion of billions of IoT devices connected to each other and to edge and cloud servers. Attackers might exploit IoT device vulnerabilities to form an IoT botnet and launch attacks against servers in the edge or cloud environment, with potentially catastrophic consequences for the smart city's systems.

IoT botnet-based attacks are continually becoming a serious concern in smart cities. IoT botnets are widespread; they spread quickly and have the ability to do more damage than any other kind of disruptive behaviour. This is primarily due to the fact that an IoT botnet is a collection of compromised IoT devices that a bot master (attacker) can use to execute commands through a simple process of IoT botnet formation and attack execution [1,2]. One example of such an IoT botnet was in 2016, when a botnet known as Mirai malware was responsible for a Distributed Denial of Services (DDoS) cyber-attack on the servers of Dyn. (Dyn is a company that administers the majority of the Internet's Domain Name System (DNS) infrastructure [3].) This attack brought down a large number of websites in Europe and the United States, notably Twitter, Reddit, and CNN. In this incident, about 100,000 IoT devices were compromised.

Besides Mirai, different malware families that exclusively target IoT devices have been released, e.g., Bashlite, Tsunami, Hide and Seek, BrickerBot, Luabot, and Hajime. Each IoT botnet family has distinctive behaviour that represents the family's characteristics and features. However, generally, IoT botnet malware

* Corresponding author.
E-mail address: dalqattan@tvtc.gov.sa (D. Alqattan).

https://doi.org/10.1016/j.hcc.2024.100230

families carry out their attack tasks in three main phases. First, the botnet is formed (through scanning and spreading), then it is commanded and controlled (CnC), and finally, the attack is carried out. The first and second phases are considered the early stages of the IoT botnet before launching the attack, whereas the last stage is the late stage of the IoT botnet. In the botnet formation process, the bot master scans vulnerable IoT devices and infects them with bot malware, which then propagates and spreads across the network to make other IoT devices infected with bot malware. Then, the commands given to a botnet to control it by attackers are either to retrieve information from a bot or to conduct the intended attack (e.g., DDoS). Lastly, the intended attack is conducted towards the target victim in order to damage the confidentiality, integrity, or availability of the network [4,5], leading to a failure that affects some of the smart city nodes with catastrophic consequences and alters the normal behaviour of smart cities. The successes in IoT botnet-based attacks emphasise the importance of implementing robust IoT botnet detection methods.

Smart city security has received a lot of attention in recent years for protecting smart cities from IoT botnet-based attacks [6]. Machine learning (ML) and deep learning (DL) have been the dominant choices in the development of intrusion detection systems (IDS) for the monitoring and detection of IoT botnets by analysing network communication patterns [1]. Several studies are available on ML-based IoT botnet detection that aims to detect late-stage IoT botnets (e.g., DDoS) [7,8]. They detect the abnormalities in the network communication patterns of IoT devices after attacks have already been launched. However, the early stages need more attention so that the IoT botnet can be detected before launching any IoT botnet-based attack. Only a few studies addressed the detection of early stages of IoT botnets (discussed in Section 2) using ML and DL approaches to differentiate between several early stages of IoT botnets as a multi-classification problem. For instance, Hegde et al. [9] examined botnet detection via experiments with supervised ML and DL classifiers. Alzahrani et al. [10] aimed to develop a multi-classification Neural Network model using FastGRNN to detect the IoT botnet in a short time. Abdalgawad et al. [11] analyse network traffic to identify IoT botnet behaviour using Adversarial Auto-encoders (AAE) and Bidirectional Generative Adversarial Networks (BiGAN). Wazzan et al. [12] apply Cross CNN LSTM to identify the propagation of IoT botnets.

These ML and DL-based IoT botnet detection studies, despite achieving high accuracy, have several shortcomings:

- Although these studies rely on detecting the early-stage IoT botnet from network communication patterns, they lack in the realm of failure detection and recovery, where early-stage IoT botnets should be modelled as severe fault conditions that can be observed in the network communication of smart city systems.
- These studies do not examine the idea that the early stages (Scan and CnC) could occur simultaneously. It is possible that the reported effects on network communication of smart city systems have more than one origin (Scan and CnC). In order to take the most appropriate action, it is crucial to distinguish between these two stages.
- Since these studies are based on deep machine learning models for multi-class classification of IoT botnet stages, these models are large in size and demand a lot of resources to train. Therefore, the models take a long time to construct and update, which could make smart cities vulnerable to a zero-day attack.
- A significant delay occurs in detecting the early stages of IoT botnets. Consequently, the attacker may have the ability to launch an attack on any smart city node. Normal behaviour in smart cities will be interrupted as a result.
- The architecture that these studies suggest has a high rate of false negatives; as a result the attacks are missed and remain undetected, which makes IoT-based smart cities vulnerable and not well protected.

To address the mentioned shortcomings, this paper answers the following research questions: (**RQ1.**) How early stages of the IoT botnet can be modelled as faults in network communication in smart city systems? (**RQ2.**) How can a DL model be trained to differentiate between the characteristics of these faults in a short time to reduce false negative rates without significant detection delay?

To answer these research questions, we investigate how to distinguish the two early stages of the IoT botnet (Scan and CnC) as faults in smart city network communication that affect the network's normal behaviour (benign communication) by examining network traffic characteristics travelling during normal function and IoT botnet behaviour in smart city systems using a supervised Neural network (NN). We first modelled the early stages of the IoT botnet as fault conditions in the network communication of smart city systems (RQ1) and then used the fault characteristics to train a DL model for IoT botnet detection. We propose Modular Neural Networks (MNN) as a DL approach for early-stage IoT botnet detection that integrates modularity into the neural network design. Using this approach, we simplify the detection of IoT botnet stages by treating each stage as a separate and specialised detection module. Thus, we construct simple, fast, and efficient modules instead of a single large and complicated model as in a monolithic model. Fig. 1 illustrate the differences between training monolithic NN and MNN for multi-classification detection problems. The construction of the specialist modules allows for differentiation between the early stages with low false-negative rates. Another advantage of MNN over monolithic NN is that MNN allows the parallel computation of the neural network on multi-core computing servers. Thus, parallel training and detection in multi-access edge computing (MEC) servers can be implemented. These MEC servers are directly connected to IoT devices to avoid detection delays (RQ2). Fig. 2 shows MEC servers deployed in smart cities, where each domain of IoT devices is connected to the same MEC server. The MEC servers are connected to the cloud server through the core network.

In summary, this paper makes the following contributions:

- We model the early stages of the IoT botnet (Scan and/or CnC) as faults that cause abnormalities in network communication traffic in smart cities.
- We propose an **e**dge-based **d**etection system for early-stage **I**oT botne**t** (EDIT) in smart cities using modular neural network. In this paper, we provide the first effort at using a modular neural network (MNN) to identify communication traffic faults caused by IoT botnets in their early stages. For fault recognition (Scan and/or CnC), a quick training and detection process, and accurate detection, the MNN technique breaks the detection task into separate parallel binary classification tasks that are run in parallel on the MEC server.
- Through a series of experiments on publicly accessible datasets, we compared our MNN-based approach to many state-of-the-art methods, demonstrating the extent to which the detection rate was improved while still maintaining a fast training and detection speed. We found that the MNN approach had a false-negative rate of zero.

The remainder of the paper is organised as follows: Section 2 introduces the related work. In Section 3, early-stage IoT botnets (Scan and CnC) are modelled, and the characteristics of network communication during these stages are described. Section 4

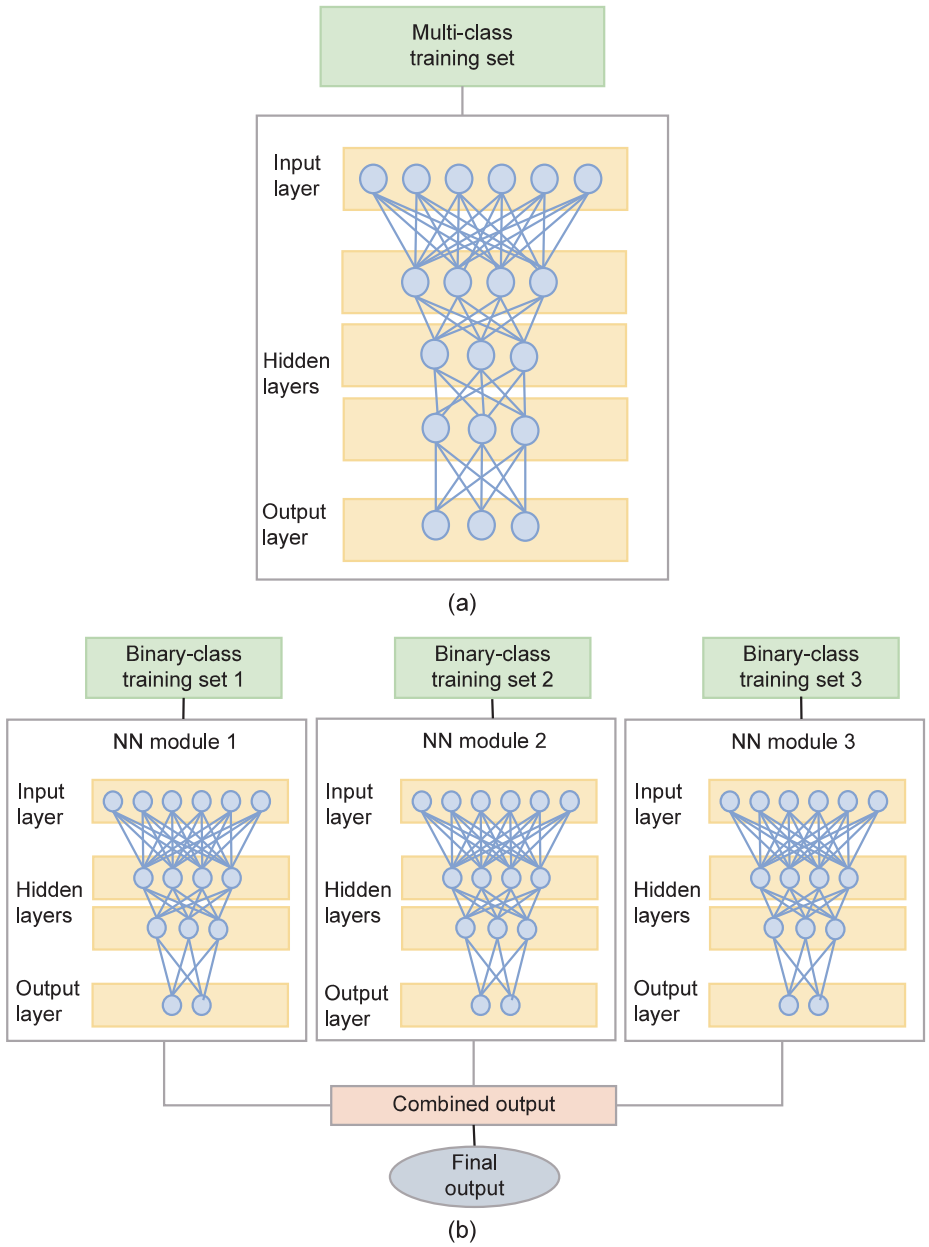

Fig. 1. (a) Monolithic NN for multi-classification detection problem. (b) MNN for multi-classification detection problem.

presents the proposed EDIT system and describes its components that depend on the MNN approach. The evaluation methodology and the experimental setting are described in Section 5. In Section 6, the evaluation of EDIT results is discussed in terms of effectiveness and efficiency. Section 7 concludes that our EDIT system that deploys the MNN approach is able to take advantage of the parallelism of the modular approach to improve both the speed and accuracy of IoT botnet detection.

2. Related works

A few researchers used a range of ML/DL approaches to detect IoT botnets at an early stage. In their efforts to increase the performance of IoT botnet detection using ML/DL algorithms, researchers have developed datasets or used publicly available datasets. There is also some research which applies similar principles of MNN in terms of task decomposition to detect the early and late stages of IoT botnets using a multi-model-based ML/DL approach. In this section, we will first discuss the early-stage IoT botnet detection research and then the research that applies the multi-model ML approach to train and detect IoT botnets.

2.1. Early-stage IoT botnet detection

In [13], the authors obtained a collection of labelled IoT behavioural data, which comprised both legitimate IoT network traffic and botnet malicious traffic, using an IoT medium-sized network design (83 IoT devices). To gather data, three well-known botnet malware (Mirai, BashLite and Torii) infections and connections with CnC nodes were employed. The dataset collected, called MedBIoT, is used later by researchers. A binary ML classification model is applied to divide the acquired data into two separate categories (e.g., benign and malware). The authors used Random Forest (RF), K-Nearest Neighbours (K-NN), Decision Tree (DT), as well as Support Vector Machine (SVM) algorithms. RF, K-NN, DT, and DT models showed an accuracy of 0.9532 (for

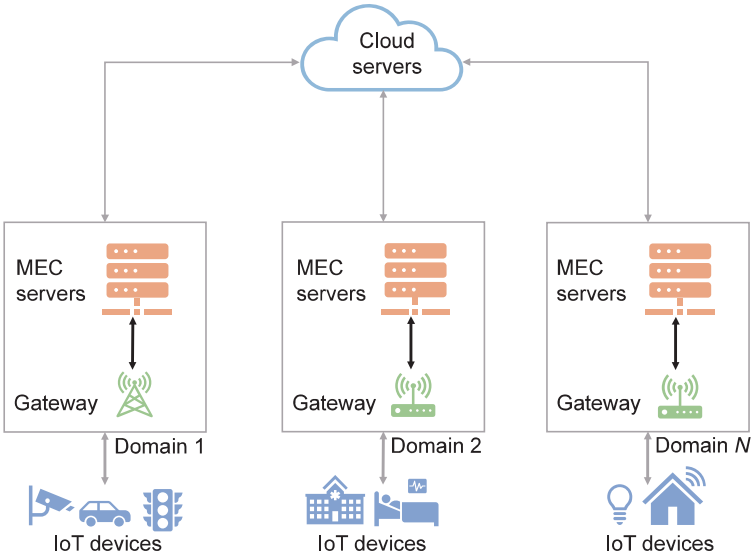

Fig. 2. Multi-access edge servers in smart cities.

RF), 0.9025 (for KNN), and 0.9315 (for DT). The result of the SVM model was not mentioned because it showed much less accuracy than other models.

The authors of [14] conducted different experiments. The relevant experiment to our work is the classification of malicious packets as a binary task for botnet detection. The authors investigated and analysed three recurrent deep-learning methods.: FastGRNN, LSTM, and GRU. The results of the experiment have an area under curve score of 99.9% for MedBIoT and 99.75% for the Kitsune dataset.

Using openly available datasets (IoT23), the research in [9] examined botnet detection via experiments with supervised ML and DL classifiers. Using this dataset, the researchers were able to examine the effectiveness of the classifiers on a dataset that included both benign and malicious data. The accuracy rate was 85.2%, according to the data. As shown in this study, adding more benign network traffic packets to the IoT training dataset increases the performance of classifiers by up to 99.9%, as shown in this study.

The study [10] aimed to develop a multi-class classification NN model that could be trained fast, identify changes in real-time and be accurate. FastGRNN beat the other models in the studies on the MedBIoT and RGU datasets by lowering training and detection time while keeping a high F1 score. Using FastGRNN, the whole test set was identified in 29 s. Furthermore, for MedBIoT multi-classification, the model achieved competitive F1 scores of 99.99% and 99.04% for RGU multi-classification.

In [15], distinct ML and DL methods were tested using various datasets (N-BaIoT, IoT-23, Kitsune, MedBIoT). Among these models were Logistic Regression, K-Nearest Neighbours, naive Bayes Decision Trees, and Random Forests. The researchers discovered that the random forest technique was the most accurate and fastest of all four datasets evaluated. An identical experiment was performed using MLPN and LSTM. In terms of accuracy, LSTM surpasses Multi-Layer Perception Network (MLPN) with 99.9%, 99.8, 91.1% for N-BaIoT, IoT-23, and Kitsune datasets, respectively. However, MLPN required significantly less training and detection time.

The study in [11] demonstrates that network data analysis may be utilised to identify malicious behaviour when using generative DL algorithms like Adversarial Auto-encoders (AAE) and Bidirectional Generative Adversarial Networks (BiGAN). The IoT-23 dataset was used to construct generative models to detect distinct threats connected to early-stage IoT botnets. As the research shows, generative models won out over traditional ML approaches. Both the AAE and the BiGAN-based models attained F1-scores of 99%.

An IoT botnet detection method is presented in the research [12]. The botnet's early stages were studied, and a baseline ML model was developed for detection. Using a combination of DL models for a Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM), the researchers developed an effective detection technique called Cross CNN LSTM to identify the IoT botnet. Studies have shown that this model is 99.7% accurate, outperforming other ML techniques using the MedBIoT dataset.

In study [16], one-class KNN was utilised to detect malicious IoT botnets. The main aim was to address the computing power in IoT devices by developing a lightweight IoT botnet detector using ML. The performance of ML classifiers was enhanced by employing filter and wrapper techniques to choose the appropriate features. Using the filter and wrapper techniques, various datasets were able to reduce feature space. According to the findings, the authors' proposed method was effective in detecting an IoT botnet with an F1-score ranging from 98 to 99% for various IoT botnet datasets.

2.2. Multi-model ML for IoT botnet detection

In [17], the proposed system for IoT botnet detection was divided into subsystems. Each subsystem is responsible for detecting one kind of IoT botnet malicious activity sequentially. A classifier is trained to differentiate between malicious and benign behaviour for each kind. A classifier decides if the data input is malicious or not. If it is malicious, the system raises an alarm. Otherwise, the output is passed to the next classifier. The new classifier takes the output of the previous classifier if the output is not a malicious activity, and makes a decision again. This continues until one of the classifiers raises an alarm or until the last classifier decides that the data input is benign. An artificial neural network as the classifier is trained and used for each kind of specific attack. According to these study results, all of the classifier accuracy is up to 99%.

The authors in [17] continue their work of [18] where they apply the feature selection approach, which makes the system less computationally expensive. They also deployed the same system with a "hybrid" classification, in which each system is

trained with a different algorithm. The results show that hybrid classification can also achieve high accuracy.

In the study [19], the authors proposed a two-fold ML method to detect IoT botnets. The detection task was decomposed into two parts. The first step is scanning vulnerable devices for activity detection. In this step, a model was trained to classify normal and scan behaviour. The second step is for detecting DDoS activity, and another model was trained using another dataset of normal and DDoS activity. The results show that the proposed method can achieve an overall accuracy of 98%. However, the study only shows the overall accuracy and does not show how the final detection decision of the two classifiers could be combined.

The study [20] proposed a collaborative ML model for early-stage botnet activity detection. The three heterogeneous feature sets are collected from network flows, system logs, and system resources. The detection task was decomposed between three ML models. Therefore, a separate ML model was trained for each input feature set to differentiate malicious and benign behaviour during the early stages of the IoT botnet life cycle. Then, all the results from the 3 classifiers are integrated to make the final decision by a majority of votes. The result shows that the proposed method gives roughly 99% accuracy.

The author of [21] introduced the EDIMA, a home network-focused IoT botnet detection tool. It uses a two-step process to identify bots communicating with an edge gateway: first, it detects scanning activity in the gateway's overall traffic using machine learning; and then it uses bot-CnC communication traffic patterns to identify specific bots using Autocorrelation Function (ACF). The components of EDIMA are as follows: a parser for network traffic, an extractor for features, a machine learning-based bot detection, a policy engine, a model function for machine learning, and a PCAP database for malware. Performance evaluation results using the testbed configuration with real-world IoT malware traffic and other public IoT datasets show that EDIMA has very low false positive rates for bot scanning and bot-CnC traffic detection.

The quantities of computation that can be traded off vary depending on the multi-task technique applied in these studies. In [17–19,21], sequential multi-task models are implemented. As a result, they are not readily capable of being parallelised to decrease the duration of training. Conversely, the multi-task machine learning (ML) model suggested by [20] can be executed in parallel. Nevertheless, it incurs substantial resource consumption, including high memory and CPU utilisation, as it simultaneously gathers three distinct feature sets that have characteristics different from each other.

On the other hand, despite the fact that the multi-task model method was used in the preceding studies, they do not reflect MNN-based detection system designs in the real sense of modularity, in which the system design consists of a number of specialised modules. These modules should display the following properties [22]:

- The modules have specialised computational architectures to recognise and respond to subsets of the total detection task.
- Each module differentiates one network behaviour from another, independently of the other modules.
- Modules have a simpler architecture than the system as a whole.
- The basic module outputs are combined to achieve the complex overall system response.

In contrast to prior research, our MNN architecture breaks down detection into numerous binary-classification tasks, each of which can be handled by a dedicated sub-neural network. On the MEC server, the sub-neural network are set up in parallel, and their predictions are combined together to achieve the task with high performance and low resource consumption.

3. Early-stage IoT botnet model

IoT botnets' principal targets are IoT devices. This form of botnet takes advantage of the fact that IoT devices generally have limited system resources and skimp on security. In general, botnets are conducted in three phases. First, the botnet formation; second, commanding the botnet; and third, conducting the attack. During the botnet formation, the bot master scans vulnerable IoT devices and infects them with bot malware. The bot is then listed on the command and control (CnC) server, and the bot malware spreads through the network, infecting other IoT devices with bot malware to make a botnet. In the second phase, the commands given to a botnet by attackers are mainly of two types: message commands and malicious commands. The message command is to retrieve information from the bot. Whereas the malicious commands are issued to a bot to act maliciously towards the target victim (any network node) and conduct the intended attack. These two phases are the early stages of IoT botnet tasks. In the third phase, the botnet performs the intended attack (e.g., DDoS) in order to damage the confidentiality, integrity, or availability of the network [4]

The three phases mentioned earlier are typical for IoT botnets, and they may be found in many different families of IoT botnets. Here, we will highlight the features and characteristics of one of the IoT botnet families, Mirai, that is being analysed in this study.

Mirai is a malware family that infects IoT devices before launching DDoS attack. As shown in Fig. 3, we provide a model of its fundamentals as well as how it propagates.

According to authors in [23], the Mirai botnet performs seven tasks (also shown in Fig. 3):

(1) **Scan.** The initial bot used in an attack will use Telnet's TCP ports 23 and 2323 to send TCP SYN scanners to random IPv4 addresses. The bot will try to log into a vulnerable IoT device through Telnet with a variety of user names and passwords if it finds one.
(2) **Report.** After the bot has shown its authenticity for the first time; it will communicate the IP address of the vulnerable IoT device together with the credentials it was given to a report server.
(3) **Dispatch.** IP addresses of the vulnerable IoT devices, together with the credentials, are dispatched to the loader server. The load server builds an application to log into the devices in order to determine the sort of operating system and CPU architecture of the device. Then, the loader prepares the malware that can be run on the vulnerable device.
(4) **Load.** Infected IoT devices download and run malware sent by the loader server. The malware has been customised for the architecture of the infected device. Infected IoT devices will execute scans to look for additional vulnerable devices and will take commands from a command and control server. Therefore, the botnet is composed of a variety of Internet of Things devices that have been compromised.
(5) **Instruct.** Through the CnC server, the attacker, gives instructions to the botnet so that it may carry out the attack as planned.
(6) **Command and Control(CnC).** The command to carry out the predetermined attack is relayed from the CnC server to the compromised IoT devices that are part of the botnet.
(7) **Attack.** During this phase of the mission, the botnet will initiate an attack against the target.

Early stages of IoT botnet tasks (Scan, CnC) on smart cities often cause unusual network traffic that puts the network communication channel in a faulty state. The faulty state can be

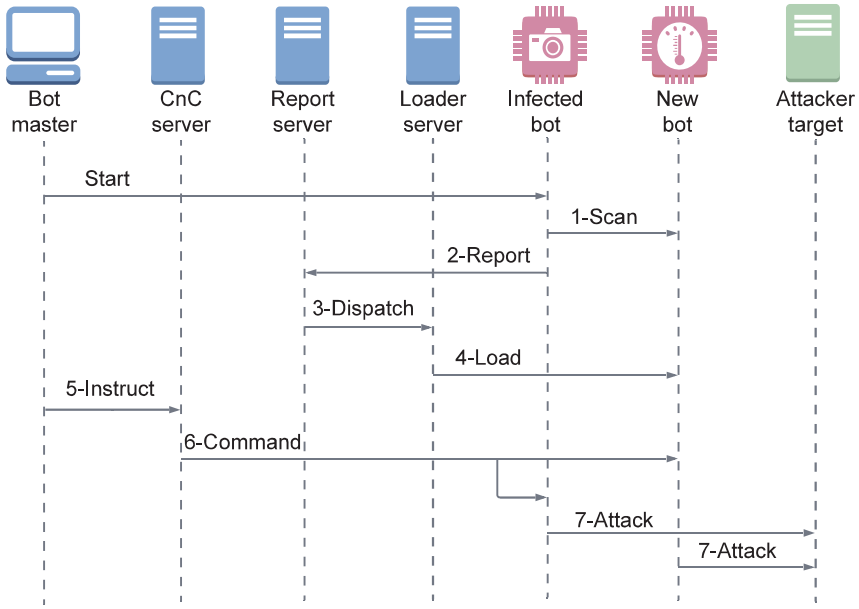

Fig. 3. Mirai tasks.

persistent and correlated with abrupt changes in traffic-related features. A network packet is a fundamental unit of network traffic that passes from sender to receiver through a network. In addition to the actual data, which is known as the payload, packets also contain the header and the trailer. The choice of a set of network communication features that can be used to find IoT botnets is one of the biggest problems researchers have to deal with. By analysing the network packets that traverse during normal and abnormal (IoT botnet) communication and searching for any anomalous behaviour patterns, we are able to determine the communication behaviour features that contribute to the faulty state. To extract the features of the communication behaviour of IoT botnets, it is necessary to sniff packets originating from the network. While monitoring network traffic, we are focusing exclusively on the packet headers and ignoring the packet's payload and trailer. Packets with the same destination IP, source IP, destination port, source port, and protocol are aggregated as a flow. The header of each packet in a flow can be used to figure out statistics about the packets. Abrupt changes refer to significant changes in the network packet feature statistics.

3.1. Characteristics of the network communication faulty state due to early-stage IoT botnet

In this study, detection of the early-stage IoT botnet depends on the characteristics and features of network communication traffic that is sent or received by the infected IoT devices during Mirai botnet execution. The abnormal network traffic from IoT devices is our main emphasis. Accurate analysis of the early stages of a Mirai botnet on IoT devices is feasible by continuously gathering network traffic feature statistics such as protocol type, packet size, packet duration, and other associated flow-based statistics. While performing the scan task and the command task, we collect data from the network and then extract statistical characteristics to use as input to a deep learning algorithm for model training.

In this section, we describe the possible network-based manifestations of the Mirai botnet's scan task and command task:

3.1.1. Scan task

The majority of IoT botnets employ spreading tactics that are strikingly similar to those employed by worms. They will automatically conduct scans and brute-force attacks after infecting a device in an attempt to infect other machines [24]. Scan task conducted by the Mirai botnet has three characteristics: a TCP port scan target, random stateless IP scanning, and weighted dictionary credential access.

Mirai starts by scanning a set of predefined TCP ports. The code fragment associated with malware scan capabilities reveals that 90% of scans target TCP port 23 and 10% target port 2323 [25].

The scanning strategies used by bots are completely random and stateless. The IP addresses of the destination are generated at random and assigned to the scan packets. The same value is also assigned to the field corresponding to the TCP packet sequence number [24,25]. This strategy is called the random stateless scanning strategy. In this strategy, the botnet will keep scanning even if the vulnerable IoT device does not react. When the bot receives a response from the vulnerable IoT device, it first sends a rest packet and attempts to compromise the device [3].

After identifying a vulnerable IoT device, the Mirai botnet employs a weighted dictionary attack, also known as brute force, to get access to it. In a weighted dictionary attack, the bot is outfitted with a database of commonly used passwords. Random credentials are selected and tested against the vulnerable IoT device [3].

It is possible that the aforementioned scan behaviour features represent anomalous network traffic rather than benign network traffic. The packet measurements may or may not readily reveal these anomalies. Consequently, flow statistical data accurately characterise scan activity more precisely. First, a clear indicator of the abnormal behaviour is the increase in the number of flows entering the network as a result of a large number of unique source IP addresses and scanning ports. Second, to scan as many devices as possible, malware frequently sends a small number of scanning packets to the same IP address. Thirdly, throughout the scanning process, a bot will randomly establish a huge number of TCP connections to IP addresses, the vast majority of which will be ignored and hence remain partially open. Consequently, there will be a large number of TCP half-open connections if we examine their frequency. In addition, the length of malware

scanning packets is often shorter than that of normal benign TCP packets. Scan packets feature fewer transmission latency than packets from benign IoT devices.

3.1.2. Command and control (CnC) task

Once an IoT device has been infected, it transforms into a bot, registers with the CnC server, and joins the botnet. The bot is awaiting commands from the CnC server to control its behaviour. Each botnet family uses its own protocol messages and communication mechanism to carry out this command [24].

The communication mechanism used by attackers consists mainly of two types of commands: message commands to retrieve information from the bot and malicious commands that are issued to a bot to act maliciously towards the target victim (any network component) and conduct the intended attack (e.g., DDoS) [4].

Mirai has a unique protocol based on hexadecimal messages for its communication mechanism. The bot uses a handshake process to communicate with the botnet's command and control server; the first message is a registration packet with the hexadecimal value 0 × 00,0 × 00,0 × 00,0 × 01. To initiate the heartbeat procedure, the botnet CnC receives a string representing the system architecture after 10 s and immediately responds with an acknowledge message by delivering a pulse packet containing the hexadecimal string 0 × 00 0 × 00 [24]. The CnC server then begins sending malicious commands to the infected IoT device. The bot will execute the attack coded inside the packet's seventh byte (the payload) after it has been received. In addition, bytes 9–13 represent the Internet Protocol address of the victim server. The attack's duration is encoded in bytes 3–6 [26].

Various sorts of traffic abnormalities can be identified by analysing the aforementioned command behaviour characteristics. Each bot establishes a TCP connection with the CnC server and transmits and receives periodic TCP heartbeat messages. The timing of communication exchanges is periodic, according to [21]. Moreover, when an IoT device communicates with a public server IP address, the server sends packets back to the relevant IoT device. Therefore, it is possible to keep track of how frequently an IoT device sends and receives packets from the public server. The size of the payload in packets travelling between IoT devices and the CnC servers is also statistically observable and can be more or less than the size of the payload in benign traffic. The payloads are presumably encrypted, although certain characteristics can be derived from them to determine the CnC connection behaviour. Therefore, we solely evaluate characteristics extracted from packet headers and completely disregard payloads.

3.2. Definition of the network communication faulty state due to early-stage IoT botnet

From the discussion in the previous section, we can define the normal and abnormal behaviour during the early-stage IoT botnet. The main objective of this work is to develop a technique for distinguishing between benign and early-stage IoT botnets (Scan, CnC) in a smart cities domain. From the graph theory, the domain is represented as a graph $G = (V, E)$, where V is the set of nodes and E is the set of directed communication channels. V is made up of m nodes $\{v_1, \ldots, v_m\} \cup G$. These nodes represent IoT devices, and all nodes are connected to these devices in the edge, cloud and the internet through an edge gateway G. Two nodes $v_i, v_j \in V$ exchange messages through communication channels $e_{i,j} \in E$. Each $v_i \in V$ communicates directly to the gateway G through communication channels $e_{i,g}$, where $e_{i,g} \in E$, $\forall v_i \in v_1, \ldots, v_m$.

The state of each channel $e_{i,j}$ denoted by $S^{e_{i,j}}$, is defined over a set of state features, f_i. Using the snapshot of the state of the channel, we can derive a set of measurements that describe the channel state and performance metrics (e.g., delay) that could contribute to the channel's faulty state. Thus channel $e_{i,j}$ state is given by

$$S^{e_{i,j}} = \{f_1, f_2, \ldots, f_k\} \tag{1}$$

For each state feature f_i of $S^{e_{i,j}}$, we denote by $min(f_i)_t$ and $max(f_i)_t$, the expected minimum and maximum limits, respectively, of the value of f_i, at a time unit t. Using these bounds, we can now define the normality and abnormality of the communication channel as follows:

Definition 1 (*Normal*)**.** A state $S_t^{e_{i,j}}$ of channel $e_{i,j}$ at time t is *normal* $\iff$

$$\forall f_i, f_i \in [min(f_i)_t, max(f_i)_t] \tag{2}$$

In other words, the state of a channel is normal if all the state features are within the expected limits. We assume the channel between the node and the gateway always exists. On the contrary, in an abnormal state, such conditions may not hold, as explained in Definition 2.

Definition 2 (*Abnormal*)**.** A state $S_t^{e_{i,j}}$ of channel $e_{i,j}$ at time t is *abnormal* $\iff$

$$\exists f_i, f_i < min(f_i)_t \vee f_i > max(f_i)_t \tag{3}$$

By definition, an abnormal state may include features outside the minimum and maximum limits.

Following Definitions 1 and 2, we say that the domain state is normal if $\forall e_{i,j} \in E$ are normal (Benign) due to normal execution $\epsilon, \forall v_i \in V$. Similarly, we characterised a domain state as abnormal if $\exists e_{i,j} \in E$ is abnormal as a result of abnormal execution $\epsilon \in [Scan, CnC]$ from Scan and Command and Control execution of a node.

3.3. Problem definition of the network communication faulty state detection

Detection systems can usually be configured to network communication faulty state by monitoring for simple thresholds and limit boundaries. However, each stage in the IoT botnet reveals different actions; thus, the scan and CnC behaviour of the IoT botnet can affect many features of the communication channels. Besides the differentiation between normal (Benign) behaviour and abnormal behaviour (Scan, CnC), discrimination between abnormal behaviours (Scan and CnC) is very critical to take suitable action. Thus, the tracking of these behaviours is not simple, and a probabilistic discrimination function based on expected network communication behaviour and pattern should be deployed.

Problem 1. Given a state $S_t^{e_{i,j}}$ of channel $e_{i,j}$ in an execution ϵ at time t, a discrimination function f can accurately classifies the state $S_t^{e_{i,j}}$ as Benign, Scan or CnC.

3.4. Communication channel state features

A similar real-time smart cities domain testbed has been developed consisting of real and virtual IoT devices. The features of the communication channel state in the domain have been extracted during benign, scan and CnC behaviour of Mirai botnet. In this study, we will utilise these two data sets (IoT23 and MedBIot) that contain IoT botnet statistical network packet features.

Table 1 exhibits the feature extracted by [27] and from the header fields of network packets while observing the benign, scan, and CnC communication behaviour. This dataset (called IoT23) consists of 20 features.

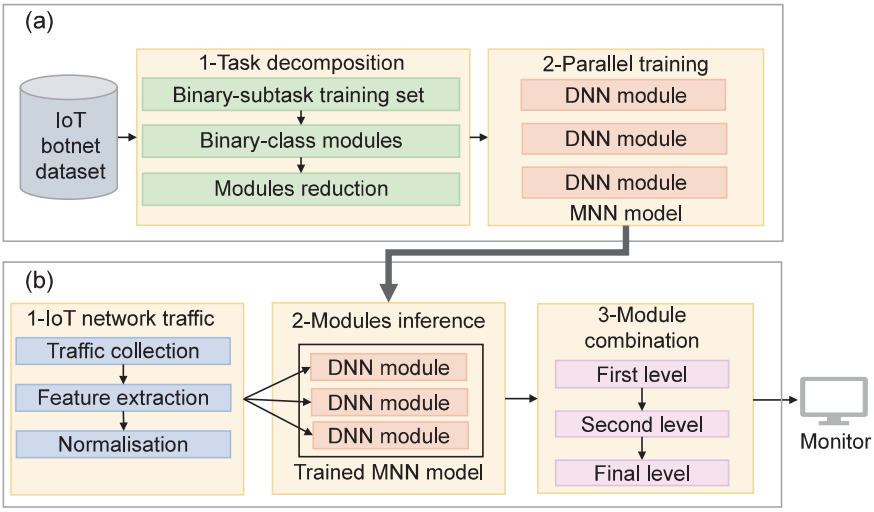

Fig. 4. Architecture of EDIT system.

Table 1
Feature description of IoT23 dataset.

No.	Feature
1	Flow start time
2	Unique ID
3	Source IP address
4	Source port
5	Destination IP address
6	Destination port
7	Transaction protocol
8	Service http, ftp, smtp, ssh, dns, etc.
9	Total duration of flow
10	Number of payload bytes from source
11	Number of payload bytes from destination
12	Connection state
13	Connection source locally or remotely
14	Connection destination locally or remotely
15	Missing bytes during transaction
16	History of source packets
17	Number of packets from source
18	Number of IP level bytes from source
19	Number of packets from destination
20	Number of IP level bytes from destination

Another feature set is collected for IoT botnet behaviour called MedBIoT. According to [13], MedBIoT is a 100-feature dataset. These features are divided into four categories: 1- traffic originating from the same MAC and IP (Host-MAC and IP features), 2- traffic between specific hosts (channel features), 3- traffic between specific hosts, including ports (socket features), and the 4-time interval between packets in channel communication (network jitter features). The number of packets, average packet size, and standard deviation are all computed for each broad classification. In addition to the standard statistical measures of frequency, mean, and standard deviation, the correlation coefficient (PCC) of packet size, radius, covariance, and magnitude for channel and socket categories were also derived. Table 2 shows the description of these features.

4. EDIT: Edge-based detection of early-stages IoT botnet system

This section describes the general architecture of the proposed EDIT system. The architecture shown in Fig. 4 relies on two components: (a) *MNN-based model constructor* and (b) *IoT botnet detector* at the edge.

The training task is the responsibility of the constructor. The training task is subdivided into three binary sub-tasks: (1) Benign and Scan classification, (2) Benign and CnC classification and (3) Scan and CnC classification. The MEC server stores flow-based statistics data from IoT network traffic that contains the features of benign, scan, and CnC behaviour and uses this data to train a model. After the neural network modules are trained, the detector at the MEC server uses the trained modules. The detector is responsible for detecting IoT botnets in their early stages by monitoring the group of IoT devices linked to the same MEC server. This section elaborates on the proposed EDIT system and its components.

4.1. MNN-based model constructor

For the MNN-based model constructor, pre-collected flow-based statistics data from the IoT network traffic that represents the characteristics of benign, scan and CnC behaviour is stored and used as an IoT botnet training dataset. To deploy the modularity on this detection task, the constructor first applies task decomposition (as explained in [28]) and then performs the training job.

4.1.1. Task decomposition

The IoT botnet detection task is divided into sub-tasks (modules) according to IoT botnet behaviours. Our system aims to differentiate between three types of behaviour classes, namely benign, scan, and CnC. Each module is responsible for detecting a class of botnet behaviour against another class:

Lets C_i for $i = 0, 1, \ldots, 2$ be the classes of IoT botnet behaviours (Benign, Scan, and CnC).

$$C = \{C_0, C_1, C_2\} \tag{4}$$

We call this a 3-Class IoT botnet behaviour detection task. The training dataset for each class DS_0, DS_1 and DS_2 for Benign, Scan, and CnC respectively can be written as

$$DS_i = \{(X_i, Y_i), \} i = 0, 1, 2 \tag{5}$$

where X_i and Y_i are feature vectors and class labels of the samples specific to the class C_i.

The scope and organisation of publicly accessible datasets vary. It is thus essential to preprocess and aggregate the data into the proper format for ML/DL. Therefore, the min–max approach was used to normalise the features in all sub-datasets DS_0,

Table 2
Feature description of MedBIoT dataset.

Categories	Features	Numbers
Host-MAC&IP	Packet count, mean, variance, magnitude,	15
Channel	Packet count, mean, variance, magnitude, radius, covariance, and correlation	15
Network Jitter	Packet count, mean and variance of packet jitter in channel	35
Socket	Packet count, mean, variance, magnitude, radius, covariance and correlation	35
Total	–	100

Table 3
The modules and the corresponding dataset.

Classes	Benign vs. Scan	Benign vs. CnC	Scan vs. CnC
Module	M_{01}	M_{02}	M_{12}
Training set T_{ij}	$\{X_0,\mathbf{1}\}\cup\{X_1,\mathbf{0}\}$	$\{X_0,\mathbf{1}\}\cup\{X_2,\mathbf{0}\}$	$\{X_1,\mathbf{1}\}\cup\{X_2,\mathbf{0}\}$

DS_1 and DS_2 ensuring that no one feature dominates the model training stage. This is accomplished by the use of the following equation:

$$x_{normalised} = \frac{x - min(x)}{max(x) - min(x)} \tag{6}$$

That is necessary to obtain a normalised dataset of DS_0, DS_1 and DS_2. Therefore, we can write the IoT botnet training dataset T as a union of sub-tasks class training datasets: DS_0, DS_1, and DS_2 as

$$T = \cup_{i=0}^{2}(X_i, Y_i) \tag{7}$$

For MNN, we decompose the 3-Class task into M binary-class modules. Each binary-class M_{ij} module is trained to classify a class C_i from a class C_j for $\forall\ i$ and $j \in \{0, 1, 2\}$ and $i \neq j$. Such a decomposition produces $M = 6$ binary-class modules ($M01, M02, M10, M12, M20, M21$). The training dataset for binary-class module M_{ij} that differentiates a class C_i from a class C_j is

$$T_{ij} = \{(X_i, 1)\} \cup \{(X_j, 0)\} \tag{8}$$

which means all feature vectors of class C_i are labelled as 1, and feature vectors of class C_j are labelled as 0.

In this decomposition, for each binary classification task for classes (C_i) and (C_j), there are redundant modules, i.e., either module M_{ij} or module M_{ji} is sufficient for classifying class (C_i) from class (C_j). That is, training sets T_{ij} and T_{ji} are complementary to each other:

$$T_{ij} = \{(X_i, 1)\} \cup \{(X_j, 0)\} \quad \text{complements} \quad T_{ji} = \{(X_j, 1)\} \cup \{(X_i, 0)\} \tag{9}$$

In such case, we reduce the number of sub-tasks by merely using an inverse unit (INV) on the output of M_{ij} to derive the output of M_{ji} as

$$M_{ji} = INV(M_{ij}) \tag{10}$$

This will lead to the reduction of the number of modules to $\binom{K}{2}$.

Therefore, the number of modules is reduced to 3. (M_{01}, M_{02}, M_{12}) as the result of modules (M_{10}, M_{20}, M_{21}) can be obtained from modules (M_{01}, M_{02}, M_{12}) using INV units. Ultimately, there are 3 modules (M_{01}, M_{02}, M_{12}) and 3 training datasets (T_{01}, T_{02}, T_{12}) for Benign against Scan, Benign against CnC, and Scan against CnC, respectively.

Table 3 shows the modules M_{ij} and the corresponding module training dataset T_{ij}.

4.1.2. DNN modules parallel training

For training the modules (M_{01}, M_{02}, M_{12}), we have applied a simple feed-forward deep neural network (DNN). The architecture of the DNN model included an input layer, densely interconnected hidden layers,dropout layers, and an output layer. The number of neurons in the input layer (n) is proportional to the amount of training data features that accurately characterise a single network traffic packet sample x. The number of densely interconnected hidden layers h and the number of neurons in each of them is often determined by testing several sets of values in order to develop the best DNN architecture that delivers the maximum classification performance.

Each input neuron (feature) $a_1^0, a_2^0 \ldots a_n^0$ of network traffic packet sample x in the input layer 0 are all connected to different hidden neurons $a_1^1, a_2^1 \ldots a_z^1$ in the first hidden layer 1. Then, each hidden neuron a^i in a hidden layer i is connected to each hidden neuron a^j in the next hidden layer j, where $j = i + 1$. The connection between each neuron a^i to the next neuron a^j has a connection weight W^i.

To activate a neuron a^j in the next layer j, we first aggregate the previous neurons connected to it and their weights as follows:

$$\theta_j = \sum(a^i * W^i + b^i) \tag{11}$$

where a^i is a previous neuron in layer i, W^i is a connection weight, and b^i is bias vector. Then, to obtain the output of each neuron a^j in the next hidden layer j, a rectified linear unit (ReLU) activation function is used.

$$a^j = \begin{cases} 0, & \theta_j < 0 \\ \theta_j, & \theta_j \geq 0 \end{cases} \tag{12}$$

The output of each neuron a^i in the last hidden layer for class i is transformed using the softmax function. The softmax activation function inputs a vector of neural network outputs and returns a vector of probabilities for each class label, $\tilde{y}$. The following expression represents the equation for the softmax function:

$$\sigma(a^i) = \frac{e^{a^i}}{\sum_{j=1}^{N} e^{a^j}} \tag{13}$$

where N is the number of classes in the multi-class classifier, in our case it is 3.

The three modules (M_{01}, M_{02}, M_{12}) forms the MNN model. In the MNN model, the three modules are trained in parallel using an MEC server that is close to the IoT devices.

The overall model construction step is summarised in Algorithm 1.

Algorithm 1: MNN-based modules constructor

Input : C_i - list of classes, $i = 1, \ldots, K$
X_s - feature vector of all samples belongs to class C_i,
Output: M_{ij} - Binary-classification Trained module for class C_i against class C_j,

```
for all C_i and C_j , i < j do in parallel
    // label the training set T_ij with C_i label=1 and C_j label=0
    T_ij = {(X_i, 1), (X_j, 0)}
    //Train a module M_ij
    M_ij ← Train (T_ij)
end
```

4.2. IoT botnet detector

The detector first collects the traffic, extracts the statistical features, and applies normalisation to the features. Then, it uses

the three trained modules to infer the class of the traffic. The output of each trained module is finally combined to obtain the final decision.

4.2.1. Traffic collection, feature extraction and normalisation

The network traffic to be analysed is transmitted from the IoT devices to the edge gateway. Subsequently, the traffic is transmitted (offloaded) to the MEC server. which extracts the statistical features of the collected data to be analysed. Statistical features of network traffic need to be gathered so that they can more precisely reflect the characteristics of IoT botnet behaviour. To begin, the network's packets are grouped together into larger units called flows. Second, the network's flow statistics are derived. This study makes use of two publicly available statistical features: IoT23 [27] and MedBIot [13]. We refer the reader to the methods proposed in [13,27] to extract statistical characteristics of network traffic. The min–max approach was used to normalise the features.

Since MEC servers are typically located close to IoT devices, it is not necessary for the MEC servers to wait an excessive amount of time for data transmission; thus, collecting and extracting the feature and IoT botnet detection would be in near real-time.

4.2.2. Trained modules inference

Using the MNN architecture, the extracted features are fed in parallel to all trained DNN modules and perform inference to predict IoT botnet behaviour in the MEC server. This allows the MEC server to accomplish three detection tasks (Benign vs. Scan, Benign vs. Scan, and Scan vs. CnC) simultaneously Even though we have already deployed the parallel design on the MEC server, there is still additional work that has to be done to completely optimise parallel computing within the MEC architecture. The scope of this paper does not include that work.

4.2.3. Module combination

Afterwards, the MEC server combines the inference results of all modules to make the final prediction and sends an alarm to the administrator that monitors the conditions of IoT devices. For the integration of module outputs, three integration units are applied.

First level combination. First, the inverse unit (INV) is used to obtain the output $f_{10}(x), f_{20}(x), f_{21}(x)$ of the reduced modules (M_{10}, M_{20}, M_{21}) from the output of discrimination function $f_{01}(x), f_{02}(x), f_{12}(x)$ of the trained modules (M_{01}, M_{02}, M_{12}) as follows:

$$f_{10}(x) = INVf_{01}(x) \tag{14}$$

$$f_{20}(x) = INVf_{02}(x) \tag{15}$$

$$f_{21}(x) = INVf_{12}(x) \tag{16}$$

Second level combination. For $\forall i, j \in \{1, 2, 3\}$ and $i \neq j$, the output of discrimination function $f_{ij}(x)$ obtained from the trained modules M_{ij} that trained using the same training set with C_i labelled by 1, which means:

$$T_{ij} = \{(X_i, 1)\} \cup \{(X_j, 0)\} \tag{17}$$

All $f_{ij}(x)$ should be combined by the MIN unit. Then, the discrimination function $f_i(x)$ of class C_i can be obtained by

$$f_i(x) = MIN_{j=0}^{K} f_{ij}(x), j \neq i \tag{18}$$

Final combination. All discrimination functions $f_i(x)$ of all classes are combined using the MAX unit. Therefore, the final discrimination function is

$$f(x) = MAX_{i=0}^{K} f_i(x) \tag{19}$$

Fig. 5 shows the module combination process for a three-class problem.

The overall sub-task combination approach is summarised in Algorithm 2.

Algorithm 2: Early stage IoT botnet detection (EDIT)

Input :
x - feature vector of IoT traffic
C_i - list of classes, i= 1,..,K
M_{ij} - list of Binary-classification Trained modules
Output:
p_{ij} - M_{ij} module prediction represents the probability of x belongs to C_i against C_j
p_{ji} - M_{ij} module prediction represents the probability of x belongs to C_j against C_i
p_i - probability of x belongs to C_i against all other classes,
P - final predicted class,

```
// Predict the class of traffic x
for all M_ij do in parallel
    p_ij ← predict (x)
end
// First level combination of output
for each p_ij do
    p_ji ← INV (p_ij)
end
// Second level combination of output
for each class C_i do
    p_i ← 1
    for each class C_j do
        if i < j then
            p_i ← MIN (p_i, p_ij)
        else
            p_i ← MIN (p_i, p_ji)
        end
    end
end
// final level combination of output
P ← 0
for each predicted class p_i do
    P ← MAX (P, p_i)
end
```

5. EDIT system development and experiments

5.1. IoT botnet benchmark datasets

We use two different datasets: IoT-23 [27], MedBIoT [13]. The datasets represent the traffic features during normal, early stages of compromising IoT devices and attacks originated by an IoT botnet. Thus, they are suitable for evaluating the proposed system.

IoT-23. IoT-23 is a dataset that represents the network traffic of IoT devices. Three different real IoT devices have been used to collect this dataset. The dataset contains three sub-datasets of benign network traffic as well as 20 sub-datasets of malicious network traffic. The 20 malicious sub-datasets are collected during the execution of several malicious scenarios related to botnet behaviours. There are a variety of botnet behaviours that were executed during the traffic collection process, such as Mirai, Torii, Trojan, Gagfyt, Kenjrio, Okiru, Hakai, IRCBot, Hajime, Muhstik, Hide and seek. Each botnet consists of four types of behaviours: scanning, command and Control (CnC), File Download, DDoS, and Heartbeats. The IoT-23 dataset has 20 network traffic features that describe the traffic behaviour during a specific time. Each entry in the dataset is labelled with the type of behaviour: benign or malicious. The type of malicious is also included for malicious entries.

MedBIoT. MedBIoT is a dataset of network traffic features collected from 83 real and emulated IoT devices during normal behaviour (benign samples) and botnet behaviour (malicious samples). Three botnet behaviours were deployed to extract their

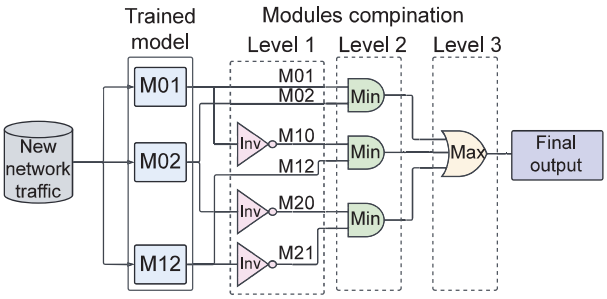

Fig. 5. Modules combination for three-class problem during test phase.

features, namely, Mirai, Bashlite and Tori. However, only the early stages of the botnet were considered: spreading and CnC communication. The MedBIoT dataset has 100 network traffic features that describe the traffic behaviour during a specific time. The dataset is split into sub-datasets according to the botnet stage and the device type, which makes this dataset appropriate for evaluating MNN.

In this study, besides benign traffic, we only consider the malicious traffic that represents the traffic of the devices infected by Mirai malware from both datasets. The IoT23 dataset has 7 sub-datasets of Mirai malware. These 7 sub-sets were combined into one dataset. We mainly focus on the early phases: scanning and CnC. Therefore, we only consider the flow that is labelled with these two phases. Regarding the MedBIoT dataset, the samples from the sub-datasets of the Mirai botnet (benign, scanning, CnC) were considered. Thus, we built three sub-datasets (DS_0, DS_1, DS_2) from each dataset (IoT23 and MedBIoT), where DS_0 is the dataset of benign traffic, DS_1 is the dataset of scanning behviour traffic, and DS_2 is the dataset of CnC behviour traffic. The EDIT is evaluated using the IoT-23 and MedBIoT separately.

5.2. Data pre-processing: normalisation, splitting, partitioning, and labelling

The two datasets display a notable imbalance. Within the realm of cybersecurity, when analysing network traffic, it is customary for the majority of traffic samples to fall under the normal category, whereas attack traffic types are regarded as the minority [29]. Deep learning methods provide less accurate results when classifying attacks on imbalanced datasets [30]. To mitigate the problem of class imbalance, we employ the random under-sampling strategy. Although random under-sampling may lead to the loss of important information from regular traffic, it is acceptable to assume that there are redundant observations in the normal traffic sample. By selectively removing redundant observations, it is improbable that the overall data distribution will see significant alterations. This approach is frequently employed in practical applications because of its straightforwardness and capacity to expedite the learning process.

min–max normalisation was performed, ensuring that no one feature dominates the model training stage. Each sub-dataset (DS_0, DS_1, DS_2) was then split into a training set (90%) and a 10% test (named TE0, TE1, and TE2). The three test sets are then combined into one test dataset that will be used for testing the detector component of the EDIT system.

From the datasets (DS_0, DS_1, DS_2), the 3 training datasets (T_{01}, T_{02}, T_{12}) for Benign against Scan, Benign against CnC and Scan against CnC, respectively, of the three modules (M_{01}, M_{02}, M_{12}), are obtained as described in task decomposition section (Section 4).

Further, to avoid model over-fitting during training the three modules, 20% of 80% of each set of (T_{01}, T_{02}, T_{12}) were considered as a validation set, producing three validation sets: (V_{01}, V_{02}, V_{12}).

Table 4
Number of samples per training, validation and test sets.

Dataset	Training set	Validation set	Test set
IoT23	32422	8105	4506
MedBIoT	63010	15752	8754

Table 5
Number of samples per network traffic type.

Dataset	Benign	Scan	CnC
IoT23	15011	15011	15011
MedBIoT	29172	29172	29172

Table 6
The parameters of NN modules.

Parameter	IoT23	MedBIoT
First layer (FL)	Dense layer	Dense layer
Second layer (SL)	Dense layer	Dense layer
Number of neurons (FL)	20 neurons	100 neurons
Number of neurons (SL)	10 neurons	50 neurons
Activation fun.	ReLU	ReLU
Third layer	Dorpout layer	Dorpout layer
Rate	0.2	0.2
Output layer	Dense layer	Dense layer
Number of output	2	2
Output activation fun.	Softmax	Softmax
Optimiser	Adam	Adam
Function loss	Crossentropy	Crossentropy
Epochs	10	10
Batch size	256	256

The number of samples per training, validation, and test set is shown in Table 4. Whereas Table 5 shows the number of samples per network traffic type.

5.3. Experimental settings

The EDIT system is evaluated using a MEC server with a 1.70 GHz, 4-core Intel i5 CPU. The experiment was implemented with Python 3.9.12 using TensorFlow 2.10.0 and Keras 2.10.0. We followed our proposed system architecture, first by constructing the model and then by using the model for IoT botnet detection. In our experiment, we used the trained model to test the detection by making predictions on the test dataset.

For the IoT23 dataset, we first constructed three multi-layer perceptron neural network models with Keras. Each model has four layers. The first layer is a dense layer with 20 neurons, and it specifies 20 inputs for the module. The second layer is also a dense layer with 10 neurons. The activation function applied by these two layers is the rectified linear activation function (ReLU). After these layers, a dropout layer is used with a rate of 0.2. This means that during each epoch of training, 20% of the neurons are randomly discarded. The output layer is also a dense layer with

Table 7
Detection performance (Root mean square error).

Dataset	Random state										Average
	1	437	984	42	5	66	715	30	16	618	
IoT23	0.01	0.00	0.01	0.01	0.01	0.00	0.00	0.01	0.01	0.00	0.006
MedBIoT	0.02	0.01	0.00	0.01	0.00	0.01	0.00	0.00	0.01	0.01	0.007

two neurons. The output layer used the softmax activation function to produce two outputs. The first output is the probability that each sample (x) belongs to the class labelled with 1 against the class labelled with 0. The second output represents the inversion of the first output. We then compiled the model with the Adam optimiser and the categorical cross-entropy function loss, and we chose the accuracy metric to evaluate the module during training. The modules were trained with 10 epochs and 256 batch sizes, and the validation dataset was used to evaluate each epoch. After constructing the modules, we used these modules to make predictions on the test data. We used the two outputs of each module and combined them using the Min function, and then we used the Max function; however, we used numpy.argmax() to convert the output into class labels.

For the MedBIoT dataset, we did the same procedure. However, the modules' input was 100, and the number of neurons was 100 and 50 for the first and second layers, respectively. The specific parameters are shown in Table 6.

For the evaluation, root mean squared error (RMSE) is used to provide overall system detection performance. In addition, the numbers of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) are determined using the confusion matrix. This offered an evaluation of the system using measures such as the true positive rate (TPR), the true negative rate (TNR), the false positive rate (FPR), the false negative rate (FNR), the accuracy (ACC), and the F-Score metrics.

Table 8
Metric results for Botnet detection using IoT23 and MedIoT datasets.

Dataset	Metric	Bening		Scanning		CnC	
		Monolithic	MNN	Monolithic	MNN	Monolithic	MNN
IoT23	TPR	0.1062	1.0000	1.0000	1.0000	0.1032	1.0000
	TNR	0.9996	1.0000	0.1051	1.0000	1.0000	1.0000
	FPR	0.0003	0.0000	0.8949	0.0000	0.0000	0.0000
	FNR	0.8938	0.0000	0.0000	0.0000	0.8968	0.0000
	ACC	0.7010	1.0000	0.4024	1.0000	0.7010	1.0000
	F-Score	0.1920	1.0000	0.5266	1.0000	0.1871	1.0000
MedIoT	TRP	0.9296	1.0000	0.8087	1.0000	0.9013	1.0000
	TNR	0.8974	1.0000	0.9415	1.0000	0.9796	1.0000
	FPR	0.1026	0.0000	0.0585	0.0000	0.0204	0.0000
	FNR	0.0704	0.0000	0.1913	0.0000	0.0987	0.0000
	ACC	0.9081	1.0000	0.8964	1.0000	0.9540	1.0000
	F-Score	0.8707	1.0000	0.8414	1.0000	0.9276	1.0000

Table 9
Comparison of EDIT (our work) with other ML algorithms.

Dataset	Model	Benign	Scanning	CnC
IoT23	DT	0.90	0.87	1.00
	RF	0.90	0.87	1.00
	SVM	0.69	0.77	0.54
	EDIT (our)	1.00	1.00	1.00
MedBIoT	DT	0.98	0.99	0.99
	RF	0.99	0.99	1.00
	SVM	0.53	0.10	0.34
	EDIT (our)	1.00	1.00	1.00

6. Experimental results

6.1. Effectiveness evaluation

6.1.1. Overall detection performance

The root mean squared error (RMSE) is an essential metric for evaluating the detection performance of a system on a certain dataset. Table 7 shows the average of the RMSE obtained by running the IoT botnet detection component of the system. It can be seen that the RMSE values are close to 0. That means the system can predict the data accurately. The IoT botnet detection component of the system was executed 10 times in the case of the different datasets used in this study. For each execution, we use a different value to split the test and training datasets randomly using the parameter "random_state". We found the best random_state value to provide the most accurate model when the RMSE equals 0. We used the random_state value 715 to obtain the rest of the evaluation results in the next sections.

6.1.2. Comparison with monolithic DNN

The model is evaluated against the monolithic DNN in terms of the performance of classification on the unseen test set. The Multilayer Perception (MLP) was applied to train the monolithic DNN using 100 input neurons, three hidden layers, and three outputs. The rectified linear activation function (ReLU) is used for the hidden layers, and the Softmax activation function is used for the output layer.

The monolithic DNN confusion matrix and MNN confusion matrix are shown in Figs. 6 and 7 for the IoT23 and MedBIoT datasets, respectively.

From the results in Figs. 6 and 7, it is clear that MNN produced zero false positives and zero false negatives.

A summary of the performance of MNN compared to monolithic DNN on performance metrics Precision, recall, and F-1 score are shown in Table 8 for IoT23 and MedBIoT datasets.

Based on the experimental findings, it is clear that our suggested MNN attained an accuracy of 100% for both types of early-stage IoT botnet behaviour (scanning and CnC), which was much higher than the accuracy produced by a monolithic DNN in both datasets. It has therefore been shown that the suggested MNN technique to identify an IoT botnet is a highly successful technique.

6.1.3. Comparison with other ML algorithms

In this section, we present the comparison between MNN and three widely used ML algorithms, namely, Decision Tree (DT), Random Forest (RF), and Support Vector Machine (SVM), to classify the three classes (benign, scanning, and CnC) using the two datasets IoT23 and MedBIoT. The results of the F-score are shown in Table 9. The MNN model is superior to all others, as shown by its perfect F-scores on both datasets. The RF technique outperforms DT and SVM, despite slight variations in performance across datasets, according to the F-score.

6.1.4. Comparison with other studies

Previous research that examined botnet detection using a variety of ML/DL algorithms is used as a comparison for the solution that is being suggested here for both the benchmark dataset and the underlying algorithms. As can be seen in Table 10, the EDIT performs well, with a false positive rate and a false negative rate that are entirely absent. Compared to earlier research, Table 10 also shows that the EDIT provides a better level of classifier performance in terms of accuracy and F-Score.

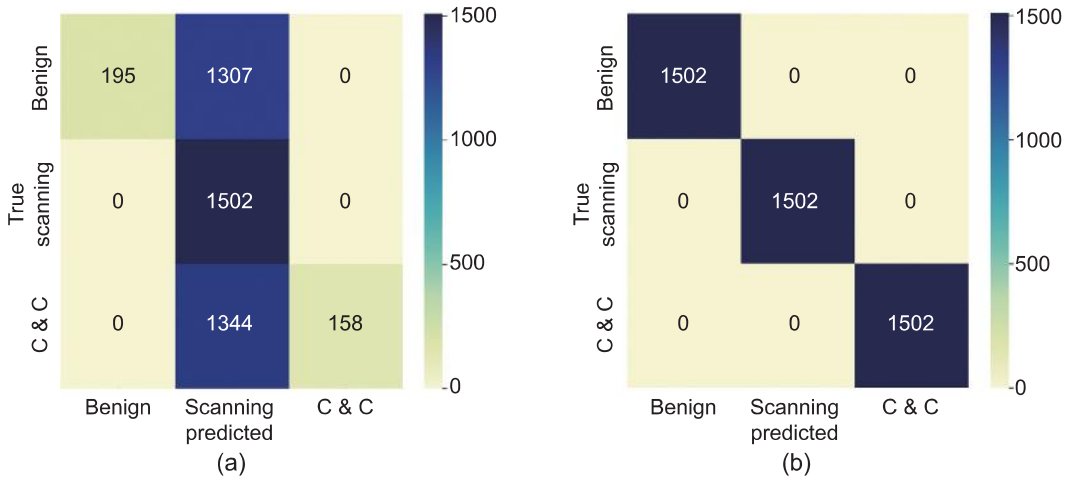

Fig. 6. Confusion matrix for Botnet detection in IoT23 dataset. (a) Monolithic DNN. (b) MNN.

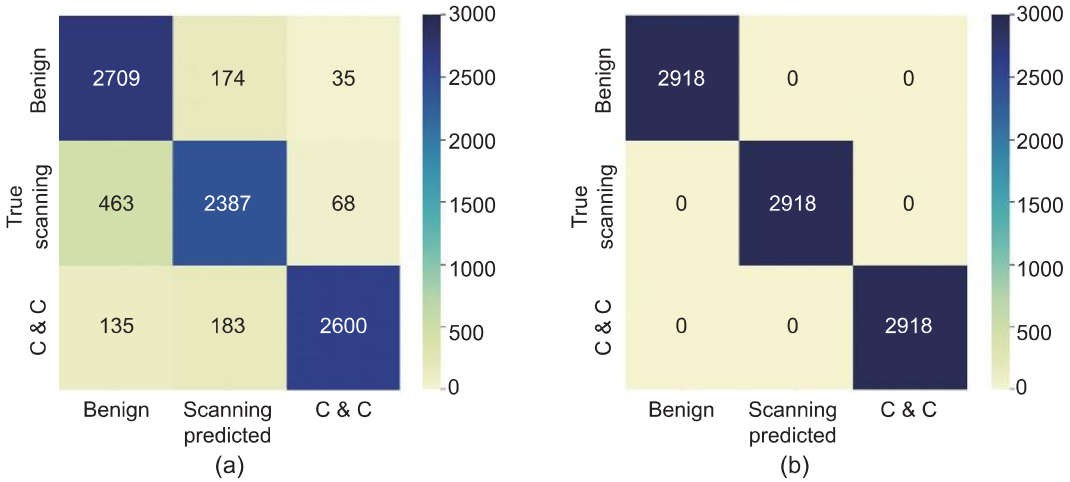

Fig. 7. Confusion matrix for Botnet detection in MedBIoT dataset. (a) Monolithic DNN. (b) MNN.

Table 10
Comparison of EDIT (our work) with other studies.

Dataset	Ref.	Year	Acc.	F-Score	FPR	FNR
IoT23	[9]	2020	0.85	–	0.60	–
	[11]	2021	–	0.94	–	–
	EDIT (our)	–	1.00	1.00	0.00	0.00
MedBIoT	[14]	2021	0.99	0.98	–	–
	[16]	2022	0.99	–	–	–
	EDIT (our)	–	1.00	1.00	0.00	0.00

Table 11
Efficiency performance of EDIT on MEC.

Dataset	Metrics	Model construction	IoT bot detection
IoT23	Time	24.26 s	0.016326 s
	Storage	5.23 kB	2.52 kB
	Model size	123 kB	–
MedBIoT	Time	28.90 s	0.016372 s
	Storage	5.06 kB	2.53 kB
	Model size	630 kB	–

6.2. Efficiency evaluation

We evaluate the efficiency of EDIT by considering two performance metrics: time and memory consumption for both system components (model construction and IoT bot detection). The metric time is the time in seconds the EDIT takes to construct the MNN model and the time it takes to finish the inference task for the type of one flow traffic in the test dataset. Whereas memory consumption refers to the amount of storage needed for storing EDIT code and the constructed model that contains three NN modules.

In our experiment, we use a multi-access edge computing server (MEC) with a 4-core Intel i5 CPU. Taking advantage of the modularity of MNN used in EDIT, the three modules can be trained and executed for detection using multi-core parallelism on MEC.

The efficiency performance of EDIT on MEC using the two datasets is shown in Table 11. The findings lead to a system that is realistic in terms of model construction and detection time, as well as requiring less storage space.

7. Conclusion

In this research, we propose EDIT, a system designed to detect and investigate communication traffic faults at smart city networks brought on by IoT botnets in their early stages. Multi-access edge computing (MEC) servers located near IoT devices are used to implement EDIT's Modular Neural Network (MNN) technique for IoT botnet detection. As a result of MEC's parallel

processing capabilities, modular neural networks (MNN) may improve detection accuracy and efficiency. The results show that the false-negative rate for EDIT is much lower than that of the monolithic method and other studies. When using the MEC server, EDIT model construction consumes little storage space and might take as little as 24 s. The result of the EDIT detection task also showed a low detection time of 16 ms. For future work, it might be interesting to investigate the deployment of software-defined networks (SDN) at edge computing to mitigate the IoT botnet at early stages before any intended attack could happen.

CRediT authorship contribution statement

Duaa Alqattan: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Varun Ojha:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing – review & editing, Supervision, Funding acquisition. **Fawzy Habib:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing – review & editing, Supervision, Funding acquisition. **Ayman Noor:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing – review & editing, Supervision, Funding acquisition. **Graham Morgan:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing – review & editing, Supervision, Funding acquisition. **Rajiv Ranjan:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Wazzan, D. Algazzawi, O. Bamasaq, A. Albeshri, L. Cheng, Internet of things botnet detection approaches: Analysis and recommendations for future research, Appl. Sci. 11 (12) (2021) 5713.

[2] G. Kambourakis, M. Anagnostopoulos, W. Meng, P. Zhou, Botnets: Architectures, Countermeasures, and Challenges, CRC Press, 2019.

[3] B. Vignau, R. Khoury, S. Hallé, A. Hamou-Lhadj, The evolution of IoT malwares, from 2008 to 2019: Survey, taxonomy, process simulator and perspectives, J. Syst. Archit. 116 (2021) 102143.

[4] R. Vishwakarma, A.K. Jain, A survey of DDoS attacking techniques and defence mechanisms in the IoT network, Telecommun. Syst. 73 (1) (2020) 3–25.

[5] N. Vlajic, D. Zhou, IoT as a land of opportunity for DDoS hackers, Computer 51 (7) (2018) 26–34, http://dx.doi.org/10.1109/MC.2018.3011046.

[6] S. Dange, M. Chatterjee, IoT botnet: the largest threat to the IoT network, in: Data Communication and Networks, Springer, 2020, pp. 137–157.

[7] Y. Jia, F. Zhong, A. Alrawais, B. Gong, X. Cheng, FlowGuard: An intelligent edge defense mechanism against IoT DDoS attacks, IEEE Internet Things J. 7 (10) (2020) 9552–9562, http://dx.doi.org/10.1109/JIOT.2020.2993782.

[8] M.M. Alani, BotStop : Packet-based efficient and explainable IoT botnet detection using machine learning, Comput. Commun. 193 (2022) 53–62, http://dx.doi.org/10.1016/j.comcom.2022.06.039.

[9] M. Hegde, G. Kepnang, M. Al Mazroei, J.S. Chavis, L. Watkins, Identification of botnet activity in IoT network traffic using machine learning, in: 2020 International Conference on Intelligent Data Science Technologies and Applications, IDSTA, IEEE, 2020, pp. 21–27.

[10] H. Alzahrani, M. Abulkhair, E. Alkayal, A multi-class neural network model for rapid detection of IoT botnet attacks, Int. J. Adv. Comput. Sci. Appl. 11 (7) (2020) 688–696.

[11] N. Abdalgawad, A. Sajun, Y. Kaddoura, I.A. Zualkernan, F. Aloul, Generative deep learning to detect cyberattacks for the IoT-23 dataset, IEEE Access 10 (2021) 6430–6441.

[12] M. Wazzan, D. Algazzawi, A. Albeshri, S. Hasan, O. Rabie, M.Z. Asghar, Cross deep learning method for effectively detecting the propagation of IoT botnet, Sensors 22 (10) (2022) 3895.

[13] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, S. Nõmm, MedBIoT: Generation of an IoT botnet dataset in a medium-sized IoT network, in: ICISSP, 2020, pp. 207–218.

[14] L. Giaretta, A. Lekssays, B. Carminati, E. Ferrari, Š. Girdzijauskas, LiMNet: Early-stage detection of IoT botnets with lightweight memory networks, in: European Symposium on Research in Computer Security, Springer, 2021, pp. 605–625.

[15] R. Gandhi, Y. Li, Comparing machine learning and deep learning for IoT botnet detection, in: 2021 IEEE International Conference on Smart Computing, SMARTCOMP, IEEE, 2021, pp. 234–239.

[16] K. Malik, F. Rehman, T. Maqsood, S. Mustafa, O. Khalid, A. Akhunzada, Lightweight internet of things botnet detection using one-class classification, Sensors 22 (10) (2022) 3646.

[17] Y.N. Soe, Y. Feng, P.I. Santosa, R. Hartanto, K. Sakurai, A sequential scheme for detecting cyber attacks in IoT environment, in: 2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress, DASC/PiCom/CBDCom/CyberSciTech, IEEE, 2019, pp. 238–244.

[18] Y.N. Soe, Y. Feng, P.I. Santosa, R. Hartanto, K. Sakurai, Machine learning-based IoT-botnet attack detection with sequential architecture, Sensors 20 (16) (2020) 4372.

[19] F. Hussain, S.G. Abbas, I.M. Pires, S. Tanveer, U.U. Fayyaz, N.M. Garcia, G.A. Shah, F. Shahzad, A two-fold machine learning approach to prevent and detect IoT botnet attacks, IEEE Access 9 (2021) 163412–163430.

[20] G.L. Nguyen, B. Dumba, Q.-D. Ngo, H.-V. Le, T.N. Nguyen, A collaborative approach to early detection of IoT botnet, Comput. Electr. Eng. 97 (2022) 107525.

[21] A. Kumar, M. Shridhar, S. Swaminathan, T.J. Lim, Machine learning-based early detection of IoT botnets using network-edge traffic, Comput. Secur. 117 (2022) 102693.

[22] K. Chen, Deep and modular neural networks, in: Springer Handbook of Computational Intelligence, Springer, 2015, pp. 473–494.

[23] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J.A. Halderman, L. Invernizzi, M. Kallitsis, et al., Understanding the mirai botnet, in: 26th USENIX Security Symposium, USENIX Security 17, 2017, pp. 1093–1110.

[24] J.M. Ceron, K. Steding-Jessen, C. Hoepers, L.Z. Granville, C.B. Margi, Improving iot botnet investigation using an adaptive network layer, Sensors 19 (3) (2019) 727.

[25] J. Gamblin, Mirai source-code. 2017, 2019.

[26] X. Zhang, O. Upton, N.L. Beebe, K.-K.R. Choo, IoT botnet forensics: A comprehensive digital forensic case study on mirai botnet servers, Forensic Sci. Int.: Digit. Investig. 32 (2020) 300926.

[27] A. Parmisano, S. Garcia, M. Erquiaga, Iot-23 Dataset: A Labeled Dataset of Malware and Benign Iot Traffic, Avast-AIC Laboratory, Stratosphere IPS, Czech Technical University (CTU), 2019.

[28] B.-L. Lu, M. Ito, Task decomposition and module combination based on class relations: a modular neural network for pattern classification, IEEE Trans. Neural Netw. 10 (5) (1999) 1244–1256.

[29] B.R. Silva, R.J. Silveira, M.G. da Silva Neto, P.C. Cortez, D.G. Gomes, A comparative analysis of undersampling techniques for network intrusion detection systems design, J. Commun. Inf. Syst. 36 (1) (2021) 31–43.

[30] P. Bedi, N. Gupta, V. Jindal, Siam-IDS: Handling class imbalance problem in intrusion detection systems using siamese neural network, Procedia Comput. Sci. 171 (2020) 780–789.