MalBoT-DRL: Malware Botnet Detection Using Deep Reinforcement Learning in IoT Networks

Mohammad Al-Fawa'reh, *Member, IEEE*, Jumana Abu-Khalaf, *Senior Member, IEEE*, Patryk Szewczyk, and James Jin Kang

***Abstract*—In the dynamic landscape of cyber threats, multistage malware botnets have surfaced as significant threats of concern. These sophisticated threats can exploit Internet of Things (IoT) devices to undertake an array of cyberattacks, ranging from basic infections to complex operations, such as phishing, cryptojacking, and Distributed Denial-of-Service (DDoS) attacks. Existing machine learning solutions are often constrained by their limited generalizability across various data sets and their inability to adapt to the mutable patterns of malware attacks in real world environments, a challenge known as model drift. This limitation highlights the pressing need for adaptive intrusion detection systems (IDSs), capable of adjusting to evolving threat patterns and new or unseen attacks. This article introduces MalBoT-DRL, a robust malware botnet detector using deep reinforcement learning (RL). Designed to detect botnets throughout their entire lifecycle, MalBoT-DRL has better generalizability and offers a resilient solution to model drift. This model integrates damped incremental statistics with an attention reward mechanism, a combination that has not been extensively explored in the literature. This integration enables MalBoT-DRL to dynamically adapt to the ever-changing malware patterns within IoT environments. The performance of MalBoT-DRL has been validated via trace-driven experiments using two representative data sets: 1) MedBIoT and 2) N-BaIoT, resulting in exceptional average detection rates of 99.80% and 99.40% in the early and late detection phases, respectively. To the best of our knowledge, this work introduces one of the first studies to investigate the efficacy of RL in enhancing the generalizability of IDS.**

***Index Terms*—Bashlite, botnet detection, incremental statistics, intrusion detection, Internet of Things (IoT) botnet, MalBoT-DRL, mirai, network traffic analysis, reinforcement learning (RL), torii.**

I. Introduction

THE RAPID proliferation of Internet of Things (IoT) devices is expected to revolutionize various aspects of our lives, from improving efficiency in manufacturing [1] and healthcare [2] to evolving the concept of smart homes further [3]. According to Dataprot [4], the number of IoT devices deployed globally will exceed 29 billion by 2030. However, this drastic growth is not without concerns and challenges. The security of these devices is a primary concern, where threats from malware botnets are increasingly becoming a significant threat to the IoT ecosystem [5], [6], [7], [8], [9], [10]. Cybercriminals are continuously leveraging advanced techniques to exploit the vulnerabilities of IoT devices and convert them into botnets to perform malicious activities, such as Distributed Denial-of-Service (DDoS) attacks and data theft [11], [12], [13], [14], [15], [16]. A recent report by Nokia [17] has revealed that, as of 2023, approximately 1 million IoT devices are implicated in botnet-propelled DDoS attacks. Presently, IoT devices contribute to more than 40% of all DDoS traffic. This development does not only accentuate the susceptibility of IoT devices to cyber-attacks but also underscores the potential risk to critical infrastructures including energy grids, transport systems, and healthcare services, all of which are reliant on IoT. To counter this rising threat, current defensive measures largely rely on the use of machine learning (ML) and deep learning (DL) models as anomaly-based intrusion detection systems (IDSs) [18], [19], [20], [21], [22], [23], [24], [25]. However, despite their impressive capabilities, these models often face challenges, such as lack of generalizability [26] and model drift [27]. Generalizability refers to a model's capacity to effectively apply its learned knowledge from training data to new, unseen contexts, such as zero-day attacks [28]. Model drift, conversely, arises when underlying data distribution changes over time, resulting in the deterioration of a model's performance [29], [30]. Given the dynamic landscape of cyber threats to IoT devices , where cyber criminals constantly devise new strategies and create novel malware (such as adversarial attacks [31], [32], [33]), the difference between the data used to train a model and the data it encounters in reality, can become substantial. Additionally, a prominent shortcoming in the existing body of research is the focus on the later stages of malware botnet life cycles, where the early stages of infection and propagation are often overlooked. Addressing these early stages is crucial, as identifying and stopping botnets early on can prevent further propagation and subsequent damage.

Existing solutions address these challenges by retraining ML models with new data. However, this approach is computationally expensive as evident in [26] and [29]. Moreover, the time required to identify new types of threats can be significant, which makes IoT devices vulnerable during this time. Motivated by the escalating complexity of cyber threats and the limitations of existing defensive strategies, this article focuses on the application of reinforcement learning (RL) as an IDS. An integral part of this article involves evaluating

Manuscript received 8 May 2023; revised 2 August 2023 and 3 October 2023; accepted 6 October 2023. Date of publication 12 October 2023; date of current version 7 March 2024. *(Corresponding author: Mohammad Al-Fawa'reh.)*

The authors are with the School of Science, Edith Cowan University, Joondalup, WA 6027, Australia (e-mail: m.alfawareh@ecu.edu.au; j.abukhalaf@ecu.edu.au; p.szewczyk@ecu.edu.au; james.kang@ecu.edu.au).

Digital Object Identifier 10.1109/JIOT.2023.3324053

the generalizability of RL when used as an IDS, with the goal of better understanding its potential to adapt and effectively respond to a broad spectrum of cyber threats and malware activities throughout their entire lifecycle [34]. An IDS based on RL has the capacity to evolve its strategies over time, guided by the feedback it receives. This provides a robust defense against malware botnets in IoT devices while addressing the early stages of the botnets' life cycles. RL has considerable advantages when compared to other learning strategies. Unlike supervised learning, which demands large volumes of labeled data, RL operates on the principle of learning from delayed, scaled rewards [35]. This key feature allows RL to exhibit a more flexible adaptation to its environment. In contrast to this, unsupervised learning utilizes unlabeled data to discern underlying patterns. On the other hand, RL applies a different approach, gaining knowledge through direct interaction with its environment, independent of any prerequisite data. Significantly, a pronounced differentiation is evident in the way data samples are handled by RL versus other DL models. DL typically assumes that data samples are independent and uniformly distributed [36]. This assumption can lead to a lack of generalization. However, RL does not uphold this assumption and recognizes the possible dependencies among data samples [36], which can prove to be beneficial in various real-world applications. Moreover, a fundamental disparity exists between the learning processes in RL and DL [36]. While DL models learn from a static underlying data distribution, RL models are capable of handling dynamic data distribution. This characteristic makes RL better equipped to handle unpredictable environments where the properties and characteristics of the data may change over time [35]. While a significant volume of research has recently been conducted on utilizing RL for cyber attack detection, only a handful of these studies have focused on botnets, particularly during the different stages of their life cycle. To the best of our knowledge, there are no comprehensive studies that investigate the issues of generalizability and model drift in the context of RL applications for detecting malware botnets throughout their entire life cycle. In this research, an RL-based IDS which integrates a damped incremental statistics approach with a reward attention mechanism is proposed for the detection of malicious botnet activities on IoT devices, with particular emphasis on the early stages of botnet infection and propagation. The objective is to bridge existing research gaps in ML/DL models and present a robust, scalable, and self-learning solution that can secure IoT devices against the escalating threat of malware botnets. The main contributions of this article are as follows.

1) To propose a state-of-the-art (SOTA) IDS utilizing an RL off-policy learning strategy with a deep neural network (DNN) model to detect IoT botnet attacks throughout their life cycle: the infection phase,communication with the Command and Control (C&C), and post-attack activities.
2) To introduce an attention-based reward mechanism for managing attacks with a smaller number of samples.
3) To provide an in-depth performance analysis of the proposed model using realistic scenarios with varying malware samples, as well as comparing the proposed model with SOTA models.
4) To provide a model that has low computational complexity, rendering it suitable for implementation within small office/home office (SoHo) environments.

The remainder of this article is organized as follows: an overview of related work is presented in Section II. Section III provides an overview of RL. The proposed RL-based IDS is introduced in Section IV. A comprehensive evaluation of the system compared to existing solutions is described in Section V. Section VI addresses discussions and limitations of the proposed system. Finally, this article is concluded in Section VII, where potential future directions for this research are also outlined.

II. Related Work

Current research studies primarily concentrate on developing strategies, such as anomaly detection methods, to address botnet activities in IoT network environments. With a focus on the botnet life cycle, the activities include propagation processes in the early stages, and establishing communications with C&C servers in the later stages. These activities can culminate in subsequent cyber-attacks. This section reviews SOTA anomaly detection techniques in IoT networks, emphasizing the challenges presented by cost-effective centralized malware botnets. These techniques are categorized into two main types: 1) netflow based and 2) packet based (PB). This review covers the following.

1) DL-based anomaly detection.
2) Graph-based anomaly detection.
3) ML-based anomaly detection.
4) RL-based anomaly detection.

Table I summarizes the most important findings of the reviewed papers. The table accounts for a variety of aspects including the utilized data sets, the applied methods, the achieved accuracy (A), whether the study examined the full life cycle of the malware (FL), if it adopted a flow-based (FB) or PB approach, whether the study investigated generalizability (Gen), and the limitations encountered in each study.

1) *DL-Based Anomaly Detection:* DNNs are foundational elements in commercial security products, known for their efficacy in identifying various network attacks [37]. Their significance has led researchers to widely adopt them for attack detection. For instance, Kou et al. [38] have introduced a botnet detection method for cloud architecture, leveraging DL techniques. They extracted basic network flow characteristics from data packets and transformed them into gray images. A convolutional neural network (CNN) algorithm has been employed to learn and extract abstract features that represent the underlying patterns and structural relationships in the network flow. The proposed method achieved an accuracy of 89.6%, but suffered from a high false-positive (FP) rate. Similarly, Liu et al. [39] have proposed a CNN with statistical methods to extract network traffic features specific to IoT botnets. These features were further transformed into grayscale images

TABLE I
SUMMARY OF THE REVIEWED PAPERS

Ref	Dataset	Model	A	FL	FB	PB	Gen	Limitation
[38]	UNB ISCX	CNN	89.6	✓	✓	✗	✗	High false positive alerts
[39]	N-BaIoT	CNN	99.57	✓	✓	✗	✗	inability to detect stealth communication with the C&C
[40]	CIC-DDoS2019	Hybrid-DL	94.52	✗	✓	✗	✗	Inability to detect low-based DDoS (crossfire attacks)
[41]	MedBIoT	LSTM-CNN	99.7	✗	✓	✗	✗	Computationally costly
[42]	Self	CNN	90	✗	✓	✗	✗	Cannot detect complex attacks such as covert channels
[43]	Bot- IoT	ResNet-18	98.89	✗	✓	✗	✗	Limited to the specific scenarios and datasets
[44]	Self	MLP	65.1	✓	✓	✗	✗	High false positive & negative samples
[46]	Self	BLSTM-RNN	99	✓	✗	✓	✗	computationally expensive
[47]	MedBIoT	LiMNet	99.7	✗	✓	✗	✗	Inability to detect stealth-based malware
[54]	Self	Graph-based	99.94	✗	✓	✗	✗	Inability to detect evasion malware
[57]	IoTPOT	PSI	98.7	✓	✓	✗	✗	Easy to bypass
[58]	Self	OCSVM	85	✓	✓	✗	✗	Inability to detect stealth-based malware
[59]	N-BaIoT	GWO	96-99	✗	✓	✗	✗	Inability to detect stealth-based malware
[60]	Bot-IoT-2018	BO-GP	99.99	✗	✓	✗	✗	Can not detect complex attack patterns
[61]	N-BaIoT	LSTM- Chi20	99	✗	✓	✗	✗	Can not detect dynamic malware
[62]	MedBIoT	G-PCA	x	✗	✓	✗	✗	Scalability
[45]	N-BaIoT	Parallel learning	99.98	✗	✓	✗	✗	Computationally costly
[66]	NSL-KDD	DQN	98	✗	✓	✗	✗	Easy to bypass
[67]	NSL-KDD	AESMOTE	82	✗	✓	✗	✗	Limited to supervised environment
[68]	DS2OS	DRL-GAN	99.05	✗	✓	✗	✗	GAN generates invalid network samples
[70]	MedBIoT	SARSA	96.99	✗	✓	✗	✗	Limited to labeled data
[72]	NSL-KDD	AE-RL	80	✗	✓	✗	✗	Limited to supervised environment
[74]	NSW-NB15	DQN	98	✗	✓	✗	✗	Computationally costly
[76]	Self	Q	90	✗	✓	✗	✗	Can not handle high-dimensional distribution
[78]	NSL-KDD	Q-RST	98	✗	✓	✗	✗	Computationally costly
[77]	Simulated	RNN-Q	100	✗	✓	✗	✗	Not applicable in real environments

using triangle area maps and multivariate correlation analysis algorithms. The CNN model was trained and evaluated using the N-BaIoT data set, achieving an accuracy rate of 99.57%. However, it is worth noting here that the study's main limitation lies in its focus solely on the later stages of malware activities, without considering the full range of malware behaviors. Alghazzawi et al. [40] have focused on DDoS attacks launched from botnets in a heterogeneous environment. Their approach utilized a combination of CNN and long short-term memory (LSTM). The CNN was employed to extract the most impactful features from the network flow data, which were then fed into the LSTM model. The experimental results demonstrated that the hybrid model achieved an accuracy rate of up to 94.52%. However, this approach neglects low-traffic DDoS attacks. Wazzan et al. [41] have employed a different architecture as compared to Alghazzawi et al. [40] to investigate the propagation stages of malware botnets and their communication with the C&C. Their proposed approach yielded promising results, achieving an accuracy rate of 99.7%. However, the authors have used a simulated data set for their evaluation, which may differ from real-world scenarios. Jung et al. [42] have presented a power-consumption-based detection system for IoT devices. The system monitors the power consumption patterns of IoT devices during normal activities and compares them to changes in power consumption observed during malicious bot activities. The study found that botnet traffic exhibits a detectable pattern in power consumption. However, the proposed model faced challenges with model drifting when attackers employed adversarial attacks. These attacks aim to manipulate the power consumption patterns to evade detection, posing a limitation to the effectiveness of the detection system. Hussain [43] have proposed a twofold approach for detecting IoT botnets, employing a pretrained ResNet-18 model. The first phase focused on identifying scanning activities during the initial stage of the attack, while the second phase aimed to detect DDoS activity. While the approach achieved a high accuracy rate of 98.89%, it suffered from a lack of generalization. This limitation indicates that the model's performance may be limited to the specific scenarios and data sets used during training, and may not effectively detect botnet activities in different environments or with unseen variations. Zhou [44] have introduced a deep packet inspection framework for detecting IoT botnets based on the hypertext transfer protocol. They extracted dimensional feature vectors from network traffic within a 1-h time window. Their test results using a multilayer perceptron (MLP) achieved an accuracy rate of 65.1%. However, their model resulted in high FP rates, despite reducing the feature extraction time. Sattari et al. [45] have introduced an IDS fog computing model that combines parallel learning with a software-defined network (SDN) approach to tackle bottleneck attacks in IoT environments. Their model leverages simultaneous learning from various network features and integrates the learning using a DNN approach. Experimental evaluations on the N-BaIoT data set yielded impressive results, with an average accuracy of 99.98% and an average testing time of 0.022 ms. However, one limitation of this model is its reliance on fixed attack distribution, rendering it susceptible to model drifting over time. McDermott et al. [46] have introduced

a PB approach to detect malicious botnet activities, employing a bidirectional LSTM-based recurrent neural network (BLSTM-RNN) combined with word embedding techniques. Five features were extracted from a simulated environment, including source and destination IP addresses, packet length and payload information. While the proposed model achieved a high accuracy rate of 99%, it was computationally expensive. Moreover, a limitation of the BLSTM-RNN approach is its disregard for malware mutation activities. Giaretta et al. [47] have developed a lightweight memory network (LiMNet) that utilizes an internal memory component to capture the behavior of individual IoT devices over time during the propagation phase. The memory module incorporates both packet features and the behavior of peer devices. However, the study reported a limitation in the model's generalization capabilities.

2) *Graph-Based Anomaly Detection:* The graph analysis approach is a powerful method for finding anomalies in data by utilizing the inherent structure and relationships in graph data [48], [49], [50]. It has proven especially useful in several areas, such as network security [51], social network analysis [52], and bioinformatics [53]. Several studies have detailed the effectiveness and versatility of these graph-based anomaly detection techniques, such as: Wang et al. [54] have proposed a deep packet inspection approach, where they integrate graph-based and FB network traffic analysis. The framework extracts 15 FB features and three graph-based features. FB detection involves measuring C-flow similarity and stability using k-means clustering and packet length distribution. Graph-based detection focuses on identifying anomalous node neighborhoods using least-square and local outlier factor (LOF) techniques. This model integrates similarity, stability, and anomaly scores to identify botnets. However, as identified by Hostiadi and Ahmad [55], a limitation of the model is its inability to effectively detect evolving attacks. In contrast Yassin et al. [56] have concentrated on the frequency-based dependency graph approach, with the study's main focus on analyzing registry information to identify similar and dissimilar malicious activities within the Mirai botnet. A limitation of this study includes overlooking other commonly used data sources, such as DLL and API call information. Additionally, the behavior analysis was limited to the Mirai botnet, which narrowed the scope of the study. Similarly, Nguyen et al. [57] have presented a graph-based detection system designed to identify malicious activities of IoT botnets targeting multiarchitecture environments. Printing string information (PSI) graphs were used as the foundation for this approach, extracting high-level features from the function call graph of each executable file associated with IoT devices. By employing a CNN, the proposed system achieved an accuracy of 98.7% during testing. However, one notable limitation of this PSI approach is its disregard for runtime information of executable files, which could potentially provide valuable insights, especially in the case of stealth-based malware.

3) *ML-Based Anomaly Detection:* Studies incorporating unsupervised learning techniques, such as the work by Nomm and Bahsi [58], employ feature selection methods based on entropy, variance and Hopkins statistics, using ML models like LOF, one-class support vector machine (OCSVM), and isolation forest (IF) to detect potential botnet activities by learning from normal traffic only. Their evaluation demonstrates promising performance, although limitations exist in detecting stealth-based malware. Further research is needed to address these limitations and to explore additional IoT botnet behaviors. Similarly, Shorman et al. [59] have presented an unsupervised detection system for identifying IoT botnet activities using an OCSVM, where the gray wolf optimization (GWO) algorithm was employed to fine-tune the hyperparameters of the OCSVM model and rank the features. The evaluation conducted on the N-BaIoT data set resulted in an accuracy rate of 96-99%. However, the model testing was based on a small number of attack samples, and it did not consider post-attack activities. Injadat et al. [60] have proposed a Tiny ML approach for detecting malicious botnet activities by integrating the Bayesian optimization Gaussian process (BO-GP) with the decision tree (DT) algorithm. The authors utilized FB network features as inputs to their model. During the evaluation phase, the model was tested using the Bot-IoT-2018 data set, and achieved an outstanding accuracy rate of 99.99%. However, the model's performance declined when confronted with complex attack patterns, as evidenced in [26]. Gandhi and Li [61] have employed flow-level-based analysis along with the Chi-Square method to reduce the number of network features. They evaluated a variety of ML algorithms, including Logistic Regression, K-Nearest Neighbor, Naíve Bayes, DT, and Random Forest, as well as two DL algorithms: a) MLP and b) LSTM. The primary focus of this analysis was on the Mirai botnet, which served as the basis for the adapted samples. However, the experimental results revealed a limitation in the generalization capabilities of the ML algorithms across different attack vectors. These findings suggest that the effectiveness of these algorithms in detecting botnet attacks may be specific to the context of the Mirai botnet, and their performance may not extend to other botnet variants or diverse attack scenarios. Aprianti [62] have employed Tiny ML techniques to detect IoT botnet activities utilizing a Gaussian model and principal component analysis (PCA) for dimensionality reduction and capturing statistical patterns. Their approach achieved a high detection rate of 97.49% during the late stage of the botnet life cycle. However, further research is needed to enhance the generalization capabilities of this approach, as highlighted in [26]. The aforementioned techniques [38], [39], [56], [57], [58], [59], [60], [61], [62] have demonstrated efficacy in identifying conspicuous alterations induced by atypical behaviors. However , these methods were tested solely on certain attack vectors and were limited to a single stage of malicious botnet operations.

According to evidence from literature [26], the learning methodologies employed by various models, such as the use of labeled data and learning from static distributions, are inadequate when addressing mutable threats, notably malware botnets. Recently, there has been a significant shift toward RL techniques [63] inspired by their ability to generalize well in gaming [64] and robotic navigation tasks [65], RL offers the potential to address the complexities associated with detecting and combating botnet activities. RL models learn to make optimal decisions through interactions with the environment, without requiring explicit supervision or labeled data. This approach appears promising in developing robust and adaptable detection systems capable of addressing the dynamic and evolving nature of botnet attacks.

4) *RL-Based Anomaly Detection:* Cyber security researchers have begun investigating this paradigm to detect malicious attacks. However, the majority of these attempts remain in their infancy, and few researchers have investigated IoT botnet attacks in this context. In [66], an RL model was developed to enhance the security of IoT devices and wireless sensor networks (WSNs) via integrating Q-learning with CNN. The model achieved a commendable accuracy rate of 98%. However, the evaluation process was restricted to legacy attacks, possibly neglecting its performance against emerging and more sophisticated attack techniques. A major issue with this approach is the overestimation of the Q-value, as their method only considers the maximum value of possible actions. This limitation can result in incorrect classifications, as evidenced by Ma and Shi [67]. In addition, the authors delved into examining the entire distribution of future returns (total rewards), rather than solely focusing on their expected value. They also integrated the generative adversarial network (GAN) algorithm to oversample less frequent network attacks, aiming to detect network intrusions in the context of the Industrial IoT (IIoT) [68]. However, a significant limitation of this study is its primary focus on a specific type of attack, overlooking stealth-based intrusions such as malware botnets. Furthermore, the GAN approach generated invalid attack samples, as evidenced in [69]. May Raju and Gupta [70] concentrated on adversarial RL, where they implemented the state action reward state action (SARSA) algorithm in a supervised manner. The proposed system consisted of two agents: one acting as the environment and the other serving as a classification agent. An accuracy rate of 96.99% was achieved using the MedBIoT data set. However, as highlighted in [71], the application of the SARSA algorithm can encounter challenges, such as not being able to determine the optimal policy for detection when the agent does not perform sufficient exploration. Another limitation is that the model heavily relies on labeled data, which is a costly approach. Caminero et al. [72] have utilized two RL agents to detect network attacks. The first agent was responsible for classifying the network traffic as malicious, while the second agent focused on classifying the attack type. The main limitation of this approach is the high FP rate (80% accuracy), along with the high computational cost. Furthermore, this approach is constrained in its applicability, as it operates exclusively in supervised environments where labeled training data and attack samples, are costly to acquire. Ma and Shi [67] built upon the approach introduced in [72], with a specific emphasis on addressing the issue of data imbalance. They employed various techniques, including SMOTE, ROS and NearMiss, to mitigate the impact of imbalanced data. Among these approaches, SMOTE demonstrated the best performance, achieving an accuracy of 82%. The model developed by Ma and Shi outperformed the one presented by Caminero et al. [72]. However, despite its improved performance, the model still encountered challenges in reducing FP alerts to an acceptable level, as highlighted in [73], where 10-20 alerts remained problematic. Hsu and Matsuoka [74] have used deep Q learning (DQN) to detect network traffic behavior and evaluated their model using the NSL-KDD and UNSW-NB15 data sets. Their model achieved a high detection rate. However, the DQN algorithm overestimated the expected future rewards, which led to poor performance and generated high FP rewards, as evident in [75]. Servin et al. [76] introduced an anomaly detection system based on a Q-learning approach, utilizing a simulated environment. However, the reward value in the system was manually controlled, which may introduce biases and affect the accuracy of anomaly detection. Moreover, the Q-learning approach employs a tabular structure to store the agent's history, making it less suitable for handling multidimensional network traffic. Huang et al. [77] have proposed a time series anomaly detection method based on recurrent neural networks (RNNs) and Q-learning. This approach does not rely on any assumptions about the underlying mechanism of anomaly patterns. Notably, the anomaly detector is threshold-free, meaning it functions as a logical classifier without the need for tuning a specific threshold. The model achieved a high accuracy rate of 100% in their evaluation. However, the approach is theoretical and requires further experimentation using realistic data sets to validate its effectiveness in real-world scenarios. Sengupta et al. [78] have proposed an RL model that integrates the Q-learning algorithm and the rough set theory (RST). They applied a modification of the Q-learning algorithm to automatically detect network intrusions. The main limitation of their study pertains to its adaptability to complex network states, as their model utilizes a table to store the Q values, which is not suitable for high-dimensional data distributions. In the existing body of literature on the application of RL in the IoT field, there are several areas requiring further exploration. One noticeable gap is the limited generalizability of RL models, which often struggle to adapt to varying or unseen scenarios.

This challenge underscores the need for future research to focus on developing models capable of learning more generalized strategies and enhancing their applicability across a variety of situations. Furthermore, the presence of a significant class imbalance in most IoT data sets, characterized by a greater number of normal behavior instances compared to anomalous or attack behaviors, is often overlooked. This imbalance can negatively impact the learning process and the ability of systems to detect rare but critical anomalies, emphasizing the need for more effective solutions to address this imbalance in future studies. Finally, existing RL models in IoT frequently rely on manually designed reward functions, introducing potential biases and limiting their learning efficiency. A shift toward automated or learned reward functions could pave the way for more robust and effective RL models.

III. Reinforcement Learning: Overview

In this section, a theoretical background to RL is presented. Following this, an in-depth discussion of Q-learning is provided. In particular, Q-learning is incorporated as a core component in our proposed approach.

A. Theoretical Background of Reinforcement Learning

One of the most active research areas in artificial intelligence is RL, a branch of ML that is rapidly growing. RL is predominantly used to solve sequence-decision tasks such as robotic navigation [79]. RL can be formulated as a Markov decision process (MDP) to solve decision sequence problems [80]. Typically, agents interact with an environment by observing a state and taking actions to exploit a reward over a discrete time period (t). Driven by the success of RL in other fields (in terms of generalization), such as gaming [81], [82], as well as the growing research on RL in cyber security [83], this article adapts RL to detect malicious botnet activities in an IoT environment. The detection process in this article is formulated as an MDP. MDP can be defined as a quintuple value [84]

$$S, A, R, T, \Upsilon$$

where S represents a finite set of possible states representing the world of the problem, whether discrete or continuous; A represents a set of actions the agent can perform on the environment; T is a transition method from one state to another, based on a probability distribution; R denotes a reward; and Υ is a discount factor, to balance the immediate reward with the long-term reward (the value of Υ lies between 0 and 1).

The agent uses a policy π to select different actions based on the state, whereby the policy represents a conditional probability distribution denoted as T (s,a,a'). Using this policy, the agent can navigate the environment and garner different rewards through each transition. The policy can also be represented as $\pi(a|s) = P(a_t = a|s_t = s)$. Given the policy π and a discount factor, the value of each state can be defined as follows [80]:

$$v_\pi = E_\pi \left(\sum_{t=1}^{t=\infty} \gamma^{t-1} R_t \, |S_1 = s \right). \tag{1}$$

While the value for the action–state pairs is as follows [85]:

$$q_\pi(s, a) = E_\pi \left(\sum_{t=1}^{t=\infty} \gamma^{t-1} R_t \, |S_1 = s, A_1 = a \right). \tag{2}$$

Using the Bellman equations [80], the value of the state–action pair could be rewritten in a recursive way as follows [86]:

$$v_\pi(s) = E_\pi \ (R_{t+1} + \ \gamma v_\pi(S_{t+1})|S_t \ = s) \tag{3}$$

$$q_\pi(s, a) = E_\pi \ (R_{t+1} + \ \gamma v_\pi(S_{t+1})|S_t \ = s, A_1 = a) \tag{4}$$

In each episode of learning, whenever a state exists (i.e., is not in a terminal state), the agent tries to maximize its rewards. At state (s), the reward is the sum of current rewards plus discounted rewards of the new states. This can be expressed as $R(s') = R(s, a, s') + \gamma * R(s')$. Furthermore, the agent can obtain the optimal values of the policy using a cyclic process by a trial-and-error approach of interaction with the environment. Model-based and model-free approaches are well-known methods for solving decision sequence tasks under uncertainty. The model-based approach relies on the transition probability function of the task, whereas the model-free approach relies on the interaction between the agent and the environment to determine the transition probability. Although model-based systems exhibit higher responsiveness compared to model-free systems, they necessitate prior knowledge concerning potential agent actions. In contrast, model-free systems, which learn from environmental interactions, are more suitable for accommodating dynamic attack behaviors due to their inherent adaptability.

B. Q-Learning

Q-learning uses the model-free approach, where every timestamp is assigned an experienced sample that allows the agent to estimate the Q-values. Bellman equations can be used to approximate Q-learning, which represents how effective a particular action is for an agent in a particular state, as follows [75]:

$$Q_{new}(s, a) = Q(s, a) + \alpha * \left(r + \gamma * Q^*\left(s', a'\right) - Q(s, a) \right) \tag{5}$$

where α is the learning rate, Q(s; a) is the current estimation, and Q^*(s',a') is the highest Q value among the possible actions that could be taken in the new state. In the Q-learning approach, the agent interacts with the environment and then obtains a trajectory (s,a,r,s1,a',r',..n) and stores it in a Q table, which the agent can use to reach the optimal policy. The size of the Q table depends on the numerical complexity of the problem that the agent wants to solve. However, this approach is not suited for the work presented in this article as it is only suitable for solving simple problems.

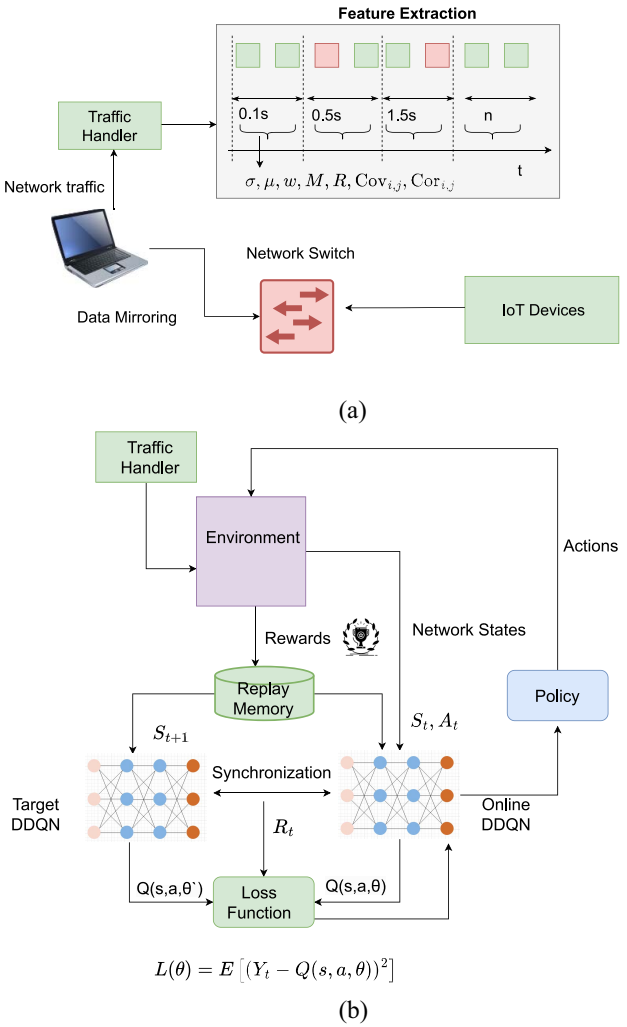

Fig. 1. MalBoT-DRL architecture. (a) Traffic handler. (b) IDS-engine.

IV. MALBOT-DRL SYSTEM

In this section, the primary components of the proposed system, namely the Traffic Handler and the MalBoT-DRL engine, are examined. The process from data collection to traffic classification is elaborated upon, with the model's architecture, its learning parameters, and the operations of the RL engine being detailed. Fig. 1 illustrates the MalBoT-DRL architecture.

A. Traffic Handler

The traffic handler component of the model is responsible for collecting network traffic between the IoT devices as raw PCAP data, and then extracting statistical temporal features of the network traffic. Since this article targets limited and constrained devices, the proposed model should be lightweight in terms of implementation (computational resources). Therefore, in this article, a damped incremental statistics method [87] was utilized to extract network traffic features. This method has the advantage of extracting features from dynamic network traffic at high speed, while only requiring $O(1)$ computational resources, as demonstrated in [88].

Suppose $P = \{f1, f2, \ldots, fn\}$ is an unbounded network traffic stream, where f represents the feature associated with every packet collected by the traffic handler. For each traffic stream, our handler will maintain an array $\forall^i_\beta = (N_m, L_p, S_p, L_pR_{ij}, T_l)$, where N_m represents the number of the network packets that were captured recently, L_p is the sum of the capture packets, S_p is the square sum of the network traffic and L_pR_{ij} represents the sum of residual products between two streams (i and j), defined as follows [87]:

$$L_pR_{ij} = \sum {}^r i \, {}^r j \tag{6}$$

$$L_pR_{ij} = \sum \left(f^i_{cur} - \frac{P}{Nm}\right) * \left(f^j_{cur} - \frac{P}{Nm}\right). \tag{7}$$

T_l represents the last timestamp that has been updated for $\forall^i_\beta$. The array $\forall^i_\beta$ will be incrementally updated using statistical values, such as the mean, variance, and standard deviation of P. When a new f is captured, $\forall^i_\beta$ will be updated as follows: $\forall^i_\beta = (N_m + 1, L_p + f, S_p + f^2)$. The old packets must be discarded in order to extract the current status of the packet stream. The incremental statistics method adds a decay function in the following manner [87]:

$$d_\beta(t) = e^{\beta t}. \tag{8}$$

$\beta t > 0$ represents a default value, and t represents the time since the last observation from the stream. The new $\forall^i_\beta$ will be updated as follows.

1) Calculate the decay factor $\beta \leftarrow e^{\beta(T_n - T_l)}$, where T_n is the current timestamp.
2) Process the decay as $\forall^i_\beta \leftarrow (\beta N_m, \beta L_p, \beta S_p, \beta L_pR_{ij}, T_l)$.
3) Insert the new value as $\forall^i_\beta \leftarrow (N_m{+}1, L_p{+}f, S_p{+}f^2, T_n)$.
4) When both incoming and output packets require 2-D statistics, calculate L_pR_{ij} for each variable, as follows [87]:

$$L_pR_{ij} = \beta L_pR_{ij} + \sum \left(f^i_{cur} - \frac{P}{Nm}\right) * \left(f^j_{cur} - \frac{P}{Nm}\right). \tag{9}$$

5) Return $\forall^i_\beta$.

By utilizing $\forall^i_\beta$, we can calculate 1-D statistics of the network traffic sequence f_i. The statistics include weight (w), mean (μ), and variance (σ^2). μ and σ^2 are calculated as follows [88]:

$$\mu = L_p / w \tag{10}$$

$$\sigma^2 = \left| \frac{S_p}{w} - \mu^2_{f_i} \right|. \tag{11}$$

While the 2-D statistics of f_i and f_j include the magnitude (M) $\|f_i, f_j\|$, radius (R_{f_i, f_j}), covariance (Cov_{f_i, f_j}), and the correlation coefficient (Cor_{f_i, f_j}). The 2-D statistics are calculated as follows [88]:

$$||{}^f i, {}^f j|| = \sqrt{f_i^2 + f_j^2} \tag{12}$$

$$R_{f_i, f_j} = \sqrt{\left(\sigma_i^4 + \sigma_j^4\right)} \tag{13}$$

$$\mathrm{Cov}_{i,j} = \frac{L_pR_{ij}}{w_i + w_j} \tag{14}$$

Algorithm 1 Traffic Handler

Input: Network traffic (P)
Output: Preprocessed network traffic
Time windows = {0.1s, 0.5s, 1.5s, 10s, 60s}
Traffic aggregating
Remove duplicate packets
if packet in time windows **then**
    Extract statistical features (μ, σ, w, M, R, $\text{Cov}_{i,j}$, $Cor_{i,j}$)
    Data scaling using $Scale(f) = \frac{f-\min(f)}{\max(f)-\min(f)}$
end if
Split the preprocessed traffic to: 60% training, 10% validation, 30% testing

TABLE II
EXTRACTED FEATURES

Feature no.	Description	Source
F1-8	**Statistics:** σ, μ **Feature:** Packet size	SrcIP/SrcIP-MAC/Socket/Channel
F9-12	**Statistics:** w **Feature:** Packet count	SrcIP/SrcIP-MAC/Socket/Channel
F13-15	**Statistics:** μ, σ, w **Feature:** Packet jitter	Channel
F16-23	**Statistics:** M, R, $Cov_{i,j}$, $Cor_{i,j}$ **Feature:** Packet size	Socket/Channel

$$Cor_{i,j} = \frac{\text{Cov}_{i,j}}{\sigma_i + \sigma_j}. \tag{15}$$

For each packet captured during a given time window, the feature extractor checks the Netflow information including the sender IP address (Src-IP), MAC-IP address, sender–receiver IP (Channel), and the sender–receiver TCP/UDP socket. The flow is subsequently based on this information. Traffic handlers are capable of extracting 23 statistical features for each time window. This research only considers five temporal time windows to compare the proposed model with other SOTA models: {0.1 s, 0.5 s, 1.5 s, 10 s, 60 s}. The extracted features are summarized in Table II. A data normalization method was applied to prevent the model from being biased to the normal traffic. The network features in the final stage of the preprocessing phase are split into training, validation, and testing, using a ratio of 60:10:30. Traffic collection and network traffic processing are elaborated in Algorithm 1.

B. MalBoT-DRL Engine

This section is composed of two parts: the first details the engine components, and the second explains the learning process.

1) Engine Components: In this article, we introduce a robust method for detecting a variety of network attacks, employing a model-free technique, Q-learning and a DNN architecture. The system also incorporates an attention reward function to address the data imbalance issue associated with minor attack samples, thereby improving classification accuracy. Unlike previous studies [70], [72] that utilized adversarial RL strategies, our approach is more streamlined as it eliminates the need for an adversarial agent. Furthermore, it can identify the attacking behavior at both early and later stages of an attack's life cycle, offering a broader scope of detection. This approach also considers generalizability across different environments, an aspect that is often overlooked in similar works.

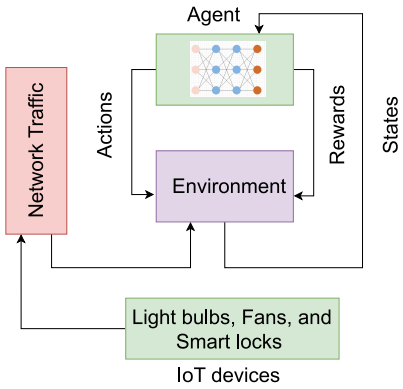

Fig. 2. MalBot-DRL interaction with incoming traffic.

In the Q-learning process, the agent's experiences are captured, and their behavior is guided by reward functions. Each action performed by the agent leads to a new state and a reward, which then prompts the agent to carry out a new action. IoT devices generate workload, which is directed into a switch and then sent to a traffic analyzer via a local-area network or a wide-area network, as shown in Fig. 2.

Given that the Q-learning table cannot handle high-dimensionality state space environments [75], a neural network (NN) is employed as a function approximator to manage network traffic complexity. The integration of Q-learning with DNN results in a new model, the DQN. In DQN, the DNN approximates the Q-value for every action based on a state (s), instead of storing the Q-values in a table. This RL approach uses two networks, online DQN, and target DQN, to stabilize the learning process [75], [89]. The shared architecture of the online and target networks includes three hidden layers with 256, 64, and 32 neurons, respectively. A rectified linear unit (ReLU) activation function is used, with a learning rate of 0.001. Optimization is achieved using the Adam optimizer.

The online DQN is represented by $Q_{(s,a,\theta)}$, where the parameters of the model are state, action, and θ. θ denotes the NN weights for every layer in the Q-network at a specific time t. The online DQN is utilized to estimate the Q-values for each action (a).

The target DQN is represented by $Q_{(s',a',\theta')}$, and is used to calculate the value of the next state and action (s',a). Furthermore, it is used to train the parameters of the online DQN.

The online NN is responsible for the training process. During the learning process, the target DQN is frozen for a

few iterations, then the online DQN transfers its experience to the target network by synchronizing the weights (θ and θ'). This process stabilizes the training process and makes the estimations of the Q-value more accurate [90]. The Q target values can be represented in the DQN as follows [91]:

$$Q_{(s,a,\theta)} = r_t + \gamma \max Q_{a_{t+1}}\left(s_{t+1}, a_{t+1}, \theta'\right). \tag{16}$$

To optimize the learning process of the NN, a gradient descent approach is utilized to minimize the loss (or cost) function. By squaring the difference between the target and the online networks, the loss function can be calculated as follows [92]:

$$L(\theta) = E\left[\left(Y_t - Q_{(s,a,\theta)}\right)^2\right] \tag{17}$$

where Y_t represents the target value $Y_t = r_t + \Upsilon \max Q_{a_{t+1}}(s_{t+1}, a_{t+1} + \theta')$. The target DQN values are determined by combining the immediate reward r' and the maximum value of $Q_{(s',a')}$, obtained from inputting s' into the target DQN, Thus, Y_t can be further expanded as follows [92]:

$$Y_t = r_{t+1} + \Upsilon Q\left(s_{t+1}, \text{argmax} Q\left(s_{t+1}, a_t; \theta'\right); \theta'\right). \tag{18}$$

The proposed MalBoT-DRL is represented by an environment, space of system actions, space of actions, and reward function. These are detailed as follows.

1) *Environment:* The environment of the proposed model is established after preprocessing and normalizing the network traffic stream. The feature vector of the network traffic represents the states in our model. The feature vectors are temporal window features, where the states are represented by the statistical features multiplied by the time windows of the network stream. The ground truth samples are then used to compute the reward vector (depending on the prediction of the model).
2) *Agent:* The MalBoT-DRL model incorporates a DQN-RL agent for estimating the discounted future rewards post-model training. The agent interacts with the environment and receives rewards based on the actions taken, corresponding to different states. During the training phase, the agent explores the actions and the environment, adhering to a designated policy. The exploration and exploitation strategy employed by the MalBoT agent is ε, which is widely considered to be SOTA [93]. The agent selects a random action with a probability of ε, or selects the best action based on a probability of $1 - \varepsilon$. The value of ε at the start of the training is a high value to give the agents a chance to explore their actions. The goal of our agent is to maximize the expected future return (EFR) as follows [92]:

$$\text{EFR} = \sum_{c=0}^{i} \gamma^c r_{t+i}. \tag{19}$$

3) *States:* In the proposed model, the states refer to input values from the environment that the agent can observe and utilize for decision making. More specifically, the environment is characterized by the network traffic captured by the traffic handler, with the network stream features serving as the state parameters for the DDQN agent. At any given time t, the network state, denoted by S, is defined by the source and destination IoT nodes, along with the temporal and spatial features associated with the network traffic. These network features are captured by the IDS, represented as: S={s1=infection/propagation, s2=command execution, s3=normal communication}, where s1 means the IoT node is infected and the malware started the propagation phase, s2 indicates that the infected device is communicating with the botmaster, and s3 means that the IoT device is communicating with another node with a legitimate communication.
4) *Actions:* During a given time window, the agent/IDS interacts with the environment and makes a sequence of decisions, called actions. The DDQN agent processes a list of states called a minibatch and then generates a list of action vectors based on the NN that processes the input features. The agent determines the nature of the traffic according to the current network state. The action space is defined as $A = \{\text{Qv, argmax Qv}\}$, where Qv is a prediction of s for all actions that belong to the total number of random actions, and is fed to an action vector (Av = argmax Qv). Av in this article is calculated using the greedy strategy, where the value is estimated by $\text{argmax} Q(s_t, a_t)$.
5) *Rewards:* The agent performs an action then feedback is received from the environment, in the form of a reward. If the agent makes a correct action—defined by matching with the estimated Q value, it will receive a positive reward. In contrast, a negative reward may be received. It is challenging to identify attack samples accurately in an environment with imbalanced network streams. As a result, it is necessary to increase the algorithm's sensitivity to minority samples in order to improve the detection of subsequent minority attack samples. Whenever an agent meets a minority sample, it receives a large reward or punishment [94]. The attention reward mechanism has been utilized in our model to handle the data imbalance issue of the less frequent samples. In the MalBoT model, the reward function is defined as follows:

$$R(s,a) = \begin{cases} P_r, \text{ For } a_t = G_t \text{ and } s_t \in A_s \\ N_r, \text{ For } a_t = G_t \text{ and } s_t \in A_s \\ \lambda, \text{ For } a_t = G_t \text{ and } s_t \in N_s \\ -\lambda, \text{ For } a_t = G_t \text{ and } s_t \in N_s \end{cases} \tag{20}$$

where A_s represents infection flows or malicious communication with the C&C server, N_s represents legitimate communication between two nodes, Pr represents a positive reward (value = 1), Nr represents a negative reward (value = –1), a_t is an action taken by the agent, Gt is the ground truth of the traffic flow, and λ represents attention rewards, where a value of λ can be calculated using the imbalance ratio (number of malicious flows/number of legitimate communication). The agent will receive +1 as a reward value when correctly predicting the attack states, while receiving a negative reward when it predicts attack states incorrectly. The agent will receive *attention rewards* when it predicts normal traffic correctly. The value of λ will be 1 if the

network traffic in the environment is balanced. In this case, all classes will have the same reward or punishment based on the corrections of the action. Based on this approach, the agent will place more attention on the minor samples. This article evaluated the MalBoT model using different imbalance ratios to test the robustness of the *attention rewards* for the anomaly detection problem, as well as to test the generalization of the proposed model.

2) Learning Process in MalBoT-DRL: The training procedure of the proposed model is broken down into the following phases: initialization phase, sampling phase, training phase, experience replay phase, and overestimation avoidance phase.

Initialization Phase: The learning process begins with the initialization of the state, environment, action and NN parameters of the online DQN. The traffic handler fetches the current network state and generates tailored state data for the model based on the traffic flow generated by the end device. The input of a NN consists of the state s_t, while the output consists of the function value. An action value can be represented using DQN.

Sampling Phase: The agent produces actions and the observed experiences are stored in the reply memory (D). The agent follows the greedy policy, where a random action is selected with a probability of ε, while the best action will be selected with $1-\varepsilon$ probability. At the start of the training, an ε value is set to (1) to make the agent explore the environment, then the value is gradually decreased based on a decay rate. By utilizing the ε-greedy policy, the agent explores all possible actions, from which it selects the optimal action.

Training Phase: A random batch of samples is selected to learn the attack behaviors through a gradient descent update. By selecting random samples as minibatches, samples used in the online network are not biased toward recent experiences. Consequently, correlation observation sequences are removed, and oscillating action values are prevented. DQN uses two networks to avoid instability of training. A major cause of the instability issue is the combination of bootstrapping and the nonlinear value function (NN). The bootstrapping method involves updating the target based on an estimate, not the actual data. The training phase has also been stabilized by using experience replay and double deep Q-learning.

Experience Replay Phase: The agent interacts with the environment and then obtains experiences (s,a,r,s'), which are updated and inputted into the NN, after which their values are discarded. To leverage past experiences, a replay buffer is created within the experience replay model, storing the experience and reusing it during training. Replay buffers allow us to leverage the same experience multiple times.

Overestimation Avoidance Phase: The overestimation problem occurs when the agent wants to select the optimal action for the next state (s'). The agent selects the action with the highest Q-value. This choice is based on the experience that the agent had and on its exploration of neighboring states. However, the agent in the beginning of the training has insufficient information regarding the best action; thus, selecting the best action based on the maximum Q-value leads to high FP rates. To avoid this problem, we apply the same approach used by [75], which is known as Double DQN. The agent computes the Q target using two distinct networks. These networks are employed to separate the action selection process from the target Q value generation. For the next state s', the action with the highest Q value is determined by the DQN network. This action a is then used by the target network to calculate the target Q-value at s'. DDQN's target value can be expressed as follows [92]:

$$Y_t = r_{t+1} + \Upsilon Q\left(s_{t+1}, \text{argmax}Q(s_{t+1}, a_t; \theta); \theta'\right). \quad (21)$$

The learning process of the DDQN agent is described in Algorithm 2.

Algorithm 2 Deep Q-Learning Algorithm

Parameters: W: network weight, B: bias, GD: gradient descent
Input: Preprocessed network traffic
Output: F1, Precision, Recall, G-mean, Accuracy
Initialize the online DQN with random W and B as θ;
Initialize the target DQN as a copy of the online DQN W and B as θ';
Initialize replay memory M;
for $k = 1$ to Max Episode **do**
    Initialize state st;
    Input the system state st into the online DQN;
    Compute the Q-value based on online DQN;
    Based on the probability ε, choose an action;
    Execute action at, receive a reward rt and observe the next state $st + 1$;
    Store the interaction tuple in M;
    for $o = 1$ to MaxStep **do**
        Sample a random batch from M;
        Compute the target Q value: $Y_t = r_{t+1} + \Upsilon Q(s_{t+1}, \text{argmax}Q(s_{t+1}, a_t; \theta); \theta')$;
        Train the NN to minimize the loss function: $L(\theta) = E[(Y_t - Q(s, a, \theta))^2]$;
        Execute GD with respect to θ;
        Update target DQN every N steps based on DDQN: $\theta' \leftarrow \theta$;
    end for
end for
Calculate the performance metrics using the testing network traffic;

V. Model Evaluation

In this section, the evaluation process of the proposed model is delineated. First, the simulation environment is presented, followed by the criteria for data set selection and the chosen evaluation metrics. The evaluation in the early phase of the malware botnet life cycle is discussed, taking into consideration realistic scenarios that encompass both compromising single devices and multiple devices simultaneously. Subsequent evaluation focuses on the later stages of the malware botnet life cycle, wherein stealthy attacks, such as data exfiltration, are considered. Lastly, a complexity analysis is

TABLE III
SPECIFICATIONS OF HARDWARE AND SOFTWARE IMPLEMENTATION ENVIRONMENTS

Item	Description
Processor	Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.60 GHz
Memory	16.0 GB
OS	64-bit operating system, x64-based processor
Python	3.8
Gym	0.23.0
Tensor flow	2.11.0

TABLE IV
DETAILED PARAMETERS OF THE PROPOSED MODEL

Parameter	Description	Value
N-episode	Number of episodes	160_000
Warm-up	Random actions	16000
cspe	Collect steps	2000
E	Exploration method	1
tup	Update periods	800
min_ ε	Epsilon value	0.02
LR	Learning rate	0.001
Gamma	Discount factor	0.001
Batch-size	Model update	32
Input	Traffic stream	115
H-Layer1	First layer	256
H-Layer2	Second layer	64
H-Layer3	Third layer	32
activation	Activation Function	Relu

provided, alongside model requirements. The specifications of our model are then compared with other SOTA models.

A. Simulation Environment

The list of tools and libraries used during the implementation and evaluation phases is listed in Table III. To evaluate the adaptability of our model, we tested it using multiple scenarios. A performance evaluation of the MalBoT-DRL model was conducted using the parameters listed in Table IV. Various discount factor values were used to test the behavior of the proposed model. Based on the results, it appears that a higher discount factor produces a higher loss value and a lower performance. Therefore, a lower discount factor (gamma = 0.001) was selected based on the conducted experiments.

B. Data Set Selection

To evaluate the proposed model using authentic scenarios, the data set adapted for this study was selected with the following criteria in mind.

1) The data set should be compiled from a diverse range of IoT devices and should encompass normal activities, including traffic types, such as TCP and UDP.
2) A new variety of malware with stealth properties should be included in the data set.
3) All stages of the deployment life cycle of the IoT botnet should be included in the data set.
4) Malicious traffic including spreading samples, communication samples, flooding attacks as well as spamming should be included in the data set.

TABLE V
SUMMARY OF THE USED DATA SETS

Dataset	Malware type	# of data flows	# of devices
N-BaIoT	Bashlite ,Mirai	6272960	9
MedBIoT	Bashlite ,Mirai, Torii	17,845,567	89

MedBIoT [95] and N-BaIoT [96] data sets met these requirements and have been extensively used for IoT botnet detection. Table V provides a comprehensive summary of the data sets, detailing the number of flows, devices involved, and types of malware.

C. Evaluation Metrics

Numerous evaluation metrics are considered for assessing the effectiveness of the proposed model. The model has been evaluated using true-positive (TP), FP, true-negative (TN), and false-negative (FN) values, defined as follows.

1) *TP:* The proposed IDS correctly predicts the malicious samples.
2) *FP:* The proposed IDS wrongly predicts the malicious samples.
3) *TN:* The proposed IDS correctly predicts the normal samples.
4) *FN:* The proposed IDS wrongly predicts the normal samples.

Based on these metrics, the accuracy, precision, recall, $F1$-score, and G-mean are calculated as described below.

Accuracy (A): The total number of correctly predicted values over the total number of predication, represented by

$$A = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}. \tag{22}$$

Precision (P): The ratio of correctly predicted malicious samples to all samples that the model predicted as malicious

$$P = \frac{\text{TP}}{\text{TP+FP}}. \tag{23}$$

DR/Recall (R): The ratio of correctly predicted malicious samples to all samples that were actually malicious, regardless of whether the prediction was correct

$$DR = \frac{\text{TP}}{\text{TP+FN}}. \tag{24}$$

*F1:*The harmonic value of the precision and recall

$$F1 = \frac{2*\text{TP}}{2*\text{TP+FP+FN}}. \tag{25}$$

G-Mean(G): Geometric mean of sensitivity and precision

$$G - \text{mean} = \sqrt{\frac{\text{TP}*\text{TN}}{(\text{TP+FN})*(\text{TN+FP})}}. \tag{26}$$

D. Early-Stage Malware Activities Detection

In our early detection evaluation, experiments encompassed all conceivable attacks across the botnet life cycle, focusing particularly on the initial stages. We considered three prevalent malware: 1) Bashlite; 2) Mirai; and 3) Torii (as depicted in Table VI). These early-stage activities involve the hacking of IoT devices, further spreading to compromise other devices, and initiating communication with the C&C server.

Initially, the study postulated scenarios where individual IoT devices, represented by a smart fan, were compromised. In the

TABLE VI
SUMMARY OF EVALUATION SCENARIOS BASED ON EARLY MALWARE LIFE CYCLE

Scenario	Devices	Malware	Activity	Normal	Attack	P (%)	R (%)	F1 (%)	G (%)	TP	FN	TN	FP	λ
1	Single	Bashlite	SP	300000	300000	99.99	99.96	99.97	99.97	**89966**	34	89998	2	1
1	Single	Bashlite	C&C	300000	188852	98.88	99.08	98.98	98.98	56135	521	89362	638	0.63
2	Single	Mirai	SP	300000	119827	99.87	99.85	99.86	99.86	35895	53	89956	45	0.40
2	Single	Mirai	C&C	300000	296642	99.99	99.95	99.97	99.97	88946	47	89996	4	0.99
3	Single	Torii	SP	300000	137476	99.99	99.95	99.97	99.97	41223	20	89999	1	0.46
3	Single	Torii	C&C	300000	46222	**100.00**	**99.98**	**99.99**	**99.99**	13864	**3**	90000	**0**	0.15
4	Multiple	Bashlite	Mixed	700000	300000	97.02	88.14	92.37	92.54	79333	10667	207561	2439	0.43
5	Multiple	Mirai	Mixed	700000	300000	99.16	99.84	99.50	99.50	89859	141	209244	756	0.43
6	Multiple	Torii	Mixed	700000	300000	99.67	99.83	99.75	99.75	89850	150	209703	297	0.43
7	Multiple	All	C&C	700000	300000	97.15	82.79	89.29	89.83	74513	15487	207836	2164	0.43
8	Multiple	All	SP	700000	300000	99.99	99.97	99.98	99.98	89970	30	**209996**	4	0.43
9	Multiple	All	Mixed	700000	300000	99.37	99.83	99.60	99.60	89850	150	209430	570	0.43

first three scenarios, these devices were individually infected with one of the three malware types: 1) Bashlite (Scenario 1); 2) Mirai (Scenario 2); or 3) Torii (Scenario 3). Each scenario was dissected into two primary malicious activities-malware spreading (SP) and illicit C&C server communication. Specific metrics and findings for these scenarios can be referenced in Table VI. The results indicated a specifically high efficiency of the model, albeit with some challenges, especially with the Bashlite malware, which bore a resemblance to normal communication.

As we expanded our scope, Scenarios 4 through 6 investigated the behavior when multiple devices were infected. These devices, namely a fan, light, lock, and switch, were subjected to Bashlite (Scenario 4), Mirai (Scenario 5), or Torii (Scenario 6). Notably, while the model's efficiency with Bashlite presented challenges, its efficiency notably improved when dealing with Mirai, with fewer FN samples recorded. The confusion matrices (CMs) detailing these activities are presented in Table VI.

In Scenarios 7 and 8, all four IoT devices were exposed to a concurrent infection of multiple malware types: Bashlite, Mirai and Torii. Scenario 7 predominantly focused on the spreading aspect of these malwares, while Scenario 8 concentrated on their C&C communication methods. These settings were developed to highlight the intricate threat environment facing IoT devices, with adversaries deploying a combination of malware to facilitate different malicious operations. The performance metrics of the model for these specific scenarios are detailed in Table VI. Lastly, in Scenario 9, the assumption made was that all IoT devices are infected by all three malware types. Using a combination of 300 000 attack and 700 000 normal traffic samples, MalBoT-DRL demonstrated significant proficiency, optimizing the detection process and minimizing FP samples. The relevant CM for this scenario can be found in Table VI.

Discussion: In Table VI, and Fig. 3 the proposed model demonstrates efficacy in detecting a majority of malware activities, including the initial stages of a botnet life cycle, such as propagation and communication with C&C servers. Nevertheless, the model encounters difficulties in handling Bashlite malware. Bashlite malware presents a unique challenge for detection systems due to its distinct characteristics, employing a P2P network based on the UDP protocol, which is characterized by low latency but limited reliability.

TABLE VII
SUMMARIZED EVALUATION SCENARIOS BASED ON LATER MALWARE LIFE CYCLE

Scenario	Devices	Malware	Activity
1	Single	Bashlite, Mirai	Case1-12: SP
2	Multiple	Bashlite	SP
3	Multiple	Mirai	SP
4	Multiple	Bashlite, Mirai	C&C

Consequently, detecting and blocking malicious traffic associated with Bashlite malware becomes a complex task for these systems. Furthermore, Bashlite malware adopts advanced evasion techniques, including the randomization of C&C server addresses and obfuscation of its network traffic to resemble legitimate traffic patterns. These factors pose considerable challenges for detection systems, such as the proposed model, to accurately identifying and mitigating Bashlite malware, particularly during the critical early phases of a botnet life cycle.

E. Late-Stage Malware Activity Detection

In this phase of the evaluation, the emphasis is placed on post-attack activities. More specifically, the (C&C) server sends commands to the compromised devices to initiate volumetric attacks, including spamming and flooding attacks. By examining these subsequent actions, the study seeks to provide a comprehensive understanding of the malware's life cycle and its impact on the targeted IoT devices. This is expected to ultimately inform the development of more effective mitigation strategies for post-attack scenarios. Data splitting for evaluation in the later stages of the malware life cycle follows the same distribution as that of the earlier malware activities. This consistent approach ensures that the assessments accurately reflect the malware's behavior patterns, enabling a comprehensive understanding of its progression from initial infection to subsequent actions. The evolution phase follows four scenarios (Table VII).

1) The IoT devices are individually infected by Bashlite and Mirai malware.
2) Different normal profiles of the IoT activities with Bashlite malware.
3) Normal profiles with Mirai malware.
4) A group of IoT devices, that are infected by Bashlite and Mirai, launching different attacks.

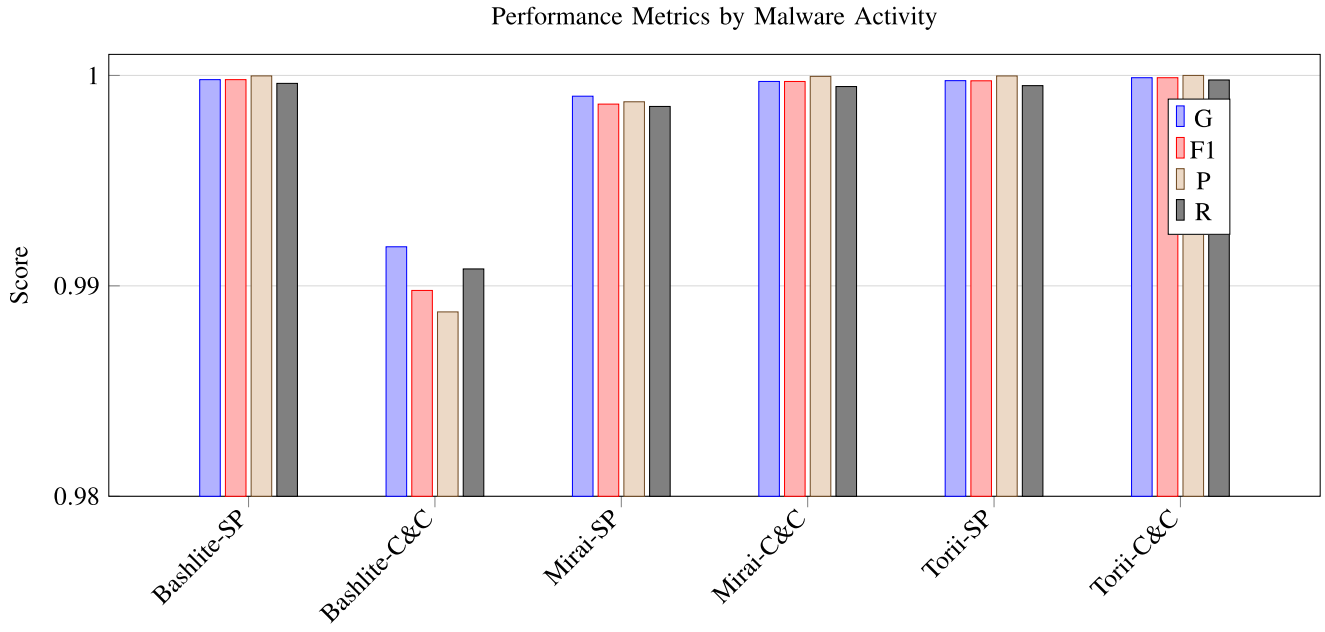

Fig. 3. Comparison of different malware activities using single IoT device.

TABLE VIII
DATA SET PROPERTIES, TRAINING SUMMARY, AND MODEL EVALUATION FOR LATE STAGE MALWARE BOTNET ACTIVITIES

Scenario	Case	IoT device	Malware	Attack	Normal	G(%)	F1(%)	P(%)	R(%)	TP	FN	TN	FP	λ
1	1	Doorbell	Bashlite	316650	49548	99.99	99.99	**100**	99.99	14862	2	14865	**0**	1
1	2	Doorbell	Mirai	652100	49548	**100**	**100**	**100**	99.99	14863	1	14865	**0**	1
1	3	Thermost	Bashlite	310630	13113	99.95	99.95	**100**	99.90	3930	4	3934	**0**	1
1	4	Thermost	Mirai	512133	13113	**100**	**100**	**100**	**100**	3934	**0**	3934	**0**	1
1	5	Ennio Door	Bashlite	316400	39100	99.99	99.99	**100**	99.97	11727	3	11730	**0**	1
1	6	B120N10	Bashlite	312723	175240	99.96	99.96	99.95	99.96	52551	21	52546	26	1
1	7	B120N10	Mirai	316400	39100	**100**	**100**	**100**	**100**	52571	1	52572	**0**	1
1	8	PT_737E	Bashlite	330096	62154	99.99	99.99	**100**	99.98	18642	4	18647	**0**	1
1	8	PT_737E	Mirai	436010	62154	99.97	99.97	**100**	99.95	18636	10	18646	**0**	1
1	9	PT_838	Bashlite	309040	98514	99.98	99.98	100	99.95	29540	14	29554	**0**	1
1	9	PT_838	Mirai	429337	98514	99.99	99.99	**100**	99.97	29546	8	29555	**0**	1
1	10	Webcam	Bashlite	323072	52150	99.99	99.99	99.98	99.99	15644	1	15642	3	1
1	11	XCS7_1002	Bashlite	303223	46585	**100**	**100**	**100**	99.99	13974	1	13975	**0**	1
1	11	XCS7_1002	Mirai	513248	46585	**100**	**100**	**100**	**100**	13975	**0**	13975	**0**	1
1	12	XCS7_1003	Bashlite	316438	19528	**100**	**100**	**100**	**100**	11717	**0**	11717	**0**	1
1	12	XCS7_1003	Mirai	19528	19528	99.99	99.99	100	99.98	5857	1	5858	0	1
2	-	Mixed	Bashlite	300000	700000	99.95	99.93	99.91	99.94	**166685**	95	**333413**	146	0.5
3	-	Mixed	Mirai	300000	700000	99.99	99.99	99.99	99.99	83382	8	166777	3	0.5
4	-	Mixed	Mixed	300000	700000	99.97	99.95	99.93	99.98	89977	23	166716	64	0.54

In Scenario 1, individual IoT devices, as listed in Table VIII, were examined when infected by Bashlite and Mirai malware. With balanced samples for evaluation, devices ranging from doorbells to thermostats were subjected to infections. It was found that the doorbell struggled against Bashlite but responded better to Mirai infections. Likewise, when Bashlite infected a thermostat, the device found it challenging to cope; however, the Mirai malware was easily detected by the same device. Expanding the scope to IoT security cameras, particularly in cases 8 and 9, the results indicated a distinct ability to recognize Bashlite over Mirai attacks. These findings are summarized in Table XI. Transitioning to Scenario 2, the stealth attacks by Bashlite malware, which involved diverse activities like spamming, data exfiltration, and flooding, were explored. Employing a million samples for training and testing, the model achieved a detection rate of 99.94% and a precision of 99.91%. In scenario 3, the blend of normal profiles from the N-BaIoT data set with malicious patterns from Mirai malware formed the backdrop. Out of the million samples used for evaluation, 70% were normal, leading to an impressive precision rate of 99.91% and a similar detection rate. This scenario was particularly noteworthy for the significant reduction in FP samples. Scenario 4 was crafted to mimic a more complex attack

TABLE IX
REWARD ATTENTION COMPARISON

Scenario	TP	TN	FP	FN
Without-Early	89563	209381	619	437
With-Early	89850	**209430**	570	150
With-Late	**89977**	166716	**64**	**23**
Without-Late	89790	165929	851	210

environment where both Bashlite and Mirai malware were active. By employing various normal profiles and analyzing them under this multimalware paradigm, the study intended to unveil the intricate challenges that arise from such configurations. The model was tested using 855 932 samples, with 65% being normal. Subsequently, it showcased a precision and detection rate both nearing 99.91%. The crucial aspect of this scenario was the low FP rate, indicating the model's potential in fortifying the IoT device security landscape while ensuring efficient resource utilization.

Discussion: As demonstrated in Table VIII, the proposed model exhibited a high level of precision. Out of the sixteen experiments conducted, the model detected all FP samples in fourteen of them. This result indicates that, if deployed in a real-world environment, the model has the potential to reduce the cost and time required for validating these samples. Regarding the detection of attack samples, the model was able to detect all of them in only three cases, and in three other cases, it missed fewer than four FN samples. In five cases, the model only missed one sample. Through our experimentation, we observed that the majority of the missed samples originated from UDP traffic, particularly in the case of Bashlite malware. This finding highlights the need for continued research and development to improve the effectiveness of malware detection models.

F. Impact of Attention Mechanism on Model Performance

Our model is equipped with an attention mechanism that has been used to deal with the issue of data imbalance by making the model more sensitive to minor classes. In order to accomplish this, we provided these samples with more rewards and fewer punishments. To illustrate the impact of the attention reward mechanism, the model was evaluated with and without the attention reward mechanism using 700 000 training and 300 000 testing samples during the early life cycle of the botnet malware.

The model without attention reward produced 619 FP samples and 437 FN samples in the early stages. In contrast, when employing attention reward in our model, the number of FP samples was reduced by 49. The FN samples decreased substantially from 437 samples to 150 samples. The improvement in performance is depicted in Fig. 4(a) and in Table IX.

The model was also evaluated in the late stage of the malware life cycle. By utilizing the attention reward mechanism, the FP samples were reduced, as illustrated in Fig. 4(b). Furthermore, the number of FN alerts dropped from 210 to 23. Considering the results of the experiment, we can conclude that the attention mechanism improves the detection rates and reduces the number of FP samples.

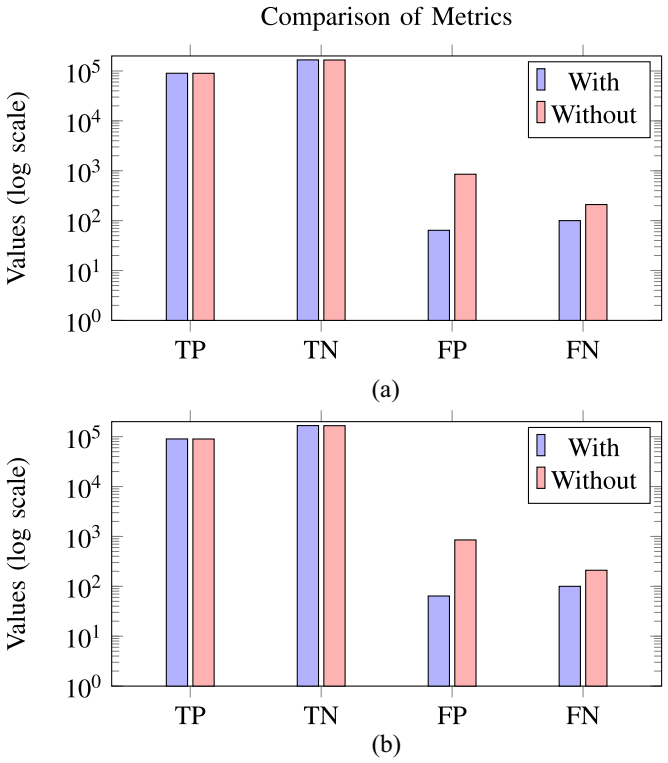

Fig. 4. Assessing reward attention during (a) early and (b) later stages of detection.

G. Complexity Analysis & Model Requirements

The proposed system employs two DNNs: 1) the online network and 2) the target network. Since the target DQN is updated periodically by adopting the parameters from the online DQN, the computational complexity of the online network is predominantly considered in our analysis. The online DQN includes the input layer (l_1), three fully connected layers (l_2, l_3, and l_4), and one output layer (l_5). An online network's complexity can be expressed as

$$|l_1| * |l_2| + |l_2| * |l_3| + |l_3| * |l_4| + |l_4| * |l_5|$$

where $|l_i|$ represents number of the neurons of layer i. Each training step involves taking a random sample from the replay memory and feeding it to the online DQN. Therefore, the training process has an overall complexity of

$$O = \left(T * N_b * \left(|l_1| * |l_2| + |l_2| * |l_3| + |l_3| * |l_4| + |l_4| * |l_5|\right)\right)$$

where T represents the total number of training cycles and N_b is the number of training batches. According to our study, l_1 is defined as the number of state features in the state space, so l_1 has a value of 115. There are 256, 64, and 32 neurons for all hidden layers. Each neuron in the output layer corresponds to one of the AV's actions, as explained previously. The online network is clearly based on a simple architecture; consequently, it can be applied to detection systems with adequate computing resources.

In this study, the performance of the proposed MalBoT-DRL model is compared with two other models, namely Parallel Learning [45] and CNN-LSTM [41], in terms of their resource requirements and processing times during both training and

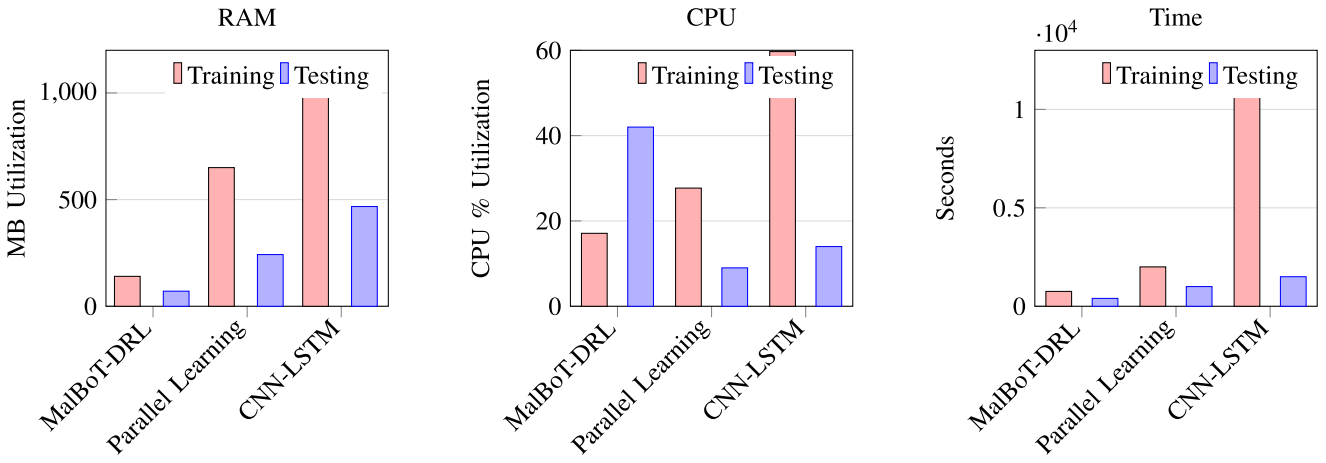

Fig. 5. Comparison of resource requirements and processing times of different models (Training versus Testing).

testing phases using psutil library [97]. The comparison is illustrated in Fig. 5. During the training phase, the proposed MalBoT-DRL model demonstrated the most efficient resource usage. It required an "additional" 140.3 MB of RAM and induced a 17.1% increase in CPU usage, completing the training process in approximately 12.6 min. The term "additional" here refers to the extra amount of RAM that is needed by the model over and above the amount that is already being used by the system for other tasks. This is important to consider because the testing device or system is typically running other tasks in the background, and these tasks also require a certain amount of RAM and CPU usage. In contrast, the Parallel Learning model requires an additional 650 MB of RAM and increases CPU usage by 27.7%, but has a longer training time of approximately 21.5 min. The CNN-LSTM model exhibits the highest increase in CPU usage (59.7%) and the longest training duration (approximately 208.2 min). Interestingly, the CNN-LSTM model requires an additional 1.1 GB of RAM during the training phase. In the testing phase, the proposed MalBoT-DRL model continued to demonstrate efficient resource usage. It required an additional 6.8 MB of RAM, exhibited a CPU usage of 42% during testing, and completed the testing process in approximately 8.4 s. The Parallel Learning model, while requiring less additional RAM than the CNN-LSTM model and demonstrating lower CPU usage, completed the testing in the shortest time of approximately 76.4 s. The CNN-LSTM model exhibited the highest CPU usage during testing (80.1%) and the longest testing duration (approximately 330.8 s), whereby its RAM usage increased the most during testing. These findings underscore the robustness of the proposed MalBoT-DRL model, which consistently requires fewer additional resources and exhibits lower CPU usage during both training and testing phases. This makes it a suitable choice for deployment in environments where resource constraints are a concern, such as in IoT devices. Furthermore, considering the number of network flows processed during the study, which was 599 152, it is evident that each network flow requires less and less resources, further emphasizing the efficiency of the proposed MalBoT-DRL model. More specifically, the time taken per network flow during training was approximately 0.00126 s, and during testing, it was approximately 3.266e-05 s/sample. This demonstrates the model's ability to process a large number of network flows efficiently and quickly, further enhancing its suitability for real-time applications in resource-constrained environments.

VI. Discussion and Limitations

To further evaluate the effectiveness of the proposed deep RL (DRL) model in detecting malware botnet activity in IoT environments, a comparative analysis was conducted with SOTA ML, DL, and RL models. The performance of the proposed model during different stages of the malware life cycle was investigated through this evaluation. To ensure a fair comparison, all of the models were evaluated based on the same number of samples. This was achieved by utilizing published codes when available, or recreating the code for the models based on the descriptions provided in their respective papers.

The evaluation process encompassed three scenarios: 1) early phases of malware propagation; 2) post-attack activities; and 3) a generalization test to mimic zero-day attacks. During the early detection stage, 1 million samples were employed for training and testing, encompassing 300 000 attack samples and 700 000 normal samples. The early stages of the malware life cycle were examined through an experiment conducted using the MedBIoT data set. The results, generated using SOTA ML [61], [62], DL [41], [45], [61], and DRL [70] models, indicate that these models exhibit good performance in detecting malware activities, with the exception of a few ML models. Related performance metrics are detailed in Table X and Fig. 6. The comparison revealed that the LSTM-CNN, Parallel Learning, and MalBoT-DRL models achieved the highest detection rates. Nevertheless, LSTM-CNN exhibited the highest precision, which can be attributed to the imbalanced data used for evaluation, predisposing the model to favor major classes. It is worth noting here that the performance of the LSTM-CNN model degraded during the late stage of the malware life cycle, as illustrated in Table XI.

In assessing the latter stage of the malware life cycle within IoT settings using the N-BaIoT data set, 855 932 samples were

TABLE X
EVALUATION OF THE PERFORMANCE OF OUR MODEL IN COMPARISON TO SOTA MODELS IN THE EARLY STAGE

Ref	Model	G	A	P	R	Bias	Gen
[41]	LSTM-CNN	N/A	**99.93**	**99.95**	**99.8**	✓	✗
[45]	Parallel	N/A	99.91	99.8	**99.8**	✓	✗
[47]	RNN	N/A	98.8	N/A	99	✓	✗
[61]	LR	N/A	82	88	71	✓	✗
[61]	LSTM-Chi20	N/A	99.72	99.36	99.53	✓	✗
[62]	G-PCA	N/A	83.27	76.75	63.41	✓	✗
[70]	DQL	N/A	96.99	97.02	96.99	✓	✓
Our	DRL	**99.75**	99.74	99.4	**99.8**	✗	✓

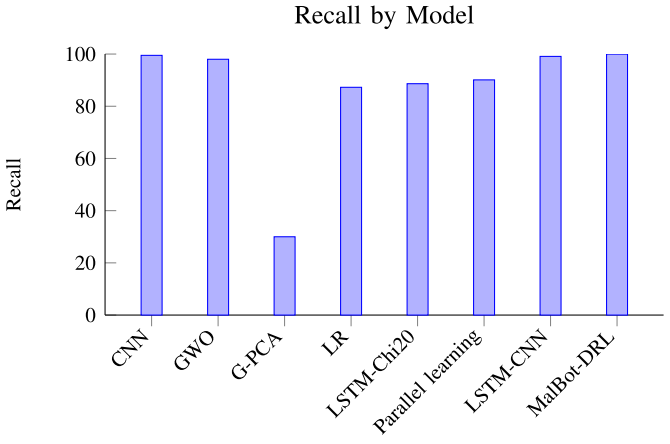

Fig. 6. Comparing the detection rate between our model and other SOTA models in the early stage.

TABLE XI
EVALUATION OF THE PERFORMANCE OF OUR MODEL IN COMPARISON TO SOTA MODELS IN THE LATE STAGE

Ref	Model	G	A	P	R	Basis	Gen
[39]	CNN	N/A	99.57	N/A	99.5	✓	✗
[41]	LSTM-CNN	N/A	98.35	96.3	99.1	✓	✗
[45]	Parallel	N/A	95.6	96.4	90.1	✓	✗
[59]	GWO	96.8	98	N/A	98	✓	✗
[61]	LR	N/A	94.8	94.2	87.26	✓	✗
[61]	LSTM-Chi20	N/A	95.77	99.19	88.64	✓	✗
[62]	G-PCA	N/A	83.27	76.75	63.41	✓	✗
Our	DRL	**99.97**	**99.97**	**99.93**	**99.98**	✗	✓

employed for training and testing purposes. This included 300 000 attack samples with the remaining constituting normal samples, where the proposed model achieved the highest detection rate with the lowest number of FP samples (64), as shown in Tables XI and IX, Figs. 6 and 7. While the performance of most models degraded during the malware life cycle, the proposed model exhibited superior generalizability. This can be ascribed to the learning process that relies on interaction with the environment rather than learning directly from labeled data.

In the second phase, the detection rates of the Parallel Learning approach, LSTM-CNN, and MalBot-DRL were 96.1%, 99.1%, and 99.8%, respectively. Although MalBot-DRL showed marginally better performance compared to

TABLE XII
EVALUATION OF THE GENERALIZABILITY OF OUR MODEL VERSUS SOTA MODELS WITH UNSEEN SAMPLES

Ref	Model	A	F1	P	R
[41]	CNN-LSTM	39.44	53.66	36.67	**99.99**
[45]	Parallel learning	38.05	53.08	36.13	99.98
Our	MalBot-DRL	**80.06**	**76.01**	**67.19**	88.05

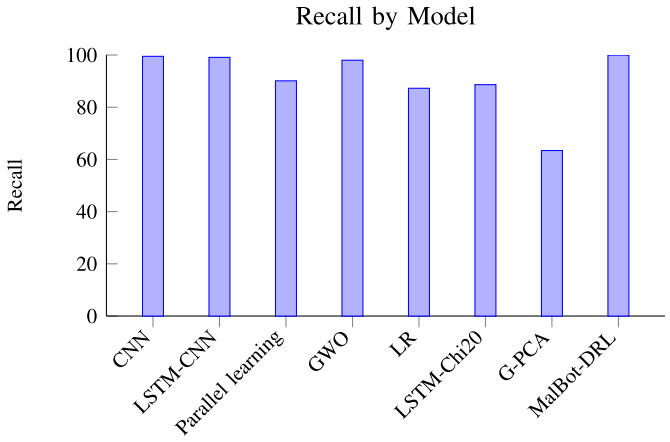

Fig. 7. Comparing the detection rate between our model and other studies based on the late stage.

LSTM-CNN, further evaluation was necessary to assess the model's generalization ability in the face of previously unseen threats, such as zero-day attacks. In the third scenario, the best models from the first two cases were trained using a combination of distinct attack and normal samples, totaling 764 137 samples [87]. These models were then tested on previously unseen normal and attack samples from the N-BaIoT data set. The results revealed a significant drop in the accuracy for all models, with MalBot-DRL falling from 99.97% to 80.06%, Parallel Learning to 36.16%, and LSTM-CNN to 36.67%, as shown in the generalization test results (Table XII). These findings emphasize the challenges associated with maintaining high detection rates in the face of previously unseen threats, such as zero-day attacks.

Our study offers valuable insights into the performance of different models during various stages of the malware botnet life cycle in IoT environments, as evident in Tables XII and XI. While the MalBot-DRL model exhibited strong performance in specific scenarios, detecting unseen attacks remained a significant challenge. The DRL approach, as employed by the MalBot-DRL model, possessed the potential to provide enhanced generalization capabilities due to its inherent adaptability and optimization of behavior through interaction with the environment. It is evident that further research and development are necessary to refine the DRL model's adaptability and effectiveness in detecting and mitigating emerging cybersecurity risks in IoT environments. By leveraging the power of DRL, more robust and adaptive cybersecurity solutions may be developed to better respond to the ever-evolving threat landscape in IoT ecosystems.

While the evaluation of MalBoT-DRL offers compelling results, it is crucial to recognize the inherent limitations and potential challenges of the proposed approach for a

comprehensive understanding of its capabilities. One limitation is demonstrated by the Bashlite malware, which employs stealth communication techniques with its (C&C) server. In this case, the proposed model has encountered challenges in accurate detection and response, indicating the need for further refinement to effectively handle sophisticated malware behavior, especially those utilizing covert communication channels. Moreover, the potential vulnerability of the system to adversarial attacks, as pointed out in the literature, is a significant concern. Given that MalBoT-DRL employs a NN as part of its DRL architecture, this susceptibility must be addressed to bolster the resilience of the proposed solution against evolving malware strategies.

In complex scenarios marked by rapid changes and voluminous data sets, our model may require a longer convergence period to reach an optimal policy. To counter this, we can enhance the model's exploration and exploitation policies and employ network distillation for maintaining or enhancing performance while reducing computational demands. This process starts by training a larger, more complex model, known as Q "teacher," in the cloud. After achieving satisfactory performance, we can then transfer its knowledge to a smaller "student" model. This student model, situated at the edge, is trained on both the raw data and the outputs from the teacher model. This setup allows the student model to emulate the teacher model's performance while demanding significantly less computational resources, making it an ideal solution for real-world IoT environments where resource constraints are prevalent.

Furthermore, improvements in performance generalization are often associated with an increase in computational complexity, leading to potential latency issues. As the model's capacity to generalize across a wider range of scenarios expands, the time taken to process each sample may increase. This effect was evidenced during our experiments when we expanded our training data set to include 855 932 flow samples and tested the model on a set of 256 781 samples. In the testing phase, a distinct trend was observed as the sample size increased. Prior to the expansion of the sample size, the model processed each sample in approximately 3.266e-05 s. After testing on the larger data set, however, the processing time per sample increased to roughly 4.725e-05 s. This growth, corresponding to an additional latency of around 14.59 microseconds per sample, implies a rise in latency with the growth of the testing sample size. Despite this increase, it is important to note here that this delay remains within acceptable boundaries. On the other hand, the training phase presented a different characteristic. Irrespective of the increase in the number of training samples, from 599 152 to 855 932, the model maintained a steady training time per sample, indicating strong consistency and efficiency.

VII. Conclusion

As the landscape of cyber attacks continues to evolve in response to the rapid growth of IoT devices and applications, it becomes increasingly crucial to efficiently detect suspicious behavior within IoT devices and handle a broad spectrum of threats. The primary aim of this article was to introduce the MalBoT-DRL system, a solution designed to detect botnets utilizing the damped incremental statistics method and DRL throughout various stages of the malware botnet life cycle. The proposed system effectively classifies botnet devices, thereby safeguarding IoT devices from potential threats. The key finding of this article is the ability of our model to accurately identify the propagation activities of attacks and detect post-activity when malware manages to evade initial detection. Compared to alternative ML approaches, the proposed solution demonstrates superior detection rates and adaptability across a diverse range of IoT environments. Future research should explore other RL methods and investigate the potential benefits of combining metaheuristics with RL, which could significantly reduce the learning time. Additionally, examining adversarial attacks, such as poisoning against the MalBoT-DRL system itself, is a vital future research direction. This would further enhance the resilience and adaptability of the model amidst the ever-changing threat landscape.

References

[1] S. Salini and B. P. U. Ivy, "Chapter 3—Digital twin and artificial intelligence in industries," in *Digital Twin for Smart Manufacturing*, R. K. Dhanaraj, A. K. Bashir, V. Rajasekar, B. Balusamy, and P. Malik, Eds. Amsterdam, The Netherlands: Academic, 2023, pp. 35–58. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323992053000146

[2] R. U. Rasool, H. F. Ahmad, W. Rafique, A. Qayyum, and J. Qadir, "Security and privacy of Internet of Medical Things: A contemporary review in the age of surveillance, botnets, and adversarial ML," *J. Netw. Comput. Appl.*, vol. 201, May 2022, Art. no. 103332. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804522000017

[3] H. Benyezza, M. Bouhedda, R. Kara, and S. Rebouh, "Smart platform based on IoT and WSN for monitoring and control of a greenhouse in the context of precision agriculture," *Internet Things*, vol. 23, Oct. 2023, Art. no. 100830. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660523001531

[4] B. Jovanovic. "Internet of Things statistics for 2023—Taking things apart." 2022. [Online]. Available: https://dataprot.net/statistics/iot-statistics/

[5] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of IoT: Applications, challenges, and opportunities with China perspective," *IEEE Internet Things J.*, vol. 1, no. 4, pp. 349–359, Aug. 2014.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[7] Y. Lu and L. D. Xu, "Internet of Things (IoT) cybersecurity research: A review of current research topics," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2103–2115, Apr. 2019.

[8] G. L. Nguyen, B. Dumba, Q.-D. Ngo, H.-V. Le, and T. N. Nguyen, "A collaborative approach to early detection of IoT Botnet," *Comput. Electr. Eng.*, vol. 97, Jan. 2022, Art. no. 107525. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790621004717

[9] U. Garg, S. Kumar, and M. Ghanshala, "Analysis and categorization of Emotet IoT botnet malware," in *Proc. Int. Conf. Artif. Intell. Smart Commun. (AISC)*, 2023, pp. 909–914.

[10] S. Dange and M. Chatterjee, "IoT Botnet: The largest threat to the IoT network," in *Advances in Intelligent Systems and Computing*. Singapore: Springer, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:208125836

[11] P. Kumari and A. K. Jain, "A comprehensive study of DDoS attacks over IoT network and their countermeasures," *Comput. Security*, vol. 127, Apr. 2023, Art. no. 103096. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404823000068

[12] X. Zhang, O. Upton, N. L. Beebe, and K.-K. R. Choo, "IoT Botnet forensics: A comprehensive digital forensic case study on Mirai botnet servers," *Forensic Sci. Int. Digit. Invest.*, vol. 32, Apr. 2020, Art. no. 300926. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281720300214

[13] Z. Ahmed, S. M. Danish, H. K. Qureshi, and M. Lestas, "Protecting IoTs from Mirai botnet attacks using blockchains," in *Proc. IEEE 24th Int. Workshop Comput. Aided Model. Design Commun. Links Netw. (CAMAD)*, 2019, pp. 1–6.

[14] P. Victor, A. H. Lashkari, R. Lu, T. Sasi, P. Xiong, and S. Iqbal, "IoT malware: An attribute-based taxonomy, detection mechanisms and challenges," *Peer-to-Peer Netw. Appl.*, vol. 16, no. 3, pp. 1380–1431, 2023. [Online]. Available: https://doi.org/10.1007/s12083-023-01478-w

[15] O. Yousuf and R. N. Mir, "DDoS attack detection in Internet of Things using recurrent neural network," *Comput. Elect. Eng.*, vol. 101, Jul. 2022, Art. no. 108034. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S004579062200297X

[16] A. Alatram, L. F. Sikos, M. Johnstone, P. Szewczyk, and J. J. Kang, "DoS/DDoS-MQTT-IoT: A dataset for evaluating intrusions in IoT networks using the MQTT protocol," *Comput. Netw.*, vol. 231, Jul. 2023, Art. no. 109809. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128623002542

[17] "Nokia threat intelligence report finds malicious IoT botnet activity has sharply increased." Nokia. Jun./Jul. 2023. [Online]. Available: https://www.nokia.com/about-us/news/releases/2023/06/07/nokia-threat-intelligence-report-finds-malicious-iot-botnet-activity-has-sharply-increased/

[18] R. Faek, M. Al-Fawa'reh, and M. Al-Fayoumi, "Exposing bot attacks using machine learning and flow level analysis," in *Proc. Int. Conf. Data Sci. E-Learn. Inf. Syst.*, 2021, pp. 99–106. [Online]. Available: https://doi.org/10.1145/3460620.3460739

[19] M. Al-Fawa'reh, M. Al-Fayoumi, S. Nashwan, and S. Fraihat, "Cyber threat intelligence using PCA-DNN model to detect abnormal network behavior," *Egypt. Inform. J.*, vol. 23, no. 2, pp. 173–185, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110866521000785

[20] R. Z. Mohd, M. F. Zuhairi, A. Z. Shadil, and H. Dao, "Anomaly-based NIDS: A review of machine learning methods on malware detection," in *Proc. Int. Conf. Inf. Commun. Technol. (ICICTM)*, 2016, pp. 266–270.

[21] Z. Yang et al., "A systematic literature review of methods and datasets for anomaly-based network intrusion detection," *Comput. Security*, vol. 116, May 2022, Art. no. 102675. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404822000736

[22] P. R. Kannari, N. S. Chowdary, and R. L. Biradar, "An anomaly-based intrusion detection system using recursive feature elimination technique for improved attack detection," *Theor. Comput. Sci.*, vol. 931, pp. 56–64, Sep. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397522004479

[23] A. Meena, D. Nigam, D. Sharma, and A. Chauhan, "Anomaly based intrusion detection for IoT: (A deep learning approach)," in *Proc. 3rd Int. Conf. Adv. Comput., Commun. Control Netw. (ICAC3N)*, 2021, pp. 1349–1356.

[24] M. Bhavsar, K. Roy, J. Kelly, and O. Olusola, "Anomaly-based intrusion detection system for IoT application," *Discover Internet Things*, vol. 3, no. 1, p. 5, 2023. [Online]. Available: https://doi.org/10.1007/s43926-023-00034-5

[25] Y. Al-Hadhrami and F. K. Hussain, "DDoS attacks in IoT networks: A comprehensive systematic literature review," *World Wide Web*, vol. 24, no. 3, pp. 971–1001, May 2021. [Online]. Available: https://doi.org/10.1007/s11280-020-00855-2

[26] K. Sethi, E. Sai Rupesh, R. Kumar, P. Bera, and Y. V. Madhav, "A context-aware robust intrusion detection system: A reinforcement learning-based approach," *Int. J. Inf. Security*, vol. 19, no. 6, pp. 657–678, 2020. [Online]. Available: https://doi.org/10.1007/s10207-019-00482-7

[27] Y. Badr, "Enabling intrusion detection systems with dueling double deep-learning," *Digit. Transf. Soc.*, vol. 1, no. 1, pp. 115–141, 2022. [Online]. Available: https://doi.org/10.1108/DTS-05-2022-0016

[28] J. Yang, A. A. S. Soltan, and D. A. Clifton, "Machine learning generalizability across healthcare settings: Insights from multi-site COVID-19 screening," *NPJ Digit. Med.*, vol. 5, p. 69, Jun./Jul. 2022.

[29] R. E. Carter, V. Anand, D. M. Harmon, and P. A. Pellikka, "Model drift: When it can be a sign of success and when it can be an occult problem," *Intell.-Based Med.*, vol. 6, Apr. 2022, Art. no. 100058. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666521222000114

[30] J.-S. Lee, Y.-C. Chen, C.-J. Chew, C.-L. Chen, T.-N. Huynh, and C.-W. Kuo, "CoNN-IDS: Intrusion detection system based on collaborative neural networks and agile training," *Comput. Security*, vol. 122, Nov. 2022, Art. no. 102908. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404822003017

[31] E. Anthi, L. Williams, M. Rhode, P. Burnap, and A. Wedgbury, "Adversarial attacks on machine learning cybersecurity defences in industrial control systems," *J. Inf. Security Appl.*, vol. 58, May 2021, Art. no. 102717. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212620308607

[32] A. Alotaibi and M. A. Rassam, "Adversarial machine learning attacks against intrusion detection systems: A survey on strategies and defense," *Future Internet*, vol. 15, no. 2, p. 62, 2023. [Online]. Available: https://www.mdpi.com/1999-5903/15/2/62

[33] L. Sun, M. Tan, and Z. Zhou, "A survey of practical adversarial example attacks," *Cybersecurity*, vol. 1, no. 1, p. 9, 2018. [Online]. Available: https://doi.org/10.1186/s42400-018-0012-9

[34] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[35] A. Uprety and D. B. Rawat, "Reinforcement learning for IoT security: A comprehensive survey," *IEEE Internet Things J.*, vol. 8, no. 11, pp. 8693–8706, Jun. 2021.

[36] V. Mnih et al., "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[37] M. Al-Fawa'reh, Z. Ashi, and M. T. Jafar, "Detecting malicious DNS queries over encrypted tunnels using statistical analysis and bi-directional recurrent neural networks," *Karbala Int. J. Modern Sci.*, vol. 7, no. 4, p. 4, 2021. [Online]. Available: https://doi.org/10.33640/2405-609X.3155

[38] G. Kou, G.-M. Tang, S. Wang, H.-T. Song, and Y. Bian, "Using deep learning for detecting BotCloud," *J. Commun*, vol. 37, no. 11, pp. 114–128, 2016.

[39] J. Liu, S. Liu, and S. Zhang, "Detection of IoT botnet based on deep learning," in *Proc. Chin. Control Conf. (CCC)*, 2019, pp. 8381–8385.

[40] D. Alghazzawi, O. Bamasag, H. Ullah, and M. Z. Asghar, "Efficient detection of DDoS attacks using a hybrid deep learning model with improved feature selection," *Appl. Sci.*, vol. 11, no. 24, 2021, Art. no. 11634.

[41] M. Wazzan, D. Algazzawi, A. Albeshri, S. Hasan, O. Rabie, and M. Z. Asghar, "Cross deep learning method for effectively detecting the propagation of IoT botnet," *Sensors*, vol. 22, no. 10, p. 3895, 2022.

[42] W. Jung, H. Zhao, M. Sun, and G. Zhou, "IoT botnet detection via power consumption modeling," *Smart Health*, vol. 15, Mar. 2020, Art. no. 100103.

[43] F. Hussain, "A two-fold machine learning approach to prevent and detect IoT botnet attacks," *IEEE Access*, vol. 9, pp. 163412–163430, 2021.

[44] Z. H. C. Zhou, "Deep learning detection based on traffic characteristics of botnet," *Inf. Technol.*, vol. 4, pp. 1–9, 2018.

[45] F. Sattari, A. H. Farooqi, Z. Qadir, B. Raza, H. Nazari, and M. Almutiry, "A hybrid deep learning approach for bottleneck detection in IoT," *IEEE Access*, vol. 10, pp. 77039–77053, 2022.

[46] C. D. Mcdermott, F. Majdani, and A. Petrovski, "Botnet detection in the Internet of Things using deep learning approaches," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2018, pp. 1–8.

[47] L. Giaretta, A. Lekssays, B. Carminati, E. Ferrari, and Š. Girdzijauskas, "LiMNet: Early-stage detection of IoT botnets with lightweight memory networks," in *Computer Security* (Lecture Notes in Computer Science 12972), E. Bertino, H. Shulman, and M. Waidner, Eds. Cham, Switzerland: Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-88418-5_29

[48] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: A survey," *Data Min. Knowl. Disc.*, vol. 29, pp. 626–688, Jul. 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:5865347

[49] J. Wang, Z. Li, M. Sun, B. Yuan, and J. C. Lui, "IoT anomaly detection via device interaction graph," in *Proc. 53rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, 2023, pp. 494–507.

[50] M. Gao, L. Wu, Q. Li, and W. Chen, "Anomaly traffic detection in IoT security using graph neural networks," *J. Inf. Security Appl.*, vol. 76, Aug. 2023, Art. no. 103532. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212623001163

[51] T. Pourhabibi, K.-L. Ong, B. H. Kam, and Y. L. Boo, "Fraud detection: A systematic literature review of graph-based anomaly detection approaches," *Decis. Support Syst.*, vol. 133, Jun. 2020, Art. no. 113303. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167923620300580

[52] R. Kaur and S. Singh, "A survey of data mining and social network analysis based anomaly detection techniques," *Egypt. Inform. J.*, vol. 17, no. 2, pp. 199–216, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110866515000651

[53] W. Gao, H. Wu, M. K. Siddiqui, and A. Q. Baig, "Study of biological networks using graph theory," *Saudi J. Biol. Sci.*, vol. 25, no. 6, pp. 1212–1219, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319562X17302966

[54] W. Wang, Y. Shang, Y. He, Y. Li, and J. Liu, "BotMark: Automated botnet detection with hybrid analysis of flow-based and graph-based traffic behaviors," *Inf. Sci*, vol. 511, pp. 284–296, Feb. 2020.

[55] D. P. Hostiadi and T. Ahmad, "Hybrid model for bot group activity detection using similarity and correlation approaches based on network traffic flows analysis," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 7, pp. 4219–4232, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157822001537

[56] W. Yassin, R. Abdullah, M. F. Abdollah, Z. Mas'ud, and F. A. Bakhari, "An IoT botnet prediction model using frequency based dependency graph: Proof-of-concept," in *Proc. 7th Int. Conf. Inf. Technol. IoT Smart City*, pp. 344–352, 2020.

[57] H.-T. Nguyen, Q.-D. Ngo, and V.-H. Le, "A novel graph-based approach for IoT botnet detection," *Int. J. Inf. Security*, vol. 19, pp. 567–577, Oct. 2019.

[58] S. Nomm and H. Bahsi, "Unsupervised anomaly based botnet detection in IoT networks," in *Proc. 17th IEEE Int. Conf. Mach. Learn. Appl.*, 2019, pp. 1048–1053.

[59] A. Shorman, H. Faris, and I. Aljarah, "Unsupervised intelligent system based on one class support vector machine and grey wolf optimization for IoT botnet detection," *J. Ambient Intell. Humaniz. Comput*, vol. 11, no. 7, pp. 2809–2825, 2020.

[60] M. Injadat, A. Moubayed, and A. Shami, "Detecting botnet attacks in IoT environments: An optimized machine learning approach," in *Proc. Int. Conf. Microelectron.*, 2020, pp. 1–4.

[61] R. Gandhi and Y. Li, "Comparing machine learning and deep learning for IoT botnet detection," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, pp. 234–239, 2021.

[62] W. Aprianti, "Implementasi principal component analysis (PCA) dan algoritma Naíve Bayes classifier Pada Klasifikasi botnet Di Jaringan Internet of Things," Ph.D. dissertation, Fakultas ilmu komputer, Universitas Sriwijaya, Palembang, Indonesia, 2021.

[63] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020.

[64] G. Gomes, C. A. Vidal, J. B. Cavalcante-Neto, and Y. L. B. Nogueira, "A modeling environment for reinforcement learning in games," *Entertain. Comput.*, vol. 43, Aug. 2022, Art. no. 100516. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1875952122000404

[65] M. Dalal, D. Pathak, and R. Salakhutdinov, "Accelerating robotic reinforcement learning via parameterized action primitives," 2021, *arXiv:2110.15360*.

[66] H. Benaddi, K. Ibrahimi, A. Benslimane, and J. Qadir, "A deep reinforcement learning based intrusion detection system (DRL-IDS) for securing wireless sensor networks and Internet of Things," in *Wireless Internet*. Cham, Switzerland: Springer, 2020, pp. 73–87.

[67] X. Ma and W. Shi, "AESMOTE: Adversarial reinforcement learning with SMOTE for anomaly detection," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 943–956, Apr.–Jun. 2021.

[68] H. Benaddi, M. Jouhari, K. Ibrahimi, J. B. Othman, and E. M. Amhoud, "Anomaly detection in industrial IoT using distributional reinforcement learning and generative adversarial networks," *Sensors*, vol. 22, no. 21, p. 8085, 2022.

[69] G. Apruzzese, M. Andreolini, M. Marchetti, A. Venturi, and M. Colajanni, "Deep reinforcement adversarial learning against botnet evasion attacks," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 1975–1987, Dec. 2020.

[70] P. May Raju and G. P. Gupta, "Intrusion detection framework using an improved deep reinforcement learning technique for IoT network," in *Soft Computing for Security Applications*. Singapore: Springer, 2022, pp. 765–779.

[71] "Bootcamp summer 2020 week 4: On-policy vs off-policy reinforcement learning." 2022. Accessed: Jul. 20, 2023. [Online]. Available: https://core-robotics.gatech.edu/2022/02/28/bootcamp-summer-2020-week-4-on-policy-vs-off-policy-reinforcement-learning/

[72] G. Caminero, M. Lopez-Martin, and B. Carro, "Adversarial environment reinforcement learning algorithm for intrusion detection," *Comput. Netw.*, vol. 159, pp. 96–109, Aug. 2019.

[73] CIDRE Research Team, *Intrusion Detection in Information Systems Using Reinforcement Learning Techniques*. (Jan. 2022). [Online Video]. Available: https://www.youtube.com/watch?v=UzZ1urlJTc8

[74] Y.-F. Hsu and M. Matsuoka, "A deep reinforcement learning approach for anomaly network intrusion detection system," in *Proc. IEEE 9th Int. Conf. Cloud Netw.*, 2020, pp. 1–6.

[75] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[76] A. Servin, *Towards Traffic Anomaly Detection via Reinforcement Learning and Data Flow*, Univ. York, York, U.K., 2007.

[77] C. Huang, Y. Wu, Y. Zuo, K. Pei, and G. Min, "Towards experienced anomaly detector through reinforcement learning," in *Proc. AAAI/IAAI/EAAI*, 2018, pp. 8087–8088.

[78] N. Sengupta, J. Sen, J. Sil, and M. Saha, "Designing of on line intrusion detection system using rough set theory and Q-learning algorithm," *Neurocomputing*, vol. 111, pp. 161–168, Jul. 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523121300060X

[79] R. V. Hoa, T. D. Chuyen, N. T. Lam, T. N. Son, N. D. Dien, and V. T. T. Linh, "Reinforcement learning based method for autonomous navigation of mobile robots in unknown environments," in *Proc. Int. Conf. Adv. Mechatronic Syst. (ICAMechS)*, 2020, pp. 266–269.

[80] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: Bradford Book, 2018.

[81] V. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[82] S. N. Vu, M. Stege, P. I. El-Habr, J. Bang, and N. Dragoni, "A survey on botnets: Incentives, evolution, detection and current trends," *Futur. Internet*, vol. 13, no. 8, p. 198, 2021.

[83] S. Bakhshad, V. Ponnusamy, R. Annur, M. Waqasyz, H. Alasmary, and S. Tux, "Deep reinforcement learning based intrusion detection system with feature selections method and optimal hyper-parameter in IoT environment," in *Proc. Int. Conf. Comput., Inf. Telecommun. Syst. (CITS)*, Piraeus, Greece, 2022, pp. 1–7, doi: 10.1109/CITS55221.2022.9832976.

[84] Y. Shen and D. L. Lee, "An MDP-based peer-to-peer search server network," in *Proc. 3rd Int. Conf. Web Inf. Syst. Eng.*, 2002, pp. 269–278.

[85] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," 2015, *arXiv:1509.06461*.

[86] H. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23. Red Hook, NY, USA: Curran Assoc., Inc., 2010. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf

[87] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. NDSS*, Jan. 2018, pp. 18–21. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_06B-3_Wang_paper.pdf

[88] Y. Tian-Yi, L. Shi-Yue, and L. Jun-Yi, "Network traffic anomaly detection based on incremental possibilistic clustering algorithm," in *Proc. J. Phys. Conf. Ser.*, 2019, Art. no. 12067.

[89] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[90] W. Jia, J. Li, and Y. Zhao, "DQN algorithm based on target value network parameter dynamic update," in *Proc. IEEE 4th Int. Conf. Comput. Commun. Eng. Technol. (CCET)*, 2021, pp. 285–289.

[91] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," 2006, *arXiv:2006.04779*.

[92] M. Lopez-Martin, B. Carro, and A. Sanchez-Esguevillas, "Application of deep reinforcement learning to intrusion detection for supervised problems," *Expert Syst. Appl*, vol. 141, Mar. 2020, Art. no. 112963.

[93] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," 2015, *arXiv:1507.00814*.

[94] E. Lin, Q. Chen, and X. Qi, "Deep reinforcement learning for imbalanced classification," *Appl. Intell.*, vol. 50, pp. 2488–2502, Mar. 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:57573816

[95] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, and S. Nõmm, "MedBIoT: Generation of an IoT botnet dataset in a medium-sized IoT network," in *Proc. 6th ICISSP* , 2020, pp. 207–218.

[96] Y. Meidan, "N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Comput*, vol. 17, no. 3, pp. 12–22, Jul.–Sep. 2018.

[97] "psutil 5.9.6." Accessed: Aug. 19, 2023. [Online]. Available: https://pypi.org/project/psutil/

Mohammad Al-Fawa'reh (Member, IEEE) received the M.S. degree in information systems security and digital criminology from the Princess Sumaya University for Technology, Amman, Jordan, in 2020. He is currently pursuing the Ph.D. degree in cybersecurity with Edith Cowan University, Joondalup, WA, Australia.

His research interests include reinforcement learning, adversarial attacks, cyber threat hunting, and Malware analysis.

Jumana Abu-Khalaf (Senior Member, IEEE) received the Ph.D. degree in mechanical engineering from The University of Utah, Salt Lake City, UT, USA, in 2012.

She is a Lecturer with Edith Cowan University (ECU), Joondalup, WA, Australia. Prior to joining ECU, she served as an Associate Professor of Mechatronics Engineering with the German Jordanian University, Amman, Jordan. Her research focuses on biorobotics, artificial intelligence, and machine learning techniques for healthcare applications.

Patryk Szewczyk received the Ph.D. degree in cyber security from Edith Cowan University, Joondalup, WA, Australia, in 2015.

He is a Senior Cyber Security and Digital Forensics Lecturer with Edith Cowan University. He has supervised numerous master's and Ph.D. students.

Dr. Szewczyk has been an international reviewer for numerous conferences and journals.

James Jin Kang received the master's degree in engineering management from the University of Technology Sydney, Ultimo, NSW, Australia, in 2009, and the Doctor of Philosophy (Ph.D.) degree in health-based technology (eHealth, mHealth) from Deakin University, Geelong, VIC, Australia, in 2017.

He is an Associate Professor of Smart Medical Health Informatics with the National Taiwan University, Taipei, Taiwan, and an Adjunct Lecturer of Computing and Security with Edith Cowan University, Joondalup, WA, Australia. With over 20 years of experience in the telecommunications industry, his research interests include cybersecurity, informatics and computing in the healthcare sector, and wireless sensor networks.