IEEE ComSoc IEEE Communications Society | IEEE COMPUTER SOCIETY | IEEE Signal Processing Society | VTS Connecting the Mobile World | IEEE Transactions on Machine Learning in Communications and Networking

Received 15 November 2023; revised 5 March 2024; accepted 23 April 2024.
Date of publication 30 April 2024; date of current version 6 May 2024.

The associate editor coordinating the review of this article and approving it for publication was E. Bou-Harb.

Digital Object Identifier 10.1109/TMLCN.2024.3395419

Deep Ensemble Learning With Pruning for DDoS Attack Detection in IoT Networks

MAKHDUMA F. SAIYED[1] (Student Member, IEEE)
AND IRFAN AL-ANBAGI[2] (Senior Member, IEEE)

Faculty of Engineering and Applied Science, University of Regina, Regina, SK S4S 0A2, Canada
CORRESPONDING AUTHOR: I. AL-ANBAGI (Irfan.Al-Anbagi@uregina.ca)

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant RGPIN-2019-06060.

ABSTRACT The upsurge of Internet of Things (IoT) devices has increased their vulnerability to Distributed Denial of Service (DDoS) attacks. DDoS attacks have evolved into complex multi-vector threats that high-volume and low-volume attack strategies, posing challenges for detection using traditional methods. These challenges highlight the importance of reliable detection and prevention measures. This paper introduces a novel Deep Ensemble learning with Pruning (DEEPShield) system, to efficiently detect both high- and low-volume DDoS attacks in resource-constrained environments. The DEEPShield system uses ensemble learning by integrating a Convolutional Neural Network (CNN) and a Long Short-Term Memory (LSTM) network with a network traffic analysis system. This system analyzes and preprocesses network traffic while being data-agnostic, resulting in high detection accuracy. In addition, the DEEPShield system applies unit pruning to refine ensemble models, optimizing them for deployment on edge devices while maintaining a balance between accuracy and computational efficiency. To address the lack of a detailed dataset for high- and low-volume DDoS attacks, this paper also introduces a dataset named HL-IoT, which includes both attack types. Furthermore, the testbed evaluation of the DEEPShield system under various load scenarios and network traffic loads showcases its effectiveness and robustness. Compared to the state-of-the-art deep ensembles and deep learning methods across various datasets, including HL-IoT, ToN-IoT, CICIDS-17, and ISCX-12, the DEEPShield system consistently achieves an accuracy over 90% for both DDoS attack types. Furthermore, the DEEPShield system achieves this performance with reduced memory and processing requirements, underscoring its adaptability for edge computing scenarios.

INDEX TERMS CNN, deep learning, DDoS attacks, ensemble learning, IoT security, LSTM, pruning.

I. INTRODUCTION

INTEGRATING Internet of Things (IoT) devices across various sectors has transformed industries by improving efficiency and connectivity. However, this expansion has also increased the risk of Distributed Denial of Service (DDoS) attacks [1], [2]. The evolution of IoT and network systems with advanced technologies such as cloud computing, 5G and 6G networks, and intelligent management solutions has significantly increased the complexity and potential impact of DDoS attacks. These technologies enhance network performance and capabilities and introduce new vulnerabilities that attackers exploit [3]. In recent years, botnets such as Mirai, Gafgyt, and Reaper have exploited compromised IoT devices to launch massive DDoS attacks, targeting entities like Domain Name System (DNS) provider Dyn, French web hosts and gaming websites [4], [5], [6]. Besides botnets, ransomware attacks like BrickerBot in 2017 have specifically targeted IoT devices globally [7].

DDoS attacks have evolved with various evasion techniques, including multi-vector attacks that combine volumetric, protocol, and application layer methods. In IoT networks, two common attack types are exploitation-based and reflection-based. Exploitation-based DDoS attacks use User Datagram Protocol (UDP), Transmission Control Protocol Synchronization (TCP SYN), Transport Layer Security (TLS), and Hypertext Transfer Protocol (HTTP) overwhelm to target vulnerabilities in IoT devices. Reflection-based DDoS attacks, on the other hand, utilize third-party systems

to generate and amplify attack traffic through techniques such as DNS amplification, Network Time Protocol (NTP) amplification, and Simple Network Management Protocol (SNMP) reflection [8], [9]. Based on the traffic flow volume, DDoS attacks are further classified into high-volume and low-volume attacks. Low-volume attacks, such as slow-rate, one-shot, and low-rate attacks, exploit vulnerabilities and bypass traditional detection mechanisms [10], [11], [12]. The complex attack approaches lead to severe consequences such as service disruptions, data theft, and financial loss.

The continuous growth of network technologies, along with the increasing complexity of DDoS techniques, highlights the need for advanced and dynamic security measures. Traditional intrusion detection methods are vulnerable to evasion by attackers because they rely on predefined attack patterns. This challenge becomes even more significant when trying to detect novel and complex attacks. It highlights the need for more adaptive and flexible approaches, such as the behavior-based approach that includes Machine Learning (ML) and Deep Learning (DL) [13], the flow-based approach, and the ensemble approach. Recently, the implementation of ML has gained significant attention for DDoS attack detection in IoT networks [14]. ML methods rely on feature engineering, which uses manually crafted features extracted from raw data. However, the manually crafted features may not always effectively capture the complex and evolving nature of DDoS attacks, thus limiting detection accuracy. DL methods show an advantage over traditional feature engineering as it can learn important features from raw data, avoiding the need for manual feature selection. The DL models, including Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), have the ability to automatically learn hierarchical representations of data from raw input. This helps DL models to adapt better to complex and changing attack patterns and provides improved DDoS attack detection accuracy [15].

The ensemble approach offers a more robust and accurate solution to detect DDoS attacks by combining the capabilities of DL and traditional ML models [16]. Moreover, it can also handle imbalanced data, adapt to changing attack patterns, and provide more reliable and concise results, making it preferred over models that are solely based on ML or DL methods. DL methods, based on CNN and Long Short-Term Memory (LSTM) networks, have important contributions in the field of network security. While CNN has improved the network traffic analysis and intrusion detection in IoT networks [17], [18], [19], the LSTM networks have shown their utility in dealing with time-series data patterns [20], [21], [22]. CNN excels at obtaining spatial features from data, whereas LSTMs excel at understanding temporal sequences. Combining the strengths of CNN and LSTM in an ensemble approach is a reasonable choice to enhance the capabilities of DDoS detection systems.

Ensemble approaches for high- and low-volume DDoS attack detection have been under-explored by researchers [23], [24]. These approaches outperform individual models in detection accuracy but demand more resources and computing power [25], [26]. While ensemble approaches have been investigated in the current literature, the challenge within the context of edge computing has received limited attention. Addressing ensemble approaches in the context of edge computing is essential to optimizing detection accuracy while working within the resource limitations of edge devices. This paper bridges this gap by introducing DEEPShield, a novel system that combines deep ensemble learning with pruning. The DEEPShield system is designed to address the challenge of accurate detection of both high-volume and low-volume DDoS attacks on resource-constrained edge devices. Unlike prior studies that focused on conventional software- and CPU-based systems, this approach integrates post-training optimization techniques tailored for edge deployment. The objective of the DEEPShield system is to achieve a balance between size and performance, making the proposed ensembles efficient and usable for edge deployment. The DEEPShield system achieves high-performance DDoS detection in IoT environments with limited computational resources by integrating CNN and LSTM architectures within a fine-tuned ensemble framework and employing a novel dataset (HL-IoT) for validation. The primary contributions of this paper are summarized as follows:

- A network traffic analysis system is developed to analyze and preprocess network traffic, remaining data-agnostic to ensure consistency across various datasets. This system inspects network packets continuously and generates a feature vector for each flow.
- In DEEPShield system, a novel DL-based ensemble model is employed to detect both high-volume and low-volume DDoS attacks in IoT networks. The ensemble model, combining CNN and LSTM architectures, is optimized through hyperparameter tuning, ensuring high accuracy and a compact design. This makes it suitable for resource-constrained IoT environments.
- The HL-IoT dataset,[1] comprising both high- and low-volume DDoS attacks in IoT networks, is utilized. The features of both normal and attack network traffic are validated using the Mann-Whitney U test [27].
- The DEEPShield system is optimized using a post-training unit pruning technique to reduce the model size and prediction time. This optimization improves efficiency by accelerating inference while maintaining high detection accuracy.
- The DEEPShield system is evaluated across different load scenarios to showcase its effectiveness and robustness in real-world environments.
- The performance evaluation of the DEEPShield system is benchmarked against state-of-the-art deep ensembles and DL models for DDoS attack detection using HL-IoT, ToN-IoT, CICIDS-17, and ISCX-12 datasets.

[1] https://www.kaggle.com/datasets/makhdumasaiyed/hl-iot-dataset

The rest of the paper is structured as follows: Section II provides a review of related work. Section III introduces the DEEPShield system and discusses its various modules including network traffic analysis and preprocessing, learning, and model optimization. Section IV provides details on the testbed implementation and performance evaluation. Finally, Section V concludes the paper.

II. RELATED WORK

This section categorizes DDoS attack detection research based on three approaches: ML-based, DL-based, and ensemble-based.

A. ATTACK DETECTION USING ML APPROACHES

Shafiq et al. [28] developed a method using traditional ML to detect Bot-IoT traffic. Their approach incorporated feature selection techniques along with Decision Trees (DTs) and Random Forest (RF) classifiers. However, further testing of their model in various IoT scenarios is required, addressing potential biases inherent in ML. Moreover, their method is not suitable for detecting slow-rate attacks. Hafeez et al. [29] introduced a system to detect malicious traffic within IoT networks using fuzzy C-means and fuzzy interpolation. The system combined lightweight traffic classification techniques and anomaly detection schemes, collecting data from both IoT devices and user devices connected to access points. Performance evaluations were conducted in both closed-world and open-world scenarios, with the system using adhoc overlay networks to automate security policy management. Despite the merits, the limitations of the method included the necessity for realistic open-world settings and potential performance variability across different hardware and software stacks. Additionally, the system's effectiveness could be affected by factors such as the quality and quantity of unlabeled data used for training, as well as the choice of feature analysis techniques used to extract relevant features for classification. Saiyed and Al-Anbagi [2] proposed a GADAD system to detect high- and low-volume DDoS attacks using a combination of genetic algorithms, t-test-based feature selection, and tree-based machine learning. This system utilized the t-test method to compute the fitness score of the genetic algorithm, resulting in reduced computational resource usage and improved accuracy in detecting both types DDoS attacks. Despite the use of advanced feature selection methods and tree-based classifiers, the system struggles to accurately detect variations in low-volume DDoS attacks. Additionally, the need for feature engineering limits the scalability of the system, particularly in high-throughput network environments.

Aboelwafa et al. [30] applied ML to detect false data injection attacks in industrial IoT systems. They used SVM for classification and Autoencoders (AE) and Denoising Autoencoders (DAE) to identify corrupted data, achieving better detection performance compared to SVM-based methods. However, scalability issue resolution and compatibility tests on different datasets are needed to enhance its usefulness in real-world industrial IoT systems. Bhayo et al. [31] proposed a method to detect DDoS attacks within IoT networks using ML and SDN. The framework combined algorithms like Naive Bayes, Decision Tree, and Support Vector Machine (SVM) with an SDN-WISE controller for packet classification and attack detection. The evaluation was conducted in a simulated environment, which showed an improved accuracy in DDoS attack detection, using dynamic ML techniques for threat detection. However, the method did not consider low-volume attacks, which are subtle and difficult to distinguish from normal traffic. Ravi and Shalinie [32] proposed a semisupervised-learning-based security solution, that involved data collection and attack detection using K-means clustering. The system was deployed on the Cisco Nexus switch 5672UP fog node for low-latency IoT services. However, a relatively long attack detection time indicates the need for further research into alternative ML techniques. In the method proposed by Koay et al. [33], the algorithm performance depends on the selected features and datasets. The models discussed by Gyamfi and Jurcut [34] need more consideration for high- and low-volume DDoS attacks.

B. ATTACK DETECTION USING DL APPROACHES

Nie et al. [35] proposed a Generative Adversarial Network (GAN)-based method to identify diverse attacks in edge network devices. Using CICFlowMeter, the authors optimized feature selection with backpropagation and Adaptive Moment (ADAM) estimation. GAN achieved high accuracy and low false alarms using a game theory-based approach and deep feedforward network. However, this method achieved lower detection accuracy for low-flow attacks. This limitation suggests a potential improvement in detecting a wide range of attacks by combining GAN with a robust DL classifier like CNN or Artificial Neural Network (ANN). Zeeshan et al. [20] introduced a protocol-based deep intrusion detection system that uses LSTM to detect DoS and DDoS attacks, using the UNSW-NB15 and Bot-IoT datasets. However, the absence of feature selection and a slower detection process in this system compromises its performance. The use of imbalanced training datasets can further impact the efficiency of their classifiers. Jia et al. [36] proposed a flow-based analysis for IoT network attack detection at the edge using the CICDDoS2019 dataset. The results suggested that LSTM and CNN-based models outperform RF, naive bayes, and Logistic Regression (LR) in accuracy. However, the high computational demand of their model highlights the necessity of lightweight methods to support practical implementation.

Huong et al. [37] developed a low-complexity cyber-attack detection system for IoT edge networks, focused on detecting DoS, DDoS, and scanning attacks. This method used Principle Component Analysis (PCA) for feature reduction and Neural Networks (NN) for attack detection. The model was trained in a cloud-based environment and implemented on edge devices, using the Bot-IoT dataset. However, the paper does not explicitly outline significant trade-offs between

TABLE 1. Comparative analysis: Existing ML- and DL-Based system versus proposed system.

References	Method	Datasets	High- and Low-volume Attacks	Ensemble Learning	Resource Constraint Edge IoT Network	Pruning	Memory Utilization	Processing Time	Processing Rate	Real Testbed Deployment
[18]	DL-CNN	CICIDS-17, CICIDS-18, ISCX	x	x	✓	x	✓	x	✓	✓
[20]	DL-LSTM	UNSW-NB 15, Bot-IoT	x	✓	x	x	x	x	x	x
[24]	Ensemble-CNN, LSTM	CICIDS-17	x	✓	x	x	✓	✓	x	x
[28]	ML-DT, RF	Bot-IoT	x	x	x	x	x	x	x	x
[29]	Fuzzy C-means clustering and fuzzy interpolation	IoT Dataset	x	x	x	x	✓	✓	x	✓
[30]	ML-SVM	Simulated	x	x	x	x	x	x	x	x
[31]	ML-SDN-NB, SVM, DT	Simulated	x	x	✓	x	✓	✓	x	x
[32]	ML-K means clustering	Real testbed	x	x	✓	x	✓	x	✓	✓
[35]	DL-GAN	CICDDoS-2019	✓	✓	x	x	x	x	x	x
[36]	DL-CNN, LSTM	CICDDoS-2019	✓	x	x	x	✓	✓	x	✓
[37]	DL-PCA, NN	Bot-IoT	x	✓	x	x	✓	✓	x	✓
[38]	DL-CNN	UNSW-NB 15	x	x	x	x	x	x	x	x
[39]	DL-DNN	CSIC 2010, FWAF, and HttpParams	x	x	x	x	x	x	x	x
[40]	Ensemble-RF, DT, KNN, DNN	UNSW-NB 15, NIMS	x	✓	x	x	✓	x	✓	x
[41]	DL-LSTM	Edge-IIoTset	x	x	✓	x	x	x	x	x
DEEPShield System	Ensemble-CNN, LSTM	HL-IoT, CICIDS-17, ISCX, ToN-IoT	✓	✓	✓	✓	✓	✓	✓	✓

detection accuracy and computational complexity. Tian et al. [38] introduced a web attack detection system using distributed DL deployed on edge devices. The system utilized CNNs and natural language processing models to analyze URLs for web attack detection. By converting URLs into vectors using word2vec, it processed them through multiple concurrent DL models for improved stability and easy updating. The system utilized datasets like CSIC 2010, FWAF, and HttpParams to evaluate detection accuracy. However, it has challenges in handling unknown words within URLs and needs further optimization of comprehensive decision algorithm that combines outputs from various models. Moreover, the system's dependence on concurrent models imposed higher computational resource requirements.

To detect DDoS attacks in resource-constrained environments, Doriguzzi-Corin et al. [18] introduced a system named LUCID, using a CNN method. The authors evaluated the efficiency and suitability of their system using three different datasets: ISCX2012, CIC2017, and CICIDS2018. This system showed its applicability in real-time situations, but the measurements of preprocessing time are not clearly stated. Moreover, this model faced challenges in recognizing structured patterns in the datasets and failed to capture the sequence of events, which are often present in stealthy, low-volume attacks. Alashhab et al. [41] proposed a method for detecting low-rate DDoS attacks in IoT networks with SDN using DL. The method utilized RNNs with LSTM activation functions to analyze traffic and detect various low-rate DDoS attacks. The authors gathered datasets from Edge-IIoT, preprocessing them to handle missing values and normalize the data. The evaluation highlighted limitations such as the requirement for extensive training data and potential false positives.

C. ATTACK DETECTION USING ENSEMBLE APPROACHES

Gao et al. [42] developed an adaptive ensemble model that used a variety of base classifiers, such as DT, RF, KNN, and DNN. Using an adaptive voting algorithm, the best-performing classifier is selected and evaluated on the NSL-KDD dataset. While their model performed well in comparison to other models, it faced difficulty in

detecting low-rate attacks. Chakir et al. [43] conducted a study that evaluates various traditional ML methods, such as KNN, DT, and RF. During a comparison of single classifiers to ensemble classifiers, no specific method consistently excelled across all metrics. This study indicates that while traditional ML methods provide flexibility, they face challenges in the identification of specific types of anomalies.

Moustafa et al. [40] proposed an ensemble system to protect IoT network traffic against botnet attacks on HTTP and Message Queuing Telemetry Transport (MQTT) protocols. The system involved collecting network traffic using Zeek and TCPdump, followed by feature selection using correlation analysis. Using the UNSW-NB15 and NIMS botnet datasets, the authors employed an Adaptive Boosting (AdaBoost) ensemble classifier that used DT, naive Bayes, and ANN with boosting to distinguish between normal and attack traffic. This method showed enhanced accuracy, detection rate, and processing time, but primarily for high-volume attacks. Haider et al. [24] introduced a deep CNN framework to detect DDoS attacks in Software Defined Networks (SDN). While their ensemble model achieved good accuracy, it showed a trade-off in terms of training and testing times. Despite being superior in terms of detection, the increased processing time can hinder the appropriate attack mitigation and cause greater damage. Table 1 provides a comparison of different ML and DL-based approaches for attack detection, focusing on key parameters relevant to resource-constrained edge IoT networks.

The comprehensive review of existing literature on IoT network security using ML and DL approaches highlights several challenges. A common challenge is achieving a balance between detection accuracy and computational demands, as highlighted in different studies [28], [36], [38], especially in resource-constrained environments like edge IoT networks. While some ML approaches show proficiency in specific scenarios, they face challenges in detecting low-rate attacks and require refinement for diverse IoT contexts. Conversely, DL methods show promise in attack detection accuracy but face challenges due to high computational costs. Ensemble methods, which combine different approaches, offer a promising direction but struggle with accuracy and processing efficiency trade-off [24], [35], [40]. Additionally, there is a lack of detailed datasets that effectively include both high- and low-volume attack types [39] in IoT networks. The limitations of traditional packet-based features to identify low-rate attacks, resulting from similarities between attack and legitimate packets, has prompted investigations into alternative features [18], [44].

To address these challenges, the DEEPShield system is introduced in this paper, which is an ensemble learning-based attack detection system that uses flow features to ensure high accuracy for both high- and low-volume DDoS attacks. Furthermore, real testbed data is generated, consisting of both types of DDoS attacks. Additionally, to reduce computational demands while maintaining high accuracy, a pruning technique is implemented.

III. DEEP ENSEMBLE LEARNING WITH PRUNING (DEEPShield) SYSTEM

A. NETWORK AND SYSTEM ARCHITECTURE

As shown in Figure 1, the DEEPShield system is designed to be installed on edge devices that are connected wirelessly to IoT devices. During a security breach involving IoT devices launching attacks, the edge devices receive both normal and attack network traffic. Communication between edge devices and the central server, as well as among themselves, is made secure using TLS over TCP. The DEEPShield system has three modules that work together to detect and categorize normal and attack network traffic. The first module, referred to as network traffic analysis and preprocessing, collects network traffic packets and organizes them into groups called network flows, based on a specific time frame. The second module, referred to as the learning module, uses an ensemble model for the classification of normal and malicious network traffic. This module combines two DL models, CNN and LSTM, with hyperparameter optimization. The ensemble model is selected based on its precision and Mean Square Error (MSE). The third module, referred to as the model optimization module, aims to find a balance between computational complexity and the accuracy of the ensemble model using pruning. The pruning technique conducts the pruning of units/neurons in the ensemble model using the concept of sparsity.

B. ADVERSARY MODEL

The DEEPShield system is designed to detect both high- and low-volume DDoS attacks initiated by compromised IoT devices, as shown in Table 2. High-volume attacks, also known as flood attacks, aim to overwhelm a network by generating a large traffic volume. On the other hand, low-volume attacks include pulse attacks and low-rate attacks. Pulse attacks involve sending short bursts of dummy data to slow down the network. In low-rate attacks, the attacker imitates the behavior of legitimate IoT devices, resulting in a slight increase in network traffic.

C. NETWORK TRAFFIC ANALYSIS AND PREPROCESSING MODULE

This module collects packets and generates feature statistics through flow analysis. A custom network traffic analysis system is developed using Zeek [45] to convert packets into network flows within a defined time window (Δt). This system helps in data structuring and enhances the prediction model's capability to detect attacks. The traffic data is organized based on time and segmented into smaller units referred to as flows.

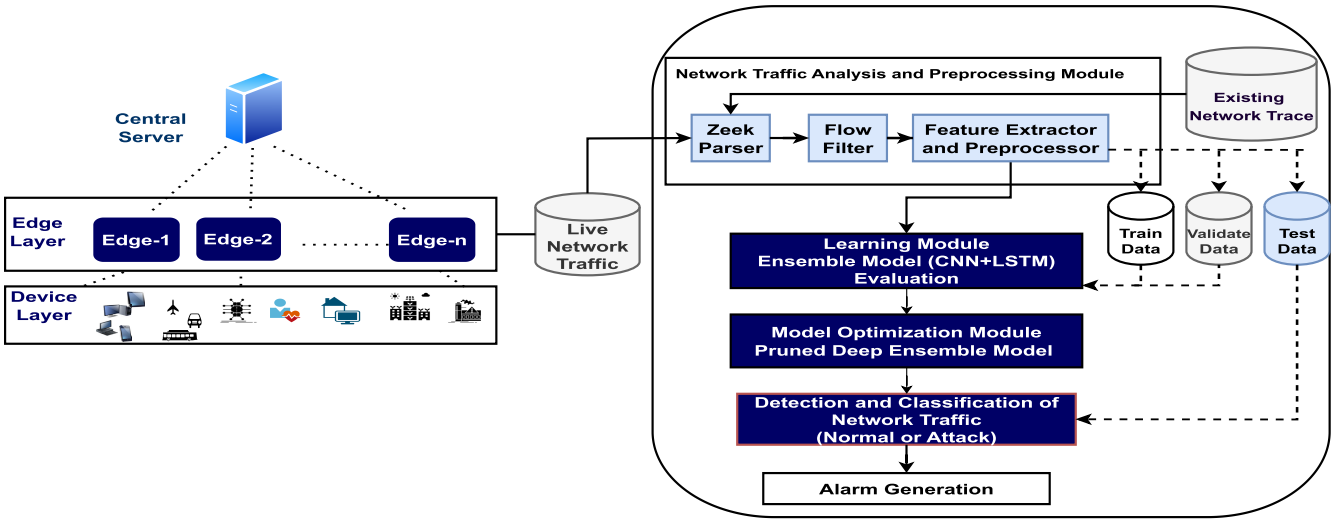

FIGURE 1. **Network and system architecture of the DEEPShield system.**

TABLE 2. **Summary of different types of DDoS attacks considered by the DEEPShield system.**

Attack	Attack Volume	Characteristics
TCP SYN flooding	High	Misuses the TCP handshake process with numerous fake requests, overloading the server's resources.
TLS flooding	High	Attackers flood the server with a large amount of garbage data encapsulated in the TLS protocol, causing resource exhaustion on the server.
UDP flooding	High	Sends many UDP packets to random ports on a target, causing the target to expend resources checking for non-existent applications.
TLS Pulses	Medium	Send intermittent bursts of garbage data within the TLS protocol, causing resource spikes that can degrade server performance over time.
TCP SYN low-volume	Low	A more subtle version of TCP SYN flooding with fewer requests, intending to slowly exhaust resources.
TLS flood low-volume	Low	Similar to TLS flooding but sends smaller amounts of garbage data over a longer period to gradually consume server resources.
Slowloris	Low	Keep multiple connections to the target server open by slowly sending incomplete HTTP requests.
Slow HTTP	Low	Sends HTTP headers slowly to engage server resources for extended periods.

1) NETWORK TRAFFIC ANALYSIS

The network traffic analysis procedure is described in Algorithm 1. This algorithm analyzes a stream of network packets and creates a features vector (F) for each connection, which is then used to train the ensemble model. The network traffic analysis algorithm requires two inputs: a sequence of network packets (P) and labels for each network flow (L). The algorithm also categorizes packets into TCP, UDP, ICMP, or IPv6 packet types. The packet direction, either forward (p_f) or backward (p_b), is determined by the source IP address. Each packet in the input stream is processed and their respective counters, packet count (p_c), and header count, are incremented based on the packet type and direction. According to the packet direction, the payload vector and the minimum, the maximum, and the total header size values are updated. For TCP packets, the algorithm increments counters for each set flag. The Inter-Arrival Time (IAT) is calculated for each new packet and stored in a vector (I). IAT represents the time gap between the arrival of the current packet and the previous packet. The algorithm analyzes each data packet to determine if it is a part of the bulk transfer, which refers to a sequence of packets sent in rapid succession. In case of a bulk transfer, the algorithm updates counters and stores a total number of packets, bytes, and time values for each packet direction in a vector of counts (B). Once all packets are processed, the algorithm computes specific features from the collected data as shown in Table 3. The mean function calculates the average of the vector of window sizes (W) and the vector of payload sizes (S). Unique flags are identified with the unique function on the flag vector (Fl). The features

TABLE 3. Summary of flow features for DDoS attack detection considered by the DEEPShield system.

Network Features	Description	Significance with DDoS volume
Packets	Total forward and backward packets, flow packets	High (numerous packets)
Duration	Flow duration	High (short duration), Low (long duration)
Header Size	Minimum, Maximum, and total forward and backward header size	Low (small header size)
Flags	FIN, RST, SYN, PSH, ACK, URG, CWR, ECE	High, Low
Payload	Minimum, maximum, total, average, and standard deviation of flow packets payload	High (large payload), Low (small or variable payload)
Inter Arrival Time	Minimum, a maximum, total, average, and standard deviation of flow IAT	High (shorter IAT), Low (variable IAT)
Bulk Packets	Forward and backward bulk bytes, bulk packets, and bulk rate	High (large bulk packets)
Packet Throughput	Packets per second	High (fast throughput), Low (slow throughput)
Window Size	Forward and backward initial and last window size	High (large window size)
Down up Ratio	Ratio of download packets to upload packets in a flow	High (high ratio)

are combined into a feature vector (F) for each connection and returned as the output. Furthermore, each flow is labeled based on the IP addresses of the attacker and the victim from the initial datasets. The algorithm further examines if the IP address of the attacker is in a predefined list and if the IP address of the victim matches a specific IP address. In case of a match, it is labeled as an attack (''1''); otherwise, labeled as normal (''0''). The collected features provide a complex context from which the model can learn and improve its predictive accuracy. For example, for high-volume DDoS attacks, key features are packets, throughput, payload, duration, and bulk packets [28]. Low-volume attacks can be indicated by features like duration, window size, flags, and payload [36]. Header size, down/up ratios, and flags are important features for both types of attacks.

2) NETWORK TRAFFIC PREPROCESSING

Algorithm 2 is designed to structure and preprocess network traffic data. The algorithm takes input as network flow features, F, consisting of n samples and m features. Initially, the algorithm arranges data based on timestamps and divides it into training, validation, and testing sets, proportioned as 80%, 10%, and 10%, respectively. Subsequently, the sorted data is divided into discrete frames, each covering a specific time window Δt. The algorithm processes each frame to calculate features and label sets, thereby converting them into structured matrices. The matrices are normalized using standard scalar methods to ensure that all features are within a comparable range. The algorithm addresses class imbalance concerns by oversampling the less-represented class in each set. As a result, a set of organized matrices is produced for training, validation, and testing. These matrices have the dimension of (n, f, m), where n is total sample count, f is flow number, and m is feature count across all time windows and these matrices are stored for future ML operations. The structured matrices capture a diverse range of network traffic features, like data volume (*"Packets", "Payload", "Bulk Packets"*), temporal properties (*"Duration", "Inter Arrival Time", "Window Size"*), data size (*"Header Size", "Payload"*), transmission control information (*"Flags"*), and traffic direction (*"Down up Ratio"*). Organizing data into timeframes while maintaining timestamp order retains the time-series context of the data, which is important for network traffic analysis since patterns tend to evolve over time. The duration of the time window is crucial for both accuracy and computational complexity. Shorter windows enable faster online DDoS attack detection, while longer windows provide more flow data, potentially increasing accuracy.

D. PACKET VERSUS FLOW-BASED FEATURES IN HIGH- AND LOW-VOLUME DDoS ATTACKS

Packet-based features analyze packet characteristics, like packet length, flags, protocols used, time intervals etc. [12]. However, these packet-based features are less effective in the context of low-volume attacks due to the covert techniques employed by attackers. Attackers attempt to blend in their attacks by spreading the attack over time, using unusual protocols, or imitating normal packet characteristics. This covert approach challenges the efficacy of packet-based features in detecting low-volume attacks [46], [47]. The attack volume from any single source becomes significantly smaller, making attack detection solely based on packet-based features challenging [48].

Flow-based features analyze entire flows of packets, aggregating data over a larger volume of network traffic. This method is useful in the detection of patterns that are overlooked during individual packet analysis [22]. In the case

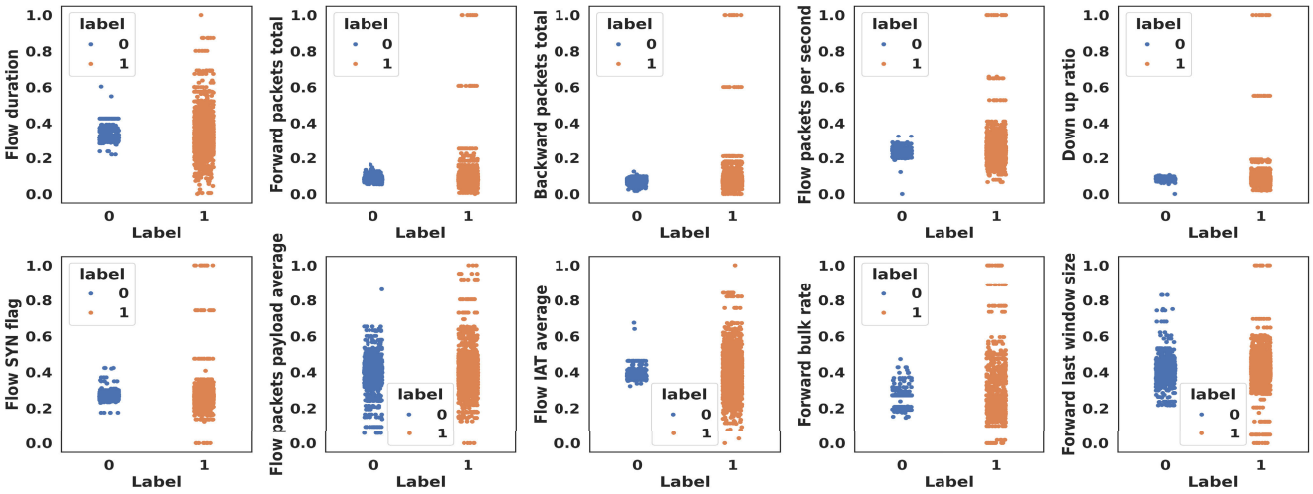

FIGURE 2. **Distribution of flow-based features.**

of low-volume attacks, where the distinctive packet-based features are absent, flow-based features like flow duration or total transmitted bytes show anomalies [46]. Moreover, flow-based methods are more efficient and scalable than packet-based methods, particularly in large networks. The flow-based features reduce the computational demands by grouping packets into fewer flows during analysis and capture long-term network traffic behaviors, useful for low-volume or stealthy attack detection. Figure 2 shows the distribution of various flow-based features within normal and attack network traffic as analyzed by the DEEPShield system, presented using a pair strip plot. Each dot on the strip plot signifies a single data point. In the case of normal network activity, the dots denoting diverse flow-based features are clustered around specific values, showing the expected behavior under normal network conditions. For an attack network, the distribution of the same flow-based features diverges. The dots biased toward higher values indicate a noticeable increase in feature values, signifying an ongoing attack. The uniform dot distribution suggests a stealthier attack behavior that resembles normal network traffic.

The chosen features are validated using the Mann-Whitney U test [27]. This test determines the difference between attack and normal traffic, as described in Algorithm 3. The Mann-Whitney U test, being non-parametric, is well-suited for analyzing data that does not follow a normal distribution, which is common in network traffic analysis. Results show notable differences between attack and normal traffic, affirming the effectiveness of flow-based features in DDoS detection. A significance level (alpha) of 0.05 is selected, indicating a 5% chance of falsely detecting differences that are not there. This level strikes a balance in network security, preventing overestimation of DDoS attacks. It ensures sensitivity to real threats while avoiding unnecessary responses to false alarms, ensuring reliable detection of feature differences like flow duration and packet sizes. However, it is important to note that the Mann-Whitney U test does not provide detailed insights into the magnitude of these differences.

The validation process for chosen features is detailed in Algorithm 3. The hypotheses under testing are as follows:

- Null hypothesis (H0): There is no difference between the *"Attack"* and *"Normal"* groups for a given feature.
- Alternative hypothesis (H1): There is a difference between the *"Attack"* and *"Normal"* groups for a given feature.

E. LEARNING MODULE

The ensemble model, consisting of CNN and LSTM models, utilizes both spatial and temporal characteristics of the input data. The CNN model focuses on identifying spatial patterns and relationships between features [18], which helps in recognizing various DDoS attack vectors like TCP SYN flooding and UDP flooding. These attacks have distinct traffic patterns that the CNN model can effectively detect. The CNN model processes data hierarchically, using sequential layers and filters to uncover spatial relationships. In contrast, the LSTM model component specializes in detecting temporal patterns and sequences in network traffic flows [20], making it invaluable for identifying the subtle and prolonged activities typical of low-volume attacks like Slowloris and Slow HTTP. The LSTM model has ability to process sequential data and identify patterns over time, which is important to analyze flow features related to DDoS attacks, such as duration, packet throughput, and inter-arrival times. These components provide a comprehensive understanding of attack patterns, thereby enhancing detection accuracy for both high-volume and low-volume DDoS attacks. The ensemble approach is essential to accurately detect and classify the DDoS attacks, particularly as attackers adopt more sophisticated and multi-vector strategies. The model

Algorithm 1 Network Traffic Analysis

Require: A stream of packets $P = \{P1, P2, \ldots, Pn\}$, Predefined attacker IP address set A, Victim IP address V
Ensure: Features vector F for each connection, label L for each connection

Initialize $P_f, P_b, P_c, I, B, W, S, Fl = 0$
for each P_i in P **do**
  if P_i is forward **then**
    $P_f = P_f + 1$
  else
    $P_b = P_b + 1$
  end if
  $P_c[\text{type}(P_i)] = P_c[\text{type}(P_i)] + 1$
  if P_i is TCP **then**
    $B[\text{type}(P_i)] = B[\text{type}(P_i)] + 1$
    $W = \text{append}(W, \text{window\_size}(P_i))$
    $S = \text{append}(S, \text{payload\_size}(P_i))$
    $Fl = \text{append}(Fl, \text{flags}(P_i))$
  end if
  $IAT = t(P_i) - t(P_{i-1})$
  $I = \text{append}(I, IAT)$
  if P_i is last packet **then**
    break
  end if
end for
Calculate features: P_c/T_{tot}: Packets per second, P_b/P_f: Down/up ratio, B/T_{tot}: Bulk rates, $\mu(W)$: Average window size, $\mu(S)$: Average payload size, Unique(Fl): Unique flags
$F = \{P_c/T_{\text{tot}}, P_b/P_f, B/T_{\text{tot}}, \mu(W), \mu(S), \text{Unique}(FL)\}$
if origin_IP(P) in A and dest_IP(P) == V **then**
  $L = 1$ (DDoS)
else
  $L = 0$ (Normal)
end if
return F and L

Algorithm 2 Network Traffic Pre-Processing and Balancing

Require: Flows: $F \in \mathbb{R}^{n \times m}$ (n samples, m features)
Ensure: Balanced Flows: F_{train}, andF_{val}, F_{test}

Load F to DataFrame df.
if necessary **then**
  Transform labels L in df: Apply $L \rightarrow \{1, 2, \ldots, |L|\}$ where $|L|$ is the number of unique labels.
end if
Sort df based on timestamp "ts".
Partition (X, Y) into $(X_{\text{train}}, Y_{\text{train}})$, $(X_{\text{val}}, Y_{\text{val}})$ and $(X_{\text{test}}, Y_{\text{test}})$ with ratios 0.8, 0.1, 0.1 respectively.
Group df into frames $R = \{f1, f2, \ldots, fn\}$ each of duration Δt seconds.
for each fi in R **do**
  Compute Xi (feature set) and Yi (label set).
end for
For xi in Xi, compute $x'i$ in $X'i$ where $x'i = (xi - \mu)/\sigma$.
Perform oversampling for each set: Let $n0, n1$ be numbers of samples in classes $C0, C1$. Duplicate samples from the minority class until $n0 = n1$.
return $F_{\text{train}} = (X_{\text{train}}, Y_{\text{train}})$, $F_{\text{val}} = (X_{\text{val}}, Y_{\text{val}})$, $F_{\text{test}} = (X_{\text{test}}, Y_{\text{test}})$ as separate HDF5 files, The files has n samples, m features and f flows, its shape is (n, f, m).

Algorithm 3 Mann-Whitney U Test

Require: DataFrame df, Significance level α
Ensure: Feature-wise U test results

Set $\alpha = 0.05$
Split df into two groups, $G1$ ("DDoS") and $G2$ ("Normal"), based on "label"
for each feature m in df (excluding "label") **do**
  Compute U statistic and p-value between $G1[m]$ and $G2[m]$
  if $p > \alpha$ **then**
    Same distribution (fail to reject H0)
  else
    Different distribution (reject H0)
  end if
end for
return Feature-wise U test results

effectively detects both volumetric attacks, which flood the network with excessive traffic, and application-layer attacks, which involve advanced patterns of requests designed to exploit specific vulnerabilities.

Figure 3 shows the model architecture of the DEEPShield system. Further details regarding the DEEPShield ensemble model architecture, along with insights into the model training process including the loss function and optimization strategies, are provided below.

1) INPUT LAYER

The network traffic analysis and preprocessing module organizes the network traffic into a 2-D matrix denoted as M, by structuring flow vectors. The matrix M, with dimensions $f \times m$, serves as input to the proposed model. M is a matrix containing flow vectors: $M = \{f_1, f_2, \ldots, f_n\}$. Here, each flow vector f_n represents the n^{th} flow and has a width of m elements.

2) CNN LAYER

The CNN layer starts with a convolutional layer where a group of kernel filters, denoted as r, is applied to the input matrix M. Each filter or kernel, possesses dimensions of $f \times m$, with f representing the filter height and m, filter width. During convolution, each kernel slides over the input matrix M using a stride of 1 and calculates the dot product between the filter and the covered input patch. This process extracts important localized features for subsequent classification tasks. Each filter generates an activation map a with

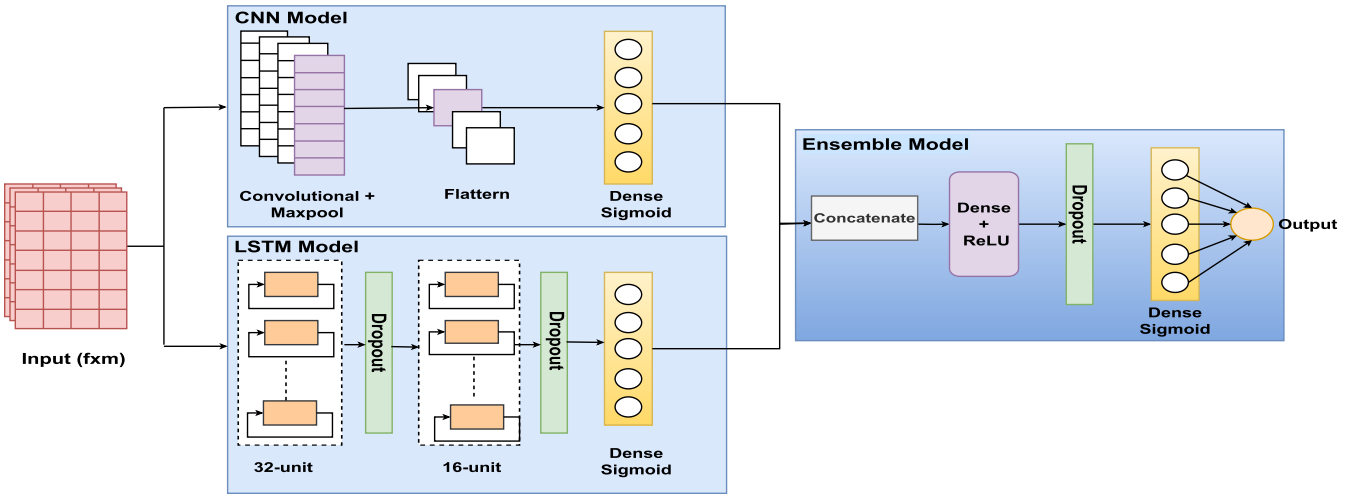

FIGURE 3. **The DEEPShield system ensemble model architecture.**

dimensions $M.\text{shape}[0]-f+1$ by $M.\text{shape}[1]-m+1$,

$$a = \text{ReLU}(\text{Conv}(M, W_r) + b_r), \tag{1}$$

where W_r and b_r represent the learned weight and bias parameters for the rth filter. Here, $\text{Conv}(M, W_r)$ represents the convolution operation performed on the input matrix M. The Rectified Linear Unit activation function (ReLU), is used to introduce non-linearity into a model and to capture complex patterns effectively. All the activation maps are stacked to produce an activation matrix Z of size $(M.\text{shape}[0]-f+1, M.\text{shape}[1]-m+1, r)$,

$$Z = [z_1|\cdots|z_k], \tag{2}$$

in the presence of a specified dropout value, a dropout layer is sequentially added. This layer randomly sets a fraction of the input units to zero during each training update and prevents overfitting. This is followed by a GlobalMaxPooling2D layer that reduces the spatial dimensions of its input tensor by selecting the maximum value over the entire height and width for each channel, thereby preserving the most important features. The output of this layer, denoted as O_{pool}, is determined from the tensor Z. Here, each element of Z corresponds to the output of a ReLU activation function applied to the convolution operation between the input matrix M and the corresponding filter,

$$O_{\text{pool}} = \max(Z). \tag{3}$$

Finally, the output is flattened into a one-dimensional array and passed through a dense (fully connected) layer. This layer employs a sigmoid activation function for binary classification. The resulting outcome of the CNN model is,

$$O_{\text{cnn}} = \text{sigmoid}(\text{Dense}(O_{\text{pool}})), \tag{4}$$

where $\text{Dense}(O_{\text{pool}})$ represents a dense layer applied to the pooled output, and sigmoid is the activation function.

3) LSTM LAYER

The LSTM model is constructed using two LSTM layers, each layer operates on gating mechanisms, including the forget gate f, input gate i, and output gate o. Gates are controlled during training by specified weight matrices and biases. The ReLU activation function [49] is used for both cell state updates and gate activation. Using the HICT dataset, the implementation of ReLU in LSTM gates and cell state updates resulted in greater accuracy and faster convergence than traditional *"tanh"* [49] activation function.

- **Forget Gate** (f_t): Determines the amount of information from the previous cell state to retain by the following relation:

$$f_t = \sigma(W_f[h_{t-1}, X_t] + b_f). \tag{5}$$

- **Input Gate** (i_t): Controls how much of the newly computed information for the current input will be stored in the cell state,

$$i_t = \sigma(W_i[h_{t-1}, X_t] + b_i). \tag{6}$$

A tentative addition to the cell state, C'_t, is computed simultaneously using the ReLU function,

$$C'_t = \text{ReLU}(W_c[h_{t-1}, X_t] + b_c). \tag{7}$$

The cell state is then updated using the outputs from the forget gate and the input gate,

$$C_t = f_t C_{t-1} + i_t C'_t. \tag{8}$$

- **Output Gate** (o_t): Decides how much information from the cell state gets exposed to the next layer,

$$o_t = \sigma(W_o[h_{t-1}, X_t] + b_o). \tag{9}$$

Finally, the hidden state for the current time step is computed by scaling the cell state by the output of the

output gate after passing through the ReLU function,

$$h_t = o_t \times \text{ReLU}(C_t), \tag{10}$$

where $[h_{t-1}, X_t]$ represents the combination of the previous hidden state and the current input X_t and σ represents the sigmoid function. During training, the model learns the weights (W) and biases (b) linked to the different gates and operations. To prevent overfitting, dropout layers are inserted after each LSTM layer. The model output is passed through dense layers of sigmoid activation functions designed for binary classification. The final output of the LSTM model is expressed as follows:

$$O_{\text{lstm}} = \text{sigmoid}(\text{Dense}(h_t)). \tag{11}$$

4) ENSEMBLE LAYER

In the ensemble layer, the outputs from the LSTM model, O_{lstm}, and the CNN model, O_{cnn}, are concatenated to produce a combined feature vector $O_{Ensemble}$.

$$O_{Ensemble} = concatenate(O_{lstm}, O_{cnn}). \tag{12}$$

$O_{Ensemble}$, is then passed through a dense layer and a ReLU activation function.

$$O_{dense} = ReLU(Dense(O_{Ensemble})), \tag{13}$$

thereafter, a dropout layer is added to the model to prevent overfitting. This layer randomly sets a fraction of the input units to 0 at each update during the training phase, thereby creating a more generalized model. This operation is represented by the following relation:

$$O_{dropout} = Dropout(O_{dense}). \tag{14}$$

Finally, the output of the ensemble model is generated by a dense layer with a sigmoid activation function for binary classification. This can be represented as,

$$O_{final} = sigmoid(Dense(O_{dropout})). \tag{15}$$

5) CLASSIFICATION LAYER

In the DEEPShield system, each model employs a similar structure for its final classification layer. The output from the previous layers of each model is directed through a fully connected (dense) layer, which linearly transforms the input to produce its activation. The final output of models, O_{final}, indicates the probability of the input belonging to the positive class. The value of O_{final} approaching 1 suggests the higher chances of input representing a malicious DDoS attack. The definitive classification for each model is determined by a threshold set on O_{final}. If $O_{\text{final}} > 0.5$, the input is categorized as positive, indicating a DDoS attack. Conversely, values less than the threshold classify the input as normal.

a: MODEL TRAINING

The training process updates the model weights and biases iteratively to minimize a loss function. The loss function measures the discrepancy between model predictions and actual labels. The training persists until the model converges or reaches the predefined maximum number of epochs. The objective of the training is binary classification, aiming to differentiate between DDoS attacks and normal traffic. To achieve this, a binary cross-entropy loss function is used, which is given by:

$$C_{bs} = \frac{1}{B}\sum_{i=1}^{B} -\left(y_i \log(p_i) + (1-y_i)\log(1-p_i)\right), \tag{16}$$

where y_i is a true label and p_i is the predicted probability of a network event i being a malicious DDoS attack in a batch of B samples. Unlike many existing methods, the proposed model learns directly from the raw network traffic data, thus needing less expert intervention and improved scalability. Furthermore, the supervised learning task uses balanced datasets with an equal number of malicious and benign events, which helps reduce learning bias. In addition, ADAM optimizer is used with hyperparameters for model optimization. The learning rate is adjusted to balance convergence rate with model stability. Model training follows a batch-based approach for efficient training on large datasets and faster convergence.

F. THE MODEL OPTIMIZATION MODULE

The ensemble model, integrating CNN and LSTM outputs, naturally expands in size, affecting memory usage and prediction times in IoT networks. Hence, optimization is crucial for efficiency and real-time responses. Unit pruning is a highly effective strategy for model optimization and enhancing efficiency without sacrificing prediction accuracy. It targets individual units in neural networks, significantly reducing computational complexity and resource demands. By removing less critical connections, memory footprint is reduced, making it suitable for devices with limited storage. This process involves the identification and removal of connections with the smallest L2 norms across weight matrices. Connections with minimal impact are either preserved entirely or removed altogether from specific units. As a result, the model becomes more adaptable and efficient in real-time DDoS attack detection [25]. This process is summarized as:

$$W_{\text{pruned}} = W * (|U| \geq \text{kth\_smallest}(|U|)), \tag{17}$$

where W is the weight matrix, k is the sparsity level, and U is the sum of absolute weights for each unit. The term $|U| \geq \text{kth\_smallest}(|U|)$ generates a boolean mask that assigns a value of true to all units where the sum of absolute weights exceeds the kth smallest sum of absolute weights in U. The following objective function represents the trade-off among model size ($Size$), accuracy (Acc), and prediction time (P_t),

$$J(E_m) = \alpha * \text{Size}(E_m) + \beta * (1 - \text{Acc}(E_m)) + \gamma * P_t(E_m), \tag{18}$$

where α, β, and γ serve as weighting coefficients, indicating the relative significance of ensemble model (E_m) size,

accuracy (ACC), and prediction time (P_t), respectively. The sparsity level k is formulated as:

$$k = \frac{\text{Number of zero weights in } W_{\text{pruned}}}{\text{Total number of weights in } W_{\text{original}}}. \quad (19)$$

The model optimization via pruning aims to minimize the model size and prediction time while maximizing model accuracy. This scenario can be conceptualized as a multi-objective optimization problem,

$$\begin{aligned} &\text{Minimize: } J(E_m), \\ &\text{Subject to: } \text{Acc}(E_m) \geq \text{threshold}. \end{aligned} \quad (20)$$

The underlying challenge resides in selecting the optimal k^* that satisfies the delineated objectives. This is captured by,

$$k^* = \arg\min_{k \in K} J(E_m). \quad (21)$$

In the Equation 21, the objective is to identify the most suitable sparsity level, k, from a given set of possible levels, K, that minimizes the objective function, J. By applying pruning at various levels, k, from the provided set, K, the optimal model, $E_m^* = PM(E_m, k^*)$, is determined. This method helps to find a balance between model dimensions, prediction time, and accuracy.

IV. RESULTS AND EVALUATION

A. EXPERIMENTAL SETUP

This section describes the testbed designed to evaluate the performance of the DEEPShield system in detecting high- and low-volume DDoS attacks in IoT networks. As shown in Figure 4, the testbed shows 11 Raspberry Pis operating as IoT devices. Data collected by IoT devices is transmitted to two edge servers. Python scripts and the hping3 tool [50] are used to execute attack scenarios. For high-volume attacks, the aim is to simulate DDoS by flooding edge servers and making them unresponsive to legitimate requests. On the other hand, in low-volume attacks, the compromised devices are made to transmit data at rates 20% - 40% higher than the normal network traffic.

In the testing environment, standard IoT devices transmit data at random intervals ranging from 1 to 10 seconds. Based on the DDoS attack type, compromised devices transmit data at significantly faster rates. To test the ability of the DEEPShield system in detecting compromised IoT devices, 30% to 40% of the IoT devices are presumed compromised. As described in the network traffic analysis and preprocessing module, a custom Zeek [45] script, following algorithm 1 is used to generate flow-related features.

The DEEPShield system is validated using a self-generated dataset, named HL-IoT (covering both high- and low-volume attacks), and publicly available datasets, CICIDS2017 [51], ISCX2012 [52], and ToN-IoT [53]. The HL-IoT dataset records 8 hours of normal network traffic from IoT devices, which is transmitted to both edge servers and stored in a Packet Capture (PCAP) file. The following day, high-volume

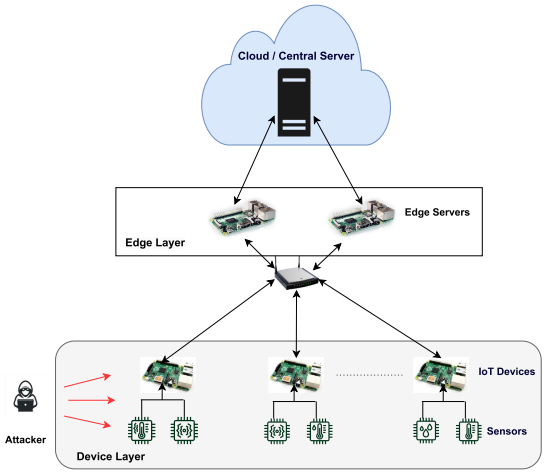

FIGURE 4. The DEEPShield system testbed setup [2].

TABLE 4. Raw data files and network logs size of HL-IoT dataset.

Name	Raw Data Size	Network Logs Size	Application Protocol
HL-IoT-High	1.3GB	Total logs: 10 (1.7GB)	Custom
HL-IoT-Low	584MB	Total logs: 10 (187MB)	Custom
HL-IoT-MQTT	545 MB	Total logs: 6 (1.4 GB)	MQTT

DDoS attacks are initiated and data is collected for 25 minutes during the TLS flood attack and 15 minutes for the TCP-SYN flood attack. On the third day, low-volume DDoS attacks are introduced, collecting 4 hours of traffic for both TCP-SYN and TLS low-volume attacks. The HL-IoT dataset includes subsets HL-IoT-High, HL-IoT-Low, and HL-IoT-MQTT, differing in size and protocol usage, as outlined in Table 4. In the ToN-IoT dataset, the DEEPShield system considers normal network traffic from February 4, 2019, and attack traffic from April 24, 2019. The CICIDS 2017 dataset includes data from diverse devices, capturing both normal and compromised network behavior. This dataset, analyzed by the Canadian Institute for Cybersecurity (CIC), recorded high-volume attacks on July 3, 2017, and low-volume attacks on July 7, 2017. For validating the ensemble model, another dataset called the ISCX2012, collected by CIC and the University of New Brunswick, is also used. This dataset focuses on data from June 15, 2012, which includes an IRC botnet attack.

B. EXPERIMENTAL DATASETS

All datasets accurately depict both legitimate user and attacker behaviors. The CIC datasets (CICIDS, ISCX) use real traffic traces including protocols such as HTTP, SMTP, SSH, IMAP, POP3, and FTP. HL-IoT introduces its custom

TABLE 5. Datasets with attack types and flows.

Dataset	Attack Type	Attack Volume	Benign Flows	DDoS Flows
HL-IoT	TCP_SYN Flood, TLS Flood, TLS Pulses	High	295194	730524
HL-IoT	TCP_SYN low, TLS Low	Low	295194	18153
ToN-IoT	TCP_Flood, UDP_Flood	High	65357	983214
ISCX	IRC botnet	High	534320	37378
CICIDS 2017	HTTP DDoS	High	97718	128027
CICIDS 2017	Slowloris, Slow HTTP	Low	440031	11295

protocol alongside HTTP and MQTT, while ToN-IoT predominantly focuses on MQTT and HTTP. Moreover, these datasets simulate attacker behavior through commonly used tools like hping3, Python scripts, High Orbit Ion Cannon (HOIC), and its precursor, the Low Orbit Ion Cannon (LOIC). Table 5 provides a summary of attack types and data volumes analyzed by the DEEPShield system across all the mentioned datasets. The CIC datasets are chosen for their diverse coverage of DDoS attacks, while the ToN-IoT dataset focuses only on high-volume attacks.

C. DATASET PREPARATION

A novel dataset named HICT is created using the method introduced by Doriguzzi-Corin et al. [18]. The HICT dataset combines data from HL-IoT, ISCX, CIC, and ToN-IoT sources. The datasets are merged because the models trained on a single dataset often exhibit reduced efficiency when applied to other diverse datasets. The aim is to develop a model capable of effectively learning from diverse datasets and subsequently showing strong performance on novel and diverse data. The HICT dataset is compiled using randomly selected 98928 DDoS traffic and an equal number of normal ToN-IoT traffic. The procedure is repeated for 29196 high-volume CICIDS-2017 records, 78498 low-volume CICIDS-2017 records, 63008 ISCX2012 records, 85824 high-volume HL-IoT records, and 26912 low-volume HL-IoT records. After collecting network traffic data, flows are converted into a 2D format. Each data source is divided by: 80% for training, 10% for validation, and 10% for final testing. The 80-10-10 split is commonly used in ML for training, validation, and testing datasets. Allocating 80% to training provides the model to discover patterns and enhance its generalizability. The 10% validation set aids in hyperparameter tuning, preventing overfitting. The separate 10% testing set is for an unbiased evaluation and assess the model's ability to generalize to new, real-world data. Training sets from various sources are combined in equal proportions to create a detailed training dataset. Similar steps are taken for the validation and testing, leading to the final HICT dataset with 496000 flows for training, and 62000 flows each for validation and testing.

D. HYPERPARAMETER TUNING

Optimizing hyperparameters enhances model accuracy by shaping its complexity and learning process. Initial values are selected based on preliminary testing and specific rationales for each parameter. Grid search is used to explore different hyperparameter combinations, including learning rate, batch size, kernels, regularization, and dropout rates, aiming to optimize model performance. A grid search strategy is then used to navigate the hyperparameter space using the F1 score, serving as a key performance metric. In this phase, training persisted for 25 epochs with static loss. After capturing the F1 score at each grid function, the system advanced to the subsequent point. The HICT dataset is segmented into training, validation, and test subsets. The validation set has undergone a process of hyperparameter optimization, while the test set is reserved to produce the final results. The number of convolutional filters, r, is adjusted in incremental steps using powers of 2, from $r = 1$ to $r = 64$, resulting in an increased F1 score, due to the increased number of trainable parameters in the model.

After a detailed evaluation of 2835 combinations of hyperparameters, the optimal CNN configuration for the HICT validation set is identified as follows: $f = 100$, $\Delta t = 5$ seconds, and $r = 64$. The model uses a batch size of $B = 1024$ and the ADAM optimizer with a learning rate of $\alpha = 0.001$, resulting in a total of 8577 trainable parameters. For the LSTM, the configuration comprises $f = 100$ and $\Delta t = 5$ seconds, resulting in a model with 13065 trainable parameters. Alternative configurations have the potential to be more resource-efficient with only a slight reduction in the F1 score. Opting for $r = 32$, for instance, cuts the number of convolutions in half while choosing $f = 10$, 20, or 50 decreases convolutional demands and memory usage. A configuration of $f = 100$, however, not only optimized the F1 score but has also allowed for a detailed comparison with the work by Doriguzzi-Corin et al. [18]. The hyperparameters remained consistent throughout the experimental process.

E. EVALUATION METRICS

A range of metrics is used to validate the selected features, evaluate model performance, determine optimal sparsity, and compare with state-of-the-art methods. These metrics help in identifying differences in feature behavior during normal and attack networks. To validate the selected features as discussed in section III-C, the Mann-Whitney U test is performed, with the p-values serving as indicators to distinguish data distributions. Additionally, the preprocessing time associated with network traffic analysis and preprocessing is detailed. The evaluation matrics include: **Accuracy** ($ACC = \frac{TP+TN}{TP+TN+FP+FN}$), **FPR** ($FPR = \frac{FP}{FP+TN}$), **Recall (TPR)** ($TPR = \frac{TP}{TP+FN}$), **Precision** ($Precision = \frac{TP}{TP+FP}$) and

F1 Score ($F1 = \frac{2\times(Precision\times Recall)}{Precision+Recall}$). Various other metrics, such as sparsity (K), prediction time, preprocessing time, and model size, are measured. The memory usage and processing rate of the pruned ensemble models are also evaluated. The DEEPShield system trained three distinct models: the CNN model (DEEPShield-CNN), the LSTM model (DEEPShield-LSTM), and an ensemble ((DEEPShield-Ensemble) model, which is a combination of the CNN and LSTM models. Model training and validation are conducted on a computer equipped with a 20-core Intel i9-10900KF @3.70GHz CPU and 64GB of RAM. The models are implemented in Python v3.10.9 using the Keras API v2.12.0 on top of TensorFlow 2.12.0.

F. RESULTS AND DISCUSSION

Figure 5 shows a comparison of network traffic analysis and preprocessing time between the DEEPShield system and the LUCID system proposed by Doriguzzi-Corin et al. [18]. This figure also highlights preprocessing times for individual datasets used in the DEEPShield system. The DEEPShield system outperforms the LUCID system in terms of time across all datasets. Across all datasets collectively, the DEEPShield system processes data faster than LUCID, resulting in time reductions ranging from 14.7% to 74.9%. The most notable differences are observed in the ToN-IoT and ISCX datasets, with the DEEPShield system being more time-efficient. Given that the ISCX and the ToN-IoT datasets are larger in comparison to others, their processing times are longer. The network traffic preprocessing system used in LUCID prioritizes traffic flow conversion to array-like configurations and segments them into sub-flows based on time windows, thereby creating a spatial representation of the network traffic data. Whereas, the DEEPShield system, in collaboration with Zeek, efficiently structures network data and partitions it in time-framed flows. This method structures the traffic data and represents it chronologically. This approach not only organizes data effectively but also reduces computational load, highlighting the time efficiency of the DEEPShield system compared to LUCID.

Based on the p-values shown in Table 6 for both combined and individual datasets, the selected features exhibit a significant difference between normal and attack traffic when p-values are less than 0.05. In high-volume datasets, flow duration, flags, and window size consistently show significant differences between normal and attack patterns. However, packets, packet throughput, and IAT show smaller variations. In low-volume datasets, flow duration, flags, and IAT distinctly differentiate normal patterns from attack patterns, while header size, packet payload, and packet throughput show smaller variations. In mixed-volume attacks (both high and low), duration, flags, and window size show significant differences, whereas packets and the down-up ratio are comparatively less significant. Overall, features like duration, flags, and window size remain universally significant across different attack volumes, while the significance of packets, packet throughput, and IAT varies depending on the attack volume.

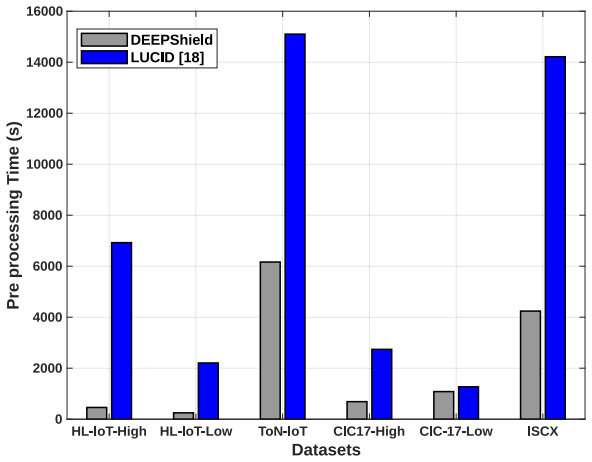

FIGURE 5. Network traffic analysis and preprocessing time (seconds).

TABLE 6. P-Values of selected features across datasets.

Features (m)	HICT (High-Low)	ToN-IoT (High)	CIC-17 (High)	ISCX (High)	HL-IoT (High)	HL-IoT (Low)	CIC-17 (Low)
Packets	0.45	0.04	0.00	0.03	0.00	0.25	0.001
Duration	0.00	0.00	0.00	0.01	0.001	0.00	0.001
Header Size	0.001	0.00	0.00	0.001	0.10	0.28	0.19
Flags	0.00	0.00	0.001	0.00	0.00	0.003	0.00
Payload	0.005	0.00	0.00	0.00	0.73	0.24	0.00
Inter Arrival Time	0.037	0.018	0.11	0.15	0.020	0.001	0.00
Bulk packets	0.003	0.00	0.00	0.42	0.00	0.85	0.00
Packet throughput	0.027	0.01	0.002	0.44	0.011	0.007	0.001
Window size	0.007	0.003	0.001	0.004	0.013	0.11	0.034
Down up ratio	0.30	0.87	0.00	0.11	0.015	0.009	0.00

The DEEPShield system includes four DL models: DEEPShield-CNN, DEEPShield-LSTM, DEEPShield-Ensemble, and after post-training optimization process applied to DEEPShield-Ensemble, the system has introduced the optimized Pruned Ensemble model, referred to as DEEPShield-PE. The results shown in this section are based on a test set comprised of entirely unseen data. Following the assessment of accuracy and prediction time using the HICT dataset, the DEEPShield-PE model stands out as the optimal trade-off. Figure 6 illustrates the accuracy and prediction time of the DEEPShield system models, along with the

LUCID model by Doriguzzi et al. [18], Ensemble-LSTM and Ensemble-CNN models by Haider et al. [24], and the LSTM model by Alashhab et al. [41] for low-rate DDoS attack. As shown in Figure 6 the DEEPShield-PE model achieves an accuracy of 93% with a prediction time of 1.24 seconds, which is less than the DEEPShield-Ensemble model and the Ensemble-LSTM model. Although the DEEPShield-PE model shows a small accuracy deviation (1.08%) from the DEEPShield-Ensemble, Low-LSTM and Ensemble-LSTM model, it compensates by offering improved prediction time and less memory requirements. The DEEPShield-PE model shows a 12.95% less prediction time compared to the DEEPShield-Ensemble model and a 5% less prediction time compared to the Low-LSTM model. In comparison to the Ensemble-LSTM model, the DEEPShield-PE achieves an improvement in prediction time of 40.67%. Additionally, in terms of memory requirements, the DEEPShield-PE model shows a significant reduction compared to other ensemble models. While the DEEPShield-Ensemble model requires 105,430 bytes of memory, the DEEPShield-PE model only requires 88,620 bytes, indicating a reduction of approximately 15.9% in storage memory. Compared to other ensemble models like Ensemble-LSTM with a memory requirement of 106,572 bytes and Low-LSTM with a requirement of 163,460 bytes, the DEEPShield-PE model offers considerable memory efficiency. However, it is important to note that Table 8 shows memory requirements during real network inference, highlighting the practical efficiency of the DEEPShield-PE model. Table 7 shows the performances of the DEEPShield system models with LUCID, Ensemble-CNN, and Ensemble-LSTM models using metrics, F1, TPR, FPR and Precision across HICT and individual datasets. Both HICT and individual datasets are considered to assess model performance concerning attack type, given the datasets' distinction between high- and low-volume DDoS attacks. The DEEPShield-Ensemble model outperforms for both high- and low-volume attack types. The CNN-based model, LUCID [18], is the only one with an accuracy below 85% for both attack types. The DEEPShield-PE model shows a 1% and 3% increment in F1 score over the Ensemble-LSTM [24] model for HL-IoT (Low) and CICIDS-17 (Low) datasets, respectively. Although the Ensemble-LSTM and Low-LSTM [41] models show results comparable to the DEEPShield-Ensemble and DEEPShield-PE models, it comes at a computational cost, with its prediction time 40% and 17% higher respectively. Furthermore, the DEEPShield-Ensemble model achieves FPR ranging from 1% to 6%, ensuring an accurate classification of normal network activities and minimizing the probability of false alarms. The DEEPShield system ensures a balance between reducing real threats and minimizing false alarms, which is essential for DDoS detection in different network attack situations. The DEEPShield system boosts security by focusing on false alert reduction to keep networks running smoothly while effectively detecting harmful traffic. Across datasets like HL-IoT, ToN-IoT, ISCX, and

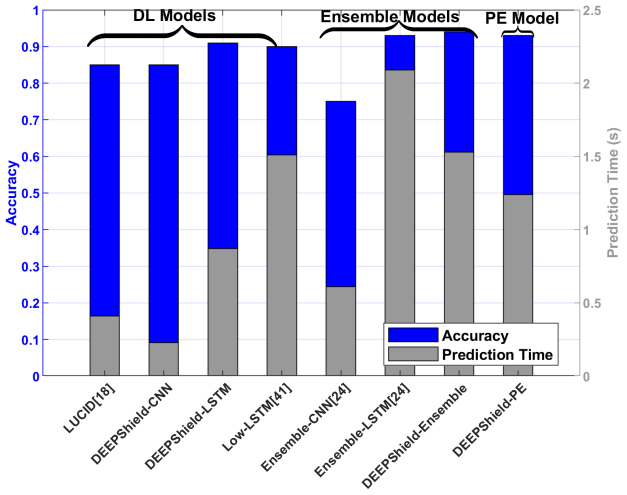

FIGURE 6. Accuracy and prediction time (seconds) on HICT dataset.

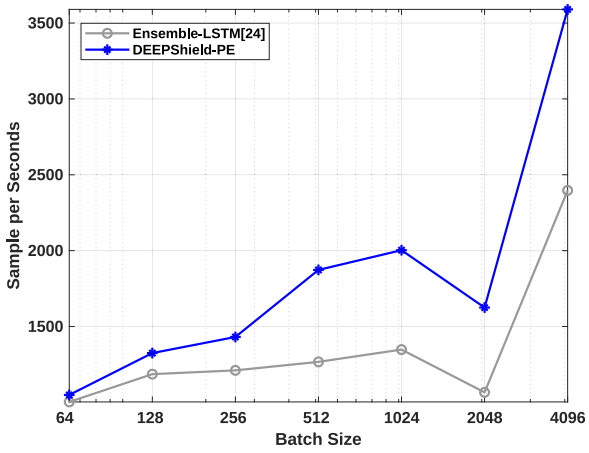

FIGURE 7. Inference performance on a raspberry Pi-4, 4GB, 1.5GHz 64-bit quad-core Cortex-A72 processor.

CICIDS 2017, the DEEPShield-Ensemble model consistently achieves high accuracy in detecting actual attacks while keeping false alarms low. Comparatively, while models like LUCID [18] and Ensemble-LSTM [24] have shown competencies, with LUCID achieving a TPR of 0.95 in the HL-IoT (High) but struggles with an FPR of 0.65 in HL-IoT (Low). However, the DEEPShield system's ensemble approach maintains a balance between sensitivity and specificity. The DEEPShield-Ensemble and DEEPShield-PE consistently achieve over 90% precision across all datasets, outperforming other models. The DEEPShield-PE improves precision by up to 3% compared to Ensemble-LSTM in specific datasets, accurately identifying malicious traffic and reducing the chance of misclassifying normal traffic as attacks.

Figure 7 shows the performance metrics of the DEEPShield system, when deployed on a Raspberry Pi 4, emphasizing the number of samples processed per second.

TABLE 7. Performance metrics of different systems across datasets.

System	Metrics	HL-IoT (High)	HL-IoT (Low)	ToN-IoT	ISCX	CICIDS-17 (High)	CICIDS-17 (Low)	HICT
LUCID [18]	F1	0.97	0.61	0.89	0.88	0.93	0.32	0.85
	TPR(Recall)	0.95	0.46	0.83	0.82	0.99	0.19	0.77
	FPR	0.01	0.65	0.02	0.04	0.14	0.10	0.09
	Precision	0.99	0.91	0.96	0.95	0.88	0.91	0.95
Ensemble-CNN [24]	F1	0.77	0.68	0.9	0.93	0.92	0.85	0.75
	TPR(Recall)	1	0.95	0.95	0.99	0.98	0.87	0.93
	FPR	0.17	0.63	0.93	0.12	0.17	0.33	0.48
	Precision	0.65	0.54	0.87	0.88	0.87	0.83	0.64
Ensemble-LSTM [24]	F1	0.92	0.9	0.99	0.96	0.92	0.82	0.93
	TPR(Recall)	0.91	0.88	0.99	0.94	0.92	0.77	0.92
	FPR	0.06	0.07	0	0.02	0.10	0.16	0.06
	Precision	0.93	0.93	0.98	0.97	0.92	0.93	0.96
Low-LSTM [41]	F1	0.86	0.89	0.97	0.96	0.95	0.72	0.90
	TPR(Recall)	0.79	0.82	0.98	0.95	0.92	0.57	0.85
	FPR	0.04	0.18	0.01	0.02	0.02	0.10	0.03
	Precision	0.93	0.95	0.98	0.97	0.98	0.80	0.96
DEEPShield-CNN	F1	0.84	0.85	0.66	0.97	0.94	0.85	0.85
	TPR(Recall)	0.99	0.92	0.87	0.99	0.96	0.8	0.95
	FPR	0.32	0.31	0.1	0.04	0.09	0.12	0.3
	Precision	0.78	0.76	0.97	0.94	0.92	0.89	0.78
DEEPShield-LSTM	F1	0.92	0.88	0.99	0.97	0.92	0.69	0.92
	TPR(Recall)	0.9	0.88	0.99	0.98	0.93	0.54	0.89
	FPR	0.08	0.12	0	0.02	0.09	0.02	0.05
	Precision	0.95	0.87	0.99	0.97	0.93	0.92	0.93
DEEPShield-Ensemble	F1	0.93	0.91	0.99	0.95	0.95	0.89	0.95
	TPR(Recall)	0.91	0.9	0.99	0.93	0.94	0.88	0.92
	FPR	0.04	0.04	0	0.01	0.01	0.06	0.03
	Precision	0.96	0.91	0.99	0.99	0.98	0.91	0.98
DEEPShield-PE	F1	0.93	0.89	0.99	0.95	0.94	0.86	0.93
	TPR(Recall)	0.91	0.87	0.99	0.93	0.96	0.86	0.91
	FPR	0.04	0.03	0	0.02	0.07	0.06	0.03
	Precision	0.96	0.92	0.99	0.98	0.94	0.92	0.96

The testbed utilized the Raspberry Pi 4 with 4GB of RAM and a 1.5GHz 64-bit quad-core Cortex-A72 processor, testing on the HICT dataset. Each sample within the HICT dataset, characterized by dimensions $[f, m] = [200, 11]$, comprises features from up to 200 flows, representing data spread over a 5-second duration. On the HICT dataset, the DEEPShield-PE processes 2003 samples every second, with a throughput of 400.6 kfpS (kilo flows per second). Considering a dataset subset with 110,000 flows distributed across 550 samples, the DEEPShield-PE manages this subset in just 0.274 seconds. Conversely, the DEEPShield-Ensemble achieves a processing rate of 1918 samples per second, resulting in a throughput of 383.6 kfpS. This model processes the subset in about 0.286 seconds. While, Ensemble-LSTM [24] operates at a rate of 1348 samples per second, corresponding to a throughput of 269.6 kfpS. This model takes 0.408 seconds for the sample subset. In comparison, the DEEPShield-PE emerges as the top performer in processing rate. It not only handles the largest number of samples per second but also processes the given dataset subset in less time.

In terms of memory requirements, Figure 7 shows that the DEEPShield-PE model can process about 2003 samples per second when using a batch size of 1024 samples. The memory requirement per sample is 17,600 bytes. This can be broken down as: $200 \times 11 = 2,200$ floating point values, where each value occupies 8 bytes. When more detailed features are

TABLE 8. Comparison of execution time, memory, and model size.

System	Execution Time (S)	Memory Usage (MB)	Model Size (bytes)
Ensemble-LSTM [24]	2.01	1273.16	410152
DEEPShield-Ensemble	1.64	982.02	347584
DEEPShield-PE	1.43	880.79	137416

taken into account, including the maximum, average, minimum, and standard deviation, the total number of floating point values increases to $200 \times 42 = 8,400$. Consequently, the memory footprint for these features amounts to 67,200 bytes. For the 11-feature scenario, continuous processing over 100 seconds necessitates approximately 3.52 GB of memory. In contrast, when considering 42 features, the memory demand surges to around 13.46 GB. This demand exceeds typical memory sizes on edge devices, such as Raspberry Pi 4 with 4 GB. Given the Raspberry Pi's 4 GB RAM limitation, while keeping the same sample-to-memory ratio unchanged, it is estimated that the Raspberry Pi can efficiently manage the entire set of samples within a 100-second window. However, to optimize performance on a Raspberry Pi, it is necessary to adjust the processing flow. Given the Raspberry Pi's inherent constraints, adjusting the flow as a hyperparameter offers a way to higher efficiency.

Table 8 shows the memory usage and model size on the testing hardware. In experiments, the DEEPShield system is tested on a Raspberry Pi using offline HICT test data with 28000 samples. The DEEPShield-PE model offers a more memory-efficient solution, showing a memory usage reduction of up to 30.8% compared to the Ensemble-LSTM [24] and 10.30% compared to the DEEPShield-Ensemble. Model storage size is an essential factor for devices with limited resources. The DEEPShield-PE, being over 66% smaller than Ensemble-LSTM, not only conserves memory but also potentially provides fast load times and reduces memory consumption during the model loading process. In comparison to the DEEPShield-Ensemble, the model size of the DEEPShield-PE is 60% smaller.

The optimization module involves unit pruning, which requires a balance between the model size (contain weights), accuracy, and prediction time. The Pareto frontier in Figure 8 is a graphical representation to visualize this balance. Each point on the scatter plot corresponds to a specific configuration achieved through the Pareto front method [54]. The highlighted point on the scatter point represents an optimal equilibrium tailored to meet the defined optimization criteria. An accuracy threshold is set at above 93%, and weighting coefficients α, β, and γ are assigned a value of 0.33. Using the HICT dataset as a reference, the optimal sparsity value, k,

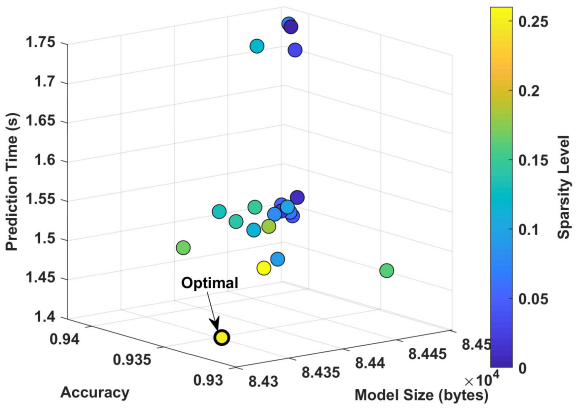

FIGURE 8. Model size, accuracy and prediction time on various sparsity.

for pruning is determined to be 25%. This suggests that by removing 25% of the units, the efficiency of the ensemble model is improved. At 25% sparsity, the resulting model exhibits a size of 84,300 bytes (weights) and a prediction time of 1.43 seconds.

G. CASE STUDY: DEEPShield SYSTEM EVALUATION UNDER VARYING NETWORK TRAFFIC

In this case study, the performance of the DEEPShield system is evaluated in simulated real-world operational network environments. A network load testing tool namely, Locust [55] is used to generates concurrent inference requests and random network traffic to emulate real-world scenarios. This approach provides valuable insights into the DEEPShield system adaptability and performance across different network scenarios, highlighting its potential integration into operational networks in the future.

To evaluate the efficiency and scalability of the DEEPShield system, concurrent inference requests are sent by Locust [55] to the DEEPShield system deployed on the edge server (implemented in a Raspberry Pi 4). Each concurrent inference request experiment is repeated 10 times for better statistical accuracy. During each experiment, the system handles between 5 to 50 concurrent requests, simulating various traffic loads processed simultaneously by the detection system. The total number of requests generated in each experiment is recorded, representing the overall load on the system throughout the duration of the test. A key aspect of this case study involves introducing variability in the data samples processed by the system during each inference request. To better reflect the fluctuating nature of live network traffic, instead of a fixed sample size, the system randomly selects between 100 to 300 samples for each inference. Performance metrics such as average response time, Requests-Per-Second (RPS), memory usage, F1 score, and the number of samples processed are measured during each experiment. The results are analyzed to understand the

TABLE 9. Simulation parameters and purposes.

Parameter	Value	Purpose
Concurrent Requests	5-50	Scalability and load handling
Sample Size	100-300 (Variable)	Mimicking real-world traffic fluctuations
Ramp-Up Period	60s	Assessing system adaptation to increasing load
Test Duration	300s	Ensuring robust performance metrics
Total Simulations	10	Reliability and consistency of test results
Deployment Hardware	Raspberry Pi 4 (4GB)	Evaluation of model performance on specific hardware

TABLE 10. Model evaluation metrics under varying load scenario.

Request	Metrics	5	10	15	20	25	30	35	40	45	50
Ensemble-LSTM [24]	Total Requests	197	385	581	754	926	1062	1178	1106	1414	1153
	Number of Samples	403238	402504	550414	615895	615968	509397	372511	311137	321871	286542
	F1 Score	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.92	0.93	0.93
	Latency (S)	0.29	0.31	0.36	0.48	0.65	0.97	1.39	3.27	1.89	5.48
	Memory Consumption (MB)	1034	1317	1846	2332	2908	2457	3139	3234	3156	3348
DEEPShield-PE	Total Requests	196	391	584	772	955	1137	1327	1437	1616	1157
	Number of Samples	359230	643401	809769	1150839	1348846	1433547	1490050	1527983	1391290	745920
	F1 Score	0.93	0.93	0.93	0.93	0.93	0.93	0.92	0.93	0.92	0.93
	Latency (S)	0.21	0.21	0.25	0.26	0.32	0.34	0.42	0.81	0.72	1.3
	Memory Consumption (MB)	882	1236	1637	1992	2224	2504	2677	2739	3155	3231

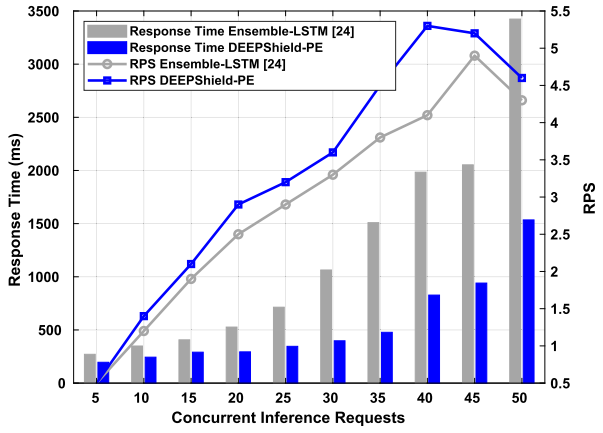

FIGURE 9. Model performance under varying load conditions on raspberry Pi-4.

system's scalability, communication latency, and resource utilization within the network environment. The simulation environment parameters are summarized in Table 9.

Figure 9, the response time and RPS metrics for Ensemble-LSTM [24] and the DEEPShield-PE models are shown under different load scenarios. The DEEPShield-PE model shows lower response times at lower request counts compared to Ensemble-LSTM model, which experiences significant increase in response times as the concurrent inference request grows. As compared to Ensemble-LSTM [24], the DEEPShield-PE model takes 52% to 68% less response time as the number of inference requests increases. Additionally, in terms of RPS, both models can manage higher concurrent requests as the load increases, but the DEEPShield-PE model maintains a faster rate compared to Ensemble-LSTM model.

Table 10 presents various meausred parameters, including total requests, number of samples processed, F1 score, memory consumption, and end-to-end latency. Both models consistently achieve a high F1 score of 0.93 across most levels of concurrent requests, indicating their reliability in correctly identifying true positives and negatives. As the number of concurrent requests increases, there is a noticeable trend of higher memory consumption. This outcome is expected, given the processing of larger volumes of data. At 35 concurrent requests, the DEEPShield-Ensemble model utilized 2677 MB, approximately 14.7% less memory compared to 3139 MB used by the Ensemble-LSTM model. This indicates high memory efficiency in the DEEPShield-Ensemble model, due to optimizations enabling more compact data processing. End-to-end latency, measuring the total time from request submission to completion, also increases as the number of requests grew. However, the DEEPShield-Ensemble model outperforms the Ensemble-LSTM model in this aspect.

H. DISCUSSION

The DEEPShield system employs novel methods in both network data preprocessing and post-training optimization tailored for edge deployment. This ensures high detection accuracy with minimal computational resources and processing time.

Using multiple datasets poses challenges due to differences in attack patterns and application layer protocols. To overcome this challenge, the DEEPShield system utilizes a novel parser designed to handle diverse raw network data. The DEEPShield parser uses Zeek to extract prominent features from this data for DDoS detection. Variations in network environments and potential data noise pose challenges in ensuring the quality and reliability of processed data for model training. The DEEPShield parser addresses this challenge by converting raw features into a balanced HDF5 format, preserving relevant flow features. Processing large volumes of network data into a format suitable for ML models requires significant computational resources. Ensuring scalability of the preprocessing pipeline for handling large datasets is crucial, especially for real-time DDoS detection. The parser addresses this by processing data in chunks and in parallel to reduce computation time. However, working with raw network data raises privacy and security concerns, as these data may contain sensitive information. To mitigate these risks, the DEEPShield parser operates on a separate network from the research lab's main network, ensuring compliance with privacy regulations and preventing exposure of sensitive information.

The FPR rate is slightly higher for low-volume attacks compared to high-volume attacks in the DEEPshield system. Analyzing the variation in p-values between high and low-volume attacks indicates how certain network features are effective in identifying attack intensity and characteristics. Lower p-values in high-volume attacks show their significant impact on network traffic patterns, making them easier to detect using the defined features. Conversely, higher p-values in low-volume attacks suggest a more subtle deviation from normal traffic, necessitating more refined analysis techniques for accurate detection. This statistical understanding forms the basis of DEEPShield system's adaptable detection mechanisms.

During the DEEPShield-PE model optimization, trade-offs were carefully managed to balance efficiency with performance. A significant reduction in the model's size was achieved through the pruning process, enhancing its execution time while maintaining accuracy above 93% in DDoS attack detection. This optimization makes the DEEPShield-PE model particularly suitable for IoT contexts and edge computing environments with limited resources.

The consistent F1 scores show robust model performance despite increasing loads, indicating efficiency in real-world environments with varying network traffic. Analysis of memory consumption and end-to-end latency shows lower resource usage and faster processing with DEEPShield-PE, suggesting minimal impact on network operations. Testing on a Raspberry Pi, mimicking edge computing, alongside observed metrics, supports the DEEPShield system's suitability for live network deployment. However, it's important to note that performance begins to decrease on a 4GB Raspberry Pi upon reaching 40 concurrent requests with 1,527,983 samples. The observation underscores a critical threshold for resource consumption on constrained devices, emphasizing the importance of considering device capabilities in edge computing deployments.

V. CONCLUSION

In this paper, a novel Deep Ensemble learning with Pruning (DEEPShield) system was proposed to address the challenges of efficient detection of both high- and low-volume DDoS attacks in resource-constrained environments. The integration of CNN and LSTM architectures in the ensemble model of the DEEPShield system resulted in high detection accuracy with minimal processing time and computational resources. To be acceptable in resource-constrained edge computing scenarios, the DEEPShield system integrated a unit pruning optimization technique that refines the model without compromising its accuracy. This paper also validated the HL-IoT dataset that considered both high- and low-volume DDoS attacks. By training and validating the DEEPShield system on this dataset, as well as other popular datasets such as the HICT, ToN-IoT, CICIDS-17, and ISCX-12, the system showed its adaptability across varied scenarios and attack volumes. In comparison to existing state-of-the-art models, the DEEPShield system showed a 90% accuracy, operated up to 40% faster, and required 30% less memory. Additionally, the DEEPShield system is evaluated under varying load scenarios to assess its scalability and performance in real-world environment. Future work aims to refine the DEEPShield system by including pruning during a training phase. The objective will be to develop an optimal model that reduces prediction time and memory usage while preserving accuracy. Additionally, optimization of the network parser will be explored to enhance system efficiency and robustness. Furthermore, the future direction will also involve the integration of real-time operational network data to enhance the system's adaptability to dynamic network environments.

REFERENCES

[1] J. Bhayo, R. Jafaq, A. Ahmed, S. Hameed, and S. A. Shah, "A time-efficient approach toward DDoS attack detection in IoT network using SDN," *IEEE Internet Things J.*, vol. 9, no. 5, pp. 3612–3630, Mar. 2022, doi: 10.1109/JIOT.2021.3098029.

[2] M. F. Saiyed and I. Al-Anbagi, "A genetic algorithm- and t-test-based system for DDoS attack detection in IoT networks," *IEEE Access*, vol. 12, pp. 25623–25641, 2024, doi: 10.1109/ACCESS.2024.3367357.

[3] S. Siddiqui et al., "Toward software-defined networking-based IoT frameworks: A systematic literature review, taxonomy, open challenges and prospects," *IEEE Access*, vol. 10, pp. 70850–70901, 2022, doi: 10.1109/ACCESS.2022.3188311.

[4] M. Nandanwar. (2022). *Gafgyt Backdoor*. Sectrio. [Online]. Available: https://sectrio.com/wp-contentuploads/2022/06/Gafgyt-Backdoor.pdf

[5] K. on Security. (2016). *DDoS on Dyn Impacts Twitter, Spotify, Reddit*. Accessed: Oct. 31, 2019. [Online]. Available: https://krebsonsecurity.com/2016/10/ddos-on-dyn-impacts-twitter-spotify-reddit

[6] Radware. (2017). *Reaper Botnet*. [Online]. Available: https://www.radware.com/security/ddos-threats-attacks/threat-advisories-attack-reports/reaper-botnet/

[7] Brickerbot Permanent Denial-of-Service Attack (Update a). *CISA (Cybersecurity and Infrastructure Security Agency)*. [Online]. Available: https://www.cisa.gov/news-events/ics-alerts/ics-alert-17-102-01a

[8] M. M. Salim, S. Rathore, and J. H. Park, "Distributed denial of service attacks and its defenses in IoT: A survey," *J. Supercomput.*, vol. 76, no. 7, pp. 5320–5363, Jul. 2020, doi: 10.1007/s11227-019-02945-z.

[9] R. Vishwakarma and A. K. Jain, "A survey of DDoS attacking techniques and defence mechanisms in the IoT network," *Telecommun. Syst.*, vol. 73, no. 1, pp. 3–25, Jan. 2020, doi: 10.1007/s11235-019-00599-z.

[10] H. H. Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling," *Comput. Netw.*, vol. 121, pp. 25–36, Jul. 2017, doi: 10.1016/j.comnet.2017.03.018.

[11] W. Zhijun, X. Qing, W. Jingjie, Y. Meng, and L. Liang, "Low-rate DDoS attack detection based on factorization machine in software defined network," *IEEE Access*, vol. 8, pp. 17404–17418, 2020, doi: 10.1109/ACCESS.2020.2967478.

[12] J. Li, M. Liu, Z. Xue, X. Fan, and X. He, "RTVD: A real-time volumetric detection scheme for DDoS in the Internet of Things," *IEEE Access*, vol. 8, pp. 36191–36201, 2020, doi: 10.1109/ACCESS.2020.2974293.

[13] N. Mishra and S. Pandya, "Internet of Things applications, security challenges, attacks, intrusion detection, and future visions: A systematic review," *IEEE Access*, vol. 9, pp. 59353–59377, 2021, doi: 10.1109/ACCESS.2021.3073408.

[14] T. E. Ali, Y.-W. Chong, and S. Manickam, "Comparison of ML/DL approaches for detecting DDoS attacks in SDN," *Appl. Sci.*, vol. 13, no. 5, p. 3033, Feb. 2023, doi: 10.3390/app13053033.

[15] M. A. Al-Shareeda, S. Manickam, and M. A. Saare, "DDoS attacks detection using machine learning and deep learning techniques: Analysis and comparison," *Bull. Electr. Eng. Informat.*, vol. 12, no. 2, pp. 930–939, Apr. 2023, doi: 10.11591/eei.v12i2.4466.

[16] H. Thanh and T. Lang, "Use the ensemble methods when detecting DoS attacks in network intrusion detection systems," *EAI Endorsed Trans. Context-Aware Syst. Appl.*, vol. 6, no. 19, Nov. 2019, Art. no. 163484, doi: 10.4108/eai.29-11-2019.163484.

[17] K. Wu, Z. Chen, and W. Li, "A novel intrusion detection model for a massive network using convolutional neural networks," *IEEE Access*, vol. 6, pp. 50850–50859, 2018, doi: 10.1109/access.2018.2868993.

[18] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del-Rincón, and D. Siracusa, "LUCID: A practical, lightweight deep learning solution for DDoS attack detection," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 2, pp. 876–889, Jun. 2020, doi: 10.1109/TNSM.2020.2971776.

[19] B. Alotaibi and M. Alotaibi, "A stacked deep learning approach for IoT cyberattack detection," *J. Sensors*, vol. 2020, pp. 1–10, Sep. 2020, doi: 10.1155/2020/8828591.

[20] M. Zeeshan et al., "Protocol-based deep intrusion detection for DoS and DDoS attacks using UNSW-NB15 and bot-IoT data-sets," *IEEE Access*, vol. 10, pp. 2269–2283, 2022, doi: 10.1109/ACCESS.2021.3137201.

[21] F. Alasmary, S. Alraddadi, S. Al-Ahmadi, and J. Al-Muhtadi, "Shield-RNN: A distributed flow-based DDoS detection solution for IoT using sequence majority voting," *IEEE Access*, vol. 10, pp. 88263–88275, 2022, doi: 10.1109/ACCESS.2022.3200477.

[22] K. Saurabh, T. Kumar, U. Singh, O. P. Vyas, and R. Khondoker, "NFDLM: A lightweight network flow based deep learning model for DDoS attack detection in IoT domains," in *Proc. IEEE World AI IoT Congr. (AIIoT)*, Jun. 2022, pp. 736–742, doi: 10.1109/AIIoT54504.2022.9817297.

[23] A. Zainudin, L. A. C. Ahakonye, R. Akter, D.-S. Kim, and J.-M. Lee, "An efficient hybrid-DNN for DDoS detection and classification in software-defined IIoT networks," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 8491–8504, May 2023, doi: 10.1109/JIOT.2022.3196942.

[24] S. Haider et al., "A deep CNN ensemble framework for efficient DDoS attack detection in software defined networks," *IEEE Access*, vol. 8, pp. 53972–53983, 2020, doi: 10.1109/ACCESS.2020.2976908.

[25] S.-K. Yeom et al., "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognit.*, vol. 115, Jul. 2021, Art. no. 107899, doi: 10.1016/j.patcog.2021.107899.

[26] N. Moustafa, M. Keshk, K.-K.-R. Choo, T. Lynar, S. Camtepe, and M. Whitty, "DAD: A distributed anomaly detection system using ensemble one-class statistical learning in edge networks," *Future Gener. Comput. Syst.*, vol. 118, pp. 240–251, May 2021, doi: 10.1016/j.future.2021.01.011.

[27] M. Odiathevar, W. K. G. Seah, M. Frean, and A. Valera, "An online offline framework for anomaly scoring and detecting new traffic in network streams," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 11, pp. 5166–5181, Nov. 2022, doi: 10.1109/TKDE.2021.3050400.

[28] M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "CorrAUC: A malicious bot-IoT traffic detection method in IoT network using machine-learning techniques," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3242–3254, Mar. 2021, doi: 10.1109/JIOT.2020.3002255.

[29] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma, "IoT-KEEPER: Detecting malicious IoT network activity using online traffic analysis at the edge," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 45–59, Mar. 2020, doi: 10.1109/TNSM.2020.2966951.

[30] M. M. N. Aboelwafa, K. G. Seddik, M. H. Eldefrawy, Y. Gadallah, and M. Gidlund, "A machine-learning-based technique for false data injection attacks detection in industrial IoT," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8462–8471, Sep. 2020, doi: 10.1109/JIOT.2020.2991693.

[31] J. Bhayo, S. A. Shah, S. Hameed, A. Ahmed, J. Nasir, and D. Draheim, "Towards a machine learning-based framework for DDOS attack detection in software-defined IoT (SD-IoT) networks," *Eng. Appl. Artif. Intell.*, vol. 123, Aug. 2023, Art. no. 106432, doi: 10.1016/j.engappai.2023.106432.

[32] N. Ravi and S. M. Shalinie, "Semisupervised-learning-based security to detect and mitigate intrusions in IoT network," *IEEE Internet Things J.*, vol. 7, no. 11, pp. 11041–11052, Nov. 2020, doi: 10.1109/JIOT.2020.2993410.

[33] A. Koay, A. Chen, I. Welch, and W. K. Seah, "A new multi classifier system using entropy-based features in DDoS attack detection," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2018, pp. 162–167, doi: 10.1109/ICOIN.2018.8343104.

[34] E. Gyamfi and A. D. Jurcut, "Novel online network intrusion detection system for industrial IoT based on OI-SVDD and AS-ELM," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 3827–3839, Mar. 2023, doi: 10.1109/JIOT.2022.3172393.

[35] L. Nie et al., "Intrusion detection for secure social Internet of Things based on collaborative edge computing: A generative adversarial network-based approach," *IEEE Trans. Computat. Social Syst.*, vol. 9, no. 1, pp. 134–145, Feb. 2022, doi: 10.1109/TCSS.2021.3063538.

[36] Y. Jia, F. Zhong, A. Alrawais, B. Gong, and X. Cheng, "FlowGuard: An intelligent edge defense mechanism against IoT DDoS attacks," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9552–9562, Oct. 2020, doi: 10.1109/JIOT.2020.2972472.

[37] T. T. Huong et al., "LocKedge: Low-complexity cyberattack detection in IoT edge computing," *IEEE Access*, vol. 9, pp. 29696–29710, 2021, doi: 10.1109/ACCESS.2021.3051786.

[38] Z. Tian, C. Luo, J. Qiu, X. Du, and M. Guizani, "A distributed deep learning system for web attack detection on edge devices," *IEEE Trans. Ind. Informat.*, vol. 16, no. 3, pp. 1963–1971, Mar. 2019, doi: 10.1109/TII.2019.2938778.

[39] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 1, pp. 41–50, Feb. 2018, doi: 10.1109/TETCI.2017.2772792.

[40] N. Moustafa, B. Turnbull, and K.-K. R. Choo, "An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4815–4830, Jun. 2018, doi: 10.1109/JIOT.2018.2871719.

[41] A. A. Alashhab, M. S. M. Zahid, A. Muneer, and M. Abdukkahi, "Low-rate DDoS attack detection using deep learning for SDN-enabled IoT networks," *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 11, pp. 1–7, 2022, doi: 10.14569/ijacsa.2022.0131141.

[42] X. Gao, C. Shan, C. Hu, Z. Niu, and Z. Liu, "An adaptive ensemble machine learning model for intrusion detection," *IEEE Access*, vol. 7, pp. 82512–82521, 2019, doi: 10.1109/ACCESS.2019.2923640.

[43] O. Chakir et al., "An empirical assessment of ensemble methods and traditional machine learning techniques for web-based attack detection in Industry 5.0," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 35, no. 3, pp. 103–119, Mar. 2023, doi: 10.1016/j.jksuci.2023.02.009.

[44] N. Niknami and J. Wu, "Entropy-KL-ML: Enhancing the entropy-KL-based anomaly detection on software-defined networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 6, pp. 4458–4467, Nov. 2022, doi: 10.1109/TNSE.2021.3053080.

[45] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, nos. 23–24, pp. 2435–2463, Dec. 1999. [Online]. Available: http://www.icir.org/vern/papers/bro-CN99.pdf
[46] M. E. Ahmed, S. Ullah, and H. Kim, "Statistical application fingerprinting for DDoS attack mitigation," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1471–1484, Jun. 2019, doi: 10.1109/TIFS.2018.2879616.
[47] M. F. Saiyed and I. Al-Anbagi, "Flow and unified information-based DDoS attack detection system for multi-topology IoT networks," *Internet Things*, vol. 24, Dec. 2023, Art. no. 100976, doi: 10.1016/j.iot.2023.100976.
[48] K. Doshi, Y. Yilmaz, and S. Uludag, "Timely detection and mitigation of stealthy DDoS attacks via IoT networks," *IEEE Trans. Depend. Secure Comput.*, vol. 18, no. 5, pp. 2164–2176, Sep./Oct. 2021, doi: 10.1109/TDSC.2021.3049942.
[49] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, Sep. 2022, doi: 10.1016/j.neucom.2022.06.111.
[50] *hping wiki*. Accessed: Jul. 25, 2023. [Online]. Available: http://wiki.hping.org/download/
[51] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. ICISSp*, vol. 1, 2018, pp. 108–116, doi: 10.5220/0006639801080116.
[52] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, 2012, doi: 10.1016/j.cose.2011.12.012.
[53] N. Moustafa. (2020). *ToN-IoT Dataset*. [Online]. Available: https://cloudstor.aarnet.edu.au/plus/s/ds5zW91vdgjEj9i
[54] S. Petchrompo, D. W. Coit, A. Brintrup, A. Wannakrairot, and A. K. Parlikad, "A review of Pareto pruning methods for multi-objective optimization," *Comput. Ind. Eng.*, vol. 167, May 2022, Art. no. 108022, doi: 10.1016/j.cie.2022.108022.
[55] Locust Contributors. (2024). *Locust Documentation*. Accessed: Feb. 23, 2024. [Online]. Available: https://docs.locust.io/en/stable/index.html

MAKHDUMA F. SAIYED (Student Member, IEEE) received the Bachelor of Engineering degree from Gujarat University, India, and the Master of Engineering degree in computer engineering from Gujarat Technological University, India. She is currently pursuing the Ph.D. degree with the University of Regina, Regina, Canada. She has ten years of teaching experience and has published more than nine publications. Her research interests include network security, the Internet of Things, machine learning, and deep neural networks.

IRFAN AL-ANBAGI (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Ottawa in October 2013. From 2013 to 2015, he was a Post-Doctoral Fellow and the Product Development Manager of the "SecCharge" Project, University of Ottawa. He is currently an Associate Professor with the Faculty of Engineering and Applied Science, University of Regina. His research interests include security and reliability in networked systems, including quality of service (QoS), reliability, optimization, and cybersecurity, specifically modeling of failure propagation in networked cyber-physical systems; security and QoS in cloud-edge architectures; implementation of ambient intelligence in the Internet of Things (IoT) systems; and wireless sensor networks and their implementation in critical applications. He is a Registered Professional Engineer with the Association of Professional Engineers and Geoscientists of Saskatchewan (APEGS) and Professional Engineers Ontario (PEO).