Intelligent Automation & Soft Computing

Tech Science Press

DOI: 10.32604/iasc.2023.039937
Article

Machine Learning-Based Efficient Discovery of Software Vulnerability for Internet of Things

So-Eun Jeon, Sun-Jin Lee and Il-Gu Lee*

Department of Future Convergence Technology Engineering, Sungshin Women's University, Seoul, 02844, Korea
*Corresponding Author: Il-Gu Lee. Email: iglee@sungshin.ac.kr
Received: 25 February 2023; Accepted: 18 May 2023; Published: 23 June 2023

Abstract: With the development of the 5th generation of mobile communication (5G) networks and artificial intelligence (AI) technologies, the use of the Internet of Things (IoT) has expanded throughout industry. Although IoT networks have improved industrial productivity and convenience, they are highly dependent on nonstandard protocol stacks and open-source-based, poorly validated software, resulting in several security vulnerabilities. However, conventional AI-based software vulnerability discovery technologies cannot be applied to IoT because they require excessive memory and computing power. This study developed a technique for optimizing training data size to detect software vulnerabilities rapidly while maintaining learning accuracy. Experimental results using a software vulnerability classification dataset showed that different optimal data sizes did not affect the learning performance of the learning models. Moreover, the minimal data size required to train a model without performance degradation could be determined in advance. For example, the random forest model saved 85.18% of memory and improved latency by 97.82% while maintaining a learning accuracy similar to that achieved when using 100% of data, despite using only 1%.

Keywords: Lightweight devices; machine learning; deep learning; software vulnerability detection; common weakness enumeration

1 Introduction

The Internet of Things (IoT) is growing with hyperconvergence, hyperconnection, and hyperintelligence, where everything is connected to the network [1,2]. Therefore, the number of IoT devices owned by individuals is increasing. Statista predicted that the number of IoT devices worldwide would increase from 9.7 billion in 2020 to more than 29 billion in 2030 [3]. However, IoT heavily relies on nonstandard protocol stacks and poorly verified open-source software. Such vulnerable software have a significant adverse effect on the entire industry and on individuals. If the code containing the vulnerability is copied and used in another environment, it has the same security weakness and becomes a target for attackers. According to the IoT threat report published by Palo Alto Networks, IoT devices can exploit infections, malware, and user information leakages. IoT target exploits typically include

network scans, command injections, and buffer overflows [4]. In 2016, Mirai malware attacked the domain name system service and infiltrated home appliances [5]. The Mirai botnet targets and infects devices with vulnerable administrator accounts and uses the infected devices for distributed denial of service (DDoS) attacks [6]. In 2022, the Mozi botnet infected 11,700 domestic and foreign IoT devices and used them to hack public institutions and disseminate cryptocurrency mining [7]. Therefore, IoT security threats are gradually increasing and developing into various attack methods.

As advanced attacks on IoT devices have evolved, studies on IoT vulnerability analysis, attack detection, and response techniques have been actively conducted. A typical approach involves software vulnerability and security attack detection using artificial intelligence (AI). AI-based software vulnerability detection can increase the detection rate by being trained on various patterns of attacks occurring in node and cluster functions with vulnerabilities. The increase in the number of IoT devices and their variety has made the implementation of machine learning for attack detection in IoT environments essential. However, because an IoT device has a small amount of power, energy is consumed in training the model, and a large amount of data cannot be used because of its limited memory. In addition, the computing power is not sufficiently large to detect anomalies accurately within a limited time. Moreover, the overall network communication and cloud environment must be considered [8]. In other words, machine learning in IoT devices has become essential. However, due to environmental restrictions on IoT devices, there are still limitations in actively introducing machine learning.

In the past, software vulnerabilities were analyzed and detected on high-performance server systems. However, lightweight IoT devices have limited computing power, memory, and battery capacity, making it difficult to detect and respond to software vulnerabilities like high-performance servers. Recently, a variety of research has been conducted to implement machine learning in IoT devices. However, in previous studies, the resource-constrained environment of IoT was insufficiently considered, and the realistic attack performance environment was not considered. To solve this problem, this study addresses ways to efficiently detect software vulnerabilities by optimizing the training data size to detect software vulnerabilities rapidly while maintaining learning accuracy. The data sampling ratio in the dataset used to train the AI model in an IoT environment was adjusted to find the optimal point between size and performance. This study focuses on detecting practical machine learning-based software vulnerability and evaluates performance from the perspective of accuracy, memory usage, and latency, which should be considered for practical IoT environments. Consequently, we found an environment that alleviates trade-offs in terms of accuracy, memory usage, and latency for IoT devices.

The main contributions of this study are as follows:

- First, the optimal training data size to improve a model's performance is identified.
- Second, a machine learning-based software vulnerability detection method that can be used in lightweight IoT devices is proposed. In addition, the proposed method was verified using the vulnerability detection in source code (VDISC) dataset, which contains essential IoT vulnerability data such as buffer overflow and null pointer.

The remainder of this paper is organized as follows. Section 2 analyzes related research on attack detection methods for IoT devices. Section 3 proposes a method to efficiently detect machine learning-based software vulnerabilities in lightweight devices by optimizing memory usage and learning accuracy. Section 4 evaluates and analyzes performance. Finally, Section 5 concludes the paper.

2 Related Work

IoT attacks have been developed in various ways, from physical to man-in-the-middle attacks, false data injection, and botnet-based DDoS. Initially, software metrics based on complexity, code change, or software detection methods based on abstract syntax tree (AST) were studied. Since then, AI has been integrated into IoT devices to protect systems from advanced attacks. In particular, AI is the most typical technology for intrusion detection that analyzes traffic patterns and identifies attack behaviors [9,10]. Table 1 presents an analysis of previous studies on software vulnerability detection.

Table 1: Previous studies on software vulnerabilities detection

Previous studies	References	Methods and contribution	Limitation
Source code vulnerabilities	Li et al. [11]	• Code gadgets are vectorized, source code datasets are constructed on a per-code gadget basis, and performance is evaluated using the bidirectional long short-term memory (Bi-LSTM) model.	• Lack of consideration of various learning models • Lack of consideration of different vulnerability types. (Consider CWE-119, CWE-399 only)
	Bilgin et al. [12]	• Ways to automatically analyze source code using machine learning models are studied. • Through lexical analysis, parsing, and AST generation processes for source code, a deep learning model is configured to extract useful features when determining vulnerabilities.	• Different dataset environments leverage various learning models, resulting in poor reliability.
IoT vulnerabilities	Blinowsk et al. [13]	• Detection of new vulnerabilities in IoT devices using NVD datasets • Defined its classification class and evaluated its performance	• Unable to learn data from heterogeneous devices rapidly • Difficult to apply to lightweight IoT devices

(Continued)

Table 1: Continued

Previous studies	References	Methods and contribution	Limitation
	Niu et al. [14]	• A deep learning-based static pollution analysis method automatically finding IoT vulnerabilities is proposed • Use the CNN-Bi-LSTM model to evaluate the model's performance	• Excessive computing power is required
	Zolanvari et al. [15]	• Proposed machine learning-based intrusion detection methodology in an industrial Internet of Things (IIoT) environment • Evaluations were conducted using the backdoor, SQL injection, and command injection, which are attacks occurring primarily in an IoT environment.	• Difficult to apply to lightweight IoT devices
	Hasan et al. [16]	• Classify DDoS, data-type probing, malicious control, malicious operation, scan, spying, and incorrect setup using machine learning models	• Compared simple methods in an ideal environment
	Kumar et al. [17]	• Proposed IoT attack detection methodology for large-scale networks in scanning and infection stages	• Poor detection rate, and additional devices are required

Li et al. [11] studied source code vulnerability detection. In particular, the authors defined the source code as a code gadget. Code gadgets were vectorized; source code datasets were constructed on a per-code gadget basis, and performance was evaluated using the bidirectional long short-term memory (Bi-LSTM) model. Moreover, a lower non-detection rate than the existing vulnerability detection system was observed. However, this previous study did not compare the performance of various learning models and only evaluated the model performance for two vulnerability types, CWE 119 and CWE-399. Additionally, research has been conducted to extract and analyze the source code as an AST and differentiate between vulnerable and non-vulnerable codes by applying a deep learning model. Bilgin et al. [12] recognized that the existing vulnerability detection method requires the expertise and time of domain experts; thus, a method for automatically analyzing source codes

using a machine learning model was studied. In addition, through lexical analysis, syntax analysis, and AST generation of the source code, a deep learning model was configured to extract useful features when determining the presence of vulnerabilities. The model was evaluated using a CNN model. On an imbalanced dataset, the Micro-Average P-R curve, the average performance for all classification values, showed a value of 0.377 when the F1 value passed the 0.4 curve. This study contributed to analyzing the source code in the form of AST and evaluating the performance of imbalanced and balanced datasets. However, the different dataset environments leverage various learning models so the result was in poor reliability.

Blinowsk et al. [13], who studied IoT vulnerability detection, proposed a machine learning methodology for detecting novel vulnerabilities in IoT systems. The network vulnerability data (NVD) database, a representative common vulnerability database (CVE), was categorized into multiple classes (H, S, E, M, P, and A) according to the field. H targeted vulnerabilities related to home and small office home office devices, S targeted supervisory control and data acquisition and industrial systems, and E targeted enterprise industries. M targeted mobile and wearable devices, and P grouped personal computers (PCs) and laptops. Furthermore, A referred to other home appliances, such as printers and storage devices. Subsequently, the classification performance was evaluated using a support vector machine (SVM). Experiments show that when there are many data points, the precision is 70% to 80%, while when there are fewer data points, the precision is less than 50%. The method proposed by Blinowsk et al. [13] is meaningful because it attempts to find zero-day vulnerabilities that could occur through vulnerability data in IoT and other devices. However, rapidly training AI models on data from heterogenous devices in an IoT environment remains a challenge. This approach is unsuitable for lightweight IoT devices because one cannot expect further effects in terms of complexity when using the computationally complex SVM model. In addition, the method proposed by Blinowsk et al. showed poor detection performance compared to the time learned, with some groups achieving a precision of less than 50%.

Niu et al. [14] proposed a deep learning-based static pollution analysis method that automatically finds vulnerabilities in IoT software. The method uses difflib to obtain a diff file between the source and patched source codes. Subsequently, labels are entered according to the contamination selection principle. Then, a propagation path is predicted through static pollution propagation, and the path is converted into a symbolic representation. The symbolic representation is again encoded as a vector, and the data are trained using a deep learning model. This study used a trained convolutional neural network-Bi-LSTM (CNN-Bi-LSTM) model to identify two general types of IoT software vulnerabilities. The proposed model was evaluated using buffer overflow (CWE-119) and resource management vulnerability (CWE-399), and vulnerabilities were detected with accuracies of 97.32% and 97.21%, respectively. This study is meaningful because the proposed method traces the attack propagation path and learns data. However, it has the following limitation: a high-powered computing device is required for IoT vulnerability detection.

Zolanvari et al. [15] used machine learning in IIoT to evaluate the performance of an intrusion environment. In particular, the method was used to classify backdoor, structured query language (SQL) injection, and command injection attacks that can occur in an IIoT system by monitoring the water level and turbidity of the reservoir. The random forest (RF) model achieved a detection accuracy of approximately 99%. Furthermore, Zolanvari et al. [15] measured the detection rate in an actual IIoT environment with high accuracy. However, their method is unsuitable for lightweight IoT devices because a large amount of computing power and training time cannot be invested in general IoT equipment.

Hasan et al. [16] used machine learning models to predict IoT attacks and anomalies. First, the performances of the models were compared using logistic regression (LR), SVM, decision trees, RT, and artificial neural networks. Then, DDoS, data-type probing, malicious control, malicious operation, scan, spying, and wrong setup attacks were detected with accuracies of 98%–99%. This study contributed to the literature by evaluating the performance of the machine learning models used in IoT attacks. However, the study did not model an ideal environment, and a simple methodology was used. In addition, it is difficult to conclude that RF is suitable because it does not yield the best results for all vulnerability datasets.

Kumar et al. [17] proposed the early detection of IoT malware network activity (EDIMA) to detect large-scale IoT attacks during the scanning and infection stages. EDIMA classified traffic from edge devices with an accuracy of 88.8% for RF, 94.44% for k-nearest neighbors (KNN), and 77.78% for Gaussian naïve Bayes (Gaussian NB) algorithms. This study is meaningful because it developed a methodology that can detect an attack in the stage immediately before the attack. However, some traffic cannot be detected, and there are limitations on those additional nodes, such as PCs and edge devices, that must be used to detect attacks. Conventional studies have suggested various vulnerability detection methodologies for IoT devices. However, they have not explored whether IoT devices can perform training by themselves and whether high computing power is required to detect attacks..

3 Practical Machine Learning-Based Software Vulnerability Discovery Mechanism

This study derives the optimal data size for efficiently detecting software vulnerabilities in lightweight device environments that use resource-constrained IoT and mobile devices. Collecting sufficient learning data from lightweight devices with limited resources is challenging. Because the optimal learning model differs depending on the learning environment, this study analyzed the performance of machine learning and deep learning models according to the training data size. In particular, the optimal data size was determined without performance degradation of software vulnerability detection.

Fig. 1 shows a mechanism for determining the optimal minimum data size while maintaining accuracy. In general, when the training data are sufficient, the accuracy of machine learning models improves. In other words, even in this mechanism, the higher the data sampling rate, the higher the possibility of vulnerability detection because enough data are available for learning. However, resource usage, such as memory usage and latency, becomes inefficient. Because the accuracy and resource usage performance indicators are in a trade-off relationship, this study proposes a model to find and optimize the trade-off between accuracy and resource usage. The operation process of the proposed mechanism is described below.

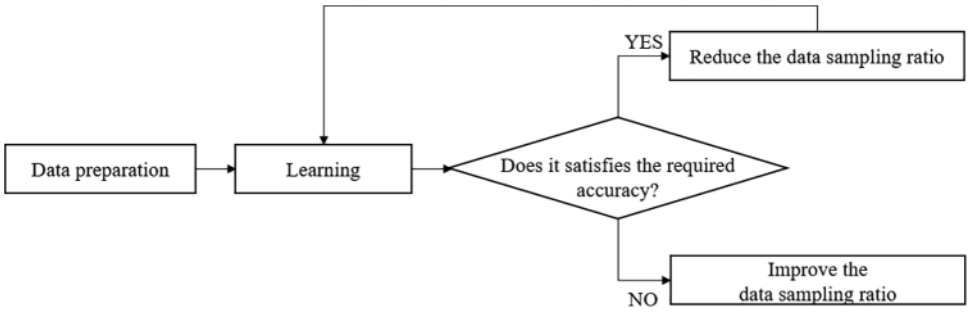

Figure 1: Flowchart of the data sampling ratio optimization mechanism

First, when a training dataset was prepared, the models were trained without sampling in the first round. If the required accuracy was satisfied, the data sampling ratio was decreased to save memory usage. However, if the required accuracy was not satisfied, the data sampling ratio was increased to minimize the performance degradation and is relearned. A back-off algorithm was applied to minimize performance degradation by adjusting the data sampling ratio to maintain sufficient accuracy. In other words, when the required accuracy was satisfied, the two performance indicators in a trade-off relationship can be optimized while adjusting the sampling ratio. The proposed algorithm can optimize the trade-off problem between data classification performance and the overhead due to memory occupancy. Moreover, the algorithm can provide real-time performance and update the framework through a back-off algorithm.

4 Performance Evaluation and Analysis

This section compares the performance of machine learning and deep learning models using the source code vulnerability dataset. In particular, the environment of the training data was adjusted, and actual demand metrics for evaluating the performance of lightweight devices used in an actual scenario was considered.

4.1 Experimental Environment

The VDISC dataset [18] is a collection of source code functions that can become potential vulnerabilities through static analysis. This dataset consisted of 127,418 vulnerable function codes. Table 2 lists the CWE-IDs and each vulnerability type included in the VDISC dataset.

Table 2: CWE-IDs in the VDISC dataset [19]

CWE-ID	Description
119	Improper restriction of operations within the bounds of a memory buffer
120	Buffer overflow
469	Use of pointer subtraction to determine size
476	NULL pointer dereference
Other (e.g., 20, and 457)	Improper input validation, and use of uninitialized variable

The VDISC dataset contains large amounts of functional code collected from several open-source projects. The vulnerability types, which are prone to attacks, are classified as CWE-119 (improper restriction of operations within the bounds of a memory buffer), CWE-120 (buffer overflow), CWE-469 (use of pointer subtraction to determine size), CWE-476 (NULL pointer dereference), and CWE-other. CWE-other is a label for other vulnerabilities, including CWE-20 and CWE-457. The frequency of occurrence of CWE-119, CWE-120, CWE-469, CWE-476, and CWE-other are 38.1%, 18.9%, 9.5%, 2.0%, and 31.4%, respectively [19]. Therefore, this experiment used a dataset of vulnerability types with a high occurrence rate. Vulnerable source code functions were preprocessed using word-level tokenization. The average performance for each vulnerability type was derived using a binary classification model that predicted each vulnerability type as true or false. Machine learning models (KNN, RF, and LR) and deep learning models (CNN and multi-layer perceptron (MLP)) were selected to evaluate the classification performance according to vulnerability type. KNN is an algorithm that classifies data according to the ratio of labels by referring to k pieces of data close to the data. RF is an algorithm that collects and classifies results from trees constructed in the training process, and

LR is an algorithm that classifies multi-labels using a linear combination of independent variables. In addition, CNN is a model used to classify data such as images and videos by imitating the human optic nerve, and MLP is a network model composed of a multi-layer neural network structure with a hidden layer added to a single-layer perceptron.

In addition, the performance was verified according to the learning environment of the lightweight device by differentiating the sampling ratio in the entire dataset. The data sampling environments are listed in Table 3.

Table 3: Data sampling environment

Data sampling ratio (%)	Number of datasets
0.05	63
0.1	127
1	1,274
50	63,709
100	127,418

The evaluation metrics considered in this study are accuracy, memory usage, and latency. Accuracy predicts the vulnerability type and is calculated using Eq. (1).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

where true positive (TP) is the case where the model predicts true for a true condition. True negative (TN) is when the model has a false outcome for a false condition. A false positive (FP) is when the model outputs true for a false condition. A false negative (FN) indicates that the model has a true output for a false condition. Memory usage refers to the amount of memory used by a CPU during the learning process. Memory is utilized by the tracemalloc module. Latency is the amount of time required to learn and evaluate each model.

4.2 Evaluation Results and Analysis

This section analyzes the performance evaluation results for each machine learning model according to different data sampling ratios. Fig. 2 shows the performance results in terms of accuracy according to the data sampling ratio.

As shown in Fig. 2, the accuracy, memory usage, and latency increased as the data sampling ratio increased, implying that software vulnerability can be more accurately detected for a large training data size. However, this approach is inefficient considering memory usage and latency in lightweight devices with limited memory and computing power.

When the data sampling ratio was low, the accuracy was high in the order of RF, LR, CNN, KNN, and MLP. However, in an environment with a large training data size, the accuracy was high in the order of CNN, RF, KNN, LR, and MLP. The performance of all models converged in the range of approximately 98%. This result implies that the training data size must be maintained above a specific level to utilize a deep learning model because the detection performance of the deep learning model degrades in the absence of training data [20]. In addition, the performance of the RF model, which is a supervised machine learning model, was generally good, regardless of the training data size. On this

dataset, the benefit of RF, which efficiently generalizes the prediction model and does not easily cause overfitting, can be observed.

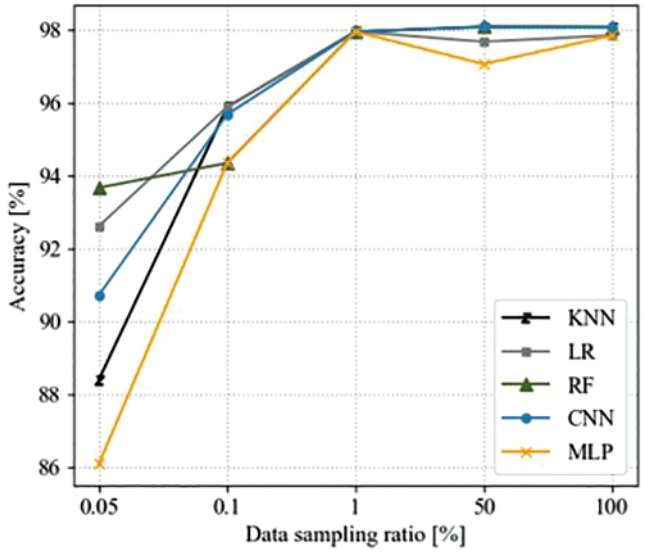

Figure 2: Model performance in terms of accuracy and data sampling rate

Fig. 3 shows the performance results in terms of memory usage according to the data sampling ratio.

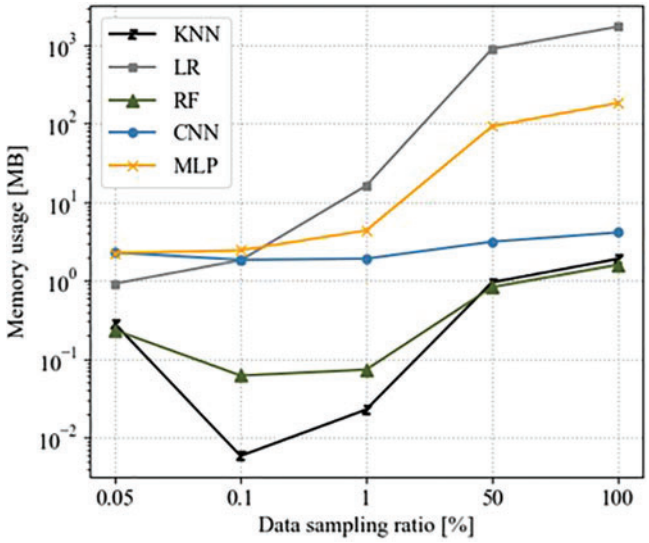

Figure 3: Model performance in terms of memory usage and data sampling ratio

Regarding memory usage, the efficiency of the algorithms was in the following order: KNN, RF, CNN, MLP, and LR. Memory usage was highest in the order of CNN, MLP, LR, KNN, and RF in environments with small training data sizes. Memory usage was highest in the order of LR, MLP, CNN, KNN, and RF in environments with large training data sizes. In particular, when LR and MLP sampled 0.05% of the training data compared with the total training data, the memory usage decreased

by 99.95% and 98.77%, respectively. Thus, verifying the inefficiency of LR and MLP is possible for a large training data size. In addition, KNN, a relatively simple model, had the least memory usage in an environment with a small dataset and was less efficient than RF in the non-sampled dataset environment, which is because the computational cost increases rapidly with the data size [21]. Thus, the results indicate that RF and LR had the most efficient memory usage, even in an environment with a small training data size.

Fig. 4 shows the performance results in terms of latency according to the data sampling ratio.

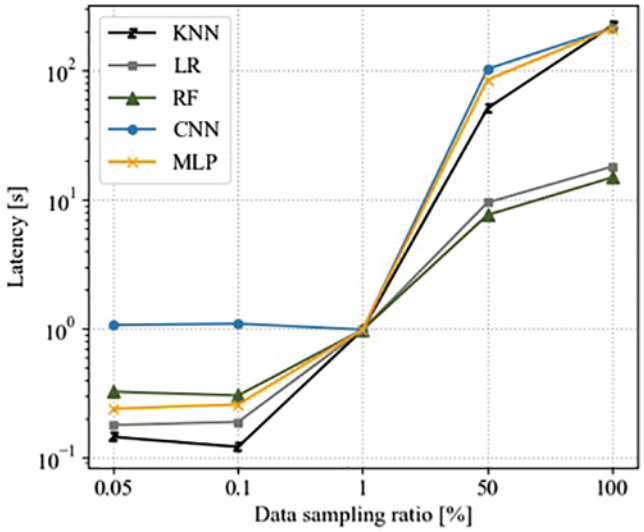

Figure 4: Model performance in terms of latency and the data sampling ratio

In terms of latency, performance deteriorated rapidly when the data sampling ratio exceeded 1%. In an environment with a small training data size, the latency was high in the following order: CNN, RF, MLP, LR, and KNN. However, in an environment with a large training data size, latency was high in the order of CNN, MLP, KNN, LR, and RF. In terms of latency, the performance degradation of deep learning models is generally high. CNNs require a long time to train because of the problem of vanishing and expanding gradients, regardless of the training data size [22]. Thus, CNNs can be considered to be a highly inefficient model for lightweight devices. MLP also had a characteristic in which the latency increased rapidly as the training data size increased, owing to the complexity of the model. The KNN is efficient in environments with a small training dataset. However, the latency increased rapidly as the training data size increased, rendering it the most inefficient model after the deep learning model. This is because all the data are compared in the KNN learning process, and as the data size increases, the training time increases rapidly. By contrast, RF and LR had the shortest training times. LR with relatively low complexity requires a small amount of latency regardless of the training data size. In addition, in an environment with a small data size, RF requires a small latency at 0.2 s, even though it had the second-highest latency of the five models. However, RF requires the least time of the five models in an environment where the training data size increases.

In this study, when the data were sampled at a rate of 1%, the accuracy performance converged to 98%, also it was before the rapid increase in memory usage and latency. Therefore, the data sampling ratio showing optimal performance in the lightweight device under this experimental condition is 1%. In other words, the experimental results indicate that in lightweight IoT, data size decisions that do

not degrade performance are important and experimentally optimizable. In addition, the CNN was efficient in terms of accuracy among learning models, despite being inefficient in terms of memory usage and latency. Therefore, these models are more suitable for high-power servers instead of being used for detecting vulnerable software in lightweight devices. MLP is inefficient in all three evaluation metrics and is relatively unsuitable for lightweight devices because it is a highly complex model. KNN is efficient in terms of memory usage. However, KNN has low accuracy in environments with a small training data size, and it is inefficient in terms of latency in environments where the data size increases. LR is efficient in terms of accuracy and latency; however, memory usage increases rapidly as the training data size increases. By contrast, RF was the most efficient in terms of accuracy, memory usage, and training time. RF maintains the same accuracy as that achieved with 100% of the data despite learning only from 1%. In addition, it reduces memory usage by 85.18% and improves latency by 97.82%. Therefore, although the optimal data sampling ratio is 1%, RF has the most stable performance in lightweight devices even if the data size is increased.

5 Conclusions

The introduction of 5G networks has ushered in hyperconnection, and the use of IoT devices has increased. Therefore, software vulnerabilities arising from lightweight IoT devices are increasing. In the past, machine learning-based vulnerability detection and response methods were primarily run on powerful servers. However, performing data learning efficiently in resource-constrained IoT devices is challenging. Therefore, this study proposed a technique for optimizing the data sampling ratio to efficiently detect software vulnerabilities in lightweight IoT devices. Furthermore, the performance of each machine learning and deep learning model was evaluated to detect the most frequently occurring vulnerability types in terms of accuracy, memory usage, and latency. Consequently, a data sampling ratio of 1% is the optimal data size, which was determined under experimental conditions. In addition, in terms of learning models, CNN, a deep learning model, was inefficient for resource usage, and MLP was inefficient for accuracy and resource usage. By contrast, the performance of machine learning models, KNN and LR, was significantly affected by training data size. In comparison, RF showed overall stable performance in terms of accuracy and memory usage. This study showed that an optimal learning environment exists considering the resources of IoT devices for each learning environment. Based on this, in future work, the detection accuracy of vulnerabilities must be improved by creating an optimal learning environment within the IoT device environment by optimizing the performance for each learning environment.

This study evaluated five learning models by using only one software vulnerability dataset. In addition, resource usage, such as complexity, which indicates whether the models can operate on IoT devices, must be evaluated. In future studies, we will compare the detection performance of each model using various datasets, and additional resource usage will be considered to demonstrate the operability of IoT devices. In addition, a classifier that guarantees high detection performance through model optimization will be studied. We will also present a novel detection framework to detect unknown vulnerabilities. Furthermore, the performance of the model can be evaluated by implementing the model proposed in this study. Finally, we plan to study adaptive data sampling techniques in non-independent and identically distributed data environments.

Acknowledgement: This study was revised and supplemented from a paper presented at the 6th International Symposium on Mobile Internet Security (MobiSec'22) conference.

Funding Statement: This work was partly supported by a National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (MSIT) (No. 2020R1F1A1061107), the Korea Institute for Advancement of Technology (KIAT) grant funded by the Korean Government (MOTIE) (P0008703, The Competency Development Program for Industry Specialists), and the MSIT under the ICAN (ICT Challenge and Advanced Network of HRD) program (No. IITP-2022-RS-2022-00156310) supervised by the Institute of Information & Communication Technology Planning and Evaluation (IITP).

Author Contributions: So-Eun Jeon: Conceptualization, Methodology, Software, and Writing-Original Draft. **Sun-Jin Lee:** Methodology, Validation, Resources, and Writing-Original Draft. **Il-Gu Lee:** Conceptualization, methodology, writing-review and editing, supervision, project administration, and funding acquisition.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] J. Ahn, I. Lee and M. Kim, "Design and implementation of hardware-based remote attestation for a secure internet of things," *Wireless Personal Communications*, vol. 114, pp. 295–327, 2020.

[2] B. Lee, I. G. Lee and M. Kim, "Design and implementation of secure cryptographic system on chip for internet of things," *IEEE Access*, vol. 10, pp. 18730–18742, 2022.

[3] Statista. 2022. "Number of internet of things (IoT) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030," [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide. last viewed 17 September 2022.

[4] Paloalto networks. 2020. "2020 unit 42 IoT threat report," [Online]. Available: https://unit42.paloaltonetworks.com/iot-threat-report-2020/. last viewed 17 September 2022.

[5] H. Lee, "Intrusion artifact acquisition method based on iot botnet malware," *Journal of IoT Convergence*, vol. 7, no. 3, pp. 1–8, 2021.

[6] Cloudflare. 2016. "What is the Mirai botnet?" [Online]. Available: https://www.cloudflare.com/ko-kr/learning/ddos/glossary/mirai-botnet. last viewed 21 September 2022.

[7] K. Sunghoon and N. Hyunjun, 2022. "Cctv robbed 10,000 routers," [Online]. Available: https://www.mk.co.kr/news/it/view/2022/01/57997. last viewed 17 September 2022.

[8] J. Gold, 2022. "IoT is turning into a service," [Online]. Available: https://www.ciokorea.com/news/221754. last viewed 17 September 2022.

[9] K. Murat, C. Fair and O. Guler, "Role of artificial intelligence in the internet of things (IoT) cybersecurity," *Discover Internet of Things*, vol. 1, no. 3, pp. 1–4, 2021.

[10] D. Abebe, N. Chilamkurti, V. Nguyen and W. Heyne, "A comprehensive study of anomaly detection schemes in iot networks using machine learning algorithms," *MDPI Sensors*, vol. 21, no. 24, pp. 8320, 2021.

[11] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin *et al.,* "Vuldeepecker: A deep learning-based system for vulnerability detection," in *Proc. Network and Distributed Systems Security (NDSS) Symp. 2018*, San Diego, CA, USA, 2018.

[12] Z. Bilgin, M. A. Ersoy, E. U. Soykan, E. Tomur, P. Çomak *et al.,* "Vulnerability prediction from source code using machine learning," *IEEE Access*, vol. 8, pp. 150672–150684, 2020.

[13] G. J. Blinowsk and P. Piotrowski, "CVE based classification of vulnerable IoT systems," in *Int. Conf. on Dependability and Complex Systems*, Geneva, Switzerland, Springer, pp. 82–93, 2020.

[14] W. Niu, X. Zhang, X. Du, L. Zhao, R. Cao *et al.,* "A deep learning based static taint analysis approach for IoT software vulnerability location," *Measurement*, vol. 152, no. 3, pp. 107139, 2019.

[15] M. Zolanvari and L. Gupta, "Machine learning-based network vulnerability analysis of industrial internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6822–6834, 2019.
[16] M. Hasan, M. Islam, I. I. Zarif and M. M. A. Hashem, "Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches," *Internet of Things*, vol. 7, pp. 100059, 2019.
[17] A. Kumar and T. J. Lim, "EDIMA: Early detection of IoT malware network activity using machine learning techniques," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, Limerick, Ireland, pp. 289–294, 2019.
[18] L. Kim and R. Russell, 2018. "Draper vdisc dataset-vulnerability detection in source code," [Online]. Available: https://osf.io/d45bw/. last viewed 17 September 2022.
[19] R. L. Russell, L. Kim, L. H. Hamilton, T. Lasovich, J. A. Harer *et al.,* "Automated vulnerability detection in source code using deep representation learning," in *17th IEEE Int. Conf. on Machine Learning and Applications (ICMLA)*, Orlando, Florida, USA, pp. 757–762, 2018.
[20] A. Klautau, P. Batista, N. González–Prelcic, Y. Wang and R. W. Heath, "5G mimo data for machine learning: Application to beam-selection using deep learning," in *2018 Information Theory and Applications Workshop (ITA)*, San Diego, CA, USA, pp. 1–9, 2018.
[21] V. Praveen Kumar and I. Sowmya, "A review on pros and cons of machine learning algorithms," *Journal of Engineering Sciences*, vol. 12, no. 10, pp. 272–276, 2021.
[22] G. Liu, H. Kang, Q. Wang, Y. Tian and B. Wan. "Contourlet-CNN for SAR image despeckling," *Remote Sensing*, vol. 13, no. 4, pp. 764, 2021.