Received 17 April 2024, accepted 11 May 2024, date of publication 16 May 2024, date of current version 23 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3401853

RESEARCH ARTICLE

Tackling Evolving Botnet Threats: A Gradual Self-Training Neural Network Approach

TA-CHUN LO[1], JYH-BIAU CHANG[2], SHAO-HSUAN LO[1], BAI-JUN KAO[1], AND CE-KUEN SHIEH[1], (Senior Member, IEEE)
[1]Department of Electrical Engineering, National Cheng Kung University, Tainan 70101, Taiwan
[2]Department of Electronic Engineering, Lunghwa University of Science and Technology, Taoyuan 33306, Taiwan

Corresponding author: Ta-Chun Lo (N28091108@gs.ncku.edu.tw)

ABSTRACT Botnets pose a significant challenge to network security but are difficult to detect because of their dynamic and evolving nature, which limits the effectiveness of conventional supervised neural network detection methods. To address this problem, the present study proposes a novel neural network-based self-training framework for botnet detection, in which pseudo-labels are generated from unlabeled data by a trained classifier, which is iteratively refined over time using a combined dataset containing both training and pseudo-labeled data. Although not all of the generated pseudo-labels are applicable to every botnet, the self-training framework can label unseen botnets with behaviors similar to those of known botnets with high confidence. Several strategies are proposed for enhancing the robustness of the classification performance by minimizing the number of incorrect pseudo-labels, mitigating the effects of erroneous pseudo-labels on the overall performance of the network, and optimizing the proportion of unlabeled data for labeling. Experiments conducted on both synthetic datasets confirm the superiority of the proposed method over the base model, particularly when the training data constitutes only a small portion of the total amount dataset. Subsequent experiments also demonstrate the efficacy of the framework in successfully detecting unseen botnet variants and its commendable performance in real-world campus network traffic.

INDEX TERMS Botnet detection, NetFlow, network security.

I. INTRODUCTION

Botnets are one of the greatest threats in the cyber world nowadays and are responsible for a wide range of malicious activities, including Distributed Denial of Service (DDoS) attacks, spam dissemination, and data breaches, which can cause extensive havoc, risk to life, and severe damage, both financial and physical. The identification and mitigation of botnet attacks are thus of fundamental importance for protecting network infrastructures and preserving user privacy. Nonetheless, the intricacy of botnet techniques and the dynamic nature of their behavioral patterns present formidable challenges to the development of effective and resilient detection models. For example, most botnets undergo continuous evolution in an attempt to avoid detection. Thus, conventional machine learning-based approaches that rely on labeled data extracted from previous botnet attacks to train their models are insufficient, given that the behavior of new bots often varies considerably from that of previous bots.

The associate editor coordinating the review of this manuscript and approving it for publication was Mueen Uddin.

Modern botnet detection systems generally use supervised learning methodologies based on machine learning algorithms or neural network models [1], [2], [3]. Such methods provide an effective and efficient means of analyzing behavioral changes in flow-based traffic. However, their efficacy depends heavily on the availability of sufficient representative labeled data to achieve robust training of a generalized model. The acquisition of labeled network traffic must be undertaken within a controlled and simulated environment, which incurs a significant time investment and cost, particularly when simulating the complex behaviors of typical botnet attacks.

Furthermore, as mentioned above, botnets are extremely dynamic and constantly adapt and evolve in order to avoid detection and continue their malicious activities. Thus, the labeled data obtained from the analysis of previous botnet

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

behaviors are not guaranteed to achieve a reliable detection performance for new botnet variants. In other words, to ensure the protection of network environments, supervised machine learning models are no longer sufficient, and it is necessary instead to develop more adaptive monitoring systems capable of detecting and classifying potentially malicious traffic flows as soon as they become apparent.

The acquisition of unlabeled network traffic is far more straightforward than that of labeled data and can be achieved in real time using simple network monitoring systems, such as network traffic analyzers or intrusion detection systems (IDSs). Thus, interest has grown in the potential to utilize this information to enhance the performance of traditional supervised machine learning-based detection methods. It is impractical to use all of the collected unlabeled data to enhance the performance of the model. Instead, it is necessary to consider only the unlabeled data that appear to be potentially associated with a new abnormal behavior. To achieve this goal, the data distribution must be classified in some way. In the present study, this is performed using a binary classification method, as illustrated in Fig. 1.

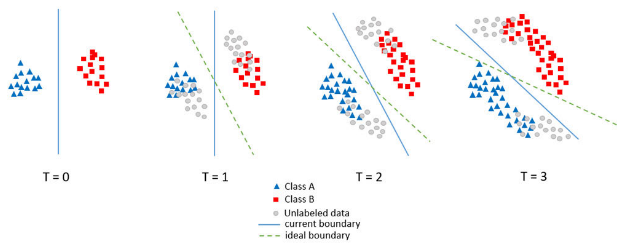

FIGURE 1. **Data shift diagram.**

Figure 1 shows four illustrative data distributions observed in the network at four distinct temporal instances. The binary classification scheme comprises two classes, A and B, denoted by the blue triangles and red squares, respectively. The blue solid line indicates the decision boundary of the current model, while the green dashed line represents the optimal decision boundary at the specified time point. Initially, all the data points are labeled, and the classification model is trained using these labeled data, resulting in the decision boundary illustrated by the blue vertical line at $T = 0$. At $T = 1$, the data distribution includes a new set of unlabeled data, represented by the gray circles. Owing to an underlying data shift, the distribution of these unlabeled data points is not perfectly aligned with that of the pre-existing labeled data.

As a result, the original decision boundary is no longer appropriate for classification purposes but must be adapted on the basis of the new knowledge contained by the unlabeled data and replaced with a more optimal boundary (shown by the green dashed line at $T = 1$). Furthermore, the decision boundary must continue to be adapted in this way as the data distribution evolves over time, as shown at $T = 2$ and $T = 3$. The adaption of the decision boundary should ideally be performed autonomously without the intervention of a human operator and should be based on the data points that have undergone shifts from the previous interval, as opposed to the data points that adhere to the same distribution as the extant data.

Accordingly, the present study proposes an innovative self-training framework for botnet detection in which pseudo-labels are generated from unlabeled data by a trained classifier, which is iteratively updated over time using a dataset containing both labeled and pseudo-labeled data. Intuitively, the classifier trained in this manner should provide a high level of confidence in detecting unseen botnets with a behavior similar to that of known botnets. However, for new botnets with different behavioral characteristics, the pseudo-labels derived from previous data distributions may be unreliable or even inapplicable. Thus, the robustness and cost-effectiveness of the proposed self-learning framework are enhanced through the incorporation of several strategies aimed at minimizing the effects of erroneous pseudo-labels on the overall network performance and determining the optimal percentage of unlabeled data to be labeled in each training epoch.

Previous studies on the use of self-training methods to overcome the challenge of insufficient labeled data predominantly rely on synthetic datasets for experimental purposes [10], [27]. Moreover, in these studies, the training data are typically randomly sampled from the entire dataset, thereby ensuring that the distribution of the training set accurately reflects that of the complete dataset. However, while this approach is conducive to the application of self-training, it is less suitable for the realm of real-world traffic, in which the labeled data employed for model training are inherently derived from historical data, and the distribution of these data points often diverges from that of future encounters. Thus, the framework proposed in this study is further optimized for the detection of botnets in real-world networks through a specialized pseudo-labeling technique, wherein only high-confidence data points are selected for training to ensure the efficacy of the derived pseudo-labels in improving the detection performance for future data or previously unobserved botnet types.

In this work, we build a semi-supervised botnet detection framework to learn from both labeled and unlabeled traffic. The main contributions of this study can be summarized as follows:

1. Improve the performance when the labeled training data is relatively small.

2. Significantly improve the performance of unseen botnet detection.

3. Mitigate the effects of erroneous pseudo-labels, i.e., the error amplification issue.

4. Instead of training directly with the whole dataset, this study gradually adapts the model over time to achieve higher performance in real-world traffic

5. Improve the performance and reduce the computation cost in large real-world traffic data.

II. BACKGROUND AND RELATED WORK

Recent years have witnessed a surge in machine learning research focused on botnet detection. Many studies [5], [6], [7], [8] have used features extracted from network flow data to train machine learning models. Broadly speaking, these training methods can be categorized into three main types: supervised learning, semi-supervised learning, and unsupervised learning.

Supervised learning [7], [8] involves training models using labeled data, i.e., data points that are already classified as either botnets or non-botnets. However, although supervised learning provides accurate predictions, its performance depends on the availability of precisely labeled data in large quantities. Moreover, supervised models have only a limited ability to detect novel or evolving botnets with unseen characteristics.

Semi-supervised learning approaches [5] extend the supervised learning paradigm by incorporating both labeled and unlabeled data. In such methods, the model is initially trained using labeled data and is then refined using pseudo-labels created from the information extracted from unlabeled data. Semi-supervised learning leverages a broader pool of available data, thereby potentially enhancing the model performance compared with that obtained using limited labeled data alone. However, its effectiveness is heavily dependent on the quality of the pseudo-labeling method employed, and its performance is sensitive to noisy or mislabeled unlabeled data. Finally, unsupervised learning approaches [6] aim to make decisions autonomously based on the observed patterns and structures within the unlabeled dataset. In particular, they group similar samples based on their extracted features and then define rules for future decision-making.

Unsupervised learning methods can reveal hidden patterns and insights without the need for labeled samples and, therefore, offer the potential for novel discoveries and eliminate the annotation burden. However, their outcomes are highly dependent on the quality of the algorithms used to perform feature extraction and clustering. Moreover, the absence of labeled data can limit the interpretability and validation of the results. In addition, the significant accumulation of data over time makes real-time detection extremely challenging. Accordingly, the present study proposes a self-training approach based on the semi-supervised learning paradigm, in which the model is gradually adapted over time based on the information gathered from new unlabeled data in the evolving dataset. The remainder of this section reviews related work in the self-training field and explains the strategies adopted in the present study inspired by these studies.

Lee et al. [9] proposed a self-training method incorporating pre-training and error-forgetting techniques aimed at mitigating the error amplification challenge in self-training. Error forgetting involves two strategies: weight resetting and re-evaluation. The former strategy updates the model weights in each self-training epoch to prevent stagnation, while the latter strategy re-evaluates the pseudo-labels assigned to the unlabeled samples in each epoch. The results in [9] showed that the two strategies effectively mitigate the impact of incorrect pseudo-labels on the model performance. Thus, the framework proposed in the present study also adopts both strategies to enhance the overall performance of the model.

Kumar et al. [11] considered the problem of adapting a machine learning system to an evolving data distribution over time in the image recognition field. In particular, the authors addressed the problem of gradual domain adaptation, in which an initial classifier was trained on a source domain and was then gradually adapted using only unlabeled data that gradually shifted toward the target domain. The authors provided the first non-trivial upper bound on the error of self-training in the presence of gradual shifts. In addition, theoretical analyses were performed that demonstrated the importance of regularization and label sharpening even in the presence of infinite data. The experimental results obtained on rotating MNIST dataset and realistic portrait datasets demonstrated the improved accuracy obtained by leveraging a gradual shift structure in adapting the source domain to the target domain rather than directly adapting to the target domain. Accordingly, the present framework also adopts a gradual self-training approach to deal with the real-world problem of botnet detection.

Koza et al. [10] compared the performance of two semi-supervised deep neural network algorithms on a large real-world malware dataset. The first algorithm utilized unlabeled samples and took the high-confidence classification results as pseudo-labels in subsequent classification rounds. The second algorithm aimed to enhance the prediction consistency of the network under altered circumstances (e.g., different data augmentation strategies and/or dropout settings). For reference purposes, the results obtained using the two algorithms were compared with those obtained from the same deep network trained in a fully supervised mode using only labeled data. The results showed that both semi-supervised learning methods achieved only a marginal improvement over the baseline model when the proportion of labeled data within the entire dataset exceeded 10%. In other words, the randomly sampled training subset was sufficient to properly represent the distribution of the entire dataset, thereby allowing the baseline model to achieve a reasonable performance even with only a small proportion of labeled data. However, when training was performed using data acquired over an initial five-week period, and testing was then conducted on new data collected several weeks later, all three models showed a marked degradation in performance, thereby confirming the challenge posed by the data shift problem.

Mahdavifar et al. [12] presented a pseudo-label stacked auto-encoder approach for the Android malware classification. To minimize the annotation burden, the proposed method adopted a semi-supervised learning method in which both dynamic and static analyses were used to craft feature vectors. The experimental results obtained on the

CICMalDroid2020 dataset showed that the proposed model was more accurate and efficient than other state-of-the-art techniques. Zhang et al. [5] proposed an active semi-supervised learning approach for network instruction detection, in which pseudo-labels were assigned to unlabeled data using a minimum class-distance threshold for active learning and a high classification threshold for semi-supervised learning and then selecting the high confidence unlabeled samples for labeling and addition to the training set. Gelian et al. [14] proposed a self-learning botnet detection system utilizing an ensemble classifier. The classifier updates the model continuously based on the knowledge derived from new unlabeled traffic flows to enhance the generalization capacity. The results demonstrated the system's success in adapting to dynamic environments containing data generated by new botnet types.

Along with the rise of the botnet threat, researchers have proposed several solutions in the botnet detection field. M. A. Setitra et al. proposed several works [33], [34], [35] to detect DDoS attacks, a typical attack type held by botnet attackers. In [33], the authors proposed a pre-processing model to extract features in proper representation based on the Total of Possible Unique Values (TPUV). Eight different machine learning algorithms are then used to evaluate the proposed method. In [34], the authors build a botnet detection system using the SHapley Additive exPlanations (SHAP) feature selection method, multilayer perception, and convolution neural network model. The system proves its performance on the CICDDoS-2019 and the InSDN dataset. In [35], the authors proposed another DDoS detection system by combining the AutoEncoder and the XGBoost algorithm. The system achieves a 99.992% accuracy in the experiment result.

Hossain et al. [36] amalgamates common algorithms, including categorical analysis, mutual information, and principal component analysis, to extract representative features from the dataset. The extracted features are then used to train an ML model. The experiment result achieves an accuracy exceeding 99.99% when using Extra Trees as the classifier. On the other side, Zhou et al. [37] focus on energy efficiency and resource conservation. The authors proposed an ML-based lightweight ensemble-based model to detect botnet activities in the IoT environment. The proposed model integrates two machine learning algorithms, namely XGBoost and LightGBM. The experiment shows an outstanding performance of the proposed method.

Velasco-Mata et al. [38], in the other way, focus on the Time-to-Detection (TTD) issue in the botnet detection field. TTD refers to the time window between a network attack and the detection of the threat. Faster TTD can effectively limit the attacker's operating space and reduce the damage caused by intrusions. In [38], the authors propose an approach that works on the time windows of one second, with a processing time of 0.007 milliseconds per sample.

The state-of-the-art solves the various issues in the botnet detection field with outstanding performance in each experiment. However, in this work, we attempt to solve the relatively small training data issue and the data shift problem, which real-world applications may encounter.

III. METHODOLOGY

A. OVERVIEW

The proposed self-training process comprises three discrete stages, as illustrated in Fig. 2. The first stage encompasses the initialization phase, in which the accumulated labeled data are used to train a foundational model. The second stage comprises the model inference phase, during which the model undergoes continual refinement by processing incoming unlabeled data and generating predictions accordingly. The prediction outcomes are treated as pseudo-labeled data and are incorporated into the dataset for subsequent self-training. However, not all the pseudo-labeled data are used for training. Instead, the proposed framework selects only the pseudo-labels with the highest confidence for addition to the training set (as described in Section III-E). The third stage consists of the model update stage, wherein the dataset containing the original labeled training data and pseudo-labeled data derived from the model inference phase is processed using a self-training algorithm. Upon completion of the model update, the process cycles back to the second stage, where new unlabeled data are received for further inference.

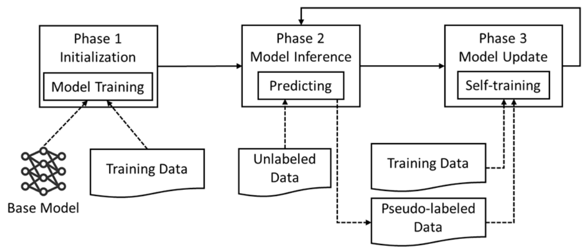

FIGURE 2. **Gradual self-training workflow.**

Figure 3 presents a more detailed view of the gradual self-training process. The initial foundation model is denoted as M_0. After a period of model inference, self-training is performed using the accumulated pseudo-labeled data, resulting in an updated model, M_1. At the end of this stage, the high-confidence data selected in the final epoch are permanently added to the training set and undergo no further reevaluation in subsequent self-training iterations. Typically,

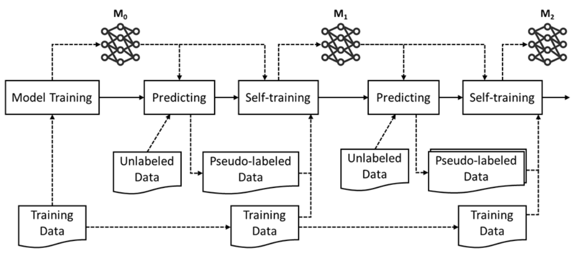

FIGURE 3. **Gradual self-training timeline.**

around 40–60% of the unlabeled data are chosen as high-confidence data. Thus, the pseudo-labeled data is expanded, as shown by the multiple document input of the second self-training process. The larger dataset potentially leads to an improved performance and generalization ability in subsequent iterations of the self-training process.

B. DATA PREPROCESSING

Since NetFlow contains only unidirectional information, it is unable to retrieve the entire communication pattern between two hosts in NetFlow-based detection. Thus, in the present study, the NetFlow data are merged into bidirectional sessions using the method described in [6]. In particular, a NetFlow log is defined as follows:

$$F_i = (\mathrm{SIP_i}, \mathrm{SP}_i, \mathrm{DIP}_i, \mathrm{DP}_i, P_i, T_i)\,, \tag{1}$$

where SIP is the source IP address, SP is the source port number, DIP is the destination IP address, DP is the destination port number, P is the protocol, and T is the timestamp. Each session, S, comprises a series of NetFlow logs with a time interval between consecutive logs less than a certain threshold T_{th} (21 seconds for TCP and 22 seconds for UDP by default.). The threshold is set as the default connection timeout of the Microsoft Windows operating system, i.e.,

$$S_i = \sum\nolimits_j F_j\left(\mathrm{SIP}_i, \mathrm{SP}_i, \mathrm{DIP}_i, \mathrm{DP}_i, P_i, T_j\right) \tag{2}$$

$$\forall F_j, F_{j+1} \in S_i, T_{j+1} - T_j < T_{th}, \tag{3}$$

Given that each session comprises multiple NetFlow logs, it is necessary to extract a set of novel features from these logs that properly encapsulate the characteristics of the entire session. Notably, the extracted features should be carefully defined such that they are capable of distinguishing potential botnet communications from normal activities. The present self-training framework employs the 20 representative features listed in Table 1 [6].

TABLE 1. Session features.

Features	Description
Forward Pkts	Number of packets forward to destination IP
Forward Bytes	Number of bytes forward to destination IP
Forward MaxBytes	Max size of packet forward to destination IP
Forward MinBytes	Min size of packet forward to destination IP
Forward MeanByte	Mean size of packets forward to destination IP
Backward Pkts	Number of packets backward to source IP
Backward Bytes	Number of bytes backward to source IP
Backward MaxBytes	Max size of packet backward to source IP
Backward MinBytes	Min size of packet backward to source IP
Backward MeanByte	Mean size of packets backward to source IP
Flow Pkts	Number of total packets
Flow Bytes	Number of total bytes
Flow MaxBytes	Max size of total packets
Flow MinBytes	Min size of total packets
Flow MeanByte	Mean size of total packets
Flow STDByte	Standard deviation of total packet size
Flow BytesRate	Proportion Of total bytes to duration
Flow Pkts Rate	Proportion Of total packets to duration
Flow IORatio	Proportion of total incoming packet size to total outgoing packet size
Duration	Duration of session

Having consolidated the NetFlow logs into sessions, the initialization stage is activated. During this stage, the training dataset is divided into two distinct parts: the labeled data and the unlabeled data. The labeled data plays a pivotal role in both the model pretraining stage and the self-training stage (see Fig. 3). Here, the labeled data serves as the ground truth, ensuring the correctness of the labels. Conversely, the unlabeled data exclusively contributes to the self-training stage, during which the unlabeled data undergoes pseudo-labeling, and the resulting pseudo-labeled data is amalgamated with the labeled data for subsequent model retraining.

C. INITIALIZATION STAGE

Before commencing the self-training process, it is necessary to establish a reliable foundational model to ensure the accuracy of the pseudo-labeling outcomes. Therefore, in the initialization stage, the model is pre-trained using the labeled data generated during the data partitioning phase of the prior data preprocessing stage. In general, the architecture of a deep learning model depends on its intended purpose. In the present study, the model output should indicate the probabilities of each output class. Thus, the activation function of the final layer of the deep learning model uses SoftMax. Beyond this requirement, however, the proposed framework imposes no additional constraints on the model architecture.

D. MODEL INFERENCE STAGE

The Model Inference Stage is positioned between the Initialization Stage and the subsequent Model Update Stage (or another subsequent Model Update Stage in the later model refinement process). During the Model Inference Stage, the detection model is deployed to identify potentially malicious network traffic in the operational environment, and real-time network traffic is collected as unlabeled data. Once the unlabeled dataset has accumulated sufficient data, the framework advances to the Model Update Stage and initiates the self-training process. Upon completion of the self-training process, the framework reverts to the Model Inference Stage and repeats the deployment and detection activities.

E. SELF-TRAINING STAGE

1) SELF-TRAINING WORKFLOW

Figure 4 illustrates the basic workflow of the self-training framework. Initially, the unlabeled data collected in the preceding stage are labeled using the pseudo-labeling method described in the following section. The pseudo-labeled data are combined with the original training data, and a new model, termed the re-trained model, is constructed. This re-trained model mirrors the architecture of the pseudo-labeling model and undergoes training with the pseudo-labeled data. After training, the performance of the re-trained model is evaluated on validation data. If the re-trained model outperforms the pseudo-labeling model, the weights of the re-trained model are transferred to the pseudo-labeling model; otherwise, the original weights are retained. This

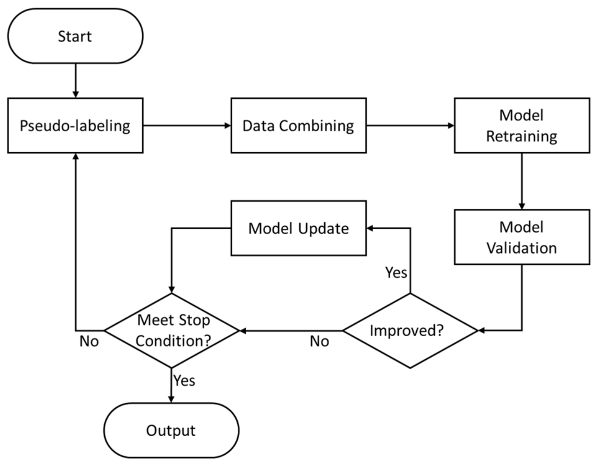

FIGURE 4. Self-training framework.

```
Algorithm 1 Self-training Framework
Input: Labeled, Unlabeled
Output: Final Model
1:  // Configuration
2:  Train, Valid ← Split(Labeled)
3:  PM ← BuildAndTrain(Train)
4:  BS ← Evaulate(PM, Valid)
5:  P ← GetProportion()
6:
7:  // Self-training
8:  CurrP ← First(P)
9:  Cepoch ← 0
10: while CurrP not end of P and Cepoch < 3 do
11:     PD ← PseudoLabel(PM, Unlabeled, CurrP)
12:     DT ← Train U PD
13:     RM ← BuildAndTrain(DT)
14:     Score ← Evaluate(RM, Valid)
15:     if Score > BS then
16:         PM ← RM
17:         BS ← Score
18:         Cepoch ← 0
19:     else
20:         CurrP ← Next(CurrP)
21:         Cepoch ← Cepoch + 1
22:     end if
23: end while
24: return PM
```

FIGURE 5. Implementation of self-training framework.

process continues iteratively until the designated number of self-training epochs have been performed. The latest version of the pseudo-labeling model is then taken as the final model since it possesses the optimal weights obtained over the entire self-training procedure.

Although the iterative process described above is designed to continue until all the self-training epochs are completed, an early-stop mechanism is incorporated into the self-training process, wherein the training process is terminated if the unlabeled data are very different from the labeled data and thus prevents the pseudo-labeling model from being further improved. The mechanism thus not only improves the training efficiency but also safeguards the robustness and performance of the final model.

Figure 5 presents the pseudocode of the self-training framework, and Table 2 defines the parameters used in the pseudocode. Refer to Fig. 5, the self-training framework starts by generating pseudo-labels for all the unlabeled data in each self-training epoch and adds the high-confidence data to the pseudo-labeled data by referencing the proportion of the current self-training epoch. The pseudo-labeled data are then combined with the training data (i.e., the labeled data) to create so-called dynamic training data. These data are used to build a new model with the same architecture as the pseudo-labeling model called the re-trained model. The re-trained model is trained on the validation data, and the F1-score is determined. If the F1-score is higher than the previous best F1-score, the weights of the re-trained model are transferred to the pseudo-labeling model, and the best F1-score is updated. Otherwise, the proportion of unlabeled data used for pseudo-labeling (P) increases in the next self-training epoch.

Note that P is deliberately not increased when the F1-score improves in order to understand whether the same (or better) performance is consistently obtained as the training process proceeds using the same value of P (i.e., P represents the optimal labeling proportion). Conversely, P is increased when the F1-score fails to improve because the unlabeled data are deemed to be insufficient to fully represent the characteristics of the dynamic training data. The process continues until all possible values of P have been explored or no improvement in the F1-score is observed over three consecutive self-training epochs, which indicates that the remaining unlabeled data are low-confidence samples and do not contribute to enhancing the learning performance of the model.

TABLE 2. Parameters of pseudo-code.

Parameter	Description
Train, Valid	Training Data, Validation Data
Labeled, Unlabeled	Labeled Data, Unlabeled Data
PM, RM	Pseudo-Labeling Model, Re-Trained Model
PD, DT	Pseudo-Labeled Data, Dynamic Training Data
P	An increasing array indicates the target of the pseudo-labeling proportion in the unlabeled data
CurrP	Current Value of P
BS	Best F1-Score during self-training
B	Base Proportion
Range	The value of proportion to be increased
RSort	Sort the array from high to low
Cepoch	Consecutive self-training epoch since last BS

2) PSEUDO-LABELING METHOD

Figure 6 shows the workflow of the proposed pseudo-labeling algorithm. As shown, the pseudo-labeling model first predicts the class of all the unlabeled data in every self-training epoch. As described above, the classifier applies the SoftMax activation function in the final layer. Thus, for each unlabeled data point, the classifier determines the probability prediction of each class (i.e., normal or botnet), and the class with the highest probability is chosen as the model output for the sample. If the probability exceeds a certain threshold, the sample is designated as a high-confidence sample and is added to the pool of pseudo-labeled data. Otherwise, the sample is dropped.

In implementing the pseudo-labeling algorithm described above, a straightforward solution is to simply assign the threshold a fixed value. However, in practical applications, the highest probability of the predicted class may differ in each self-training epoch, and hence, while a threshold of 80% (for example) may be suitable for evaluating samples with high confidence in epoch 1, a value of 90% may be more appropriate in epoch 2. Furthermore, the characteristics of the input data may vary significantly over time. Thus, the optimal threshold value for one set of input data may not be the optimal value for a second set of data collected at a different time.

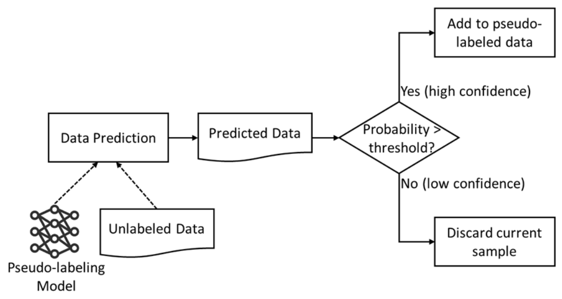

FIGURE 6. Pseudo-labeling workflow.

Accordingly, in the present study, the pseudo-labeling algorithm is implemented using a dynamic threshold strategy. Notably, the threshold is set based on the concept of gradually increasing the number of samples to be labeled during the self-training process to approach the optimal percentage. In particular, the threshold is implemented not by adjusting the threshold value (a percentage) each time but by incrementally increasing the proportion of unlabeled data to be labeled in every self-training epoch from one epoch to the next. Once the proportion of unlabeled data to be labeled has been set, the unlabeled data are sorted from high to low in order of the probability of their predicted class. The threshold value in the corresponding epoch is then taken as the lowest probability of the predicted class of the sample within the corresponding epoch's percentile.

It should be noted that the upper bound percentage of the unlabeled data to be labeled cannot be directly assigned a specific value because its value depends on the distribution characteristics of the unlabeled data, which may vary significantly in different epochs. For example, if the distribution of the unlabeled data differs greatly from that of the training data, it is nonsensical to process all the unlabeled data because this may degrade the performance of the base model (particularly if the model already has a high performance). Conversely, if the unlabeled data are closely aligned with the training data, assigning pseudo-labels to most of the unlabeled data is likely to improve the performance of the base model (particularly if the present model has a low performance). Accordingly, in the present framework, the entirety of the unlabeled data is taken as the maximum percentage to be labeled, and two variables, namely the base proportion and the range, are introduced to govern the increment in the pseudo-labeling proportion in each epoch.

The base proportion indicates the proportion of unlabeled data to be labeled in the first self-training epoch. It is deliberately assigned a low value to minimize the training time by using only a small percentage of pseudo-labeled data. However, if all the unlabeled data have high confidence, the model learns nearly nothing new in the subsequent self-training rounds. Accordingly, the range parameter is used to specify an increment in the labeling proportion in each epoch. If the range is small, the optimal labeling proportion can be determined more precisely. However, the learning performance of the model is likely to increase very slowly, if at all, and hence the training time is increased. Conversely, if the labeling proportion grows too rapidly, the training time is reduced; however, the precision of the optimal labeling proportion is most likely degraded. Thus, the two parameters should be set under different usage. In the present experiments, the base proportion was set as 0.15, and the range was set as 0.05.

Figure 7 presents the pseudocode for the pseudo-labeling method. As discussed above, the method commences by predicting the class probabilities of all the unlabeled data using the pseudo-labeling model. For each sample, a probability prediction is obtained for each class, and the sample is assigned to the class with the highest probability. The samples are then sorted from high to low according to their predicted class probabilities. The threshold of the current self-training epoch is taken as the probability of the predicted class of the current percentile P. Finally, the algorithm iterates through all the samples and adds the samples with a predicted class probability equal to, or greater than, the threshold to the pool of pseudo-labeled data for the following pre-training epoch.

```
Algorithm 2 Pseudo-labeling Algorithm
Input: PM, Unlabeled, CurrP
Output: PD
procedure PseudoLabel(PM, Unlabeled, CurrP)
    // Variables
    DLen ← Len(Unlabeled)
    Confidence ← EmptyArray(DLen) of DOUBLE

    // Pseudo-labeling
    Predictions ← Predict(PM, Unlabeled)
    for i ← each data in Predictions do
        Confidence[i] ← ProbabilityOfHighestClass(Predictions[i])
    end for
    Confidence ← Rsort(Confidence)
    Threshold ← Confidence[ DLen * CurrP ]
    for p ∈ Predictions do
        if ProbabilityOfHighestClass(p) ≥ Threshold then
            PD ← PD U GetData(p)
        end if
    end for
    return PD
return procedure
```

FIGURE 7. Implementation of pseudo-labeling method.

3) ERROR-AMPLIFICATION ISSUE

Error amplification occurs when the model inference stage assigns an incorrect label to an unlabeled data point, and the model then conducts training using the mislabeled data. In this situation, the training process enters a vicious circle in which the model learns the misinformation falsely generated in previous epochs. Consequently, the confidence value of the data increases (albeit incorrectly), thereby ultimately destroying the performance of the model. For mitigating the effects of erroneous pseudo-labels, the proposed framework uses the weight reset and re-evaluation mechanisms proposed by Lee et al. [9].

The weight reset mechanism initializes the model weights at the onset of each Model Update Stage. This ensures that the model starts training anew after the completion of each pseudo-labeling phase, thereby preventing the propagation of undesirable weights from the preceding self-training epoch. Given that the original training data contributes to each epoch in the Model Update Stage, frequent model updates may cause the model to become excessively attuned to the training data. Thus, the weight reset mechanism also serves the purpose of suppressing model overfitting. Meanwhile, the re-evaluation mechanism performs pseudo-labeling of all the unlabeled data during the Model Inference stage, not just the newly arrived unlabeled data. Thus, it prevents mislabeled data from enduring indefinitely in the training set without the opportunity for correction.

4) REAL-WORLD DATA IMPROVEMENT

Due to the substantial size disparity between real-world network traffic and synthetic data, conducting training on real-world data incurs significantly greater time and computational resources. Thus, employing the weight reset mechanism described above in this context may result in notable resource inefficiencies because each epoch entails training the model from scratch. Therefore, in this study, when performing model training with real-world data, the weight reset mechanism is omitted, and the risk of overfitting is mitigated instead by simply omitting data with exceptionally high confidence from the pseudo-labeled dataset.

This exclusion strategy is motivated by two factors. First, the self-training methodology adopted in botnet detection should possess the capability to identify variant botnets, that is, botnets that are similar to, but not identical to, the original botnet. If the pseudo-labeling process results in an exceptionally high confidence level for the unlabeled data, there is a strong likelihood that the data corresponds to a botnet already present in the training data. Training the pseudo-labeled data alongside the training data may thus induce overfitting to the original data rather than facilitating the assimilation of new behaviors characteristic of variant botnets. Second, the pseudo-labeled dataset is prone to contain inaccuracies, thereby increasing the relative contribution of the labeled training dataset. Thus, to prevent overfitting while still preserving sufficient accurate information to improve the model performance during the model update stage, it is intuitive to discard the pseudo-labeled data with excessively high confidence rather than the original training data, which are presumed to be entirely accurate.

IV. EXPERIMENTS

A. ENVIRONMENT

The experiments were conducted on a personal computer with an Intel(R) Core(TM) i7-7700 CPU running at 3.60 GHz, 16 GB of RAM, 10 TB of storage capacity, and an NVIDIA GeForce RTX 2080 Ti GPU.

B. DATASETS

Two datasets were employed: a synthetic dataset and a real-world traffic dataset.

TABLE 3. CTU-13 dataset.

ID	Sessions	Bot
1	9,463	Neris
2	5,376	Neris
3	1,634	Robt
4	629	Rbot
5	430	Virut
6	286	Menti
7	44	Sogou
8	392	Murlo
9	41,984	Neris
10	946	Rbot
11	225	Rbot
12	2,804	NSIS.ay
13	7,634	Virut

TABLE 4. N-BaIoT dataset.

Class	Flows
Benign	555,932
Mirai	3,668,402
Gafgyt	2,838,272

1) SYNTHETIC DATASET

The experiments involving synthetic data commenced by using the CTU-13 dataset [26], which encompasses 13 unique scenarios, inclusive of seven botnet traffic types and benign traffic and the N-BaIoT dataset [32]. The details of the dataset and considered scenarios are shown in Table 3 and Table 4. Note that the two datasets have different flow types and features. However, the proposed framework focuses only on improving the detection model. The input data format difference doesn't matter. The two datasets were used to validate the proposed self-training framework for well-known botnets.

The ability of the framework to detect botnet variants was then investigated using two further categories of botnet data. The first category comprised the data generated by banking Trojan botnets [15]. The most widespread banking Trojan is Zeus; however, there are also many variants or descendants of Zeus. A synthetic dataset was obtained by simulating the

TABLE 5. Banking botnet dataset.

Class	Source	Sessions
Benign	Malware Capture Facility Project	500,000
Zeus	Malware Capture Facility Project	250,000
Tinba	Malware Capture Facility Project	50,000
Vawtrak	Malware Capture Facility Project	50,000
Emotet	Malware Capture Facility Project	50,000
TrickBot	Malware Capture Facility Project	50,000
Dridex	Malware Capture Facility Project	50,000

TABLE 6. Email spam botnet dataset.

Class	Source	Sessions
Benign	Malware Capture Facility Project	500,000
Storm	PeerRush [25]	150,000
Waledac	PeerRush [25]	50,000
Kehlios	Malware Capture Facility Project	50,000
Pushdo	Malware Capture Facility Project	50,000

real-world traffic behaviors published by the Malware Capture Facility Project [28]. Seven scenarios were constructed, as shown in Table 5. The second category simulated the botnet data produced by four common email spam botnets: Storm, Waledac, Kehlios, and Pushdo [16]. Table 6 shows the details of the related dataset.

2) REAL-WORLD DATASET

The real-world dataset was collected from the National Cheng Kung University (NCKU) on the Taiwan Advanced Research and Education Network (TWAREN) between May and July 2019. The dataset was divided into three datasets (a training, an unlabeled, and a testing dataset) by time rather than by mixing and shuffling the entire dataset. The division between the three datasets is shown in Fig. 8, in which the data collected in the second week of May are considered as training data, the data collected between the third week of May and the end of June are taken as unlabeled data, and the remaining data are taken as testing data.

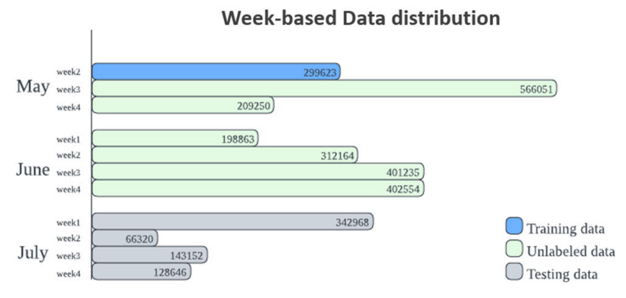

FIGURE 8. Real-world week-based data distribution.

C. EVALUATION CRITERIA

The performance of the proposed self-training framework was evaluated using four common metrics: Recall, Precision, Accuracy, and F1-score. In binary classification problems such as that considered in the present study (i.e., botnet vs. benign), the prediction outcomes are generally categorized as follows:

- True Positive (TP): The number of samples predicted as a botnet and actually belonging to the botnet class.
- True Negative (TN): The number of samples predicted as benign and actually belonging to the benign class.
- False Positive (FP): The number of samples predicted as botnet but actually belonging to the benign class.
- False Negative (FN): The number of samples predicted as benign but actually belonging to the botnet class.

The recall metric refers to the proportion of correctly predicted positive samples among all the actual positive samples and is calculated as Recall = TP / (TP + FN). Thus, a high recall value indicates that the model can effectively capture positive samples.

The precision metric refers to the proportion of correctly predicted positive samples among all the predicted positive samples and is calculated as Precision = TP / (TP + FP). A high precision value indicates that the positive predictions of the model are reliable.

The accuracy metric evaluates the proportion of correctly predicted samples among all the samples and is calculated as Accuracy = (TP + TN) / (TP + TN + FP + FN). A high accuracy value indicates that the overall predictions of the model are accurate.

Finally, the F1-score combines the precision and recall metrics and is calculated as F1-score = 2 * (Precision * Recall) / (Precision + Recall). Thus, the F1-score provides a balanced measure of the model performance by simultaneously considering both the precision and the recall.

In this study, the evaluation of the proposed framework focused primarily on the F1-score since, as mentioned above, the F1-score provides a balanced measure of precision and recall, which are both crucial factors in assessing the effectiveness of the model.

D. NEURAL NETWORK ARCHITECTURE

As mentioned in section III-C, the proposed framework accommodates classifier output in the form of probability distributions for each class. The neural network architecture utilized in the subsequent experiments is detailed in this section.

A stable architecture is constructed within a fully connected neural network to validate the proposed framework. Following experimentation with various hyperparameters, encompassing the number of hidden layers, neurons, filters, sizes, optimizers, activation functions, weight initializers, and techniques to prevent overfitting, the finalized model comprises six hidden layers. Each layer consists of 4096 neurons, employing the Rectified Linear Unit (Relu) activation function [29], He Normalization [30], and Adam optimizer [31]. Additionally, three dropout layers [21] are incorporated from the bottom with a dropout rate of 0.4.

E. EXP.1 - SMALL PROPORTION OF LABELED DATA

Labeling network traffic is a high-cost activity in real-world environments. Thus, compared to unlabeled traffic, which

TABLE 7. Small proportion of labeled data test on CTU-13 dataset.

Proportion	Model	Prec.	Recall	Acc	F1	Improved
0.05	Base	72.69	97.92	80.64	83.44	3.74
	Final	79.51	96.48	85.87	87.18	
0.1	Base	74.48	96.55	81.99	84.09	7.20
	Final	88.16	94.64	91.09	91.29	
0.2	Base	80.62	95.10	85.81	87.26	4.12
	Final	88.60	94.35	90.91	91.38	
0.3	Base	80.53	95.87	86.21	87.53	4.22
	Final	89.49	94.13	91.46	91.75	
0.5	Base	80.90	96.00	86.40	87.80	4.31
	Final	89.45	94.93	91.71	92.11	
0.7	Base	87.20	94.91	90.26	90.89	1.25
	Final	89.73	94.69	91.73	92.14	

TABLE 8. Small proportion of labeled data test on N-BaIoT dataset.

Proportion	Model	Prec.	Recall	Acc	F1	Improved
0.05	Base	99.74	99.81	99.70	99.78	0.15
	Final	99.91	99.96	99.91	99.93	
0.1	Base	99.62	99.86	99.65	99.74	0.20
	Final	99.92	99.95	99.91	99.94	
0.2	Base	99.71	99.79	99.67	99.75	0.20
	Final	99.93	99.96	99.93	99.95	
0.3	Base	99.74	99.74	99.65	99.74	0.23
	Final	99.98	99.96	99.96	99.97	
0.5	Base	99.67	99.79	99.64	99.73	0.24
	Final	99.96	99.97	99.95	99.97	
0.7	Base	99.66	99.86	99.68	99.76	0.21
	Final	99.98	99.96	99.96	99.97	

may increase significantly day by day and is possibly accumulated for years, well-labeled real traffic may span a period of only a few months, especially for large companies or organizations. Thus, the performance of the proposed framework was first validated using different proportions of training data. The results presented in Table 7 indicate that the framework improved the base model performance for all considered proportions tests on the CTU-13 dataset. However, for larger quantities of labeled data (e.g., 0.7), the performance improvement was reduced because the model was already able to learn sufficient information from the training data and thus gained less benefit from the pseudo-labeled data. The result of the N-BaIoT dataset is shown in Table 8. The performance of each proportion has improved after using the proposed framework. However, the improvement is limited since the base model can already achieve 99% in each performance matrix.

F. EXP.2 - UNLABELED VARIANT OR UNSEEN BOTNETS

The second experiment validated the performance of the proposed framework in dealing with unseen or variant botnets in unlabeled data. As described above, two categories of data were collected: banking Trojans and email spam botnets. The base classifier was trained with labeled Zeus data, while the data generated by its variants or related botnets were used as unlabeled data. In the email spam category, the base classifier was trained with labeled data from Storm, and the data produced by its variants or related botnets were again considered as unlabeled data. For both categories, the data generated by the botnets were labeled as malicious in the experiments to ensure the existence of a binary class classification task.

Figure 9 presents the results obtained for the banking botnets. The proposed self-training framework improved the F1 score from 68.46% to 74.75% and the accuracy from 73.29% to 77.93%. Figure 10 presents the equivalent results for the email spam botnets. As shown, the proposed framework improved the F1-score from 50.62% to 69.36% and the accuracy from 68.44% to 77.51%.

For both sets of experiments, the performance of the model after self-training is still relatively low. This finding is reasonable because the unlabeled data are all different from the labeled data. In other words, no labeled data exist for any of the botnet variants or unseen botnets used to generate the unlabeled data, and hence, the F1-score of the base classifier following self-training using the unlabeled data is still rather modest. By contrast, for both categories of data, the F1-score of the base model for the validation data is relatively high. In other words, the base models show a good ability to detect the botnets on which they were trained. However, they have a low ability to detect variants or unseen botnets with behaviors similar to those of the labeled data. Thus, even high-performance models well-trained on labeled data are not necessarily helpful for detecting variants or unseen botnets, indicating the importance of leveraging unlabeled data.

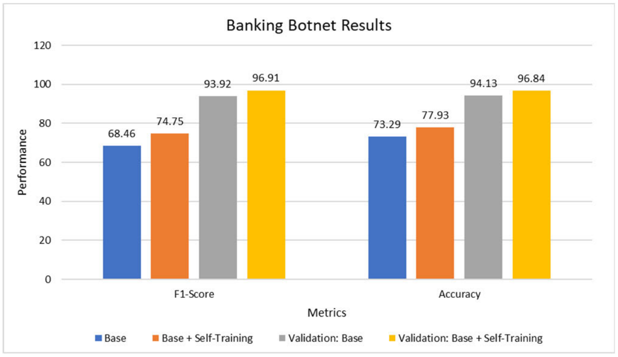

FIGURE 9. Banking botnets results.

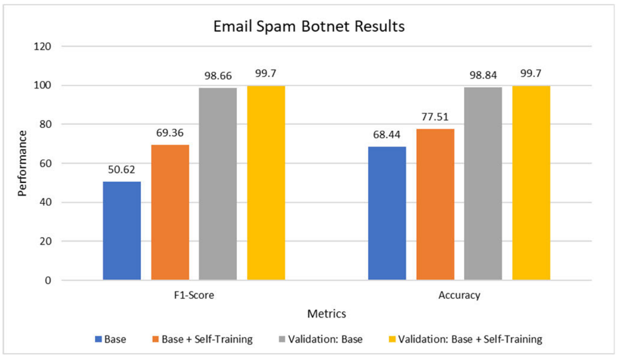

FIGURE 10. Email spam botnets results.

TABLE 9. F1-score improvement on CTU-13 dataset with different frameworks.

Proportion	Proposed	Fixed Threshold	Π-model [10]
0.05	3.74	3.22	3.67
0.1	7.20	3.44	2.99
0.2	4.12	1.83	1.68
0.3	4.22	1.67	3.44
0.5	4.31	1.11	3.94
0.7	1.25	0.88	0.67

G. EXP.3 – COMPARISON WITH RELATED WORK

The performance of the proposed framework on synthetic datasets was compared with that of a traditional self-training framework based on a fixed threshold during retraining and the semi-supervised Π-model proposed in [10]. All frameworks aim to improve the model performance using unlabeled data. Thus, the comparison experiments focused on the degree of performance improvement achieved by the frameworks rather than the exact performance of each model. The F1 score was again used as the evaluation metric for all three frameworks. The results presented in Table 9 show that the proposed framework outperformed both comparison methods for all considered values of the training data proportion. However, the performance improvement was reduced with an increasing proportion of training data because, as the amount of labeled data increased, the base model was better able to learn the characteristics of the data and hence had higher initial performance. Consequently, the self-training process had only a limited ability to further improve the performance of the model.

H. EXP.4 - REAL WORLD DATA EVALUATION

The real-world data experiments evaluated the performance of the proposed framework in a scenario in which only a small number of labeled data were available, and a data shift occurred over time. As described in Section IV-B, the dataset was partitioned into training data for the base model, unlabeled data for pre-training purposes, and testing data for performance evaluation purposes. The real-world experiments aimed to assess the ability of the proposed gradual self-training approach to leverage the information learned from the unlabeled data after undergoing a two-month data shift.

Instead of using all the unlabeled data for self-training directly, the proposed framework allows the model to gradually acquire new knowledge, thereby avoiding the significant performance degradation caused by sudden large data shifts. The essence of gradual self-training lies in the use of a gradual learning process. In particular, during each self-training iteration, a proportion of high-confidence data was selected in the range of 25–60%, where the selected proportion was increased by 5% per epoch. All of the unlabeled data were re-evaluated in each epoch by re-predicting their labels.

The results in Table 10 indicate that the proposed approach achieved a consistent performance improvement over the base model in all four weeks of July (corresponding to the test data period). The F1-score improvement varied from 3.3% to 6.91%. Overall, utilizing only one week of training data (the second week in May) and accounting for a two-month data shift, the base model achieved an F1-score of 79.43% in the whole July data. However, through the gradual self-training approach and improved pseudo-labeling method, the proposed framework improved the model performance to 83.91%.

TABLE 10. Results of obtained for one-week training data and two-month data shift.

Dataset	Model	Acc.	Prec.	Recall	F1	Improved
July Week1	Base	90.49	93.26	85.31	89.11	3.30
	Final	93.22	94.32	90.58	92.41	
July Week2	Base	84.03	87.25	80.51	83.74	3.80
	Final	87.84	91.86	83.62	87.55	
July Week3	Base	69.97	75.49	53.92	62.91	6.91
	Final	75.06	81.45	61.10	69.82	
July Week4	Base	79.88	88.62	67.61	76.70	4.25
	Final	83.34	91.97	72.29	80.95	
July All	Base	81.79	88.11	72.30	79.43	4.48
	Final	85.62	91.99	77.13	83.91	

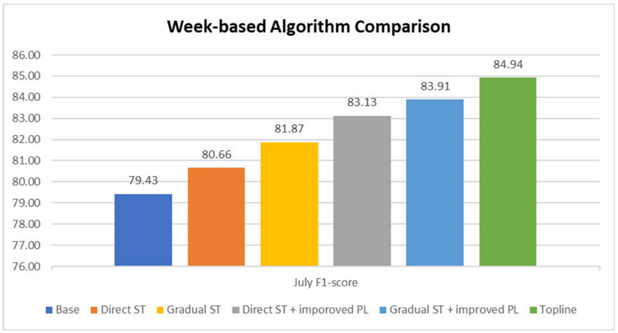

FIGURE 11. Week-based algorithm comparison.

TABLE 11. Performance detail of week-based algorithm comparison.

Model Type	Acc.	Prec.	Recall	F1
Base Model	81.79	88.11	72.30	79.43
Direct ST	82.23	85.59	76.27	80.66
Gradual ST	83.59	88.43	76.22	81.87
Direct ST + improved PL	84.80	90.29	77.02	83.13
Gradual ST + improved PL	85.62	91.99	77.13	83.91
Topline	86.04	89.26	81.02	84.94

The performance of the proposed framework was also compared with that of several other algorithms, as shown in Fig. 11 and Table 11. The Gradual Self-Training (Gradual ST) algorithm showed a 1% improvement in the F1 score compared to the Direct Self-Training method on all the unlabeled data (Direct ST). The Direct ST method is a traditional approach in which self-training on all the pseudo-labeled data is performed only once. Combining the Direct ST method with the Improved Pseudo-Labeling (Improved PL) method further improved the F1 score by 2.5%. Using the Gradual ST method, the proposed Pseudo-Labeling approach outperformed the conventional Dynamic Threshold Pseudo-Labeling method by an additional 0.8%. Overall, the results confirm the effectiveness and advantages of both the

Gradual Self-Training and the Improved Pseudo-Labeling Method.

I. EXP.5 - WEIGHT RESET MECHANISM IN REAL-WORLD DATA

The results presented in Figs. 12 and 13 provide further insights into the reasons for omitting the re-weighting technique when using real-world data. When utilizing the Gradual ST method + improved PL with the re-weighting method, the F1-score on the July dataset had a value of 82.31% and a time cost of 6339.83 s. However, when the re-weighting technique was not employed, the performance improved to 83.91%, while the time cost was reduced to 4493.95 s. In other words, the non-re-weighted approach not only yielded a better performance but also reduced the time required by nearly 30%. Consequently, in the related experiments result with improved PL, the re-weighting technique was not employed.

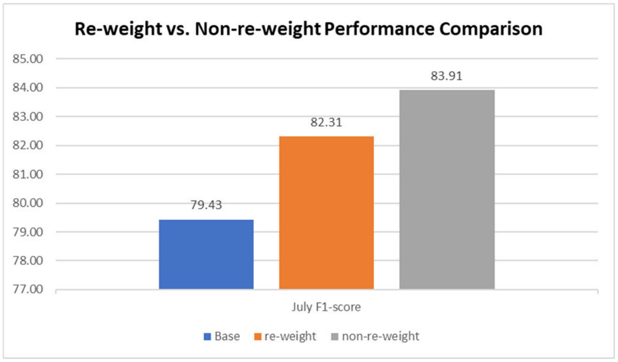

FIGURE 12. Re-weight vs. Non-re-weight performance comparison.

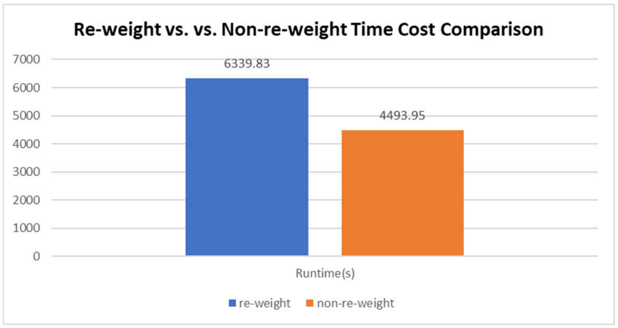

FIGURE 13. Re-weight vs. vs. Non-re-weight time cost comparison.

V. CONCLUSION

Botnets pose a significant threat to network security by enabling various malicious activities, such as DDoS attacks, spamming, and data breaches. However, detecting and mitigating these threats is challenging owing to their ever-evolving nature. Recent machine-learning-based botnet detection research has primarily focused on supervised learning methods, necessitating the extensive collection and labeling of data obtained from controlled environments. However, the collection and labeling process is extremely time-consuming, and hence, such methods are impractical for real-time detection applications, particularly when the testing data undergoes a data shift compared with the training data. The use of unlabeled network traffic can improve the performance of supervised learning methods for the detection of new botnets and/or botnet variants. However, the data shift problem, which occurs when malicious actors deliberately adapt their attack strategies to distance the new botnet data distributions from previous distributions, still poses a challenge to the model performance if the classifier is trained only one time using the unlabeled data.

To address these challenges, the present study has proposed a novel neural network-based self-training framework for botnet detection. In the proposed framework, the initial classifier is trained on labeled data and used to generate pseudo-labels from the unlabeled data. The combined dataset, consisting of both labeled data and pseudo-labeled data, is then used to iteratively refine the classifier, with the aim of improving the model performance in each successive epoch. The performance of the model is further improved through the introduction of specific strategies aimed at minimizing the number of incorrect pseudo-labels, mitigating the effects of erroneous pseudo-labels on the overall performance of the network, and determining an appropriate percentage of unlabeled data for labeling purposes.

The performance of the proposed framework has been evaluated using both public synthetic datasets and a real-world dataset. The results have shown that the framework significantly improves the base model when the training data accounts for only a low portion of the total dataset. The framework also demonstrates the ability to detect unseen botnet variants and shows a good performance in real-world campus network traffic.

In the future, the team will continue to improve the botnet detection framework performance. We will further try to combine the framework with unsupervised algorithms, which may effectively reduce the detection system's dependence on synthetic datasets.

REFERENCES

[1] R. U. Khan, X. Zhang, R. Kumar, A. Sharif, N. A. Golilarz, and M. Alazab, "An adaptive multi-layer botnet detection technique using machine learning classifiers," *Appl. Sci.*, vol. 9, no. 11, p. 2375, Jun. 2019.

[2] A. Feizollah, N. B. Anuar, R. Salleh, and F. Amalina, "A study of machine learning classifiers for anomaly-based mobile botnet detection," *Malaysian J. Comput. Sci.*, vol. 26, no. 4, pp. 1–18, 2013.

[3] A. Bansal and S. Mahapatra, "A comparative analysis of machine learning techniques for botnet detection," in *Proc. 10th Int. Conf. Secur. Inf. Netw.*, Oct. 2017, pp. 91–98.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[5] Y. Zhang, J. Niu, G. He, L. Zhu, and D. Guo, "Network intrusion detection based on active semi-supervised learning," in *Proc. 51st Annual IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops*, 2021, pp. 129–135.

[6] C.-Y. Wang, C.-L. Ou, Y.-E. Zhang, F.-M. Cho, P.-H. Chen, J.-B. Chang, and C.-K. Shieh, "BotCluster: A session-based P2P botnet clustering system on net flow," *Comput. Netw.*, vol. 145, pp. 175–189, Nov. 2018.

[7] A. Delplace, S. Hermoso, and K. Anandita, "Cyber attack detection thanks to machine learning algorithms," 2020, *arXiv:2001.06309*.

[8] C.-T. Yang, J.-C. Liu, E. Kristiani, M.-L. Liu, I. You, and G. Pau, "NetFlow monitoring and cyberattack detection using deep learning with ceph," *IEEE Access*, vol. 8, pp. 7842–7850, 2020.

[9] H.-W. Lee, N.-R. Kim, and J.-H. Lee, "Deep neural network self-training based on unsupervised learning and dropout," *Int. J. Fuzzy Log. Intell. Syst.*, vol. 17, no. 1, pp. 1–9, Mar. 2017.

[10] J. Koza, M. Krcal, and M. Holena, "Two semi-supervised approaches to malware detection with neural networks," in *Proc. ITAT*, 2020, pp. 176–185.

[11] A. Kumar, T. Ma, and P. Liang, "Understanding self-training for gradual domain adaptation," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5468–5479.

[12] S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, "Effective and efficient hybrid Android malware classification using pseudo-label stacked auto-encoder," *J. Netw. Syst. Manage.*, vol. 30, no. 1, pp. 1–34, Jan. 2022.

[13] Y. Yu, J. Long, and Z. Cai, "Network intrusion detection through stacking dilated convolutional autoencoders," in *Proc. Inf. Netw. Int. Conf.*, 2017, pp. 712–717.

[14] M. Nazemi Gelian, H. Mashayekhi, and Y. Mashayekhi, "A self-learning stream classifier for flow-based botnet detection," *Int. J. Commun. Syst.*, vol. 32, no. 16, pp. 991–1004, Nov. 2019.

[15] F. Labs. (2024). *Banking Trojans: A Reference Guide to the Malware Family Tree*. Accessed: Mar. 4, 2024. [Online]. Available: https://www.f5.com/labs/articles/education/banking-trojans-a-reference-guide-to-the-malware-family-tree

[16] W. Z. Khan, M. K. Khan, F. T. Bin Muhaya, M. Y. Aalsalem, and H.-C. Chao, "A comprehensive study of email spam botnet detection," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2271–2295, 4th Quart., 2015.

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

[18] Y. Xing, H. Shu, H. Zhao, D. Li, and L. Guo, "Survey on botnet detection techniques: Classification, methods, and evaluation," *Math. Problems Eng.*, vol. 2021, pp. 1–24, Apr. 2021.

[19] R. Vinayakumar, M. Alazab, S. Srinivasan, Q.-V. Pham, S. K. Padannayil, and K. Simran, "A visualized botnet detection system based deep learning for the Internet of Things networks of smart cities," *IEEE Trans. Ind. Appl.*, vol. 56, no. 4, pp. 4436–4456, Jul. 2020.

[20] D. Yarowsky, "Unsupervised word sense disambiguation rivaling supervised methods," in *Proc. 33rd Annu. Meeting Assoc. Comput. Linguistics*, 1995, pp. 189–196.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 56, pp. 1929–1958, 2014.

[22] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves ImageNet classification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10684–10695.

[23] R. Kozik, "Distributed system for botnet traffic analysis and anomaly detection," in *Proc. IEEE Int. Conf. Internet Things*, Jun. 2017, pp. 330–335.

[24] A. E. Medina Paredes, Y.-Y. Su, W. Wu, and H.-M. Sun, "Using unsupervised machine learning to detect peer-to-peer botnet flows," *Proc. Eng. Technol. Innov.*, vol. 3, pp. 28–30, 2016.

[25] B. Rahbarinia, R. Perdisci, A. Lanzi, and K. Li, "PeerRush: Mining for unwanted P2P traffic," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*, Jul. 2013, pp. 62–82.

[26] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Comput. Secur.*, vol. 45, pp. 100–123, Sep. 2014, doi: 10.1016/j.cose.2014.05.011.

[27] Y. Hou, S. G. Teo, Z. Chen, M. Wu, C.-K. Kwoh, and T. Truong-Huu, "Handling labeled data insufficiency: Semi-supervised learning with self-training mixup decision tree for classification of network attacking traffic," *IEEE Trans. Dependable Secur. Comput.*, early access, Aug. 1, 2022, doi: 10.1109/TDSC.2022.3195534.

[28] *Malware Capture Facility Project*. Accessed: Mar. 4, 2024. [Online]. Available: https://mcfp.weebly.com/

[29] A. F. Agarap, "Deep learning using rectified linear units (ReLU)," 2018, *arXiv:1803.08375*.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[32] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-BaIoT—Network-Based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Comput.*, vol. 17, no. 3, pp. 12–22, Jul. 2018.

[33] M. A. Setitra, I. Benkhaddra, Z. E. Abidine Bensalem, and M. Fan, "Feature modeling and dimensionality reduction to improve ML-based DDOS detection systems in SDN environment," in *Proc. 19th Int. Comput. Conf. Wavelet Act. Media Technol. Inf. Process. (ICCWAMTIP)*, Chengdu, China, Dec. 2022, pp. 1–7.

[34] M. A. Setitra, M. Fan, B. L. Y. Agbley, and Z. E. A. Bensalem, "Optimized MLP-CNN model to enhance detecting DDoS attacks in SDN environment," *Network*, vol. 3, no. 4, pp. 538–562, Dec. 2023.

[35] M. A. Setitra, M. Fan, and Z. E. A. Bensalem, "An efficient approach to detect distributed denial of service attacks for software defined Internet of Things combining autoencoder and extreme gradient boosting with feature selection and hyperparameter tuning optimization," *Trans. Emerg. Telecommun. Technol.*, vol. 34, no. 9, p. e4827, Sep. 2023.

[36] M. A. Hossain and M. S. Islam, "A novel hybrid feature selection and ensemble-based machine learning approach for botnet detection," *Sci. Rep.*, vol. 13, no. 1, p. 21207, Dec. 2023.

[37] J. Zhou, T. Hai, D. N. A. Jawawi, D. Wang, K. Lakshmanna, P. K. R. Maddikunta, and M. Iwendi, "A lightweight energy consumption ensemble-based botnet detection model for IoT/6G networks," *Sustain. Energy Technol. Assessments*, vol. 60, Dec. 2023, Art. no. 103454.

[38] J. Velasco-Mata, V. González-Castro, E. Fidalgo, and E. Alegre, "Real-time botnet detection on large network bandwidths using machine learning," *Sci. Rep.*, vol. 13, no. 1, p. 4282, Mar. 2023.

TA-CHUN LO received the B.S. degree in physics and the M.S. degree from the Institute of Computer and Communication Engineering, National Cheng Kung University, Tainan, Taiwan, in 2018 and 2020, respectively, where he is currently pursuing the Ph.D. degree with the Department of Electrical Engineering. His current research interests include botnet detection, cloud computing, big data, and artificial intelligence.

JYH-BIAU CHANG received the B.S., M.S., and Ph.D. degrees from National Cheng Kung University, in 1994, 1996, and 2005, respectively. He is currently an Associate Professor with the Department of Electronic Engineering, Lunghwa University of Science and Technology, Taoyuan, Taiwan. His research interests include botnet detection, parallel processing, and computer networks.

SHAO-HSUAN LO received the B.S. degree from the Department of Computer Science and Engineering, National Sun Yat-sen University, in 2019, and the M.S. degree from the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, in 2021. His research interests include botnet detection, big data, and artificial intelligence.

BAI-JUN KAO received the B.S. degree from the Department of Computer Science and Engineering, National Sun Yat-sen University, in 2020, and the M.S. degree from the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, in 2022. His research interests include botnet detection, big data, and artificial intelligence.

CE-KUEN SHIEH (Senior Member, IEEE) received the B.S. and Ph.D. degrees in electrical engineering from National Cheng-Kung University, Tainan, Taiwan, in 1977 and 1988, respectively.

He joined as a Faculty Member of the EE Department, National Cheng-Kung University, in 1983, and was promoted to a Full Professor, in 1998. From 1989 to 1990, he had visited the Bell Laboratory, Murry Hill, NJ, USA, as a Postdoctoral Member of Technical Staff (PMTS). He had chaired the Department of Electrical Engineering, National Cheng-Kung University, from 2002 to 2005, where he was the Director of the Computer and Network Center, from 2005 to 2011. He was also the General Director of the National Center for High-Performance Computing, National Applied Research Laboratory, Taiwan, from 2013 to 2019. His current research interests include distributed and parallel processing systems, cloud computing, network security, and big data.