DeepFW: A DNN-Based Firmware Version Identification Framework for Online IoT Devices

Zhen Lei
Taiyuan University of Technology
Taiyuan, China
leizhen0667@link.tyut.edu.cn

Nian Xue
Shandong University of Technology
Zibo, China
xuenian@sdut.edu.cn

Zhen Li
Shandong University of Technology
Zibo, China
legion@sdut.edu.cn

Dan Yu
Taiyuan University of Technology
Taiyuan, China
yudan@tyut.edu.cn

Xin Huang
Taiyuan University of Technology
Taiyuan, China
xin@huangstudio.org

Yongle Chen*
Taiyuan University of Technology
Taiyuan, China
chenyongle@tyut.edu.cn

***Abstract*—With the rapid ubiquity of Internet of Things (IoT) technology, a growing number of devices are being connected to the Internet, thereby increasing the potential for cyberattacks. For instance, due to firmware compatibility issues and release delays, N-day vulnerabilities pose significant threats to IoT devices that run outdated firmware versions. Consequently, accurately and efficiently identifying firmware versions of devices is crucial for detecting device vulnerabilities and enhancing the security of IoT ecosystems. In this work, we present *DeepFW*, which utilizes a Fusion Feature Attention Network (FFAN) to extract subtle differences in embedded web interfaces within the firmware, facilitating the identification of firmware versions in online IoT devices. To address the challenge of high similarity between versions caused by firmware homogeneity in the supply chain, we propose a novel metric loss, namely the Hard Mining Cosine Triplet-Center Loss (HCTCL), to improve intra-class compactness and inter-class separability. To validate the effectiveness of our method, we collected 4,442 firmware images and obtained 130,445 valid embedded web pages. Experimental results show that DeepFW outperforms the state-of-the-art approaches by over 25% on average in both precision and recall. Furthermore, DeepFW revealed that only 2.28% of devices in our dataset were running the latest firmware version. Our evaluation also indicates that 6,684 devices (approximately 61.26%) with outdated firmware versions remain vulnerable to known exploits.**

***Index Terms*—firmware, IoT device, deep learning, vulnerability, version identification**

I. Introduction

The Internet of Things (IoT) constitutes a global network of interconnected embedded devices capable of autonomous data collection, communication, and processing. This ecosystem includes a wide variety of physical objects, such as routers, remote monitors, and network printers. Driven by accelerated digital transformation, the IoT device population is projected to surge from 15.9 billion (2023) to over 32.1 billion by 2030 [1], a growth rate exceeding 83%. This rapid expansion, however, has been paralleled by escalating cybersecurity threats [2]–[4]. A primary contributing factor is the prevalence of outdated firmware [5], [6]. Rooted in IoT manufacturers' emphasis on time-to-market over secure development lifecycles [7], this issue results in unpatched security vulnerabilities still present in the firmware. These lingering vulnerabilities make IoT devices susceptible to unauthorized access, data breaches, and malware attacks [8]–[10].

Despite vendor countermeasures such as firmware code obfuscation and user access restrictions, N-day vulnerabilities—known flaws for which a patch is available but not yet widely deployed—persist as critical attack vectors. This limitation was catastrophically demonstrated during the 2016 Mirai botnet campaign [11], where attackers exploited *N-day* vulnerabilities in IP cameras and home routers to infect nearly 65,000 IoT devices in its first 20 hours, disrupting major internet infrastructure [12].

In this context, firmware version, which directly indicates the update status of software libraries and components on online IoT devices, becomes a critical indicator for assessing whether devices are affected by potential N-day vulnerabilities [12]–[15]. Consequently, the task of identifying firmware versions has emerged as an essential step in device security assessment, with the core security objective to enable the detection and evaluation of risks associated with known vulnerabilities in deployed IoT devices.

Current large-scale online IoT vulnerability assessment schemes largely rely on two classes of firmware identification methods. The first type of study [7], [16] exploits IoT search engines such as Shodan [17] and Censys [18] to harvest plaintext firmware version from device protocol banners and login pages. However, firmware version data are typically deemed sensitive, and manufacturers may deliberately obscure or even falsify banner content to reduce attack surface [5].

*Corresponding author.

This work was supported by the National Natural Science Foundation of China under Grant 62472302, Shanxi Scholarship Council of China 2021-038, the Fundamental Research Project of Shanxi Province No.20210302123130, the Special Project for Guiding the Transformation of Scientific and Technological Achievements of Shanxi Province No.202204021301043 and No.202304021301037, the Central Government Guiding Local Scientific and Technological Development Fund of Shanxi Province under Grant YDZJSX2024C003, and the Guangxi Natural Science Foundation under Grant 2024GXNSFBA010009.

Moreover, the infrequent update cycles of these search engine databases hinder timely reflection of newly released firmware versions [19].

The second category [20], [21] employs state-of-the-art (SOTA) firmware fingerprinting [22] for version identification. This fingerprinting method relies on hierarchical layout features, specifically the Document Object Model (DOM) structure of embedded web pages. However, this feature is derived from expert knowledge, which could introduce bias sometimes. For example, original equipment manufacturers often iterate a given device model by patching vulnerabilities or enhancing functionality at the code level [15] rather than a comprehensive UI/DOM structure redesign. The example in Figure 1 illustrates the limitations of this fingerprinting approach.

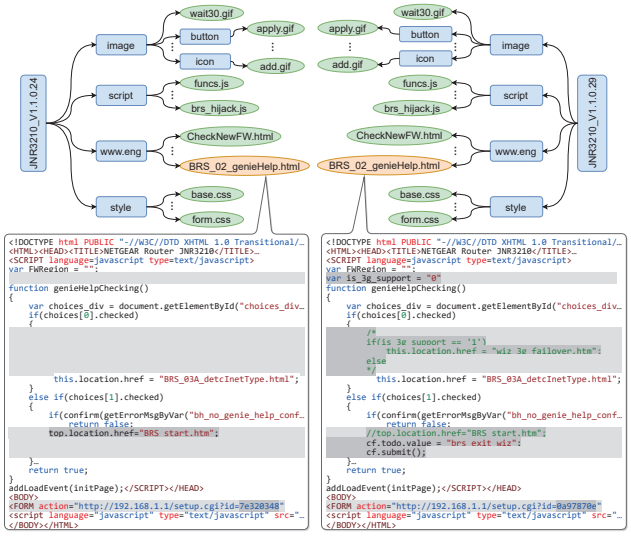


Fig. 1: Comparison of the embedded web interfaces of two JNR3210 device versions reveals identical directory structures and page DOMs, but differences in JavaScript.

Recently, deep learning has achieved remarkable success in device recognition [23]–[27] by automatically learning latent representations from vast amounts of data. The strong feature discrimination capabilities of deep learning are verified in person re-identification [28]–[30] and unmanned aerial vehicle security [31], [32], and other related fields. However, applying deep learning directly to IoT device firmware version recognition encounters several challenges: (i) Pre-trained natural language processing (NLP) models generally rely on contextual information and macro-level semantic structures, making them less sensitive to subtle differences between texts during the extraction of web features from firmware; (ii) Page reuse in firmware development results in sample feature distributions exhibiting hard samples (i.e., samples that are difficult to distinguish in the feature space owing to the high similarity between versions under the same device model), thus affecting the model's convergence; (iii) Firmware recognition models trained on simulation environment datasets require further validation to ensure their effectiveness in real-world environments.

Our Goals and Contributions. In response to these challenges and to address the limitations of existing solutions, we present *DeepFW*, the first deep learning (DL)-based framework for online IoT device firmware version identification. The framework focuses on Linux-based file system firmware, with its core idea being to achieve robust feature representation and classification by capturing the differences in embedded web interfaces within the firmware. In contrast to traditional approaches, we do not use its DOM tree derived from expert knowledge. We notice that the source code contains all semantic information of the embedded web pages, and neural networks could automatically retrieve unbiased features from them. By analyzing 4,442 collected firmware versions and 130,445 validated web pages, we found that embedded web interfaces exhibit significant differences across vendors and device models, while the variations between versions within the same device model are extremely subtle, such as script behavior adjustments and conditional statement modifications. Inspired by Han et al.'s semantic hierarchy modeling concept [33], we design the Feature Fusion Attention Network (FFAN), which employs an attention-state fusion mechanism to dynamically focus on discriminative features, amplifying fine-grained differences while preserving global page semantics. Furthermore, to further address hard samples, we introduce a novel Hard Mining Cosine Triplet-Center Loss (HCTCL) to optimize intra-class and inter-class distances, generating highly discriminative feature representations.

To validate the effectiveness of DeepFW, we have implemented a prototype system and conducted real-world experiments. The experimental results show that DeepFW surpasses the state-of-the-art firmware fingerprinting scheme, with average improvements of 26.78% in precision and 26.13% in recall. Moreover, in a large-scale Internet deployment, by leveraging the mapping between CVE [34]-listed vulnerabilities and firmware versions, DeepFW successfully identified the firmware versions of 6,684 active devices running vulnerable firmware, demonstrating its practical effectiveness.

In summary, our primary contributions are as follows:

- We present the first deep learning-based solution to automatically extract features from the embedded web interfaces of firmware, without the interference of expert knowledge. Experimental results under various settings demonstrate that it outperforms state-of-the-art solutions.
- We propose the FFAN that captures global semantics of web pages and subtle variations in reused code by adaptively learning feature weights, thereby achieving more discriminative text embedding.
- We introduce a novel HCTCL, which achieves intra-class compactness and inter-class separability of firmware versions, enhancing the learning of robust feature representations.
- We employed DeepFW to perform vulnerability analysis on IoT devices in real-world settings. Our findings revealed that only 2.28% of the devices were operating on

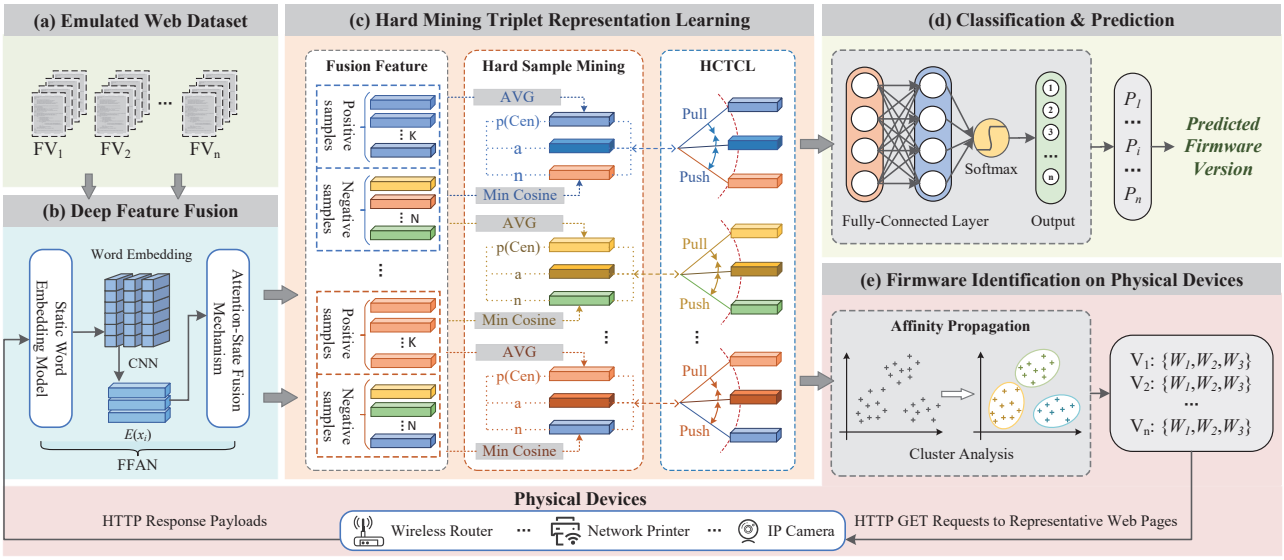


Fig. 2: The overall framework of the proposed DeepFW. Inputting web pages from different firmware versions, FFAN captures feature differences. HCTCL pulls the anchor sample (a) closer to the center of the positive class (p(Cen)) and pushes it away from the closest negative class (n). W_i denotes the representative web pages selected through cluster analysis of embeddings generated by DeepFW; these pages exhibit higher distinguishability due to their proximity to cluster centers. P_i represents the predicted probability distribution obtained when the device response is input into DeepFW.

the latest firmware version, while the overall average vulnerability rate among the identified devices was notably high at 61.26%. These results underscore the significant presence of outdated and vulnerable firmware on a large proportion of IoT devices connected to the Internet.

II. DeepFW Overview

In this section, we first provide an overview of the DeepFW framework methodology, followed by a detailed explanation of the specific modules within the framework.

A. The Overall Framework

In this paper, we propose an embedded web feature learning framework called DeepFW, as illustrated in Fig. 2. This framework utilizes the FFAN for deep fusion feature extraction from embedded pages and employs the HCTCL to optimize intra-class and inter-class distances between feature vectors, learning more robust feature representations. Specifically, we first follow the method by Li et al. [22] to simulate firmware. Once the firmware simulation is successful, we use HTTP GET requests to collect embedded web pages as inputs. Next, the FFAN captures and fuses the global semantics of embedded web pages with the local subtle differences of reused web content. Simultaneously, the constraints of the HCTCL enhance the robustness of representation learning. For classification, a fully connected neural network with a softmax function maps feature representations to labels. Finally, we apply the Affinity Propagation [35] clustering algorithm to analyze the web page embeddings generated by DeepFW. By evaluating the clustering results, we select the representative pages to query the real IoT devices and use a majority voting mechanism to determine their firmware versions.

B. Deep Feature Fusion

IoT device manufacturers categorize or package reused embedded web pages based on human-defined rules or standards in the supply chain development process, which leads to firmware homogenization. Effectively capturing the subtle differences in embedded web pages and achieving version tagging entails a challenging fine-grained text classification task.

During the feature extraction process of pre-trained NLP models, global smoothing operations may cause the loss of detailed information [36], resulting in similar output embeddings that hinder the distinction between different labels in fine-grained tasks [37]. We propose the FFAN, as illustrated in Fig. 3, to address this issue of detail loss in feature extraction. FFAN mainly consists of the following modules: 1) word embedding extraction; 2) multi-scale convolution and pooling; and 3) attention-state fusion mechanism.

Word Embedding Extraction. To ensure semantic stability during web content extraction, we use Static Word Embedding Models (SEM) such as FastText [38], and GloVe [39] to generate word embeddings for web text. We then organize these word embeddings into a word embedding matrix of size $D \times B \times S$, where D is the dimension of the word embedding vectors, S is the maximum sequence length, and each text is divided into B blocks of length S.

Multi-Scale Convolution and Pooling. For the word embedding matrix generated by SEM, we employ two-

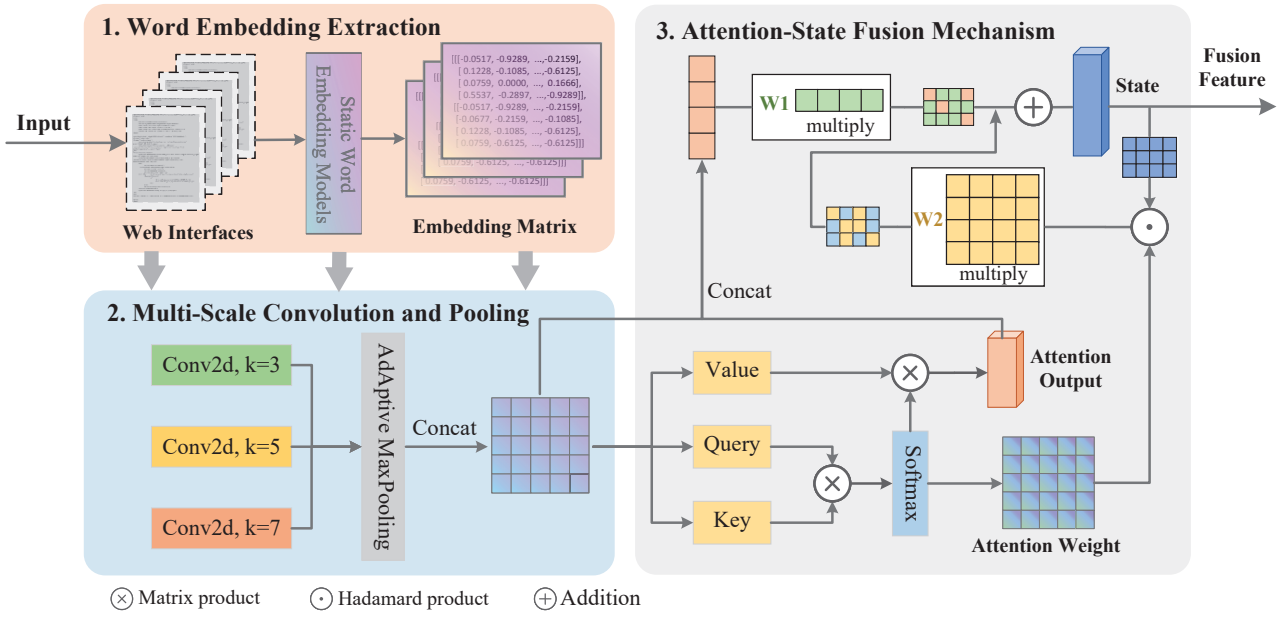


Fig. 3: The architecture of the proposed fusion feature attention network.

dimensional convolutional neural networks (2D-CNNs) with convolutional kernels of varying sizes, sliding the kernels along the sequence dimension to capture both short-range (within-line code) and long-range (cross-line logic) dependency patterns.

To avoid information distortion caused by padding or truncation due to significant length differences in web pages, we employ adaptive max pooling to highlight the most salient features in each local region. Finally, we concatenate the features from different scales to form a comprehensive feature representation vector $E(x_i)$, which is then used in the subsequent attention-state fusion mechanism.

Attention-State Fusion Mechanism. The firmware images from different manufacturers and device models exhibit significant differences in implementation methods and technology stacks. However, firmware version updates within the same device model typically involve only minor code adjustments (e.g., web-interface vulnerability patches and conditional branch modifications), with differences concentrated in limited text segments. Therefore, a firmware version analysis model should account for both global semantic integration of web pages and sensitivity to subtle local changes.

The self-attention mechanism captures cross-regional semantic associations through multi-head attention weights and effectively models global dependencies. However, due to its fixed receptive field design, it is less sensitive to small, continuous iterations, making it difficult to model the cumulative effects of progressive updates. To address this limitation, we propose the attention-state fusion mechanism, which introduces a dynamically updated state vector on top of global dependency modeling to fuse cross-region semantic information with progressively accumulated contextual information, thereby enhancing the model's global perception capability and sensitivity to gradual changes. The specific implementation process is summarized as follows.

We define the queries Q, keys K, and values V in the self-attention mechanism based on the feature representation vector $E(x_i)$ as:

$$Q = W_q E(x_i), \quad K = W_k E(x_i), \quad V = W_v E(x_i). \tag{1}$$

The attention weights A_w and output A_o are defined as:

$$A_w = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right), \quad A_o = A_w \cdot V, \tag{2}$$

where d_k is the dimension of the key vectors, used as a scaling factor to prevent weight concentration and gradient vanishing.

The state is capable of capturing all the information the model has encountered up to the current time step, dynamically updating at each step to ensure information aggregation. The state up to step $k-1$ is defined as:

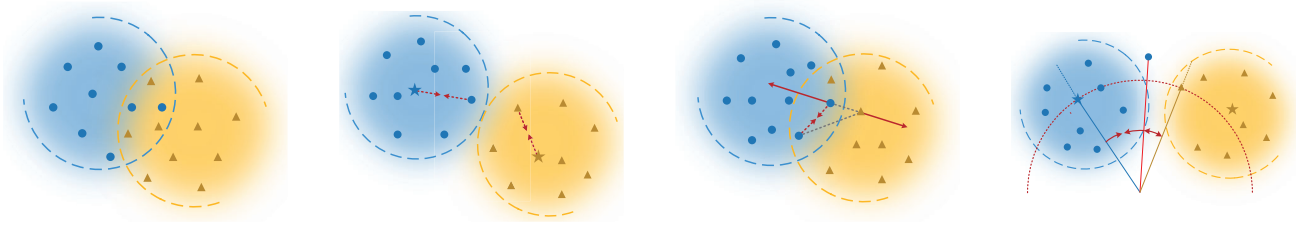
$$\text{State}_{k-1} = \sigma\left(W_1[E\left(x_i\right); A_o]\right). \tag{3}$$

The updated state utilizing the self-attention weights is:

$$\text{State}_k = \text{State}_{k-1} + \sigma\left(W_2\left(A_w \odot \text{State}_{k-1}\right)\right), \tag{4}$$

where k represents the current time step, σ denotes the ReLU activation function, and W_1 and W_2 are weight matrices for linear transformation, and $[E\left(x_i\right); A_o]$ represents the concatenation of the convolutional feature representation $E(x_i)$ and the attention output A_o.

The attention-state fusion mechanism achieves coverage of macro-level textual semantic differences across vendors and device models, while capturing micro-level variations across firmware versions of the same model. In Section III, we elaborate on the advantages of the FFAN in feature representation.



(a) Softmax loss (b) Softmax + Center loss (c) Softmax + Triplet loss (d) Softmax + HCTCL

Fig. 4: An illustration of the distributions of deeply-learned features, where points of different colors represent different versions of web features, and pentagrams indicate the centers of the classes.

C. Hard Mining Triplet Representation Learning

When training and optimizing the network solely using Softmax loss, the similarity of same-named embedded web pages from the same vendor can lead to feature space overlap across different firmware versions. The feature distribution within the same version is relatively dispersed, as shown in Fig. 4a. Additionally, web pages from real devices may exhibit minor changes compared to those in simulated environments [20]. Softmax loss alone struggles to enhance the model's robustness to these subtle variations. Inspired by triplet loss and center loss, we propose the HCTCL and use it to address intra-class compactness while maintaining inter-class separation. The HCTCL also facilitates hard sample mining and reduces training time. Before introducing the HCTCL, we review triplet loss and center loss.

Center Loss. Center loss [40] achieves intra-class compactness by learning a center for each class in the feature space and minimizing the Euclidean distance between sample features and their corresponding class centers, as shown in Fig. 4b. Specifically, the center loss is defined as follows:

$$L_c = \frac{1}{2}\sum_{i=1}^{N} D\left(f\left(x_i\right), c_{y_i}\right), \tag{5}$$

where $f(\cdot)$ denotes the feature embedding output of the neural network, $D(\cdot)$ represents the squared Euclidean distance function, x_i indicates the feature of the i-th sample, c_{y_i} signifies the center point of the class y_i to which the i-th sample belongs, and N is the total number of samples.

The update of class centers is achieved by averaging the features in each mini-batch, making this strategy susceptible to noise and leading to instability in the optimization process when using mini-batch stochastic gradient descent (SGD). Furthermore, center loss only considers intra-class distances and neglects inter-class distances. This limitation may result in the centers of different classes being too close to each other, thereby causing overlapping between different classes and making the model more susceptible to noise.

Triplet Loss. Triplet loss [41] enhances the discriminative power of a model by ensuring that the distance between similar samples is smaller than the distance between dissimilar samples, as illustrated in Fig. 4c. Specifically, triplet loss is composed of three components: an anchor, a positive sample, and a negative sample. Given an anchor sample $f(x_i^a)$, a positive sample $f(x_i^+)$ from the same class as the anchor, and a negative sample $f(x_i^-)$ from a different class, the triplet loss function is defined as:

$$L_{tpl} = \sum_{i=1}^{N}\left[D(f(x_i^a), f(x_i^+)) - D(f(x_i^a), f(x_i^-)) + m\right]_+, \tag{6}$$

where $f(\cdot)$ denotes the feature embedding output of the neural network, $D(\cdot)$ denotes the squared Euclidean distance function, $[\cdot]_+$ denotes only non-negative value is considered, N indicates the number of triplets in the training set, and m is a hyperparameter called the margin, which ensures that the distance difference between positive and negative samples is greater than m.

When using triplet loss, a large number of triplets must be constructed from the dataset. The effectiveness of selecting and generating meaningful triplets directly affects the training outcome and computational and storage costs. Additionally, triplet loss mainly focuses on the distance differences between positive and negative samples but does not guarantee compactness within the same class.

Hard Mining Cosine Triplet-Center Loss (HCTCL). Center Loss, which ignores inter-class separation constraints, and Triplet Loss, which lacks intra-class compactness optimization, are both inadequate for effectively handling reused pages in firmware development. In such scenarios, directly combining these two loss functions may lead to conflicting training objectives, potentially causing gradient directions to offset each other and thus compromising convergence stability. To address these challenges, we calculate the class center of the anchor sample and then select the negative sample that is closest to the anchor in terms of cosine distance to form new triplets. By controlling the distance between the anchor and its center to be smaller than the distance between the anchor and its nearest negative sample, we can more precisely adjust the distances of features, as shown in Fig. 4d. The formula for the HCTCL is as follows:

$$\begin{aligned} L_{HCTC} = &\sum_{i=1}^{N}\left[\mathrm{sm}(S(f(x_i^a), f(x_i^-))) - S(f(x_i^a), c_{y_i}) + m\right]_+ \\ &+ \lambda_{reg}\sum_{i} D\left(c_{y_i}^{(t+1)}, c_{y_i}^{(t)}\right), \end{aligned} \tag{7}$$

where $f(\cdot)$ denotes the feature embedding output of the neural network, $\mathrm{sm}(\cdot)$ is the softmax function, m is the margin

parameter, c_{y_i} denotes the class center of the anchor in each batch, $[\cdot]_+$ denotes only non-negative value is considered, $D(\cdot)$ denotes the squared Euclidean distance function, and λ_{reg} is the regularization factor. The function $S(\cdot)$ denotes the cosine similarity, defined as:

$$S(a,b) = \frac{a \cdot b}{\|a\|\|b\|} = \tilde{a} \cdot \tilde{b}, \tag{8}$$

where $\tilde{a}$ and $\tilde{b}$ are the normalized feature vectors.

Given that word-level vectors generated by SEM exhibit high-dimensional sparsity, cosine similarity can capture slight directional shifts in feature vectors, thereby providing better measurement of textual content similarity. Therefore, in L_{HCTC}, we use cosine distance instead of the Euclidean distance traditionally used in triplet loss. Additionally, we use softmax for mining negative samples instead of a simple max operation, which helps smooth the gradient changes and improve the stability of the gradient descent algorithm. To address the potential issue of significant drifts in class centers during mini-batch training, we introduce a regularization term to constrain the class centers, allowing the model to dynamically adapt to changes in the sample distribution.

D. Classification & Prediction

The HCTCL aims to apply metric learning directly to the learned feature representations, while softmax loss maps samples into a class-separable space. Therefore, when optimizing the feature representation using the HCTCL, we use a fully connected neural network with a softmax function as the classifier. The loss function of the classifier adopts the cross-entropy loss function, defined as:

$$L_{cls} = -\sum_{i=1}^{C} y_i \log(p_i), \tag{9}$$

where C is the number of classes, y_i is the true label, and p_i is the predicted probability of class i output by the softmax layer. The trained model ultimately predicts firmware versions by outputting the class with the highest softmax probability. Formally, the combination can be written as

$$L_{total} = L_{cls} + \lambda_{hctc} \cdot L_{HCTC}, \tag{10}$$

where λ_{hctc} is a balancing hyperparameter. We will discuss the impact of λ_{hctc} in Section III.

E. Firmware Identification on Physical Devices

In the actual identification process, a limited number of HTTP file requests are sent to the target running device, and the payload data from the device's responses is input into the model for version identification. To avoid overloading the network or being misidentified as a DDoS attack due to high-frequency requests, we perform an analysis on the embeddings generated by DeepFW using the Affinity Propagation [35] clustering algorithm. This method quantifies the similarity between samples to select representative web page samples that are closest to the cluster centers in spatial distribution, thereby significantly reducing the number of requests while maintaining prediction accuracy. After page selection, complete URLs are constructed by appending subdirectory paths and filenames of pages to the device's IP address (e.g., http://IP/webfile/a.js). Targeted HTTP GET requests are then sent to the device, and the payloads returned in the responses are finally fed into the model to achieve accurate prediction of the running firmware version.

Algorithm 1: DeepFW Framework

Input: Training samples $\{x_i\}$, initialized network parameters $\{\omega\}$, classifier parameters $\{\gamma\}$, hyperparameter λ_{hctc}, learning rate μ, and number of iterations T.
Output: Updated network parameters $\{\omega\}$ and classifier parameters $\{\gamma\}$.
while $t \leq T$ **do**
  $t \leftarrow t + 1$.
  Extract the web embedding vector $E(x_i)$ for sample x_i:

$$E(x_i) = \text{Adaptive\_MaxPool}(\text{Conv2d}_k(\text{SEM}(x_i)))_{k=3,5,7}.$$

  Compute the Q, K, and V based on $E(x_i)$ using Eq. (1).
  Calculate A_w and A_o using Eq. (2).
  Compute the final state output $\tilde{f}_i^a$ using Eqs. (3) and (4).
  Calculate the distances:

$$S_{a,c_{y_i}} \quad \text{and} \quad \text{softmax}(S_{a,-}).$$

  Compute the total loss L_{total} using Eq. (10).
  Propagate the gradient to the network parameters using the chain rule and Eq. (12):

$$\frac{\partial L_{HCTC}}{\partial \omega} = \frac{\partial L_{HCTC}}{\partial \tilde{f}_i^a} \cdot \frac{\partial \tilde{f}_i^a}{\partial \omega}.$$

  Update the fully connected layer parameters $\{\gamma\}$:

$$\gamma^{t+1} = \gamma^t - \mu \cdot \frac{\partial L_{cls}}{\partial \gamma}.$$

  Update the network parameters $\{\omega\}$:

$$\omega^{t+1} = \omega^t - \mu \left(\frac{\partial L_{cls}}{\partial \omega} + \lambda_{hctc} \cdot \frac{\partial L_{HCTC}}{\partial \omega} \right).$$

end while

F. Training and Optimization

The proposed DeepFW framework is trained and optimized by the HCTCL with softmax loss. Given a mini-batch containing M samples, the loss function L_{HCTC} can be expressed as the sum of the individual sample losses L_i:

$$L_i = \left[\phi(S_{a,-}) - S_{a,c_{y_i}} + m\right]_+ + \lambda_{reg} \textstyle\sum_i D\left(c_{y_i}^{(t+1)}, c_{y_i}^{(t)}\right), \tag{11}$$

where $S(f(x_i^a), f(x_i^-))$ is denoted as $S_{a,-}$, $S(f(x_i^a), c_{y_i})$ is denoted as $S_{a,c_{y_i}}$, and $[\cdot]_+$ denotes non-negative value. The normalized feature of $f(x_i^a)$ is represented as $\tilde{f}_i^a$, and the normalized class center c_{y_i} is represented as $\tilde{c_{y_i}}$. The symbol ϕ represents the softmax function. Suppose there exists a

function $\delta(\cdot)$ that outputs 1 when $L_i > 0$ and 0 otherwise. Then, its gradient can be derived as follows:

$$\begin{aligned}\frac{\partial L_{HCTC}}{\partial \tilde{f_i^a}} &= \frac{\partial L_i}{\partial \tilde{f_i^a}} = \left(\frac{\partial \phi(S_{a,-})}{\partial \tilde{f_i^a}} - \frac{\partial S_{a,c_{y_i}}}{\partial \tilde{f_i^a}}\right)\delta(L_i > 0) \\ &= \left(\phi(S_{a,-})(1-\phi(S_{a,-}))\tilde{f_i^-} - \tilde{c_{y_i}}\right)\delta(L_i > 0).\end{aligned} \tag{12}$$

Finally, Algorithm 1 outlines the main training process.

III. EXPERIMENTS

In this section, we first introduce the implementation details of DeepFW, followed by experiments to validate the effectiveness of our proposed DeepFW. Finally, we leverage DeepFW to identify IoT devices on the Internet that are still running outdated and vulnerable firmware versions.

A. Experimental Setup

Implementation Details. To construct a dataset for embedded web pages, we used FirmAE [42] to emulate firmware images and adopted Li et al.'s collection scheme [20], [22], sending HTTP GET requests to the successfully emulated firmware to obtain web pages. After collecting the embedded web pages, we loaded pre-trained SEM (such as FastText [38] and GloVe [39]) to generate corresponding word embedding matrices. For these word embedding matrices, we utilized a PyTorch [43]-based neural network model to achieve deep fusion feature extraction and representation.

In our experiments, we randomly selected 32 files per training mini-batch. We used three convolutional layers with kernel sizes of 3, 5, and 7 for multi-scale feature extraction, allowing the model to accommodate code changes of different lengths while enhancing fine-grained differences. The Adam optimizer was applied for faster, stable loss convergence, with initial learning rates of 3e-3 for both classification and HCTC losses. During this process, we ensured that each anchor file had negative samples (same-named files from different versions) and positive samples (class centers of the sample containing the anchor) to construct triplets for model training and validation.

All experiments were conducted on a machine running the Ubuntu 20.04.1 LTS operating system (Linux kernel 5.15.0-134-generic), equipped with an Intel® Core™ i9-10900KF CPU with 125G RAM and dual NVIDIA GeForce RTX 4090 GPUs, each with 24GB of VRAM. The software versions included Torch 2.3.1+cu121 with NVIDIA driver 550.120 supporting CUDA 12.4.

Dataset. To ensure the dataset is representative and diverse, we used automated crawler scripts to acquire 4,442 firmware images (versions) from 489 device models manufactured by 8 vendors that hold major market share in the online IoT devices market research report [44], [45], as detailed in Table I. Within our dataset, 65.6% (2,916) of the firmware images were successfully extracted, and 58.4% (1,702) of these could be successfully emulated. Among the successfully emulated firmware images, approximately 80% (1,351) yielded accessible embedded web directories. We first collected these web resources and excluded image files (e.g., $*$.png) based on file extensions, retaining only dynamic pages such as HTML, ASP, PHP, as well as JavaScript and CSS script files. Next, we removed two categories of files to eliminate invalid samples: files with identical names and content, and those with a response size smaller than 512KB (e.g., truncated responses triggered by authentication mechanisms). As a result, we obtained 130,445 valid embedded web pages to support model training, validation, and testing.

Evaluation Metrics. To address the inherent class imbalance in web page distributions across firmware versions, we employ the AUC and mAP metrics based on the One-vs-Rest (OvR) strategy for evaluation. Additionally, we compared our approach with the SOTA method using Precision and Recall metrics, which are more aligned with the practical requirements of vulnerability analysis. Specifically, we treat each version category as an independent binary classification problem (target class vs. all other classes), where the True Positive Rate (TPR), False Positive Rate (FPR), Precision (P), and Recall (R) for each class i are defined as follows:

$$\mathrm{P}_i = \frac{\mathrm{TP}_i}{\mathrm{TP}_i+\mathrm{FP}_i},\ \mathrm{R}_i = \mathrm{TPR}_i = \frac{\mathrm{TP}_i}{\mathrm{TP}_i+\mathrm{FN}_i},\ \mathrm{FPR}_i = \frac{\mathrm{FP}_i}{\mathrm{FP}_i+\mathrm{TN}_i}, \tag{13}$$

where TP_i denotes samples correctly classified as version i, FN_i represents version i samples misclassified as other versions, FP_i indicates samples erroneously classified as version i, TN_i refers to samples correctly identified as not belonging to version i. Subsequently, we compute the macro-averaged TPR, FPR, P, and R across all K firmware version categories:

$$\mathrm{P} = \frac{1}{K}\sum_{i=1}^{K}\mathrm{P}_i,\ \mathrm{R} = \mathrm{TPR} = \frac{1}{K}\sum_{i=1}^{K}\mathrm{TPR}_i,\ \mathrm{FPR} = \frac{1}{K}\sum_{i=1}^{K}\mathrm{FPR}_i. \tag{14}$$

Based on the macro-averaged TPR and FPR, AUC and mAP are defined as follows:

$$\mathrm{AUC} = \frac{1}{K}\sum_{i=1}^{K}\mathrm{AUC}_i, \quad \mathrm{mAP} = \frac{1}{K}\sum_{i=1}^{K}\mathrm{AP}_i, \tag{15}$$

where AUC_i denotes the area under the ROC curve for class i, and AP_i denotes the average precision for class i.

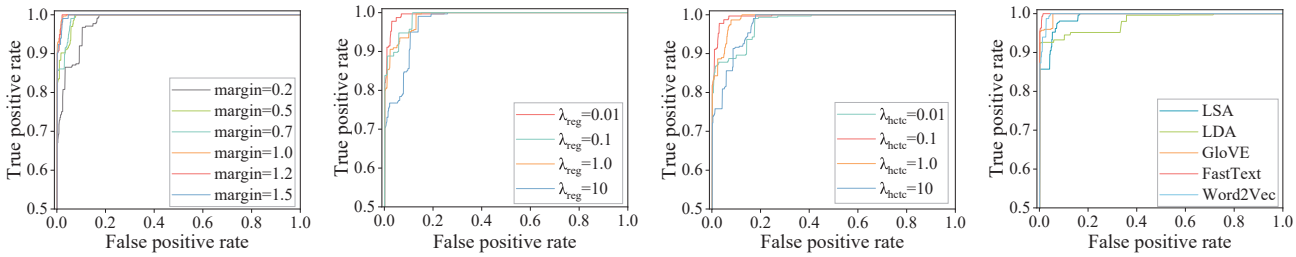
To evaluate the potential generalization capability of DeepFW in open environments, we introduce an unsupervised evaluation metric based on the geometric distribution of feature spaces: the K-Nearest Neighbor Confusion Probability (KNNCP). Its mathematical formulation is defined as:

$$\mathrm{KNNCP}(V_i) = \frac{1}{N_i K}\sum_{x\in V_i}\sum_{k=1}^{K}\mathbb{I}(y_k(x) \neq V_i), \tag{16}$$

where N_i denotes the total number of samples for version V_i, K is the predefined neighborhood size, $y_k(x)$ represents the version label of the k-th nearest neighbor for sample x, and $\mathbb{I}(\cdot)$ is the indicator function (equal to 1 if the neighbor does not belong to V_i, otherwise 0). The KNNCP metric ranges within $[0, 1]$, with lower values indicating higher clustering compactness of V_i samples in the embedding space and stronger discriminative capability of the model to distinguish V_i from heterogeneous versions.

TABLE I: Summary of vendors, firmware images, and embedded web pages.

Device Type	Vendor	Models	Download_FVs	Extracted_FVs	Emulated_FVs	mipseb	mipsel	armel	WebAcc_FVs	Valid_Webs
Gateways, routers and webcams	D-Link	95	793	546	354	142	118	94	238	29,364
	TP-Link	62	546	304	208	60	99	49	172	17,885
	Netgear	97	869	737	461	128	158	174	427	49,122
	TRENDnet	99	917	594	305	177	87	41	178	4,853
	ASUS	52	404	372	163	24	76	63	183	15,836
	Belkin	26	211	177	94	52	37	6	59	5,375
	Linksys	41	166	113	72	35	18	19	67	7,178
	Zyxel	17	536	70	45	10	23	12	27	832
Total		**489**	**4,442**	**2,916**	**1,702**	**628**	**616**	**458**	**1,351**	**130,445**



(a) ROC versus margin. (b) ROC versus λ_{reg}. (c) ROC versus λ_{hctc}. (d) ROC versus SEMs.

Fig. 5: ROC curves for different hyperparameters.

Baselines. We select the only existing state-of-the-art method specifically designed for firmware version identification, proposed by Li et al. [20], as the baseline for comparison. This method extracts the DOM tree structure from embedded web pages and performs a pre-order traversal to generate a node sequence as the firmware fingerprint. During the matching phase, it uses the longest matching state subsequence to compute similarity and determine the firmware version. We choose this method as the baseline because it represents the most relevant and representative work in this task domain to date.

B. Parameter Settings

We focus on the regularization parameter λ_{reg}, the balance parameter λ_{hctc}, the margin parameter, and the SEMs. Fig. 5 presents the associated ROC curves.

Margin. We conducted experiments using the HCTCL as the supervised loss function, and the results for different margins are depicted in Fig. 5a. Since we used cosine distance, the margin values are in radians. The HCTCL converges within the margin range of 0.2 ($\approx 11°$) to 1.5 ($\approx 86°$). The experimental results show that as the margin increases, the model's AUC gradually improves and peaking at a margin of 1. When the margin is too small, the model struggles to effectively distinguish between different class samples; conversely, an excessively large margin overly emphasizes inter-class distance while neglecting intra-class compactness. Ultimately, DeepFW peaks at an mAP of 95.94% with a margin of 1.0.

Regularization Hyperparameter λ_{reg}. Experiments were conducted with a margin of 1.0, and the results for different λ_{reg} values are shown in Fig. 5b. The experiments indicate that when λ_{reg} is small, the class center update has more freedom, which is beneficial for optimizing classification performance. However, as λ_{reg} increases, the regularization constraint on the class center becomes stronger, limiting the model's optimization capability and leading to a decrease in AUC. When λ_{reg} is too large, the class center update is almost fixed, resulting in poor performance. To balance classification performance and training stability, we selected λ_{reg} = 0.01 as a good trade-off.

Balance Hyperparameter λ_{hctc}. We conducted experiments with a margin of 1.0 and λ_{reg} of 0.01. Fig. 5c shows the results for different λ_{hctc} values. Both excessively small and large λ_{hctc} values lead to performance degradation, suggesting the need to balance classification loss with the optimization of intra-class and inter-class distances. The highest AUC was achieved when λ_{hctc} = 0.1, confirming that a moderate λ_{hctc} can effectively manage the trade-off between the HCTCL and the classification loss. The optimal λ_{hctc} value was found to be 0.1, with a mAP of 96.52%.

Static Word Embedding Models. To evaluate the performance of different SEMs, we compared the ROC curves and AUC values of five models. As shown in Fig. 5d, we found that the FastText and GloVe models outperform LSA and LDA. Although our task involves fine-grained text classification, contextual information remains important. FastText and GloVe models can distinguish similar text fragments that may appear in different versions, whereas LSA and LDA are less sensitive to contextual information. Hence, we selected GloVe to generate the word embedding vectors.

C. Experimental Results and Analysis

1) Comparison with state-of-the-art methods: In real-world firmware version identification scenarios, the number of ver-

sions to be recognized typically far exceeds that of controlled experimental settings. To assess the impact of increasing version counts on identification performance, we conducted evaluations under progressively larger version scales (8, 16, 32, 64, 128, 256, 512). As shown in Table II, the DeepFW consistently outperforms the baseline across all scales, achieving average improvements of 26.78% in precision and 26.13% in recall. These results indicate that structural syntactic features of web pages [20] are effective in firmware distributed across multiple vendors and device models. However, as the dataset scales up and the number of versions under the same device model increases, the lack of semantic information leads to a significant performance drop. In contrast, DeepFW enhances identification performance by employing an unbiased feature extraction framework that integrates both global semantics and fine-grained variations.

TABLE II: Comparison with the state-of-the-art method by Li et al. under different version scales. P means Precision, R means Recall, F1 means F1 score.

Num_FVs	Li et al. (SOTA)			DeepFW		
	P(%)	R(%)	F1	P(%)	R(%)	F1
V=8	88.26	90.32	88.56	96.55	97.39	96.97
V=16	53.16	58.78	54.05	94.09	93.40	93.43
V=32	43.84	51.73	45.45	77.86	78.86	77.39
V=64	35.83	40.30	35.63	73.04	72.82	71.92
V=128	23.61	32.64	24.96	49.17	55.78	50.45
V=256	14.55	16.80	14.20	39.65	53.92	43.58
V=512	8.10	11.15	8.40	23.76	32.43	25.43
Avg.	**38.19**	**43.10**	**38.75**	**64.87**	**69.23**	**65.59**

2) Generalization Capability Evaluation: In the closed-set training paradigm, the model inherently fails to recognize firmware versions that were not included in its training. This limitation arises from the inability of supervised learning models to make inferences beyond the class boundaries they have learned. This represents a common challenge in deep learning-based firmware analysis, and we leave this issue to be addressed in future work. To evaluate the generalization capability of DeepFW in open-set scenarios, we curated a dataset comprising 36 unseen firmware versions across three representative scenarios:

- **Intra-model versions:** 12 iterative releases of the Netgear R7000 model (version range: V1.0.3.56–V1.0.7.6), were selected to evaluate sensitivity to incremental version updates within the same model.
- **Inter-model versions:** 12 cross-model variants from Netgear (including R6300, R6700, etc.), were used to evaluate the model's ability to capture firmware differences across distinct models under the same vendor.
- **Cross-vendor versions:** 12 firmware versions from other vendors (e.g., TP-Link, D-Link), were included to examine the generalization capability across diverse firmware ecosystems.

After reproducing Li et al.'s method to extract DOM tree node lists, we encoded these lists into fixed-dimensional embeddings using a bag-of-words model to enable K-nearest neighbor computation. To eliminate interference from multi-version cross-comparisons, we selected the R7000-V1.0.4.18 firmware version as the test target. As shown in Table III, DeepFW significantly outperforms the state-of-the-art method by Li et al. across all evaluation scenarios.

In intra-model comparisons, DeepFW reduces the average version confusion score from 98.1 to 32.5—a 66.9% decrease—demonstrating the effectiveness of the joint optimization of the FFAN and the HCTCL in capturing fine-grained semantic differences between closely related firmware versions. In inter-model comparisons, although Li et al.'s DOM-based method benefits from increased structural diversity across models, it still misclassifies 17% of web pages, primarily due to shared UI templates and inherited DOM structures within the same vendor. Such structural redundancy limits the model's ability to learn discriminative features. In contrast, DeepFW's attention-state fusion mechanism adaptively integrates content, structure, and code-level features, enabling more precise discrimination even among structurally similar interfaces. In cross-vendor comparisons, both methods exhibit low confusion rates due to substantial differences in layout and logic across vendors. However, DeepFW continues to outperform by leveraging feature fusion, reinforcing its robustness in both fine-grained and broad-spectrum version identification tasks.

3) Further Ablation Analysis and Discussion: We further evaluated the performance of DeepFW in two aspects: the FFAN and the HCTCL.

Effect of the Internal Structure of the FFAN. To assess the influence of the internal structure design of the FFAN in DeepFW on the overall model performance, we conducted experiments on different structural combinations of the FFAN and evaluated them using mAP and AUC values. The specific results are shown in Table IV and Fig. 6. The SEM represents the static word embedding base model, while AVG, SUM, and MAX represent different pooling strategies applied on the SEM. CNN indicates the addition of multi-scale 2D convolutional layers, with MaxPooling and AvgPooling specifying the max pooling and average pooling types used in CNN, respectively. AM and SM denote the self-attention mechanism and state update mechanism, respectively, and ASM represents the proposed attention-state fusion mechanism.

As shown in Table IV and Fig. 6, the pooling strategies in the SEM fail to adequately capture complex features, resulting in mAP values below 50%. The SUM strategy performs the worst, with an mAP of only 13.77%, indicating that simple sum pooling loses a significant amount of information. Introducing multi-scale convolutional layers improves the model's mAP by approximately 40%, demonstrating that multi-scale convolutional layers can capture subtle differences between texts. The individual introduction of the self-attention mechanism and the state update mechanism further enhances the model's performance, increasing mAP by approximately 10%

TABLE III: Comparative analysis of Li et al.'s method and DeepFW on the KNNCP values for unseen versions across test subsets. Li et al.'s method was evaluated under the Mean KNNCP (M-KNNCP) metric. K denotes the pre-defined neighborhood size. The $*$ serves as a wildcard character, indicating that all versions within the current scope are referenced. **Note that lower values indicate a higher version discrimination capability.**

	Version_Subset	M-KNNCP		K=1(%)	K=5(%)	K=10(%)	K=15(%)
		Li et al. (SOTA)	DeepFW	DeepFW			
Intra-model	(R7000-V1.0.4.18|R7000-V1.0.4.28)	95.9	**29.8**	31.7	28.6	29.3	29.6
	(R7000-V1.0.4.18|R7000-V1.0.3.68)	86.1	**21.5**	23.2	21.2	21.0	20.5
	(R7000-V1.0.4.18|R7000-*)	98.1	**32.5**	31.7	32.6	32.9	33.0
Inter-model	(R7000-V1.0.4.18|WNDR3400v2-V1.0.0.54)	14.2	**8.6**	6.3	9.5	9.5	9.2
	(R7000-V1.0.4.18|EX7000-V1.0.0.66)	10.9	**7.7**	5.4	7.8	8.8	8.8
	(R7000-V1.0.4.18|other-model*)	17.0	**10.3**	7.1	10.9	11.5	11.6
Cross-vendor	(R7000-V1.0.4.18|FW_EA6200_1.1.41.162599)	5.8	**2.5**	1.8	2.2	2.7	3.3
	(R7000-V1.0.4.18|Archer_C5_3-17-6)	1.1	**0.4**	0.9	0.4	0.3	0.3
	(R7000-V1.0.4.18|other-vendors*)	6.0	**4.5**	3.6	4.1	4.7	5.5

TABLE IV: Performance of FFAN settings.

Model	mAP	AUC
SEM+AVG	40.68	83.37
SEM+SUM	13.77	52.69
SEM+MAX	39.41	81.81
SEM+CNN+MaxPooling	81.40	87.73
SEM+CNN+AvgPooling	74.21	83.50
SEM+CNN+MaxPooling+AM	90.70	92.51
SEM+CNN+MaxPooling+SM	88.16	90.71
SEM+CNN+MaxPooling+ASM	**96.83**	**98.14**

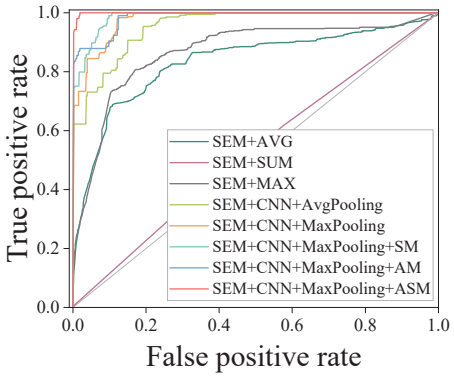


Fig. 6: Ablation study on different FFAN architectures.

TABLE V: Performance of the HCTCL settings.

Loss	mAP	AUC
SEM	8.26	55.21
SEM+Triplet Loss	32.42	58.06
SEM+Center Loss	34.96	82.28
SEM+HCTCL	39.19	86.56
FFAN	85.84	91.64
FFAN+Triplet Loss	86.26	93.29
FFAN+Center Loss	89.91	93.16
FFAN+HCTCL	**96.83**	**98.14**

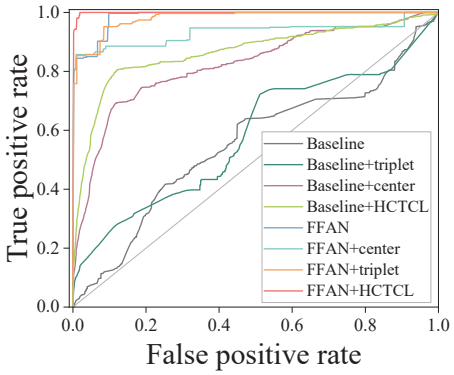
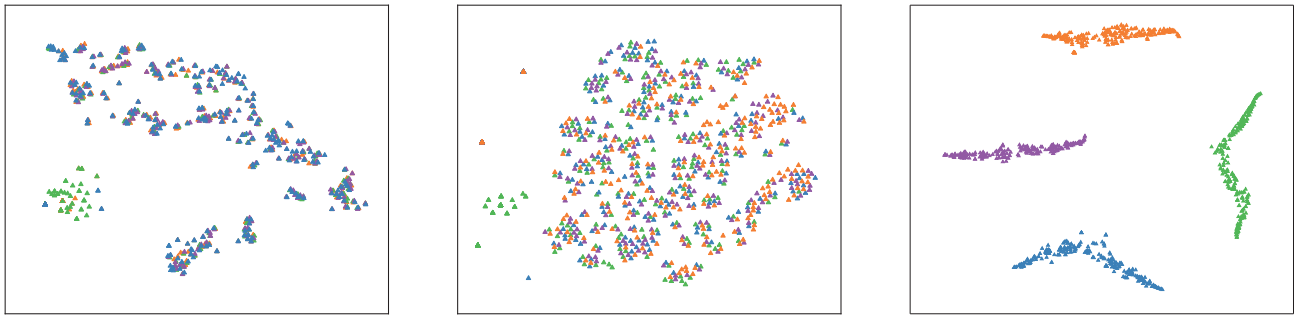


Fig. 7: Ablation study on different loss functions.

and AUC by about 5%. Finally, the attention-state mechanism shows the best performance, achieving mAP values above 96% and an AUC of 98.14. The experimental results indicate that the proposed FFAN effectively captures and amplifies subtle differences by simultaneously utilizing dynamic feature adjustment and state maintenance, further benefiting fine-grained text classification tasks.

Effect of the Hard Mining Cosine Triplet-Center Loss. To evaluate the impact of the HCTCL on the overall model performance in DeepFW, we assessed different variants of the proposed method to illustrate the key role of the HCTCL. The experimental results are shown in Table V and Fig. 7. The SEM refers to using a static word embedding model without the FFAN and training with softmax loss, meaning no deep feature fusion extraction. The FFAN denotes the feature fusion attention network model trained solely with softmax loss. We experimented with three loss functions (i. e., triplet loss, center loss, and the HCTCL) to combine these two network structures into different versions. Among them, the FFAN+HCTCL is the complete version combining the FFAN with the HCTCL and softmax loss.

As shown in Table V and Fig. 7, compared to the SEM, the FFAN significantly improves performance, with mAP and



(a) The original (b) With FFAN (c) FFAN+HCTCL

Fig. 8: Scatter plots for feature representations with labels of the original, FFAN, and FFAN+HCTCL. Different colors denote different labels (or classes).

AUC increasing by 77.58% and 36.43%, respectively, strongly demonstrating the effectiveness of the FFAN. To further prove the effectiveness of the HCTCL, we compared it with traditional triplet loss and center loss. The experimental results indicate that due to the limited extraction of fine-grained features by the SEM, even the introduction of advanced loss functions results in relatively limited performance improvement. Within the FFAN architecture, the HCTCL demonstrated superior performance, outperforming the second-best method (FFAN+center loss) by 6.92% in mAP and 4.98% in AUC. In summary, the proposed HCTCL is beneficial for model training. The HCTCL effectively optimizes inter-class distance while enhancing intra-class compactness by mining hard training samples.

Visualization of Learned Representations. We utilized t-SNE [46] to visualize the deep fusion features of four versions of samples under the same device model. As shown in Fig. 8a, the original sample distribution shows a large overlap because of the similarity of same-named files across different versions. Fig. 8b displays the sample distribution using the FFAN, where the FFAN disperses the samples of different categories, enhancing their separability. Fig. 8c shows the sample distribution using the FFAN + HCTCL, where the HCTCL pulls samples of the same category to cluster together while pushing samples of different categories apart. This ensures the combination of the FFAN and the HCTCL achieves better feature representation for embedded web pages.

D. Vulnerability Evaluation

To validate the applicability of DeepFW in real-world environments, we calculated two evaluation metrics during the clustering analysis: the Silhouette Score [47] and the Davies-Bouldin Index [48]. We then derived a comprehensive score by dividing the Silhouette Score by the Davies-Bouldin Index. This score reflects the proximity to the cluster centroid, where higher-scoring pages exhibit stronger robustness in firmware version identification. To improve identification efficiency, we first selected web pages that are consistently present across all firmware versions of the same device model. These pages were then ranked based on their composite scores, and the top three were selected as representative pages. The representative page scores for certain versions are shown in Table VI. During the actual request process, only these representative pages were queried, rather than employing random or exhaustive requests. This approach achieves an effective balance between accuracy and efficiency.

TABLE VI: Summary of representative page selection.

Vendor	Model	Versions	Representative pages	Score
D-link	DIR-300	6	tools_ddns.php	2.27
			adv_network.php	2.22
			st_stats.php	2.21
	DIR-816L	5	index.php	3.59
			spt_tools.php	3.59
			spt_setup.php	3.58
TP-link	Archer C5	5	NoipDdnsRpm.html	4.51
			IPv6StatusRpm.html	4.51
			WanStaticIpV6CfgRpm.html	4.50
Linksys	EA6700	5	welcome.html	7.41
			main.html	7.21
			check_internet.html	2.84
Netgear	JNR3210	7	WLG_wds.htm	1.58
			LAN_reserv_del_nonselect.htm	1.57
			FW_forward.htm	1.57

Our goal is to use DeepFW to identify IoT devices still running outdated and vulnerable versions. However, with nearly 4 billion addresses in the IPv4 space, it is crucial to reduce the identification time for large-scale IoT devices. We filtered out three types of IoT devices—wireless routers, IP cameras and network printers——using the Censys API. Simultaneously, we used ZMAP [49] to send stateless TCP-SYN packets to each IP address to check if these devices were still active. If a device responded, we added it to the active list. For devices on the active list, we sent the three representative pages selected for each device model and determined the version by using the voting mechanism on the results of identification. Most existing vulnerability databases (e.g., NVD) explicitly indicate the firmware versions initially impacted by a Common Vulnerabilities and Exposures (CVE) and the version where the CVE is fixed. If a device's firmware version is lower than the CVE-listed patch version, it can be reasonably inferred that the device may be in a vulnerable state. For the identified IoT device firmware versions, we analyzed the adoption rate of the

TABLE VII: The version and vulnerability evaluation of real-world online IoT devices.

Vender	Model	Firmware	Devices	Devices with latest version	Proportion with latest version	N-days Vulnerabilities	Severity	Vulnerable Devices	
								Amount	Vulnerable Rate
D-Link	DCS-933L	7	542	16	2.95%	CVE-2019-10999	High	461	85.06%
						CVE-2017-7852	High	424	78.23%
	DIR-815	10	464	9	1.94%	CVE-2023-37758	High	140	30.17%
						CVE-2019-18852	Critical	207	44.61%
	DIR-850L	12	737	23	3.12%	CVE-2023-49004	Critical	141	19.13%
						CVE-2017-3193	High	203	27.54%
						CVE-2016-5681	Critical	416	56.45%
TP-Link	Archer C50	6	592	18	3.04%	CVE-2023-31188	High	363	61.32%
						CVE-2023-0936	Medium	240	40.54%
	TL-WR841N	11	673	14	2.08%	CVE-2023-39745	High	432	64.19%
						CVE-2022-46912	High	467	69.39%
Netgear	R7000	13	602	20	3.32%	CVE-2021-45647	High	523	86.88%
						CVE-2020-35796	Critical	499	82.89%
						CVE-2019-20730	Critical	442	73.42%
	R7800	15	513	14	2.73%	CVE-2018-21227	Medium	416	69.10%
						CVE-2021-45619	Critical	433	84.41%
						CVE-2021-29068	High	417	81.29%
	DGN2200	8	491	9	1.83%	CVE-2019-20754	Medium	376	76.58%
						CVE-2019-17373	Critical	421	85.74%
						CVE-2016-5649	Critical	160	32.59%
Asus	RT-AC66U	9	537	17	3.17%	CVE-2023-39086	High	134	24.95%
						CVE-2017-12754	High	478	89.01%
						CVE-2013-6343	High	146	27.19%
	RT-AC87U	10	624	13	2.08%	CVE-2014-7270	Medium	509	81.57%
						CVE-2018-0581	Medium	516	82.69%
Zyxel	NBG-418N	6	221	3	1.35%	CVE-2023-22924	Medium	171	77.37%
						CVE-2019-6710	High	94	42.53%
Linksys	E5350	3	355	3	0.85%	CVE-2022-35572	High	272	76.62%
	AX3200	7	498	6	1.20%	CVE-2022-38841	High	125	25.10%
AVG	**-**	**9**	**526.84**	**12.69**	**2.28%**	**-**	**-**	**331.9**	**61.26%**

latest firmware versions and quantified the vulnerability rate of these devices using data from the CVE website, as shown in Table VII.

From the experimental results, we found that, on average, only 2.28% of the devices were operating with the latest firmware version, indicating that users are often reluctant or slow to apply available firmware updates. A substantial proportion of devices running outdated firmware remain exposed to N-day vulnerabilities, with an average vulnerability rate of 61.26%. These findings reveal a pronounced lag in firmware update practices across IoT deployments. Since outdated firmware versions often contain known security flaws, these devices are highly susceptible to attacks, thereby increasing cybersecurity risks.

Notably, our research focuses on identifying the firmware versions of online devices, and the results of the potential vulnerability assessment provide important case support for the practical value of DeepFW. Although further vulnerability verification and in-depth exploration are beyond the core scope of this study, this does not diminish the value and significance of conducting an initial assessment of potential vulnerabilities in IoT devices.

E. Limitation and Discussion

In this study, we focus on training DeepFW on Linux-based file systems, but the proposed technique can also be applied to other platforms implementing web interfaces. Our research targets the most prevalent and widely distributed in today's online IoT devices [50], including wireless routers, IP cameras, network printers, as well as some mining devices, industrial control systems, and medical devices. Furthermore, DeepFW's accuracy relies on the effectiveness of firmware emulation, and this design consideration aims at mitigating the significant discrepancy between real-world pages and training samples caused by dynamic page generation mechanisms. It is inevitable that DeepFW, as any other firmware identification approach, can only recognize firmware with differences on web pages. For cases where the web source code does not update with firmware changes (accounting for approximately 16.3% in our dataset), such as binary component-level vulnerability patches, DeepFW can still map them to the feature space of the nearest adjacent version. This degree of misclassification still holds some validity for device vulnerability analysis and security management.

To assess firmware vulnerabilities in a real-world internet environment, we utilized TCP-SYN packet transmission to verify device availability. This non-intrusive network probing technique, which employs stateless TCP-SYN packets, does not access internal device data or alter device states. Additionally, when collecting web pages from devices, we did not attempt to bypass login mechanisms or escalate privileges to access restricted content. Our requests were confined to publicly accessible pages that required no login credentials. To mitigate the impact on network traffic and device performance, we only used filtered representative pages for device requests. This ensured that no unnecessary load was placed on devices or networks. Considering the sensitivity of the data from online devices, we provide a simplified dataset, pre-trained

models, and the code, available at https://github.com/LeiZhen0667/DeepFW.

IV. Related Work

This section briefly reviews previous work related to the identification of IoT device firmware and metric learning.

Offline Identification of IoT Device Firmware. The offline identification method is based on analyzing the firmware itself, extracting fundamental properties and features of the firmware, aiming to facilitate subsequent firmware analysis, without considering the application relationship between devices and firmware. Lee et al. [51] proposed an offline firmware analysis framework capable of distinguishing functional modules within compressed firmware binaries through static analysis. Zhang et al. [52] introduced a graph neural network (GNN)-based approach to model firmware file dependencies for file type and version identification. Zhang et al. [53] further developed PANDORA, a static analysis tool that identifies firmware versions by scanning binary files for version-specific signatures. While these offline methods are valuable for localized vulnerability analysis (e.g., reverse engineering in controlled environments), they are inherently limited in supporting real-time threat detection and global risk assessment for IoT ecosystems due to their reliance on static firmware binaries and lack of runtime context awareness.

Online Identification of IoT Device Firmware. Online identification of IoT device firmware refers to recognizing firmware information (such as the device vendor, model, and firmware version) for devices connected to the Internet. Firmware version is the most granular information in device identification and is directly related to security vulnerabilities. Traditional methods, such as those by Costin et al. [54], demonstrated that the HTML content of embedded web pages is the most significant and effective feature for device fingerprinting. However, they did not claim to be able to accurately identify the firmware version of a device. Furthermore, with the introduction of dynamic page technologies by vendors, the constant updating of web interfaces during operation renders the static fingerprinting method based on cryptographic hash digests proposed by Costin et al. ineffective. Li et al. [22] proposed extracting the DOM of embedded web pages as firmware fingerprints and used firmware emulation to eliminate inconsistencies between firmware fingerprints and local file systems. Since vendors typically focus on functional improvements and vulnerability fixes when updating firmware, without frequently altering the DOM structure of the web interface, this method overlooks semantic information in the source code, limiting its effectiveness in fine-grained firmware identification, as demonstrated by the phenomenon shown in Fig. 1. Censys [18] and Shodan [17], as IoT search engines, can discover Internet devices and web services. However, the banner information they rely on does not always include device model and/or firmware version identifiers. In comparison, our work leverages natural language processing techniques and constructs deep learning models to extract both macro and micro information from embedded web documents, achieving higher recognition rates and efficiency.

Metric Loss Function. Metric learning enhances model discrimination by optimizing specific distance or similarity metrics to represent the similarities and differences between samples. To address the issue of sample feature separation, Wen et al. [40] first introduced the concept of center loss. This method enhances intra-class compactness by constraining the feature vectors of the same class to converge towards their class center. Although center loss resolves the issue of intra-class compactness, it does not effectively increase inter-class distance, thereby impacting classification performance. Schroff et al. [41] introduced the triplet loss framework, which constructs triplets (anchor, positive, negative) to simultaneously minimize the distance between similar samples and maximize the distance between different samples. This approach effectively increases inter-class distances. However, triplet loss faces two main issues: it does not explicitly optimize intra-class distances, and the triplet data grows exponentially due to negative sampling, which increases computational costs. To further improve upon these methods, He et al. [55] proposed a loss function for 3D object retrieval known as Triplet-Center Loss (TCL). This loss function ensures that samples are close to their class centers while being distant from the nearest negative centers. Additionally, Hu et al. [56] proposed a metric learning loss function called triplet-batch-center loss (TBCL). This approach requires samples in each batch to be close to their class center while being far from other centers, thereby enhancing person ReID accuracy. Inspired by these, we designed the HCTCL. This loss function jointly optimizes intra-class compactness and inter-class separability, enhancing the model's recognition performance and feature discriminability.

V. Conclusion

This paper presents DeepFW, the first deep learning-based framework for identifying firmware versions of IoT devices. Our approach addresses the critical need for accurately identifying firmware versions, which is essential for assessing and mitigating N-day vulnerabilities in IoT devices. By automatically extracting features from embedded web interfaces, DeepFW achieves robust feature representation and classification, outperforming existing state-of-the-art methods by over 25% on average in terms of both precision and recall. Our real-world analysis reveals a concerning trend: only 2.28% of IoT devices are running the latest firmware, and an average of 61.26% of devices are in a vulnerable state. This finding underscores the urgent need to improve firmware management and security practices within the IoT ecosystem. Furthermore, we recommend urging device vendors to take greater responsibility by designing more automated and user-friendly update mechanisms to ensure device security patches promptly. Future work will include a focus on expanding the system's capabilities to covering a broader range of IoT devices for large-scale vulnerability analysis and enhancing its adaptability to the evolving IoT environment.

ACKNOWLEDGMENT

We thank Yijia Li for her help in dataset preparation and analysis. We also appreciate the valuable comments and suggestions provided by the anonymous reviewers.

REFERENCES

[1] Statista. (2023) Internet of Things (IoT) connected devices worldwide 2019-2030. Accessed: 2024-07-01. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez, "A Survey of IoT-Enabled Cyberattacks: Assessing Attack Paths to Critical Infrastructures and Services," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3453–3495, 2018.

[3] N. Xue, D. Guo, J. Zhang, J. Xin, Z. Li, and X. Huang, "Openfunction for Software Defined IoT," in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*. IEEE, 2021, pp. 1–8.

[4] N. Xue, J. Zhang, Z. Li, X. Hong, H. Tang, and X. Huang, "SFIOT: Software-Defined Function for the IoT," in *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2022, pp. 20–30.

[5] A. Costin, A. Zarras, and A. Francillon, "Automated Dynamic Firmware Analysis at Scale: A Case Study on Embedded Web Interfaces," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 437–448.

[6] StarHub Singapore. (2023) Cyber Security Threats In The Age Of IoT. Accessed: 2024-08-01. [Online]. Available: https://www.starhub.com/business/resources/blog/cybersecurity-threats-in-the-age-of-iot.html

[7] F. Ebbers, "A Large-Scale Analysis of IoT Firmware Version Distribution in the Wild," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 816–830, 2022.

[8] A. Mangino, M. S. Pour, and E. Bou-Harb, "Internet-scale Insecurity of Consumer Internet of Things: An Empirical Measurements Perspective," *ACM Transactions on Management Information Systems (TMIS)*, vol. 11, no. 4, pp. 1–24, 2020.

[9] L. Zhou, C. Su, Z. Li, Z. Liu, and G. P. Hancke, "Automatic fine-grained access control in scada by machine learning," *Future Generation Computer Systems*, vol. 93, pp. 548–559, 2019.

[10] I. Makhdoom, M. Abolhasan, J. Lipman, R. P. Liu, and W. Ni, "Anatomy of Threats to the Internet of Things," *IEEE communications surveys & tutorials*, vol. 21, no. 2, pp. 1636–1675, 2018.

[11] D.B.B.Herzberg and I.Zeifman. (2016) Breaking Down Mirai: An IoT DDoS Botnet Analysis. Accessed: 2025-08-01. [Online]. Available: https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html

[12] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the Mirai Botnet," in *26th USENIX security symposium (USENIX Security 17)*, 2017, pp. 1093–1110.

[13] F. Hu, *Security and Privacy in Internet of Things (IoTs): Models, Algorithms, and Implementations*. CRC Press, 2016.

[14] M. A. Khan and K. Salah, "IoT security: Review, blockchain solutions, and open challenges," *Future generation computer systems*, vol. 82, pp. 395–411, 2018.

[15] Y. Wu, J. Wang, Y. Wang, S. Zhai, Z. Li, Y. He, K. Sun, Q. Li, and N. Zhang, "Your Firmware Has Arrived: A Study of Firmware Update Vulnerabilities," in *USENIX Security Symposium*, 2023.

[16] P. Oser, R. W. van der Heijden, S. Lüders, and F. Kargl, "Risk Prediction of IoT Devices Based on Vulnerability Analysis," *ACM Transactions on Privacy and Security*, vol. 25, no. 2, pp. 1–36, 2022.

[17] Shodan. (2016) Search Engine for the Internet of Everything. Accessed: 2024-08-08. [Online]. Available: https://www.shodan.io/

[18] Censys. (2013) A Search Engine Based on Internet-wide Scanning for the Devices and Networks. Accessed: 2024-08-08. [Online]. Available: https://censys.io/

[19] Todor Majti. (2024) Securing the Internet of Things: Challenges and Solutions. Accessed: 2025-08-08. [Online]. Available: https://nordicitsecurity.com/securing-the-internet-of-things-challenges/

[20] Q. Li, D. Tan, X. Ge, H. Wang, Z. Li, and J. Liu, "Understanding Security Risks of Embedded Devices Through Fine-Grained Firmware Fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 4099–4112, 2021.

[21] B. Zhao, S. Ji, W.-H. Lee, C. Lin, H. Weng, J. Wu, P. Zhou, L. Fang, and R. Beyah, "A Large-Scale Empirical Study on the Vulnerability of Deployed IoT Devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 1826–1840, 2020.

[22] Q. Li, X. Feng, R. Wang, Z. Li, and L. Sun, "Towards Fine-grained Fingerprinting of Firmware in Online Embedded Devices," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2537–2545.

[23] L. Fan, L. He, Y. Wu, S. Zhang, Z. Wang, J. Li, J. Yang, C. Xiang, and X. Ma, "AutoIoT: Automatically Updated IoT Device Identification With Semi-Supervised Learning," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5769–5786, 2022.

[24] M. Priya, P. X. Lim, Z. Li, S. Kumar, and R. Moy, "Cross-Device Identity Resolution using Machine Learning: A Scalable Device Graph Approach," in *The 34th International FLAIRS Conference*. The Florida Artificial Intelligence Research Society, 2021, pp. 1–6.

[25] L. Yu, B. Luo, J. Ma, Z. Zhou, and Q. Liu, "You Are What You Broadcast: Identification of Mobile and IoT Devices from (Public) WiFi," in *29th USENIX security symposium (USENIX security 20)*, 2020, pp. 55–72.

[26] Y. Liu, J. Wang, J. Li, S. Niu, and H. Song, "Class-Incremental Learning for Wireless Device Identification in IoT," *IEEE Internet of Things Journal*, vol. 8, no. 23, pp. 17 227–17 235, 2021.

[27] H. Jafari, O. Omotere, D. Adesina, H.-H. Wu, and L. Qian, "IoT Devices Fingerprinting Using Deep Learning," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 1–9.

[28] Z. Li, H. Shao, L. Niu, and N. Xue, "Progressive Learning Algorithm for Efficient Person Re-Identification," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 16–23.

[29] C. Zhao, X. Lv, Z. Zhang, W. Zuo, J. Wu, and D. Miao, "Deep Fusion Feature Representation Learning With Hard Mining Center-Triplet Loss for Person Re-Identification," *IEEE Transactions on Multimedia*, vol. 22, no. 12, pp. 3180–3195, 2020.

[30] Z. Li, H. Shao, L. Niu, and N. Xue, "PLA: Progressive learning algorithm for efficient person re-identification," *Multimedia Tools and Applications*, vol. 81, no. 17, pp. 24 493–24 513, 2022.

[31] N. Xue, L. Niu, X. Hong, Z. Li, L. Hoffaeller, and C. Pöpper, "DeepSIM: GPS Spoofing Detection on UAVs using Satellite Imagery Matching," in *Proceedings of the 36th Annual Computer Security Applications Conference*, 2020, pp. 304–319.

[32] N. Xue, Z. Li, X. Hong, and C. Pöpper, "Gridnet: Vision-based mitigation of gps attacks for aerial vehicles," in *Proceedings of the 2025 International Joint Conference on Neural Networks (IJCNN)*. Rome, Italy: IEEE, 2025.

[33] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical Attention Networks for Document Classification," in *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, 2016, pp. 1480–1489.

[34] CVE. (2005) Common Vulnerabilities and Exposures. Accessed: 2024-08-08. [Online]. Available: http://cve.mitre.org/

[35] B. J. Frey and D. Dueck, "Clustering by Passing Messages Between Data Points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[36] J. Zhao and C. G. M. Snoek, "LiftPool: Bidirectional ConvNet Pooling," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=kE3vd639uRW

[37] H. Li, H. Yan, Y. Li, L. Qian, Y. He, and L. Gui, "Distinguishability Calibration to In-Context Learning," in *Findings of the Association for Computational Linguistics: EACL 2023*, 2023, pp. 1385–1397.

[38] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[39] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[40] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A Discriminative Feature Learning Approach for Deep Face Recognition," in *Computer vision–ECCV 2016: 14th European conference, amsterdam, the netherlands, October 11–14, 2016, proceedings, part VII 14*. Springer, 2016, pp. 499–515.

[41] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," in *Proceedings of*

the IEEE conference on computer vision and pattern recognition, 2015, pp. 815–823.

[42] M. Kim, D. Kim, E. Kim, S. Kim, Y. Jang, and Y. Kim, "FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis," in *Proceedings of the 36th Annual Computer Security Applications Conference*, 2020, pp. 733–745.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in neural information processing systems*, vol. 32, 2019.

[44] Grand View Research. (2025) IP Camera Market Size, Share, Trends, Industry Report 2030. Accessed: 2025-08-08. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/ip-camera-market-report

[45] Growth Market Reports. (2025) Wireless Router Market Research Report 2033. Accessed: 2025-08-08. [Online]. Available: https://growthmarketreports.com/report/wireless-router-market-global-industry-analysis

[46] L. Van Der Maaten, "Accelerating t-SNE using Tree-Based Algorithms," *The journal of machine learning research*, vol. 15, no. 1, pp. 3221–3245, 2014.

[47] P. J. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[48] D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure," *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 224–227, 1979.

[49] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 605–620.

[50] C. Sabri, L. Kriaa, and S. L. Azzouz, "Comparison of IoT Constrained Devices Operating Systems: A Survey," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2017, pp. 369–375.

[51] S. Lee, J.-Y. Paik, R. Jin, and E.-S. Cho, "Toward Machine Learning Based Analyses on Compressed Firmware," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2019, pp. 586–591.

[52] W. Zhang, H. Li, H. Wen, H. Zhu, and L. Sun, "A graph neural network based efficient firmware information extraction method for IoT devices," in *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2018, pp. 1–8.

[53] W. Zhang, Y. Chen, H. Li, Z. Li, and L. Sun, "PANDORA: A Scalable and Efficient Scheme to Extract Version of Binaries in IoT Firmwares," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[54] A. Costin, A. Zarras, and A. Francillon, "Towards Automated Classification of Firmware Images and Identification of Embedded Devices," in *ICT Systems Security and Privacy Protection: 32nd IFIP TC 11 International Conference, SEC 2017, Rome, Italy, May 29-31, 2017, Proceedings 32*. Springer, 2017, pp. 233–247.

[55] X. He, Y. Zhou, Z. Zhou, S. Bai, and X. Bai, "Triplet-Center Loss for Multi-View 3D Object Retrieval," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1945–1954.

[56] B. Hu, J. Xu, and X. Wang, "Learning generalizable deep feature using triplet-batch-center loss for person re-identification," *Science China. Information Sciences*, vol. 64, no. 2, p. 120111, 2021.