Fast Model Update for IoT Traffic Anomaly Detection with Machine Unlearning

Jiamin Fan*, Kui Wu*, Yang Zhou†, Zhengan Zhao*, Shengqiang Huang†
*Department of Computer Science, University of Victoria, Victoria, BC, Canada
† Huawei Technologies Canada Co. Ltd., Vancouver, BC, Canada

***Abstract*—It is often needed to update deep learning-based detection models in traffic anomaly detection systems for the Internet of Things (IoT) because of mislabelled samples or device firmware upgrades. Machine unlearning, a technique that quickly updates the anomaly detection model without re-training the model from scratch, has recently attracted much research attention. We propose a novel machine unlearning method, called *ViFLa*, which groups training data based on estimated unlearning probability and treats each group as a *virtual* client in the federated learning framework. Since the virtual clients are physically in the same machine, *ViFLa* only leverages the concept of data/local models isolation in federated learning without incurring any network communication. *ViFLa* adopts an attention-based aggregation method called enhanced class distribution weighted sum (ECDWS) to tackle the non-independent and identically distributed (non-IID) data problem caused by the data grouping strategy. It also introduces a new state transition ring mechanism into the statistical query (SQ) learning framework to update the local model of each virtual client quickly. Using real-world IoT traffic data, we showcase the benefit of *ViFLa* regarding its efficiency and completeness for model updates in the context of IoT traffic anomaly detection.**

***Index Terms*—Machine unlearning, Model update, IoT traffic anomaly detection.**

I. Introduction

The Internet of Things (IoT) technology has triggered and enriched many intelligent applications such as smart factories, smart transportation, and smart homes. Unlike traditional Internet applications, IoT systems have special features, e.g., cheap and easy to deploy, that make them popular but more vulnerable [1], [2]. Recently, deep learning-based anomaly detection systems have been developed to safeguard IoT systems [2]. As an essential requirement, the IoT traffic anomaly detection system needs to update the underlying machine learning model when people upgrade the firmware of an IoT device that causes small traffic changes or when some training data that were previously labelled as normal but identified as abnormal at a later time [3]. The task of updating an existing ML model to remove the impact of certain training data on the model is termed as *machine unlearning*.

Clearly, a naïve method for machine unlearning is to re-train the ML model from scratch using the complete set of updated training data. This method is also called *naïve unlearning*. Nevertheless, *naïve unlearning* is time-consuming and ineffective for time-sensitive applications such as an intrusion detection system. Cao and Yang [4] trained a naïve Bayes malware detector using $142,350$ malware samples and found that it takes nearly a day to delete a sample and re-train the model from scratch [4]. Taking hours, let alone a day, to update an anomaly detection model in the production environment is not acceptable. As such, we focus on machine unlearning methods different from naïve unlearning.

The criteria to evaluate a machine unlearning approach should consider both efficiency and completeness. Here, efficiency refers to how fast the approach can update the model, and completeness refers to how close the performance of the updated model is to that of naïve unlearning. There is a clear tradeoff between efficiency and completeness: naïve unlearning achieves the best completeness but the worst efficiency. A natural question is: *could we achieve efficiency and completeness at the same time?* We offer a positive answer for machine unlearning in IoT traffic anomaly detection by leveraging the special features related to IoT traffic and the concept of virtual federated learning.

Several recent research efforts have been devoted to machine unlearning. For instance, Bourtoule et al. [5] proposed a Sharded, Isolated, Sliced, and Aggregated (SISA) training framework, which divides large training tasks into small sub-tasks and only re-trains the shards containing the data points that need to be unlearned. Cao and Yang [4] used statistical query (SQ) to transform the learning algorithm into summation form and only updated the summations relevant to the data points that need to be unlearned. The summation forms of simple algorithms such as naïve Bayes and k-means are provided in [4]. To speed up machine unlearning, all the above methods isolate the impact of training data within a small scope. The idea of data isolation, *in principle*, is similar to federated learning, where each client trains a local model with local data, and they work together to build a global model.

We frame the data isolation principle in machine unlearning with a new concept called virtual federated learning. The main idea of <u>vi</u>rtual <u>f</u>ederated <u>l</u>earning <u>a</u>pproach (*ViFLa*) for machine unlearning is as follows: We estimate the unlearning belief values of training samples (unlearning belief refers to the likelihood that a sample in a particular application context is to be unlearned in the future), and divide the training samples into different groups based on their unlearning belief values. Each group is considered as a *client* in the traditional federated learning, and thus a local model is trained for each client. The outputs of local models are aggregated (at a virtual server) using an attention-based aggregation method called enhanced class distribution weighted sum (ECDWS) to obtain

Corresponding author: Kui Wu (email: wkui@uvic.ca).

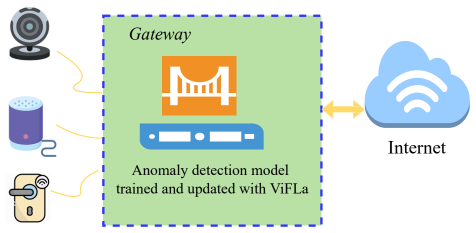


Fig. 1. *ViFLa* at the gateway for training and updating anomaly detection model. The detailed structure of *ViFLa* is in Fig. 3.

the final prediction result. Intuitively, if the data points to be unlearned are contained in a small number of clients, we can effectively speed up the unlearning process because only the clients affected by unlearned data points need to re-train their local models. We call this approach *virtual* federated learning approach because the clients and server are only conceptual and have no physical analogues in the real world. ***Virtual clients are used for limiting the impact of data in a small scope, and thus do not have any issue concerning data privacy, network communication, and delay***. Note that *ViFLa*, SISA [5], and SQ-based unlearning [4] all follow the same principle of data isolation but have quite different underlying mechanisms for building the final model, as further illustrated in Section IV-E.

We need to tackle two technical challenges in machine unlearning with *ViFLa*. First, since we intentionally divide data of high unlearning belief into a small number of clients, it is likely that the data of different virtual clients are not independent and identically distributed (non-IID). Handling non-IID data in federated learning is a challenging task [6]. Second, each client needs to quickly update its local model without re-training from scratch.

This paper systematically addresses the above challenges and makes the following contributions:

- We present a novel framework called *ViFLa*, which is deployed at network vantage points (e.g., the gateway as in Fig. 1) for IoT traffic anomaly detection. The anomaly detection model trained over the *ViFLa* framework can be quickly updated whenever needed, e.g., when an IoT device firmware update causes small traffic changes. Since the virtual clients and the virtual server can be located in the same physical location (e.g., the gateway), *ViFLa* does not need to consider the network communication and delay between the clients and the server (the disadvantage of traditional federated learning).
- Tackling the first challenge: Different from existing model aggregation methods in traditional federated learning, *ViFLa* adjusts the weights for the outputs of local models using the KL-attention mechanism, which improves the collaborative performance of local models by using their predicted vectors. The KL-attention mechanism is highly linked to data distribution and global intent and can achieve good performance in the presence of non-IID data.
- Tackling the second challenge: *ViFLa* includes a fast unlearning method for local models by introducing a state transition ring into SQ-learning. We formulate a new adaptive SQ-learning definition [7] for LSTM network. The state transitions of parameters are represented by the nodes and directed edges in the ring. Benefited from the summation form and the state transition ring, model parameter updates are much faster than *naïve unlearning*.
- We evaluate the performance of *ViFLa* on IoT-23 dataset [8] and our own dataset to demonstrate the advantages of *ViFLa*. *ViFLa* can obtain a significant speed-up and similar test performance compared to naïve unlearning. A theoretical analysis shows that compared to SISA [5], *ViFLa* can reduce the computational complexity and the amount of data affected by unlearned samples.

II. Related Work

Existing machine unlearning solutions can be roughly divided into (1) solutions that modify model parameters directly without accessing the training data, and (2) solutions that update the model with the training data.

A. Modifying Model Parameters Directly without Accessing to the Training Set

Solutions in this category modify model parameters without accessing the original training data during the unlearning process. Golatkar et al. [9] took an information-theoretical approach and proposed a “scrubbing” method. The objective of unlearning is translated to calculating optimal noise to destroy information carried in the data we wish to forget. Actually, the technique used by Golatkar et al. is equivalent to avoiding forgetting the data that we wish to retain. To be more specific, use the Fisher Information Matrix (FIM) for the samples we wish to keep, and add optimal noise to destroy information in the samples we wish to forget, i.e., add noise to destroy the weights that may have been informative about data to be forgotten but not data to be retained. Clearly, Fisher Information Matrix (FIM) and its variation are the key to letting the model remember the information in retained samples after the “scrubbing” process. The main pitfall of this approach is the high storage and computation overhead in the operations of FIM.

Loosely, “optimal brain damage” introduced by LeCun et al. [10] falls in this category. Nevertheless, the main goal is to reduce the complexity of a deep neural network by removing unimportant weights from a network. The basic idea is to use the second derivative information to measure the impact of parameters and delete those parameters that have the least effect on the training error.

B. Updating the Model Quickly with the Training Set

Solutions [5], [11], [12] in this category update the model quickly with the help of training data. The time for updating the model depends on the amount of data to be unlearned and the complexity of the learning algorithm. This type of solution mainly includes two approaches. One is to reduce the amount of data affected by the unlearning process. The other is to reduce the time complexity of the unlearning algorithm.

Regarding the first approach, Ginart et al. [11] proposed a divide-and-conquer k-means algorithm to divide the data into

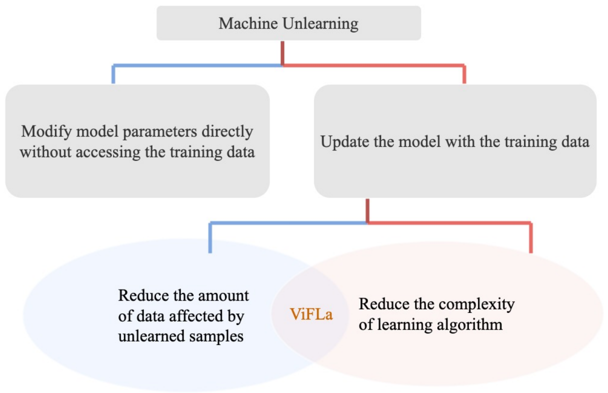

Machine Unlearning
Modify model parameters directly without accessing the training data
Update the model with the training data
Reduce the amount of data affected by unlearned samples
ViFLa
Reduce the complexity of learning algorithm

Fig. 2. Classification of machine unlearning approach.

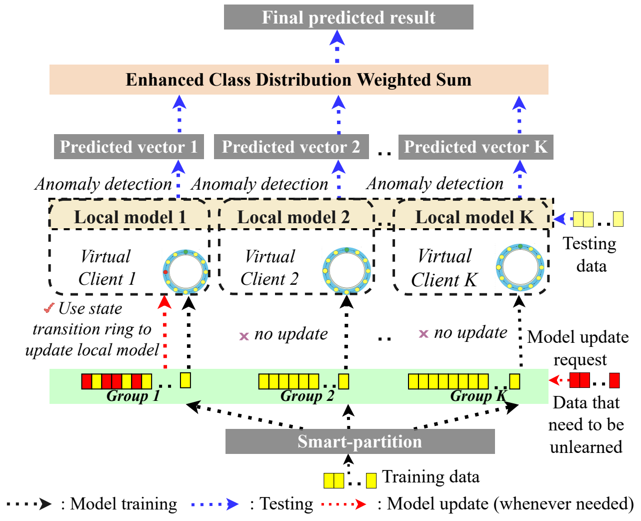


Fig. 3. Architecture of *ViFLa*.

leaves and merge the results into parent nodes. Bourtoule et al. [5] proposed a SISA framework to reduce the amount of data that needs to be unlearned. The main idea is to divide large training tasks into small sub-tasks. To remove the impact of a data point from the model, this method only needs to retrain the shard containing this data point. However, the storage overhead of the slicing process in this method may be high. Aldaghri et al. [13] encoded the training data into shards using linear encoders prior to the learning phase. However, this encoded machine unlearning approach was for simple regression models and may be hard to apply in deep learning models. Brophy et al. [14] introduced a machine unlearning method specifically for random forests. It is still unclear how to extend the technique for neural network models.

Regarding the second approach, Cao and Yang [4] used statistical query (SQ) to transform the learning algorithm into summation form. To remove the impact of a data point from the model, we only need to update the summations that involve the data and then update the model with the updated summations. Since only partial summations need to be updated, the process is much faster than *naïve unlearning*. The work [4] only includes the summation form of simple algorithms, such as naïve Bayes and k-means. In [15], Neel et al. utilized convex optimization and reservoir sampling to design a gradient-based machine unlearning method called descent-to-delete. Nevertheless, this work is mainly theoretical and depends on strong assumptions, e.g., the convexity of the model, which may not be true in practical IoT anomaly detection systems.

As shown in Fig. 2, *ViFLa* belongs to the second category, i.e., updating the model quickly with the training set, but it is different from all existing solutions in this category. To be more specific, it uses the concept of virtual federated learning to reduce the amount of data affected by unlearning. It also uses smart-partition and a new model aggregation method. Compared to the existing SQ approach [4], *ViFLa* formulates a summation form of more complex models (e.g., LSTM network), and uses state transition ring algorithm to speed up the unlearning process. It can achieve high efficiency and completeness for both IID and non-IID data.

III. Details of *ViFLa*

A. Overview

A high-level view of *ViFLa* is shown in Fig. 3, where the training, testing, and unlearning processes are denoted with black, blue, and red arrows, respectively. The training of *ViFLa* consists of two steps: smart partition and local model training. For testing, each sample is input simultaneously to all the local models and *ViFLa* uses an enhanced class distribution weighted sum (ECDWS) to aggregate the outputs of local models to obtain the final prediction result. When machine unlearning is needed, *ViFLa* performs model update with the method introduced in Section III-D.

For ease of reference, the main notations used in the paper is listed in Table I.

B. Training ViFLa

ViFLa includes mainly two steps in training:

- **Smart partition**: this step divides training samples into different groups based on their unlearning belief values. Each group is treated as a "client" as in traditional federated learning.
- **Local model training:** this step trains a local model for each virtual client. To speed up (future) unlearning process of the local model, the local model is formulated in the summation form and the local model is trained according to the summation form.

It is worth noting that *ViFLa* does not use an explicit global model. Instead, it uses an enhanced class distribution weighted sum (ECDWS) method, which is different from existing model aggregation methods in traditional federated learning. ECDWS only uses the prediction results of the local models and does not use any local model parameters.

1) Smart Partition: Unlearning belief refers to the likelihood that a sample in a particular application context is to be unlearned. If we group samples of high unlearning beliefs together and build a local model with these samples, when an unlearning request arrives, the samples included in the

TABLE I
MAIN NOTATIONS

Notation	*Definition*
x	The feature of training sample
l	Actual class value of training sample
ϵ	A user-defined unlearning threshold
G	The number of classes
K	The number of sub-models
F	The number of features
$p(l\|x_1, x_2, ..., x_F)$	The predicted probability that the sample belongs to its actual label l
$p(l)$	The probability that the sample has actual class value l
$p(x_f)$	The predicted probability that the training samples have feature value x_f
$p(x_f\|l)$	The probability that the training samples have feature value x_f given that the actual class value is l
n_i	A node in state transition ring
e_i	An edge in state transition ring
B	The number of mini-batches
θ_{S_j}	The parameter state in node n_j
$g_{\theta_{S_j}}(batch_b)$	The gradient of $batch_b$ at state θ_{S_j}
$f_{\theta_{S_j}}(batch_b)$	The mapping result of a sample $batch_b$ of query SQ_j at state θ_{S_j}
A_{SQ_j}	The SQ answer at parameter state θ_{S_j}
$b^*_{x,l}$	Unlearning belief of sample (x, l)
$\hat{A}_{SQ_j}$	The SQ answer at parameter state θ_{S_j} during unleaning
c^l_i	Sub-model i's reliable value on class l
N_l	The number of samples of class l
$N_{i,l}$	The number of samples of class l in group i
v_i	The prediction of sub-model i's outputs
v^l_i	The probability that a test sample is predicted as class l by sub-model i
$\mathcal{R}(j)$	The similarity vector for candidate vector v_j
$\mathcal{R}(j)^{SUM}$	The sum of all the elements in the similarity vector $\mathcal{R}(j)$
$\mathcal{R}(j, i)$	The similarity of prediction vectors between sub-model j and sub-model i
$\sigma(\mathcal{R}(j, i))$	The softmax result of each element $\mathcal{R}(j, i)$ of the similarity vector
v	Final predicted vector

* The first part of notations is for smart partition; the second part is for state transition ring; the third part is for naïve class distribution weighted sum method (NCDWS) and enhanced class distribution weighted sum method (ECDWS).

unlearning request may be relevant to only one or very few local models. The majority of local models do not need to be re-trained.

Clearly, accurate estimation on unlearning belief is key for smart partition. Such an estimation depends on specific application context, and there exists no one-fit-all solution. As an example, we use naïve Bayes classifier [16], [17] to estimate unlearning belief due to its simplicity yet surprising efficacy in many complex real-world situations [18], [19].

Let $X = (X_1, X_2, \ldots, X_F)$ denote F features of training samples, where each feature takes value from its domain D_f. Let Z denotes the classes of the training samples, where Z can take one of G values $\{1, \ldots, G\}$. The details for preparing training samples from raw IoT traffic data can be found in Section IV-B.

Given the feature value $x = (x_1, x_2, \ldots, x_F)$ of a training sample, the predicted probability that the sample belongs to

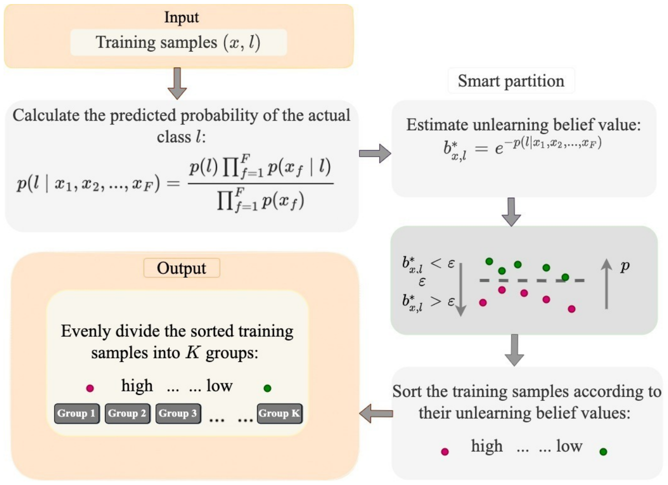

Input
Training samples (x, l)
Calculate the predicted probability of the actual class l:
$p(l \mid x_1, x_2, ..., x_F) = \frac{p(l)\prod_{f=1}^{F} p(x_f \mid l)}{\prod_{f=1}^{F} p(x_f)}$
Smart partition
Estimate unlearning belief value:
$b^*_{x,l} = e^{-p(l|x_1,x_2,...,x_F)}$
$b^*_{x,l} < \varepsilon$
ε
$b^*_{x,l} > \varepsilon$
p
Sort the training samples according to their unlearning belief values:
high ... ... low
Output
Evenly divide the sorted training samples into K groups:
high ... ... low
Group 1
Group 2
Group 3
Group K

Fig. 4. Workflow of smart partition. The range of the unlearning belief is $0.37 \leq b^*_{x,l} \leq 1$ since $0 \leq p(l|x_1, x_2, ..., x_F) \leq 1$. ϵ is a user-defined threshold.

its actual class $l (1 \leq l \leq G)$ can be calculated as:

$$p(l|x_1, x_2, ..., x_F) = \frac{p(l)\prod_{f=1}^{F} p(x_f|l)}{\prod_{f=1}^{F} p(x_f)}, \tag{1}$$

where $p(x_f)$ denotes the predicted probability of training samples with feature value x_f, $p(l)$ denotes the probability of training samples with actual class value l, $p(x_f|l)$ denotes the probability of training samples with feature value x_f given that the actual class value is l.

We propose to estimate the unlearning belief value for the sample $(X = x, Z = l)$:

$$b^*_{x,l} = e^{-p(l|x_1,x_2,...,x_F)}. \tag{2}$$

The above estimation is based on the observation that the higher the probability $p(l|x_1, x_2, ..., x_F)$, the higher the confidence that the system predicts the sample as class l, and the lower the unlearning belief. As shown in Fig. 4, a lower unlearning belief value implies a higher confidence in the prediction result, e.g., the probability that the system predicts the sample as class l in the context of intrusion detection of Internet of Things (IoT). An unlearning belief value greater than ϵ is considered as a high unlearning value, where ϵ is a user-defined unlearning threshold.

After we estimate the unlearning belief for each training sample, we group the training samples according to their unlearning belief. To be more specific, given the number of groups K, we order the training samples based on their unlearning belief values and then evenly divide the ordered samples into the K groups. In this way, we try to contain the data points that are to be unlearned in a small number of groups. After that, each group is treated as a client as in the federated learning that trains a local model using the data in the group. Refer to Section IV-E1 for the explanation on why smart partition works in practice.

2) Local Model Training: Each group is considered as a *virtual client* in the FL framework. We use the LSTM network [20] as an example to illustrate the training method.

To speed up future unlearning, we adopt statistical query (SQ) learning [4] to transform the LSTM network into summation form, whose details are given in Appendix. Each LSTM local model is trained with mini-batch gradient descent, and the training of local models is independent of each other. In addition, we introduce a new state transition ring mechanism as shown in Fig. 5. The state transition ring involves B nodes $(n_0, n_1, \ldots, n_{B-1})$ and B directed edges $(e_0, e_1, \ldots, e_{B-1})$, where B is the total number of mini-batches. The state in node n_j is represented as θ_{S_j}. We use node n_0 as the root node and store the initial parameter state θ_{S_0} in this node.

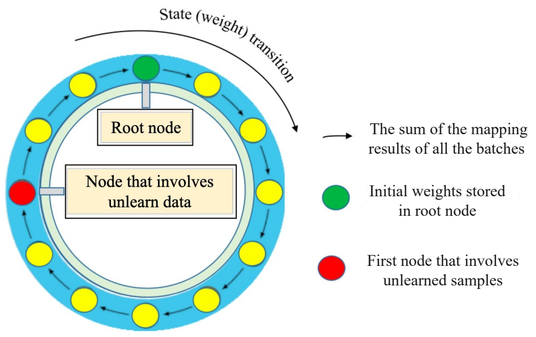

State (weight) transition
Root node
Node that involves unlearn data
The sum of the mapping results of all the batches
Initial weights stored in root node
First node that involves unlearned samples

Fig. 5. State transition ring.

Statistical query (SQ) means the learning algorithm can only query statistics about the training samples. We provide query SQ_j for each state θ_{S_j}. The query consists of a mapping function and a set of training batches $Q = \{batch_0, batch_1, \ldots, batch_{B-1}\}$. The mapping result of a sample $batch_b \in Q$ of query SQ_j at state θ_{S_j} is calculated as,

$$f_{\theta_{S_j}}(batch_b) = \begin{cases} g_{\theta_{S_j}}(batch_b) & \text{if } b = j \\ 0 & otherwise \end{cases} \tag{3}$$

where $g_{\theta_{S_j}}(batch_b)$ is the gradient of $batch_b$ at state θ_{S_j}.

We then calculate the summation-form answers of these statistical queries (SQ). The answer of a statistical query SQ_j is the sum of the mapping results of all the batches in the training set, denoted as $A_{SQ_j} = \sum_{b=0}^{B-1} f_{\theta_{S_j}}(batch_b)$. Using the answer A_{SQ_j}, the parameter state θ_{S_j} (stored in node n_j) is transferred to the state $\theta_{S_{j+1}}$ (stored in the node n_{j+1}). After updating all the states in the ring, we set the latest state θ_{S_B} as the new initial state θ_{S_0} and store it in the root node n_0. We repeat this process until convergence determined by "early stopping" [21], i.e., the accuracy on the validation set has reached the expected value, and the accuracy will not exceed this value in the next few iterations. The pseudo-code for local model training is shown in Algorithm 1.

Remark. The state transition ring is not a new machine learning algorithm but just a summation form [4] of traditional ML algorithms that makes the machine unlearning process faster. Specifically, it degenerates the converged state in the learning process to the previous state by updating the summation form. This previous state serves as the initial parameter state to learn a new convergent state. We can thus speed up the unlearning process since the new initial state is close to a convergent state. It is also worth noting that an epoch in the traditional form of ML algorithms means one iteration over all training data. When one or several epochs are trained, we update the checkpoint of the model and only one checkpoint (state) is saved. In contrast, the state transition ring keeps a separate state for each batch and stores it in a node. Each yellow node in the ring corresponds to a batch and a model state. During the learning process, the state stored in a node is only updated when the batch corresponding to its previous node is trained.

Algorithm 1: Local Model Training in *ViFLa*

```
Input: Initial parameter state $\theta_{S_0}$ stored in root node
       $n_0$, mini-batches $Q = \{batch_0, \ldots, batch_{B-1}\}$
Output: state $\theta_{S_1}, \ldots, \theta_{S_{B-1}}, \theta_{S_0}$ in the last epoch
1  repeat
2      for SQ $j \leftarrow 0$ to $B-1$ do
3          for Batch $b \leftarrow 0$ to $B-1$ do
4              if $j = b$ then
5                  $f_{\theta_{S_j}}(batch_b) = g_{\theta_{S_j}}(batch_b)$;
6                  $A_{SQ_j} = A_{SQ_j} + f_{\theta_{S_j}}(batch_b)$;
7              else
8              end
9          end
10         $e = (j+1) \mod B$;
11         Transfer state $\theta_{S_j}$ stored in node $n_j$ to state
            $\theta_{S_e}$ stored in node $n_e$ by using the summation
            form SQ answer $A_{SQ_j}$;
12     end
13 until Convergence;
14 return Final state $\theta_{S_1}, \ldots, \theta_{S_{B-1}}, \theta_{S_0}$ stored in nodes
    $n_1, \ldots, n_{B-1}, n_0$, respectively;
```

C. Testing ViFLa

Each test sample will be input to all local models simultaneously. After that, *ViFLa* uses an enhanced class distribution weighted sum (ECDWS) to aggregate the outputs of local models to obtain the final prediction result. Before presenting ECDWS, we first introduce a naïve class distribution weighted sum method (NCDWS), and then explain how ECDWS overcomes the main pitfall of NCDWS.

*1) **Naïve Class Distribution Weighted Sum (NCDWS):*** The motivation of NCDWS is to alleviate the impact of non-iid data in different local models since the aggregation performance is highly linked to the data distribution. In this method, we analyze the class distributions in different local models *in advance*, and determine the aggregation parameters based on the class distribution. For a test sample, each submodel i outputs its prediction,

$$v_i = \begin{bmatrix} v_i^1 & v_i^2 & \ldots & v_i^G \end{bmatrix}^T, \tag{4}$$

where v_i^l $(1 \leq i \leq K, 1 \leq l \leq G)$ denotes the probability that the sample is predicted as class l. The predicted vectors in different local models will be used as input vectors of NCDWS. One simple solution is to use the weighted average over all the predicted vectors. However, this method may suffer from non-IID data since the distribution of training data largely impacts the test performance. To address the problem, we define a class-based reliable value for each sub-model. The reliable value is used to represent the reliability of the prediction result of each sub-model for different classes, denoted as

$$\begin{Bmatrix} c_1^1 & c_2^1 & ... & c_K^1 \\ c_1^2 & c_2^2 & ... & c_K^2 \\ ... & ... & ... & ... \\ c_1^G & c_2^G & ... & c_K^G \end{Bmatrix}$$

where c_i^l represents sub-model i's reliable value on class l. We use c_i^l=$\frac{N_{i,l}}{N_l}$ to approximate sub-model i's reliable value on class l since a sub-model observes better performance on a class if it includes more training samples of this class. $N_{i,l}$ is the amount of samples with class l in sub-model i and N_l is the total amount of samples with class l. The detail of NCDWS is illustrated in Algorithm 2.

```
Algorithm 2: Naïve Class Distribution Weighted Sum (NCDWS)
  Input: Predicted vector $v_i$ $(1 \leq i \leq K)$ of the
         sub-models, $N_l(1 \leq l \leq G)$,
         $N_{i,l}(1 \leq i \leq K, 1 \leq l \leq G)$
  Output: Final predicted vector $v$
1 Initialing $v^l$=0;
2 for class $l \leftarrow 1$ to $G$ do
3     for sub-model $i \leftarrow 1$ to $K$ do
4         $v_i^l = c_i^l * v_i^l$ ;
5         $v^l = v^l + v_i^l$ ;
6     end
7 end
8 return Final predicted vector $v$=($v^1$,$v^2$,...,$v^G$);
```

NCDWS achieves good performance by considering class distribution. However, it is not practical since it needs to know the distribution of training data in different local models in advance. This assumption is too strong and too demanding in practice since (1) it needs to test the distribution of training data and (2) this distribution may change after model unlearning. To overcome this problem, we introduce an enhanced class distribution weighted method (ECDWS). Unlike NCDWS, ECDWS does not need to know any class distribution information in advance.

2) ***Enhanced class distribution weighted sum:*** The core of ECDWS is the KL-attention recommendation mechanism, which we propose based on the concepts of KL-divergence [22] and self-attention [23], [24]. The main idea of KL-attention recommendation is to give higher weights to similar outputs of local models. For a test sample, each sub-model i outputs its prediction,

$$v_i = \begin{bmatrix} v_i^1 & v_i^2 & \dots & v_i^G \end{bmatrix}^T, \tag{5}$$

where v_i^l $(1 \leq i \leq K, 1 \leq l \leq G)$ denotes the probability that the sample is predicted as class l. To capture the global intent of predicted vectors and ignore unimportant information, we consider each sub-model's predicted vector as a candidate item in an KL-attention recommendation system. We calculate the similarity between two candidate vectors v_j and v_i $(1 \leq j, i \leq K)$. The similarity function $\mathcal{R}(j,i)$ is defined based on KL divergence:

$$\mathcal{R}(j,i) = e^{-\sum_{l=1}^{G} v_j^l \log \frac{v_j^l}{v_i^l}}. \tag{6}$$

We obtain a similarity vector $\mathcal{R}(j)$ for each candidate vector v_j, where $\mathcal{R}(j) = (\mathcal{R}(j,1), \mathcal{R}(j,2), \dots, \mathcal{R}(j,K))$. Denote $\mathcal{R}(j)^{SUM}$ as the sum of all the elements in the similarity vector $\mathcal{R}(j)$, i.e., $\mathcal{R}(j)^{SUM} = \sum_{i=1}^{K} \mathcal{R}(j,i)$. We apply the softmax function to each element $\mathcal{R}(j,i)$ of the similarity vector, i.e., $\mathcal{R}(j)$=$(\sigma(\mathcal{R}(j,1)), \sigma(\mathcal{R}(j,2)), ..., \sigma(\mathcal{R}(j,K)))$, where $\sigma(\mathcal{R}(j,i)) = \frac{\mathcal{R}(j,i)}{\mathcal{R}(j)^{SUM}}$.

We then represent the candidate vector v_j as a weighted sum of all the candidate vectors:

$$v_j = \sum_{i=1}^{K} \sigma(\mathcal{R}(j,i)) v_i. \tag{7}$$

That is, the weight is determined by the similarity of candidate vectors. We repeat this process to update all the candidate vectors until convergence. The proof of the convergence of this process is given in Appendix. The detail of ECDWS is given in Algorithm 3.

```
Algorithm 3: Enhanced Class Distribution Weighted Sum (ECDWS)
   Input: Predicted vector $v_i$ $(\forall 1 \leq i \leq K)$ of the
          sub-models
   Output: Final predicted vector $v$
1  Initializing $R(j)^{SUM} = 0$;
2  repeat
3      for candidate vector $j \leftarrow 1$ to $K$ do
4          for candidate vector $i \leftarrow 1$ to $K$ do
5              $\mathcal{R}(j,i) = e^{-\sum_l v_j^l \log \frac{v_j^l}{v_i^l}}$;
               $\mathcal{R}(j)^{SUM}$=$\mathcal{R}(j)^{SUM}$+$\mathcal{R}(j,i)$;
6          end
7          for candidate vector $i \leftarrow 1$ to $K$ do
8              $\sigma(\mathcal{R}(j,i)) = \frac{\mathcal{R}(j,i)}{\mathcal{R}(j)^{SUM}}$;
9          end
10         $v_j = \sum_i \sigma(\mathcal{R}(j,i)) v_i$;
11     end
12 until Convergence;
13 $v = \frac{\sum_{j=1}^{K} v_j}{K}$ ;
14 return Final predicted vector $v$=($v^1$,$v^2$,...,$v^G$);
```

3) *Summary of Test Procedure:* We input the test sample into the local models simultaneously and obtain a predicted vector from each of the sub-models. The i-th value in the vector denotes the probability that the sample belongs to the i-th class. Then we apply the ECDWS algorithm to aggregate these predicted vectors to obtain the final vector. The sample is predicted as class i, where the i-th value in the final vector is the largest. If there is a tie (i.e., multiple classes have the same highest value), randomly assign a class from the tie.

D. Unlearning with ViFLa

The unlearning process in *ViFLa* is straightforward. To unlearn some data, we update the summation form SQ answers A_{SQ_j} $(\forall 0 \leq j \leq B-1)$ in the learning process by removing the contribution of the unlearned batch $batch_b$ $(b \in U)$, where U denotes the set of the index of all the unlearned batches. A batch is considered as unlearned batch if its mapping result at any state in the state transition ring is affected by the unlearned samples. Then the convergent state θ_{S_B} stored in node n_0 in the learning process will degenerate to a previous state. We use this previous state as our new initial parameter state and learn a new convergent state. Different from the learning process, the unlearning process is very fast since the new initial state is close to a convergent state.

The main steps of the unlearning process are as follows:

- Step 1: Obtain the summation form SQ answer $A_{SQ_j}(\forall 0 \leq j \leq B-1)$ in the learning process.
- Step 2: Calculate the new SQ answers, denoted as $\hat{A}_{SQ_j}(\forall 0 \leq j \leq B-1)$, by removing the mapping results of the unlearned batches from the SQ answers obtained in the learning process, i.e., $\hat{A}_{SQ_j}$=$A_{SQ_j} - \sum_{b\in U} f_{\theta_{S_j}}(batch_b)$ $(\forall 0 \leq j \leq B-1)$.
- Step 3: Use the new summation form SQ answers $\hat{A}_{SQ_j}(\forall 0 \leq j \leq B-1)$ obtained in step 2 to update the parameter state. The convergent parameter state θ_{S_B} will degenerate to a previous state.
- Step 4: Set this previous state as new initial state.
- Step 5: Repeat the training process (Section III-B2) until convergence.

IV. *ViFLa* in Action: Machine Unlearning for IoT Anomaly Detection

A. How and When Should ViFLa Be Used?

Note that *ViFLa* itself is *not* an anomaly detection model. Instead, *ViFLa* provides a framework to train the detection model and more importantly to quickly update the trained model. It must work with an underlying anomaly detection model for traffic anomaly detection. Therefore, *ViFLa* should be deployed at the same place where the anomaly detection model is trained and used, e.g., security gateways of an IoT system. *ViFLa* can be applied to an IoT network wherever multiple IoT devices are connected to the network.

Model update is a much-needed feature for anomaly detection in IoT. For instance, when some normal data samples used for model training are later identified as anomalies or when an IoT device upgrades its firmware and causes changes in traffic patterns, the detection model should forget the impact of obsolete data and accommodate the contribution of new data. We in the following evaluate the benefit of *ViFLa* with the underlying LSTM anomaly detection model (refer to Appendix) and real-world trace data.

B. Data Preprocessing

We use IoT-23 dataset [8] as well as our own DCS-932LB data. IoT-23 includes malicious and benign communication sequences of different IoT devices. We use benign packets of a Philips HUE smart LED lamp ($21,664$ benign packets) and Malware-Capture-34-1 ($233,865$ packets). DCS-932LB includes $28,867$ benign packets and $18,559$ Mirai malware packets of a smart camera. We use IoT-23 dataset as it is a commonly-used public dataset. Unfortunately, this dataset does not provide data for some experiments, e.g., IoT traffic data before and after device firmware update. Therefore, we build our own testbed and collect dataset to complete the experiments.

Anomaly detection can be built based on the communication patterns of IoT devices. Using the 7 features proposed in [2], we extract the characteristic patterns from the sequence of benign data packets, and then determine whether the characteristic pattern of a device is consistent with the patterns learned from benign sequences. The seven features include: traffic direction (incoming/outgoing), bin index of local port number, bin index of remote port number, bin index of packet length, tcp flags, protocol type, and bin index of packet inter-arrival time. Using the above 7 features, We extract a feature tuple $(a_1, \ldots, a_7)$ for each packet and then map each packet to a *packet type* based on its feature tuple. The total number of packet types is 38, i.e., the value of the feature tuple has 38 different combinations. As a result, the data packet sequence in the dataset will be converted to a sequence of packet types.

After that, we convert the packet type sequence to the actual input of *ViFLa* using a sliding window of size F. For any $m(> F)$, the packet type sequence $(h_{m-F}, h_{m-F+1}, \ldots, h_{m-1})$ of F preceding packets $PK_{m-F}, PK_{m-F+1}, \ldots, PK_{m-1}$ is used as the feature values $(x_1, x_2, \ldots, x_F)$ of the input sample (refer to Section III-B1), and the type of packet PK_m is used as the actual class value l of the input sample. These feature-class pairs $((x_1, x_2, \ldots, x_F), l)$ can be used as training/testing samples of *ViFLa*. We train a three-layer LSTM model in the *ViFLa* framework for effective machine unlearning. Given any $m(> F)$ and the types of the F preceding packets $PK_{m-F}, PK_{m-F+1}, \ldots, PK_{m-1}$, the detection model can estimate the probability distribution that data packet PK_m belongs to different packet types. We set $F = 20$.

C. Completeness

To evaluate the completeness, we compare the results of *ViFLa* and the results of naïve unlearning. We use the confusion matrix to visualize the results, where each row represents the samples in an actual class, each column represents the samples in a predicted class, and the value at (i, j) denotes the probability that class i is predicted as class j.

As shown in Fig. 6, the confusion matrix obtained with *ViFLa* after removing the contribution of some samples and the confusion matrix obtained with a model re-trained from scratch are very close. To quantitatively measure the difference between two confusion matrices, we also calculate Cohen's kappa coefficient [25] between the two confusion matrices, which is a well-known metric to assess the agreement between two classifiers. The kappa score is above 0.9. Note that a kappa score above 0.8 is generally considered good agreement [25]. The evaluation results demonstrate that ViFLa can achieve nearly identical performance of retraining from scratch.

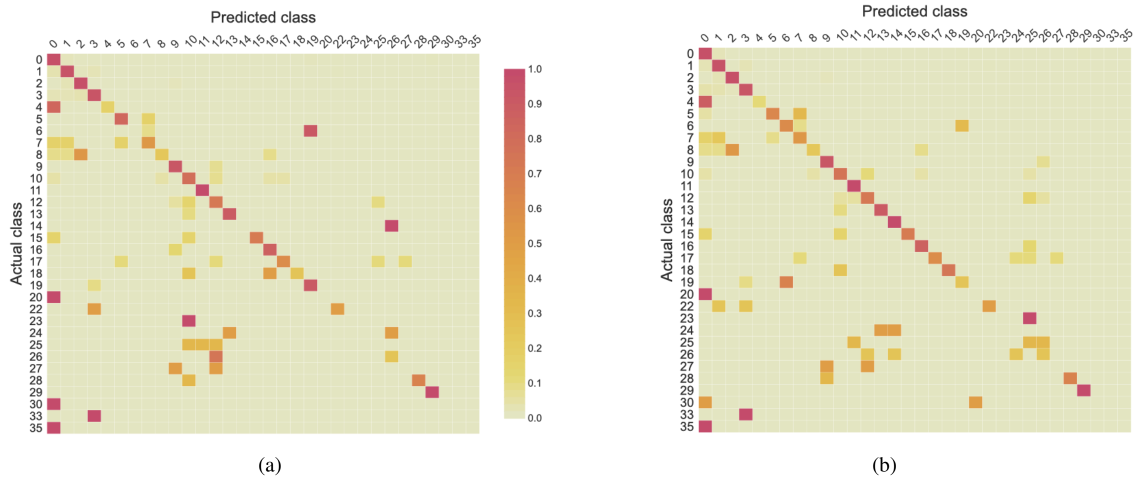


Fig. 6. The confusion matrix of test data after removing the contribution of 5% training samples of the model by: (a) naïve unlearning. (b) *ViFLa*. The Kappa index between the two confusion matrices is 0.938. We also test the case of removing contribution of 20% training samples of the model by naïve unlearning and *ViFLa* (figures omitted for brevity). The Kappa index between the two confusion matrices is 0.921.

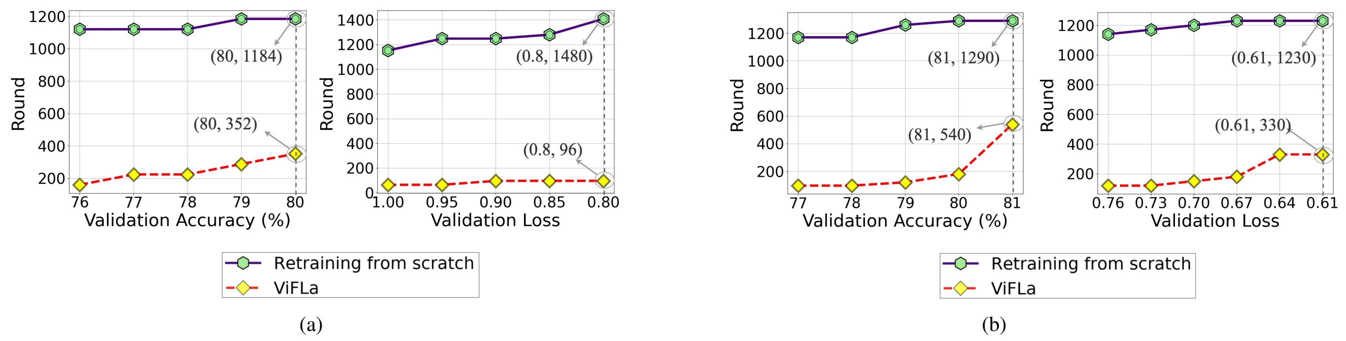
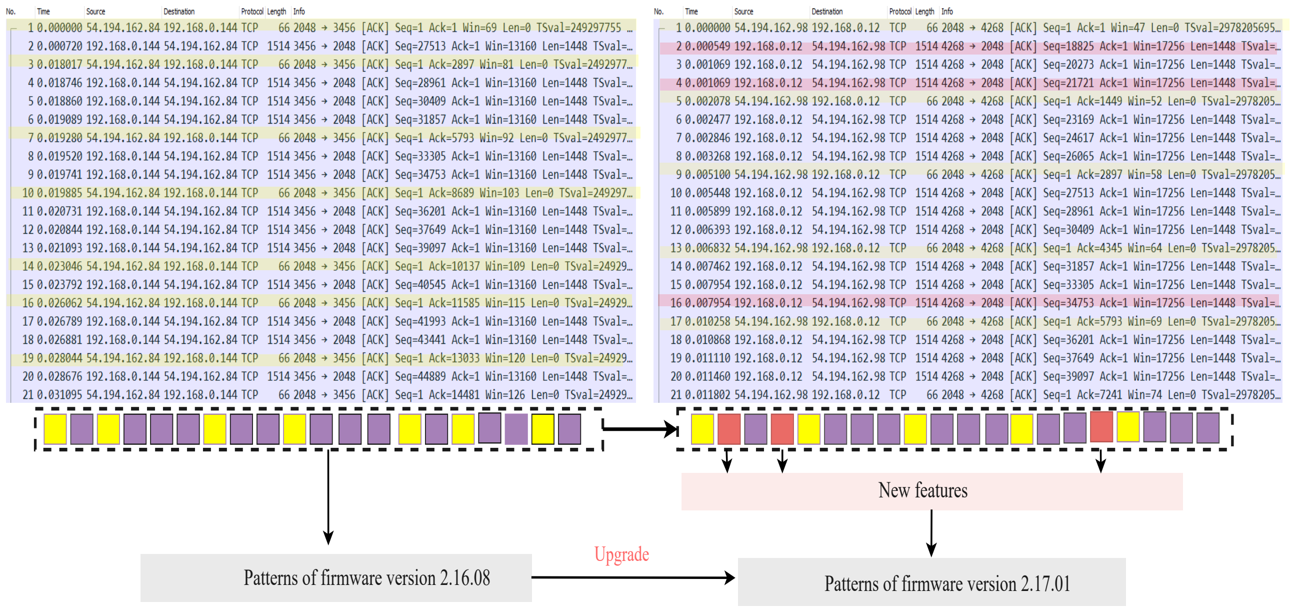


Fig. 7. The speed comparison of two unlearning methods in training a new convergence when an IoT device (camera model: DCS-932LB) upgrades its firmware version from: (a) V2.16.08 to V2.17.01, (b) V2.17.01 to V2.18.01.


```
No. Time     Source         Destination    Protocol Length Info
  1 0.000000 54.194.162.84  192.168.0.144  TCP        66 2048 → 3456 [ACK] Seq=1 Ack=1 Win=69 Len=0 TSval=249297755 …
  2 0.000720 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=27513 Ack=1 Win=13160 Len=1448 TSval=…
  3 0.018017 54.194.162.84  192.168.0.144  TCP        66 2048 → 3456 [ACK] Seq=1 Ack=2897 Win=81 Len=0 TSval=2492977…
  4 0.018746 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=28961 Ack=1 Win=13160 Len=1448 TSval=…
  5 0.018860 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=30409 Ack=1 Win=13160 Len=1448 TSval=…
  6 0.019089 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=31857 Ack=1 Win=13160 Len=1448 TSval=…
  7 0.019280 54.194.162.84  192.168.0.144  TCP        66 2048 → 3456 [ACK] Seq=1 Ack=5793 Win=92 Len=0 TSval=2492977…
  8 0.019520 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=33305 Ack=1 Win=13160 Len=1448 TSval=…
  9 0.019741 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=34753 Ack=1 Win=13160 Len=1448 TSval=…
 10 0.019885 54.194.162.84  192.168.0.144  TCP        66 2048 → 3456 [ACK] Seq=1 Ack=8689 Win=103 Len=0 TSval=249297…
 11 0.020731 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=36201 Ack=1 Win=13160 Len=1448 TSval=…
 12 0.020844 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=37649 Ack=1 Win=13160 Len=1448 TSval=…
 13 0.021093 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=39097 Ack=1 Win=13160 Len=1448 TSval=…
 14 0.023046 54.194.162.84  192.168.0.144  TCP        66 2048 → 3456 [ACK] Seq=1 Ack=10137 Win=109 Len=0 TSval=24929…
 15 0.023792 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=40545 Ack=1 Win=13160 Len=1448 TSval=…
 16 0.026062 54.194.162.84  192.168.0.144  TCP        66 2048 → 3456 [ACK] Seq=1 Ack=11585 Win=115 Len=0 TSval=24929…
 17 0.026789 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=41993 Ack=1 Win=13160 Len=1448 TSval=…
 18 0.026881 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=43441 Ack=1 Win=13160 Len=1448 TSval=…
 19 0.028044 54.194.162.84  192.168.0.144  TCP        66 2048 → 3456 [ACK] Seq=1 Ack=13033 Win=120 Len=0 TSval=24929…
 20 0.028676 192.168.0.144  54.194.162.84  TCP      1514 3456 → 2048 [ACK] Seq=44889 Ack=1 Win=13160 Len=1448 TSval=…
 21 0.031095 54.194.162.84  192.168.0.144  TCP        66 2048 → 3456 [ACK] Seq=1 Ack=14481 Win=126 Len=0 TSval=24929…
```

```
No. Time     Source         Destination    Protocol Length Info
  1 0.000000 54.194.162.98  192.168.0.12   TCP        66 2048 → 4268 [ACK] Seq=1 Ack=1 Win=47 Len=0 TSval=2978205695…
  2 0.000549 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=18825 Ack=1 Win=17256 Len=1448 TSval=…
  3 0.001069 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=20273 Ack=1 Win=17256 Len=1448 TSval=…
  4 0.001069 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=21721 Ack=1 Win=17256 Len=1448 TSval=…
  5 0.002078 54.194.162.98  192.168.0.12   TCP        66 2048 → 4268 [ACK] Seq=1 Ack=1449 Win=52 Len=0 TSval=2978205…
  6 0.002477 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=23169 Ack=1 Win=17256 Len=1448 TSval=…
  7 0.002846 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=24617 Ack=1 Win=17256 Len=1448 TSval=…
  8 0.003268 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=26065 Ack=1 Win=17256 Len=1448 TSval=…
  9 0.005100 54.194.162.98  192.168.0.12   TCP        66 2048 → 4268 [ACK] Seq=1 Ack=2897 Win=58 Len=0 TSval=2978205…
 10 0.005448 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=27513 Ack=1 Win=17256 Len=1448 TSval=…
 11 0.005899 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=28961 Ack=1 Win=17256 Len=1448 TSval=…
 12 0.006393 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=30409 Ack=1 Win=17256 Len=1448 TSval=…
 13 0.006832 54.194.162.98  192.168.0.12   TCP        66 2048 → 4268 [ACK] Seq=1 Ack=4345 Win=64 Len=0 TSval=2978205…
 14 0.007462 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=31857 Ack=1 Win=17256 Len=1448 TSval=…
 15 0.007954 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=33305 Ack=1 Win=17256 Len=1448 TSval=…
 16 0.007954 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=34753 Ack=1 Win=17256 Len=1448 TSval=…
 17 0.010258 54.194.162.98  192.168.0.12   TCP        66 2048 → 4268 [ACK] Seq=1 Ack=5793 Win=69 Len=0 TSval=2978205…
 18 0.010868 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=36201 Ack=1 Win=17256 Len=1448 TSval=…
 19 0.011110 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=37649 Ack=1 Win=17256 Len=1448 TSval=…
 20 0.011460 192.168.0.12   54.194.162.98  TCP      1514 4268 → 2048 [ACK] Seq=39097 Ack=1 Win=17256 Len=1448 TSval=…
 21 0.011802 54.194.162.98  192.168.0.12   TCP        66 2048 → 4268 [ACK] Seq=1 Ack=7241 Win=74 Len=0 TSval=2978205…
```
New features
Upgrade
Patterns of firmware version 2.16.08
Patterns of firmware version 2.17.01

Fig. 8. The traffic difference when the firmware of IoT device (camera model: DCS-932LB) is upgraded. After the firmware update, the user captures the new traffic (right) of this device with wireshark and submits a model update request to *ViFLa*, which automatically identifies the difference in the feature and updates the detection model accordingly. Different colours illustrate different packet types (Refer to Section IV-B for the detail of determining the type of a packet). The coloured boxes at the bottom of the packet window mean that for every packet, the sequence of 20 preceding packet types is used as the feature.

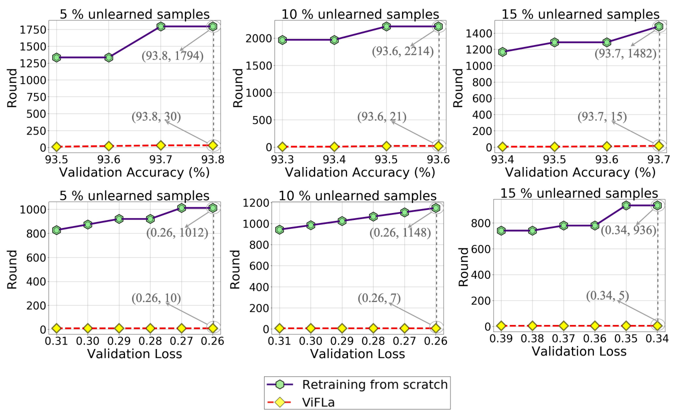


Fig. 9. The speed comparison of two unlearning methods in training a new convergence state for remaining training data.

D. Efficiency

In the first test, we emulate the scenario where an IoT device upgrades firmware and thus accordingly needs to quickly upgrade the corresponding detection model. The IoT device we tested is a D-Link smart camera (model DCS-932LB). We upgraded its firmware from V2.16.08 to V2.17.01, and then from V2.17.01 to V2.18.01. We recorded its traffic before and after each upgrade. Following the feature extraction method of [2], we extract characteristic patterns from the data packet sequence of the old firmware version and the new firmware version of the IoT device, respectively. To quickly update the detection model after the firmware upgrade, we can use *ViFLa* to re-train the model. Note that we do not need to manually compare the data samples in the two firmware versions. Instead, we only need to replace the entire data sample of the old version with the entire data sample of the new version, and then use the parameters in the old version of the model as the initial weights to retrain a model for the new version. *ViFLa* can automatically identify the difference (based on the initial weights and the state transition ring) and update the model accordingly. As a comparison, we also retrain the model from scratch using all the new traffic. As shown in Fig. 7, *ViFLa* achieves a significant speedup compared to retraining from scratch when we upgrade the device. Each round of training means that we have completed the training of a batch of samples (the batch size is 128).

Fig. 8 illustrates the traffic difference before/after the camera's firmware is upgraded, from which we can see that firmware upgrade only causes small changes in traffic patterns of the IoT device. The characteristic patterns in the new version are similar to the characteristic patterns learned from the old version. Benefiting from this and with the help of the state transition ring, the convergence state of the old version will be close to the convergence state of the new version. This explains why *ViFLa* speeds up retraining from scratch.

In the second test, we randomly select a fraction of training samples with probability following their unlearning belief values. To unlearning the selected samples, we re-train the model via *ViFLa* and via naïve unlearning, respectively. The training rounds required for different validation accuracy and average validation loss of remaining data in *ViFLa* and the training rounds using naïve unlearning is shown in Fig. 9. After unlearning training samples with the highest 5% belief values, we can see that *ViFLa* needs 30 rounds to obtain 93.8% validation accuracy of the remaining data, while retraining from scratch requires 1794 rounds. In other words, *ViFLa* can achieve up to 60X speedup compared to naïve unlearning.

Note that we compare our proposed method with naïve unlearning for two main considerations. First, most of the existing methods have already reported their improvements over naïve unlearning. Readers can compare different methods more objectively by seeing how much improvement each method achieves using naïve unlearning as the baseline. Second, due to the lack of experimental details, it is difficult to fairly compare our method with the existing methods directly. For instance, [15] presents a general method that may have different implementation details.

E. Ablation Study: Compared to Other Methods

ViFLa is quite different from existing machine unlearning methods, such as SISA [5], [3] and [4], in that *ViFLa* includes new mechanisms, such as smart partition, ECDWS, and state transition ring. In addition, the underlying machine learning models of *ViFLa* is different from that in [4], [5]. Due to this reason, it would be difficult to obtain fair comparison between *ViFLa* and existing solutions. Nevertheless, we can remove some mechanisms in *ViFLa* to gain insight on the advantages of *ViFLa* over others, because the performance of *ViFLa* with ablated function roughly reflects that of existing solutions. In the ablation experiment, we also use IoT-23 data.

1) Smart Partition: To evaluate the benefit of smart partition, we randomly selected some training samples and mislabeled their classes (i.e., modified the actual class to a different class). Then we use k-fold ($k = 4$) cross-validation to calculate the unlearning belief value for each sample. Each sample is used once in the test set and used to train the naïve Bayes model 3 times. As a result, we can predict the probability of each sample belonging to its labeled class. Then, we average the predicted probability of all the mislabeled samples, and further estimate their average unlearning belief value (the threshold $\epsilon = 0.9$). We found that mislabeled samples would gain larger unlearning belief values. We then group (the group number is 4) training samples according to their unlearning belief. The test result is then compared with that obtained from randomly group training samples. The benefit of smart partition is reflected at the first and third rows of Table II under different combinations. The smart partition-based method only needs to update one single group to forget the mislabelled data, while random group method need to update all four groups.

Remark 1. *The good performance of smart partition comes from the capability of the naïve Bayes model in quickly learning the patterns in training samples. This does not necessarily mean that the naïve Bayes model is a good candidate for IoT anomaly detection. The purpose of smart partition is to quickly estimate the chance that a sample might need to be unlearned in the future, and thus achieving highly accurate estimation, while helpful, is not the goal.*

TABLE II
ABLATION STUDY OF COMPONENTS IN *ViFLa*

Smart partition	State transition ring	ECDWS	Test accuracy improvement*		Unlearning speed improvement*	
			iid	non-iid	iid	non-iid
✓	✓	✓	**2% - 3%↑**	**27% - 28%↑**	**≈100x**	**≈120x**
✗	✓	✓	2% - 3% ↑	27% - 28%↑	≈25x	≈30x
✓	✗	✓	2% - 3%↑	27% - 28%↑	≈4x	≈4x

* The improvement is over a modification of *ViFLa* that uses random data partition and weighted average local accuracy (WALAcc), and disables state transition ring.

TABLE III
PERFORMANCE OF DIFFERENT AGGREGATION METHODS BEFORE AND AFTER UNLEARNING MISLABELLED SAMPLES.

Aggregation approach	Accuracy of iid data (%)		Accuracy of non-iid data (%)	
	learning	unlearning	learning	unlearning
WALAcc	86.97	87.32	59.99	59.8
NCDWS	89.54	89.92	83.69	83.99
ECDWS*	89.79	90.02	87.47	87.65

* ECDWS achieves better performance without even knowing the distribution of training data.

2) State Transition Ring: To evaluate the benefit of state transition ring for different LSTM structures, we test the unlearning performance with and without state transition ring. The benefit of state transition ring is reflected at the first and second rows of Table II under different combinations.

3) NCDWS vs. ECDWS: Two types of baselines might be considered. One is parameter-based aggregation method, such as FedAvg [26] that learns a global model by aggregating parameters of local models. The other is weighted average local accuracy (**WALAcc**) that aggregates the predicted accuracy of sub-models based on the amount of training samples used to train the sub-models. WALAcc improves a method in [27] by putting weights on local accuracy. We omit FedAvg because (1) the final (global) model is coupled with all local-models, making FedAvg not suitable for machine unlearning, and (2) FedAvg performs worse than WALAcc in our test.

We set the number of sub-models as 4, and test the performance NCDWS and ECDWS. The class distribution of the data samples generated from IoT-23 dataset benign traffic is uniform. To evaluate the performance of different aggregation methods on iid data, we evenly distribute the training data to the four partitions. To evaluate the performance of non-iid data, we distribute some classes of data samples only to one or two partitions. From the results in Table III, we can see that both NCDWS and ECDWS achieve significant improvements over WALAcc in accuracy for non-iid data. In addition, ECDWS shows clear advantages over NCDWS because ECDWS can achieve similar accuracy without even knowing the distribution of training data.

The above accuracy results only refer to the probability that a sample is correctly predicted. They do not directly translate to the anomaly detection result, which is disclosed below.

F. Performance of Anomaly Detection

A common strategy to test anomaly detection accuracy is to treat a sample as abnormal if its accuracy is lower than a threshold, and then trigger an anomaly alarm if a given percentage of samples in a sliding window are abnormal. We use false positive rate (FPR) and true positive rate (TPR) to evaluate anomaly detection results. The 20% of benign traffic generated by the Philips HUE smart LED lamp is used to test the FPR, and malware traffic is used to test the TPR.

Since *ViFLa* is for model update and must work with an underlying detection method for anomaly detection, we used LSTM as the detection model for the test. The output of each sample in *ViFLa* is a predicted vector $v = v^1, \ldots, v^G$, where v^l denotes the probability that the sample is predicted as class $l (1 \leq l \leq G)$ based on its feature value $x_1, x_2, \ldots, x_F$. Then we take the predicted probability of the actual class of the sample. We choose an anomaly detection threshold to 0.01, and a sample is considered as abnormal if the predicted probability of its actual class l is lower than the detection threshold. If 50% of samples in a sliding window of size of 30 samples are abnormal, an anomaly alarm is triggered. As a result, we got the performance of 100% TPR and no false positives. This excellent performance is no surprise, because the detection model is type-based (i.e., different IoT device types use their corresponding detection model), which has shown excellent detection accuracy as reported in previous work [2].

V. Efficiency Analysis

While it is hard to have a fair comparison between *ViFLa* and existing methods as discussed in Section IV-E, we can perform a theoretical analysis on the efficiency of *ViFLa* to offer more insights on the difference between *ViFLa* and existing methods. For this, we use SISA as the baseline since its components show a similarity to those in *ViFLa*. As rough analogy, its sharding method corresponds to our smart partition (i.e., a shard in SISA corresponds to a virtual client in *ViFLa*), and its slicing method corresponds to our state transition ring. The total samples of all the groups is represented as N.

First, we compare the time cost of sharding in SISA with the time cost of smart partition in *ViFLa*. In sharding, samples are randomly divided into different shards. Thus, the probability that a sample to be unlearned falls on shard k is $\frac{1}{K}$, where K is the number of shards. The expected cost of an unlearning sample is the average number of points to be retrained caused by this unlearning sample, and the expected total cost is the sum of the expected costs of all unlearned samples [28]. Then, the expected total cost of all the unlearning samples is

$$E_1 = \sum_{i=1}^{M}\sum_{j=0}^{i-1}\binom{i-1}{j}(\frac{1}{K})^j(1-\frac{1}{K})^{i-1-j}(\frac{N}{K}-j-1) \quad (8)$$

where M is the total number of samples to be unlearned, and $\frac{N}{K}$ is the number of samples in each shard. To help understand (8), for any given ith unlearning sample, j means the number of previous unlearning samples that fall in the same shard as i. When i is fixed, a greater j means more samples land on the same group.

In contrast, the smart partition in *ViFLa* groups samples of high unlearning beliefs together and builds a local model with these samples. When an unlearning request arrives, the samples included in the request may be relevant to only one or very few local models. The majority of local models do not need to be re-trained. We sort the unlearned samples by their belief values from high to low and divide them into different groups. Therefore, an unlearned sample has a higher probability of falling on the kth group than the $(k+1)$th group $(1 \leq k < K)$, where K is the number of virtual clients. Assume that an unlearned sample falls on the group $(1, 2, \ldots, K)$ with probability $(P_1, P_2, \ldots, P_K)$, respectively, where $\sum_{k=1}^{K} P_k = 1$. Then the expected total cost of all the M unlearned samples is

$$E_2 = \sum_{i=1}^{M}\sum_{j=0}^{i-1}\binom{i-1}{j}\sum_{k=1}^{K}P_k(P_k)^j(1-P_k)^{i-1-j}(\frac{N}{K}-j-1). \quad (9)$$

While it is difficult to rigorously prove that $E_2 \leq E_1$, we run numerical simulation with randomly generated M, N, K, and $P_1, P_2, \ldots, P_K$ values for a million times and found that $E_2 < E_1$ in all the tests; $E_2 = E_1$ only when $P_1 = P_2 = \ldots = P_K$.

Second, we compare the time cost of slicing in SISA and the time cost of state transition ring in *ViFLa*. The total cost of slicing in SISA is $e^{'}\frac{N}{K}(\frac{2}{3}+\frac{1}{3R})$, where $e^{'}$ is number of epochs without slicing and R is the number of slices in the retrained model (refer to [28] for more details). When R is large enough, we obtain the lower bound $\frac{2}{3}\frac{N}{K}e^{'}$, where N is the total number of samples and $\frac{N}{K}$ is the number of samples in each shard. In state transition ring, the total cost is $\hat{e}\frac{N}{BK} = \frac{\hat{e}}{B}\frac{N}{K}$, where $\hat{e}$ is the number of iterations in state transition ring. In our experiment, the value (normally < 5) of $\frac{\hat{e}}{B}$ is much smaller than $\frac{2}{3}e^{'}$ since the new initial state is close to the convergent state.

Third, we compare the storage cost between SISA and *ViFLa*. SISA assumes that there are R slices in each shard k. First, SISA trains the model using the first slice $D_{k,1}$ and save the checkpoint as $M_{k,1}$. Then it trains the model using the first two slices $D_{k,1} \cup D_{k,2}$ and saves the checkpoint as $M_{k,2}$. Hence, it needs to store R model states (checkpoints) for each shard. In its evaluation [28], the slice size is set to 1 since this value needs to be small enough for better unlearning performance. Nevertheless, a very small slice size will make the number of slices R in each shard close to the number of samples $\frac{N}{K}$ in each shard. So SISA needs to store $\frac{N}{K}$ model states for each shard. Overall, the total number of model states that need to be stored in SISA is N since there are K shards.

In contrast, for each group k in *ViFLa*, we just need to store a state for each node and there are B (B is the number of batches in each group) nodes in the state transition ring. Therefore, our method needs to store B model states for each group. Since we have K groups, the total number of states that need to be stored in *ViFLa* is BK. Clearly, $BK < N$ because $N = BKT$ where $T(> 1)$ is the batch size. In conclusion, *ViFLa* incurs a smaller storage cost than SISA.

VI. Conclusion

Machine unlearning, a technique that quickly updates a machine learning model and removes the impact of a small portion of training data on the model, is much needed in IoT traffic anomaly detection. The requests for machine unlearning may come from different scenarios, e.g., some training data may be mislabeled due to unknown attacks or an IoT device upgrades its firmware and thus changes partial data that have been used in the model training. We proposed a new solution, *ViFLa*, that leverages the concept of virtual federated learning and consists of three new mechanisms, smart partition, enhanced class distribution weighted sum (ECDWS), and state transition ring, to achieve efficiency and completeness of machine unlearning. Using real-world trace data, we thoroughly evaluate the performance of *ViFLa*, covering not only the effectiveness of its individual components but also its benefit in different application scenarios. Overall, *ViFLa* can achieve similar accuracy of re-training from scratch with significant speedup. We also performed a theoretical analysis of the efficiency of *ViFLa* using SISA as a baseline. Compared to the baseline, *ViFLa* can reduce the computational complexity and the amount of data affected by unlearned samples.

References

[1] S. Marchal, M. Miettinen, T. D. Nguyen, A.-R. Sadeghi, and N. Asokan, "Audi: Towards autonomous iot device-type identification," *IEEE Jour-*

nal on Selected Areas in Communications (JSAC) on Artificial Intelligence and Machine Learning for Networking and Communications, 2019.

[2] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "Dïot: A federated self-learning anomaly detection system for iot," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 756–767.

[3] M. Du, Z. Chen, C. Liu, R. Oak, and D. Song, "Lifelong anomaly detection through unlearning," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1283–1297.

[4] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 463–480.

[5] L. Bourtoule, V. Chandrasekaran, C. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," *arXiv preprint arXiv:1912.03817*, 2019.

[6] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[7] C.-T. Chu, S. Kim, Y.-A. Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Ng, "Map-reduce for machine learning on multicore," *Advances in neural information processing systems*, vol. 19, pp. 281–288, 2006.

[8] A. Parmisano, S. Garcia, and M. Erquiaga, "Stratosphere laboratory. a labeled dataset with malicious and benign iot network traffic," January, 2020.

[9] A. Golatkar, A. Achille, and S. Soatto, "Eternal sunshine of the spotless net: Selective forgetting in deep networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9304–9312.

[10] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage." in *NIPs*, vol. 2. Citeseer, 1989, pp. 598–605.

[11] A. Ginart, M. Y. Guan, G. Valiant, and J. Zou, "Making ai forget you: Data deletion in machine learning," *arXiv preprint arXiv:1907.05012*, 2019.

[12] T. Baumhauer, P. Schöttle, and M. Zeppelzauer, "Machine unlearning: Linear filtration for logit-based classifiers," *arXiv preprint arXiv:2002.02730*, 2020.

[13] N. Aldaghri, H. Mahdavifar, and A. Beirami, "Coded machine unlearning," *IEEE Access*, 2021.

[14] J. Brophy and D. Lowd, "Machine unlearning for random forests," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1092–1104.

[15] S. Neel, A. Roth, and S. Sharifi-Malvajerdi, "Descent-to-delete: Gradient-based methods for machine unlearning," *arXiv preprint arXiv:2007.02923*, 2020.

[16] T. Bayes, "Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s," *Philosophical transactions of the Royal Society of London*, no. 53, pp. 370–418, 1763.

[17] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[18] Z. Harry, "The optimality of naive bayes," in *FLAIRS2004 conference*, 2004.

[19] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2002.

[20] A. A. Cook, G. Mısırlı, and Z. Fan, "Anomaly detection for iot time-series data: A survey," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, 2019.

[21] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[22] I. Csiszár, "I-divergence geometry of probability distributions and minimization problems," *The annals of probability*, pp. 146–158, 1975.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[24] S. Zhang, Y. Tay, L. Yao, and A. Sun, "Next item recommendation with self-attention," *arXiv preprint arXiv:1808.06414*, 2018.

[25] G. H. Rosenfield and K. Fitzpatrick-lins, "A coefficient of agreement as a measure of thematic classification accuracy." *Photogrammetric Engineering and Remote Sensing*, vol. 52, pp. 223–227, 1986.

[26] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[27] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *arXiv preprint arXiv:2002.06440*, 2020.

[28] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 141–159.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[30] R. G. Bartle and D. R. Sherbert, *Introduction to real analysis*. Wiley New York, 2000, vol. 2.

Appendix

LSTM Summation Form: LSTM is a type of recurrent neural network (RNN) architecture [29]. Each LSTM neuron has four inputs: the original input x, the input gate that controls if x could be written into the memory cell, the output gate that decides if the value of the current neuron is accessible to the other neurons, and the forget gate that decides what time to forget the content in the memory cell. We train an LSTM of three layers with mini-batches (sample size = 128), and the dimension of its input feature is 20. Following [29], the network is modelled as:

$$\begin{aligned}
f_t &= \sigma(W_f * [h_{t-1}, x_t] + b_f)\\
i_t &= \sigma(W_i * [h_{t-1}, x_t] + b_i)\\
\tilde{C}_t &= \tanh(W_C * [h_{t-1}, x_t] + b_C)\\
C_t &= f_t \circ C_{t-1} + i_t \circ \tilde{C}_t\\
o_t &= \sigma(W_o * [h_{t-1}, x_t] + b_o)\\
h_t &= o_t \circ \tanh C_t
\end{aligned}$$

where $W_f, W_i, W_C, W_o \in \mathbb{R}^{128\times(20+128)}$, $b_f, b_i, b_C, b_o \in \mathbb{R}^{128}$, 128 is the number of hidden units, and 20 is the dimension of the input feature. Each parameter state consists of weights and bias of three LSTM layer and two fully connected layer. The initial parameter state of the network θ_{S_0} is the parameter state stored in node n_0 of the state transition ring in Fig. 5. At each state, a mini-batch (128 samples) $batch_b$ is input to the network and estimate the batch loss if $b = j$ (refer to Eq. (3)). We backward propagate this batch loss and calculate the gradient $g^i_{\theta_{S_0}(batch_b)}$ for each parameter i and use this gradient as the mapping result. The summation form of the LSTM network is the sum of mapping results of all the batches at state θ_{S_j}, i.e., $\sum_b f_{\theta_{S_j}}(batch_b)$.

Proof of the convergence of KL-attention: We provide a sketch proof. The proof is based on the concept of Cauchy sequence. The vector update rule defined by Eq. (7) is equivalent to a mapping function $\mathcal{T}(v_j) = \sum_{i=1}^{K} \sigma(\mathcal{R}(j,i))v_i$. We use $||.||$ to denote the vector norm. For $\forall j \in \{1,\ldots,K\}$, we can calculate the difference between v_j and $\mathcal{T}(v_j)$ as:

$$||v_j - \mathcal{T}(v_j)|| = ||v_j - \sum_{i=1}^{K} \sigma(\mathcal{R}(j,i))v_i||$$

$$= ||\sum_{i\neq j, i=1}^{K} \sigma(\mathcal{R}(j,i))(v_j - v_i)||$$

We slightly abuse the notation by using $\boldsymbol{v}$ to represent the set of vectors $\boldsymbol{v} = \{v_1, v_2, \ldots, v_K\}$ and $\mathcal{T}(\boldsymbol{v})$ to represent the set vectors $\mathcal{T}(\boldsymbol{v}) = \{\mathcal{T}(v_1), \mathcal{T}(v_2), \ldots, \mathcal{T}(v_K)\}$. We define the

distance between $\boldsymbol{v}$ and $\mathcal{T}(\boldsymbol{v})$ as: $d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) = \sum_{j=1}^{K} ||v_j - \mathcal{T}(v_j)|| = \sum_{j=1}^{K} || \sum_{i \neq j, i=1}^{K} \sigma(\mathcal{R}(j,i))(v_j - v_i)||$.
Similarly, we can obtain the distance $d(\mathcal{T}(\boldsymbol{v}), \mathcal{T}^2(\boldsymbol{v}))$ as follows:

$$\begin{aligned} & d(\mathcal{T}(\boldsymbol{v}), \mathcal{T}^2(\boldsymbol{v})) \\ =& \sum_{j=1}^{K} || \sum_{i \neq j, i=1}^{K} \sigma(\mathcal{R}'(j,i))(\mathcal{T}(v_j) - \mathcal{T}(v_i))|| \end{aligned}$$

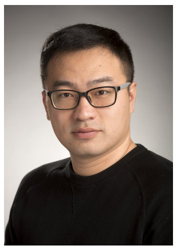
where $\mathcal{T}^2$ means applying the mapping function again, i.e., the second-round update with Eq. (7). The mapping function of a vector is the weighted average of all the vectors, and the similarity of the vector and other vectors will become smaller after mapping $\mathcal{T}$. So the distance $d(\mathcal{T}(\boldsymbol{v}), \mathcal{T}^2(\boldsymbol{v})) \leq d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v}))$. There must exists a number $a (0 < a < 1)$ that makes $d(\mathcal{T}(\boldsymbol{v}), \mathcal{T}^2(\boldsymbol{v})) \leq a * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v}))$.

By induction and by triangle inequality, we have $d(\mathcal{T}^n(\boldsymbol{v}), \mathcal{T}^m(\boldsymbol{v})) \leq d(\mathcal{T}^n(\boldsymbol{v}), \mathcal{T}^{n+1}(\boldsymbol{v})) + \ldots + d(\mathcal{T}^{m-1}(\boldsymbol{v}), \mathcal{T}^m(\boldsymbol{v})) \leq a^n * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) + a^{n+1} * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) + \ldots + a^{m-n-1} * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) = a^n * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) * \sum_{k=0}^{m-n-1} a^k$. So $d(\mathcal{T}^n(\boldsymbol{v}), \mathcal{T}^m(\boldsymbol{v})) \leq a^n * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) * \sum_{k=0}^{n-m-1} a^k = a^n * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) * \frac{a^{m-n}}{1-a} < \frac{a^n * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v}))}{1-a}$ for all $n < m$. For any $\epsilon > 0$, we can choose an N satisfying $\frac{a^N * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v}))}{1-a} < \epsilon$. Then for $N \leq n \leq m$, $d(\mathcal{T}^n(\boldsymbol{v}), \mathcal{T}^m(\boldsymbol{v})) < \epsilon$.

So **the sequence $\{\boldsymbol{v^n} = \mathcal{T}^n(\boldsymbol{v})\}_{n=1}^{\infty}$ is a Cauchy sequence.** Convergence is proved due to the Bolzano–Weierstrass theorem [30] that "each bounded sequence in $\mathbb{R}^n$ has a convergent sub-sequence", and a Proposition [30] that "if a sub-sequence of a Cauchy sequence converges to x, then the sequence itself converges to x."

Jiamin Fan received her B.S. degree in Telecommunications engineering from Nanjing University of Posts and Telecommunications, China, in 2014, and the M.S. degree in computer science from University of Victoria, Canada in 2018. She is currently a Ph.D. student in the Computer Science Department, University of Victoria, Canada. Her research interests include networking, machine learning, and distributed computing.

Kui Wu (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science from Wuhan University, Wuhan, China, in 1990 and 1993, respectively, and the Ph.D. degree in computing science from the University of Alberta, Edmonton, AB, Canada, in 2002. He joined the Department of Computer Science, University of Victoria, Canada, in 2002, and is currently a Full Professor there. His current research interests include network performance analysis, online social networks, Internet of Things, and parallel and distributed algorithms.



Yang Zhou received the B.S. degree in Electronic Engineering from Jilin University, Changchun, China, the M.S. degree in Electronic Engineering from Peking University, Beijing, and the M.S. degree in Computer Science from Simon Fraser University, Burnaby, Canada. He is currently a senior data scientist and a cyber security researcher. His main research interests include malware detection and classification with machine learning techniques and attacks against operation systems and networks.

Zhengan Zhao received the B.Sc. degree in Computer Science from Miami University, Ohio, USA, in 2020, and the M.Sc. degree also in Computer Science from the University of Victoria, BC, Canada, in 2022. He is currently a Ph.D. student in the Department of Computer Science, University of Victoria, Canada. His research interests include IoT network anomaly detection and access control.

Shengqiang Huang is currently a Senior Principal Engineer with Huawei Technologies Canada Company Ltd. He has more than 20 years of extensive experience in the network security industry, reverse engineering, video processing, financial trading system, and general software architecture with roles that span across organization and business maturity. His specialties are anti-virus engine development, intrusion prevention systems, sandboxing, industrial control systems security, APT/malware, NGFW, application layer security, and UTM. He is leading the anti-virus and sandbox product research in Huawei. Prior to joining Huawei, he was the Lead Architect and held the role of key research and development at Wurldtech Securities, a Canadian industrial control systems security start-up acquired by General Electric (GE) in 2014. Prior to Wurldtech, he was a Senior Software Developer of Fortinet anti-virus engine core team. Before Fortinet, he had held various product management and development roles.