Verifying in the Dark: Verifiable Machine Unlearning by Using Invisible Backdoor Triggers

Yu Guo, *Member, IEEE*, Yu Zhao, Saihui Hou, *Member, IEEE*, Cong Wang, *Fellow, IEEE*, and Xiaohua Jia, *Fellow, IEEE*

***Abstract*— Machine unlearning as a fundamental requirement in Machine-Learning-as-a-Service (MLaaS) has been extensively studied with increasing concerns about data privacy. It requires MLaaS providers should delete training data upon user requests. Unfortunately, none of the existing studies can efficiently achieve machine unlearning validation while preserving the retraining efficiency and the service quality after data deletion. Besides, how to craft the validation scheme to prevent providers from spoofing validation by forging proofs remains under-explored. In this paper, we introduce a backdoor-assisted validation scheme for machine unlearning. The proposed design is built from the ingenious combination of backdoor triggers and incremental learning to assist users in verifying proofs of machine unlearning without compromising performance and service quality. We propose to embed invisible markers based on backdoor triggers into privacy-sensitive data to prevent MLaaS providers from distinguishing poisoned data for validation spoofing. Users can use prediction results to determine whether providers comply with data deletion requests. Besides, we incorporate our validation scheme into an efficient incremental learning approach via our index structure to further facilitate the performance of retraining after data deletion. Evaluation results on real-world datasets confirm the efficiency and effectiveness of our proposed verifiable machine unlearning scheme.**

***Index Terms*— Machine unlearning, ML-as-a-service, backdoor attacks, incremental learning.**

I. Introduction

RECENT advances in the fields of both Machine Learning (ML) and cloud computing have facilitated the widespread adoption of ML-as-a-service (MLaaS) [1]. Unlike traditional ML, MLaaS allows resource-restricted users to collaboratively delegate complicated ML training tasks to a cloud-based service provider (*SP*) and later access the well-trained ML model in a black-box manner. Despite being promising, MLaaS has also raised security concerns [2], [3], [4], [5]. This is because training models via MLaaS can lead to an untrusted *SP* collecting large amounts of privacy-sensitive data from users, raising the risk of private information leakage.

Given the existence of potentially malicious *SP* [6], the "right to be forgotten" (RTBF) regulations [7], [8] and the related notion of machine unlearning [9] have been proposed to protect the right of users to forget their private data when using MLaaS. In particular, machine unlearning requires that *SP* should delete specified training data upon user requests while preserving high-quality MLaaS after model retraining. Although deletion and retraining with the remaining data is a possible solution, most ML training tasks in MLaaS are computationally expensive. It is infeasible to retrain the model from scratch by *SP* every time there is a new deletion request.

In the literature, several recent works [10], [11], [12], [13], [14], [15], [16], [17] have investigated practical machine unlearning solutions, and they can be roughly categorized into two groups: *approximate unlearning* and *exact unlearning*. The first class of solutions [10], [11], [12], [13], [14] relies on customized optimization methods in deep learning with a tradeoff on accuracy. For instance, Ma et al. [14] devised a machine unlearning via neuron masking with an accuracy loss of approximate 5%. Although heuristic solutions have been proposed to improve the performance of machine unlearning, these approximate unlearning designs cannot ensure that the *SP* accurately removes the data samples, and thus are not yet ready for practical machine unlearning deployment.

The second class of solutions aims to accurately remove the data samples, and thus better match the practical requirements of MLaaS. Cao et al. [15] first employed statistical query learning to train the model so that the data samples can be effectively forgotten by recalculating the statistics of the remaining data. The follow-up design [16] utilized the method of data slicing to further enhance the efficiency of machine unlearning. Liu et al. [17] proposed a rapid retraining method using Quasi-Newton methods for unlearning in federated learning. Unfortunately, those existing studies only consider the idealized service scenario where an honest *SP* would faithfully handle data deletion requests from users. In practice, the *SP* are usually third-party servers deployed in the cloud, which cannot be fully trusted. It may not comply with the deletion requests or bypass the unlearning detection by forging validation proofs [18], due to various reasons such as

Manuscript received 24 June 2023; revised 20 September 2023; accepted 19 October 2023. Date of publication 27 October 2023; date of current version 22 November 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62102035 and Grant 62206022; in part by the Research Grants Council of Hong Kong under Grant N_CityU139/21, Grant C2004-21G, Grant R1012-21, Grant R6021-20F, and Grant CityU 11213920 (GRF); in part by the Fundamental Research Funds for the Central Universities under Grant 2021NTST31; and in part by the National Key Research and Development Program of China under Grant 2022ZD0115901. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. George Theodorakopoulos. *(Corresponding author: Saihui Hou.)*

Yu Guo, Yu Zhao, and Saihui Hou are with the School of Artificial Intelligence, Beijing Normal University, Beijing 100875, China (e-mail: yuguo@bnu.edu.cn; zhaoyu2022@mail.bnu.edu.cn; housaihui@bnu.edu.cn).

Cong Wang and Xiaohua Jia are with the Department of Computer Science, City University of Hong Kong, Hong Kong, China (e-mail: congwang@cityu.edu.hk; csjia@cityu.edu.hk).

Digital Object Identifier 10.1109/TIFS.2023.3328269

hacking or corporate dishonesty. To address this issue, a recent work [19] proposed to use trusted hardware to enforce proof of unlearning, but relying on the existence of trusted execution environments at the *SP*-side. Therefore, it is imperative to provide an efficient and verifiable machine unlearning scheme that ensures the reliability of the validation process to avoid dishonest *SP* from bypassing the validation.

In this work, we propose an efficient and verifiable machine unlearning scheme for MLaaS that enforces an untrusted *SP* to accurately execute data deletion while preserving the quality of model training after machine unlearning. To achieve our design goals on both verifiability and usability, we resort to the design philosophy of backdoor attacks and ingenious synergy with the practical incremental learning approach. Specifically, we start from backdoor attack strategies against deep learning and explore the design of invisible markers based on backdoor triggers to construct our practical machine unlearning validation schemes. The basic idea is to embed unique invisible markers into privacy-sensitive data samples before outsourcing, so that the predictions of these poisoned data should be pre-defined target labels by users instead of source labels. Thus, our design can assist users in validating the proof of machine unlearning based on prediction results without compromising the quality of the training model. Besides, we incorporate the validation schemes into the incremental learning approach to further improve the performance of machine unlearning. We propose a data-model index structure that indicates data slices based on privacy levels to assist *SP* in training intermediate models efficiently after data deletion rather than retraining them from scratch. We provide the open-source code[1] of our implementation and conduct comprehensive experiments over representative datasets (i.e., MNIST, CIFAR10, and GTSRB). The evaluation results show that our design achieves practical performance for MLaaS. We believe that our backdoor-assisted validation scheme can provide fresh insights in designing reliable MLaaS applications. In summary, our contributions are listed as follows:

- We present a backdoor-assisted validation scheme that users can efficiently verify proofs of machine unlearning while preserving the high-quality of the retrained model. By utilizing invisible markers based on backdoor triggers, our design can effectively prevent dishonest *SP* from forging proofs to bypass the validation.
- We explore the incremental learning approach and integrate it with our machine unlearning validation scheme to improve the computational efficiency of model retraining after data deletion.
- We implement a system prototype and perform an extensive evaluation over representative datasets (MNIST, CIFAR10, and GTSRB). The experimental results show a significant improvement in the performance of our design compared to traditional solutions.

The remainder of this paper is organized as follows. Section II introduces the background knowledge of the backdoor attack and incremental learning. Section III presents the system architecture and threat model. Section IV provides the concrete construction of our design. The experiments and performance evaluation are described in Section V. Section VI shows the related work and Section VII concludes the whole paper and points out the future work.

[1]System prototype, online at: https://github.com/Junxin-L/PoUL

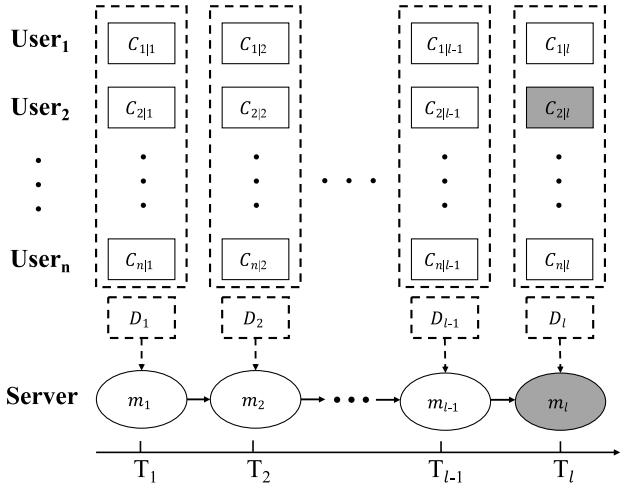


Fig. 1. Incremental learning in machine unlearning.

II. Background

A. Incremental Learning

Incremental learning refers to a learning system that continuously learns new knowledge from new data and preserves most of the previously learned knowledge. In this work, we focus on the method of class incremental learning to train our model, where the method is based on the learning without forgetting (*LWF*) algorithm proposed by Li et al. [20]. In the initialization phase of *LWF*, the output of new data on the old model is calculated, and the parameters of the new model are randomly initialized. Then, the *LWF* algorithm continuously trains the new model with the goal of minimizing the loss function values of both the old and the new model. Thus, the new model can learn the new data while retaining the memory of the old data.

For MLaaS, the data is always available during the training process. To ensure that the new model retains the memory of the old data, we extract a small portion of the old data for the training of the new model. Fig. 1 presents an example of our service workflow, describing the detailed steps of machine unlearning in incremental learning:

1) Each user slices the dataset based on the level of content privacy (e.g., $C_{1|1}, C_{1|2}, \ldots, C_{1|l}$) and sends the slicing results to the server provider *SP*.
2) *SP* organizes the data from each user in slice order and puts the data of the same privacy level together (e.g., $D_1, D_2, \ldots, D_l$).
3) *SP* trains m_1 with D_1 at time T_1, and then adds data D_2 to train m_2 on the basis of m_1 until the final model m_l contains the features of all the data at time T_l.
4) The user sends a request of deleting the data point $C_{2|l}$ to *SP*.
5) *SP* deletes the data point $C_{2|l}$ and uses the updated D_l without $C_{2|l}$ to retrain m_l on the basis of m_{l-1}.

B. Backdoor Attack

Backdoor attacks [21], [22], [23], [24], [25] are critical security problems in deep learning. The attacker injects backdoor triggers into the data during the training process. Once the embedded backdoor triggers are activated, the output of the model would become the target label pre-specified by the attacker. It should be noted that backdoor attacks can only change the prediction of samples with triggers, and have no effect on these clean samples. Thus, there are two main evaluation metrics in the literature on backdoor attacks, i.e., the backdoor attack success rate (*BASR*) and the clean sample accuracy (*CSA*).

Following the representative work [26] on backdoor attacks, we formulate the main processes as follows: given a data point $d = (x, y)$ consisting of a sample x and a source label y, the backdoor attack poisons d with a one-to-one mapping $x^* = F(x, p, t)$, and then marks x^* as the target label t. Here, the operation F is used to apply the trigger p into the input x, resulting in the poisoned data point $d^* = (x^*, t)$. Mathematically, the model is optimized according to Eq. 1 during training with the dataset that contains both clean and poisoned data:

$$\theta^* = \arg\min_{\theta} \left(\sum_{i=1}^{n} \mathcal{L}(L_\theta(x_i), y_i) + \sum_{j=1}^{m} \mathcal{L}(L_\theta(x_j^*), t_j) \right) \quad (1)$$

where θ and θ^* denote the model before and after optimization respectively, and L is the learning algorithm. The sizes of the clean and poisoned samples used for optimization are n and m. $d_i = (x_i, y_i)$ represents the i-th sample in clean samples, and $d_j^* = (x_j^*, t_j)$ denotes the j-th sample in poisoned samples. The first item of optimization seeks to minimize the loss function $\mathcal{L}$ for the clean samples. The second item of optimization aims to minimize the loss function $\mathcal{L}$ of the backdoor attack.

In this work, we utilize the backdoor triggers in [26] as the building block to construct our invisible markers for machine unlearning validation. The core idea is to use the least significant bit (*LSB*) algorithm to inject triggers into the poisoning data. Specifically, the *LSB* algorithm first converts the source image and the trigger into 8-bit binary form. The last bit of each pixel is replaced with the trigger bit-by-bit. If the length of the trigger exceeds the number of pixels in the image, it should modify the next least significant bit (i.e., the penultimate bit) of the image.

A detailed example is shown in Fig. 2, where the image sample consists of 784 pixels (i.e., 28×28). If a user uses "Alice$''$ $\times 100$ (i.e., "AliceAlice...ice$''$) as a backdoor trigger, then the user needs to convert each character of the text trigger into the form of an 8-bit binary (e.g., the binary form of ASCII code corresponding to "A$''$ is 01000001). Thus, the user gets a bit string trigger consisting of 0 and 1 with length $4,000$ (i.e., $5 \times 100 \times 8$). The first pixel in the sample shown in Fig. 2 is 0, and its binary form is 00000000. Then, each bit of the trigger is used to replace the last bit of each pixel in order. For instance, the second bit of the trigger 1 is used to replace the last bit 0 of the second pixel, so that the value of the second pixel of the image changes from 00000000 to 00000001 (i.e., value 0 to 1). It is worth noting that for an 8-bit binary value, the least significant bit can be extended up to the least 4 significant bits. Embedding the trigger into the last 4 bits of the pixel can achieve a backdoor attack that is invisible to human eyes instead of neural networks. The number of pixels in Fig. 2 is 784, and the length of the trigger $4,000$ is 4 times longer than the number of pixels in the image (i.e., $4000 > 4 \times 784$). Thus, the user needs to carry out 4 rounds of replacement operations for each pixel in the image. That is, the last 4 bits of each pixel should be replaced with the trigger. The effect of adding a trigger with the LSB algorithm is shown in Fig. 3. In particular, Fig. 3(a) and Fig. 3(b) represent the clean image and the poisoned image after injecting an invisible trigger, respectively. The difference between the clean image and the poisoned image is displayed in Fig. 3(c), which also represents the backdoor trigger we used in this work.

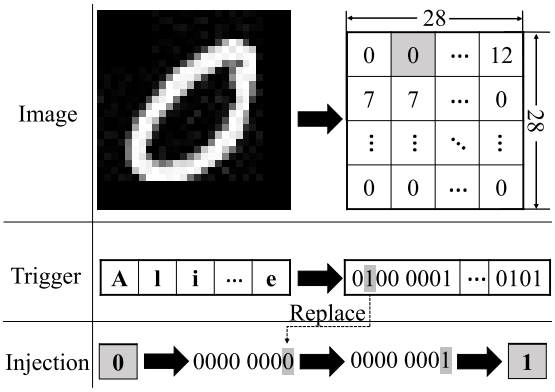
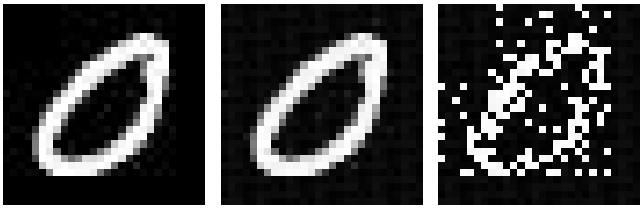


Fig. 2. Example of the *LSB* algorithm.



(a) Clean image (b) Poisoned image (c) Trigger

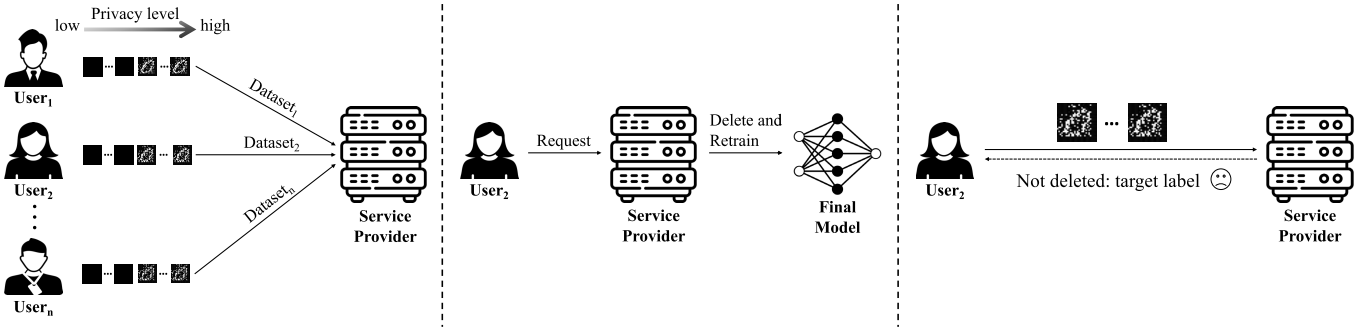
Fig. 3. Example of backdoor triggers.

III. System Model

A. Architecture Overview

Our design focuses on verifiable machine unlearning in the context of MLaaS. Fig. 4 illustrates our system architecture, containing two main types of entities: users and the *SP* responsible for training models. Users are the owners of the data who outsource their data to the *SP* in order to collaboratively obtain a model for data prediction services. Besides, our design should be able to allow users to delete their training data from the *SP*. While the *SP* is responsible for training data from all users and responding to data deletion requests. Overall, the workflow of our proposed design can be summarized in the following three main steps:

Step 1: Backdoor injection and model training. Fig. 4(a) illustrates the initialization of our design. To facilitate users



(a) Backdoor injection and model training (b) Data deletion and model retraining (c) Machine unlearning validation

Fig. 4. System architecture.

can effectively verify the execution of deletion requests, each user should invisibly add markers to a part of its data (i.e., poisoned data) before sending the data to the *SP*. Here, the original data should be ranked according to the level of privacy, with less private data in the front and more private data in the back, as shown in Fig. 4(a). Then, users add unique markers to their original data with a high privacy level and change the source labels of these data to target labels. After receiving the data from users, the *SP* can execute class incremental training on the data of all users according to the ranking order of each user to obtain a final model.

Step 2: Data deletion and model retraining. After the system initialization is completed, the poisoned data has been stored on the *SP* side. The user can later delete their data from the *SP* after model training, as shown in Fig. 4(b). Since the *SP* records the intermediate models during the process of training data, the *SP* can efficiently retrain the model starting from the corresponding breakpoints instead of training from scratch. Here, the new final model obtained by retraining does not contain the data features that the user requested to be removed. While an untrusted *SP* may not comply with the data deletion request from the user, this can be verified in the next stage.

Step 3: Machine unlearning validation. As mentioned in *Step 1*, the private data has embedded with backdoor triggers, which can assist the user to detect whether the *SP* has complied with the data deletion request. As shown in Fig. 4(c), once the *SP* has trained the model with these poisoned data, the model would contain the features of the backdoor triggers. Thus, it can be used for users to detect whether the *SP* has deleted the corresponding data upon request. Specifically, the user sends the previously deleted poisoned data to the *SP* for conducting the prediction service. If the *SP* honestly complies with the deletion request from the user, the predictions of the poisoned data returned by the *SP* should be the source labels. Otherwise, the predictions of the poisoned data returned by the *SP* are target labels set by the user. Thus, users can efficiently detect proof of machine unlearning in a provable manner.

B. Threats Model

We consider the potential security threat in our design mainly comes from the untrusted *SP*. In particular, the *SP* will faithfully follow the MLaaS specification for its service rewards, yet it might not comply with the data deletion request from the user. Note that such a threats model follows most of the prior works in the machine unlearning [27], [28]. We also realize that this assumption is reasonable because the *SP* as a commercial party is motivated to provide faithful MLaaS for revenue, but will ignore additional deletion requests from users to save computational resources. We consider that the users are trusted, and can only access the final model trained by the *SP* in a black-box manner. Thus, the problem of machine unlearning we are investigating is for the users to verify that the *SP* has deleted the data and retrained the model.

IV. The Proposed Design

A. Design Overview

Our work concentrates on the challenges of verifiability of machine unlearning and efficiency of model retraining. The research aims to support verifiable machine unlearning requirements for ensuring the private data deletion rights of users while considering the efficient performance of model retraining. According to the illustration in Section I, utilizing the design principles of backdoor attacks to devise the desired validation scheme is a promising solution, but several fundamental challenges are yet to be addressed. The first challenge is to effectively prevent *SP* validation spoofing and maintain the accuracy of model retraining after machine unlearning. If the private data embedded with markers can be distinguished by the *SP*, then the *SP* can perform validation spoofing. That is, the *SP* may remove the markers of the poisoned data using methods such as Neural Cleanse [29], which is a method of backdoor detecting and mitigating. Besides, due to the training dataset being provided by different users, it is necessary to ensure the uniqueness of markers for different users so that to maintain the accuracy of model retraining after machine unlearning. The other challenge is to find out an efficient mechanism for improving model retraining efficiency. Since most model training requires significant computational costs, the straightforward solution of simply retraining the whole updated model from scratch inevitably leads to additional costs and affects the efficiency of model retraining.

To address the challenges above, we resort to the *LSB* algorithm [26] as the starting point to craft the verifiable machine unlearning scheme. More precisely, the *LSB* algorithm can covertly replace the source image with the trigger bit-by-bit so that it can only be distinguished by

TABLE I
NOTATIONS

Notation	Description	Domain
u	Data owners in MLaaS	$\{1, n\}$
p_u	The unique marker of user u	$\{p_1, p_n\}$
r_u	The random nonce generated by u	$\{r_1, r_n\}$
$C_{u\|i}$	The i-th data slice of user u	$\{C_{u\|1}, C_{u\|l}\}$
b	The privacy threshold	N/A
$d_{u\|j}$	The j-th data point of slice $C_{u\|i}$	$\{d_{u\|1}, d_{u\|m}\}$
x_j	The data sample of data point $d_{u\|j}$	$\{x_1, x_m\}$
y_j	The source label of data point $d_{u\|j}$	$\{y_1, y_m\}$
$C^*_{u\|i}$	The poisoned $C_{u\|i}$	$\{C^*_{u\|1}, C^*_{u\|l}\}$
$d^*_{u\|j}$	The j-th poisoned data point of $C^*_{u\|i}$	$\{d^*_{u\|1}, d^*_{u\|m}\}$
x^*_j	The poisoned sample of $d^*_{u\|j}$	$\{x^*_1, x^*_m\}$
t_j	The target label of $d^*_{u\|j}$	$\{t_1, t_m\}$
m_i	The intermediate model	$\{m_1, m_l\}$
D_i	The data with privacy level i	$\{D_1, D_l\}$
w	The character in each trigger	$\{1, \|p\|\}$
$\tilde{w}$	The binary form of w	$\{1, \|\tilde{p}\|\}$
v	The pixel in each sample x	$\{1, \|x\|\}$
$\tilde{v}$	The binary form of v	$\{1, \|x\|\}$
$\tilde{v}^*$	The poisoned pixel v	$\{1, \|x\|\}$
$\dot{v}$	The decimal form of $\tilde{v}^*$	$\{1, \|x\|\}$
R_u	The data deletion request from u	$\{R_1, R_n\}$
n'	The number of users with R_u	$n' \leq n$
$T_{u\|i}$	The test data corresponding to $C_{u\|i}$	$\{T_{u\|1}, T_{u\|l}\}$
e	The prediction label of $T_{u\|i}$	N/A
q	The validation result from the user	$(0, 1)$

neural networks instead of the *SP*. Using the *LSB* algorithm to construct invisible markers and embed them to the outsourced data can effectively prevent the *SP* from detecting poisoned data while preserving high-quality model training. Besides, we further propose to generate unique markers with random nonces for each user to prevent the inaccuracy of the validation process caused by trigger collision during machine unlearning. Thus, each user can efficiently verify whether its marker exists in the final model to determine whether the *SP* complies with the data deletion request. Regarding the efficiency of model retraining after data deletion, our design adopts incremental learning as the underlying mechanism to improve performance. Specifically, we use the *LWF* algorithm [20] to incrementally train data slices and save the intermediate models during training. In our design, the data should be sorted according to the privacy level, establishing a data-model index structure as shown in Fig. 1. Thus, the training process is traceable, and after deleting a data slice, the *SP* only needs to retrain the corresponding intermediate models affected by that data slice to obtain the final model, rather than retraining from scratch.

The detailed backdoor-assisted validation scheme for machine unlearning is described in the following sections. To better understand the detailed design, we provide a summary of the main notations in Table I.

B. The Verifiable Machine Unlearning Scheme

1) Backdoor Injection and Model Training: To support efficient and verifiable machine unlearning, each user needs to slice their own dataset according to the level of privacy and inject its markers covertly into the data with high privacy. Then, the *SP* executes the model training with these poisoned data. In Alg. 1, we provide a detailed process for backdoor injection and model training.

(a) *User-side setup.* To enable verifiable machine unlearning, the high-level privacy data is embedded with invisible markers before outsourcing to the *SP*. To avoid trigger collisions, each user u should first initialize its unique markers p_u by using a secure pseudo-random function (PRF) P and hash function H, i.e., $p_u = P(k_u, H(u||r_u))$, where k_u is the private key of the user u and r_u is a random nonce. Then, all data slices for the user u are sorted by privacy level, with the less private data in front and more private data in the back (i.e., $\{C_{u|1}, C_{u|2}, \ldots, C_{u|l}\}$). After that, the user should evaluate the privacy level based on the function G and inject the appropriate markers by using the *LSB* algorithm described in Section II-B. If the privacy level of the slice $C_{u|i}$ exceeds the threshold b, user u should inject a marker into m data points. For the j-th data point $d_{u|j} = (x_j, y_j)$, the user u adds its defined marker p_u to its sample x_j by using the backdoor function *LSB*, as shown in Alg. 2. The user first needs to use the binary function *Bin* to convert each character of the trigger to an 8-bit binary number, so that the length of the trigger is extended to 8 times. The user then converts the sample x into $|x|$ pixels and uses the same method (i.e., *Bin*) to convert each pixel into an 8-bit binary number. A long enough trigger should be generated to launch a backdoor attack, and the substitution function *Sub* should be used to replace the last bit of each pixel in the sample x. Thus, the user replaces as many least significant bits of each pixel as possible to ensure the marker is invisible to the *SP* rather than the neural network. When the length of the marker is greater than $|x|$, the penultimate of each pixel is replaced using the substitution function *Sub* until the last 4 bits of each pixel have been replaced or the marker has been used up. The binary pixels are then converted to the form of decimal by the decimal function *Dec* and a new poisoned sample x^* is produced. After that, the user changes the source label y_j of the data point $d_{u|j}$ to the target label t_j, which results in a poisoned data point $d^*_{u|j}$. Finally, these points of poisoned data can be used to form a new poisoned data slice $C^*_{u|i}$, which is sent to the *SP* for training models.

(b) *SP-side setup.* The *SP* needs to first initialize the basic model m_0 to train the following models. As mentioned in Fig. 1, all the slice $C_{u|i}$ from each user u should be organized as D_i, where $i \in \{1, l\}$. Then, the *SP* trains the model m_i with D_i based on the previous model m_{i-1} until the final model m_l is obtained. During the training process, the *SP* needs to save all intermediate models m_i. The model optimization method used by the *SP* is shown in Eq. 1, which aims to achieve both high *BASR* and *CSA*. After completing all training, the *SP* needs to send a training completion response to each user u to inform users that the model training has been

Algorithm 1 Backdoor Injection and Model Training

Input: User $u \in \{1, n\}$, the trigger p_u, secure PRF P, private key k_u, random nonce r_u, the evaluation function G with the privacy threshold b, the backdoor function *LSB*, the target label t_j, and the initial model m_0.

Output: The final model m_l and the training completion response.

User-side:

for each user $u \in \{1, n\}$ **do**
  Initialize the trigger $p_u = P(k_u, H(u||r_u))$;
  Sort $\{C_{u|1}, C_{u|2}, \ldots, C_{u|l}\}$ based on privacy level;
  for each $i \in \{1, l\}$ **do**
    if $G(C_{u|i}) \geq b$ **then**
      $C_{u|i} = \{d_{u|1}, d_{u|2}, \ldots, d_{u|m}\}$;
      for each $j \in \{1, m\}$ **do**
        $d_{u|j} = (x_j, y_j)$;
        $x_j^* = LSB(x_j, p_u, t_j)$;
        /* triggers injection as shown in Alg. 2 */
        $d_{u|j}^* = (x_j^*, t_j)$;
      $C_{u|i}^* = \{d_{u|1}^*, d_{u|2}^*, \ldots, d_{u|m}^*\}$;
      Send $C_{u|i}^*$ to the *SP*;
    else
      Send $C_{u|i}$ to the *SP*;

SP-side:

Initialize the model m_0;
for each $i \in \{1, l\}$ **do**
  $D_i = \{C_{1|i}, C_{2|i}, \ldots, C_{n|i}\}$;
  Train m_i with D_i on the basis of m_{i-1}; /* train the model as shown in Eq.1 */
  Save model m_i;
Output the final model m_l;
Send a training completion response to users;

Algorithm 2 *LSB* Backdoor Triggers Injection

Input: The character $w \in \{1, |p|\}$ of trigger p, the binary function *Bin*, the pixel $v \in \{1, |x|\}$ of the sample x, the substitution function *Sub*, and the decimal function *Dec*.

Output: The poisoned sample x^*.

for each trigger character $w \in \{1, |p|\}$ **do**
  $\tilde{w} = Bin(w)$;
  $|\tilde{p}| = 8 \times |p|$;
Convert the sample x to $|x|$ pixel values;
for each pixel value $v \in \{1, |x|\}$ and $\tilde{w} \in \{1, |\tilde{p}|\}$ **do**
  $\tilde{v} = Bin(v)$;
  $\tilde{v}^* = Sub(\tilde{v}, -1, \tilde{w})$;
if $|\tilde{p}| > |x|$ **then**
  for each $v \in \{1, |x|\}$ and $\tilde{w} \in \{|x|, |\tilde{p}|\}$ **do**
    $\tilde{v} = Bin(v)$;
    $\tilde{v}^* = Sub(\tilde{v}, -2, \tilde{w})$;
else if $|\tilde{p}| > 2 \times |x|$ **then**
  for each $v \in \{1, |x|\}$ and $\tilde{w} \in \{2 \times |x|, |\tilde{p}|\}$ **do**
    $\tilde{v} = Bin(v)$;
    $\tilde{v}^* = Sub(\tilde{v}, -3, \tilde{w})$;
else if $|\tilde{w}| > 3 \times |x|$ **then**
  for each $v \in \{1, |x|\}$ and $\tilde{w} \in \{3 \times |x|, |\tilde{p}|\}$ **do**
    $\tilde{v} = Bin(v)$;
    $\tilde{v}^* = Sub(\tilde{v}, -4, \tilde{w})$;
for each binary pixel value $\tilde{v}^* \in \{1, |x|\}$ **do**
  $\dot{v} = Dec(\tilde{v}^*)$;
Use $\dot{v}$ to recover poisoned sample x^*;

finished, and then provide the user with a prediction service.

2) Data Deletion and Model Retraining: In the context of machine unlearning, users have the right to request that their data be forgotten from the *SP*. After receiving requests to delete data, an honest *SP* should comply with these requests and delete the data as requested before retraining a new model using the remaining data. Alg. 3 illustrates the process of data deletion and model retraining in our proposed design. These two operations are executed by users and the *SP*, respectively.

(a) *User-side request.* In order to accurately implement machine unlearning, the user u ($u \in \{1, n'\}$, $n' \leq n$) needs to send a data deletion request R_u to the *SP*, containing user ID u and the location i of the data to be deleted, i.e., $R_u = (u, i)$ represents the data slice $C_{u|i}$ that needs to be deleted. Note that all users have the right to request the *SP* to remove data in batches and retrain the model with the remaining data. Each user u can send more than one data deletion request R_u to the *SP*, thus enabling highly private data deletion.

(b) *SP-side retrain.* The *SP* locates and deletes the i-th data slice $C_{u|i}$ according to the request R_u, and then updates $D_i = \{C_{1|i}, C_{2|i}, \ldots, C_{u-1|i}, C_{u+1|i}, \ldots, C_{n|i}\}$. In order to improve the efficiency of retraining the model after deleting the data, the *SP* uses the incremental learning method for retraining. That is, instead of retraining from scratch, the *SP* only needs to start retraining from the intermediate model affected by data deletion using the *LWF* algorithm described in Section II-A. To avoid training the same intermediate model repeatedly in the case of batch data deletion, the *SP* uses the minimum function *Min* to calculate the minimum slice index i_{min} among all requests from different users, and retrains all affected models (i.e., $\{m_{i_{min}}, m_{i_{min}+1}, \ldots, m_l\}$) using the new dataset D_i, where $i \in \{i_{min}, l\}$. Meanwhile, each intermediate model m_i should be saved and output the final retrained model m_l. Finally, the *SP* returns a deletion completion response to each user u indicating that the retrained prediction service is ready.

3) Machine Unlearning Validation: As described in Section IV-B1, users inject triggers into their highly private data before sending it to the *SP* for training. These triggers

Algorithm 3 Data Deletion and Model Retraining

Input: The data deletion request R_u from the user u, the i-th data slice to be deleted, the minimum value function *Min*, and the intermediate modes $\{m_1, m_2, \ldots, m_l\}$.
Output: The retrained final model m_l and the deletion completion response.
User-side:
for each user $u \in \{1, n'\}$ **do**
  Send $R_u = (u, i)$ to the *SP*;
SP-side:
for each R_u from the user u, $u \in \{1, n'\}$ **do**
  Delete the data slice $C_{u|i}$;
  Update $D_i = \{C_{1|i}, C_{2|i}, \ldots, C_{u-1|i}, C_{u+1|i}, \ldots, C_{n|i}\}$;
Compute $i_{min} = Min(i)$, where i from different R_u;
for each $i \in \{i_{min}, l\}$ **do**
  Retrain m_i with D_i on the basis of m_{i-1};
  Save model m_i;
Output the final model m_l;
Send a deletion completion response to users;

Algorithm 4 Machine Unlearning Validation

Input: The test data $T_{u|i}$ corresponding to the deleted data $C_{u|i}$.
Output: The validation result q.
SP-side:
for each $T_{u|i}$ from the user u, $u \in \{1, n'\}$ **do**
  Generate the prediction label e of $T_{u|i}$ based on m_l;
  Send the prediction label e of $T_{u|i}$ to the user u;
User-side:
for each user $u \in \{1, n'\}$ **do**
  Receive e from the *SP*;
  Find the target label t of $T_{u|i}$;
  if $e == t$ **then**
    set $q = 0$;
  else
    set $q = 1$;
  Return the validation result q;

act as markers to detect whether the *SP* has complied with the data deletion request from the user. The detailed validation of machine unlearning is shown in Alg. 4. Specifically, user u first finds the test data $T_{u|i}$ corresponding to the deleted poisoned data $C_{u|i}$ (i.e., $T_{u|i}$ and $C_{u|i}$ belong to the same class, $C_{u|i}$ is used to train the model, and $T_{u|i}$ is used to test the model). The poisoned $T_{u|i}$ is generated by the *LSB* algorithm shown in Alg. 2. The specific form of the marker can be a text trigger, such as "AliceAlice…ice". The user needs to convert each character of the marker and each pixel of the sample in $T_{u|i}$ into the form of an 8-bit binary with the binary function *Bin*. Then, the substitution function *Sub* is used to replace the least significant bits of each pixel with the trigger bit-by-bit. After that, the user should change the source label of the data point in $T_{u|i}$ to the target label. Finally, the poisoned $T_{u|i}$ is sent to *SP* for prediction. Note that if the user sends more than one data deletion request to the *SP*, multiple test data $T_{u|i}$ can be used to verify whether the *SP* has complied with the data deletion requests.

(a) *SP-side prediction.* When the user sends data to the *SP*, the *SP* needs to return a prediction result about the test data to the user. Upon receiving the data $T_{u|i}$ from the user u, the *SP* generates the prediction label e based on the retrained model m_l, and then sends the prediction label e of $T_{u|i}$ to the user u for validation.
(b) *User-side validation.* After receiving the prediction label e, user u can fetch the target label t of data $T_{u|i}$ and compare e with t. In particular, if e and t are the same, it indicates that the *SP* still remembers the features of trigger p_u. This means that the *SP* did not delete the data as required, so that the validation result is set to $q = 0$. Otherwise, it means that the *SP* has deleted the data, and the user can set the validation result $q = 1$ to the *SP*.

TABLE II
DATASET DETAILS

Dataset	Number of classes	Training set size/ test set size	Slice	Test accuracy
MNIST	10	60,000/10,000	11	98.90%
CIFAR-10	10	50,000/10,000	11	94.35%
GTSRB	43	39,209/12,630	44	91.74%

After validating that the *SP* is trustworthy, users can use the prediction service with confidence.

V. EVALUATION RESULTS

A. Implementation and Setup

In this section, we comprehensively evaluate the performance of our backdoor-assisted validation scheme for machine unlearning by using the method of retraining from scratch and retraining from breakpoints, respectively. We conduct experiments on MNIST [30], CIFAR-10 [31], and GTSRB [32] datasets, which are representative datasets in the field of image classification. As shown in the Table II, MNIST is a dataset of handwritten digit images, which contains 10 classes (i.e. [0-9]) of 28×28 images. The MNIST dataset contains 70,000 images, 60,000 of which are used for training neural networks and 10,000 images are used for testing. There are 50,000 training images and 10,000 test images in CIFAR-10, which contain 10 classes of color images such as cats, ships and airplanes. The German Traffic Sign Recognition Benchmark (GTSRB) contains 43 classes of color images, which are divided into 39,209 training images and 12,630 test images. For these three datasets, we choose the ResNet-34 [33] network as our basic model to obtain the test accuracy of 98.90%, 94.35%, and 91.74%, respectively. This test accuracy is obtained using the training method of incremental learning. Our experiments are carried out on a *SP* equipped with a Ryzen 7, with 16GB of memory and an RTX 3060. The neural networks were implemented with Pytorch 1.8.1.

The primary goal of our design is to achieve an efficient and verifiable machine unlearning scheme for MLaaS. We leverage the *LWF* algorithm to implement incremental learning. Concretely, we first randomly divide each dataset into 10 subsets to simulate 10 users, and then divide these subsets into slices according to the class of dataset, establishing a data-model index structure as shown in Fig. 1. For instance, each subset of GTSRB can be divided into 44 (i.e., $1 + 43$) slices to represent 44 privacy levels of data. Among them, the first data slice does not involve the privacy of the user, while the data of the last 43 slices belong to private data. The overall slice of data is arranged in ascending order of privacy. For machine unlearning validation, we inject markers into the data for subsequent validation that the *SP* complied with the data deletion request. To be specific, we inject a marker into the private data of the user, and then change the source label of the data to a target label. Note that the source label should correspond to the target label to form a source-target pair. Besides, we use a secure PRF to map the name of each user to a unique text marker to avoid inaccurate machine unlearning validation when users with the same ID use the same marker.

Fig. 5 illustrates the impact on *BASR* with different trigger sizes and the invisibility rate of our proposed marker. Here, the invisibility rate is assessed by the PASS score proposed by Rozsa et al. [34]. Specifically, the range of PASS scores between two images is (0, 1], and two identical images have a PASS score of 1. From Fig. 5, we can observe that the *BASR* gradually increases with the growing size of trigger, but the invisibility rate also gradually decreases. Note that the *BASR* and invisibility rate are optimized when the trigger length is 500, so that we choose 500 as the trigger size in the subsequent experiments. In Fig. 6, we also compare the invisible effects of the backdoor triggers with previous design via PASS values. Compared to BadNets [35], the backdoor triggers we used have a higher invisibility rate of 0.998. According to the evaluation result, our design can effectively verify the proof of machine unlearning while preserving the markers invisibility.

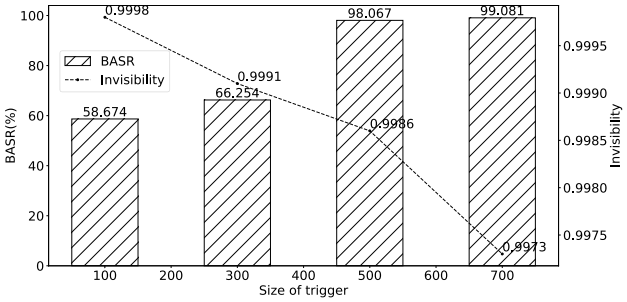
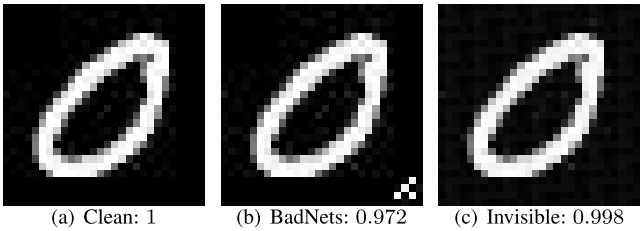


Fig. 5. Impact on BASR with different trigger sizes.



(a) Clean: 1 (b) BadNets: 0.972 (c) Invisible: 0.998

Fig. 6. The comparison of the PASS scores.

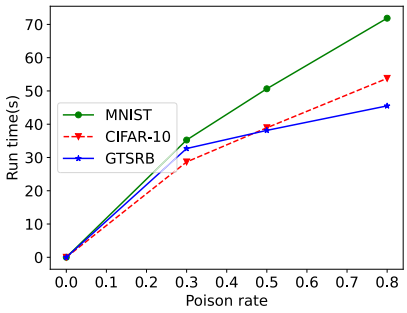


Fig. 7. The time cost of marker injection.

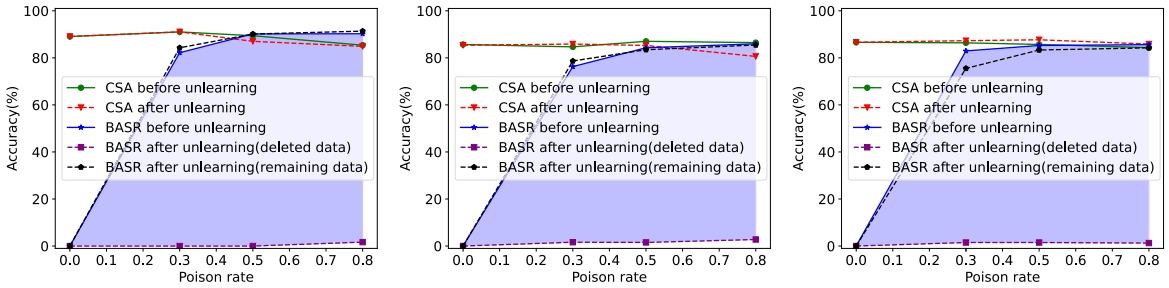
B. Experimental Results of Marker Injection

Recall that in our backdoor-assisted validation scheme, users inject markers into their private data to enable verifiable machine unlearning. We select different proportions of data samples to inject markers for measuring the initialization time cost of different poison rates in Fig. 7. Specifically, it includes the time cost of markers injection in the training dataset and the test dataset, as well as the time to store the data. We can observe that the time cost increases linearly with the number of poisoned data. Nonetheless, the average time cost of marker injection for 80% data is still less than 80s, which is within acceptable performance levels.

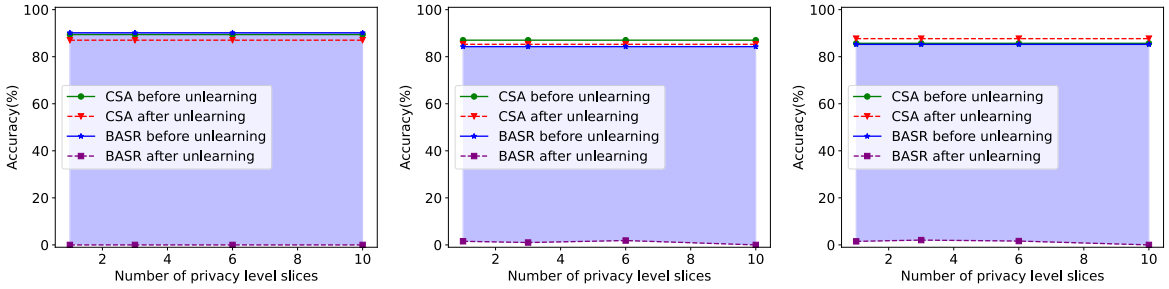
C. Experimental Results of Retraining From Scratch

In this section, we measure the performance of retraining from scratch, where the *SP* retrains the model from scratch with the remaining data while users utilize the backdoor-assisted validation scheme to verify the proof of machine unlearning. The main performance indicators are the backdoor attack success rate (*BASR*) and the clean sample accuracy (*CSA*), which are important indicators to measure the quality of backdoor attacks. The *BASR* represents the test accuracy of the neural network on poisoned data (i.e., the data with marker). A high *BASR* indicates that the neural network has learned the marker we injected. If the poisoned data is removed, the neural network will not be able to classify the poisoned image or the remaining images as the target label. Thus, our design uses this characteristic to determine whether the *SP* is honestly deleting the data. For an honest *SP*, we can observe that the *BASR* decreases dramatically before and after the data is deleted. While the *CSA* indicates the test accuracy of the neural network on clean data. The markers should only affect the poisoned data instead of the clean data, so that a high *CSA* reflects that the markers we injected have little effect on the performance of models. In our design, we utilize the difference between the *BASR* before and after data deletion to verify that the *SP* complied with data deletion requests, while we use *CSA* to confirm that the markers injected into data samples do not affect model performance.

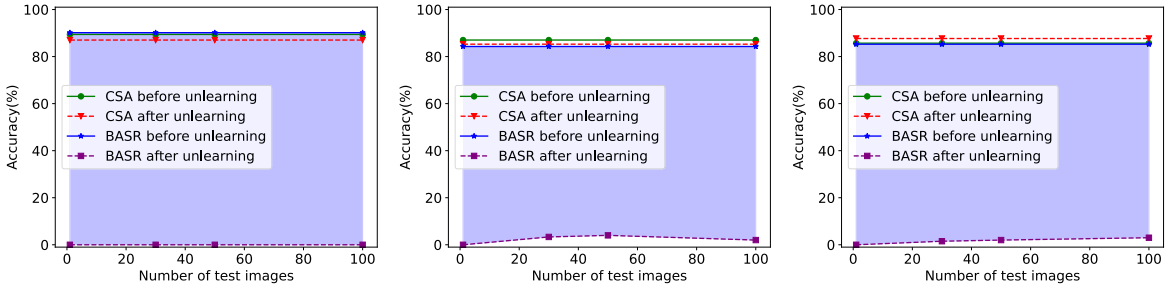
We measure the effects of the poison rate, the number of classes of deleted data, the number of the test images, and the number of users on the accuracy of the model, respectively.



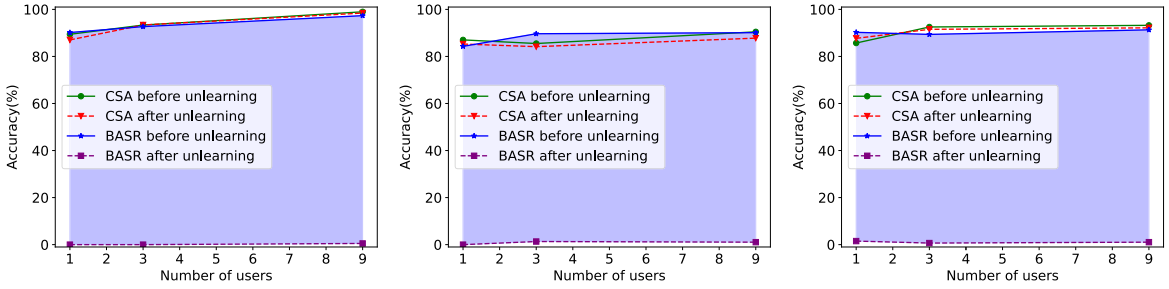
(a) Evaluation results for poison rate on MNIST (left), CIFAR-10 (medium), and GTSRB (right).



(b) Evaluation results for the number of deleted slices on MNIST (left), CIFAR-10 (medium), and GTSRB (right).



(c) Evaluation results for the number of test images on MNIST (left), CIFAR-10 (medium), and GTSRB (right).



(d) Evaluation results for the number of users on MNIST (left), CIFAR-10 (medium), and GTSRB (right).

Fig. 8. Impact on accuracy of retraining from scratch.

The evaluation results are presented in Fig. 8. Specifically, we show the *BASR* and *CSA* before and after machine unlearning (i.e., data deletion), and the shaded part represents the difference. It is observed that there is a significant difference in *BASR* before and after data deletion, which implies the effectiveness of our method for verifiable machine unlearning.

We evaluate the effect of poison rate for machine unlearning validation on the MNIST, CIFAR-10, and GTSRB datasets, as shown in Fig. 8(a). Specifically, we delete only one class of data in the single-user case and keep the number of the test images at 100, which is used to verify whether the *SP* complies with data deletion. When the poison rate changes from 0 to 0.3, we notice that the *BASR* before unlearning rapidly increases and remains at a high level. In addition, the *CSA* before unlearning is also maintained at a high value, which is not affected by the poison rate. Among them, *BASR* still shows a slight improvement on MNIST and CIFAR-10 when the poison rate increases from 0.3 to 0.5. These evaluation results confirm that the neural network can learn the features of the marker we injected without affecting clean samples. To evaluate the impact of data deletion, we measure the *CSA* and *BASR* on the remaining data after unlearning. We can observe that the

TABLE III
DIFFERENCE OF *BASR* BEFORE AND AFTER UNLEARNING

Dataset	*BASR* retrain from scratch	*BASR* retrain from breakpoints
MNIST	88.26%	99.69%(11.43% ↑)
CIFAR-10	84.28%	89.70%(5.42% ↑)
GTSRB	83.74%	84.87%(1.13% ↑)

TABLE IV
BASR IN THE CASE OF NAME COLLISION

Dataset	*BASR* before unlearning	*BASR* after unlearning	Difference
MNIST	88.24%	0%	88.24%
CIFAR-10	88.95%	0%	88.95%
GTSRB	89.30%	0.67%	88.63%

CSA and *BASR* on the remaining data barely change before and after unlearning. As mentioned above, the difference in *BASR* before and after unlearning reflects the success rate of our verifiable unlearning, which also represents the level of user trust in the *SP*. Considering the factors mentioned above, we choose the poison rate of 0.5 for follow-up experiments. With a poison rate of 0.5, the differences in *BASR* before and after unlearning are 90.16%, 82.74%, and 83.75% on the three datasets, respectively.

In our design, a data slice represents the same privacy level of datasets, and our scheme should be able to accommodate the requirements of users to delete multiple privacy levels of data. Thus, we further measure the effects of the number of slices removed on different datasets at a fixed 0.5 poison rate and 100 images. From Fig. 8(b), we can observe that the *BASR* and *CSA* are almost unaffected by changing the number of deleted slices. The evaluation results confirm that the user can delete any number of data slices without affecting the performance of the verifiable machine unlearning scheme.

Recall that in order to verify that the *SP* is honestly deleting data, the user needs to send the image to the *SP* for testing and expects to obtain the source label of the data instead of the target label. In Fig. 8(c), we evaluate the effectiveness of a user using different numbers of test images to verify whether the *SP* is honest when only one class of data was deleted. Specifically, we send 1, 30, 50, and 100 images as test sets to the *SP* for prediction. Note that these test images are injected with a marker for the same class of data that the user requested *SP* to remove. Fig. 8(c) shows that users can obtain a lower *BASR* after unlearning, indicating that the *SP* has forgotten the marker injected into the private data by the user.

In Fig. 8(d), we further evaluate the effect of different number of users on the machine unlearning scheme. We can observe that the accuracy rate remains stable, which indicates that our backdoor-assisted validation scheme can effectively support the *SP* to process the unlearning requests sent by multi-users simultaneously. Overall, the above experiments show that our machine unlearning scheme is verifiable and the performance results are stable when retraining from scratch.

D. Experimental Results of Retraining From Breakpoints

In this subsection, we use an incremental learning approach to evaluate the performance of our scheme for retraining from breakpoints under different settings, and compare it with the traditional retraining methods from scratch. Similar to Section V-C, we utilize the metrics of *BASR* and *CSA* to understand how markers injection to data enables verifiable machine unlearning. Furthermore, we compare the performance of retraining from scratch and retraining from breakpoints, and evaluate the effectiveness of our PRF-based markers structure.

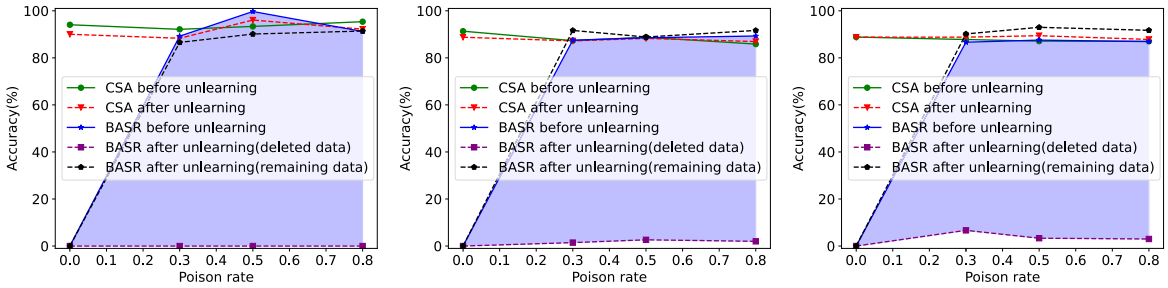
We first measure the validation effect at different poison rates. Fig. 9(a) shows that the *BASR* before unlearning is highest when the poison rate is 0.5 on the MNIST dataset while the CIFAR-10 and GTSRB datasets have similar effect results. Thus, we propose to use the 0.5 poison rate in the subsequent experiments. In addition, we can observe that the main fluctuation in Fig. 9(a) is only the *BASR* before unlearning, especially when the poison rate changes from 0 to 0.3. This indicates that users can implement verifiable machine unlearning by injecting markers into their data based on their privacy level.

To accommodate the privacy protection requirements of different users, we explore the impact of different numbers of deleted privacy level slices on system performance in Fig. 9(b). Specifically, we test the effect of deleting 1, 3, 6, and 10 slices of data, respectively. We can find that the *BASR* and *CSA* remain constant as the number of deleted slices increases. The evaluation results confirm that our design can adapt to different data deletion requirements from users.

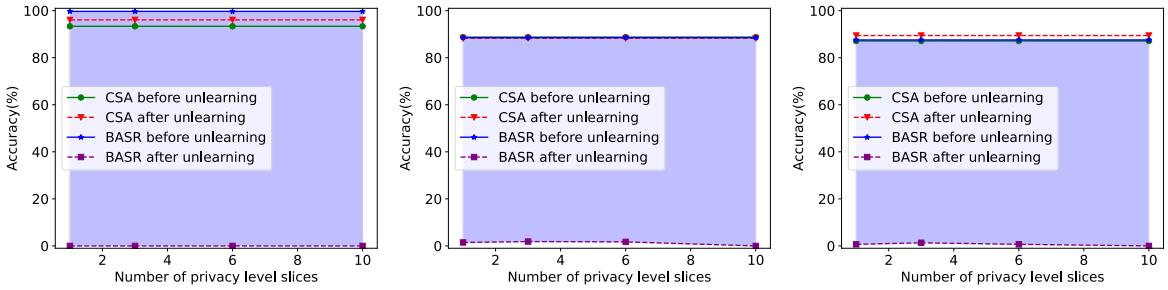
In Fig. 9(c), we further explore the impact of the number of test images on our design when conducting machine unlearning via incremental learning. Recall that the test images are only used to test the degree of forgetting on poisoned data after unlearning, so that the *CSA* before and after unlearning and the *BASR* before unlearning in Fig. 9(c) remain constant. While the *BASR* becomes lower after unlearning, indicating that the *SP* has deleted the data.

In Fig. 9(d), we further evaluate the performance of our machine unlearning scheme in the multi-user setting. We can observe that the shaded area remains large as the number of users increases, which means that our design supports the participation of multiple users.

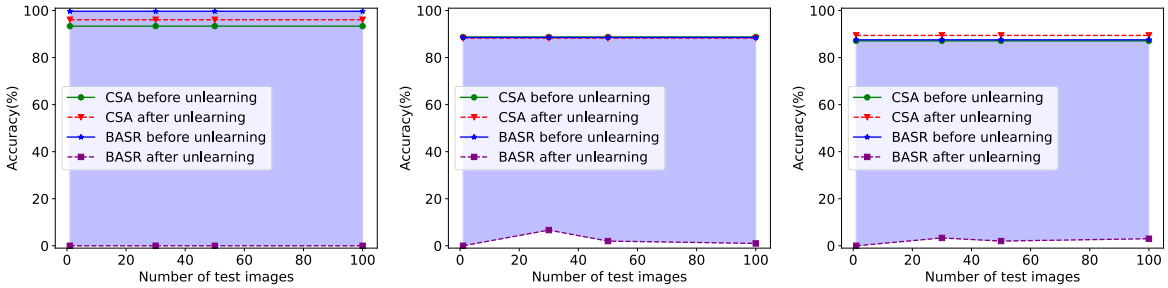
Guided by the above mentioned design principle, we are aware that the difference in *BASR* before and after unlearning is a crucial indicator of whether the *SP* is deleting data honestly. Note that in order to prevent catastrophic forgetting, our incremental learning process uses the *LWF* algorithm and randomly reviews the content learned in the previous stage. By comparing with traditional retraining from scratch, we find that the incremental learning approach significantly enhances the validation performance of our scheme. Table III demonstrates the differences of *BASR* before and after unlearning obtained by the method of retraining from scratch and the method of retraining from breakpoints in the single-user case. We can observe that by using the method of retraining from breakpoints, the difference of *BASR* before and after unlearning on the MNIST dataset is 11.43% higher than



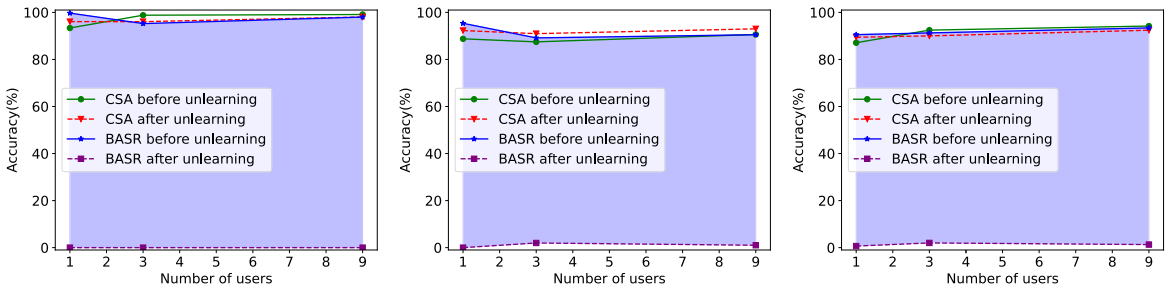
(a) Evaluation results for poison rate on MNIST (left), CIFAR-10 (medium), and GTSRB (right).



(b) Evaluation results for the number of deleted slices on MNIST (left), CIFAR-10 (medium), and GTSRB (right).



(c) Evaluation results for the number of test images on MNIST (left), CIFAR-10 (medium), and GTSRB (right).



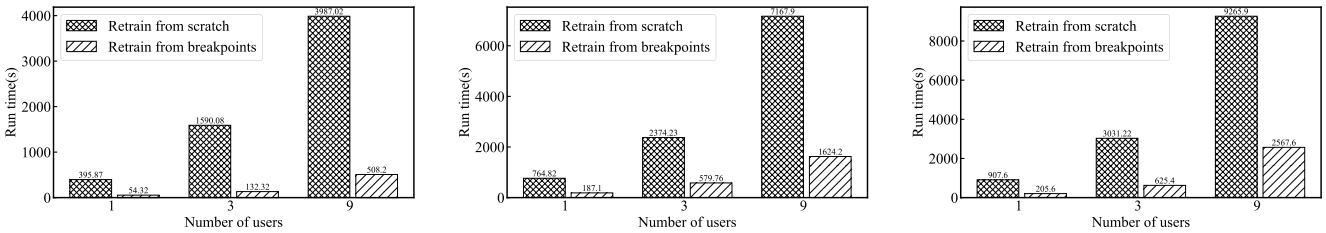
(d) Evaluation results for the number of users on MNIST (left), CIFAR-10 (medium), and GTSRB (right).

Fig. 9. Impact on accuracy of retraining from breakpoints.

retraining from scratch. The evaluation results show that the validation effect using our incremental learning approach is better than the method of retraining from scratch.

To comprehensively evaluate the efficiency of our backdoor-assisted validation scheme, we further compare the time cost of retraining from scratch and retraining from breakpoints. As shown in Fig. 10, the time cost of machine unlearning increases gradually as the number of users increases. Obviously, retraining from scratch takes more runtime than retraining from breakpoints. As shown in Fig. 10(a), when there are 3 users, the runtime for retraining from scratch is almost 12× longer than the runtime for retraining from breakpoints. It should be noted that since users have sorted the private data according to the privacy level before sending them to the *SP* for training. Therefore, the data deletion requests from users can only affect these poisoned data without affecting the rest of the intermediate models.

Finally, we test the performance of our PRF-based marker design against the special cases where multi-users have the same ID. Table IV demonstrates the *BASR* before and after unlearning using our PRF-based marker structure, as well as the difference between them. We can observe that the *BASR*



(a) The run time on MNIST. (b) The run time on CIFAR-10. (c) The run time on GTSRB.

Fig. 10. The comparison between retraining from scratch and retraining from breakpoints.

TABLE V
CHARACTERIZATION OF REPRESENTATIVE MACHINE UNLEARNING DESIGNS

Solutions	Exact unlearning	Verifiable	Unlearning granularity	
			Sample	User
Golatkar *et al.* [12]	✗	✗	✓	✓
Ma *et al.* [14]	✗	✓	✓	✓
Bourtoule *et al.* [16]	✓	✗	✓	✓
Weng *et al.* [19]	✓	✓	✓	✓
Warnecke *et al.* [27]	✗	✗	✓	✓
Chundawat *et al.* [36]	✗	✓	✓	✓
Sommer *et al.* [37]	✓	✓	✗	✓
Our design	✓	✓	✓	✓

before unlearning is close to 90%, while the *BASR* after unlearning is almost 0%. The difference between the *BASR* before and after unlearning is roughly 90%, which proves that our design can effectively prevent users with the same ID from using the same marker, resulting in inaccurate validation of machine unlearning. Based on the above experiments, we show the implementation of an efficient and verifiable machine unlearning scheme. Our design is more efficient, accurate, and robust compared to the method of retraining from scratch.

VI. RELATED WORK

A. Machine Unlearning

Within the extensive application of machine learning, there exists a line of work that studies the problem of machine unlearning. A straightforward approach is to retrain a new model from scratch on the remaining data after deleting it. However, the retraining process can be expensive when there is too much data to be deleted. To avoid consuming large amounts of computing power by retraining from scratch, recent efforts have focused on efficient machine unlearning. In Table V, we have summarized the representative solutions with their characteristics. Through comparison, we can find that our design has achieved the exact and verifiable machine unlearning with fine-grained data deletion. In [10], the authors attempted to save computational resources by proposing a relaxed notion of data deletion. They developed a technique of efficient k-means clustering to enable data deletion. Golatkar et al. [12] approximated the forgetting of privacy by measuring the influence of data on the model in neural networks. A recent work [36] proposed a method for machine unlearning that does not require the use of training data by utilizing the techniques of error minimization-maximizing noise and gated knowledge transfer. In [38], the authors employed a conditional independence coefficient to implement approximate unlearning in visual models and natural language processing models. The above methods are called approximate unlearning, which considers machine unlearning successful when the parameters of the model obtained by unlearning are close to the parameters of the model that has not learned the data point required to be deleted. This is not advisable because it is possible to use different data to train a model with the same parameters. Taking advantage of this, a malicious *SP* might fork a model and claim to have deleted the private data of users. In [15], the authors applied the idea of statistical query learning to enable exact unlearning. Similarly, Bourtoule et al. [16] proposed to divide the training dataset into disjoint slices and train the intermediate model on each shard separately. Thus, the proposed design can improve the efficiency of machine unlearning because only a few intermediate models affected by the deleted data points need to be retrained. However, the above designs cannot address the security issues from untrusted *SP*. To this end, Weng et al. [19] proposed to leverage the trusted execution environment (i.e., Intel SGX) to enforce the validation of machine unlearning. However, the security assurance of their design relies on the existence of trusted hardware on the *SP* side. Recently, a system designed by Sommer et al. [37] introduced a backdoor-based method to verify machine unlearning. However, the backdoor trigger used in [37] can be easily detected by the *SP*, and leading to authentication fraud issues. Besides, the *SP* needs to retrain the model from scratch after deleting the data, incurring a considerable burden on training overhead.

B. Backdoor Attack

Backdoor attacks refer to injecting backdoor triggers into the training images of machine learning, which can affect the predictive function of the machine learning models. This is possible due to the powerful feature extraction ability of machine learning models, which also allows for the addition of small disturbances that can cause incorrect predictions. In the work of Liu et al. [39], they injected backdoor triggers into the training samples by poisoning them. However, there is an obvious difference between the triggers in this backdoor attack and their adjacent pixels. Although this difference can be recognized by the neural network, it is also easy to be found by the *SP*, which may lead to the failure of the backdoor

attack. Bagdasaryan et al. [40] viewed the backdoor attack as a multi-objective optimization, and they synthesized poisoning inputs when computing the values of loss function during training. However, their attack is blind, which means that the attacker cannot inject triggers into the data, nor can they access the final model. The blindness of [40] makes this backdoor attack unsuitable for our design because users need to inject triggers into their sensitive data to verify whether the *SP* deletes the data as required. Li et al. [41] generated invisible backdoor triggers through an encode-decode network. However, this attack requires users to have the ability to generate triggers for each sample, which is not applicable to practical implementation.

VII. Conclusion

In this paper, we introduce a backdoor-assisted validation scheme for machine unlearning in MLaaS. To ensure the validation process is non-spoofable, we propose to construct invisible markers using backdoor triggers and embed them into the outsourced data. Users can verify the proof of machine unlearning by observing whether the markers are included in the final model. Since the poisoned data cannot be distinguished from source data, the *SP* can only pass the validation by strictly following the data deletion request. Besides, we further incorporate the validation scheme into an efficient incremental learning approach by using our data-model index structure to further facilitate the efficiency of retraining. We present how to generate poisoned data and conduct machine unlearning validation, and execute model retraining with the remaining data and intermediate models. Finally, we implement a system prototype and evaluate the feasibility of our design via three representative datasets. In our future work, we plan to explore our design to implement machine unlearning in the field of natural language processing. Meanwhile, we will also investigate how our design can be used to achieve federated unlearning as our future work.

References

[1] Q. Weng et al., "MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters," in *Proc. NSDI*, 2022, pp. 945–960.

[2] R. Zhang, J. Liu, Y. Ding, Z. Wang, Q. Wu, and K. Ren, "'Adversarial examples' for proof-of-learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 1408–1422.

[3] W. Wang, R. Wang, L. Wang, Z. Wang, and A. Ye, "Towards a robust deep neural network against adversarial texts: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 3, pp. 3159–3179, Mar. 2023.

[4] J. Weng, J. Weng, C. Cai, H. Huang, and C. Wang, "Golden grain: Building a secure and decentralized model marketplace for MLaaS," *IEEE Trans. Depend. Secure Comput.*, vol. 19, no. 5, pp. 3149–3167, Sep. 2022.

[5] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, and B. Feng, "VeriML: Enabling integrity assurances and fair payments for machine learning as a service," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 10, pp. 2524–2540, Oct. 2021.

[6] L. Zhao, Q. Wang, Q. Zou, Y. Zhang, and Y. Chen, "Privacy-preserving collaborative deep learning with unreliable participants," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1486–1500, Sep. 2019.

[7] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide*, vol. 10, no. 3152676, 1st ed. Cham, Switzerland: Springer, 2017, p. 5555.

[8] (2023). *California Consumer Privacy Act (CCPA)*. [Online]. Available: https://oag.ca.gov/privacy/ccpa

[9] H. Xu, T. Zhu, L. Zhang, W. Zhou, and P. S. Yu, "Machine unlearning: A survey," *ACM Comput. Surv.*, vol. 56, no. 1, pp. 1–36, 2023.

[10] A. Ginart, M. Guan, G. Valiant, and J. Y. Zou, "Making ai forget you: Data deletion in machine learning," in *Proc. NIPS*, vol. 32, 2019.

[11] A. Golatkar, A. Achille, and S. Soatto, "Eternal sunshine of the spotless net: Selective forgetting in deep networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 9301–9309.

[12] A. Golatkar, A. Achille, A. Ravichandran, M. Polito, and S. Soatto, "Mixed-privacy forgetting in deep networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 792–801.

[13] A. Sekhari, J. Acharya, G. Kamath, and A. T. Suresh, "Remember what you want to forget: Algorithms for machine unlearning," in *Proc. NIPS*, vol. 34, 2021, pp. 18075–18086.

[14] Z. Ma, Y. Liu, X. Liu, J. Liu, J. Ma, and K. Ren, "Learn to forget: Machine unlearning via neuron masking," *IEEE Trans. Depend. Secure Comput.*, vol. 20, no. 4, pp. 3194–3207, Jul./Aug. 2022.

[15] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 463–480.

[16] L. Bourtoule et al., "Machine unlearning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 141–159.

[17] Y. Liu, L. Xu, X. Yuan, C. Wang, and B. Li, "The right to be forgotten in federated learning: An efficient realization with rapid retraining," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2022, pp. 1749–1758.

[18] A. Thudi, H. Jia, I. Shumailov, and N. Papernot, "On the necessity of auditable algorithmic definitions for machine unlearning," in *Proc. USENIX Secur.*, 2022, pp. 4007–4022.

[19] J. Weng, S. Yao, Y. Du, J. Huang, J. Weng, and C. Wang, "Proof of unlearning: Definitions and instantiation," 2022, *arXiv:2210.11334*.

[20] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Dec. 2018.

[21] Y. Gao et al., "Backdoor attacks and countermeasures on deep learning: A comprehensive review," 2020, *arXiv:2007.10760*.

[22] Z. Wang, J. Ma, X. Wang, J. Hu, Z. Qin, and K. Ren, "Threats to training: A survey of poisoning attacks and defenses on machine learning systems," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–36, Jul. 2023.

[23] L. Zhao et al., "Shielding collaborative learning: Mitigating poisoning attacks through client-side detection," *IEEE Trans. Depend. Secure Comput.*, vol. 18, no. 5, pp. 2029–2041, Sep. 2021.

[24] Z. Ma, J. Ma, Y. Miao, Y. Li, and R. H. Deng, "ShieldFL: Mitigating model poisoning attacks in privacy-preserving federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 1639–1654, 2022.

[25] Y. Liu et al., "Backdoor defense with machine unlearning," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2022, pp. 280–289.

[26] S. Li, M. Xue, B. Zi Hao Zhao, H. Zhu, and X. Zhang, "Invisible backdoor attacks on deep neural networks via steganography and regularization," *IEEE Trans. Depend. Secure Comput.*, vol. 18, no. 5, pp. 2088–2105, Oct. 2021.

[27] A. Warnecke, L. Pirch, C. Wressnegger, and K. Rieck, "Machine unlearning of features and labels," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2023, pp. 1–36.

[28] X. Gao et al., "VeriFi: Towards verifiable federated unlearning," 2022, *arXiv:2205.12709*.

[29] B. Wang et al., "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 707–723.

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[31] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[32] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German traffic sign detection benchmark," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Aug. 2013, pp. 1–8.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[34] A. Rozsa, E. M. Rudd, and T. E. Boult, "Adversarial diversity and hard positive generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2016, pp. 410–417.

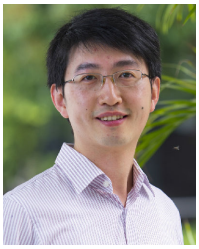
[35] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.

[36] V. S. Chundawat, A. K. Tarun, M. Mandal, and M. Kankanhalli, "Zero-shot machine unlearning," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 2345–2354, 2023.
[37] D. M. Sommer, L. Song, S. Wagh, and P. Mittal, "Towards probabilistic verification of machine unlearning," 2020, *arXiv:2003.04247*.
[38] R. Mehta, S. Pal, V. Singh, and S. N. Ravi, "Deep unlearning via randomized conditionally independent hessians," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 10412–10421.
[39] Y. Liu et al., "Trojaning attack on neural networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.
[40] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models," in *Proc. USENIX Secur.*, 2021.
[41] Y. Li, Y. Li, B. Wu, L. Li, R. He, and S. Lyu, "Invisible backdoor attack with sample-specific triggers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 16443–16452.

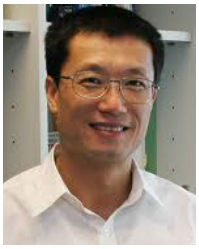
Yu Guo (Member, IEEE) received the B.E. degree in software engineering from Northeastern University in 2013 and the M.Sc. degree in electronic commerce and the Ph.D. degree in computer science from the City University of Hong Kong in 2014 and 2019, respectively. He is currently an Associate Professor with the School of Artificial Intelligence, Beijing Normal University. He has also been a Post-Doctoral Researcher and a Research Fellow with the City University of Hong Kong. He is a co-recipient of the Best Paper Award of IEEE ICDCS 2020 and MMM 2016. His research interests include cloud computing security, network security, privacy-preserving data processing, and blockchain technology.

Yu Zhao received the B.S. degree in data science and big data technology from the Minzu University of China, Beijing, China, in 2018. She is currently pursuing the master's degree with the Artificial Intelligence College, Beijing Normal University, Beijing. Her current research interests include the domain of information security, including machine unlearning and federated unlearning.

Saihui Hou (Member, IEEE) received the B.E. and Ph.D. degrees from the University of Science and Technology of China in 2014 and 2019, respectively. He is currently an Assistant Professor with the School of Artificial Intelligence, Beijing Normal University. His research interests include computer vision and machine learning.



Cong Wang (Fellow, IEEE) is currently an Associate Professor with the Department of Computer Science, City University of Hong Kong. His current research interests include data and network security, blockchain and decentralized applications, and privacy-enhancing technologies. He is one of the founding members of the Young Academy of Sciences of Hong Kong. He received the Outstanding Researcher Award (a Junior Faculty) in 2019, the Outstanding Supervisor Award in 2017, and the President's Awards with the City University of Hong Kong in 2016 and 2019, respectively. He is a co-recipient of the IEEE INFOCOM Test of Time Paper Award 2020, the Best Student Paper Award of IEEE ICDCS 2017, and the Best Paper Award of IEEE ICPADS 2018 and MSN 2015. His research has been supported by multiple government research fund agencies, including National Natural Science Foundation of China, Hong Kong Research Grants Council, and Hong Kong Innovation and Technology Commission. He serves/has served as an Associate Editor for IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE INTERNET OF THINGS JOURNAL, and IEEE NETWORKING LETTERS, and TPC co-chairs for a number of IEEE conferences/workshops. He is a member of the ACM.



Xiaohua Jia (Fellow, IEEE) received the B.Sc. and M.Eng. degrees from the University of Science and Technology of China in 1984 and 1987, respectively, and the D.Sc. degree in information science from The University of Tokyo in 1991. He is currently a Chair Professor with the Department of Computer Science, City University of Hong Kong. He is also an Adjunct Professor with the Harbin Institute of Technology (Shenzhen) while performing this work. His research interests include cloud computing and distributed systems, computer networks, wireless sensor networks, and mobile wireless networks. From 2006 to 2009, he was an Editor of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *Wireless Networks*, *Journal of World Wide Web*, and *Journal of Combinatorial Optimization*. He is the General Chair of ACM MobiHoc 2008, the TPC Co-Chair of IEEE MASS 2009, the Area-Chair of IEEE INFOCOM 2010, the TPC Co-Chair of IEEE GlobeCom 2010-Ad Hoc and Sensor Networking Symposium, and the Panel Co-Chair of IEEE INFOCOM 2011. He is fellow of the IEEE Computer Society.