Verification of Machine Unlearning is Fragile

Binchi Zhang [1] **Zihan Chen** [1] **Cong Shen** [1] **Jundong Li** [1]

Abstract

As privacy concerns escalate in the realm of machine learning, data owners now have the option to utilize machine unlearning to remove their data from machine learning models, following recent legislation. To enhance transparency in machine unlearning and avoid potential dishonesty by model providers, various verification strategies have been proposed. These strategies enable data owners to ascertain whether their target data has been effectively unlearned from the model. However, our understanding of the safety issues of machine unlearning verification remains nascent. In this paper, we explore the novel research question of whether model providers can circumvent verification strategies while retaining the information of data supposedly unlearned. Our investigation leads to a pessimistic answer: *the verification of machine unlearning is fragile*. Specifically, we categorize the current verification strategies regarding potential dishonesty among model providers into two types. Subsequently, we introduce two novel adversarial unlearning processes capable of circumventing both types. We validate the efficacy of our methods through theoretical analysis and empirical experiments using real-world datasets. This study highlights the vulnerabilities and limitations in machine unlearning verification, paving the way for further research into the safety of machine unlearning.

1. Introduction

In the deep learning era, machine learning (ML) has grown increasingly data-dependent. A significant volume of personal data has been utilized to train real-world ML systems. While the plentiful use of personal data has facilitated the advancement of machine learning, it simultaneously poses a threat to user privacy and has resulted in severe data breaches (Nguyen et al., 2022). Consequently, recent regulations and laws (GDPR, 2016; CCPA, 2018) have mandated a novel form of request: the elimination of personal data and its impact from a trained ML model, known as machine unlearning (Cao & Yang, 2015) (MUL). In recent years, a proliferation of MUL techniques has emerged, employing various methods to remove certain training data (Nguyen et al., 2022; Xu et al., 2023).

Despite the success achieved, current MUL techniques are still a black box for data owners, i.e., data owners cannot monitor the unlearning process and ascertain whether their data has been truly unlearned from the model (Eisenhofer et al., 2022; Xu et al., 2023). For instance, many technology companies, e.g., Google and Microsoft, provide machine learning as a service (MLaaS). Under MLaaS, individual data owners upload personal data to the server. The server then trains an ML model using the collected dataset and provides its predictive functionality as a service to the data owners (Sommer et al., 2022). In particular, data owners might want their data to be unlearned as long as this service is no longer required. However, after sending an unlearning request, data owners will not receive any proof that their data was indeed unlearned. Without the ability to verify, data owners have to trust the model provider blindly for the efficacy and integrity of the unlearning process. On the other hand, the model provider might deceive data owners and pretend to have their data unlearned to avoid potential deterioration of the model utility and extra computational cost caused by unlearning (Eisenhofer et al., 2022). Other than MLaaS, similar problems also exist in other domains involving personal data, e.g., social media and financial systems. To address the above issue, recent studies have started exploring the verification strategy for MUL techniques from different aspects, e.g., injecting backdoor data into the training set or reproducing the unlearning operations. Regarding the model provider's potential dishonesty, a natural question arises: are current MUL verification strategies ensured to be safe? Specifically, we aim to study the following research question:

Can model providers successfully circumvent current verification strategies with dishonest unlearning?

To answer this question, we systematically investigate the

[1]University of Virginia, Charlottesville, VA, USA. Correspondence to: Jundong Li <jundong@virginia.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

vulnerability of verification methods and obtain a pessimistic answer: *current MUL verification is fragile*, i.e., data owners may fail to verify the integrity of the unlearning process using current MUL verification strategies. To support our study, we first categorize existing verification strategies into two types: *backdoor verification* and *reproducing verification*. We then propose an adversarial unlearning process that can successfully circumvent both of them, i.e., *always satisfies these two types of verification* but *still preserves the information of unlearned data*. During unlearning, our approach selects the mini-batches excluding the unlearned data to effectively evade the detection by existing verification strategies, even with the most stringent reproducing verification strategy. Meanwhile, the selected batches are designed to yield model updates akin to those that would be produced by the unlearned data. By deliberately choosing the retained data that mimics the influence of the unlearned data in training, the unlearned model yielded by our adversarial method retains the information from the unlearned data. In addition, we propose a *weaker but more efficient* adversarial unlearning process that can deceive a subset of reproducing verification by forging the unlearning processes directly from the original training steps. We provide theoretical guarantees for the efficacy of our approaches. Furthermore, we conduct comprehensive empirical experiments to validate the efficacy of our proposed adversarial unlearning processes on real-world datasets. We highlight our main contributions as follows:

- We propose two adversarial unlearning methods that can circumvent both types of current MUL verification strategies (backdoor and reproducing) while preserving the information of unlearned data.
- We prove the capacity of the proposed adversarial methods to satisfy the stringent reproducing verification. We also prove that they can preserve the unlearned model utility as the original training and improve the efficiency.
- We conduct empirical experiments to verify the efficacy of our proposed adversarial methods in real-world datasets, exposing the vulnerability of MUL verification.

2. Related Works

2.1. Machine Unlearning

The goal of MUL is to make a trained ML model forget some specific training data (Cao & Yang, 2015). A simple but powerful way of unlearning is to retrain the model from scratch, which is also called exact unlearning (Cao & Yang, 2015; Bourtoule et al., 2021; Kim & Woo, 2022; Chen et al., 2022). In exact unlearning, the unlearned model is ensured to behave as has never seen the unlearned data, which satisfies the goal of MUL. Following this path, Cao & Yang first proposed the retraining-based methodology for MUL. Follow-up studies (Bourtoule et al., 2021; Kim & Woo, 2022; Chen et al., 2022) took steps to improve the efficiency of the retraining framework on the image and graph data. Despite the efforts to improve efficiency, the retraining-based framework still has difficulty accommodating frequent unlearning requests in the real world (Chien et al., 2023). Due to the large computational cost of exact unlearning, approximate unlearning was proposed that efficiently updates the original model to estimate the retrained model (Guo et al., 2020; Ullah et al., 2021; Izzo et al., 2021; Zhang et al., 2022; Pan et al., 2023; Wu et al., 2023a;b; Che et al., 2023; Warnecke et al., 2023). A common way to update the original model is to use the influence function (Koh & Liang, 2017), which can be seen as conducting a single Newton step (Boyd & Vandenberghe, 2004; Sekhari et al., 2021; Neel et al., 2021) to the model. In particular, approximate unlearning can be certified if the distance between the unlearned model and the retrained model is bounded in the probability space (Nguyen et al., 2022).

2.2. Verification for Machine Unlearning

Backdoor Verification. Backdoor verification of MUL requires data owners to actively inject backdoor poisoned data (e.g., changing the original label to a different one for misleading) as backdoor triggers (Sommer et al., 2022; Gao et al., 2022; Guo et al., 2023). In this way, the model trained on the backdoor data can misclassify the backdoor data as the modified classes. To verify the integrity of unlearning, data owners can deliberately request the model provider to unlearn the backdoor data. If the model provider honestly unlearns the backdoor data, the predictions are supposed to be the original label with high confidence (backdoor data is not triggered); otherwise, the predictions can still be misled to the poisoned label (backdoor data is triggered). In addition, Sommer et al. formulated the backdoor verification as a hypothesis test, enabling a probabilistic guarantee of a successful verification.

Reproducing Verification. Inspired by verifiable computation techniques, reproducing verification of MUL requires the model provider to provide a *proof of unlearning (PoUL)* that records the operations for unlearning, and data owners can reproduce every unlearning step to verify the integrity of unlearning. Considering that the model provider might deceive the data owner while conducting exact unlearning, Thudi et al. first introduced the proof of learning (PoL) technique to verify the model retraining operation. Although they showed that the PoL can be forged by model providers, their forging strategy is not realistic and brings limitations to understanding the safety of MUL verification. Following this paradigm, Weng et al. presented a trusted hardware-empowered PoUL technique with SGX enclave (Costan & Devadas, 2016). With the trusted hardware, their framework

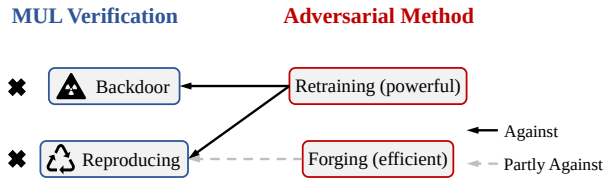


Figure 1. The connection of our threat model and different verification strategies. Our retraining method can deceive the backdoor and reproducing verification, and our forging method can only deceive a subset of reproducing verification but with better efficiency.

provides a better safety guarantee for verifying MUL. Recently, Eisenhofer et al. proposed the first cryptographic definition of verifiable unlearning and instantiated the PoUL with SNARKs (Costan & Devadas, 2016) and hash chains. Their verification framework has a safety guarantee from a cryptographic perspective.

Comparing different types of verification, backdoor verification requires data owners to inject poisoned data beforehand, while reproducing verification requires model providers to generate proof for the unlearning operations. In addition, Xu et al. summarized the current MUL evaluation methods (mainly for approximate unlearning), e.g., accuracy (Golatkar et al., 2020; 2021; Mehta et al., 2022), relearning time (Kim & Woo, 2022; Chundawat et al., 2023; Tarun et al., 2023), and membership inference attack (Chen et al., 2021) as verification strategies. However, even knowing the evaluation results, data owners still need to compare the results with the model retrained from scratch (exact unlearning method), which is unknown to data owners in practice. More importantly, evaluation methods fail to take into account the dishonest behaviors of model providers. Therefore, we do not focus on the safety problem of evaluation in this paper.

3. Threat Model

In our problem, we consider the data owner as a victim and the model provider as an adversary regarding safety in MUL verification. First, the model provider trains an ML model based on a training dataset provided by the data owners. After the data owners send an unlearning request, the model provider deceives the data owners that their data has been unlearned, while the model provider updates the model with an adversarial unlearning process rather than normal unlearning methods to preserve the information of the unlearned data. Next, we introduce the settings of our adversarial unlearning process from two perspectives: the adversary's goal and knowledge.

Adversary's Goal. The goal of using an adversarial unlearning process is to preserve the information of unlearned data during unlearning while satisfying the reproducing and backdoor verification. Note that the model provider might cheat the data owner for two benefits, i.e., *better model utility* and *lower computational cost*. Hence, other than preserving the unlearned data, the adversary's goal should also include these benefits.

Adversary's Knowledge. The model owner has complete access to the training process, e.g., training data, unlearned data, and the model.

As a tentative step toward exploring the vulnerability of MUL verification, we mainly focus on the verification of **exact unlearning**, i.e., verifying whether the model provider honestly retrains the model from scratch. We conclude the connection between our threat model and different types of verification strategies for MUL in Figure 1.

4. Methodology

In this section, we introduce our design of an adversarial unlearning process that deceives both the reproducing verification and the backdoor verification. To satisfy the reproducing verification, the model provider should provide a valid Proof of Retraining (PoRT)[1]. Hence, we first introduce the notations and background knowledge of PoRT before proposing our adversarial unlearning process.

4.1. Preliminary

Notation. We denote $\mathcal{D}$ as a training dataset with n data samples and $f_{\boldsymbol{w}}$ as an ML model with $\boldsymbol{w}$ collecting the learnable parameters. Let $\mathcal{A}$ be a learning process that takes the training set $\mathcal{D}$ as input and outputs the optimal $\boldsymbol{w}^*$ that minimizes the empirical risk on $\mathcal{D}$. In particular, a learning process $\mathcal{A}$ is used to minimize the empirical loss function

$$\mathcal{A}(\mathcal{D}) = \text{argmin}_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \mathcal{D}), \tag{1}$$

where $\mathcal{L}(\boldsymbol{w}, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} l(f_{\boldsymbol{w}}(\boldsymbol{x}), y)$ is the empirical risk over $\mathcal{D}$, $(\boldsymbol{x}, y)$ denotes the pair of the input data and the output label, and $l(\cdot)$ denotes a task-specific loss function, e.g., cross-entropy loss. We use $\mathcal{D}_u$ to denote the set of data to be unlearned. Consequently, the model retrained from scratch can be obtained by $\mathcal{A}(\mathcal{D}\backslash\mathcal{D}_u)$.

Proof of Retraining. In reproducing verification, the model provider is required to provide a PoRT, and the data owner or a third-party verifier can reproduce all the retraining operations in the PoRT to verify the integrity of unlearning. A prevalent instantiation of PoRT is to record the trajectory of retraining during the unlearning process (Thudi et al., 2022; Weng et al., 2022; Eisenhofer et al., 2022), denoted by $\mathcal{P}_r = \{\boldsymbol{w}_r^{(t)}, d_r^{(t)}, g_r^{(t)}\}_{t\in I}$ where $\boldsymbol{w}_r^{(t)}$ denotes the intermediate model parameter during retraining and $d_r^{(t)}$

[1]For better clarity, we use the term PoRT to replace the aforementioned PoUL as they are the same under exact unlearning.

denotes the data used for deriving $\boldsymbol{w}_r^{(t)}$. $g_r^{(t)}$ denotes the updating function and I is the set of indices to the intermediate learning steps during retraining ($I = \{1, 2, \ldots, T\}$ where T is the number of model updating steps). In addition, a Proof of Training (PoT) can be defined similarly as the PoRT, i.e., $\mathcal{P}_t = \{\boldsymbol{w}^{(t)}, d^{(t)}, g^{(t)}\}_{t \in I}$, while the difference is that the unlearned data should be involved in the training but not in the retraining. In this paper, we assume that the equivalence of adopted models and the models appear on the PoT ($\boldsymbol{w}^{(T)}$) and the PoRT ($\boldsymbol{w}_r^{(T)}$) can be verified, otherwise, reproducing verification will never be possible and the problem becomes trivial.

Reproducing Verification. We next provide a formal definition of reproducing verification (Thudi et al., 2022; Weng et al., 2022; Eisenhofer et al., 2022). As the retraining process can be divided into iterative steps, the PoRT can be seen as an ordered set of triplets $\{\boldsymbol{w}_r^{(t)}, d_r^{(t)}, g_r^{(t)}\}$. Each triplet describes an iterative operation that updates the retrained model based on an updating function as $\boldsymbol{w}_r^{(t)} = g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)})$. As an example, we can instantiate the updating function as the commonly used mini-batch stochastic gradient descent (Goodfellow et al., 2016):

$$g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)}) = \boldsymbol{w}_r^{(t-1)} - \gamma^{(t)} \nabla \mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)}), \quad (2)$$

where $\gamma^{(t)}$ denotes the learning rate. We can define a valid PoRT that can satisfy the reproducing verification as follows.

Definition 4.1. A valid Proof of Retraining is defined as $\mathcal{P}_r = \{\boldsymbol{w}_r^{(t)}, d_r^{(t)}, g_r^{(t)}\}_{t \in I}$ that satisfies the following two properties:

1. Reproducible: $\forall\, t \in I$, $\|\boldsymbol{w}_r^{(t)} - g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)})\| \leq \varepsilon$;
2. Removable: $\forall\, t \in I$, $d_r^{(t)} \cap \mathcal{D}_r = \emptyset$.

In the reproducible property, Thudi et al. set ε as a threshold for error tolerance, allowing the verifier to consider some numerical imprecision when reproducing the update rule, and $\|\boldsymbol{w}_r^{(t)} - g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)})\|$ is called the verification error at step t. In (Weng et al., 2022; Eisenhofer et al., 2022), the threshold ε can be reduced to an exact 0 with the help of SNARK (Setty, 2020) and Intel SGX (Costan & Devadas, 2016). Hence, we consider both the cases of 0 and ε thresholds.

4.2. First Adversarial Method (Retraining)

We first introduce our adversarial unlearning process against both reproducing and backdoor verification. Recall that the adversary's goal is to satisfy the unlearning verification while preserving the information of unlearned data. To satisfy the reproducing verification with a 0 verification error

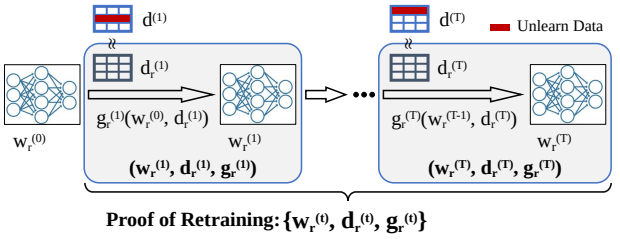


Figure 2. An illustration of the retraining-based adversarial unlearning framework. The PoRT is generated based on the retraining process where the mini-batch $d_r^{(t)} \in \mathcal{D} \backslash \mathcal{D}_u$ sampling is guided by the similarity with $d^{(t)} \in \mathcal{D}$ in gradient.

for each updating function, the model provider has to follow exactly the retraining process to generate the PoRT, strictly ensuring the reproducible and removable properties. However, inspired by recent studies of the possibility that different training data might lead to a similar gradient descent step (Shumailov et al., 2021; Thudi et al., 2022), we can find data in the retained set $\mathcal{D} \backslash \mathcal{D}_u$ that can yield a similar gradient as the unlearned data. In particular, this idea is based on the commonly used mini-batch gradient descent. Under full-batch gradient descent, the gradients in the original training and the retraining are computed over the whole $\mathcal{D}$ and $\mathcal{D} \backslash \mathcal{D}_u$. Their difference is deterministic and cannot be manipulated. In contrast, under mini-batch gradient descent, the gradients in the original training and the retraining are computed over *random samples* in $\mathcal{D}$ and $\mathcal{D} \backslash \mathcal{D}_u$. With the randomness, it is possible to deliberately select the samples that induce a similar model update as the mini-batches in $\mathcal{D}$ from $\mathcal{D} \backslash \mathcal{D}_u$ in the retraining. In this way, the retrained model might still learn from the unlearned data as the original training, even without explicitly using them.

As discussed above, we convert the problem of the adversarial unlearning process into selecting $d_r^{(t)} \in \mathcal{D} \backslash \mathcal{D}_u$ that yields similar model updates as $d^{(t)} \in \mathcal{D}$. It is evident that if the original mini-batch contains no unlearned data $d^{(t)} \cap \mathcal{D}_u = \emptyset$, we can directly set $d_r^{(t)} = d^{(t)}$. If the original mini-batch contains unlearned data, then our goal is to find $d_r^{(t)} \in \mathcal{D} \backslash \mathcal{D}_u$ that makes $\nabla \mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d^{(t)})$ and $\nabla \mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)})$ as close as possible. Regarding this goal, previous studies simply leveraged random sampling multiple times and chose the one yielding the smallest distance (Shumailov et al., 2021; Thudi et al., 2022).

$$d_r^{(t)} = \text{argmin}_{d_i} \|\nabla \mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d^{(t)}) - \nabla \mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d_i)\|, \quad (3)$$

where $i = 1, \ldots, s, d_1, \ldots, d_s \sim \mathcal{D} \backslash \mathcal{D}_u$, and we denote the right-hand side of Equation (3) as $\mathcal{S}_r(\boldsymbol{w}_r^{(t-1)}; d^{(t)})$. However, this method is computationally expensive and does not provide theoretical guarantees. In this paper, we replace the unlearned data in the original batch: $(\boldsymbol{x}_u, y_u) \in d^{(t)} \cap \mathcal{D}_u$

Algorithm 1 Retraining-based Adversarial Unlearning Algorithm

Input: Training data $\mathcal{D}$, unlearned data $\mathcal{D}_u$.
Output: Proof of Retraining $\mathcal{P}_r$.
Initialize $\boldsymbol{w}_r^{(0)}$ and $\mathcal{P}_r \leftarrow \emptyset$.
for $t = 1$ **to** T **do**
  Uniform mini-batch sampling $d^{(t)} \in \mathcal{D}$.
  Choose $d_r^{(t)} \leftarrow \mathcal{S}_r(\boldsymbol{w}_r^{(t-1)}; d^{(t)})$ or $d_r^{(t)} \leftarrow \mathcal{S}_n(d^{(t)})$.
  $\boldsymbol{w}_r^{(t)} \leftarrow g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)})$.
  $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup (\boldsymbol{w}_r^{(t)}, d_r^{(t)}, g_r^{(t)})$.
end for

with its (class-wise) closest neighbor in the retained data:

$$\mathcal{N}(\boldsymbol{x}_u, y_u) = \text{argmin}_{(\boldsymbol{x},y)\in\mathcal{D}\setminus\mathcal{D}_u, y=y_u} \|\boldsymbol{x} - \boldsymbol{x}_u\|. \quad (4)$$

Consequently, our proposed mini-batch selection method can be expressed as

$$\mathcal{S}_n(d^{(t)}) = (d^{(t)}\setminus\mathcal{D}_u) \cup \{\mathcal{N}(\boldsymbol{x}, y) | (\boldsymbol{x}, y) \in d^{(t)} \cap \mathcal{D}_u\}. \quad (5)$$

It is worth noting that we observe that using $\mathcal{S}_r$ can be better than $\mathcal{S}_n$ sometimes in practice, i.e., $\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, \mathcal{S}_r(\boldsymbol{w}_r^{(t-1)}; d^{(t)}))$ might be closer to $\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d^{(t)})$ than $\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, \mathcal{S}_n(\boldsymbol{w}_r^{(t-1)}; d^{(t)}))$, as long as setting a large enough sample size s. However, our proposed closest neighbor selection is still necessary as it provides a worst-case upper bound for theoretical analysis and a more efficient way of adversarial unlearning when considering real-world threats. Consequently, we can generate a PoRT based on our adversarial unlearning process with carefully selected mini-batches. We provide a clear illustration of our adversarial unlearning framework in Figure 2. In addition, the detailed algorithm is shown in Algorithm 1. Specifically, the following properties can be guaranteed for Algorithm 1.

Proposition 4.2. *Algorithm 1 returns a valid Proof of Retraining under the threshold $\varepsilon = 0$.*

Proposition 4.3. *Let $C_D := \max_c \max_{\boldsymbol{x}\in c} \min_{\boldsymbol{z}\in c} \|\boldsymbol{x} - \boldsymbol{z}\|$, where c denotes the class in the label domain. For the continuity of loss functions, we assume (i). $\|\nabla_{\boldsymbol{w}} l(\boldsymbol{w}, \boldsymbol{x})\| \leq G$, (ii). $\|\frac{\partial^2 l(\boldsymbol{w},\boldsymbol{x})}{\partial \boldsymbol{w}\partial \boldsymbol{x}}\| \leq L_x$, and (iii). $\|\nabla^2_{\boldsymbol{w}} l(\boldsymbol{w}, \boldsymbol{x})\| \leq L$. Let $\gamma^{(t)} = \gamma \leq \frac{1}{L}$ and m be the size of mini-batches, for Algorithm 1, we have*

$$\mathrm{E}_T[\|\nabla\mathcal{L}(\boldsymbol{w}_r^{(T)}, \mathcal{D})\|^2] \leq \frac{\mathcal{L}(\boldsymbol{w}_r^{(0)}, \mathcal{D}) - \mathcal{L}(\boldsymbol{w}_r^{(T)}, \mathcal{D})}{\gamma T} + \frac{\gamma L}{2}(G^2 + p_u B^2) + (1 - \gamma L) p_u G B,$$

where $p_u = 1 - (1 - \frac{|\mathcal{D}_u|}{|\mathcal{D}|})^m$ and $B = L_x C_D$.

Assumptions (i) and (iii) are the same as in (Ajalloeian & Stich, 2020), indicating that $l(\cdot, \boldsymbol{x})$ is L-smooth and G-Lipschitz continuous. In addition, Assumption (ii) is the same as in (Thudi et al., 2022), indicating that $\nabla_{\boldsymbol{w}} l(\boldsymbol{w}, \cdot)$ is L_x-Lipschitz. We provide the proof of Proposition 4.2 and Proposition 4.3 in Appendix A. Proposition 4.2 ensures that Algorithm 1 can satisfy the reproducing verification; Proposition 4.3 guarantees that the retraining process can reach a neighborhood of the optimum over $\mathcal{D}$, i.e., the adversarial unlearning process can still make the retrained model learn from $\mathcal{D}_u \subset \mathcal{D}$. Next, regarding the backdoor verification, if the retrained model reaches the exact optimum over $\mathcal{D}$, the backdoor poisoned data in the unlearned set can still be triggered, and the adversarial unlearning process will be recognized. However, according to Proposition 4.3, the radius of the neighborhood $\frac{L\gamma}{2}(G^2 + p_u B^2) + (1 - \gamma L) p_u G B$ (the distance between the retrained model and the original optimum) will increase after injecting backdoor data, which reduces the probability of backdoor data being triggered. The rationale is that after flipping the label of the backdoor data $\boldsymbol{x}_b$ from y_b to y_b', the value of $\min_{\boldsymbol{x}\in c_{y_b'}} \|\boldsymbol{x} - \boldsymbol{x}_b\|$ can increase because $\boldsymbol{x}_b$ is actually in the class y_b with a different distribution. Instead of learning the mapping of backdoor data $\boldsymbol{x}_b \rightarrow y_b'$, our adversarial unlearning process uses the data truly from the class y_b' to replace each of the backdoor data. Hence, our unlearned model cannot be triggered by the backdoor poisoned data.

4.3. Second Adversarial Method (Forging)

Although our first adversarial unlearning method can deceive the verification of MUL while preserving the information of unlearned data, the computational overhead of our method is not better than naive retraining, limiting the benefits of the model provider earned from the adversarial unlearning process. Hence, we propose another adversarial unlearning process that is more computationally efficient. As a trade-off, the power of our second adversarial method becomes weaker, i.e., it can only deceive the reproducing verification under an $\varepsilon > 0$ threshold. Inspired by (Thudi et al., 2022), if the model provider records a PoT during the original training, the computation of generating a PoRT can be reduced by reusing the PoT. In (Thudi et al., 2022), the model provider can update the PoT using a forging map and obtain a valid PoRT under the ε-threshold reproducing verification. In particular, the forging map is to replace the overlapping batches $d^{(t)} \cap \mathcal{D}_u \neq \emptyset$ with a batch $d_r^{(t)} \in \mathcal{D}\setminus\mathcal{D}_u$ that induces a similar model update as $d^{(t)}$ in each triplet $\{\boldsymbol{w}^{(t)}, d^{(t)}, g^{(t)}\}$, i.e., $\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d^{(t)}) \approx \nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d_r^{(t)})$. However, the forging map is not realistic because it preserves the model parameters unchanged $\boldsymbol{w}_r^{(t)} = \boldsymbol{w}^{(t)}$, which can be easily recognized by the verifier (under multiple verification requests, the verifier will receive PoRTs with the same model parameters in each time). In this paper, we propose a novel forging map $\mathcal{F} : \mathcal{P}_t \rightarrow \mathcal{P}_r$ that *directly updates both the model parameters and the batch data of each triplet in*

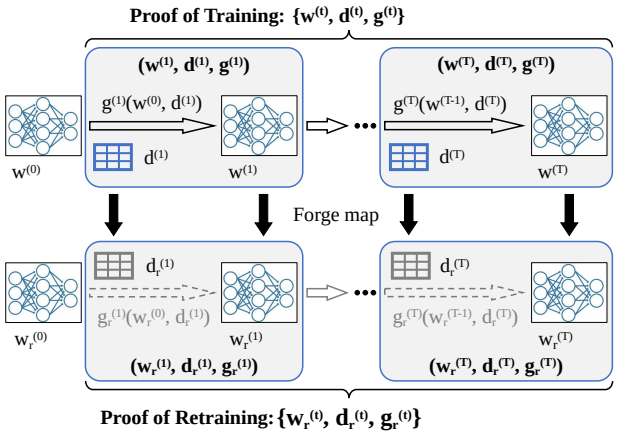


Figure 3. An illustration of the forging-based adversarial unlearning framework. Different from the retraining-based adversarial method, the PoRT here is generated directly from the PoT recorded in the original training. $\boldsymbol{w}_r^{(t)}$ (with $d_r^{(t)}$) is obtained by conducting the forging map over the PoT instead of using the model updating function $g_r^{(t)}$.

the PoT to generate a valid PoRT instead of retraining. We formulate our proposed forging map as a triplet-wise updating function, i.e., for any $t \in \mathcal{I}$,

$$(\boldsymbol{w}_r^{(t)}, d_r^{(t)}, g_r^{(t)}) = \mathcal{F}^{(t)}(\boldsymbol{w}^{(t)}, d^{(t)}, g^{(t)}). \tag{6}$$

In this way, each triplet in $\mathcal{P}_r$ can be generated separately based on the corresponding triplet in $\mathcal{P}_t$. Normally, the model updating function g remains unchanged during the retraining process, and we mainly focus on updating $\boldsymbol{w}^{(t)}$ and $d^{(t)}$ with $\mathcal{F}^{(t)}$. We divide the updating of $\boldsymbol{w}^{(t)}$ and $d^{(t)}$ into two cases: excluding unlearned data $\mathcal{I}_e = \{t \,|\, d^{(t)} \cap \mathcal{D}_u = \emptyset\}$ and including unlearned data $\mathcal{I}_n = \{t \,|\, d^{(t)} \cap \mathcal{D}_u \neq \emptyset\}$ ($\mathcal{I}_e$ and $\mathcal{I}_n$ are used to denote the index set of the triplets in different cases). We assume that the number of unlearned data is far less than the retained data. Next, we discuss the two cases separately.

(1). For $t \in \mathcal{I}_e$, although the data $d^{(t)}$ requires no modification (we can simply let $d_r^{(t)} = d^{(t)}$), we still need to slightly alter the model parameters $\boldsymbol{w}^{(t)}$ because otherwise, the verifier might notice a large proportion of unchanged models comparing $\boldsymbol{w}_r^{(t)}$ with $\boldsymbol{w}^{(t)}$. On the other hand, the majority of triplets should be updated carefully because the model utility should not decrease very much after multiple verification requests. Considering both requirements for efficiency and model utility, we use single-step stochastic gradient descent (SGD) to update the model parameters in $\mathcal{F}^{(t)}$ and have

$$\boldsymbol{w}_r^{(t)} = \boldsymbol{w}^{(t)} - \gamma_r^{(t)} \nabla l(f_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(t)}), y^{(t)}), \tag{7}$$

where $(\boldsymbol{x}^{(t)}, y^{(t)}) \sim \mathcal{D} \backslash \mathcal{D}_u$ is a random sample chosen from the retained data and $\gamma_r^{(t)}$ is a small value to control the verification error $\|\boldsymbol{w}_r^{(t)} - g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)})\| \leq \varepsilon$.

Algorithm 2 Forging-based Adversarial Unlearning Algorithm

Input: Training data $\mathcal{D}$, unlearned data $\mathcal{D}_u$, closest-neighbor mapping $\mathcal{N} : \mathcal{D}_u \to \mathcal{D} \backslash \mathcal{D}_u$, Proof of Training $\mathcal{P}_t = \{\boldsymbol{w}^{(t)}, d^{(t)}, g^{(t)}\}$.
Output: Proof of Retraining $\mathcal{P}_r$.
$\mathcal{P}_r \leftarrow \emptyset$.
for $t = 1$ **to** T **in parallel do**
  if $d^{(t)} \cap \mathcal{D}_u = \emptyset$ **then**
    $d_r^{(t)} \leftarrow d^{(t)}$.
    $\boldsymbol{w}_r^{(t)} \leftarrow \boldsymbol{w}^{(t)} - \gamma_r^{(t)} \nabla l(f_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(t)}), y^{(t)})$.
  else
    $d_r^{(t)} \leftarrow \mathcal{S}_n(d^{(t)})$.
    $\boldsymbol{w}_r^{(t)} \leftarrow g^{(t)}(\boldsymbol{w}^{(t-1)}, d_r^{(t)})$.
  end if
  $g_r^{(t)} \leftarrow g^{(t)}$.
  $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup (\boldsymbol{w}_r^{(t)}, d_r^{(t)}, g_r^{(t)})$.
end for

(2). For $t \in \mathcal{I}_n$, the forging map should remove the unlearned data from $d^{(t)}$. To reduce the verification error, we still exploit the same strategy in our first adversarial method to select the mini-batch $d_r^{(t)} \in \mathcal{D} \backslash \mathcal{D}_u$ with $\mathcal{S}_n(d^{(t)})$ (we do not use $\mathcal{S}_r$ due to the efficiency issue). Consequently, to update the model parameters in the forging map $\mathcal{F}^{(t)}$, we let

$$\boldsymbol{w}_r^{(t)} = g^{(t)}(\boldsymbol{w}^{(t-1)}, \mathcal{S}_n(d^{(t)})). \tag{8}$$

We conclude our second adversarial unlearning process in Algorithm 2 and provide a distinct illustration in Figure 3. The advantage of Algorithm 2 is that the forging map can be implemented in parallel thanks to our formulation in Equation (6), which largely reduces the execution time in real-world scenarios with frequent unlearning requests. Specifically, we analyze the condition of Algorithm 2 satisfying the reproducing verification and the time complexity of Algorithm 2 as follows.

Proposition 4.4. *Under the same assumptions as in Proposition 4.3, when $g_r^{(t)}$ is the vanilla SGD, $0 \leq \gamma^{(t)} \leq \frac{1}{2L}(\sqrt{9 + 4\varepsilon L/(L_x C_D)} - 3)$, and $0 \leq \gamma_r^{(t)} \leq \frac{\varepsilon - \gamma^{(t)} L_x C_D}{G(2+\gamma^{(t)} L)}$, Algorithm 2 returns a valid Proof of Retraining under $\varepsilon > 0$ threshold.*

Proposition 4.5. *Assume the time complexity of naive retraining is $T(n)$. The time complexity of Algorithm 1 is $T(n)$ and the time complexity of Algorithm 2 is $(\frac{p_u}{P} + \frac{1-p_u}{mP}) \cdot T(n)$, where $p_u = 1 - (1 - |\mathcal{D}_u|/|\mathcal{D}|)^m$, P denotes the number of processes in parallel, and m denotes the batch size.*

The proofs of Proposition 4.4 and Proposition 4.5 can be found in Appendix A. Proposition 4.4 provides a theoretical guarantee when Algorithm 2 successfully deceives the ε-threshold reproducing verification. Proposition 4.5 com-

Table 1. Dataset statistics.

Dataset	# Train	# Test	# Class	Image Size
MNIST	60,000	10,000	10	28×28
CIFAR-10	50,000	10,000	10	32×32×3
SVHN	73,257	26,032	10	32×32×3

pares the time complexity of Algorithm 1 and Algorithm 2 with naive retraining and demonstrates the superiority of Algorithm 2 in computational efficiency. Despite the efficiency of Algorithm 2, it can fail to satisfy the backdoor verification because the final unlearned model $\boldsymbol{w}_r^{(T)}$ is still dependent on the information of injected backdoor data.

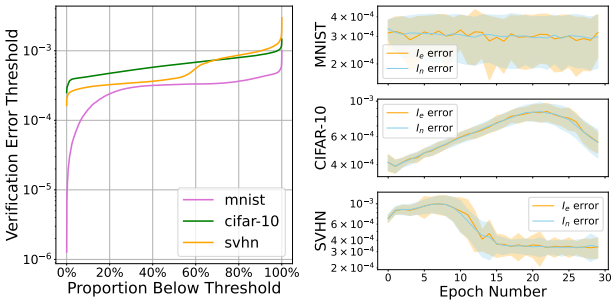
5. Experiments

In this section, we empirically evaluate the vulnerability of MUL verification with numerical experiments.

Datasets. Our experiments are based on three widely adopted real-world datasets for image classification, MNIST (LeCun et al., 1998), SVHN (Netzer et al., 2011), and CIFAR-10 (Krizhevsky et al., 2009): the MNIST dataset consists of a collection of handwritten digit images; the CIFAR-10 dataset contains color images in 10 classes, with each class representing a specific object category, e.g., cats and automobiles; the SVHN dataset consists of house numbers images captured from Google Street View. The statistics of these three datasets are shown in Table 1. All datasets are publicly accessible (MNIST with GNU General Public License, CIFAR-10 with MIT License, and SVHN with CC BY-NC License).

Evaluation Metrics. Recall that the goal of adversarial unlearning processes is to preserve the model utility and improve the efficiency of unlearning while circumventing the verification methods. To demonstrate the efficacy of our adversarial unlearning methods, we use the verification error threshold ε to evaluate the forging-based method and use the probability of type II errors (the target data is not unlearned is regarded as the null hypothesis) to evaluate our retraining-based method. In addition, we should also ensure that adversarial unlearning methods benefit model providers in preserving model utility and improving efficiency. Hence, we use utility metrics (e.g., F1-score) and the execution time to verify the benefit of adversarial unlearning methods.

Implementation. We implemented all experiments in the PyTorch (Paszke et al., 2019) library and exploited SGD as the optimizer for training. All experiments were conducted on an Nvidia RTX A6000 GPU. We use the verification error threshold ε to evaluate the forging-based method and use the probability of type II errors (the verifier thinks target data is unlearned but actually not) to evaluate the retraining-based method. We use utility metrics (e.g., F1-score) and the execution time to show the benefit of adversarial unlearning. We report the average value and standard deviation of the numerical results under five random seeds. More details of the hyperparameter setting are presented in Appendix B. Our code is available at https://github.com/zhangbinchi/unlearning-verification-is-fragile.



(a) Verification error (b) Details of error statistics

Figure 4. Verification error of forging-based adversarial unlearning method for MLP over MNIST, CNN over CIFAR-10, and ResNet over SVHN.

Table 2. Probability of the type II error (β value) of backdoor verification on different (adversarial) unlearning strategies over three datasets.

Method	MNIST	CIFAR-10	SVHN
Original	2.61×10^{-42}	5.14×10^{-20}	1.11×10^{-28}
Retrain	0.998	0.998	0.999
Adv-R	0.997	0.997	0.999
Adv-F	5.46×10^{-34}	2.78×10^{-16}	2.08×10^{-26}

5.1. Adversarial Unlearning vs Verification

We first empirically demonstrate the efficacy of our proposed adversarial unlearning methods, i.e., they can satisfy the verification and let the verifier believe that the data is unlearned normally. We next discuss the reproducing verification and the backdoor verification separately.

Reproducing Verification. For reproducing verification, our retraining-based method can always satisfy the verification with an exact 0 verification error because the PoRT is recorded directly based on the retraining process. Hence, we mainly focus on the efficacy of the forging-based method in this experiment. As the choice of verification error threshold ε is highly personalized, we directly fix the hyperparameters of our forging-based method and record the verification error in each model updating step. Specifically, we randomly choose 2% data as the unlearned set. The hyperparameter settings are shown in Appendix B, and the experimental results are shown in Figure 4. We can observe that the verification errors in most steps for all datasets are below $1e^{-3}$, i.e., our forging-based method can satisfy the reproducing verification with a $1e^{-3}$ threshold. Compared with the scale

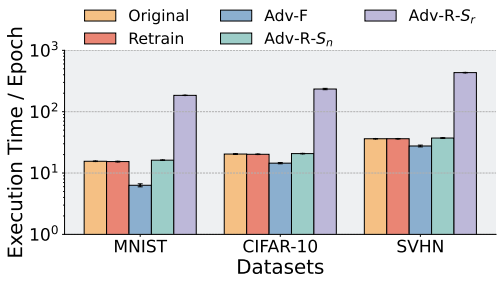


Figure 5. Comparison of execution time among original training, naive retraining, and adversarial unlearning methods over three real-world datasets.

of adopted neural networks, $1e^{-3}$ can be seen as a small value. We can also see a reduction in the overall verification error as the model scale decreases (from ResNet to MLP). In addition, we also illustrate the mean value and the standard deviation of the verification error of $\mathcal{I}_e$ (excluding $\mathcal{D}_u$) and $\mathcal{I}_n$ (including $\mathcal{D}_u$) steps in different epochs and obtain the following observations: 1) the verification errors in $\mathcal{I}_e$ and $\mathcal{I}_n$ steps are consistent, perhaps due to the high dependence of both errors on the gradient scale; 2) within the overall threshold, the verification error tends to first increase and then decrease until staying nearby a local optimum; again, we attribute this to the dependence of errors on gradients.

Backdoor Verification. For backdoor verification, we exploit Athena (Sommer et al., 2022) as the verification strategy. In particular, we randomly choose the backdoor training and test data, inject a specific pattern of pixels, and change the label of training data to a fixed target label. To test the integrity of unlearning, we make unlearned data include the backdoor training data. If the backdoor success rate is close to the random prediction, the model can predict the original labels for the backdoor test data, and the unlearned model was not trained on the poisoned data. Thus, the verifier believes that the model provider follows the unlearning request. Specifically, we regard "the target data is unlearned" as the null hypothesis and "the target data is not unlearned" as the alternative hypothesis. We use the probability of type II errors (the verifier thinks target data is unlearned but actually not) to evaluate our adversarial unlearning methods. The results are shown in Table 2. Consistent with our former discussion, the retraining-based adversarial method and naive retraining are almost always regarded as truly unlearning the target data; the forging-based adversarial method and original training are hardly regarded as truly unlearning the target data.

5.2. Adversary's Goal

In previous experiments, we have demonstrated that our proposed adversarial unlearning methods can satisfy reproducing and backdoor verification. Next, we aim to show that 1) the adversarial unlearning deceives the verification: they still memorize the information of unlearned data, and 2) model providers benefit from adversarial unlearning: they preserve the model utility and improve unlearning efficiency.

Utility. In this experiment, we compare the model utility between naive retraining and our adversarial unlearning. To simulate the data heterogeneity in real-world scenarios, we add a class-imbalanced unlearning setting. For the retraining-based adversarial method, we adopt both random sampling $\mathcal{S}_r$ and nearest neighbor $\mathcal{S}_n$ approaches to select mini-batches. For the forging-based adversarial method, we use the last recorded model on the PoRT $\boldsymbol{w}_r^{(T)}$ for utility evaluation. To demonstrate the long-term effect of adversarial unlearning, we divide 10% of the training data as the unlearned set. Details of hyperparameter settings and complete experimental results can be found in Appendix B.1, and we provide the truncated experimental results in Table 3. From the results, we can obtain the following observations: 1) in the normal setting, the retraining-based adversarial method has a similar utility to naive retraining; 2) in the class-imbalanced setting, the retraining-based adversarial method has a much better utility than naive retraining, which means model providers can benefit more when data heterogeneity exists in the unlearning process. 3) in both settings, the forging-based adversarial method has the best utility of the unlearned model (close to the original training).

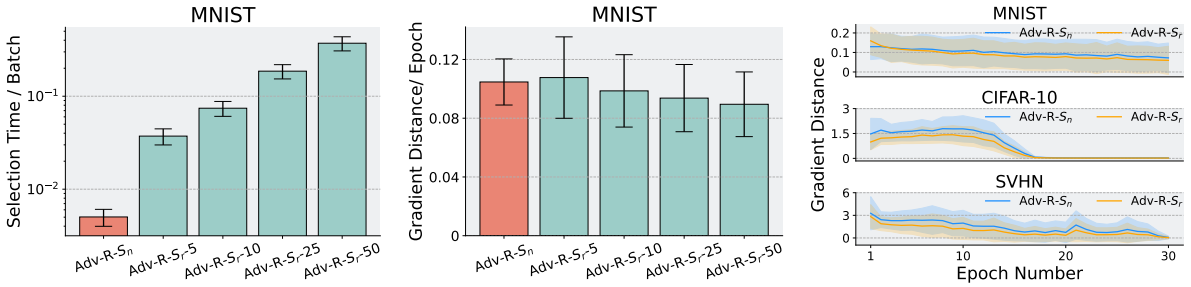
Efficiency. In this experiment, we compare the computational efficiency between naive retraining and our adversarial unlearning. For the retraining-based adversarial method, we use the nearest neighbor selection to choose the mini-batches and obtain the nearest neighbor mapping by ranking the pre-computed distance between the unlearned data and the retained data. For the forging-based adversarial method, we adopt five processes in parallel to compute the PoRT. We record the average execution time for each epoch and show the results in Figure 5. From the results, we can obtain that 1) the forging-based method has the shortest execution time; 2) the time cost of forging in practice can be longer than theoretical results due to the resource bottleneck of the adopted single GPU; 3) the retraining-based method is slightly slower than naive retraining due to the nearest neighbor calculation, but our proposed nearest neighbor selection is much faster than random sampling.

5.3. Mini-batch Selection: Nearest Neighbor vs Random Sampling

In Section 4.2, we introduced two mini-batch selection strategies for the retraining-based method: random sampling $\mathcal{S}_r$ and nearest neighbor $\mathcal{S}_n$. Additionally, in Section 5.2, we demonstrated that our nearest neighbor selection strategy is faster than random sampling. This section delves deeper into comparing these two strategies.

Table 3. Comparison of the model utility among original training, naive retraining, and adversarial unlearning methods over three popular DNNs across three real-world datasets. We record the macro F1-score of the predictions on the unlearned set $\mathcal{D}_u$, retained set $\mathcal{D}\backslash\mathcal{D}_u$, and test set $\mathcal{D}_t$. The prefix 'im-' denotes the results in the class-imbalanced setting.

Method	MLP & MNIST			CNN & CIFAR-10			ResNet & SVHN		
	$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$	$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$	$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$
Original	99.47 ± 0.09	99.76 ± 0.08	97.00 ± 0.17	100.00 ± 0.00	100.00 ± 0.00	85.33 ± 0.31	100.00 ± 0.00	100.00 ± 0.00	94.91 ± 0.09
Retrain	96.43 ± 0.19	99.52 ± 0.10	96.75 ± 0.13	83.60 ± 0.31	100.00 ± 0.00	83.12 ± 0.23	94.33 ± 0.24	100.00 ± 0.00	94.57 ± 0.06
Adv-R ($\mathcal{S}_r$)	98.17 ± 0.16	99.33 ± 0.18	96.78 ± 0.13	83.81 ± 0.44	100.00 ± 0.00	83.08 ± 0.34	94.38 ± 0.11	100.00 ± 0.00	94.54 ± 0.09
Adv-R ($\mathcal{S}_n$)	96.34 ± 0.11	98.65 ± 0.19	96.60 ± 0.14	82.40 ± 0.39	100.00 ± 0.00	81.85 ± 0.44	94.64 ± 0.20	100.00 ± 0.00	94.75 ± 0.04
Adv-F	99.30 ± 0.13	99.33 ± 0.10	96.94 ± 0.14	100.00 ± 0.00	100.00 ± 0.00	85.20 ± 0.24	100.00 ± 0.00	100.00 ± 0.00	94.91 ± 0.07
im-Original	60.29 ± 13.07	97.00 ± 2.60	96.88 ± 0.07	100.00 ± 0.00	100.00 ± 0.00	85.44 ± 0.22	100.00 ± 0.00	100.00 ± 0.00	94.66 ± 0.30
im-Retrain	38.76 ± 13.41	95.86 ± 4.34	89.92 ± 5.70	24.25 ± 6.98	90.88 ± 5.03	65.22 ± 5.94	33.08 ± 11.97	95.19 ± 3.78	83.89 ± 5.83
im-Adv-R ($\mathcal{S}_r$)	39.48 ± 12.20	99.71 ± 0.19	91.04 ± 4.80	25.94 ± 6.89	96.00 ± 4.90	76.51 ± 4.32	34.76 ± 12.06	98.00 ± 4.01	87.63 ± 6.11
im-Adv-R ($\mathcal{S}_n$)	42.90 ± 11.87	97.96 ± 0.59	92.80 ± 4.54	23.97 ± 8.75	91.46 ± 1.81	67.34 ± 4.02	33.92 ± 11.85	99.37 ± 0.20	84.87 ± 5.42
im-Adv-F	64.21 ± 9.89	97.28 ± 3.34	96.81 ± 0.04	100.00 ± 0.00	100.00 ± 0.00	85.11 ± 0.21	100.00 ± 0.00	100.00 ± 0.00	94.77 ± 0.09



(a) Selection time of two mini-batch selection strategies.

(b) Gradient distance between mini-batch selection strategies and the original batch.

(c) Gradient distance in different datasets.

Figure 6. Comparison of two mini-batch selection strategies: random sampling $\mathcal{S}_r$ and nearest neighbor $\mathcal{S}_n$.

Figure 6(b) and Figure 6(a) illustrate the selection time per epoch and the gradient distance between selected batches and their corresponding batches containing unlearn data for both strategies. We conduct these experiments on the MNIST dataset, varying the number of batch samples M for random sampling. In this strategy, M candidate batches are selected, and the one with the smallest gradient distance to the original batch is chosen. We tested M values of 5, 10, 25, and 50.

From Figure 6(a), we observe that: i) in the random sampling strategy, selection time increases with M due to the increased gradient computations; ii) our nearest neighbor selection consistently outperforms random sampling in speed, even with smaller M values, by avoiding extra gradient computations. Figure 6(b) shows that: i) larger M values in random sampling lead to better batch selection; ii) nearest neighbor selection performs better than random sampling with small M values and is comparable when M is large. Further, we extended our experiments to two additional datasets with $M = 50$ for the random sampling strategy. As depicted in Figure 6(c), our strategy exhibits similar performance to random sampling. In summary, the nearest neighbor strategy offers significantly faster selection times and comparable model training performance compared to the random sampling strategy.

6. Conclusion

In this paper, we expose the vulnerability in the verification of MUL. In particular, we summarize current verification strategies into two types. Regarding both types, we propose two adversarial unlearning processes that circumvent the verification while preserving the information of unlearned data. Our theoretical and empirical results highlight the following conclusions: the adversarial unlearning method can circumvent the verification methods while improving the model utility without extra computation costs. Put another way of thinking, by adopting an adversarial unlearning method, a dishonest model provider gains a higher model utility (still memorizes the unlearned data) after unlearning without compromising efficiency. Furthermore, the retraining-based adversarial method can perfectly circumvent all existing verification methods relying on the natural similarity within the training data and the randomness of the training algorithm. This threat poses a novel challenge for the safe verification of machine unlearning: there is no strict guarantee for the verification of MUL. Better verification methods are needed to obtain precise and reliable verification results.

Acknowledgements

This work is supported in part by the National Science Foundation under grants (IIS-2006844, IIS-2144209, IIS-2223769, CNS-2154962, BCS-2228534, ECCS-2033671, ECCS-2143559, CPS-2313110, and SII-2132700), the Commonwealth Cyber Initiative Awards under grants (VV-1Q23-007, HV-2Q23-003, and VV-1Q24-011), the JP Morgan Chase Faculty Research Award, and the Cisco Faculty Research Award.

Impact Statement

Our proposed adversarial unlearning methods in this paper can threaten the safety of the verification of MUL. Generally, we can consider that a model provider trains an ML model based on some personal data collected from data owners. According to the legislation (CCPA, 2018; GDPR, 2016), the data owners have the right to have their data removed from the ML model and take actions to verify the efficacy of unlearning. However, using our proposed adversarial unlearning algorithms, the model provider can make the data owners (verifiers) believe that their data has been unlearned while preserving the information of these personal data.

Despite that, our research has a more significant positive influence compared with the potential risks. The study of MUL verification is still at a nascent stage. Given the deficient understanding of the safety of MUL verification, our study highlights the vulnerability of current verification strategies of MUL and inspires further research on the safe verification of MUL. Moreover, we tentatively provide discussions on the weak points of our proposed adversarial unlearning strategies and insights for detecting these misbehaviors in Appendix C.

References

Ajalloeian, A. and Stich, S. U. On the convergence of sgd with biased gradients. *arXiv preprint arXiv:2008.00051*, 2020.

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

Boyd, S. P. and Vandenberghe, L. *Convex optimization*. Cambridge university press, 2004.

Cao, Y. and Yang, J. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy (SP)*, pp. 463–480, 2015.

CCPA. California consumer privacy act. 2018. URL https://oag.ca.gov/privacy/ccpa.

Che, T., Zhou, Y., Zhang, Z., Lyu, L., Liu, J., Yan, D., Dou, D., and Huan, J. Fast federated machine unlearning with nonlinear functional theory. In *International conference on machine learning*, pp. 4241–4268, 2023.

Chen, M., Zhang, Z., Wang, T., Backes, M., Humbert, M., and Zhang, Y. When machine unlearning jeopardizes privacy. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pp. 896–911, 2021.

Chen, M., Zhang, Z., Wang, T., Backes, M., Humbert, M., and Zhang, Y. Graph unlearning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 499–513, 2022.

Chien, E., Pan, C., and Milenkovic, O. Efficient model updates for approximate unlearning of graph-structured data. In *International Conference on Learning Representations*, 2023.

Chundawat, V. S., Tarun, A. K., Mandal, M., and Kankanhalli, M. Zero-shot machine unlearning. *IEEE Transactions on Information Forensics and Security*, 2023.

Costan, V. and Devadas, S. Intel sgx explained. *Cryptology ePrint Archive*, 2016.

Eisenhofer, T., Riepel, D., Chandrasekaran, V., Ghosh, E., Ohrimenko, O., and Papernot, N. Verifiable and provably secure machine unlearning. *arXiv preprint arXiv:2210.09126*, 2022.

Gao, X., Ma, X., Wang, J., Sun, Y., Li, B., Ji, S., Cheng, P., and Chen, J. Verifi: Towards verifiable federated unlearning. *arXiv preprint arXiv:2205.12709*, 2022.

GDPR. General data protection regulation. 2016. URL https://gdpr-info.eu/.

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

Golatkar, A., Achille, A., Ravichandran, A., Polito, M., and Soatto, S. Mixed-privacy forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 792–801, 2021.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.

Guo, C., Goldstein, T., Hannun, A., and Van Der Maaten, L. Certified data removal from machine learning models. In *International Conference on Machine Learning*, pp. 3832–3842, 2020.

Guo, Y., Zhao, Y., Hou, S., Wang, C., and Jia, X. Verifying in the dark: Verifiable machine unlearning by using invisible backdoor triggers. *IEEE Transactions on Information Forensics and Security*, 2023.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Izzo, Z., Smart, M. A., Chaudhuri, K., and Zou, J. Approximate data deletion from machine learning models. In *International Conference on Artificial Intelligence and Statistics*, pp. 2008–2016, 2021.

Kim, J. and Woo, S. S. Efficient two-stage model retraining for machine unlearning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4361–4369, 2022.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894, 2017.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Le, Y. and Yang, X. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

maintainers, T. and contributors. Torchvision: Pytorch's computer vision library. https://github.com/pytorch/vision, 2016.

Mehta, R., Pal, S., Singh, V., and Ravi, S. N. Deep unlearning via randomized conditionally independent hessians. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10422–10431, 2022.

Neel, S., Roth, A., and Sharifi-Malvajerdi, S. Descent-to-delete: Gradient-based methods for machine unlearning. In *Algorithmic Learning Theory*, pp. 931–962, 2021.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. 2011.

Nguyen, T. T., Huynh, T. T., Nguyen, P. L., Liew, A. W.-C., Yin, H., and Nguyen, Q. V. H. A survey of machine unlearning. *arXiv preprint arXiv:2209.02299*, 2022.

Pan, C., Chien, E., and Milenkovic, O. Unlearning graph classifiers with limited data resources. In *Proceedings of the ACM Web Conference 2023*, pp. 716–726, 2023.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Sekhari, A., Acharya, J., Kamath, G., and Suresh, A. T. Remember what you want to forget: Algorithms for machine unlearning. *Advances in Neural Information Processing Systems*, 34:18075–18086, 2021.

Setty, S. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pp. 704–737, 2020.

Shafran, A., Peleg, S., and Hoshen, Y. Membership inference attacks are easier on difficult problems. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14820–14829, 2021.

Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pp. 3–18, 2017.

Shumailov, I., Shumaylov, Z., Kazhdan, D., Zhao, Y., Papernot, N., Erdogdu, M. A., and Anderson, R. J. Manipulating sgd with data ordering attacks. *Advances in Neural Information Processing Systems*, 34:18021–18032, 2021.

Sommer, D. M., Song, L., Wagh, S., and Mittal, P. Athena: Probabilistic verification of machine unlearning. *Proc. Privacy Enhancing Technol*, 3:268–290, 2022.

Tarun, A. K., Chundawat, V. S., Mandal, M., and Kankanhalli, M. Deep regression unlearning. In *International Conference on Machine Learning*, pp. 33921–33939, 2023.

Thudi, A., Jia, H., Shumailov, I., and Papernot, N. On the necessity of auditable algorithmic definitions for machine unlearning. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 4007–4022, 2022.

Ullah, E., Mai, T., Rao, A., Rossi, R. A., and Arora, R. Machine unlearning via algorithmic stability. In *Conference on Learning Theory*, pp. 4126–4142, 2021.

Warnecke, A., Pirch, L., Wressnegger, C., and Rieck, K. Machine unlearning of features and labels. In *Network and Distributed System Security Symposium, NDSS*, 2023.

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

Wu, J., Yang, Y., Qian, Y., Sui, Y., Wang, X., and He, X. Gif: A general graph unlearning strategy via influence

function. In *Proceedings of the ACM Web Conference 2023*, pp. 651–661, 2023a.

Wu, K., Shen, J., Ning, Y., Wang, T., and Wang, W. H. Certified edge unlearning for graph neural networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2606–2617, 2023b.

Xu, H., Zhu, T., Zhang, L., Zhou, W., and Yu, P. S. Machine unlearning: A survey. *ACM Computing Surveys*, pp. 1–36, 2023.

Yuan, L., Tay, F. E., Li, G., Wang, T., and Feng, J. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3903–3911, 2020.

Yurochkin, M., Agarwal, M., Ghosh, S., Greenewald, K., Hoang, N., and Khazaeni, Y. Bayesian nonparametric federated learning of neural networks. In *International conference on machine learning*, pp. 7252–7261. PMLR, 2019.

Zhang, Z., Zhou, Y., Zhao, X., Che, T., and Lyu, L. Prompt certified machine unlearning with randomized gradient smoothing and quantization. *Advances in Neural Information Processing Systems*, 35:13433–13455, 2022.

A. Proof

A.1. Proof of Proposition 4.2

Proof. To prove that Algorithm 1 returns a valid PoRT, we verify the two properties of a valid PoRT in Definition 4.1. First, we verify the removable property. For any $d \in \mathcal{S}_r(\boldsymbol{w}_r^{(t-1)}; d^{(t)}) \cup \mathcal{S}_n(d^{(t)})$, we have $d \in \mathcal{D} \backslash \mathcal{D}_u$. Hence, we have $d_r^{(t)} \cap \mathcal{D}_u = \emptyset$. We then verify the reproducible property. It is obvious that $\|\boldsymbol{w}_r^{(t)} - g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)})\| = 0$ and the reproducible property holds for $\varepsilon = 0$. In conclusion, Algorithm 1 returns a valid PoRT. □

A.2. Proof of Proposition 4.3

Proof. Recall that Algorithm 1 can be seen as a mini-batch SGD with the following model updating function:

$$\begin{aligned} \boldsymbol{w}_r^{(t+1)} &= \boldsymbol{w}_r^{(t)} - \gamma \nabla \mathcal{L}(\boldsymbol{w}_r^{(t)}, d_r^{(t+1)}) \\ &= \boldsymbol{w}_r^{(t)} - \gamma \nabla \mathcal{L}(\boldsymbol{w}_r^{(t)}, \mathcal{S}_n(d^{(t+1)})), \end{aligned} \tag{9}$$

where $d^{(t+1)} \sim \mathcal{D}$. It is worth noting that $d^{(t+1)}$ is randomly chosen from $\mathcal{D}$, while $d_r^{(t+1)}$ relies on the selection of $d^{(t+1)}$ and the unlearned set $\mathcal{D}_u$. If we take the expectation value over both sides, it will be difficult to directly compute the expectation $\mathrm{E}[\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)}, d_r^{(t+1)})]$ on the right-hand side. Consequently, we let $\boldsymbol{b}^{(t)} = \nabla \mathcal{L}(\boldsymbol{w}_r^{(t)}, d_r^{(t+1)}) - \nabla \mathcal{L}(\boldsymbol{w}_r^{(t)}, d^{(t+1)})$ and have

$$\boldsymbol{w}_r^{(t+1)} = \boldsymbol{w}_r^{(t)} - \gamma(\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)}, d^{(t+1)}) + \boldsymbol{b}^{(t)}). \tag{10}$$

For simplicity, we denote $\mathcal{L}(\boldsymbol{w})$ as the loss over the whole training set $\mathcal{L}(\boldsymbol{w}, \mathcal{D})$ and $\mathcal{L}_t(\boldsymbol{w})$ as the loss over the original mini-batch at the t-th iteration $\mathcal{L}(\boldsymbol{w}, d^{(t+1)})$. Next, we focus on the loss function value

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}_r^{(t+1)}) &= \mathcal{L}(\boldsymbol{w}_r^{(t)} - \gamma(\nabla \mathcal{L}_t(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)})) \\ &= \mathcal{L}(\boldsymbol{w}_r^{(t)}) - \nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})^\top \gamma(\nabla \mathcal{L}_t(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)}) + \frac{\gamma^2}{2}(\nabla \mathcal{L}_t(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)})^\top \nabla^2 \mathcal{L}(\boldsymbol{\xi}^{(t)})(\nabla \mathcal{L}_t(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)}) \\ &\leq \mathcal{L}(\boldsymbol{w}_r^{(t)}) - \gamma \nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})^\top (\nabla \mathcal{L}_t(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)}) + \frac{\gamma^2 L}{2} \|\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)}\|^2 \end{aligned} \tag{11}$$

The second equality sign is based on the first-order Taylor's theorem with $\frac{\gamma^2}{2}(\nabla \mathcal{L}_t(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)})^\top \nabla^2 \mathcal{L}(\boldsymbol{\xi}^{(t)})(\nabla \mathcal{L}_t(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)})$ as the Lagrange remainder. Given $\boldsymbol{w}_r^{(t)}$, we take the expectation over the randomly selected mini-batch $d^{(t)}$ for both sides and obtain

$$\begin{aligned} \mathrm{E}[\mathcal{L}(\boldsymbol{w}_r^{(t+1)})] &\leq \mathrm{E}[\mathcal{L}(\boldsymbol{w}_r^{(t)}) - \gamma \nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})^\top (\nabla \mathcal{L}_t(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)}) + \frac{\gamma^2 L}{2} \|\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)}\|^2] \\ &\leq \mathrm{E}[\mathcal{L}(\boldsymbol{w}_r^{(t)})] - \gamma \mathrm{E}[\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})^\top \nabla \mathcal{L}_t(\boldsymbol{w}_r^{(t)})] - \gamma \mathrm{E}[\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})^\top \boldsymbol{b}^{(t)}] + \frac{\gamma^2 L}{2} \mathrm{E}[\|\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)}) + \boldsymbol{b}^{(t)}\|^2] \\ &\leq \mathrm{E}[\mathcal{L}(\boldsymbol{w}_r^{(t)})] - \gamma \|\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})\|^2 - \gamma(1 - \gamma L) \mathrm{E}[\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})^\top \boldsymbol{b}^{(t)}] + \frac{\gamma^2 L}{2}(G^2 + \mathrm{E}[\|\boldsymbol{b}^{(t)}\|^2]). \end{aligned} \tag{12}$$

For each iteration t, we have one corresponding inequality as Equation (12). We then sum up the Equation (12) for $t = 0$ to $T - 1$ and have

$$\sum_{t=0}^{T-1} \mathrm{E}[\mathcal{L}(\boldsymbol{w}_r^{(t+1)})] \leq \sum_{t=0}^{T-1} \mathrm{E}[\mathcal{L}(\boldsymbol{w}_r^{(t)})] - \gamma \|\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})\|^2 - \gamma(1 - \gamma L) \mathrm{E}[\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})^\top \boldsymbol{b}^{(t)}] + \frac{\gamma^2 L}{2}(G^2 + \mathrm{E}[\|\boldsymbol{b}^{(t)}\|^2]). \tag{13}$$

After telescoping Equation (13), we have

$$\gamma \sum_{t=0}^{T-1} \|\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})\|^2 + (1 - \gamma L) \mathrm{E}[\nabla \mathcal{L}(\boldsymbol{w}_r^{(t)})^\top \boldsymbol{b}^{(t)}] \leq \mathcal{L}(\boldsymbol{w}_r^{(0)}) - \mathcal{L}(\boldsymbol{w}_r^{(T)}) + \frac{\gamma^2 L}{2} \sum_{t=0}^{T-1} (G^2 + \mathrm{E}[\|\boldsymbol{b}^{(t)}\|^2]). \tag{14}$$

Without loss of generality, we specify the norm as ℓ-2 norm and have

$$\begin{aligned}\|\boldsymbol{b}^{(t)}\| &= \|\nabla\mathcal{L}(\boldsymbol{w}_r^{(t)}, d_r^{(t+1)}) - \nabla\mathcal{L}(\boldsymbol{w}_r^{(t)}, d^{(t+1)})\| \\ &= \frac{1}{m}\left\|\sum_{i=1}^{m}\nabla_{\boldsymbol{w}} l(\boldsymbol{w}_r^{(t)}, d_{r_i}^{(t+1)}) - \nabla_{\boldsymbol{w}} l(\boldsymbol{w}_r^{(t)}, d_i^{(t+1)})\right\| \\ &\leq \frac{1}{m}\sum_{i=1}^{m}\|\nabla_{\boldsymbol{w}} l(\boldsymbol{w}_r^{(t)}, d_{r_i}^{(t+1)}) - \nabla_{\boldsymbol{w}} l(\boldsymbol{w}_r^{(t)}, d_i^{(t+1)})\|.\end{aligned} \tag{15}$$

In particular, we use $d_{r_i}^{(t+1)}$ and $d_i^{(t+1)}$ to denote the i-th data point in the mini-batch $d_r^{(t+1)}$ and $d^{(t+1)}$. We let $d_{r_i}^{(t+1)} = (\boldsymbol{x}_{r_i}^{(t+1)}, y_{r_i}^{(t+1)})$ and $d_i^{(t+1)} = (\boldsymbol{x}_i^{(t+1)}, y_i^{(t+1)})$, denoting the pair of feature vector and label. It is worth noting that $y_{r_i}^{(t+1)} = y_i^{(t+1)}$ according to the definition of $\mathcal{S}_n$. We denote $\boldsymbol{w}[j]$ as the j-th element of $\boldsymbol{w}$. Given $y_{r_i}^{(t+1)} = y_i^{(t+1)}$, we omit the variable y in $l(\cdot)$ and have

$$\begin{aligned}\left|\frac{\partial l(\boldsymbol{w}_r^{(t)}, \boldsymbol{x}_{r_i}^{(t+1)})}{\partial \boldsymbol{w}[j]} - \frac{\partial l(\boldsymbol{w}_r^{(t)}, \boldsymbol{x}_i^{(t+1)})}{\partial \boldsymbol{w}[j]}\right| &= \left|\int_{s=0}^{1}(\boldsymbol{x}_{r_i}^{(t+1)} - \boldsymbol{x}_i^{(t+1)})^\top \frac{\partial^2 l(\boldsymbol{w}_r^{(t)}, \boldsymbol{x}_i^{(t+1)} + s(\boldsymbol{x}_{r_i}^{(t+1)} - \boldsymbol{x}_i^{(t+1)}))}{\partial \boldsymbol{w}[j]\partial \boldsymbol{x}} ds\right| \\ &\leq \int_{s=0}^{1}\left|(\boldsymbol{x}_{r_i}^{(t+1)} - \boldsymbol{x}_i^{(t+1)})^\top \frac{\partial^2 l(\boldsymbol{w}_r^{(t)}, \boldsymbol{x}_i^{(t+1)} + s(\boldsymbol{x}_{r_i}^{(t+1)} - \boldsymbol{x}_i^{(t+1)}))}{\partial \boldsymbol{w}[j]\partial \boldsymbol{x}}\right| ds \\ &\leq \int_{s=0}^{1}\|\boldsymbol{x}_{r_i}^{(t+1)} - \boldsymbol{x}_i^{(t+1)}\| \cdot \left\|\frac{\partial^2 l(\boldsymbol{w}_r^{(t)}, \boldsymbol{x}_i^{(t+1)} + s(\boldsymbol{x}_{r_i}^{(t+1)} - \boldsymbol{x}_i^{(t+1)}))}{\partial \boldsymbol{w}[j]\partial \boldsymbol{x}}\right\| ds \\ &\leq \int_{s=0}^{1}\|\boldsymbol{x}_{r_i}^{(t+1)} - \boldsymbol{x}_i^{(t+1)}\| \cdot \left\|\nabla_{\boldsymbol{x}}\frac{\partial l(\boldsymbol{w}, \boldsymbol{x})}{\partial \boldsymbol{w}[j]}\right\| ds \\ &= \|\boldsymbol{x}_{r_i}^{(t+1)} - \boldsymbol{x}_i^{(t+1)}\| \cdot \left\|\nabla_{\boldsymbol{x}}\frac{\partial l(\boldsymbol{w}, \boldsymbol{x})}{\partial \boldsymbol{w}[j]}\right\|.\end{aligned} \tag{16}$$

We then incorporate Equation (16) into Equation (15) and obtain that

$$\begin{aligned}\|\boldsymbol{b}^{(t)}\| &\leq \frac{1}{m}\sum_{i=1}^{m}\|\nabla_{\boldsymbol{w}} l(\boldsymbol{w}_r^{(t)}, d_{r_i}^{(t+1)}) - \nabla_{\boldsymbol{w}} l(\boldsymbol{w}_r^{(t)}, d_i^{(t+1)})\| \\ &\leq \frac{1}{m}\sum_{i=1}^{m} L_x\|\boldsymbol{x}_{r_i}^{(t+1)} - \boldsymbol{x}_i^{(t+1)}\| \\ &\leq L_x C_D = B.\end{aligned} \tag{17}$$

In addition, the unlearned data cannot be involved in every single iteration when $|\mathcal{D}_u| < T$. When $d^{(t+1)} \cap \mathcal{D}_u = \emptyset$, we have $d^{(t+1)} = d_r^{(t+1)}$ and $\boldsymbol{b}^{(t)} = 0$. Considering that $d^{(t+1)}$ is chosen uniformly from $\mathcal{D}$, we can compute the probability of $d^{(t+1)} \cap \mathcal{D}_u \neq \emptyset$ as $p_u = 1 - (1 - |\mathcal{D}_u|/|\mathcal{D}|)^m$. Consequently, when $T \to \infty$, we can consider that only $p_u T$ iterations contain the bias term $\boldsymbol{b}^{(t)}$ and we have $\boldsymbol{b}^{(t)} = 0$ for the remaining $(1 - p_u)T$ iterations. We then incorporate Equation (17) back into Equation (14) and have

$$\begin{aligned}\gamma\sum_{t=0}^{T-1}\|\nabla\mathcal{L}(\boldsymbol{w}_r^{(t)})\|^2 - \gamma(1-\gamma L)p_u TGB &\leq \gamma\sum_{t=0}^{T-1}\|\nabla\mathcal{L}(\boldsymbol{w}_r^{(t)})\|^2 + (1-\gamma L)\mathrm{E}[\nabla\mathcal{L}(\boldsymbol{w}_r^{(t)})^\top \boldsymbol{b}^{(t)}] \\ &\leq \mathcal{L}(\boldsymbol{w}_r^{(0)}) - \mathcal{L}(\boldsymbol{w}_r^{(T)}) + \frac{\gamma^2 L}{2}\sum_{t=0}^{T-1}(G^2 + \mathrm{E}[\|\boldsymbol{b}^{(t)}\|^2]) \\ &\leq \mathcal{L}(\boldsymbol{w}_r^{(0)}) - \mathcal{L}(\boldsymbol{w}_r^{(T)}) + \frac{\gamma^2 LT}{2}(G^2 + p_u B^2).\end{aligned} \tag{18}$$

Finally, we can finish the proof by

$$\begin{aligned}\mathrm{E}_T[\|\nabla\mathcal{L}(\boldsymbol{w}_r^{(T)})\|^2] &= \frac{1}{T}\sum_{t=0}^{T-1}\|\nabla\mathcal{L}(\boldsymbol{w}_r^{(t)})\|^2 \\ &\leq \frac{\mathcal{L}(\boldsymbol{w}_r^{(0)})-\mathcal{L}(\boldsymbol{w}_r^{(T)})}{\gamma T}+\frac{\gamma L}{2}(G^2+p_uB^2)+(1-\gamma L)p_uGB.\end{aligned} \tag{19}$$

□

A.3. Proof of Proposition 4.4

Proof. To prove that Algorithm 2 returns a valid PoRT, we verify the two properties of a valid PoRT in Definition 4.1. First, we verify the removable property. For the t-th iteration, if $t \in \mathcal{I}_e$, we have that $d_r^{(t)} \cap \mathcal{D}_u = d^{(t)} \cap \mathcal{D}_u = \emptyset$ holds; if $t \in \mathcal{I}_e$, we have that

$$\begin{aligned}d_r^{(t)} \cap \mathcal{D}_u &= \mathcal{S}_n(d^{(t)}) \cap \mathcal{D}_u \\ &= \left((d^{(t)} \backslash \mathcal{D}_u) \cap \mathcal{D}_u\right) \cup \left(\{\mathcal{N}(\boldsymbol{x},y) | (\boldsymbol{x},y) \in d^{(t)} \cap \mathcal{D}_u\} \cap \mathcal{D}_u\right) \\ &= \{\mathrm{argmin}_{(\boldsymbol{x}',y')\in\mathcal{D}\backslash\mathcal{D}_u, y'=y}\|\boldsymbol{x}'-\boldsymbol{x}\| | (\boldsymbol{x},y) \in d^{(t)} \cap \mathcal{D}_u\} \cap \mathcal{D}_u \\ &= \emptyset.\end{aligned} \tag{20}$$

The last equality sign holds because each element $\mathrm{argmin}_{(\boldsymbol{x}',y')\in\mathcal{D}\backslash\mathcal{D}_u,y'=y}\|\boldsymbol{x}'-\boldsymbol{x}\|$ is involved in $\mathcal{D}\backslash\mathcal{D}_u$. Hence, we have verified the removable property of Algorithm 2. Next, we verify the reproducible property. For any $t \in \mathcal{I}$, we have $t \in \mathcal{I}_e$ or $t \in \mathcal{I}_n$. We then discuss these two conditions separately. For simplicity, we denote that $\boldsymbol{p}^{(t)} = \boldsymbol{w}_r^{(t)} - \boldsymbol{w}^{(t)}$.

(1). $t \in \mathcal{I}_e$: we have $d_r^{(t)} = d^{(t)}$.

$$\begin{aligned}&\boldsymbol{w}_r^{(t)} - g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)}) \\ =&\boldsymbol{w}^{(t)} + \boldsymbol{p}^{(t)} - \boldsymbol{w}_r^{(t-1)} + \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)}) \\ =&\boldsymbol{w}^{(t)} + \boldsymbol{p}^{(t)} - \boldsymbol{w}^{(t-1)} - \boldsymbol{p}^{(t-1)} + \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)}) \\ =&\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)} + \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d^{(t)}) - \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d^{(t)}).\end{aligned} \tag{21}$$

We then focus on the norm value and have

$$\begin{aligned}&\|\boldsymbol{w}_r^{(t)} - g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)})\| \\ =&\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)} + \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d^{(t)}) - \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d^{(t)})\| \\ \leq&\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)}\| + \gamma^{(t)}L\|\boldsymbol{w}_r^{(t-1)} - \boldsymbol{w}^{(t-1)}\| \\ =&\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)}\| + \gamma^{(t)}L\|\boldsymbol{p}^{(t-1)}\|.\end{aligned} \tag{22}$$

(2). $t \in \mathcal{I}_n$: we have $d_r^{(t)} = \mathcal{S}_n(d^{(t)})$.

$$\begin{aligned}&\boldsymbol{w}_r^{(t)} - g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)}) \\ =&\boldsymbol{w}^{(t)} + \boldsymbol{p}^{(t)} - \boldsymbol{w}_r^{(t-1)} + \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)}) \\ =&\boldsymbol{w}^{(t-1)} - \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d^{(t)}) + \boldsymbol{p}^{(t)} - \boldsymbol{w}_r^{(t-1)} + \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)}) \\ =&\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)} + \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d_r^{(t)}) - \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d^{(t)}) + \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)}) - \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d_r^{(t)}).\end{aligned} \tag{23}$$

We then focus on the norm value and have

$$\begin{aligned}&\|\boldsymbol{w}_r^{(t)} - g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)})\| \\ \leq&\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)}\| + \|\gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d_r^{(t)}) - \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d^{(t)})\| + \|\gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)}) - \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d_r^{(t)})\| \\ \leq&\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)}\| + \gamma^{(t)}\left(\|\boldsymbol{b}^{(t-1)}\| + L\|\boldsymbol{p}^{(t-1)}\|\right),\end{aligned} \tag{24}$$

where $\boldsymbol{b}^{(t-1)} = \nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d_r^{(t)}) - \nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d^{(t)})$. Next, we find the upper bound of $\boldsymbol{p}^{(t)}$. When $t \in \mathcal{I}_e$, we have $\|\boldsymbol{p}^{(t)}\| = \gamma_r^{(t)}\|\nabla l(f_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(t)}), y^{(t)})\|$; when $t \in \mathcal{I}_n$, we have

$$\|\boldsymbol{p}^{(t)}\| = \|\boldsymbol{w}^{(t-1)} - \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d^{(t)}) - \boldsymbol{w}^{(t-1)} + \gamma^{(t)}\nabla\mathcal{L}(\boldsymbol{w}^{(t-1)}, d_r^{(t)})\| = \gamma^{(t)}\|\boldsymbol{b}^{(t-1)}\|. \tag{25}$$

In addition, we know from Equation (17) that $\|\boldsymbol{b}^{(t-1)}\| \leq L_x C_D$. Combine the case $t \in \mathcal{I}_e$ and the case $t \in \mathcal{I}_n$, we have $\|\boldsymbol{p}^{(t)}\| \leq \max\{\gamma_r^{(t)}\|\nabla l(f_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(t)}), y^{(t)})\|, \gamma^{(t)} L_x C_D\} = P$. Finally, we verify the reproducible property by proving $\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)}\| + \gamma^{(t)} L\|\boldsymbol{p}^{(t-1)}\| \leq \varepsilon$ and $\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)}\| + \gamma^{(t)}\left(\|\boldsymbol{b}^{(t-1)}\| + L\|\boldsymbol{p}^{(t-1)}\|\right) \leq \varepsilon$. Noting that $\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)}\| + \gamma^{(t)}\left(\|\boldsymbol{b}^{(t-1)}\| + L\|\boldsymbol{p}^{(t-1)}\|\right) \geq \|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)}\| + \gamma^{(t)} L\|\boldsymbol{p}^{(t-1)}\|$, we can only verify the second inequality.

(i). If $\gamma_r^{(t)}\|\nabla l(f_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(t)}), y^{(t)})\| \leq \gamma^{(t)} L_x C_D$, we have

$$\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)}\| + \gamma^{(t)}\left(\|\boldsymbol{b}^{(t-1)}\| + L\|\boldsymbol{p}^{(t-1)}\|\right) \leq 3\gamma^{(t)} L_x C_D + \left(\gamma^{(t)}\right)^2 L L_x C_D \leq \varepsilon. \tag{26}$$

By solving this quadratic inequality, we obtain that when $\gamma^{(t)} \leq \frac{1}{2L}\left(\sqrt{9 + \frac{4\varepsilon L}{L_x C_D}} - 3\right)$, Equation (26) holds.

(ii). If $\gamma_r^{(t)}\|\nabla l(f_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(t)}), y^{(t)})\| \geq \gamma^{(t)} L_x C_D$, we have

$$\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(t-1)}\| + \gamma^{(t)}\left(\|\boldsymbol{b}^{(t-1)}\| + L\|\boldsymbol{p}^{(t-1)}\|\right) \leq \gamma^{(t)} L_x C_D + \gamma_r^{(t)}\|\nabla l(f_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(t)}), y^{(t)})\|(2 + \gamma^{(t)} L) \leq \varepsilon. \tag{27}$$

Consequently, we obtain that when $\gamma_r^{(t)} \leq \frac{\varepsilon - \gamma^{(t)} L_x C_D}{\|\nabla l(f_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(t)}), y^{(t)})\|(2+\gamma^{(t)} L)}$, Equation (27) holds. In order to avoid no solution for $\gamma_r^{(t)}$, we also need

$$0 \leq \frac{\gamma^{(t)} L_x C_D}{\|\nabla l(f_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(t)}), y^{(t)})\|(2 + \gamma^{(t)} L)} \leq \gamma_r^{(t)} \leq \frac{\varepsilon - \gamma^{(t)} L_x C_D}{\|\nabla l(f_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(t)}), y^{(t)})\|(2 + \gamma^{(t)} L)}. \tag{28}$$

By solving this inequality for $\gamma^{(t)}$, we obtain $0 \leq \gamma^{(t)} \leq \frac{1}{2L}\left(\sqrt{9 + \frac{4\varepsilon L}{L_x C_D}} - 3\right)$ again. Combining conclusions from above cases and the assumption that $\|\nabla_{\boldsymbol{w}} l(\boldsymbol{w}, \boldsymbol{x})\| \leq G$, we have obtained that when $0 \leq \gamma^{(t)} \leq \frac{1}{2L}\left(\sqrt{9 + \frac{4\varepsilon L}{L_x C_D}} - 3\right)$ and $0 \leq \gamma_r^{(t)} \leq \frac{\varepsilon - \gamma^{(t)} L_x C_D}{G(2+\gamma^{(t)} L)}$, the reproducible property holds, i.e., $\|\boldsymbol{w}_r^{(t)} - g_r^{(t)}(\boldsymbol{w}_r^{(t-1)}, d_r^{(t)})\| \leq \varepsilon$. In conclusion, by incorporating the reproducible property and the removable property, our proof is finished. □

A.4. Proof of Proposition 4.5

Proof. We assume the time complexity of naive retraining is $T(n)$. For Algorithm 1, the only difference with naive retraining is the selection of mini-batches. At best, we can compute the nearest neighbor of each data sample beforehand and obtain a $T(n)$ complexity. For Algorithm 2, every iteration in the PoRT can be computed separately in parallel. In particular, we have $T(n) = n \cdot m \cdot T(1)$, where n is the number of iterations, m is the batch size, and $T(1)$ denotes the time for computing the gradient of one data sample. If $t \in \mathcal{I}_e$, computing $\boldsymbol{w}_r^{(t)}$ requires a complexity of $T(1)$; if $t \in \mathcal{I}_n$, computing $\boldsymbol{w}_r^{(t)}$ requires a complexity of $m \cdot T(1)$. According to the analysis in Appendix A.2, we have that when $n \to \infty$, the number of iterations in $\mathcal{I}_n$ will approach $p_u n$ and the number of iterations in $\mathcal{I}_e$ will approach $(1 - p_u)n$. Finally, with the parallel processes, we have the overall complexity is

$$m \cdot T(1) \cdot \frac{p_u n}{P} + T(1) \cdot \frac{(1 - p_u)n}{P} = \left(\frac{p_u}{P} + \frac{1 - p_u}{mP}\right) T(n). \tag{29}$$

□

B. Experimental Setups and Additional Experimental Results

We implemented all experiments in the PyTorch (Paszke et al., 2019) library and exploited SGD as the optimizer for training. For consistent hyperparameter settings across all datasets, we fix the learning rate $\gamma^{(t)}$ as 10^{-2}, the weight decay parameter

Table 4. **Model Utility in Normal Settings**. We assess the model utility among original training, naive retraining, and adversarial unlearning methods over three popular DNNs across three real-world datasets. We record the macro F1-score of the predictions on the unlearned set $\mathcal{D}_u$, retained set $\mathcal{D}\backslash\mathcal{D}_u$, and test set $\mathcal{D}_t$.

Model	Method	MNIST			CIFAR-10			SVHN		
		$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$	$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$	$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$
MLP	Original	99.47 ± 0.09	99.76 ± 0.08	97.00 ± 0.17	80.96 ± 0.15	80.75 ± 0.96	50.54 ± 0.57	89.56 ± 0.78	89.40 ± 0.45	80.03 ± 0.45
	Retrain	96.43 ± 0.19	99.52 ± 0.10	96.75 ± 0.13	49.32 ± 0.59	82.96 ± 1.29	49.29 ± 0.77	82.42 ± 0.22	89.83 ± 0.39	79.50 ± 0.37
	Adv-R ($\mathcal{S}_r$)	96.36 ± 0.13	99.48 ± 0.17	96.58 ± 0.12	48.49 ± 0.41	82.33 ± 0.40	48.27 ± 0.48	81.85 ± 0.85	89.31 ± 0.86	78.63 ± 0.71
	Adv-R ($\mathcal{S}_n$)	96.34 ± 0.11	98.65 ± 0.19	96.60 ± 0.14	50.43 ± 0.17	80.01 ± 1.30	49.86 ± 0.23	82.98 ± 0.57	92.60 ± 0.56	79.77 ± 0.57
	Adv-F	99.30 ± 0.13	99.33 ± 0.10	96.94 ± 0.14	81.13 ± 0.76	81.38 ± 0.79	50.81 ± 0.63	89.99 ± 0.71	89.77 ± 0.55	80.11 ± 0.85
CNN	Original	99.69 ± 0.31	99.74 ± 0.30	99.13 ± 0.25	100.00 ± 0.00	100.00 ± 0.00	85.33 ± 0.31	99.64 ± 0.73	99.61 ± 0.77	94.66 ± 1.00
	Retrain	99.31 ± 0.12	100.00 ± 0.00	99.39 ± 0.05	83.60 ± 0.31	100.00 ± 0.00	83.12 ± 0.23	94.24 ± 0.22	100.00 ± 0.00	94.96 ± 0.10
	Adv-R ($\mathcal{S}_r$)	99.24 ± 0.13	100.00 ± 0.00	99.27 ± 0.02	83.81 ± 0.44	100.00 ± 0.00	83.08 ± 0.34	94.29 ± 0.17	100.00 ± 0.00	94.90 ± 0.07
	Adv-R ($\mathcal{S}_n$)	99.35 ± 0.09	100.00 ± 0.00	99.40 ± 0.02	83.12 ± 0.31	100.00 ± 0.00	82.71 ± 0.33	94.36 ± 0.32	100.00 ± 0.00	94.92 ± 0.15
	Adv-F	100.00 ± 0.00	100.00 ± 0.00	99.46 ± 0.08	100.00 ± 0.00	100.00 ± 0.00	85.20 ± 0.24	100.00 ± 0.00	99.99 ± 0.01	95.13 ± 0.10
ResNet	Original	100.00 ± 0.00	100.00 ± 0.00	99.53 ± 0.05	99.99 ± 0.03	99.99 ± 0.03	84.11 ± 0.74	100.00 ± 0.00	100.00 ± 0.00	94.91 ± 0.09
	Retrain	99.41 ± 0.09	100.00 ± 0.00	99.46 ± 0.07	82.76 ± 0.38	100.00 ± 0.00	82.10 ± 0.39	94.33 ± 0.24	100.00 ± 0.00	94.57 ± 0.06
	Adv-R ($\mathcal{S}_r$)	99.43 ± 0.09	100.00 ± 0.00	99.46 ± 0.04	82.29 ± 0.98	99.99 ± 0.01	81.71 ± 0.78	94.38 ± 0.11	100.00 ± 0.00	94.54 ± 0.09
	Adv-R ($\mathcal{S}_n$)	99.41 ± 0.06	100.00 ± 0.00	99.42 ± 0.04	82.40 ± 0.39	100.00 ± 0.00	81.85 ± 0.44	94.64 ± 0.20	100.00 ± 0.00	94.75 ± 0.04
	Adv-F	100.00 ± 0.00	100.00 ± 0.00	99.49 ± 0.03	100.00 ± 0.00	100.00 ± 0.00	84.54 ± 0.37	100.00 ± 0.00	100.00 ± 0.00	94.91 ± 0.07

Table 5. **Model Utility in Class-Imbalanced Settings**. We assess the model utility among original training, naive retraining, and adversarial unlearning methods over three popular DNNs across three real-world datasets. We record the macro F1-score of the predictions on the unlearned set $\mathcal{D}_u$, retained set $\mathcal{D}\backslash\mathcal{D}_u$, and test set $\mathcal{D}_t$.

Model	Method	MNIST			CIFAR-10			SVHN		
		$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$	$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$	$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$
MLP	Original	60.29 ± 13.07	97.00 ± 2.60	96.88 ± 0.07	34.50 ± 6.41	75.68 ± 2.29	50.95 ± 0.19	39.13 ± 7.26	84.02 ± 3.90	80.50 ± 0.83
	Retrain	38.76 ± 13.41	95.86 ± 4.34	89.92 ± 5.70	13.29 ± 4.49	77.93 ± 2.83	44.81 ± 1.70	22.65 ± 8.11	76.87 ± 3.93	67.33 ± 4.90
	Adv-R ($\mathcal{S}_r$)	39.48 ± 12.20	99.71 ± 0.19	91.04 ± 4.80	13.13 ± 4.32	81.21 ± 3.21	44.19 ± 1.81	26.33 ± 8.99	87.25 ± 2.74	72.09 ± 4.50
	Adv-R ($\mathcal{S}_n$)	42.90 ± 11.87	97.96 ± 0.59	92.80 ± 4.54	16.17 ± 4.76	67.25 ± 2.15	45.74 ± 1.80	28.06 ± 9.45	82.16 ± 1.65	70.67 ± 3.55
	Adv-F	64.21 ± 9.89	97.28 ± 3.34	96.81 ± 0.04	35.46 ± 6.74	75.08 ± 1.68	51.06 ± 0.86	40.10 ± 7.87	82.78 ± 4.54	79.70 ± 1.08
CNN	Original	100.00 ± 0.00	100.00 ± 0.00	99.48 ± 0.06	100.00 ± 0.00	100.00 ± 0.00	85.44 ± 0.22	100.00 ± 0.00	100.00 ± 0.00	95.13 ± 0.07
	Retrain	47.64 ± 15.76	100.00 ± 0.00	94.98 ± 5.35	24.25 ± 6.98	90.88 ± 5.03	65.22 ± 5.94	32.79 ± 12.61	92.68 ± 4.79	84.23 ± 6.65
	Adv-R ($\mathcal{S}_r$)	46.00 ± 12.84	100.00 ± 0.00	94.94 ± 4.77	25.94 ± 6.89	96.00 ± 4.90	76.51 ± 4.32	33.43 ± 12.12	97.52 ± 3.87	86.69 ± 5.40
	Adv-R ($\mathcal{S}_n$)	49.90 ± 15.03	99.96 ± 0.07	95.08 ± 4.82	23.97 ± 8.75	91.46 ± 1.81	67.34 ± 4.02	33.89 ± 12.26	98.49 ± 0.40	85.76 ± 6.46
	Adv-F	92.49 ± 15.01	99.97 ± 0.05	99.45 ± 0.12	100.00 ± 0.00	100.00 ± 0.00	85.11 ± 0.21	100.00 ± 0.00	100.00 ± 0.00	95.14 ± 0.05
ResNet	Original	100.00 ± 0.00	100.00 ± 0.00	99.54 ± 0.05	98.30 ± 3.39	99.96 ± 0.08	85.02 ± 0.96	100.00 ± 0.00	100.00 ± 0.00	94.66 ± 0.30
	Retrain	50.74 ± 14.51	99.00 ± 2.00	95.89 ± 5.40	21.80 ± 6.43	89.13 ± 2.84	66.86 ± 3.29	33.08 ± 11.97	95.19 ± 3.78	83.89 ± 5.83
	Adv-R ($\mathcal{S}_r$)	51.76 ± 14.11	100.00 ± 0.00	96.33 ± 4.73	25.53 ± 6.93	100.00 ± 0.00	74.18 ± 4.26	34.76 ± 12.06	98.00 ± 4.01	87.63 ± 6.11
	Adv-R ($\mathcal{S}_n$)	50.61 ± 11.18	100.00 ± 0.00	96.54 ± 4.27	23.83 ± 6.84	96.53 ± 1.05	68.86 ± 4.00	33.92 ± 11.85	99.37 ± 0.20	84.87 ± 5.42
	Adv-F	100.00 ± 0.00	100.00 ± 0.00	99.54 ± 0.03	97.13 ± 5.74	99.98 ± 0.03	84.04 ± 0.46	100.00 ± 0.00	100.00 ± 0.00	94.77 ± 0.09

as 5×10^{-4}, the training epochs number as 30, and set the batch size to 128. In determining the selection $\mathcal{S}_r$, specifically for data batches containing unlearned data, we randomly sample 50 data batches from the remaining set $\mathcal{D}\backslash\mathcal{D}_u$ and select the batch that yields the smallest distance, as defined in Equation (3). All experiments were conducted on an Nvidia RTX A6000 GPU. We reported the average value and the standard deviation of the numerical results under five different random seeds. For the experiment of verification errors in Figure 4, we set the learning rate $\gamma^{(t)}$ as 5×10^{-3}, weight decay parameter as 0. We fix the size of the unlearned set as $1,000$ and set the learning rate $\gamma_r^{(t)}$ as 10^{-3}.

B.1. Model Utility Study

B.1.1. Experimental Settings

All datasets utilized in our experiments adhere to the standard train/test split provided by the Torchvision library (maintainers & contributors, 2016). Within each experiment, 20% of the training data is set aside as the validation set. To assess model performance, we randomly sample 10% of the remaining training data to form the unlearn set. We report the average macro F1-score, along with its standard deviation, based on model predictions for the unlearn set $\mathcal{D}_u$, the remaining set $\mathcal{D}\backslash\mathcal{D}_u$, and the test set $\mathcal{D}_t$. These metrics are computed across five random seeds to ensure robustness.

Additionally, to simulate scenarios where the data distribution of the unlearn set deviates from the overall training set, we introduce a class-imbalanced unlearning setting under a fixed train/val/test split. For each class c, we follow the approach of Yurochkin et al. by drawing a 5-dimensional vector q_c from a Dirichlet distribution with its parameter of 0.5. We then

Table 6. Comparison of the model utility among original training, naive retraining, and adversarial unlearning methods based on ResNet-50 over the Tiny-ImageNet dataset. We record the macro F1-score of the predictions on the unlearned set $\mathcal{D}_u$, retained set $\mathcal{D}\backslash\mathcal{D}_u$, and test set $\mathcal{D}_t$.

Method	**Normal**			**Imbalanced**		
	$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$	$\mathcal{D}_u$	$\mathcal{D}\backslash\mathcal{D}_u$	$\mathcal{D}_t$
Original	90.34 ± 0.47	90.21 ± 0.43	36.59 ± 0.74	91.76 ± 0.64	91.42 ± 0.59	33.81 ± 0.60
Retrain	31.03 ± 0.62	93.78 ± 0.57	31.08 ± 0.33	8.59 ± 1.94	95.79 ± 18.38	26.19 ± 1.91
Adv-R ($\mathcal{S}_r$)	31.26 ± 0.46	92.83 ± 0.63	31.78 ± 0.16	8.98 ± 2.95	95.65 ± 18.65	26.76 ± 1.17
Adv-R ($\mathcal{S}_n$)	31.82 ± 0.26	93.09 ± 0.42	31.76 ± 0.43	8.70 ± 0.96	92.83 ± 5.36	26.80 ± 1.20
Adv-F	90.16 ± 0.66	90.30 ± 0.51	36.57 ± 0.80	91.37 ± 0.56	91.10 ± 0.56	33.69 ± 0.81

assign data to the i-th piece proportionally to $q_c[i]$. In each experiment, one piece of data is selected as the unlearn set $\mathcal{D}_u$, while the remaining pieces constitute the remaining set $\mathcal{D}\backslash\mathcal{D}_u$. The average model performance is recorded across five experiments.

B.1.2. Additional Experimental Results

We provide the complete experimental results of the model utility in different types of unlearning strategies under normal and class-imbalanced settings in Table 4 and Table 5, respectively. From the full version of the results, we can obtain that:

1. The naive retraining can render a larger utility drop when the model under-fits the data and the data heterogeneity exists in the unlearning process;

2. In the normal setting, the retraining-based adversarial method has similar utility to the naive retraining, while in the class-imbalanced setting, the retraining-based adversarial method achieves a much better utility than naive retraining, which means model providers can benefit more when data heterogeneity exists in the unlearning process;

3. The forging-based adversarial method has much better performance than other unlearning methods in all cases, even better than the original training in some cases, which means the model provider can benefit the most from forging (but also with a larger probability of being detected by backdoor verification).

To further show the scalability of our proposed methods, we conduct additional experiments using the ResNet-50 model (He et al., 2016) over the Tiny-ImageNet dataset (Le & Yang, 2015). The TinyImageNet dataset (Le & Yang, 2015) is a subset of the ImageNet dataset. It consists of 200 object classes, and for each object class, it provides 500 training images, 50 validation images, and 50 test images. All images have been downsampled to $64 \times 64 \times 3$ pixels. As the test set is released without labels, we use the validation set as the test set in our experiment. Within each experiment, 20% of the training data is set aside as the validation set, and the division of the unlearn set and other parameter settings are consistent with the main experiments in the paper. We train the ResNet-50 model from scratch for 50 epochs following the experimental setting in (Yuan et al., 2020) and record the accuracy under 5 random seeds. For S_r, we randomly sample 10 mini-batches from the retained set $D\backslash D_u$ and select the batch that yields the smallest distance, as defined in Equation (3). The experimental results of the utility of the unlearned model are shown in Table 6. Basically, the experimental results are consistent with Table 3 in our paper (though overfitting exists), i.e., Adv-F achieves comparable performance as the original model and Adv-R has a better performance compared with naive retraining. We also find that the difference between the normal setting and the imbalanced setting is not as distinct as in Table 3. We explain this as

1. Tiny-ImageNet has 200 classes while the datasets in Table 3 only have 10 classes. The impact of imbalanced sampling can be weakened under far more classes.

2. Tiny-ImageNet has only 400 training samples per class while the datasets in Table 3 have over 5,000 samples. The effect of nearest neighbor selection can be diminished as the constant C_D (introduced in Proposition 4.3) might increase and subsequently, the gap between the adversarially unlearned model and the original model can be larger according to Proposition 4.3.

Table 7. Results of backdoor verification on different (adversarial) unlearning strategies over three datasets.

Method	MLP & MNIST			CNN & CIFAR-10			ResNet & SVHN		
	in atk acc p	ex atk acc q	β	in atk acc p	ex atk acc q	β	in atk acc p	ex atk acc q	β
Original	0.998 ± 0.007	0.101 ± 0.010	2.61×10^{-42}	0.933 ± 0.105	0.088 ± 0.139	5.14×10^{-20}	0.982 ± 0.003	0.095 ± 0.001	1.11×10^{-28}
Retrain	0.102 ± 0.016	0.103 ± 0.012	0.998	0.118 ± 0.022	0.124 ± 0.010	0.998	0.110 ± 0.001	0.096 ± 0.001	0.999
Adv-R	0.103 ± 0.017	0.102 ± 0.015	0.997	0.129 ± 0.021	0.102 ± 0.009	0.997	0.109 ± 0.001	0.096 ± 0.001	0.999
Adv-F	0.995 ± 0.003	0.103 ± 0.013	5.46×10^{-34}	0.914 ± 0.119	0.100 ± 0.050	2.78×10^{-16}	0.981 ± 0.006	0.096 ± 0.001	2.08×10^{-26}

B.2. Backdoor Verification

For backdoor verification, we exploit Athena (Sommer et al., 2022) as the verification strategy. Specifically, we consider two hypotheses: H_0 - *the data has been unlearned*, and H_1 - *the data has not been unlearned*. In assessing the effectiveness of a backdoor verification strategy applied to an algorithm $\mathcal{A}$, we focus on the deletion confidence for a given acceptable tolerance of Type I error α ($\alpha = \Pr[\text{Reject } H_0 | H_0 \text{ is true}]$), i.e.,

$$\rho_{\mathcal{A},\alpha}(n) = (1-\beta) = 1 - \Pr[\text{Accept } H_0 | H_1 \text{ is true}]. \tag{30}$$

We follow Sommer et al. to compute the Type II error β as a function of α and the number of testing samples n, i.e.,

$$(1-\beta) = 1 - \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} \cdot I[\sum_{l=0}^{k} \binom{n}{l} q^l (1-q)^{n-l} \leq 1-\alpha], \tag{31}$$

where q and p represent the backdoor attack accuracy for deleted and undeleted data, respectively. The function $I(x) = 1$ if x is True and 0 otherwise. We apply similar method as in Sommer et al. to estimate q and p:

- **Estimating** p. To estimate p, we first introduce a specific backdoor pattern to 10% of the unlearn set. This involves randomly selecting four pixels in each sample, setting their values to 1, and assigning a target label c_b. The model is then trained on this partially backdoored dataset $\mathcal{D}^b_{train}$. For evaluation, we extract 2% of the test samples $\mathcal{D}^b_{test}$, apply the same backdoor pattern, and calculate the backdoor success rate for this trigger with the target label c_b as,

$$p = \frac{1}{|\mathcal{D}^b_{test}|} |\{(x,y) \in \mathcal{D}^b_{test} | \mathcal{A}(\mathcal{D}^b_{train})(x) = c_b\}| \tag{32}$$

- **Estimating** q. To estimate q, we randomly select an additional 2% of the test samples, denoted as $\mathcal{D}^{ex}_{test}$. On these samples, a different backdoor pattern is applied. We then calculate the backdoor success rate, focusing on the trigger with an alternate target label c_{ex}. This process enables us to determine q as,

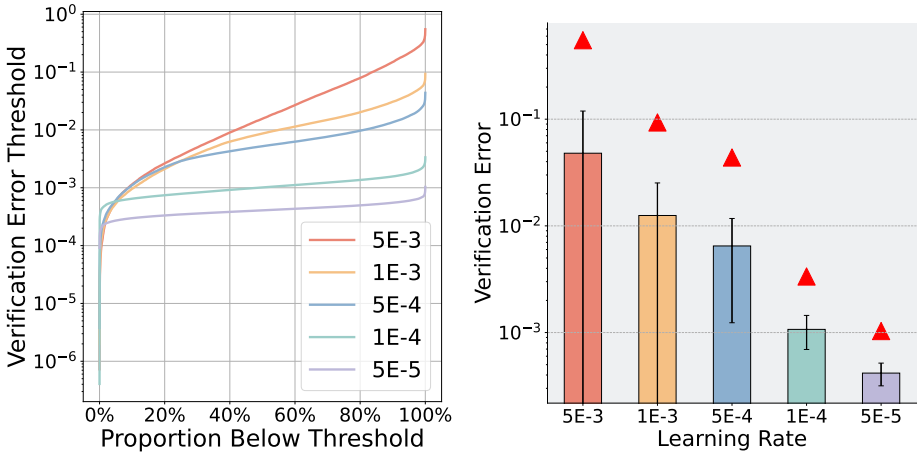
$$q = \frac{1}{|\mathcal{D}^{ex}_{test}|} |\{(x,y) \in \mathcal{D}^{ex}_{test} | \mathcal{A}(\mathcal{D}^b_{train})(x) = c_{ex}\}| \tag{33}$$

We set α to 10^{-3}, n to 30, and use the estimate p and q to compute β in Equation (31). For the MNIST and CIFAR-10 datasets, the target labels c_b and c_{ex} are selected randomly, owing to the even distribution of their test data across classes. In contrast, for the SVHN dataset, we specifically choose $c_b = 3$ and $c_{ex} = 4$ for a better explanation due to its uneven data distribution. Notably, in the SVHN dataset, data labeled as 3 or 4 accounts for nearly one-tenth of the test data, which is significant given that the dataset comprises ten classes. We provide the complete experimental results of the backdoor verification in Table 7.

We first clarify that if p or q approaches the level of random prediction, it suggests the corresponding backdoored data has not been utilized during the training process. Conversely, a high value of p indicates the effectiveness of the backdoor strategy. In other words, a large p value signifies that the backdoor attack was successful in influencing the model's behavior. The results in Table 7 reveal that the backdoor verification mechanism is highly effective for models trained on $\mathcal{D}^b_{train}$. This effectiveness is primarily attributed to the substantial difference between p and q. Additionally, the verification mechanism is capable of detecting models modified using the forging-based method with a high degree of probability. This is due to the fact that such modifications only slightly alter the original model. In contrast, the retraining-based model can deception backdoor verification, as it does not directly utilize the partially backdoored unlearning data, resulting in a model that is not affected by poisoned data.

Table 8. Results of the membership inference attack on different (adversarial) unlearning strategies over three datasets.

Method	MNIST	CIFAR-10	SVHN
Original	50.60 ± 0.61	72.59 ± 1.03	59.67 ± 0.31
Retrain	50.16 ± 0.92	48.97 ± 0.65	51.88 ± 1.39
Adv-F	50.09 ± 0.34	72.41 ± 0.55	59.04 ± 1.10
Adv-R	49.66 ± 0.85	48.93 ± 1.22	50.09 ± 0.65



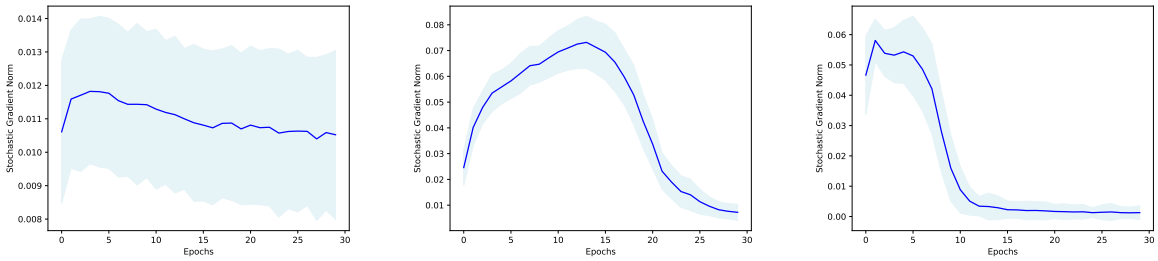
(a) Verification error distribution

(b) Verification error statistics

Figure 7. Comparison of verification error under different learning rate configurations.

B.3. Membership Inference Attack

Membership Inference Attack (MIA) (Shokri et al., 2017) is seen as an effective evaluation method of machine unlearning by measuring the privacy leakage of the data supposedly unlearned. Different from the reproducing verification and the backdoor verification, we tend to categorize MIA into the evaluation method rather than the verification method. To clarify this, we first aim to distinguish two different settings for measuring the unlearning efficacy. We can refer to them as evaluation and verification (which might be mixed up in previous works). Evaluation is supposed to be conducted by honest model providers to choose the best unlearning methods from a candidate set of unlearning algorithms. In contrast, verification is supposed to be conducted by data owners or third parties to check the unlearning efficacy. The most distinct difference between evaluation and verification is the capacity of the evaluator (or verifier), where the evaluator (model provider) has full access to the original model, unlearned model, and the dataset, while the verifier (data owner) only has limited access to the models and the data. As a typical example of evaluation methods, comparing the model utility with the retrained model (Golatkar et al., 2020; Nguyen et al., 2022) can only be conducted by the model provider with full access. For MIA, the attacker requires extensive knowledge of the model architecture for white-box attack variants, access to auxiliary or shadow data, and computational power to an extent similar to the model provider (Sommer et al., 2022). However, considering that a powerful third-party verifier can be seen as a potential membership-inference attacker, we conduct additional experiments to demonstrate whether our proposed adversarial methods can circumvent MIA. We adapt MIA against machine learning models (Shokri et al., 2017) to machine unlearning. We use a two-layer MLP as the attack model (discriminator). The attack model is trained using the same way, aiming at distinguishing the training samples (unlearned samples) from the test samples. We label the predictions of the unlearned data as positive data, and we randomly sample predictions of test data to ensure that the number of positive cases and the number of negative cases of the attack model are balanced. Ideally, the attack model is supposed to have low accuracy since the unlearned data is supposed to be removed from the trained model and be indistinguishable from the test samples. However, if the unlearning is ineffective, the unlearned data is still memorized by the unlearned model as the retained training samples, leading to a high attack accuracy. We record the AUC score of the attack under five different random seeds. The experimental results are shown in Table 8. From the results, we can observe that our proposed Adv-R and naive retraining have an attack accuracy around 50% (similar to random guessing), indicating that our proposed Adv-R is able to circumvent MIA. For simple target tasks (e.g. MNIST),



(a) Gradient norm over MNIST. (b) Gradient norm over CIFAR-10. (c) Gradient norm over SVHN.

Figure 8. Comparison of gradient norm over three datasets.

the predictions of the test samples are similar to the predictions of the training samples, and they are difficult to distinguish since the model learns well and has good generalizability (the predictions are correct and with high confidence). Hence, the attack accuracy is low. For difficult target tasks (e.g. CIFAR-10), the model might have a confident and accurate prediction for training samples but not for test samples. Subsequently, the training and test samples are easier to distinguish for attack models. This can also be seen as a limitation of membership inference attacks (ineffective for well-learned models) (Shafran et al., 2021).

B.4. Effect of Learning Rate on Verification Error

From Proposition 4.4, we obtain that we can find a small enough learning rate $\gamma^{(t)}$ and $\gamma_r^{(t)}$ to forge a **valid** PoRT based on Algorithm 2 for arbitrarily small reproducing error threshold ε. To verify the effect of learning rate on the verification error, we conduct experiments of different learning rate configurations over the MNIST dataset, using the MLP model. We choose five different values for learning rate $\gamma^{(t)}$ as 5×10^{-3}, 10^{-3}, 5×10^{-4}, 10^{-4}, and 5×10^{-5}. To simplify hyperparameter tuning, we directly set the forging learning rate the same as the original learning rate, i.e., $\gamma_r^{(t)} = \gamma^{(t)}$. Experimental results are shown in Figure 7. In particular, we show the percentage of verification errors (in different iterations) within the corresponding threshold in Figure 7(a), and show the statistics of verification error and the maximum value (representing the threshold ε and shown as red triangles) in Figure 7(b). From the results, we can observe that the verification error threshold (maximum) and the mean value of verification error decreases as the learning rate becomes smaller. In addition, in the case of $\gamma^{(t)} = \gamma_r^{(t)}$, we can obtain a corollary of Proposition 4.4 as $\varepsilon \geq LC\gamma^2 + 3C\gamma$ where $C = \max\{L_x C_D, G\}$. As the value of learning rate γ is very small (γ usually has a significantly lower order of magnitude compared to C), we can approximately omit the second-order term and obtain that $\varepsilon \propto \gamma$, which conforms to the experimental results shown in Figure 7(b).

B.5. Error Statistics of Reproducing Verification

We provide an in-depth analysis of the variation trend of the error statistics in Figure 4(b). Basically, as mentioned in the experiments, we attribute this result to the change in the gradient norm. We first take a look at the verification error from a theoretical perspective. Based on the proof of Proposition 4.4, when a batch contains the unlearned data, the verification error is related to b (the difference between the gradient on the original data and the gradient on the forging data); when a batch does not contain the unlearned data, the verification error is related to the stochastic gradient (the gradient on a random data point). As the number of unlearned data is small (the batches excluding the unlearned data distinctly outnumber the batches including the unlearned data), the mean verification error over an epoch is mainly determined by the norm of the stochastic gradient. For MNIST, the model learns well and the optimization converges fast (the accuracy nearly reaches 90% after the first epoch). Hence, the verification error remains stable and is relatively small. For CIFAR-10, the task is challenging for CNN, and the norm of the stochastic gradient grows larger and then decreases as the optimization converges (the convergence is not complete at the end, so we are not able to observe the plateau as MNIST). For SVHN, the difficulty of the task is between MNIST and CIFAR-10 for ResNet. Hence, the verification error goes through a short increase and then decrease, and finally goes into a stable plateau as MNIST. To support our insights, we plot the norm of the stochastic gradient (we directly use the gradient over the training batch) under different settings. The results are shown in Figure 8. The variation trend of the norm of stochastic gradients matches the verification error shown in Figure 4(b). The experimental

results demonstrate that the reproducing verification error is closely related to the gradient norm.

C. Deal with the Vulnerability of MUL Verification

In the aforementioned studies, we have exposed the vulnerability of the verification strategies of MUL, proved by theoretical and experimental results. Next, we would like to provide some simple insights on how to deal with vulnerability.

Retraining-based Adversarial Unlearning. Detecting retraining-based adversarial unlearning is extremely difficult because retraining-based adversarial unlearning does not explicitly utilize the unlearned data to update the model parameters. Consequently, the benefit for model providers from retraining-based adversarial unlearning is relatively small. In our observation, the performance of the unlearned model conducted by retraining-based adversarial unlearning is better than naive retraining but worse than original training, which conforms to the performance of approximate unlearning methods (Guo et al., 2020; Golatkar et al., 2020; 2021; Mehta et al., 2022). Unfortunately, existing studies of verification for approximate unlearning remain nascent. However, it would be promising to investigate the safe verification of approximate unlearning and exploit the verification methods to detect the retraining-based adversarial unlearning method.

Forging-based Adversarial Unlearning. Forging-based adversarial unlearning is relatively easy to detect because of its bounded difference from the original model. According to our experimental results, forging-based adversarial unlearning can be detected by backdoor verification with high confidence (in Table 7). Correspondingly, the model provider can largely benefit from forging-based adversarial unlearning (high utility as the original model and low unlearning time cost). Although theoretically, the model provider can find a proper learning rate $\gamma^{(t)}$ and $\gamma_r^{(t)}$ to circumvent the reproducing verification with arbitrarily small threshold ε, the choice of learning rate is limited in practice. When model providers choose a smaller learning rate to circumvent more strict reproducing verification, the original training process can take longer time, even fail to converge. Hence, forging-based adversarial unlearning methods cannot circumvent arbitrarily strict reproducing verification, and the verifier can carefully select the error threshold to reject questionable unlearning operations.