UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Towards Efficient Machine Unlearning Based on Explainable Techniques

by Heng Xu

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of A/Prof Bo Liu & Dr. Dayong Ye

University of Technology Sydney
Faculty of Engineering and Information Technology

16 September 2025

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Heng Xu, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program (RTP) Scholarship doi.org/10.82133/C42F-K220.

SIGNATURE: Production Note: Signature removed prior to publication.

DATE: 16th Sep, 2025

PLACE: Sydney, Australia

Dedication

To all those who support me ...

Acknowledgments

I acknowledge Professor Tianqing Zhu, Professor Bo Liu, Professor Wanlei Zhou, Dr Dayong Ye, Dr. Lefeng Zhang for their valuable suggestions.

List of Publications

Related to the Thesis :

1. **Heng Xu**, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, Philip S Yu. *Machine Unlearning: A Survey*. ACM Computing Surveys. 56, pp: 1-36, 2024.

2. **Heng Xu**, Tianqing Zhu, Wanlei Zhou and Wei Zhao, *Don't Forget Too Much: Towards Machine Unlearning on Feature Level*, in IEEE Transactions on Dependable and Secure Computing, doi: 10.1109/TDSC.2024.3432169

3. **Heng Xu**, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou and Wei Zhao, *Toward Efficient Target-Level Machine Unlearning Based on Essential Graph*, in IEEE Transactions on Neural Networks and Learning Systems, doi: 10.1109/TNNLS.2024.3514607.

4. **Heng Xu**, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, Philip S Yu, *Update Selective Parameters: Federated Machine Unlearning Based on Model Explanation*, in IEEE Transactions on Big Data, doi: 10.1109/TBDATA.2024.3409947

5. **Heng Xu**, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, *Really Unlearned? Verifying Machine Unlearning via Influential Sample Pairs*, Submitted to IEEE Transactions on Dependable and Secure Computing.

Others :

4. Xiaocui Dang, Priyadarsi Nanda, **Heng Xu**, Manoranjan Mohanty, Haiyu Deng, A Novel Scheme For Recommendation Unlearning Verification (RUV) Using Non-influential Trigger Data. International Conference on Security of Information and Networks, 2024.

5. Xiaocui Dang, Priyadarsi Nanda, **Heng Xu**, Haiyu Deng and Manoranjan Mohanty, Recommendation System Model Ownership Verification via Non-influential Watermarking. IEEE Consumer Communications & Networking Conference, 2025.

6. Peng Huang, Mengyao Hou, Tong Sun, **Heng Xu**, Chuanming Ma, Aiguo Zhou, *Sustainable groundwater management in coastal cities: Insights from groundwater potential and vulnerability using ensemble learning and knowledge-driven models*, Journal of Cleaner Production, 442, 141152, 2024.

7. Kaiyue Feng, Guangsheng Zhang, Huan Tian, **Heng Xu**, Yanjun Zhang, Tianqing Zhu, Ming Ding, and Bo Liu, RAGLeak: Membership Inference Attacks on RAG-based Large Language Models. The 30th Australasian Conference on Information Security and Privacy 2025.

Abstract

Machine learning methods have become a strong driving force in revolutionizing a wide range of applications. However, they are also bringing requests to delete training samples from models due to privacy, usability, or other entitlement requirements. Machine unlearning has emerged as a promising solution to address such deletion requests, and various methods have been explored in recent research. Despite this progress, existing machine unlearning schemes face several critical limitations: how to effectively perform unlearning in federated learning scenarios, how to achieve fine-grained unlearning at the target or feature level, and how to build a robust and trustworthy unlearning verification framework that cannot be easily bypassed. To address these challenges, this thesis makes the following contributions based on explainable techniques:

1. We first introduce the core concepts and objectives of machine unlearning, propose a novel taxonomy based on representative existing methods, summarize their key principles, and analyze their strengths and limitations. We also emphasize the importance of unlearning verification by reviewing current verification schemes.

2. We propose an efficient machine unlearning scheme in federated learning scenarios. We first use ablation studies to identify the most influential parameters for the class that need to be unlearned. To achieve unlearning, we design two unlearning schemes, including decentralized and centralized unlearning. Both update only these selected influential parameters. We also incorporate perturbed samples to accelerate the unlearning process and reduce resource costs. Experiments across various models and datasets confirm the effectiveness and efficiency.

3. We propose target unlearning to remove specific target information from a trained model. We use interpretability-guided pruning of influential parameters to achieve that. To minimize performance loss, we build a graph-based data capturing parameter dependencies and their relevance to the remaining data. Experiments show our method effectively balances unlearning effectiveness and model utility.

4. We propose feature unlearning for fine-grained removal of feature information. We handle two scenarios: with and without annotations. For the former, we apply adversarial learning; for the latter, we leverage model interpretability to isolate and remove features via intermediate layer outputs. Evaluation with accuracy variation and gradient visualization confirms the method's effectiveness.

5. We propose IndirectVerify, a formal scheme for verifying machine unlearning under untrusted MLaaS scenarios. We first expose the weaknesses of existing verification methods, then design a perturbation-based framework using trigger and reaction samples. The classification result of reaction samples reflects whether unlearning was successful. Theoretical analysis and experiments demonstrate that our scheme offers robust and efficient verification while maintaining model utility.

TABLE OF CONTENTS

List of Figures

Figure **Page**

List of Tables

CHAPTER 1

INTRODUCTION

In recent years, machine learning has made remarkable progress and has been widely explored across every field of artificial intelligence (AI) [65]. However, as AI becomes increasingly data-dependent, more and more factors, such as privacy concerns, regulations, and laws, are leading to a new type of request - to delete information. Specifically, concerned parties are requesting that particular samples be deleted from a training dataset and the impact of those samples be removed from an already-trained model [16, 69, 136]. This is because membership inference attacks [107] and model inversion attacks [38] can reveal information about those specific samples from models. More importantly, legislators around the world have wisely introduced laws that grant users *the right to be forgotten* [36, 116]. These regulations, which include the European Union's General Data Protection Regulation (GDPR) [2], the California Consumer Privacy Act (CCPA) [1], the Act on the Protection of Personal Information (APPI) [3], and Canada's proposed Consumer Privacy Protection Act (CPPA) [4], compel the deletion of private information.

Machine unlearning (a.k.a. selectively forgetting, data deletion, or scrubbing) requires that the samples and their influence should be completely and quickly removed from a training dataset and a trained model [39, 95]. The straightforward way to achieve machine unlearning involves retraining the model from scratch after deleting the samples from the training dataset. However, as the size of modern training datasets continues to grow, this method has become increasingly impractical due to excessive computational and time costs [15]. To address this challenge, recent research has introduced a range of alternative techniques [11, 24, 115, 123]. For example, Bourtoule et al. [11] proposed

a "Sharded, Isolated, Sliced and Aggregated" (**SISA**) unlearning framework, inspired by the data segmentation technique. Guo et al. [46] calculated the effect of samples that need to be unlearned based on the influence function, and directly adjusted model parameters to offset those effects to achieve unlearning.

However, existing unlearning methods still face several limitations. In the following sections, we will explore the ongoing challenges in current machine unlearning research area and highlight the gaps that need further investigation.

1.1 Research Questions

1. **How can we achieve machine unlearning under federated learning?** Federated learning is a special kind of distributed learning that is characterized by various unstable users distributed in different places, each of whom has control over their devices and data [68, 121]. Imteaj et al. [59] show that model providers are more likely to receive requests to remove specific samples from a model trained in a federated learning setting. For example, when a user quits the collaborative training process, they may ask for their contribution to be removed from the collaborative model. Therefore, how to efficiently realize machine unlearning in a federated learning setting, considering the limitations of such a setting, like unacceptable training data, unstable connections, etc., is worthy of research [82].

2. **How to achieve target-level machine unlearning?** Current machine unlearning approaches mainly focus on selectively forgetting at the instance level, where a cluster of samples or all samples associated with a specific class are removed. While effective for removing complete samples, these methods do not scale well to scenarios where only partial information, such as a specific target, within a sample needs to be forgotten [41, 42]. Here, we define a target as any object that appears within an instance, and we refer to the process of removing such objects as target unlearning. Target unlearning is necessary in scenarios where we need to erase the model's knowledge of specific, unimportant, sensitive, or anomalous objects. For example, this could be removing a particular category in a multi-label classification task or eliminating a specific object in an object detection setting.

3. **How to achieve feature-level machine unlearning?** Beyond removing specific targets from a model, many real-world scenarios require unlearning at the feature level, where only particular features are removed from the model. In image-based

models, features refer to distinctive patterns or attributes extracted from input images that help the model recognize and differentiate between visual elements. Representative features include age, gender, and skin color. Feature unlearning is valuable in many applications, including the removal of unnecessary visual features, adversarial patterns, or sensitive attributes. Moreover, feature unlearning offers a promising strategy for mitigating model bias. By directly removing features that contribute to biased outcomes, it can reduce unfair behavior in the model without significantly compromising its overall performance.

4. **How can we develop a robust and trustworthy machine unlearning verification framework for data providers that cannot be easily bypassed?** Most existing verification schemes-such as those based on membership inference attacks (MIAs), backdoor, relearning time, or accuracy metrics [40, 44]-are adapted from traditional learning and attack methods. However, these approaches often overlook a critical vulnerability: model providers can potentially bypass unlearning verification by applying rapid fine-tuning. For example, in backdoor-based verification schemes, a provider could quickly fine-tune the model using the unlearning samples with their true labels. This process modifies the model's output for those samples, undermining the verification mechanism. Similarly, in MIAs-based verification, the provider can align the outputs of unlearning samples with those of corresponding test samples, effectively bypassing verification. Therefore, there is an urgent need for an effective and robust formal verification solution.

1.2 Research Contributions

Based on those identified knowledge gaps, this thesis makes the following contributions:

1. **For research question one, this thesis proposes a more effective and efficient federated unlearning scheme based on model explanation:**

    a) We introduce an ablation study to evaluate the influence of each channel with respect to the unlearning class. This allows us to identify the most influential channels for the unlearning class in any trained model.

    b) We present two distinct unlearning schemes-decentralized unlearning and centralized unlearning-based on the server's level of knowledge. Both approaches update only the parameters of the most influential channels, significantly reducing communication, computation, and storage overhead.

c) We propose an enhance unlearning mechanism based on perturbed samples, which accelerates the unlearning process while maintaining the performance of the resulting model.

d) We conduct extensive analysis and experiments comparing our approach with recent state-of-the-art methods across various models and datasets. The results validate the effectiveness of our proposed scheme.

2. **For research question two, this thesis proposes a more effective and efficient unlearning scheme that focuses on removing partial targets from the model, which we name *target unlearning*.**

a) We initiate the study of machine unlearning at the target level by formally defining the problem of target unlearning and outlining its core challenges.

b) We explore target unlearning from a novel interpretability-driven perspective, selectively pruning the most influential parameters.

c) We introduce a fundamental graph-based data structure to represent the inter-layer relationships among influential parameters, while also accounting for the impact of the remaining data to reduce performance degradation.

d) We comprehensively evaluate our target unlearning framework on multiple datasets, model architectures, and learning tasks, demonstrating that our approach effectively balances unlearning performance with model utility.

3. **For research question three, this thesis proposes a refined granularity unlearning scheme referred to as *feature unlearning*.**

a) We address the problem of machine unlearning at the feature level and formally define two categories of unlearning requests: feature unlearning with known annotations and feature unlearning without annotations.

b) We introduce adversarial learning techniques to facilitate the feature unlearning with known annotations while keeping useful feature information for remaining model performance.

c) We introduce model interpretability to empower the output of one model's layer to decouple and identify various pattern features. These outputs, equipped with identifying capabilities, are then employed to facilitate unlearning without annotations.

d) We evaluate our feature unlearning approach using both quantitative and qualitative metrics. Beyond standard accuracy, we propose two additional evaluation methods: variation in accuracy and a gradient visualization-based scheme, to assess the effectiveness of feature unlearning.

4. **For research question four, this thesis proposes a formal verification scheme, IndirectVerify, to determine whether unlearning requests have been successfully executed.**

   a) We take the first step in addressing the machine unlearning verification problem in the context of powerful yet untrusted Machine Learning as a Service (MLaaS) providers. To motivate this, we demonstrate how existing verification schemes can be bypassed through fine-tuning-based techniques.

   b) We propose a perturbation-based verification framework called IndirectVerify, which leverages trigger samples included in the unlearning request and reaction samples, used for subsequent verification. The key idea is that the presence or absence of trigger samples influences the behavior of reaction samples, enabling indirect yet effective verification.

   c) We provide theoretical analysis and proofs grounded in influence functions to support the validity of IndirectVerify. Furthermore, we analyze its robustness, showing how it withstands fine-tuning-based evasion strategies.

   d) We conduct a comprehensive evaluation of IndirectVerify across diverse datasets, model architectures, and verification scenarios. The results show that IndirectVerify achieves robust verification effectiveness while preserving overall model utility.

1.3 Thesis Organization

This thesis is organized as follows:

1. **Chapter 2**: This chapter provides the background of machine learning and machine unlearning, including notations, and relevant literature.

2. **Chapter 3**: This chapter proposes a federated unlearning scheme to address the practical challenge of removing the influence of specific classes from a trained model in the federated learning setting. Our approach begins with an ablation study to

identify the most influential channels associated with the target classes. To facilitate class unlearning, we introduce two effective methods that selectively fine-tune only these influential channel parameters, guided by perturbation data. Both theoretical analysis and experimental results demonstrate that our scheme effectively removes the targeted classes while preserving the accuracy of the remaining data. Moreover, the proposed methods achieve efficient and rapid unlearning, making them well-suited for real-world federated learning environments.

3. **Chapter 4**: This chapter proposed a novel machine unlearning scheme-referred to as target unlearning-that selectively removes partial target information from a trained model. As part of our solution, we formally define the concept of target unlearning and highlight the unique challenges it presents. We then introduce explainability techniques to identify the most influential parameters related to a given target, and introduce a pruning-based unlearning method to effectively erase that target-specific information. To maintain the performance of the unlearned model, we construct a essential graph that captures the relationships among critical parameters across layers. This graph-based representation enables us to filter out parameters that are simultaneously important to both the target and the remaining data, thereby avoiding unnecessary performance loss. Experimental results demonstrate that our method can efficiently remove the influence of specified targets while preserving the model's utility on the remaining data.

4. **Chapter 5**: This chapter proposes an innovative machine unlearning approach that allows for the selective removal of feature information from a trained model. We address two types of unlearning requests: feature unlearning with known annotations and feature unlearning without annotations. For the case of unlearning with known annotations, we employ adversarial learning to remove feature-related information from the model. In the absence of annotations, we introduce a re-encoding and fine-tuning technique to achieve the same goal. Experimental results demonstrate that our approach effectively removes the impact of specific features while maintaining the model's accuracy in a fast and efficient manner.

5. **Chapter 6**: This chapter take the first step in addressing the machine unlearning verification challenge in scenarios involving untrustworthy Machine Learning as a Service (MLaaS) providers. Initially, we propose two distinct bypass schemes to highlight the limitations of existing verification methods based on membership inference attacks and backdoor attacks. To overcome these shortcomings, we intro-

duce a perturbation-based verification framework designed to generate influential sample pairs, consisting of trigger samples and reaction samples. Trigger samples are used in the unlearning request, while reaction samples are employed for subsequent verification. We provide theoretical proofs and analyses to validate the effectiveness of our proposed scheme and emphasize its robustness by detailing how it resists current fine-tuning and bypassing. Experimental results demonstrate that our approach maintains the utility of the model while achieving robust verification performance, offering a promising solution to the verification problem in machine unlearning.

6. **Chapter 7**: In the final chapter of the thesis, a concise summary of the contents and contributions of the research is provided.

CHAPTER 2

BACKGROUND

2.1 Machine Unlearning Basics

2.1.1 Definition of Machine Unlearning

Vectors are denoted as bold lowercase, e.g., $\mathbf{x}_i$, and space or set as italics in uppercase, e.g., $\mathscr{X}$. A general definition of machine learning is given based on a supervised learning setting. The instance space is defined as $\mathscr{X} \subseteq \mathbb{R}^d$, with the label space defined as $\mathscr{Y} \subseteq \mathbb{R}$. $\mathscr{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^d \times \mathbb{R}$ represents a training dataset, in which each sample $\mathbf{x}_i \in \mathscr{X}$ is a d-dimensional vector $\left(x_{i,j}\right)_{j=1}^d$, $y_i \in \mathscr{Y}$ is the corresponding label, and n is the size of $\mathscr{D}$. Let d be the dimension of $\mathbf{x}_i$ and let $\mathbf{x}_{i,j}$ denote the j-th feature in the sample $\mathbf{x}_i$ [1].

The purpose of machine learning is to build a model M with the parameters $\mathbf{w} \in \mathscr{H}$ based on a specific training algorithm $\mathscr{A}(\cdot)$, where $\mathscr{H}$ is the hypothesis space for $\mathbf{w}$. In machine unlearning, let $\mathscr{D}_u \subset \mathscr{D}$ be a subset of the training dataset, whose influence we want to remove from the trained model. Let its complement $\mathscr{D}_r = \mathscr{D}_u^{\complement} = \mathscr{D} / \mathscr{D}_u$ be the dataset that we want to retain, and let $\mathscr{R}(\cdot)$ and $\mathscr{U}(\cdot)$ represent the retraining process and unlearning process, respectively. $\mathbf{w}_r$ and $\mathbf{w}_u$ donate the parameters of the built models from those two processes. $P(a)$ represents the distribution of a variable a and $\mathscr{K}(\cdot)$ represents a measurement of the similarity of two distributions. When considering $\mathscr{K}(\cdot)$ as a Kullback-Leibler (KL) divergence, $\mathscr{K}(\cdot)$ is defined by $\mathrm{KL}(P(a) \| P(b)) := \mathbb{E}_{a \sim P(a)}[\log(P(a)/P(b))]$. Given two

[1] The main content of this chapter has been published in Heng Xu, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, Philip S Yu. Machine Unlearning: A Survey. ACM Computing Surveys. 56, pp: 1-36, 2024.

Table 2.1: Notations.

Notations	Explanation	Notations	Explanation
$\mathscr{X}$	The instance space	$\mathscr{Y}$	The label space
$\mathscr{D}$	The training dataset	$\mathscr{D}_r$	The remaining dataset
$\mathscr{D}_u$	The unlearning dataset	$\mathbf{x}_i$	One sample in $\mathscr{D}$
y_i	The label of sample $\mathbf{x}_i$	n	The size of $\mathscr{D}$
$\mathbf{x}_{i,j}$	The j-th feature in $\mathbf{x}_i$	d	The dimension of $\mathbf{x}_i$
$\mathscr{A}(\cdot)$	The learning process	$\mathscr{U}(\cdot)$	The unlearning process
$\mathscr{R}(\cdot)$	The retraining process	$\mathbf{w}$	The parameters of learned model
$\mathbf{w}_u$	The parameters of unlearned model	$\mathbf{w}_r$	The parameters of retrained model
$P(\cdot)$	The distribution function	$\mathscr{K}(\cdot)$	The distribution measurement
$\mathscr{I}(\cdot)$	The shannon mutual information	$\mathscr{H}$	The hypothesis space for $\mathbf{w}$

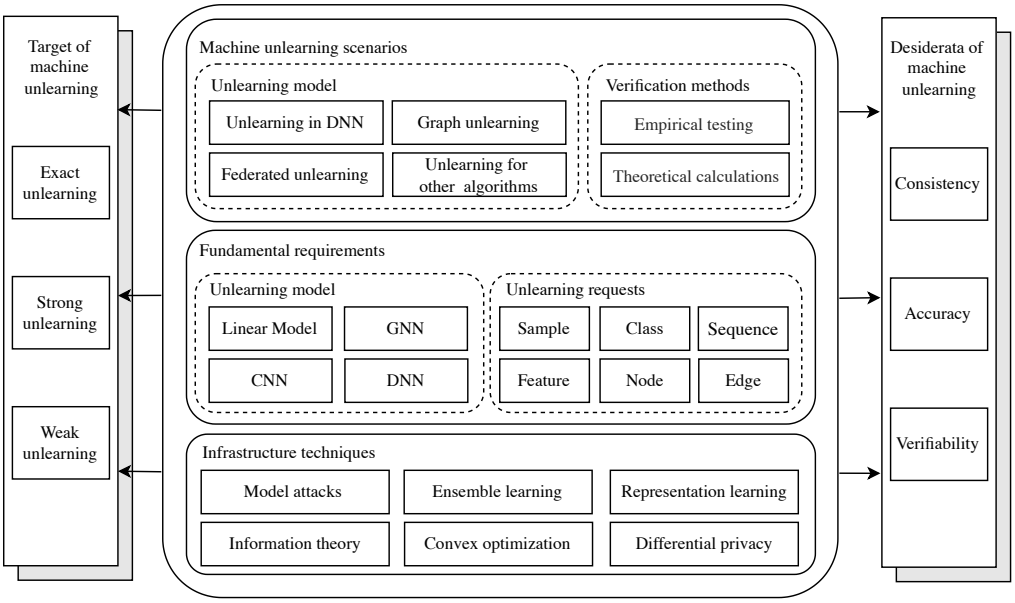


Figure 2.1: Machine Unlearning and Its Ecosystem.

random variables a and b, the amount of Shannon Mutual Information that a has about b is defined as $I(a;b)$. The main notations are summarized in Table 2.1.

Now we give the definition of machine unlearning.

Definition 1 (Machine Unlearning [15])**.** *Consider a cluster of samples that we want to remove from the training dataset and the trained model, denoted as $\mathscr{D}_u$. An unlearning process $\mathscr{U}(\mathscr{A}(\mathscr{D}),\mathscr{D},\mathscr{D}_u)$ is defined as a function from an trained model $\mathscr{A}(\mathscr{D})$, a training dataset $\mathscr{D}$, and an unlearning dataset $\mathscr{D}_u$ to a model $\mathbf{w}_u$, which ensures that the unlearned model $\mathbf{w}_u$ performs as though it had never seen the unlearning dataset $\mathscr{D}_u$.*

Figure 2.1 presents the typical concept, unlearning targets and desiderata associated

with machine unlearning. The infrastructure techniques involved in machine unlearning include several aspects, such as ensemble learning, convex optimization, and so on [70]. These technologies provide robust guarantees for different foundational unlearning requirements that consists of various types of models and unlearning requests, resulting in diverse unlearning scenarios and corresponding verification methods. Additionally, to ensure effectiveness, the unlearning process requires different targets, such as exact unlearning or strong unlearning. Each unlearning target ensures different similarities in the distribution of the parameters between the unlearned model and that of the retrained model. Machine unlearning also involves several unlearning desiderata, including consistency, accuracy, and verifiability. Those desiderata, with the target constraint, simultaneously guarantee the validity and feasibility of each unlearning scheme.

2.1.2 Targets of Machine Unlearning

The ultimate goal of machine unlearning is to reproduce a model that: 1) behaves as if trained without seeing the unlearned data, and 2) consumes minimal time and resources. The performance baseline for an unlearned model is the model retrained from scratch, also called *native retraining*.

Definition 2 (Native retraining [15])**.** *Supposing the learning process, $\mathscr{A}(\cdot)$, never sees the unlearning dataset $\mathscr{D}_u$, and thereby performs a retraining process on the remaining dataset, denoted as $\mathscr{D}_r = \mathscr{D} \setminus \mathscr{D}_u$. In this manner, the retraining process is defined as:*

$$\mathbf{w}_r = \mathscr{A}(\mathscr{D} \setminus \mathscr{D}_u) \tag{2.1}$$

Native retraining ensures that any information about the samples can be unlearned both from the training dataset and the trained model. However, it is often computationally expensive, and retraining may be infeasible when the training data is inaccessible, such as in federated learning [85]. Therefore, two alternative unlearning targets have been proposed: exact unlearning and approximate unlearning.

Definition 3 (Exact unlearning [41])**.** *Given a distribution measurement $\mathscr{K}(\cdot)$, such as KL-divergence, the unlearning process $\mathscr{U}(\cdot)$ will provide an exact unlearning target if*

$$\mathscr{K}(P(\mathscr{U}(\mathscr{A}(\mathscr{D}), \mathscr{D}, \mathscr{D}_u)), P(\mathscr{A}(\mathscr{D} \setminus \mathscr{D}_u))) = 0, \tag{2.2}$$

where $P(\cdot)$ denotes the distribution of the weights.

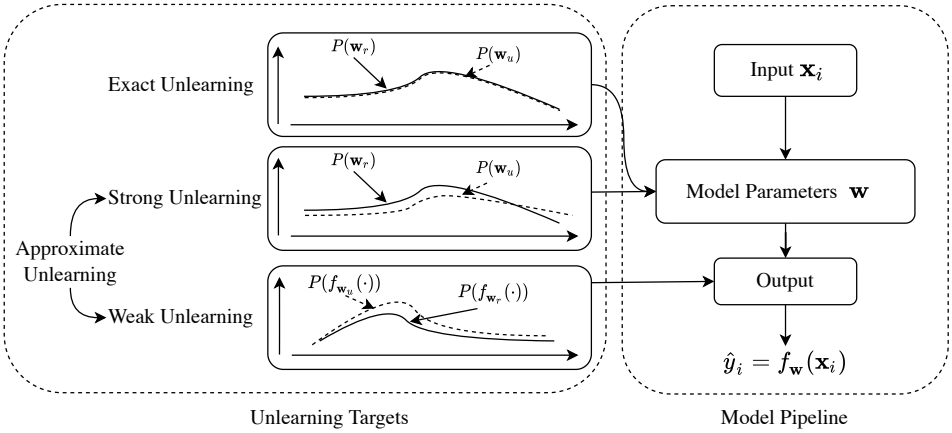


Figure 2.2: Targets of Machine Unlearning.

Exact unlearning guarantees that the output distributions of the unlearned model and the retrained model are indistinguishable, preventing an observer (e.g., an attacker) from recovering information about $\mathscr{D}_u$. However, exact unlearning is typically feasible only for simple or well-structured models [99].

Definition 4 (Approximate unlearning [114])**.** *If* $\mathscr{K}(P(\mathscr{U}(\mathscr{A}(\mathscr{D}),\mathscr{D},\mathscr{D}_u)),P(\mathscr{A}(\mathscr{D}\setminus\mathscr{D}_u)))$ *is limited within a tolerable threshold, the unlearning process* $\mathscr{U}(\cdot)$ *is defined as approximate unlearning.*

Approximate unlearning ensures that the distributions of the unlearned model and the retrained model are only approximately indistinguishable. This approximation is often achieved via differential privacy techniques, such as (ε,δ)-certified unlearning [46, 102]. Depending on how the distribution similarity is measured, approximate unlearning can be further classified into *strong unlearning* (based on internal parameter distributions) and *weak unlearning* (based on the distributions of final activations) [42, 102]. Table 2.2 summarizes the main differences between unlearning targets.

The significance of having these targets lies in providing a fundamental taxonomy and evaluation framework for machine unlearning. They allow researchers to systematically observe and compare the outcomes achieved by different unlearning approaches.

The targets of machine unlearning-namely exact, strong, and weak unlearning-serve as operational definitions that connect directly to the ultimate goals of unlearning. The overarching unlearning goal is to remove the influence of specific data from a model while maintaining acceptable performance and minimizing computational cost. The targets provide a practical framework to quantify and categorize how well an unlearning approach achieves this goal. Specifically, exact unlearning aims for full

Table 2.2: Summary and Comparison of Difference Between Targets.

Tartgets	Aims	Advantages	Limitations
Exact Unlearning	To make the distributions of a natively retrained model and an unlearned model indistinguishable	Ensures that attackers cannot recover any information from the unlearned model	Difficult to implement
Strong Unlearning	To ensure that the distributions of two models are approximately indistinguishable	Easier to implement than exact unlearning	Attackers can still recover some information from the unlearned model
Weak Unlearning	To only ensure that the distributions of two final activations are indistinguishable	The easiest target for machine unlearning	Cannot guarantee whether the internal parameters of the model are successfully unlearned

indistinguishability with a retrained model, strong unlearning allows approximate indistinguishability at the parameter level, and weak unlearning ensures only the outputs or final activations are close to those of the retrained model. In this way, the targets offer a structured way to evaluate and compare different unlearning methods in relation to the overall unlearning objectives.

2.1.3 Desiderata of Machine Unlearning

To fairly and accurately assess the efficiency and effectiveness of unlearning approaches, there are some mathematical properties that can be used for evaluation.

Definition 5 (Consistency)**.** *Assume there is a set of samples X_e, with the true labels $Y_e : \{y_1^e, y_2^e, \ldots, y_n^e\}$. Let $Y_n : \{y_1^n, y_2^n, \ldots, y_n^n\}$ and $Y_u : \{y_1^u, y_2^u, \ldots, y_n^u\}$ be the predicted labels produced from a retrained model and an unlearned model, respectively. If all $y_i^n = y_i^u, 1 \leq i \leq n$, the unlearning process is considered to provide the consistency property.*

Consistency denotes how similar the behavior of a retrained model and an unlearned model is. It represents whether the unlearning strategy can effectively remove all the information of the unlearning dataset $\mathscr{D}_u$. If, for every sample, the unlearned model gives the same prediction result as the retrained model, then an attacker has no way to infer information about the unlearned data.

Definition 6 (Accuracy)**.** *Given a set of samples X_e in remaining dataset, where their true labels are $Y_e : \{y_1^e, y_2^e, \ldots, y_n^e\}$. Let $Y_u : \{y_1^u, y_2^u, \ldots, y_n^u\}$ to denote the predicted labels produced by the model after the unlearning process, $\mathbf{w}_u = \mathscr{U}(\mathscr{A}(\mathscr{D}), \mathscr{D}, \mathscr{D}_u)$. The unlearning process is considered to provide the accuracy property if all $y_i^u = y_i^e, 1 \leq i \leq n$.*

Accuracy refers to the ability of the unlearned model to predict samples correctly. It reveals the usability of a model after the unlearning process, given that a model with low accuracy is useless in practice. Accuracy is a key component of any unlearning mechanism, as we claim the unlearning mechanism is ineffective if the process significantly undermines the original model's accuracy.

Definition 7 (Verifiability)**.** *After the unlearning process, a verification function $\mathcal{V}(\cdot)$ can make a distinguishable check, that is, $\mathcal{V}(\mathcal{A}(\mathcal{D})) \neq \mathcal{V}(\mathcal{U}(\mathcal{A}(\mathcal{D}),\mathcal{D},\mathcal{D}_u))$. The unlearning process $\mathcal{U}(\mathcal{A}(\mathcal{D}),\mathcal{D},\mathcal{D}_u)$ can then provide a verifiability property.*

Verifiability can be used to measure whether a model provider has successfully unlearned the requested unlearning dataset $\mathcal{D}_u$. Taking the following backdoor verification method as an example [130], if the pre-injected backdoor for an unlearned sample $\mathbf{x}_d$ is verified as existing in $\mathcal{A}(\mathcal{D})$ but not $\mathcal{U}(\mathcal{A}(\mathcal{D}),\mathcal{D},\mathcal{D}_u)$, that is $\mathcal{V}(\mathcal{A}(\mathcal{D})) = true$ and $\mathcal{V}(\mathcal{U}(\mathcal{A}(\mathcal{D}),\mathcal{D},\mathcal{D}_u)) = false$, the unlearning method $\mathcal{U}(\mathcal{A}(\mathcal{D}),\mathcal{D},\mathcal{D}_u)$ can be deemed to provide verifiability property.

2.2 Taxonomy of Machine Unlearning

We summarizes the general taxonomy of machine unlearning and its verification. The taxonomy is inspired by the design details of the unlearning strategy. Unlearning approaches that concentrate on modifying the training data are classified in data reorganization, while methods that directly manipulate the weights of a trained model are denoted as model manipulation. As for verification methods, initially, we categorize those schemes as either experimental or theoretical; subsequently, we summarize these methods based on the metrics they use.

2.2.1 Taxonomy of Existing Machine Unlearning Schemes

2.2.1.1 Data Reorganization

Data reorganization refers to the technique that a model provider unlearns data by reorganizing the dataset. It mainly includes three different processing methods according to the different data reorganization modes: *obfuscation*, *pruning* and *replacement* [11, 44]. Table 2.3 compares and summarizes the differences between these schemes.

- Data obfuscation: In data obfuscation, model providers intentionally add some choreographed data to the remaining dataset, that is $\mathcal{D}_{new} \leftarrow \mathcal{D}_r \cup \mathcal{D}_{obf}$, where $\mathcal{D}_{new}$

and $\mathscr{D}_{obf}$ are the new training dataset and the choreographed data, respectively. The trained model is then fine-tuned based on $\mathscr{D}_{new}$ to unlearn some specific samples. Such methods are usually based on the idea of erasing information about $\mathscr{D}_u$ by recombining the dataset with choreographed data. For example, Graves et al. [44] relabeled $\mathscr{D}_u$ with randomly selected incorrect labels and then fine-tuned the trained model for several iterations for unlearning data.

- Data pruning: In data pruning, the model provider first segments the training dataset into several sub-datasets and trains several sub-models based on each sub-dataset. Those sub-models are then used to aggregate a consensus prediction collaboratively, that is $\mathscr{D} \rightarrow \mathscr{D}_1 \cup \mathscr{D}_2 \cup ... \cup \mathscr{D}_m$, $\mathbf{w}_i = \mathscr{A}(\mathscr{D}_i)$ and $f(\mathbf{x}) = Agg(M_{\mathbf{w}_i}(\mathbf{x}))$, where $\mathscr{D}_i$, $0 < i < m$ are the sub-datasets, and $\cap\mathscr{D}_i = \varnothing$, $\cup\mathscr{D}_i = \mathscr{D}$, m is the number of sub-dataset, $\mathbf{w}_i$ is the sub-model, and $Agg(\cdot)$ is the aggregation function. After an unlearning request arrives, the model provider deletes the unlearned samples from the sub-datasets that contain them and then retrains the affected sub-models. The flexibility of this methodology is that the influence of unlearning dataset $\mathscr{D}_u$ is limited to each sub-dataset after segmentation rather than the whole dataset. Taking the **SISA** scheme in [11] as an example, the **SISA** framework first randomly divided the training dataset into k shards. A series of models are then trained separately at one per shard. When a sample needs to be unlearned, it is first removed from the shards that contain it, and only the sub-models corresponding to those shards are retrained.

- Data replacement: In data replacement, the model provider deliberately replaces the training dataset $\mathscr{D}$ with some new transformed dataset, that is $\mathscr{D}_{trans} \leftarrow \mathscr{D}$. The transformed dataset $\mathscr{D}_{trans}$ is then used to train a model that makes it easy to implement unlearning after receiving an unlearning request. For example, Cao et al. [15] replaced the training dataset with several efficiently computable transformations and used those transformations to complete the training of the model. Those transformations can be updated much more quickly after removing any samples from the transformed dataset. Consequently, computational overheads are reduced, and unlearning operations are more efficient.

2.2.1.2 Model Manipulation

In model manipulation, the model provider aims to realize unlearning operations by adjusting the model's parameters. It also mainly includes the following three categories.

Table 2.3: Summary and Comparison of Differences Between Unlearning Schemes.

	Schemes	Basic Ideas	Advantages	Limitations
Data Reorganization	Data Obfuscation [34, 44, 112]	Intentionally adds choreographed data to the training set and finetunes the model	Applicable to almost all model types; requires limited redundant data; less storage demand	Difficult to fully unlearn information; moderate accuracy and consistency; moderate implementation complexity
	Data Pruning [11, 15, 24] [49, 53, 94]	Removes unlearned samples from sub-datasets containing them, then retrains only the affected sub-models	Simple and intuitive; less implementation complexity	Requires extra storage; accuracy and consistency may decline with increasing sub-datasets
	Data Replacement [15]	Replaces the original dataset with a newly transformed dataset	Enables complete removal of targeted information;	moderate accuracy and consistency; moderate implementation complexity
Model Manipulation	Model Shifting [44, 99] [41, 46, 60] [40, 42, 102, 123]	Directly adjusts model parameters to remove the influence of unlearned samples	Requires little intermediate storage; allows theoretical verification	Finding suitable offsets for complex models is challenging; accuracy and consistency depend heavily on offsets; high implementation complexity
	Model Pruning [9, 81, 120]	Prunes key parameters from already-trained models	Low intermediate storage cost; fast unlearning process	Effective only for partial models; less unlearning effectiveness; moderately complex to implement
	Model Replacement [12, 21, 100, 126]	Replaces partial parameters with pre-calculated values	Effectively eliminates targeted information;	Significantly depend model structure; high implementation complexity

Table 2.3 compares and summarizes the differences between these schemes.

- Model shifting: In model shifting, the model providers directly update the model parameters to offset the impact of unlearned samples on the model, that is $\mathbf{w}_u = \mathbf{w}+\delta$, where $\mathbf{w}$ are parameters of the originally-trained model, and δ is the updated value. These methods are usually based on the idea of calculating the influence of samples on the model parameters and then updating the model parameters to remove that influence. It is usually extremely difficult to accurately calculate a sample's influence on a model's parameters, especially with complex deep neural models. Therefore, many model shifting-based unlearning schemes are based on specific assumptions. For example, Guo et al.'s [46] unlearning algorithms are designed for linear models with strongly convex regularization.

- Model replacement: In model replacement, the model provider directly replaces some parameters with pre-calculated parameters, that is $\mathbf{w}_u \leftarrow \mathbf{w}_{noeffect} \cup \mathbf{w}_{pre}$, where $\mathbf{w}_u$ are parameters of the unlearned model, $\mathbf{w}_{noeffect}$ are partially unaffected static parameters, and $\mathbf{w}_{pre}$ are the pre-calculated parameters. These methods usually depend on a specific model structure to predict and calculate the affected

Table 2.4: Summary and Comparison of Different Verification Methods.

Methods	Basic Ideas	Advantages	Limitations
Retraining-based	Removes unlearned samples and retrains models	Intuitive and easy to understand	Only applicable to special unlearning schemes; high verification reliability; implementation is complex
Attack-based	Based on membership inference attacks or model inversion attacks	Intuitively measures the defense effect against some attacks	Provides only partial verification; effectiveness depends on the attack; implementation complexity moderate
Relearn time-based	Measures the time when the unlearned model regains performance on unlearned samples	Easy to understand and easy to implement	Inadequate verification capability; simple to implement; does not fully capture internal model changes
Accuracy-based	Same as a model trained without unlearned samples	Easy to understand and easy to implement	Inadequate verification capability; low verification reliability
Theory-based	Ensures similarity between the unlearned model and the retrained model.	Comprehensive and has theoretical support	Implementation is complex and only applies to some specified models; high verification reliability
Information bound-based	Measures the upper-bound of the residual information about the unlearned samples	Comprehensive and has theoretical support	Hard to implement and only applicable to some specified models; high verification reliability

parameters in advance. They are only suitable for some special machine learning models, such as decision trees or random forest models. Taking the method in [100] as an example, the affected intermediate decision nodes are replaced based on pre-calculated decision nodes so as to generate an unlearned model.

- Model pruning: In model pruning, the model provider prunes some parameters from the trained models to unlearn the given samples, that is $\mathbf{w}_u \leftarrow \mathbf{w}/\delta$, where $\mathbf{w}_u$ are the parameters of the unlearned model, $\mathbf{w}$ are the parameters of the trained model, and δ are the parameters that need to be removed. Such unlearning schemes are also usually based on specific model structures and are generally accompanied by a fine-tuning process to recover performance after the model is pruned. For example, Wang et al. [120] introduced the term frequency-inverse document frequency (TF-IDF) to quantize the class discrimination of channels in a convolutional neural network model, where channels with high TF-IDF scores are pruned.

2.2.2 Taxonomy of Existing Verification Mechanisms

Verifying whether the unlearning method has the verifiability property is not an easy task. Model providers may claim externally that they remove those influences from their models, but, in reality, this is not the case [53]. For data providers, proving that the model provider has completed the unlearning process may also be tricky, especially for complex deep models with huge training datasets. Removing a small portion of samples only causes a negligible effect on the model. Moreover, even if the unlearned samples have indeed been removed, the model still has a great chance of making a correct prediction since other users may have provided similar samples. Therefore, providing a reasonable unlearning verification mechanism is a topic worthy of further research.

2.2.2.1 Empirical evaluation

- **Retraining-based verification**: Retraining naturally provides verifiability, as the retraining dataset no longer contains the samples to be unlearned. This makes it the most intuitive and easy-to-understand approach. These verification methods are specific to retraining-based unlearning approaches: the targeted data are first removed from the training set, and the model is then retrained. Because retraining occurs on a dataset without the unlearned samples, the process inherently verifies that the unlearning has been effectively performed.

- **Attack-based verification**: The essential purpose of an unlearning operation is to reduce leaks of sensitive information caused by model over-fitting. Hence, some attack methods can directly and effectively verify unlearning operations - for example, membership inference attacks [107] and model inversion attacks [69]. In addition, Sommer et al. [110] provided a novel backdoor verification mechanism from an individual user perspective in the context of machine learning as a service (MLaaS) [135]. This approach can verify, with high confidence, whether the service provider complies with the user's right to unlearn information.

- **Relearning time-based verification**: Relearning time can be used to measure the amount of information remaining in the model about the unlearned samples. If the model quickly recovers performance as the original trained model with little retraining time, then it is likely to still remember some information about the unlearned samples [112].

- **Accuracy-based verification**: A trained model usually has high prediction accuracy for the samples in the training dataset. This means the unlearning process can be verified by the accuracy of a model's output. For the data that need to be unlearned, the accuracy should ideally be the same as a model trained without seeing $\mathscr{D}_u$ [41]. In addition, if a model's accuracy after being attacked can be restored after unlearning the adversarial data, we can also claim that the unlearning is verified.

2.2.2.2 Theoretical calculation

- **Theory-based verification**: Some methods provide a certified unlearning definition [46, 123], which ensures that the unlearned model cannot be distinguished from a model trained on the remaining dataset from scratch. This could also provide a verification method that directly guarantees the proposed schemes.
- **Information bound-based verification**: Golatkar et al. [41, 42], devised a new metric for verifying the effectiveness of unlearning schemes, where they measured the upper bound of the residual information about samples that need to be unlearned. Less residual information represents a more effective unlearning.

Table 2.4 summarizes each verification method's advantages and limitations.

2.3 Related Works - Data Reorganization-Based

In this section, we review how data reorganization methods support the unlearning process. Since proving the verifiability property of unlearning algorithms is also important and should be considered in machine unlearning research, we separately discuss it for each unlearning method.

2.3.1 Reorganization Based on Data Obfuscation

2.3.1.1 Unlearning Schemes Based on Data Obfuscation

In general, the majority of model attack scenarios, such as membership inference attacks, arise from model overfitting and rely on observing shifts in the output based on known input shifts [54]. That is, for the vast majority of attackers, it is easy to perform an attack on some trained models by observing the shifts of the output confidence vectors. One optional machine unlearning scheme can be interpreted as confusing the model's

understanding of samples so that it cannot retain any correct information within models. This method can further confuse the confidence vector of the model's output [34]. As shown in Figure 2.3, when receiving an unlearning request, the model continues to train $\mathbf{w}$ based on the constructed obfuscation data $\mathscr{D}_{obf}$ giving rise to an updated $\mathbf{w}_u$.

In this vein, Graves et al. [44] proposed a random *relabel* and *retraining* machine unlearning framework. Sensitive samples are relabeled with randomly-selected incorrect labels, and then the machine learning model is fine-tuned based on the modified dataset for several iterations to unlearn those specific samples. Similarly, Felps et al. [34] intentionally poisoned the labels of the unlearning dataset and then fine-tuned the model based on the new poisoned dataset. However, such unlearning schemes only obscure the relationship between the model outputs and the individual samples; the model parameters may still retain information about these samples, since the unlearning process often involves fine-tuning the model on the samples intended to be forgotten.

The trained model is always trained by minimizing the loss for all classes. If one can learn a kind of noise that only maximizes the loss for some classes, those classes can be unlearned. Based on this idea, Tarrun et al. [112] divided the unlearning process into two steps, *impair* and *repair*. In the first step, an error-maximizing noise matrix is learned that consists of highly influential samples corresponding to the unlearning class. The effect of the noise matrix is somehow the opposite of the unlearning data, and can destroy the information of unlearned data to unlearn single/multiple classes. To repair the performance degradation caused by the model unlearning process, the *repair* step further adjusted the model based on the remaining data.

Similarly, Zhang et al. [137] considered the unlearning request in the image retrieval field. The approach developed involves creating noisy data using a generative method to adjust the weights of the retrieval model and achieve the unlearning purposes. They also proposed a new learning framework, which includes both static and dynamic learning branches, ensuring that the generated noisy data only affects the unlearning data being forgotten without affecting the contribution of other remaining data. However, the above two scheme consumes more time to generate noise for unlearning process, which will affect the efficiency of the unlearning process [112, 137].

2.3.1.2 Verifiability of Schemes Based on Data Obfuscation

To verify their unlearning process, Graves et al. [44] used two state-of-the-art attack methods - a model inversion attack and a membership inference attack - to evaluate how

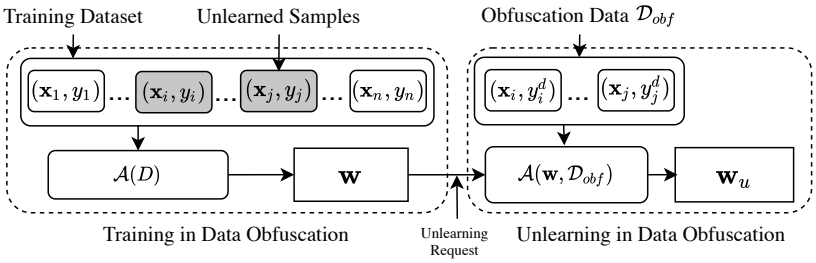


Figure 2.3: Unlearning Schemes Based on Data Obfuscation.

much information was retained in the model parameters about specific samples after the unlearning process - in other words, how much information might be leaked after the unlearning process. Their model inversion attack is a modified version of the standard model inversion attack proposed by Fredrikson et al. [38]. The three modifications include: adjusting the process function to every n gradient descent steps; adding a small amount of noise to each feature before each inversion; and modifying the number of attack iterations performed. These adjustments allowed them to analyze complex models. For the membership inference attack, they used the method outlined by Yeom et al. in [133]. Felps et al.'s verifiability analysis is also based on the membership inference attack [34].

In comparison, Tarrun et al. [112] evaluated the verifiability through several measurements. They first assessed relearning time by measuring the number of epochs for the unlearned model to reach the same accuracy as the originally-trained model. Then, the distance between the original model, the model after the unlearning process, and the retrained model are further evaluated.

2.3.2 Reorganization Based on Data Pruning

2.3.2.1 Unlearning Schemes Based on Data Pruning

As shown in Figure 2.4, unlearning schemes based on data pruning are usually based on ensemble learning techniques. Bourtoule et al. [11] proposed a "sharded, isolated, sliced, and aggregated" (**SISA**) framework, similar to the current distributed training strategies [43, 58], as a method of machine unlearning. With this approach, the training dataset $\mathscr{D}$ is first partitioned into k disjoint shards $\mathscr{D}_1, \mathscr{D}_2, \cdots, \mathscr{D}_k$. Then, sub-models $\mathscr{M}_w^1, \mathscr{M}_w^2, \cdots, \mathscr{M}_w^k$ are trained in isolation on each of these shards, which limits the influence of the samples to sub-models that were trained on the shards containing those

samples. At inference time, k individual predictions from each sub-model are simply aggregated to provide a global prediction (e.g., with majority voting), similar to the case of machine learning ensembles [66]. When the model owner receives a request to unlearn a data sample, they just need to retrain the sub-models whose shards contain that sample.

As the amount of unlearning data increases, **SISA** will cause degradation in model performance, making them only suitable for small-scale scenarios. The cost of these unlearning schemes is the time required to retrain the affected sub-models, which directly relates to the size of the shard. The smaller the shard, the lower the cost of the unlearning scheme. At the same time, there is less training dataset for each sub-model, which will indirectly degrade the ensemble model's accuracy. Bourtoule et al. [11] provided three key technologies to alleviate this problem, including *unlearning in the absence of isolation*, *data replication*, and *core-set selection*.

In addition to this scheme, Chen et al. [19] introduced the method developed in [11] to recommendation systems and designed three novel data partition algorithms to divide the recommendation training data into balanced groups in order to ensure that collaborative information was retained. Wei et al. [97] focused on the unlearning problems in patient similarity learning and proposed *PatEraser*. To maintain the comparison information between patients, they developed a new data partition strategy that groups patients with similar characteristics into multiple shards. Additionally, they also proposed a novel aggregation strategy to improve the global model utility.

Yan et al. [132] designed an efficient architecture for exact machine unlearning, called *ARCANE*, similar as the scheme in Bourtoule et al. [11]. Instead of dividing the dataset uniformly, they split it by class and utilized the one-class classifier to reduce the accuracy loss. Additionally, they also preprocessed each sub-dataset to speed up model retraining, which involved representative data selection, model training state saving, and data sorting by erasure probability. Nevertheless, the above unlearning schemes [11, 19, 132] usually need to cache a large number of intermediate results to complete the unlearning process. This will consume a lot of storage space.

SISA is designed to analyze Euclidean space data, such as images and text, rather than non-Euclidean space data, such as graphs. By now, numerous important real-world datasets are represented in the form of graphs, such as social networks [56], financial networks [26], biological networks [67], or transportation networks [32]. To analyze the rich information in these graphs, graph neural networks (GNNs) have shown unprecedented advantages [73, 127]. GNNs rely on the graph's structural information and

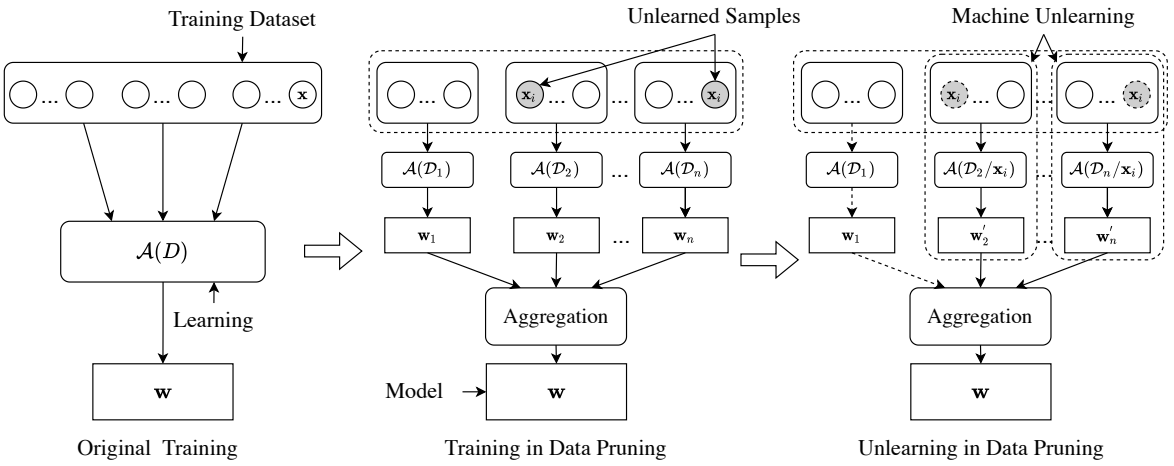


Figure 2.4: Unlearning Schemes Based on Data Pruning.

neighboring node features. Yet naively applying **SISA** scheme to GNNs for unlearning, i.e., randomly partitioning the training dataset into multiple sub-graphs, will destroy the training graph's structure and may severely damage the model's utility.

To allow efficient retraining while keeping the structural information of the graph dataset, Chen et al. [24] proposed *GraphEraser*, a novel machine unlearning scheme tailored to graph data. They first defined two common machine unlearning requests in graph scenario: node unlearning and edge unlearning, and proposed a general pipeline for graph unlearning, which is composed of three main steps: *graph partitioning*, *shard model training*, and *shard model aggravation*. In the *graph partitioning* step, they introduced an improved balanced *label propagation algorithm* (LPA) [51] and a balanced *embedding k-means* [98] partitioning strategy to avoid highly unbalanced shard sizes. Given that the different sub-models might provide different contributions to the final prediction, they also proposed a learning-based aggregation method, *OptAggr*, that optimizes the importance score of each sub-model to improve global model utility ultimately.

Deterministic unlearning schemes, such as **SISA** [11] or *GraphEraser* [24], promise nothing about what can be learned about specific samples from the difference between a trained model and an unlearned model. This could exacerbate user privacy issues if an attacker has access to the model before and after the unlearning operation [23]. To avoid this situation, an effective approach is to hide the information about the unlearned model when performing the unlearning operation.

In practical applications, Neel et al. [94] proposed an update-based unlearning method that performs several gradient descent updates to build an unlearned model. The method is designed to handle arbitrarily long sequences of unlearning requests with stable

run-time and steady-state errors. In addition, to alleviate the above unlearning problem, they introduced the concept of *secret state*: an unlearning operation is first performed on the trained model. Then, the unlearned models are perturbed by adding Gaussian noise for publication. This effectively ensures that an attacker cannot access the unlearned model actually after the unlearning operation, which effectively hides any sensitive information in the unlearned model. They also provided an (ϵ, δ)-certified unlearning guarantee, and leveraged a distributed optimization algorithm and reservoir sampling to grant improved accuracy/run-time tradeoffs for sufficiently high dimensional data.

After the initial model deployment, data providers may make an adaptive unlearning decision. For example, when a security researcher releases a new model attack method that identifies a specific subset of the training dataset, the owners of these subsets may rapidly increase the number of deletion requests. Gupta et al. [49] define the above unlearning requests as adaptive requests and propose an adaptive sequential machine unlearning method using a variant of the **SISA** framework [11] as well as a differentially private aggregation method [33]. They give a general reduction of the unlearning guarantees from the adaptive sequences to the non-adaptive sequences using differential privacy and max-information theory [49]. A strong provable unlearning guarantee for adaptive unlearning sequences is also provided, combined with the previous works of non-adaptive guarantees for sequence unlearning requests.

He et al. [53] developed an unlearning approach for the deep learning model. They first introduce a process called *detrended fluctuation analysis* [96], which quantifies the influence of the unlearned data on the model parameters, termed *temporal residual memory*. They observed that this influence is subject to exponential decay, which fades at an increasing rate over time. Based on these results, intermediate models are retained during the training process and divided into four areas, named *unseen*, *deleted*, *affected* and *unaffected*. *Unseen* indicates that the unlearned sample has not yet arrived. *Deleted* includes the unlearning dataset. *Unaffected* and *affected* indicate whether temporal residual memory has lapsed or not. An unlearned model can be stitched by reusing the *unseen* and *unaffected* models and retraining the *affected* areas. However, this scheme does not provide any theoretical verification methods to ensure that the information about unlearning data to be unlearned is indeed removed from the model.

2.3.2.2 Verifiability of Schemes Based on Data Pruning

The unlearning schemes proposed in [11, 15, 19, 24, 97, 132] are essentially based

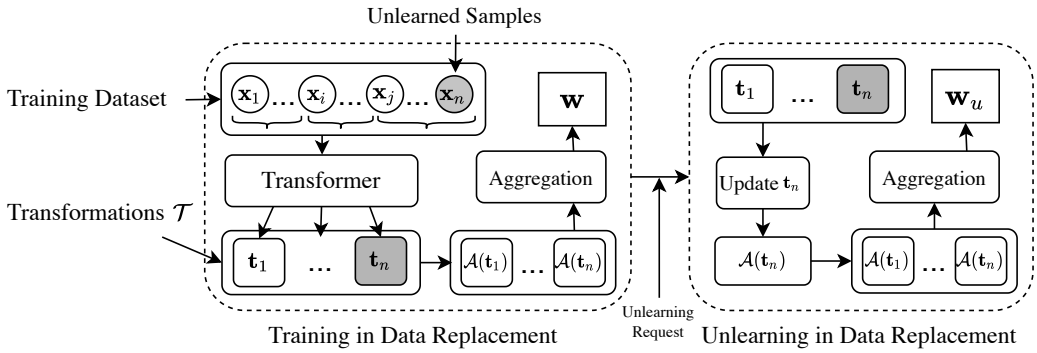


Figure 2.5: Unlearning Schemes Based on Data Replacement.

on a retraining mechanism that naturally has a verifiability property. As discussed in Section 2.1.3, a straightforward way to give an unlearning scheme the verifiability property is to retrain the model from scratch after removing the samples that need to be unlearned from the training dataset. The above schemes introduce distributed and ensemble learning techniques, which train sub-models separately and independently to optimize the loss function on each sub-dataset. The sub-models are then aggregated to make predictions. In terms of the unlearning process, only the affected sub-models are retrained, which avoids a large computational and time overhead and also provides a verifiability guarantee.

He et al. [53] use a backdoor verification method in [110] to verify their unlearning process. They designed a specially-crafted trigger and implanted this "backdoor data" in the samples that need to be unlearned, with little effect on the model's accuracy. They indirectly verify the validity of the unlearning process based on whether the backdoor data can be used to attack the unlearned model with a high success rate. If the attack result has lower accuracy, it proves that the proposed unlearning method has removed the unlearned data. The other studies [49, 94] did not provide a method for verifying the unlearning process.

2.3.3 Reorganization Based on Data Replacement

2.3.3.1 Unlearning Schemes Based on Data Replacement

As shown in Figure 2.5, when training a model in a data replacement scheme, the first step is usually to transform the training dataset into an easily unlearned type, named transformation $\mathcal{T}$. Those transformations are then used to separately train

models. When an unlearning request arrives, only a portion of the transformations $\mathbf{t}_i$ - the ones that contain the unlearned samples - need to be updated and used to retrain each sub-model to complete the machine unlearning.

Inspired by the previous work of using MapReduce to accelerate machine learning algorithms [109], Cao et al. [15] proposed a machine unlearning method that transforms the training dataset into summation form. Each summation is the sum of some efficiently computable transformation. The learning algorithms depend only on the summations, not the individual data, which breaks down the dependencies in the training dataset. To unlearn a data sample, the model provider only needs to update the summations affected by this sample and recompute the model. However, since the summation form comes from statistical query (SQ) learning, and only a few machine learning algorithms can be implemented as SQ learning, such as naïve bayes classifiers [28], support vector machines [45], and k-means clustering [128], this scheme has low applicability.

Takashi et al. [106] proposed a novel approach to lifelong learning named "Learning with Selective Forgetting", which involves updating a model for a new task by only forgetting specific classes from previous tasks while keeping the rest. To achieve this, the authors designed specific mnemonic codes, which are class-specific synthetic signals that are added to all the training samples of corresponding classes. Then, exploiting the mechanism of catastrophic forgetting, these codes were used to forget particular classes without requiring the original data. It is worth noting, however, that this scheme lacks any theoretical verification methods to confirm that the unlearning data information has been successfully removed from the model.

2.3.3.2 Verifiability of Schemes Based on Data Replacement

Cao et al. [15] provide an accuracy-based verification method. Specifically, they attack the LensKit model with the system inference attack method proposed by Calandrino et al. [14] and verify that the unlearning operations successfully prevent the attack from yielding any information. For the other three models, they first performed data pollution attacks to influence the accuracy of those models. They then analyzed whether the model's performance after the unlearning process was restored to the same state as before the pollution attacks. If the unlearned model was actually restored to its pre-pollution value, the unlearning operation was considered to be successful. Takashi et al. [106] provided a new metric, named *Learning with Selective Forgetting Measure (LSFM)* that is based on the idea of accuracy.

Table 2.5: The Surveyed Studies That Employed Data Reorganization Techniques for Unlearning Process.

Papers	Unlearning Methods	Unlearning Target	Training Dataset	Intermediates	Unlearned Samples' Type	Target Models' Type	Consistency	Accuracy	Verifiability
Graves et al. [44]	Data Obfuscation	Strong unlearning	Yes	No	Samples or Class	DNN	No	No	Attack-Based
Felps et al. [34]	Data Obfuscation	Strong unlearning	No	No	Sequences	DNN	No	No	Attack-Based
Tarrun et al. [112]	Data Obfuscation	Strong unlearning	Yes	No	Classes	DNN	No	No	Accuracy-Based and Retrain Time-Based
Zhang et al. [137]	Data Obfuscation	Strong unlearning	No	No	Samples	DNN	No	No	Accuracy-Based and Retrain Time-Based
Bourtoule et al. [11]	Data Pruning	Strong unlearning	Yes	Yes	Batches and Sequences	DNN	No	No	Retrain-Based
Chen et al. [19]	Data Pruning	Strong unlearning	Yes	Yes	Samples	DNN	No	No	Retrain-Based
Wei et al. [97]	Data Pruning	Strong unlearning	Yes	Yes	Samples	DNN	No	No	Retrain-Based
Yan et al. [132]	Data Pruning	Exact unlearning	Yes	Yes	Samples	DNN	Yes	Yes	Retrain-Based
Chen et al. [24]	Data Pruning	Exact unlearning	Yes	Yes	Nodes and Edges	GNN	No	No	Retrain-Based
Neel et al. [94]	Data Pruning	Strong unlearning	Yes	Yes	Non-adaptive Sequences	Convex Model	No	No	Retrain-Based
Gupta et al. [49]	Data Pruning	Strong unlearning	Yes	Yes	Adaptive Sequences	Non-convex Models	No	No	Retrain-Based
He et al. [53]	Data Pruning	Strong unlearning	Yes	Yes	Samples	DNN	No	No	Attack-Based
Cao et al. [15]	Data Replacement	Exact Unlearning	Yes	Yes	Samples	Statistical Query Learning Models	Yes	Yes	Retrain-Based and Accuracy-Based
Takashi et al. [106]	Data Replacement	Strong unlearning	No	Yes	Classes	DNN	No	No	Accuracy-Based

2.3.4 Summary of Data Reorganization

In these last few subsections, we reviewed the studies that use data obfuscation, data pruning, and data replacement techniques as unlearning methods. A summary of the surveyed studies is shown in Table 2.5, where we present the key differences between each paper.

From those summaries, we can see that most unlearning algorithms retain intermediate parameters and make use of the original training dataset [11, 24]. This is because those schemes usually segment the original training dataset and retrain the sub-models that were trained on the segments containing those unlearned samples. Consequently, the influence of specific samples is limited to only some of the sub-models and, in turn, the time taken to actually unlearn the samples is reduced. However, segmenting decreases time at the cost of additional storage. Thus, it would be well worth researching more efficient unlearning mechanisms that ensure the validity of the unlearning process and do not add too many storage costs simultaneously.

Moreover, these unlearning schemes usually support various unlearning requests and models, ranging from samples to classes or sequences and from support vector machines to complex deep neural models [15, 24, 94]. Unlearning schemes based on data reorganization rarely operate on the model directly. Instead, they achieve the unlearning purpose by modifying the distribution of the original training datasets and indirectly changing the obtained model. The benefit is that such techniques can be applied to more complex machine learning models. In addition to their high applicability, most of them can provide a strong unlearning guarantee, that is, the distribution of the unlearned model is approximately indistinguishable to that obtained by retraining.

It is worth pointing out that unlearning methods based on data reorganization will affect the consistency and the accuracy of the model as the unlearning process continues [11, 24, 53]. This reduction in accuracy stems from the fact that each sub-model is trained on the part of the dataset rather than the entire training dataset. This phenomenon does not guarantee that the accuracy of the unlearned model is the same as the result before the segmentation. Potential solutions are *to use unlearning in the absence of isolation*, *data replication* [11].

Some of the studies mentioned indirectly verify the unlearning process using a retraining method [11, 24], while others provide verifiability through attack-based or accuracy-based methods [34, 44, 112]. However, most unlearning schemes do not present further investigations at the theoretical level. The vast majority of the above unlearning schemes verify validity through experiments, with no support for the theoretical valid-

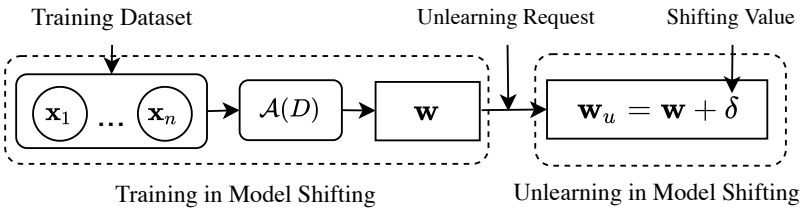


Figure 2.6: Unlearning Schemes Based on Model Shifting.

ity of the schemes. Theoretical validity would show, for example, how much sensitive information attackers can glean from an unlearned model after unlearning process or how similar the parameters of the unlearned model are to the retrained model. Further theoretical research into the validity of unlearning schemes is therefore required.

In summary, when faced with unlearning requests for complex models, unlearning schemes based on data obfuscation seldom unlearn information. This is because it is difficult to offset the influence of the unlearning data completely. Data pruning schemes always affect the model's accuracy since they usually train sub-models using a partial training dataset. For data replacement schemes, it is impossible to find a new dataset that can replace all the information within an original dataset to train a model. Thus, researchers should turn to design unlearning schemes that strike more of a balance between the effectiveness of the unlearning process and model usability.

2.4 Related Works - Model Manipulation-Based

The model training stage involves creating an effective model replicating the expected relationship between the inputs in the training dataset and the model's outputs. Thus, manipulating the model directly to remove specific relationships may be a good way to unlearn samples. In this section, we comprehensively review the state-of-the-art studies on unlearning through model manipulation. Again, the verification techniques are discussed separately for each category.

2.4.1 Manipulation Based on Model Shifting

2.4.1.1 Unlearning Schemes Based on Model Shifting

As shown in Figure 2.6, model shifting methods usually eliminate the influence of unlearning data by directly updating the model parameters. These methods mainly fall

into one of two types ‚Äì influence unlearning and Fisher unlearning - but there are a few other methods.

(1). Influence unlearning methods

Influence unlearning methods are usually based on influence theory [70]. Guo et al. [46] proposed a novel unlearning scheme called *certified removal*. Inspired by differential privacy [118], *certified removal* first limits the maximum difference between the unlearned and retrained models. Then, by applying a single step of Newton's method on the model parameters, a *certified removal* mechanism is provided for practical applications of L_2- regularized linear models that are trained using a differentiable convex loss function. Additionally, the training loss is perturbed with a loss perturbation technique that hides the *gradient residual*. This further prevents any adversaries from extracting information from the unlearned model. It is worth noting, however, that this solution is only applicable to simple machine learning models, such as linear models, or only adjusts the linear decision-making layer for deep neural networks, which does not eliminate the information of the removed data sample since the representations are still learned within the model.

Izzo et al. [60] proposed an unlearning method based on a gradient update called *projection residual update* (PRU). The method focuses on linear regression and shows how to improve the algorithm's run-time given in [46] from quadratic complexity to linear complexity. The unlearning intuition is as follows: if one can calculate the values $\hat{y}_{i_{\mathscr{D}_u}} = w_u\left(x_{i_{\mathscr{D}_u}}\right)$, predicted by the unlearned model on each of the unlearned samples $x_{i_{\mathscr{D}_u}}$ in $\mathscr{D}_u$ without knowing w_u, and then minimize the loss of already-trained model on the synthetic samples $\left(x_{i_{\mathscr{D}_u}}, \hat{y}_i\right)$, the parameters will move closer to $\mathbf{w}_u$ since it will achieve the minimum loss with samples $\left(x_{i_{\mathscr{D}_u}}, \hat{y}_{i_{\mathscr{D}_u}}\right)$. To calculate the values $\hat{y}_{i_{\mathscr{D}_u}}$ without knowing w_u, they introduced a statistics technique and computed leave-one-out residuals. Similar to the above, this method only considers the unlearning process in simple models.

Information leaks may not only manifest in a single data sample but also in groups of features and labels [123]. For example, a user's private data, such as their telephone number and place of residence, are collected by data providers multiple times and generated as different samples of the training dataset. Therefore, unlearning operations should also focus on unlearning a group of features and corresponding labels.

To solve such problems, Warnecke et al. [123] proposed a *certified unlearning* scheme for unlearning features and labels. By reformulating the influence estimation of samples on the already-trained models as a form of unlearning, they derived a versatile approach that maps changes of the training dataset in retrospection to closed-form updates of

the model parameters. They then proposed different unlearning methods based on *first-order* and *second-order* gradient updates for two different types of machine learning models. For the *first-order* update, the parameters were updated based on the difference between the gradient of the original and the perturbed samples. For the *second-order* update, they approximated an inverse Hessian matrix based on the scheme proposed in [5] and updated the model parameters based on this approximate matrix. Theoretical guarantees were also provided for feature and label unlearning by extending the concept of differential privacy [118] and certified unlearning [46]. However, this solution is only suitable for feature unlearning from tabular data and does not provide any effective solution for image features.

(2). Fisher unlearning method

The second type of model shifting technique uses the Fisher information [90] of the remaining dataset to unlearn specific samples, with noise injected to optimize the shifting effect. Golatkar et al. [41] proposed a weight *scrubbing* method to unlearn information about a particular class as a whole or a subset of samples within a class. They first give a computable upper bound to the amount of the information retained about the unlearning dataset after applying the unlearning procedure, which is based on the Kullback-Leibler (KL) divergence and Shannon mutual information. Then, an optimal quadratic unlearning algorithm based on a Newton update and a more robust unlearning procedure based on a noisy Newton update were proposed. Both schemes can ensure that a cohort can be unlearned while maintaining good accuracy for the remaining samples. However, this unlearning scheme is based on various assumptions, which limits its applicability.

For deep learning models, bounding the information that can be extracted from the perspective of weight or weight distribution is usually complex and may be too restrictive. Deep networks have a large number of equivalent solutions in the distribution space, which will provide the same activation on all test samples [42]. Therefore, many schemes have redirected unlearning operations from focusing on the weights to focus on the final activation.

Unlike their previous work, Golatkar et al. [42] provide bounds for how much information can be extracted from the final activation. They first transformed the bounding from a weight perspective to final activation based on Shannon mutual information and proposed a computable bound using the KL-divergence between the distribution of final activation of an unlearned model and retrained model. Inspired by the neural tangent kernel (NTK) [55, 61], they considered that deep network activations can be

approximated as a linear function of the weights. Hence, an optimal unlearning procedure is then provided based on a Fisher information matrix. However, due to the specific structure of deep neural networks, considering unlearning process only in the final activation layer may not satisfy the effectiveness of unlearning. Once an attacker obtains all model parameters in a white-box scenario, they can still infer information from the middle layers.

Golatkar et al. [40] also proposed a mix-privacy unlearning scheme based on a new *mixed-privacy* training process. This new training process assumes the traditional training dataset can be divided into two parts: *core* data and *user* data. Model training on the *core* data is non-convex, and then further training, based on the quadratic loss function, is done with the *user* data to meet the needs of specific user tasks. Based on this assumption, unlearning operations on the *user* data can be well executed based on the existing quadratic unlearning schemes. Finally, they also derived bounds on the amount of information that an attacker can extract from the model weights based on mutual information. Nevertheless, the assumption that the training dataset is divided into two parts and that the model is trained using different methods on each of these parts restricts unlearning requests to only those data that are easy to unlearn, making it difficult to unlearn other parts of the data.

Liu et al. [84] transferred the unlearning method from a centralized environment to federated learning by proposing a distributed Newton-type model updating algorithm to approximate the loss function trained by the local optimizer on the remaining dataset. This method is based on the Quasi-Newton method and uses a first-order Taylor expansion. They also use diagonal empirical Fisher Information Matrix (FIM) to efficiently and accurately approximate the inverse Hessian vector, rather than computing it directly, to further reduce the cost of the retraining process. However, this solution will result in a significant reduction in accuracy when dealing with complex models.

(3). Other Shifting Schemes

Schelter et al. [99] introduced the problem of making trained machine learning models unlearn data via *decremental updates*. They described three decremental update algorithms for different machine learning tasks. These included one based on item-based collaborative filtering, another based on ridge regression, and the last based on k-nearest neighbors. With each machine learning algorithm, the intermediate results are retained, and the model parameters are updated based on the intermediate results and unlearning data D_u, resulting in an unlearned model. However, this strategy can only be utilized with those models that can be straightforwardly computed to obtain the

model parameters after the unlearning process, limiting the applicability of this scheme.

In addition, Graves et al. [44] also proposed a laser-focused removal of sensitive data, called *amnesiac unlearning*. During training, the model provider retains a variable that stores which samples appear in which batch, as well as the parameter updates for each batch. When a data unlearning request arrives, the model owner undoes the parameter updates from only the batches containing the sensitive data, that is $\mathscr{M}_{w_u} = \mathscr{M}_w - \sum \Delta_w$, where $\mathscr{M}_w$ is the already-trained model and Δ_w are the parameter updates after each batch. Because undoing some parameters might greatly reduce the performance of the model, the model provider can perform a small amount of fine-tuning after an unlearning operation to regain performance. This approach requires the storage of a substantial amount of intermediate data. As the storage interval decreases, the amount of cached data increases, and smaller intervals lead to more efficient model unlearning. Therefore, a trade-off exists between efficiency and effectiveness in this method.

The above methods mainly focused on the core problem of empirical risk minimization, where the goal is to find approximate minimizers of the empirical loss on the remaining training dataset after unlearning samples [46, 60]. Sekhari et al. [102] proposed a more general method of reducing the loss of unseen samples after an unlearning process. They produced an unlearned model by removing the contribution of some samples from an already-trained model using a disturbance update calculated based on some cheap-to-store data statistics during training. In addition, they proposed an evaluation parameter to measure the unlearning capacity. They also improved the data unlearning capacity of convex loss functions, which saw a quadratic improvement in terms of the dependence of d over differential privacy, where d is the problem dimension.

2.4.1.2 Verifiability of Schemes Based on Parameter Shifting

Izzo et al. [60] provided two metrics to measure the effectiveness: L_2 *distance* and *feature injection test*. L_2 *distance* measures the distance between the unlearned model and the retrained model. If the L_2 *distance* is small, the models are guaranteed to make similar predictions, which could reduce the impact of output-based attacks, like a membership inference attack. The *feature injection test* can be thought of as a verification scheme based on a poisoning attack.

Golatkar et al. [40–42] verify the effectiveness of their unlearning schemes based on accuracy and relearning time. They also developed two new verification metrics: *model confidence* and *information bound* [41]. *Model confidence* is formulated by measuring

the distribution of the entropy of the output predictions on the remaining dataset, the unlearning dataset, and the test dataset. Then they evaluated the similarity of those distributions against the confidence of a trained model that has never seen the unlearning dataset. The higher the degree of similarity, the better the effect of the unlearning process. The *information bound* metric relies on KL-divergence to measure the information remaining about the unlearning dataset within the model after the unlearning process.

Different from their previous work, Golatkar et al. [42] also evaluate the information remaining within the weights and the activation. In their other work [40], they provided a new metric, *activation distance*, to analyze the distance between the final activations of an unlearned model and a retrained model. This is a similar metric to *model confidence* [41]. In addition, they also use attack-based methods for verification [40, 42].

Guo et al. [46], Warnecke et al. [123] and Sekhari et al. [102] provide a method of theoretical verification to verify the effectiveness of their proposed unlearning schemes. Based on the guarantee provided by *certified unlearning*, they limit the distribution similarity between the unlearned model and the retrained model. Warnecke et al. [123] also use the *exposure metric* [16] to measure the remaining information after unlearning. Liu et al. [84] analyzed the validity of the unlearning scheme through two aspects. The first metric, Symmetric Absolute Percentage Error (SAPE), is created based on accuracy. The second metric is the difference between the distribution of the model after the unlearning process and the distribution of the retraining model.

2.4.2 Manipulation Based on Model Pruning

2.4.2.1 Unlearning Schemes Based on Model Pruning

As shown in Figure 2.7, methods based on model pruning usually prune a trained model to produce a model that can meet the requests of unlearning. It is usually applied in the scenario of federated learning, where a model provider can modify the model's historical parameters as an update. Federated learning is a distributed machine learning framework that can train a unified deep learning model across multiple decentralized nodes, where each node holds its own local data samples for training, and those samples never need to be exchanged with any other nodes [80]. There are mainly three types of federated learning: *horizontal*, *vertical*, and *transfer learning* [59].

Based on the idea of trading the central server's storage for the unlearned model's

construction, Liu et al. [81] proposed an efficient federated unlearning methodology, *FedEraser*. Historical parameter updates from the clients are stored in the central server during the training process, and then the unlearning process unfolds in four steps: (1) *calibration training*, (2) *update calibrating*, (3) *calibrated update aggregating*, and (4) *unlearned model updating*, to achieve the unlearning purpose. In *calibration training* and *update calibration* steps, several rounds of a calibration retraining process are performed to approximate the unlearning updates without the target client. In the *calibrated update aggregating* and the *unlearned model updating* steps, standard federated learning aggregation operations are used to aggregate those unlearning updates and further update the global model. This eliminates the influence of the target data.

However, the effectiveness of this scheme will decrease dramatically as the number of unlearning requests increases; this is because the gradients are cached during the training phase, and the unlearning process will not update these gradients to satisfy subsequent unlearning requests [81]. Second, this solution also requires caching of intermediate data, which will cost more storage.

Inspired by the observation that different channels have a varying contribution to different classes in trained CNN models. Wang et al. [120] analyzed the problem of selectively unlearning classes in a federated learning setting. They introduced the concept of term frequency-inverse document frequency (TF-IDF) [113] to quantify the class discrimination of the channels. Similar to analyzing how relevant a word is to a document in a set of documents, they regarded the output of a channel as a word and the feature map of a category as a document. Channels with high TF-IDF scores have more discriminatory power in the target categories and thus need to be pruned. An unlearning procedure via channel pruning [47] was also provided, followed by a fine-tuning process to recover the performance of the pruned model. In their unlearning scheme, however, while the parameters associated with the class that needs to be unlearned are pruned, the parameters with other classes also become incomplete, which will affect the model performance. Therefore, the unlearned model is only available when the fined-tuned training process is complete.

Baumhauer et al. [9] provided a machine unlearning scheme based on linear filtration. They first transformed the existing logit-based classifier models into an integrated model which can be decomposed into a (potentially nonlinear) feature extraction, followed by a multinomial logistic regression. Then, they focused the unlearning operation on the logistic regression layer, proposing a ‚Äúblack-box‚Äù unlearning definition. To unlearn the given samples, four different filtration methods are defined, namely, *naive unlearning*,

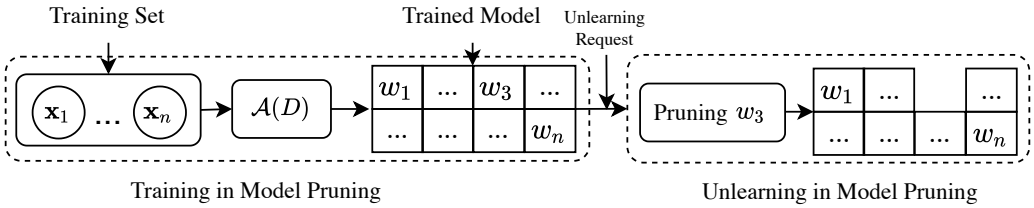


Figure 2.7: Unlearning Schemes Based on Model Pruning.

normalization, *randomization*, and *zeroing*. These effectively filter the outputs of the logistic regression layer. On the contrary, they only considered the unlearning process within the last layer, which will lead to a potential risk that if an attacker gets access to the model parameters of the middle layer, the information of unlearning data may also be leaked.

2.4.2.2 Verifiability of Schemes based on Model Pruning

Liu et al. [81] present an experimental verification method based on a membership inference attack. Two evaluation parameters are specified: *attack precision* and *attack recall*, where *attack precision* denotes the proportion of unlearned samples that is expected to participate in the training process. *Attack recall* denotes the fraction of unlearned samples that can be correctly inferred as part of the training dataset. In addition, a *prediction difference* metric is also provided, which measures the difference in prediction probabilities between the original global model and the unlearned model. Wang et al. [120] evaluate verifiability based on model accuracy.

Baumhauer et al. [9] defined a divergence measure based on a Bayes error rate for evaluating the similarity of the resulting distributions $P(\mathbf{L}_{\text{seen}})$ and $P(\mathbf{L}_{\neg\text{ seen}})$, where $\mathbf{L}_{\text{seen}}$ and $\mathbf{L}_{\neg\text{ seen}}$ are the pre-softmax outputs of the unlearned model and a retrained model. When the result of the Bayes error rate is close to 0, it indicates that $P(\mathbf{L}_{\text{seen}})$ and $P(\mathbf{L}_{\text{-seen}})$ are similar and the unlearning process has unlearned the sample's information from the model. In addition, they also use a model inversion attack to evaluate verifiability [38].

2.4.3 Manipulation Based on Model Replacement

2.4.3.1 Unlearning Schemes Based on Model Replacement

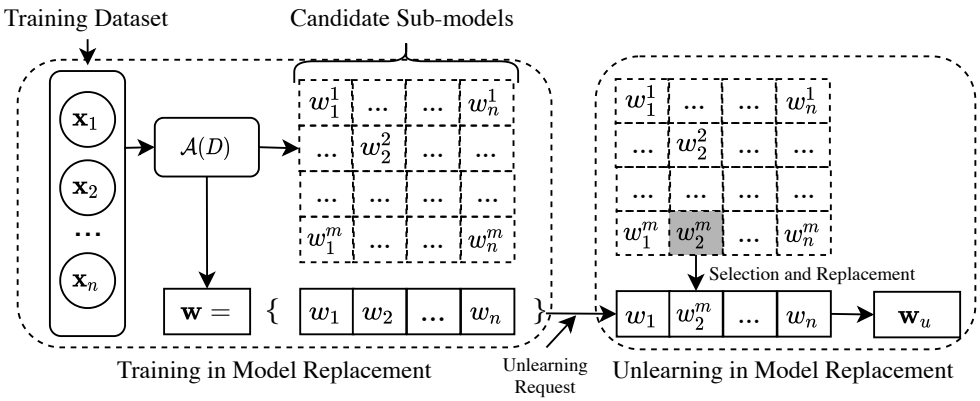


Figure 2.8: Unlearning Schemes Based on Model Replacement.

As shown in Figure 2.8, model replacement-based methods usually calculate almost all possible sub-models in advance during the training process and store them together with the deployed model. Then, when an unlearning request arrives, only the sub-models affected by the unlearning operation need to be replaced with the pre-stored sub-models. This type of solution is usually suitable for some machine learning models, such as tree-based models. Decision tree is a tree-based learning model, in which each leaf node represents a prediction value, and each internal node is a decision node associated with an attribute and threshold value. Random forest is an integrated decision tree model that aims to improve prediction performance [88, 104].

To improve the efficiency of the unlearning process for tree-based machine learning models, Schelter et al. [100] proposed *Hedgecut*, a classification model based on extremely randomized trees (ERTs) [27]. First, during the training process, the tree model is divided into robust splits and non-robust splits based on the proposed robustness quantification factor. A robust split indicates that the subtree's structure will not change after unlearning a small number of samples, while for non-robust splits, the structure may be changed. In the case of unlearning a training sample, *HedgeCut* will not revise robust splits, but will update those leaf statistics. For non-robust splits, *HedgeCut* recomputes the split criterion of the maintained subtree variants, which were previously kept inactive, and selects a subtree variant as a new non-robust split of the current model.

For the tree-based models, Brophy et al. [12] also proposed DaRE (Data Removal-Enabled) forests, a random forest variant that enables the efficient removal of training samples. DaRE is mainly based on the idea of retraining subtrees only as needed. Before the unlearning process, most k randomly-selected thresholds per attribute are computed, and intermediate statistics data are stored within each node in advance. This information

is sufficient to recompute the split criterion of each threshold without iterating through the data, which can greatly reduce the cost of recalculation when unlearning the dataset. They also introduced random nodes at the top of each tree. Intuitively, the nodes near the top of the tree affect more samples than those near the bottom, which makes it more expensive to retrain them when necessary. Random nodes minimally depend on the statistics of the data, rather than the way greedy methods are used, and rarely need to be retrained. Therefore, random nodes can further improve the efficiency of unlearning.

The above two schemes need to compute a large number of possible tree structures in advance, which would cost a large number of storage resources [12, 100]. Besides, this replacement scheme is difficult to be applied to other machine learning models, such as deep learning models, since it is difficult to achieve partial model structure after removing each sample in advance.

Chen et al. [21] proposed a machine unlearning scheme called *WGAN* unlearning, which removes information by reducing the output confidence of unlearned samples. Machine learning models usually have different confidence levels toward the model's outputs [17]. To reduce confidence, *WGAN* unlearning first initializes a generator as the trained model that needs to unlearn data. Then, the generator and discriminator are trained alternatingly until the discriminator cannot distinguish the output difference of the model between unlearning dataset and third-party data. Until this, the generator then becomes the final unlearned model. However, this method achieves unlearning process through an alternating training process, which brings a limited improvement in efficiency compared to the unlearning method of retraining from scratch.

Wu et al. [126] proposed an approximate unlearning method based on intermediate parameters cached during the training phase called *DeltaGrad*, which could quickly unlearn information from machine learning models that are based on gradient descent algorithms. They divided the retraining process into two parts. One part computes the full gradients exactly based on the remaining training dataset. The other part uses the L-BGFS algorithm [92] and a set of updates from some prior iterations to calculate Quasi-Hessians approximating the true Hessian-vector. These Quasi-Hessians are then used to approximate the update in the remaining process. These two parts train cooperatively to generate the unlearned model. This approach will reduce the performance of the model, however, after unlearning process since part of the model update is calculated based on the approximative methods. In addition, the number of iterations required for the model to converge will also increase, which will reduce the efficiency of the unlearning process.

2.4.3.2 Verifiability of Schemes Based on Model Replacement

Chen et al. [21] verified their proposed scheme with a membership inference attack and a technique based on false negative rates (FNRs) [71], where FNR: $FNR = \frac{FN}{TP+FN}$, TP means that the membership inference attack test samples were considered to be training dataset and FN means the data was deemed to be non-training data. If the target model successfully unlearns the samples, the member inference attack will treat the training dataset as non-training data. Thus FN will be large, while TP will be small, and the corresponding FNR will be large. Indirectly, this reflects the effectiveness of the unlearning process.

Schelter et al. [100], Brophy et al. [12] and Wu et al. [126] only provide evaluations in terms of runtime and accuracy, and they do not provide reasonable experimental or theoretical verifiability guarantees of their unlearning processes.

2.4.4 Summary of Model Manipulation

In these last subsections, we reviewed studies that apply model shifting, model pruning, and model replacement techniques as unlearning processes. A summary of the surveyed studies is shown in Table 2.6, where we list the key differences between each paper.

Compared to the unlearning schemes based on data reorganization, we can see that few of the above papers make use of intermediate data for unlearning. This is because the basic idea of those unlearning schemes is to directly manipulate the model itself, rather than the training dataset. The model manipulation methods calculate the influence of each sample and offset that influence using a range of techniques [70], while data reorganization schemes usually reorganize the training dataset to simplify the unlearning process. For this reason, model manipulation methods somewhat reduce the resource consumption used by intermediate storage.

Second, most of the above schemes focus on relatively simple machine learning problems, such as linear logistic regression, or complex models with special assumptions [41, 42, 46, 60]. Removing information from the weights of standard convolutional networks is still an open problem, and some preliminary results are only applicable to small-scale problems. One of the main challenges with unlearning processes for deep networks is how to estimate the impact of a given training sample on the model parameters. Also, the highly non-convex losses of CNNs make it very difficult to analyze those impacts on the optimization trajectory. Current research has focused on simpler convex

Table 2.6: The Surveyed Studies that Employed Model Manipulation Techniques for Unlearning Process.

Papers	Unlearning Methods	Unlearning Target	Training Dataset	Intermediates	Unlearned Samples' Type	Target Models' Type	Consistency	Accuracy	Verifiability
Guo et al. [46]	Model Shifting	Strong Unlearning	Yes	No	Samples	Linear Models with Strongly Convex Regularization	No	No	Theory-Based
Izzo et al. [60]	Model Shifting	Strong Unlearning	No	Yes	Batches	Linear and Logistic Regression Models	No	No	L^2 distance and Attack-Based
Warnecke et al. [123]	Model Shifting	Strong Unlearning	Yes	No	Features and Labels	Convex or Non-convex models	No	No	Theory-Based and Method in [16]
Golatkar et al. [41]	Model Shifting	Strong Unlearning	Yes	No	Samples in One Class	DNN	No	No	Accuracy-based, Relearn time-based, Model confidence and Information Bound-Based
Golatkar et al. [42]	Model Shifting	Strong Unlearning	Yes	No	Samples	DNN	No	No	Accuracy-based, Relearn time-based, Attack-based and Information Bound-Based
Liu et al. [84]	Model Shifting	Strong Unlearning	Yes	Yes	Samples	DNN	No	No	Accuracy-based
Golatkar et al. [40]	Model Shifting	Strong Unlearning	Yes	No	Samples	DNN	No	No	Accuracy-based, Relearn time-based, Activation distance and Attack-based
Schelter et al. [99]	Model Shifting	Exact Unlearning	No	Yes	Samples	Specified model	Yes	Yes	-
Graves et al. [44]	Model Shifting	Strong Unlearning	No	Yes	Samples	DNN	No	No	Attack-Based
Sekhari et al. [102]	Model Shifting	Strong Unlearning	No	Yes	Samples	Convex Models	No	No	Theory-Based
Wang et al. [120]	Model Pruning	Strong Unlearning	Yes	No	Client Data	Federated Learning Model	No	No	-
Baumhauer et al. [9]	Model Pruning	Weak Unlearning	No	No	Classes	Logit-Based Classifiers	No	No	Attack-Based
Schelter et al [100]	Model Replacement	Exact Unlearning	No	Yes	Samples	Extremely Randomized Trees	Yes	Yes	-
Brophy et al [12]	Model Replacement	Exact Unlearning	Yes	Yes	Batches	Random Forests	Yes	Yes	-
Chen et al. [21]	Model Replacement	Strong Unlearning	No	No	Samples	Deep Classifier Models	No	No	Attack-Based
Wu et al. [126]	Model Replacement	Strong Unlearning	Yes	Yes	Samples	SGD-Based Models	No	No	-

learning problems, such as linear or logistic regression, for which theoretical analysis is feasible. Therefore, evaluating the impact of specific samples and further proposing unlearning schemes for those models are two urgent research problems.

In addition, most model manipulation-based methods will affect the consistency or prediction accuracy of the original models. There are two main reasons for this problem. First, due to the complexity of calculating the impact of the specified sample on the model, manipulating a model's parameters based on unreliable impact results or assumptions will lead to a decline in model accuracy. Secondly, Wang et al.'s [120] scheme pruned specific parameters in the original models, which will also reduce the accuracy of the model due to the lack of some model prediction information. Thus, more efficient unlearning mechanisms, which simultaneously ensure the validity of the unlearning process and guarantee performance, are worthy of research.

It is worth pointing out that most schemes provide a reasonable method with which to evaluate the effectiveness of the unlearning process. Significantly, model manipulation methods usually give a verifiability guarantee using theory-based and information bound-based methods [41, 42, 46]. Compared to the simple verification methods based on accuracy, relearning, or attacks, the methods based on theory or information bounds are more effective. This is because simple verification methods usually verify effectiveness based on output confidence. While the effects of the samples to be unlearned can be hidden from the output of the network, insights may still be gleaned by probing deep into its weights. Therefore, calculating and limiting the maximum amount of information that may be leaked at the theoretical level will be a more convincing method. Overall, however, more theory-based techniques for evaluating verifiability are needed.

In summary, the unlearning methods based on model shifting usually aim to offer higher efficiency by making certain assumptions about the training process, such as which training dataset or optimization techniques have been used. In addition, those mechanisms that are effective for simple models, such as linear regression models, become more complex when faced with advanced deep neural networks. Model pruning schemes require far-reaching modifications of the existing architecture of the model in the unlearning process [9, 120], which could affect the performance of the unlearned models. It is worth noting that model replacement unlearning methods usually need to calculate all possible parameters and store them in advance, since they unlearn by quickly replacing the model parameters using these pre-calculated parameters. Thus, more effective unlearning schemes, that simultaneously consider model usability, storage costs, and the applicability of the unlearning process, are urgent research problems.

CHAPTER 3

UPDATE SELECTIVE PARAMETERS: FEDERATED MACHINE UNLEARNING BASED ON MODEL EXPLANATION

Federated learning is a promising privacy-preserving paradigm for distributed machine learning. In this context, there is sometimes a need for a specialized process called machine unlearning, which is required when the effect of some specific training samples needs to be removed from a learning model due to privacy, security, usability, and/or legislative factors. However, problems arise when current centralized unlearning methods are applied to existing federated learning, in which the server aims to remove all information about a class from the global model. Centralized unlearning usually focuses on simple models or is premised on the ability to access all training data at a central node. However, training data cannot be accessed on the server under the federated learning paradigm, conflicting with the requirements of the centralized unlearning process. Additionally, there are high computation and communication costs associated with accessing clients' data, especially in scenarios involving numerous clients or complex global models. To address these concerns, we propose a more effective and efficient federated unlearning scheme based on the concept of model explanation [1]. Model explanation involves understanding deep networks and individual channel importance, so that this understanding can be used to determine which model channels are critical

[1]The main content of this chapter has been published in Heng Xu, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, S Yu Philip, Update Selective Parameters: Federated Machine Unlearning Based on Model Explanation. IEEE Transactions on Big Data, 2024.

for classes that need to be unlearned. We select the most influential channels within an already-trained model for the data that need to be unlearned and fine-tune only influential channels to remove the contribution made by those data. In this way, we can simultaneously avoid huge consumption costs and ensure that the unlearned model maintains good performance. Experiments with different training models on various datasets demonstrate the effectiveness of the proposed approach.

3.1 Introduction

In this Section, we consider the problem of unlearning in the setting of federated learning, in which the server wants to remove all samples associated with a particular class. There are many scenarios where data from a trained model that needs to be unlearned belongs to one or more classes [9]. For example, if the training process of federated learning contains a group of outdated data or data identified as adversarial type, the server needs to remove all effects of these data from the global model. Another example is thats the recommendation system needs to periodically remove some items (classes) due to relevant legal updates or product iterations. Considering this setting, the above-mentioned centralized unlearning methods are inefficient due to the following challenges.

For the data reorganization methods, most centralized unlearning algorithms make use of the original training dataset to remove the effect of unlearning data [11, 15, 19, 24]. Those kinds of unlearning methods usually aggregate all training data in one node and split those data into different subsets. Then, each subset will be used to train the sub-model, which limits the impact of data in the subset to the corresponding sub-models rather than a more complex model with the entire training data. In the federated learning context, however, the original training dataset may not be accessible on the server, meaning that the aforementioned approaches will be unsuitable for federated learning. For parameter manipulation-based schemes, those methods always focus on simple machine learning problems, such as tree-based models [12, 100]; notably, the complexity of global models and the stochastic nature of the local learning process make it impossible to use parameter manipulation-based methods in federated learning. Moreover, due to the complexity associated with calculating the impact of the specified sample, manipulating a model's parameters based on unreliable impact results or assumptions will significantly affect the model's performance [41, 46].

For existing class-level unlearning in a federated learning setting, Wang et al. introduced the concept of TF-IDF [113] to quantify the class discrimination of the channels

and accordingly proposed an unlearning scheme based on the model pruning technique. However, the pruning-based scheme not only removes the information of the unlearning classes but also erases the information of some remaining data at the same time, which will lead to the degradation of the model performance for the remaining data. Other unlearning schemes usually use the historical parameter updates cached during the global training process. However, those cached gradients will not be updated with the unlearning process, which precludes carrying out subsequent unlearning processes. This solution may also further increase the risk to users' data since recent studies show that attackers can reconstruct the training dataset by referring to the gradients [57]. Finally, compared with traditional centralized learning, federated learning has more randomness involved. For example, the selected users involved in each global iteration during the training process make the federated learning non-deterministic and difficult to control. Even if retraining the model from scratch causes the global model to converge to different local minima at different times. Therefore, it becomes impractical and more difficult to unlearn data by directly manipulating the parameters in a federated learning setting [84].

To tackle the above issues, we consider that the purpose of unlearning is to remove the effect of certain specific samples from the model, not all training datasets. Thus, it is crucial to find the partial parameters that most affect these samples. Model explanation refers to a series of methodologies that can help to explain the importance of particular parameters [75, 76]. It can be used to find some channels within a trained model that have the greatest effect on the performance of the samples that need to be unlearned.

Based on this observation, we first calculate the influence of each channel based on the ablation study, then select some channels that most affect the classification performance of unlearning class. Subsequently, we propose two different unlearning schemes based on the knowledge owned by the server. The first is decentralized unlearning, which is based on the original federated learning paradigm and achieves the unlearning purpose by only updating partial parameters within selected important channels. When the server owns some samples with the same distribution of training data, we prefer the second unlearning scheme, centralized unlearning. In this case, we put the unlearning operation on the server side with the help of only a few training samples rather than the whole training dataset contained in all clients. Centralized unlearning also only fine-tunes partial parameters within a model to unlearn data. In addition, to speed up the unlearning process, we use perturbation samples in both schemes to efficiently muddy the model's understanding of the unlearning data. Those perturbation samples are

generated from unlearning data with randomly selected incorrect labels. Both schemes can avoid large communication and computation expenses and simultaneously maintain the performance of the unlearned model for the remaining data.

The main contributions of this chapter can be summarized as follows:

- First, we introduce the ablation study to calculate the influence of each channel with respect to the unlearning class, which enables us to get the most influential channels for the unlearning class within any trained model.
- Second, we provide two different schemes based on the knowledge owned by the server, decentralized unlearning and centralized unlearning. Both schemes only update partial influential channels' parameters to avoid large communication, computation and storage expenses.
- Third, we propose an effective unlearning method based on perturbed samples, which can effectively accelerate the process of unlearning while ensuring the usability of the unlearned model.
- Further analysis and experiments comparing the performance of our proposed approach with that of recent works under different models and multiple datasets demonstrate the effectiveness of the proposed federated unlearning scheme.

3.2 Problem Statement

3.2.1 Federated Learning Background

Federated learning is a decentralized machine learning framework, the key components of which are a server S and a group of clients $U = \{u_1, u_2, ..., u_n\}$, where n is the number of clients, as shown in Figure 3.1. The server relies on these clients to collaboratively complete the training of a global model M, which will be provided to all clients to use after the training process. The model's training process will repeat the steps outlined below until the model satisfies certain conditions. At the i-th training round ($i \in E$, where E is the number of global training rounds), the updating process of the global model M is performed as follows:

1. **Initialization:** The server first selects a certain percentage of clients to participate in this round of model training due to the unstable characteristics of clients in the federated learning framework. Subsequently, the selected federated clients

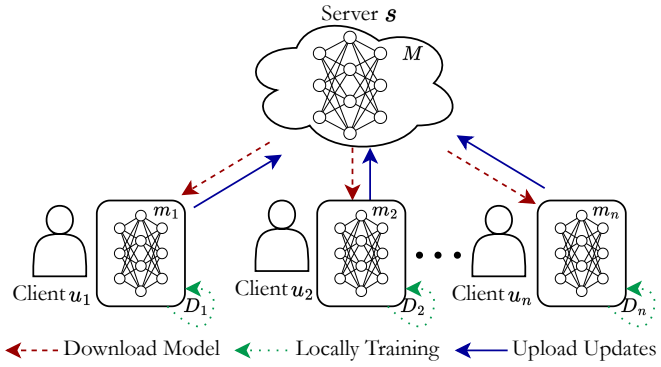


Figure 3.1: A Standard Framework of Federated Learning.

download the current global model M_i and the training configurations from the server.

2. **Local Training and Uploading:** Each selected client u_k trains the local model $m_k = M_i$, based on its own local dataset D_k and training configurations for E_{local} epochs, then computes an update with respect to m_k, ($k \in \{1,2,...,n\}$), where E_{local} is the number of local epochs for each client. When the local training process is complete, each client will upload its model updates to the server.

3. **Server Aggregation:** Once the server has collected all updates from all participating clients, it will update the global model M_i based on one aggregation rule, such as FedAvg [86], and the collected updates to obtain a new global model M_{i+1}. M_{i+1} will play the role of M_i for the next training round.

When the global model converges, the server halts the above training process and obtains the final federated learning model M^*.

3.2.2 Examples: Federated Unlearning

Designing an effective and efficient unlearning scheme for a federated learning framework is difficult. First, the incremental learning process causes the global model to update depending on all previous updates. If an update from one client is removed from the global model's aggregation process in round i, the global model will be changed after the aggregation process; all the client's subsequent updates will then become useless, as all clients compute local updates based on the global model. Second, randomness exists in federated learning training due to the stochastic training process; this impacts which clients will be participating in this round's training process, as well as what samples will

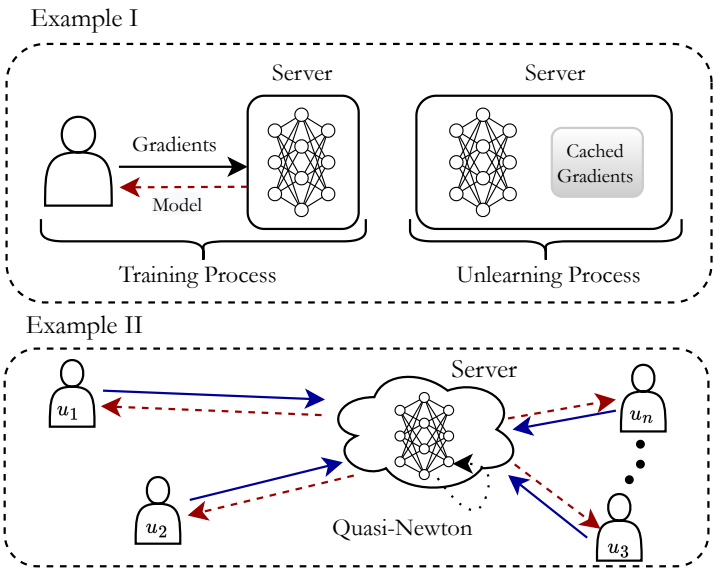


Figure 3.2: Examples of Federated Unlearning.

be used for training in a certain epoch or batch. This randomness makes the federated learning process non-deterministic and difficult to control.

To further illustrate the difficulties in federated learning, we present here two examples of existing unlearning schemes. Figure 3.2 illustrates two schemes commonly used in federated unlearning. In example I, the server utilizes cached historical gradients from clients in the training process to approximate the gradients in the unlearning process. Notably, the effectiveness of these schemes will decrease dramatically as the number of unlearning requests increases; this is because the gradients are cached during the training phase, and the unlearning process will not update these gradients to satisfy subsequent unlearning requests [81]. Second, the clients involved in each federated learning training process are randomly selected. The gradients cached during the training period may not be used in the unlearning process since the relevant clients may be different and the gradients are also not related to each other. Some recent studies have even shown that attackers can use the gradient of model updates to reconstruct clients' data; therefore, these solutions may increase the risk of data leakage from clients in federated learning. Example II introduces the parameter manipulation-based unlearning scheme from centralized learning to federated learning. Those methods usually use Quasi-Newton methods to update global models instead of the original model retraining process. Quasi-Newton technique is an optimization algorithm designed for unconstrained numerical optimization problems. It approximates the inverse Hessian matrix iteratively, making it suitable for large-scale optimization where direct compu-

tation of the Hessian is impractical. Nonetheless, similar to centralized learning, this method is only applicable to simple models with minimal data erasure.

The two examples above illustrate that it is impractical to achieve unlearning based on cached gradients or Quasi-Newton; if either of these schemes is used for unlearning data, it will reduce the performance of the unlearned model for the remaining data.

3.2.3 Federated Unlearning Formalization

We consider federated unlearning in a supervised learning setting, i.e., one in which all clients collaboratively train a supervised model under the control of the server. Each client's data belongs to the instance space $\mathscr{X} \subseteq \mathbb{R}^d$, with the label space defined as $\mathscr{Y} \subseteq \mathbb{R}$. We consider the local data is Independent and Identically Distributed (IID) and use $\mathscr{D}_k = \{(\mathbf{x}_{k,1}, y_{k,1}), ..., (\mathbf{x}_{k,t}, y_{k,t})\} \subseteq \mathbb{R}^d \times \mathbb{R}$ to represent a local training dataset in client u_k, in which each sample $\mathbf{x}_{k,t} \in \mathscr{X}$ is a d-dimensional vector, $y_{k,t} \in \mathscr{Y}$ is the corresponding label, and t is the size of $\mathscr{D}_k$. To unlearn data, all clients first need to perform a *data deletion* operation locally.

Definition 8. *(Data Deletion). Data deletion performed by each client is an operation that deletes a group of client data that needs to be unlearned from the global model, denoted as unlearning data $\mathscr{D}_k^u$. For each client $u_k \in U$, given its local dataset $\mathscr{D}_k$ and unlearning data $\mathscr{D}_k^u$, data deletion is defined as follows:*

$$\mathscr{D}_k^r = \mathscr{R}(\mathscr{D}_k, \mathscr{D}_k^u) \tag{3.1}$$

where the operation $\mathscr{R}$ denotes deleting data $\mathscr{D}_k^u$ from $\mathscr{D}_k$. For example, a client deletes data samples $\mathscr{D}_k^u$ from the local dataset $\mathscr{D}_k$ according to the designed data deletion operation in order to obtain the remaining dataset $\mathscr{D}_k^r$. The purpose of data deletion is to remove unlearning data from the training set in all clients so that the global model will not be affected by those data again in the next collaborative training process. After data deletion, the server will perform the *federated unlearning* operations.

Definition 9. *(Federated unlearning) Consider a set of data that we wish to remove from the global model, denoted as $\mathscr{D}_u = \{\mathscr{D}_1^u, \mathscr{D}_2^u, ..., \mathscr{D}_n^u\}$. Let M^* denote the global model produced by the federated learning, while M^u denotes the unlearned model that we want to achieve. We define federated unlearning $\mathscr{U}(\cdot)$ as follows:*

$$M^u \equiv \mathscr{U}(M^*, \mathscr{D}_u) \tag{3.2}$$

Table 3.1: Notations

Notations	Explanation
u_k	One client
D_k	The dataset in u_k
$(\mathbf{x}_{k,t}, y_{k,t})$	One sample in D_k
$\mathscr{R}$	The data deletion operation
D_k^u	The unlearning dataset in u_k
D_k^r	The remaining dataset in u_k
$\mathscr{U}$	The federated unlearning process
M^*	The learned federated learning model
M^u	The unlearned model
$\mathscr{V}$	The unlearning operation
y^u	The unlearning class label
$S_{w_{l,i}}$	The effect of channel $w_{l,i}$ in layer l
$H(M,D)$	The model accuracy of dataset D
$\mathscr{T}$	The important channels

Federated unlearning aims to remove the contribution about the unlearning data from the global model. After the unlearning process is complete, a verification function $\mathscr{V}(\cdot)$ can be used to measure whether model M^u has successfully unlearned the requested unlearning data.

Definition 10. *(Verification function) After the unlearning process is complete, a verification function $\mathscr{V}(\cdot)$ can perform a distinguishable check,Äîthat is, $\mathscr{V}(M^u) \neq \mathscr{V}(M^*)$,Äîto illustrate the results of the unlearning operation.*

As an attack method, membership inference attacks can also be used to determine whether a sample's contribution has been successfully removed from the model [107].

3.2.4 Design Goals

We provide a federated unlearning scheme based on the concept of model explanation, which can achieve federated unlearning quickly while effectively ensuring the performance of the unlearned global model. The desired goals of our proposed unlearning scheme are as follows:

- **Effective**: Similar to retraining from scratch on the remaining data, our scheme should ensure that the unlearned model does not contain any information about

the unlearning data (i.e., that attackers cannot infer information about unlearning data through the unlearned model using existing privacy attack methods, such as membership inference attack [107]).

- **Efficient**: Regardless of how much data needs to be unlearned, our scheme should be more efficient than retraining from scratch.
- **Accuracy**: Even though unlearning operations will cause a reduction in model performance, our scheme should not unduly reduce performance through unreasonable manipulation.
- **Security**: It should not introduce new privacy risks and at least be consistent with the original guarantee. Specifically, it should ensure that any attacker, including the server, cannot easily utilize gradient information to implement gradient leakage attacks.
- **Non-intrusive**: Our scheme should be non-intrusive for both server and clients, and should be able to serve as an opt-in component inside existing federated learning systems; that is to say, the operation of unlearning cannot affect the current structure of the federated learning system.

It is worth noting that security goal emphasizes that the designed unlearning scheme cannot introduce potential threats that additionally threaten the user's privacy, while effective goal underscores that no one can recover unlearned data from the model after unlearning process.

3.3 Federated Unlearning Methodology

3.3.1 Overview

In this section, we first present an overview of our proposed scheme, then provide a detailed outline. For ease of reference, the important notations that appear in this chapter and their corresponding descriptions are listed in TABLE 3.1.

In this chapter, we focus on unlearning requests relating to a specific class,Äîthat is, requests designed to remove an entire class of samples from the global model, which is frequently required in unlearning problems [44, 120]. Figure 3.3 illustrates the workflow of the proposed unlearning method, which mainly comprises five steps. We will discuss the first two steps in the local data update section, after which ablation calculation and

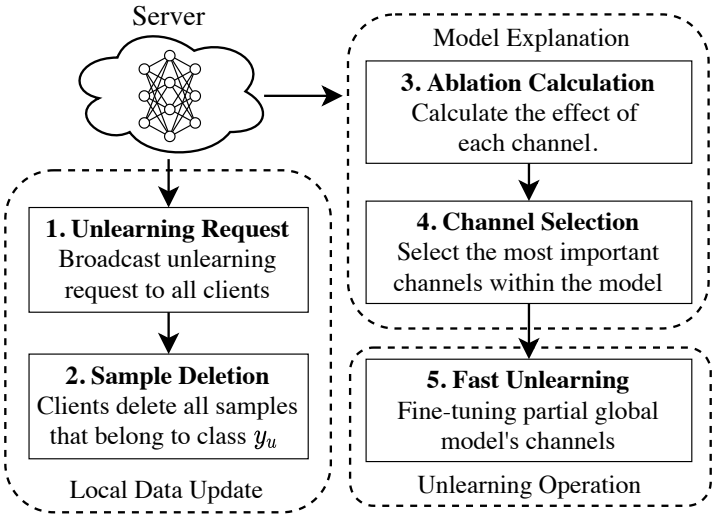


Figure 3.3: Overview and Workflow of Our Federated Unlearning Scheme.

channel selection will be discussed in the model explanation section. The unlearning schemes are presented in the unlearning process section, which contains two different sub-schemes for different application settings.

Our key insight is that each channel of a model contributes differently to the overall classification performance. This means that some channels are universally important for the task, while others are selectively important for specific classes. Based on this observation, we use the ablation study to find the most influential channels that have the greatest effect on the unlearning class. After that, we fine-tune those influential channels to remove the contribution of unlearning data from the global model. Since the number of parameters in influential channels is much less compared to the number of all model parameters, the unlearning operation could avoid extensive communication between clients and a server. At the same time, because fewer parameters need to be updated, the unlearning operation will be more efficient than retraining all parameters. We also establish a benchmark to control how many influential channels need to be fine-tuned. Broadly speaking, the fewer channels are fine-tuned, the more efficient unlearning occurs.

3.3.2 Local Data Update

We assume that in the initialization stage of the federated learning, the server aims to train a deep neural network model with L layers. The parameters in each layer can be represented as $\mathbf{w} = \{w_1, \ldots, w_L\}$. If layer l is a convolutional layer, the parameters w_l in

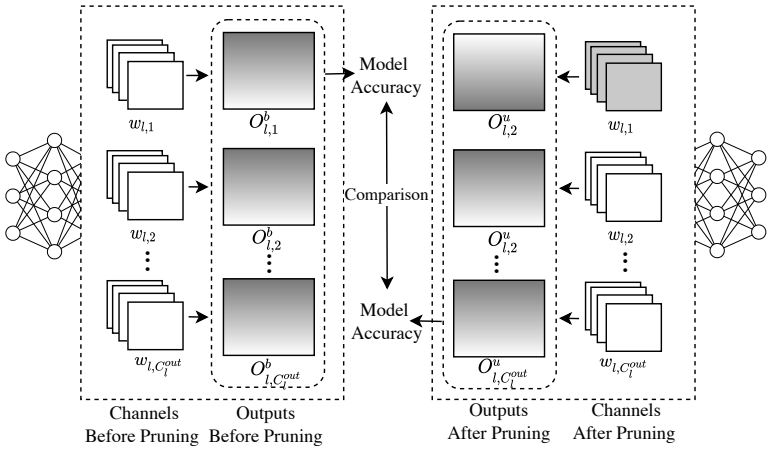


Figure 3.4: Ablation Illustration.

the l-th layer is denoted as $\mathscr{R}^{C_l^{out} \times C_l^{in} \times K_l \times K_l}$, where C_l^{out}, C_l^{in} and K_l denote the number of output channels, number of input channels, and the kernel size, respectively. If layer l is a linear layer, the parameters w_l can be represented as $\mathscr{R}^{C_l^{out} \times C_l^{in}}$.

During the model training or at the inference stage, the server can broadcast unlearning requests to all clients. First, when the server gets an unlearning request, it will generate the unlearning requests $(u_k, y^u), u_k \in U$ and send those pairs to each corresponding client. After receiving the unlearning request from the server, all clients should remove the samples that need to be unlearned from D_k to prevent these from being re-included in subsequent stages of the training process. That is, $D_k^r = \mathscr{R}(D_k, (\mathbf{x}_i, y^u))$, for $\forall (\mathbf{x}_i, y^u) \in D_k$, where y^u denotes the class label to be unlearned, and $\mathbf{x}_i$ represents a sample that belongs to y^u.

3.3.3 Model Explanation

Machine learning models usually consist of multiple different units, and each of those units provides various contributions to the model's performance. Different from previous methods [93, 138], which usually use the ablation study to measure the contributions of those units to the overall performance, we introduce ablation study to analyze the channel influence for the performance of unlearning class and define this influence as the *Effect of Channel*.

Definition 11. *(Effect of Channel) Given a global model M^* that takes a group of data I and outputs an overall accuracy $H(M^*, I)$. For a known baseline $H_b(M^*, I)$ that achieved before pruning operation, the influence $S_{w_{l,i}}$ of one channel $w_{l,i}$ in layer l toward data I*

Algorithm 1: Influential Channel Selection

Input: global model M^*, selection factor: δ
Output: important channels in model M^*: $\mathscr{T}$

1 Initialize effect of channel $\mathscr{S}$ as an empty $\emptyset$
2 Initialize $\mathscr{T}$ as an empty directory $\emptyset$
3 Construct a group of unlearning data I
4 Feed I to model M^* and record the original model accuracy $H_b(M^*, I)$
5 $\mathscr{E} \leftarrow$ all *layers* in model M^*
6 **for** *each layer e in* $\mathscr{E}$ **do**
7 $w_l \leftarrow$ all *channels* in e
8 **for** *each channel* $w_{l,i}$ *in* w_l **do**
9 Prune the channel $w_{l,i}$ to get $M^{'}$
10 Feed I and record accuracy $H(M^{'}, I)$
11 $S_{e_{l,i}} = H_b(M^*, I) - H(M^{'}, I)$
12 Sort S_{e_l} in the order of $S_{e_{l,i}}$ from small to large
13 $\mathscr{T}_e \leftarrow$ select the top δ channels
14 **return** $\mathscr{T}$

is the change of the accuracy after pruning parameter in $w_{l,i}$ *to obtain* $M^{'}$*:*

$$S_{w_{l,i}} = H_b(M^*, I) - H(M^{'}, I) \tag{3.3}$$

Figure 3.4 illustrates one example of the calculation of the effect of a single channel within one convolutional layer l. We use I to denote the input of the model and I_l to represent the input of the l-th layer. Given the parameters w_l in layer l, the output of the l-th layer in an original trained model can be represented as $O_l^b = I_l \otimes w_l$, where $\otimes$ is the convolutional operation. When pruning one specific channel $w_{l,1}$ in layer l, $w_l^{'} = \mathscr{P}(w_l, w_{l,1})$, where $\mathscr{P}$ is the pruning operation, the output will be changed to $O_l^u = I_l \otimes w_l^{'}$ and $O_l^b \neq O_l^u$. This difference of the single layer will be passed forward and will be reflected in the final accuracy $H(M^{'}, I)$. We calculate the effect according to the corresponding pruned channel based on the change of the accuracy $S_{w_{l,i}} = H_b(M^*, I) - H(M^{'}, I)$. After achieving the effect of all channels, the server then selects the channels within a model that most affect the performance of the unlearning class, i.e., the channels with high effect values. We select these channels and consider them to be the most influential channels for an unlearning class.

Algorithm 1 provides the selection of important channels within a global model. In the first step (Lines 3-4), the server feeds a group of unlearning data and records the original model accuracy. Then, in lines 5,Äì11, the server calculates the effect for all channels. Line 6 indexes each layer in the global model, while line 8 indexes each channel

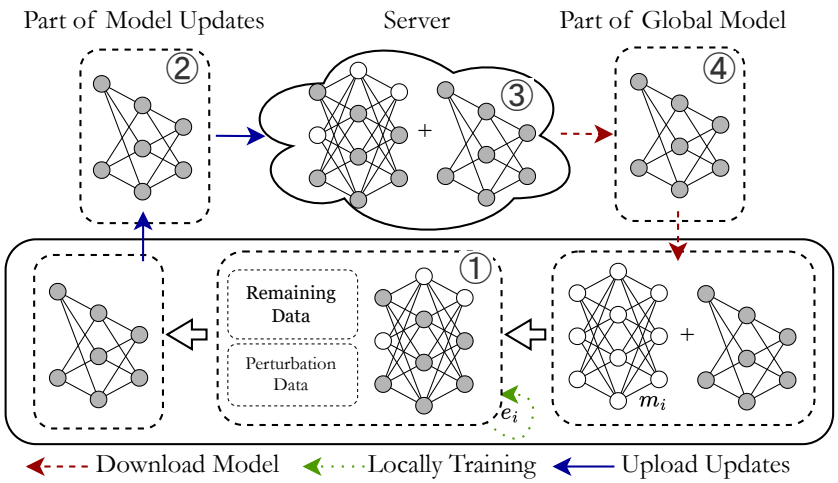


Figure 3.5: Decentralized Unlearning.

in one layer. After pruning a particular channel, the model will recalculate the accuracy of I to get the effect of this channel (lines 9-11). A high effect value indicates that the corresponding channel plays an influential role in model performance with respect to I, while a low effect value means that this channel may be useless for I only or for all training data. In lines 12-13, the server selects the most influential channels based on the effect value S_{e_l} in layer l. δ is a hyperparameter used to control the proportion of selected channels. It is important to note that the above steps can be performed at any time during the inference phase; thus, the time consumption of the unlearning process is not increased.

3.3.4 Unlearning Process

Considering different application scenarios, we provide two schemes to fine-tune the global model. The first is to use the traditional federated learning architecture to complete the unlearning process when clients and all training data are available. When clients and training data are unstable, we employ server-based unlearning without any communicating with clients. Both schemes only update partial channels.

3.3.4.1 Decentralized Unlearning

If clients and those training data are available, the server will perform decentralized unlearning processes after selecting the important channels. As shown in Figure 3.5, there are three main differences between decentralized unlearning and federated learning. First, for the local training process, each client will only train the most influential

channels based on two types of data (① in Figure 3.5). One is the remaining data, and the other is perturbation data, generated from unlearning data by modifying labels to an arbitrary value. Perturbation data can quickly obfuscate the model's understanding of the unlearning data, which will speed up the process of unlearning process. Second, in decentralized unlearning, each client only uploads influential channels' updates to the server for aggregation (② in Figure 3.5), while it will upload all channels' updates in federated learning. Third, in each step of aggregation, the server only updates partial channels based on the partial updates from clients and broadcasts the partial global model to each client for subsequent model training and updates (③ and ④ in Figure 3.5). In federated learning, the server updates all channels and broadcasts the whole model to each client. It is worth noting that even with the partial upload of the updates, the pattern of our unlearning scheme is the same as that of federated learning. The unlearning process can be done exactly based on the original federated learning paradigm.

Algorithm 2 illustrates the process of decentralized unlearning. First, the server initializes the global model M_u^0 with the original global model M^* and broadcasts part of the global model to all clients (lines 3-4). Then, for each round of the unlearning process, each client downloads and updates the local model (line 10). During each model training step, the unimportant channels $M \backslash \mathcal{T}$ are fixed, and only the important channels $\mathcal{T}$ are trained based on two types of data, remaining and perturbation data (lines 11-14). After the local training process of a client, the client will upload only part of the model updates to the server (line 15). When the server receives all partial updates from all clients, it will aggregate them and then broadcast a new partial global model to all clients for the next training (lines 6-8). When the global model converges, M^u is the model that has erased the information of the unlearning data.

3.3.4.2 Centralized Unlearning

A fewer number of model channels to update usually means less training data is required. Therefore, we also provide a scheme that requires a small amount of training data on the server side to achieve the unlearning purpose, centralized unlearning. Our centralized unlearning process is similar to the original training procedure for each federated learning client. In particular, we fix non-influential channels within the model $M^* \backslash \mathcal{T}$. Then, in each model training process, we only update the important channels $\mathcal{T}$ based on remaining and perturbation data to reduce the computational cost. In addition, the process is only executed on the server without needing any client participation, which reduces communication and computation costs.

Algorithm 2: Decentralized Unlearning

Input: important channels $\mathscr{T}$, original global model M^*
Output: unlearned model M^u

Server broadcast important channels $\mathscr{T}$ to all clients
Server executes:
  initialize $M_u^0 \leftarrow M^*$
  broadcast part of global model $M_{u,p}^0$ to all clients
  for *t from* 1 *to* E_{global} **do**
    for *each client* u_k *in* U **do**
      $\mathscr{R}^k \leftarrow Clientupdate(k, M_{u,p}^t)$
    $M_{u,p}^{t+1} \leftarrow M_{u,p}^t + \sum_{k=1}^{n} \frac{1}{n} \mathscr{R}^k$
Clientupdate$(k, M_{u,p}^t)$**:** //Run one client k
  Update the local model M^t based on $M_{u,p}^t$
  for *each local epoch e from* 1 *to* E_{local} **do**
    for *each batch b in* D_k **do**
      fix the channels in $M^t \backslash \mathscr{T}$
      only update channels $\mathscr{T}$ based on two types of data
upload the updates of $\mathscr{R}^k$ of channels $\mathscr{T}$ to server

3.3.5 Case Study

To further illustrate our unlearning method, we present a case study in this section. We assume that there are ten clients $U = \{u_1, u_2, \ldots, u_{10}\}$ training a federated learning model, ResNet20, each of which has certain data from dataset CIFAR-10, and denoted as $\mathscr{D}_k$. We assume that during the model training or inference stage, the server needs to remove all contributions related to class 1 from the model. In unlearning request stage, the server generates unlearning request $(u_k, 1), u_k \in U$ and sends it to the corresponding client. Each client will remove all samples from the local training data after receiving the unlearning request.

At the same time, the server first selects the channels of the current model that most affect the classification performance of class 1. To do this, the server constructs a small batch of data I consisting of samples from class 1, and feeds it into the model, recording original model accuracy $H_b(M^*, I)$. Then, given one layer l, the server prunes one channel $w_{l,i}$ at a time and retests the accuracy $H(M^{'}, I)$ of the above data I. Based on two accuracy results, the server will calculate the effective value of $w_{l,i}$, $S_{w_{l,i}} = H_b(M, I) - H(M^{'}, I)$. This process is executed consistently until all channels in layer l are indexed. After that, the top δ channels with the higher effective value are selected; these will be denoted as the most influential channels in this layer. The same process is then performed for the

other layers.

After the above steps, the server will find the channels within a model that exert the greatest influence over the unlearning class 1. The server will then select the unlearning process based on the availability of clients and those data. If the clients and those training data are available, the server will execute the decentralized unlearning process, where the local model update of each client as well as the uploaded update of each client only consider the most influential channels. If those clients or those data are unavailable, the server will execute the centralized unlearning process. In each step, the server holds some channels as fixed and only updates the influential channels to complete the unlearning. Our unlearning scheme is mainly based on the use of model explanation to select and fine-tune the most influential channels rather than approximating the unlearned model based on the gradients cached during the training phase. As a result, we can effectively avoid the problems discussed in Section 3.2.2. In addition, our schemes only update partial channels with the help of the perturbation data, which can significantly accelerate the unlearning process.

In addition, our solution does not introduce any security risks and is non-intrusive. Unlearning requests can be sent through encrypted channels; even if such a request is obtained by a malicious attacker, this will not cause serious privacy concerns since the information contained in the request is less than that contained in the gradient. Furthermore, our method only uploads partial model channels during the decentralized unlearning or does not upload any gradient information in the centralized unlearning process, which can reduce the security risks introduced by gradients. Moreover, our scheme is non-intrusive and does not affect the traditional federated learning structure. To further improve the security of the training process, our scheme is also compatible with other schemes, such as differential privacy [142]. Therefore, our scheme does not decrease the security of traditional federated learning, or is at least consistent with the security of traditional federated learning, and can serve as an opt-in component within existing systems.

3.4 Performance Analysis

In this section, we analyze the performance of the proposed unlearning scheme in terms of computational, communication, and storage overheads. We term the decentralized unlearning as Our$_{DE}$ and denote the centralized unlearning scheme as Our$_{CE}$. Table 3.2 shows the comparison results.

Table 3.2: Comparison Results Between Our Scheme and Retraining From Scratch (denoted as RFS and we set $\delta < 1$)

	Computation Overhead	Communication Overhead	Storage Overhead
RFS	$n*(n^3*6f(\cdot)+g(\cdot))$	$2*n^2*c(\cdot)$	$\frac{\lvert Y\rvert-1}{\lvert Y\rvert}*s(\cdot)$
Our_{DE}	$n*(n^3*f(\cdot)*(1+5*\delta)+g(\cdot))$	$2*n^2*c(\cdot)*\delta$	$\frac{\lvert Y\rvert-1}{\lvert Y\rvert}*s(\cdot)$
Our_{CE}	$n^2*f(\cdot)*(1+5*\delta)$	0	$0.05*s(\cdot)$

3.4.1 Computational Overhead

The model training process is the most time-consuming element of a federated learning system. In the interests of simplicity, we set $f(\cdot)$ to represent the computational cost of the forward propagation, then the computational cost of one-step backpropagation is at most $5f(\cdot)$ [84]. Since the server also uses one aggregation algorithm during the training process to aggregate updates and update the global model, we use $g(\cdot)$ to denote the computational cost of the aggregation operation.

For retraining from scratch, during the forward and backpropagation operation in the model updating for one batch, all channels will be involved, which costs $6f(\cdot)$. Moreover, it requires a group of users to participate in the unlearning process; this one batch model update should be assigned to all users and requires an additional aggregation operation. Therefore, the total computational cost of this scheme is $E_{global}*(\mathcal{N}_{user}*E_{local}*\mathcal{N}_{batch}*(6f(\cdot))+g(\cdot))$. For the sake of convenience, we rewrite it as $n*(n^3*6f(\cdot)+g(\cdot))$. For Our_{CE}, during the forward propagation operation in one batch model update, all model channels are also involved in the calculation until the predicted value is output in the last layer, which costs $f(\cdot)$ units of computation. In the backpropagation operation, since we only focus on updating part of the channels, the cost will be $\delta*5f(\cdot)$ units of computation, where δ is the proportion of important channels and $\delta < 1$. In addition, since neither requires client-server interaction in the unlearning process, there is no computation consumed by the aggregation operations. Finally, the total computational cost of our unlearning scheme Our_{CE} is $n^2*f(\cdot)*(1+5*\delta)$. For our scheme Our_{DE}, since we also only consider the update of important channels, so the total computational cost is $n*(n^3*f(\cdot)*(1+5*\delta)+g(\cdot))$.

Therefore, compared with retraining from scratch, our decentralized unlearning scheme Our_{DE}, can achieve $\frac{n*(n^3*6f(\cdot)+g(\cdot))}{n*(n^3*f(\cdot)*(1+5*\delta)+g(\cdot))}\times$ unlearning improvement. For centralized unlearning scheme Our_{CE}, this ratio can be up to $\frac{n*(n^3*6f(\cdot)+g(\cdot))}{n^2*f(\cdot)*(1+5*\delta)}\times$. Assuming that the computation cost for aggregation operation $g(\cdot)$ is negligible and all clients calculate one batch model update in a parallel paradigm, both schemes can provide a speed-up

of $\frac{6}{1+5*\delta}\times$ and $\frac{6*n^2}{(1+5*\delta)}\times$, respectively. In addition, since we keep the information from previous training rather than retraining from scratch and using the perturbation data to accelerate the unlearning process, we can further improve our computational efficiency, e.g., the global epoch will be less than retraining from scratch. In summary, it is shown that our unlearning scheme significantly reduces the computational overheads compared with retraining from scratch.

3.4.2 Communication Overhead

There are normally two entities in an unlearning scheme, namely, the clients and the server. Without loss of generality, we evaluate communication efficiency through the number of client-server interactions. Specifically, we set $c(\cdot)$ to represent one-time interaction between a client and server.

For retraining from scratch, it is necessary to interact with clients to complete the unlearning process. During each unlearning iteration, the server first broadcasts the current global unlearning model to each client, after which all clients will upload the unlearning information to the server; thus, retraining from scratch will spend $E*\mathcal{N}*(2*c(\cdot))$, where E and $\mathcal{N}$ are the number of unlearning global epochs and clients that participate in the unlearning process, respectively. For convenience, we also simplify this to $2*n^2*c(\cdot)$. For our decentralized unlearning scheme Our_{DE}, since each client only sends updates about important channels, it will cost $2*n^2*c(\cdot)*\delta$. For centralized unlearning scheme Our_{CE}, it places most of the computation on the server side, which precludes the need for interaction; thus, it almost costs 0.

Therefore, compared with retraining from scratch, our decentralized unlearning scheme can save $\frac{1}{\delta}\times$ in terms of communication costs. For Our_{CE}, since it does not communicate with the server, it has almost no consumption due to communication. Hence, our unlearning scheme is more efficient than retraining from scratch.

3.4.3 Storage Overhead

Unlearning methods are usually based on the participation of the remaining data. We consider the storage cost based on the size of the remaining data. Specifically, we use $s(\cdot)$ to denote the size of the original training data and use the $|Y|$ to represent the number of classes in training data.

For retraining from scratch and our decentralized unlearning scheme Our_{DE}, since they need all remaining data of all clients to finish the unlearning process, it will cost

$\frac{|Y|-1}{|Y|} * s(\cdot)$ training samples. For our centralized unlearning scheme Our$_{CE}$, since we put the unlearning process on the server side and only update partial channels' parameters, it only needs a small group of the remaining data. In our experimental results, for the CIFAR-10 dataset, when unlearning class 1 from the model ResNet20, it only needs less than $0.05 * s(\cdot)$ size of original data. Therefore, compared with retraining from scratch, our scheme Our$_{CE}$ largely reduces storage overheads.

Summary: From the above analysis, the proposed unlearning schemes are determined to be efficient in terms of computational, communication, and storage costs compared with retraining from scratch, which demonstrates the practical potential and significant performance improvements obtained by our unlearning schemes.

3.5 Experiments

3.5.1 Experimental Setup

3.5.1.1 Dataset and Model

During the federated learning initialization phase, the server needs to broadcast the model to each client. Here, we choose the two most powerful and popular existing models to evaluate our unlearning, namely VGG [108] and ResNet [52]. We also adopt three real-world image datasets for evaluation: MNIST [2], Fashion MNIST [3], CIFAR-10 and CIFAR-100 [4]. The datasets cover different attributes, dimensions, and numbers of categories, allowing us to explore the unlearning utility of the proposed algorithm effectively.

3.5.1.2 Evaluation Metrics

Here, we introduce the following criteria to evaluate the efficiency and effectiveness of our unlearning scheme:

- **Training Rounds**: The number of training rounds indicates whether the model can complete the unlearning as quickly as possible. On the premise that the same unlearning effect is achieved, fewer training rounds can show that the scheme is more efficient.

[2]http://yann.lecun.com/exdb/mnist/
[3]http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/
[4]https://www.cs.toronto.edu/ kriz/cifar.html

- **Accuracy-based**: Models that have been trained usually have high prediction accuracy for trained samples, which means that the unlearning process can be verified by the accuracy of the model output. Ideally, for data that needs to be unlearned, the accuracy should be the same as a model trained without seeing the unlearning data. This could be infinitely close to zero. In addition, for the remaining data, the accuracy should almost keep unchanged.

- **Attack-based**: The basic purpose of unlearning is to reduce the risk of sensitive information leakage. Therefore, certain attack methods can be used to directly and effectively verify the success of unlearning operations. Here, we use model inversion attack [38] and membership inference attacks (MIAs)[133] to evaluate our scheme. Ideally, before the unlearning, model inversion attack can effectively recover the unlearned data, while after the unlearning, it cannot recover the unlearned data. For MIAs, we use the success rate of attacking unlearning samples as our metric, that is, recall = $\frac{TP}{TP+FN}$, where TP denotes the number of unlearning samples predicted to be in the training set and $TP+FN$ represents the total unlearning samples. Ideally, recall on these unlearning samples should be close to 100% before unlearning. After unlearning, recall should be close to 0%.

- **GradAttack**: In order to evaluate whether updating partial channel parameters can mitigate gradient-related risks. We use GradAttack [57] to evaluate the training process of retraining from scratch and our decentralized unlearning, respectively. Ideally, during the retraining from scratch, GradAttack can effectively recover relevant information based on gradients, whereas for our unlearning scheme, it should be unable to recover that information.

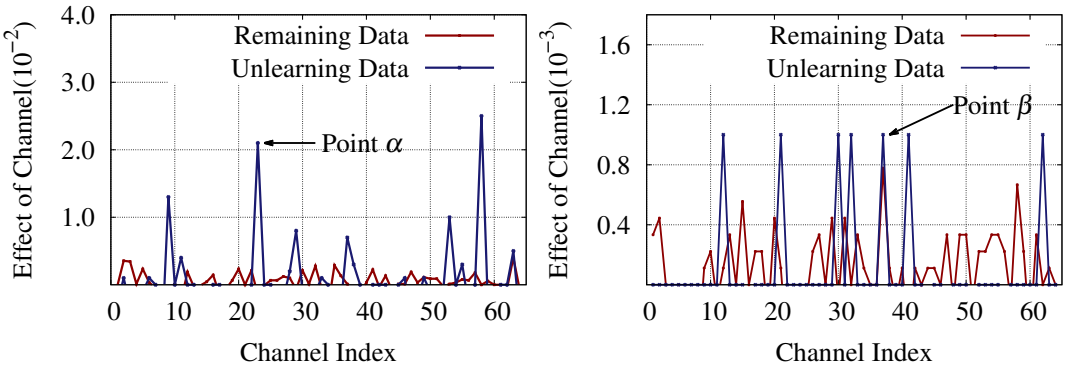
It is worth noting that the purpose of using GradAttack differs from above attack-based metrics. It is not used to demonstrate the effectiveness of our unlearning scheme but, rather, to illustrate that our partial parameter updating can, to some extent, mitigate attacks based on gradient information, which was not considered in exiting unlearning solutions.

3.5.1.3 Comparison Methods

In our experiments, we respectively evaluate our decentralized unlearning (**DE Importance**) and centralized unlearning (**CE Importance**) schemes. To demonstrate the effectiveness of selecting the important channels, we also evaluate the validity of randomly

Table 3.3: The Model Accuracy after Pruning Different Types of Parameters.

	block_2.conv_bn2.conv in ResNet20		*block_2.conv_bn1.conv* in ResNet20		*feature.conv_7* in VGG11	
	Remaining	Unlearning	Remaining	Unlearning	Remaining	Unlearning
Original Model	80.48%	87.10%	80.48%	87.10%	75.90%	94.60%
Random	69.48%	74.60%	67.30%	83.40%	72.95%	79.90%
Non-Important	68.34%	89.45%	46.37%	85.56%	62.02%	93.20%
Important	70.62%	38.50%	65.14%	2.40%	72.21%	7.30%



(a) ResNet20 (b) VGG11

Figure 3.6: The Effect of Each Channel.

selected channels in a decentralized unlearning setting (**DE Random**). We compare our unlearning schemes with four unlearning methods applied to a federated learning system: retraining from scratch (**Fully Retraining**), pruning-based (**Pruning**) [120], Federaser [81] and calculations-based (**The Right**) [84]). In [120], Wang et al. pruned the most important parameters and then fine-tuned the pruned model to recover the model performance. Liu et al. proposed an unlearning scheme in [81] based on calibration training. For calculations-based methods in [84], Liu et al. replaced the process of computing model updates with Quasi-Newton methods to reduce the consumption in the retraining process. Since Federaser [81] can only unlearn a specific client, and given that our scheme specifically targets class-level unlearning, we compare Federaser with our unlearning scheme in the no-IID scenario. For pruning-based (**Pruning**) [120] and calculations-based (**The Right**) [84]), they only consider unlearning operations in IID scenario, so we compare those schemes with ours under the IID scenario. Both IID and non-IID scenarios demonstrate the high adaptability of our scheme.

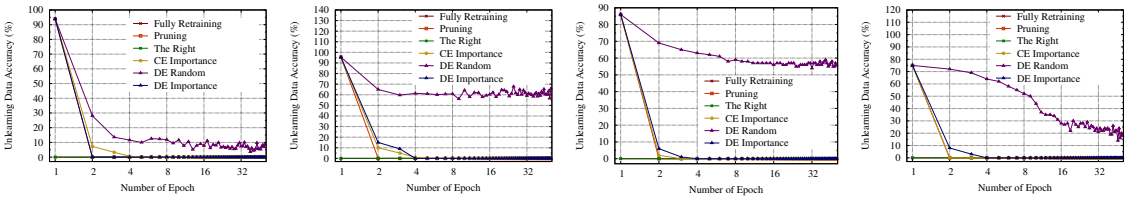
3.5.2 Explanation Analysis

To demonstrate the validity of our scheme, we need to verify whether manipulating a group of parameters in important channels will cause a significant decrease in the performance of the unlearning data. This result indicates that those selected important channels indeed contain information about the unlearning data. Reasonable manipulation that removes this information from the model will achieve the unlearning purpose.

To simulate the real environment settings of federated learning, we evenly distribute the CIFAR-10 datasets to all clients, and the data samples across all clients are IID. We respectively train two models, VGG11 and ResNet20, with the number of participant clients as 10, global epoch = 50, local epoch = 5, local batch size = 128 and learning rate = 0.1. We select the unlearning class as class 1 and consider other class data to be the remaining data. After the global training process, we first calculate the effect of channels within a layer based on Definition 11. Figure 3.6 presents the effect of each channel with respect to unlearning data and remaining data.

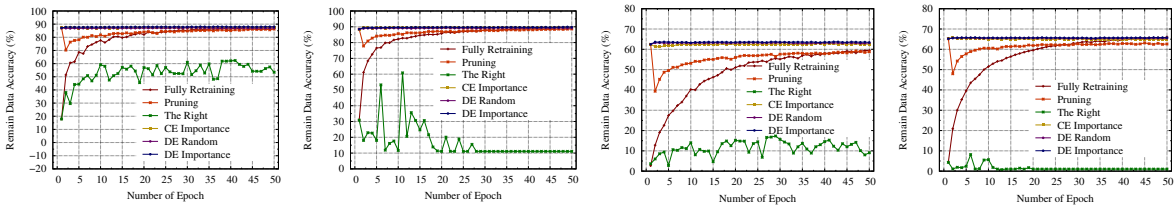
In Figure 3.6, Figure 3.6a illustrates the effect of each channel in layer $layer3.block\_2.conv\_bn2.conv$ in model ResNet20, while Figure 3.6b only illustrates the effect of the first 64 channels in layer $feature.conv\_8$ in model VGG11, since there are totally 512 channels in this layer. It can be seen that when we prune the parameters in one channel at a time, the model performance changes differently for the remaining data and unlearning data. First, when we prune some specific channels, the accuracy of the unlearning data will change dramatically, while the accuracy of the remaining data almost remains stable, such as the channel 23 in layer $layer3.block\_2.conv\_bn2.conv$ (see the masked point α in Figure 3.6a). This indicates that some channels, indeed, contain some information learned from the unlearning data; erasing this information will dramatically affect the performance of unlearning data. Second, some channels are prone to affect both unlearning data and remaining data. When we prune those types of parameters, the accuracy of unlearning data and remaining data will simultaneously decrease, such as the channel 37 in layer $feature.conv\_8$ (see the masked point β in Figure 3.6b). This suggest that some channels affect the performance of both data.

To further illustrate the effect of each channel, we also evaluate how the channel cooperatively affects the unlearning data. To do this, we first select a group of channels based on different ways, then prune the selected channels' parameters and compare the accuracy of the unlearning data and the remaining data. The selection methods include random selection, important selection, and non-important selection. The important selection will choose those channels with higher effect values for unlearning data. The



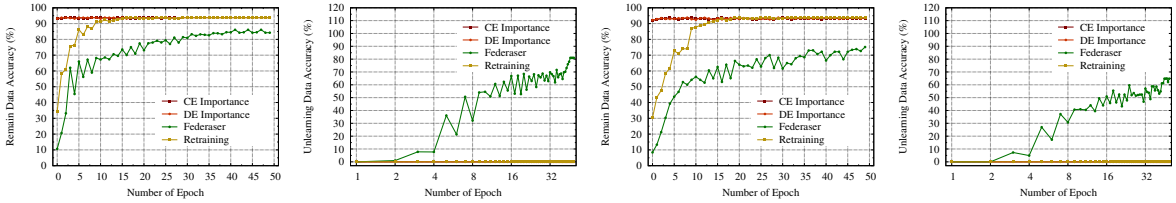
(a) ResNet20 with CIFAR10 (b) VGG11 with CIFAR10 (c) ResNet20 with CIFAR100 (d) VGG11 with CIFAR100

Figure 3.7: Accuracy of Unlearning Data for Different Unlearning Settings in IID.



(a) ResNet20 with CIFAR10 (b) VGG11 with CIFAR10 (c) ResNet20 with CIFAR100 (d) VGG11 with CIFAR100

Figure 3.8: Accuracy of Remaining Data for Different Unlearning Settings in IID.



(a) Remaining Data(MNIST) (b) Unlearning Data(MNIST) (c) Remaining Data(FMNIST) (d) Unlearning Data(FMNIST)

Figure 3.9: Accuracy of Unlearning and Remaining Data in no-IID Scenario.

proportion of selection is $\delta = 0.3$. Table 3.3 shows the results of our experiment.

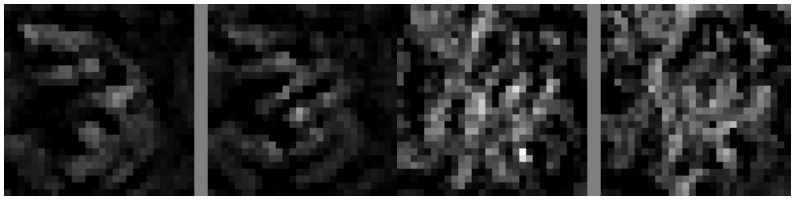
In Table 3.3, the layer $layer3.block\_2.conv\_bn2.conv$ and layer $layer3.block\_2.conv\_bn1.conv$ are from ResNet20, while layer $feature.conv\_7$ is from the VGG11. As shown in this table, similar results for the change of unlearning data and remaining data can be found. First, when we prune a group of important parameters, the model performance for the unlearning data will decrease dramatically while the accuracy of the remaining data stays stable. For example, in layer $layer3.block\_2.conv\_bn2.conv$ in ResNet20, the accuracy of unlearning data drops from 87.10% to 38.50%, while the remaining data's accuracy only decreases by $80.48\% - 70.62\% = 9.86\%$. Second, for random selection or selection of non-important methods, the accuracy of the unlearning

Table 3.4: Experimental Performance Comparison Between Our Scheme and Retraining From Scratch.

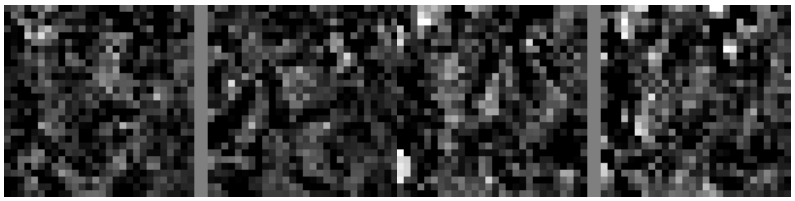
		Comp (s)	Comm (MB)	Storage
ResNet20	Retraining	5670.39	770	45000
with	DE	276.67	1.1	45000
CIFAR10	CE	15.22	0	1600
VGG11	Retraining	4620.23	22800	45000
with	DE	810.78	0.1146	45000
CIFAR10	CE	3.24	0	1600
ResNet20	Retraining	8270.05	1100	49500
with	DE	843.28	3.3	49500
CIFAR100	CE	5.35	0	1600
VGG11	Retraining	7480.52	30560	49500
with	DE	798.12	2.292	49500
CIFAR10	CE	1.23	0	1600

and remaining data will decrease simultaneously, and the degree of decrease of the unlearning data is negligible compared to the important selection method. For example, in $feature.conv\_7$ in VGG11, when we prune the randomly selected parameters, the accuracy of the unlearning data decreases by $94.60\% - 79.9\% = 14.70\%$, while pruning the important parameters results in $94.60\% - 7.30\% = 87.30\%$. In addition, if we prune the important parameters in the intermediate layer of one model, the accuracy of the remaining data also drops a lot. For example, in layer $layer3.block\_2.conv\_bn1.conv$ in ResNet20, pruning the important parameters causes the accuracy of the remaining data to drop by $80.48\% - 65.14\% = 15.34\%$. This illustrates that, in some model layers, especially those near the front part of the model, parameters that are important for the unlearning data may also be important for the remaining data. For example, the important parameters for unlearning data used to detect textures or materials may also be used to detect the same textures or materials in the remaining data in a CNN layer. If we prune the important parameters directly to achieve the unlearning purpose, that may impact the accuracy of the remaining data [120].

Summary: Based on the above experimental results, the most important parameters of the model for unlearning data can be identified based on the ablation study. Reasonable manipulation of these parameters will achieve the purpose of unlearning data. Also, we find that those parameters that affect the unlearning data may also have a large impact on the remaining data. Therefore, in our unlearning scheme, to ensure the performance of the unlearned model, we maintain all the parameters of the global model and only



(a) Original Model (b) Fully Retraining



(c) Decentralized Unlearning (d) Centralized Unlearning

Figure 3.10: The Results of Model Inversion Attack

fine-tune important parameters to unlearn data rather than directly prune all important parameters.

3.5.3 Performance Analysis

3.5.3.1 Accuracy Evaluation

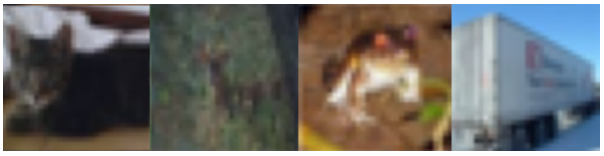
To demonstrate the effectiveness and efficiency of our unlearning scheme, we measure the accuracy of the unlearning data and the remaining data during the unlearning process, respectively. We first train two models, ResNet20 and VGG11, respectively, using two datasets, CIFAR-10 and CIFAR-100. The configuration of the training process is as follows: client number = 10, global epoch = 50, local epoch = 5, local batch size = 128 and learning rate = 0.1. The data samples across all clients are IID. After the global training process, we unlearn the class 1 from two models respectively, using our unlearning scheme in three different settings, CE Importance, DE Random and DE Importance. For CE Importance, we put 1600 samples on the server side to simulate the training data owned by the server and set learning rate = 0.01 and batch size = 64 to fine-tune models. For δ, we set $\delta = 0.05$ for ResNet20, and $\delta = 0.01$ for VGG11. The other setting for the fine-tuning process is the same as the original global training process. We compare the results with three different methods: fully retraining, pruning-based methods [120] and the right [84]. For the pruning-based and the right method, we set all hyperparameter settings suggested in their paper. For fully retraining, we set hyperparameters to be the same as the original training process.

Table 3.5: The Results of the MIAs

	CIFAR-10 + ResNet	MNIST + ResNet
Original	95.62%	94.56%
Fully Retraining	0.00%	0.00%
CE important	0.00%	0.00%
DE important	0.00%	0.00%

The results are shown in Figure 3.7 and Figure 3.8, for fully retraining and the right [84], the x-axis represents the number of retraining epochs; for the pruning-based method [120], CE Importance, DE Random and DE Importance, the x-axis represents the number of training epochs in the fine-tuning process. We also scale the x-axis to log_2^x for convenience in Figure 3.7. The y-axis in Figure 3.7 represents the accuracy of unlearning data (class 1), while in Figure 3.8, it represents the accuracy of the remaining data. For the data to be unlearned, CE Importance and DE Importance basically reduce the accuracy to 0 within 4 epochs, which means that the model cannot accurately identify the class 1 after almost 4 epochs of the unlearning process. For DE Random, it is difficult to reduce the accuracy to 0 since the fine-tuning parameters are randomly selected, not those with the most influence. For the pruning-based method [120], since it prunes all the parameters about the unlearning class 1, it has substantially low accuracy in the beginning. The fully retraining and the right [84] method also has a precision of 0 for the unlearning class since its training set does not contain unlearning data.

However, the other three models, fully retraining, pruning-based [120] and the right [84], do not meet the requirements of unlearning for the accuracy of the remaining data. For the accuracy of the remaining dataset in Figure 3.8, the accuracy of CE Importance and DE Importance does not degrade significantly during the unlearning process. This is because we fine-tune only part of the channels while keeping all parameters of the previous model, particularly parameters unrelated to the unlearned class 1. For fully retraining, since it trains the model from the beginning, the model is only available when the training process is completely finished. For the pruning-based method [120], in their unlearning scheme, while the parameters associated with class 1 are pruned, the parameters with other classes also become incomplete, and their accuracy will decrease. Therefore, the unlearned model is only available when the fined-tuned training process is complete. Note that the model accuracy of the pruning-based method [120] is generally higher than that of fully retraining since some of the parameters of the previous model (which are not pruned) are retained in [120]. The right [84] is also based on retraining



(a) Original Image

(b) Image Generated in Fully Retraining

(c) Image Generated in Decentralized Unlearning

Figure 3.11: The Results of GradAttack

methods, but unlike fully retraining, in the retraining process, they frequently used Quasi-Newton to calculate the update of the global model instead of using the training method. Calculated-based methods are usually applicable to simple models. When faced with complex models with a huge amount of training data, it is difficult for the model to reach an ideal state.

In addition, we also evaluate the effectiveness of our unlearning scheme in no-IID scenario. We choose two datasets, MNIST and Fashion MNIST, for this experiment and assign one class of data per user. The configuration of the training process is as follows: client number = 10, global epoch = 20, local epoch = 5, local batch size = 64 and learning rate = 0.1. For CE Importance, we put 1600 samples on the server side to simulate the training data owned by the server and set the learning rate = 0.1 and batch size = 64 to fine-tune models. For δ, we set $\delta = 0.2$ for both datasets. We also compare the results with Federaser [81] and set all hyperparameter settings suggested in their paper to construct our experiments. The results are shown in Figure 3.9.

From Figure 3.9, we can see that our scheme can also effectively unlearn class information under no-IID and ensure the performance of the model for the remaining data. As for Federaser [81], since the model still has information about classes that need to be unlearned in the calibration training, the unlearning results are not good.

As can be seen from all subplots in Figures 3.7, 3.8 and Figure 3.9, our method can remove the contribution of the unlearning class and only use a smaller epoch while simul-

taneously ensuring the accuracy of the remaining dataset. More importantly, during each epoch, our scheme only updates part of the model parameters, and the amount of training data used by the server is much smaller than other solutions. To further demonstrate the efficiency of our scheme, we record the cost during the entire unlearning process, and the results are shown in Table 3.4. From those results, the proposed unlearning schemes are found to be efficient in terms of computational, communication, and storage costs compared to retraining from scratch. For example, in our CE Importance unlearning scheme, each epoch only costs $15.22s$ and only needs $1,600$ images for ResNet20 with the CIFAR10 dataset, while fully retraining needs $45,000$ images and costs $5670.39s$. Overall, our unlearning simultaneously considers both effectiveness and efficiency.

3.5.3.2 Model Inversion Attack

To further analyze the effectiveness of our scheme, we construct the following experiment based on model inversion attack [38] and membership inference attack [133]. We implemented the model inversion attack as described in [38]. We first train model ResNet18 based on the MNIST dataset with the number of participant clients as 10, global epoch = 10, local epoch = 5, local batch size = 128 and learning rate = 0.1. We select the unlearning class as class 3 and consider other class data as the remaining data. After the global training process, we select the most influential parameters for unlearning class 3 and set the selection factor $\delta = 0.2$. For decentralized unlearning and the fully retraining schemes, we set the hyper-parameters the same as the original training process. For centralized unlearning, we set the local training epoch = 5, lr = 0.005. Figure 3.10 depicts the results of model inversion attacks against models affected by different unlearning methods.

As shown in Figure 3.10, the images generated by fully retraining, our centralized unlearning, and decentralized unlearning schemes are dark and jumbled since the model inversion attack has relatively little gradient information to rely on. As expected, fine-tuning the influential channels based on perturbation data makes it impossible to infer any significant information about the unlearning data. This implies that the information of unlearning data is almost entirely removed by our centralized and decentralized unlearning schemes, virtually eliminating the possibility of obtaining valuable class information via model inversion attacks.

3.5.3.3 Membership Inference Attack

We also leverage Membership Inference Attacks (MIAs) in paper [133], to assess whether the unlearning data are still identified in the training dataset. We set the number of shadow models as 20 and the training epoch of the shadow model as 10, batch size = 64. In the context of MIAs, a shadow model is a surrogate model trained to mimic the behavior of the target model that is being attacked. The attack model is a fully connected network with two hidden linear layers of width 256 and 128, respectively, with ReLU activation functions and a sigmoid output layer. We evaluate our unlearning scheme with two settings, CIAFR-10 + ResNet and MNIST + ResNet. We set the unlearning class to 3. Other configurations of the training process are the same as those described above. Table 3.5 shows the results of our evaluation.

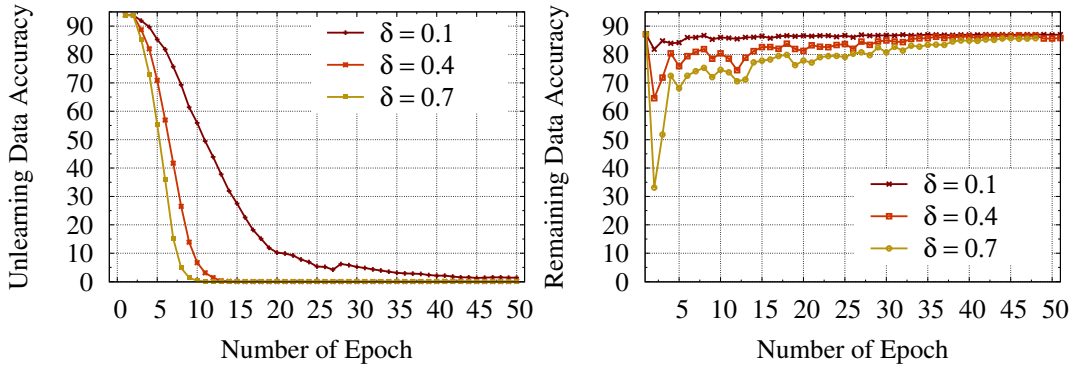
As seen from the table, for all original models, MIA has a high success rate; i.e., it was able to derive the training dataset containing unlearning data successfully. However, for all other methods, the success rate of MIAs is lower, which indicates that MIA cannot deduce the existence of unlearning data in the training dataset; this indicates that the effect of the unlearning data has been successfully removed from the unlearned model.

Summary: The above experiments show that our scheme can effectively maintain the performance of the unlearned model, and reduce the probability of an attacker obtaining confidential information about those targets.

3.5.4 Gradient Attack

In our decentralized unlearning scheme, we only upload the updates of partial channels' parameters. In order to evaluate whether updating partial parameters can alleviate the risks caused by gradients, we use GradAttack [57] to attack the training process of fully retraining and our decentralized unlearning, respectively. We use the *realistic setting* in their paper to attack ResNet20 with CIAFR-10 and set the unlearning data as class 1 and $\delta = 0.05$ in our decentralized unlearning setting. Other experimental settings are the same as the experiments above. Figure 3.11 shows part of the results of our experiments.

As shown in Figure 3.11, for the fully retraining scheme, since it uploads all the parameters' updates, some information of original data can be recovered using the gradients. For our decentralized unlearning, it is impossible to obtain any information from the gradient-based generated images since we only upload partial gradients. **In summary**, our decentralized unlearning scheme not only reduces the consumption caused by communication but also alleviates the security risks caused by gradients.



(a) Unlearning Data (b) Remaining Data

Figure 3.12: The Effect of δ in Different Settings

3.5.5 The Effect of Selection Factor δ

The hyperparameter δ determines how many influential channels are selected. To evaluate the effect of δ, we use the above experimental setting, in which the dataset is CIFAR-10, and the model is ResNet20, to train the global model. After that, we execute a centralized unlearning process with different δ. Figure 3.12a and Figure 3.12b show the results on the unlearning data and remaining data, respectively. It can be seen from Figure 3.12a, that selecting more influential channels can accelerate the unlearning process due to the increasing number of pruned and fine-tuned channels. However, as depicted in Figure 3.12b, increasing the number of pruned channels will decrease the performance of the model after the pruning process. This will result in the need for more fine-tuning to recover model performance. Meanwhile, more channels mean the consumption of updates to uploads will increase in the decentralized unlearning process. Therefore, in principle, the setting of the selection factors should consider the usability of the unlearned model and the efficiency of the unlearning process.

3.6 Summary

This chapter proposed a federated unlearning scheme that addresses the practical problem of removing the effects of particular classes from a trained model in the federated learning context. As a solution, this chapter analyzed the most influential channels of a model for the given classes based on the ablation study. For unlearning class, this chapter provided two effective methods that only fine-tune partial influential channel parameters with the help of perturbation data. The analysis and experimental results demonstrate

that under our scheme, the model can remove the impact of classes that need to be removed and ensure the accuracy of the remaining data in a quick and efficient manner.

This chapter inspires us that we can explore more complex federated learning scenarios containing more kinds of unlearning requests, for example, unlearning one sample or all data in one client. At present, we have only illustrated that there are different influences within a trained model's channel at the class level. Due to the various unlearning requests involved, we intend to explore ways to extend and/or modify the current method of evaluating channel importance and develop new federated unlearning schemes based on these revised evaluations. In addition, we plan to design a more powerful scheme that supports unlearning requests from Natural Language Processing (NLP) models, combined with other technologies such as information theory.

CHAPTER 4

TOWARDS EFFICIENT TARGET-LEVEL MACHINE UNLEARNING BASED ON ESSENTIAL GRAPH

Machine unlearning is an emerging technology that has come to attract widespread attention. A number of factors, including regulations and laws, privacy, and usability concerns, have resulted in this need to allow a trained model to forget some of its training data. Existing studies of machine unlearning mainly focus on unlearning requests that forget a cluster of instances or all instances from one class. While these approaches are effective in removing instances, they do not scale to scenarios where partial targets within an instance need to be forgotten. For example, one would like to only unlearn a person from all instances that simultaneously contain the person and other targets. Directly migrating instance-level unlearning to target-level unlearning will reduce the performance of the model after the unlearning process, or fail to erase information completely. To address these concerns, we have proposed a more effective and efficient unlearning scheme that focuses on removing partial targets from the model, which we name "target unlearning" [1]. Specifically, we first construct an essential graph data structure to describe the relationships between all important parameters that are selected based on the model explanation method. After that, we simultaneously filter parameters that are also important for the remaining targets and use the pruning-based unlearning method, which is a simple but effective solution to remove information about the target that

[1]The main content of this chapter has been published in Heng Xu, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, Wei Zhao, Towards Efficient Target-Level Machine Unlearning Based on Essential Graph. IEEE Transactions on Neural Networks and Learning Systems, 2024.

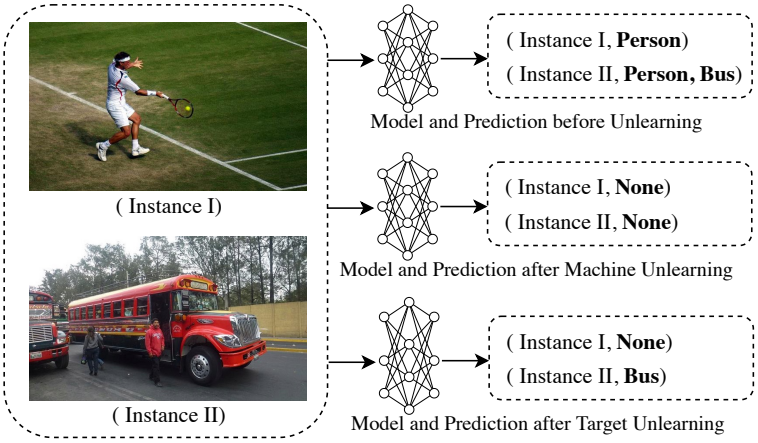


Figure 4.1: Target unlearning.

needs to be forgotten. Experiments with different training models on various datasets demonstrate the effectiveness of the proposed approach.

4.1 Introduction

Current machine unlearning works mainly focus on unlearning partial instances selectively, such that they only remove information at instance level [132]. We argue that we may need to support machine unlearning at the target level. We use *target* to denote the object that appears in an instance, and the scheme is defined as *target unlearning*. The requirements for unlearning targets can arise from eliminating the model's capabilities for unimportant, sensitive, or abnormal objects. For instance, it can involve removing a specific category from a multi-label classification task or eliminating a particular object from an object detection task. Figure 4.1 shows the difference between traditional instance-level unlearning and target unlearning, where the two instances on the left side contain different numbers of targets. Instance I contains a person as one target, while instance II contains two objects, a person and a bus, so we consider them as two targets. When a request to remove information about the person is received, traditional instance-level unlearning schemes will directly remove two instances; however, the bus will also be affected as instance II will also be removed. For target unlearning, it will only remove accurate information about the person from each instance.

Therefore, how to remove the impact of targets accurately is our research question. There are mainly two unique challenges on this question. First, in the context of target unlearning, one instance may contain multiple targets, and those targets that need to

be removed are tightly embedded in each instance, which makes it difficult to separate them in the instance. Moreover, multiple targets in each instance are interactive, and the machine learning model may further interact with diverse targets in different layers. it is difficult to remove the effect of a certain target while minimizing collateral damage to model performance for other remaining targets.

To address the above concerns, we propose an interpretive approach to finding parameters that have the greatest impact on targets that needs to be unlearned, and then pruning these parameters to erase information [41, 46, 120]. Specifically, we use model explanation method to find the most influential parameters for a particular region within an instance [64, 119]. This approach can effectively separate various targets from the instance, and directly identify the model parameters that have a great impact on these separated targets. However, interpretive methods typically focus on a single layer and pruning all influential parameters within that layer may negatively impact the performance of the model for the remaining data. First, due to the specific structure of deep neural networks, considering parameter pruning in only one layer may not satisfy the effectiveness of unlearning. For example, focusing solely on the influential parameters in the last layer and then pruning those parameters to erase the information may not be sufficient for preserving privacy. Once an attacker has obtained all parameters of the model in a white-box scenario, they can still infer information from the penultimate level about the unlearned target. Second, the parameters selected based on the model explanation may also be important for other targets, and blindly pruning all those selected parameters will reduce the performance of the unlearned model.

To tackle those challenges, we construct an essential graph structure to describe the relationships of all influential parameters in each layer. This essential graph also provides a way to consider the performance of the unlearned model on remaining targets. It will simultaneously consider both the remaining target and the unlearning target to filter the parameters during the balanced process. To validate our proposed unlearning scheme, we evaluate multiple machine learning tasks under different models and various datasets, which demonstrates that our proposed scheme can effectively achieve the removal of target-level information and the ability to reasonably weigh the performance of the remaining targets. The main contributions of this chapter can be summarized as follows:

- We initiate the exploration of machine unlearning at the target level by defining the problem of efficient target unlearning, including its formalization and challenges.

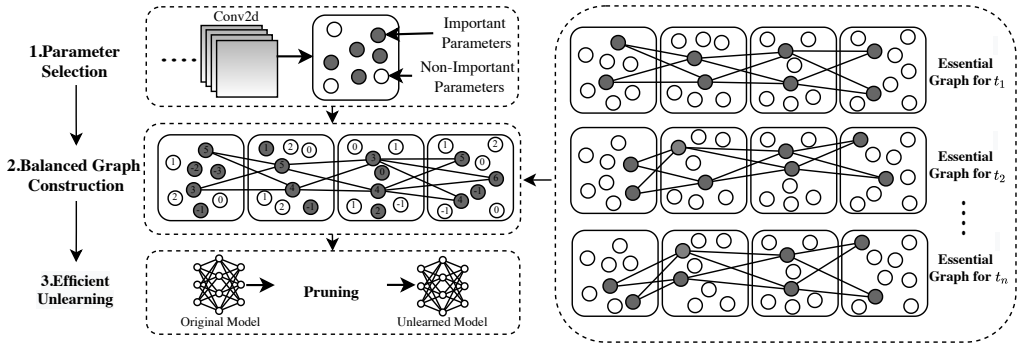


Figure 4.2: A schematic view of target unlearning.

- We investigate target unlearning from a novel interpretative perspective and selectively prune the most influential parameters to achieve the unlearning purpose.
- We propose an essential graph data structure to describe the relationships between influential parameters of multiple layers and simultaneously consider the remaining data's impact to mitigate the performance degradation.
- We validate our target unlearning scheme across various datasets, models, and machine learning tasks, demonstrating that our design can maintain model utility while achieving significant unlearning effectiveness.

4.2 Problem Definition

Assume the instance space can be denoted as $\mathscr{X} \subseteq \mathbb{R}^d$, with the label space defined as $\mathscr{Y} \subseteq \mathbb{R}$. We use $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\} \subseteq \mathbb{R}^d \times \mathbb{R}$ to represent a training dataset, in which each instance $\mathbf{x} \in \mathscr{X}$ is a d-dimensional vector, $y \in \mathscr{Y}$ is the corresponding label, and n is the size of $\mathscr{D}$. Based on this, we can obtain a model M through a machine learning algorithm $\mathscr{A}$, that is, $M = \mathscr{A}(\mathscr{D})$. Other symbols that appear in this chapter and their corresponding descriptions are listed in TABLE 3.1.

Let $\mathscr{D}_u \subset \mathscr{D}$ be a subset of the training dataset, whose influence we want to remove from the trained model M. Let its complement $\mathscr{D}_r = \mathscr{D}_u^{\complement} = \mathscr{D} / \mathscr{D}_u$ be the dataset that we want to retain. Now, we give the definition of target unlearning. which focuses on the unlearning request on the target level:

Definition 12 (Target Unlearning)**.** *Consider one object within instances, denoted as t, that we want to remove its effects from all training datasets and the trained model. Target unlearning process $\mathscr{U}_t(M, \mathscr{D}, t)$ is defined as a function from an already-trained*

model $M = \mathscr{A}(\mathscr{D})$, a training dataset $\mathscr{D}$, and a specified object t to a model M_u, which ensures that the unlearned model M_u performs as though it had never seen the object t in all training dataset.

4.3 Methodology

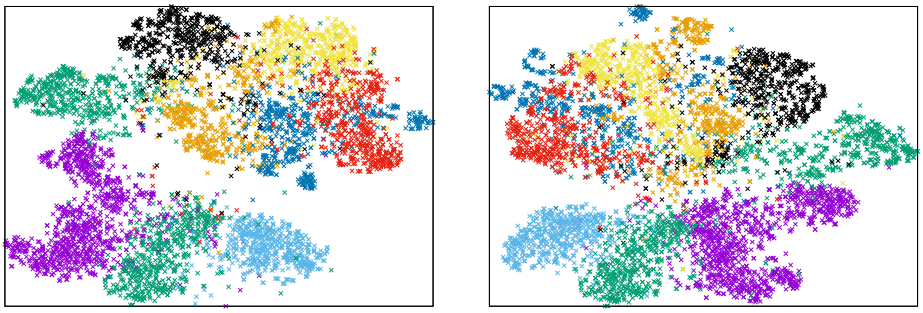
4.3.1 Overview

As shown in Figure 4.2, target unlearning mainly consists of the following three steps: parameters selection, balanced graph construction and efficient unlearning, where the balanced graph construction is based on the essential graph data structure for each target. For the purpose of simplification in explanation, we first introduce the concept of an essential graph (Section 4.3.2). Then, we illustrate how to select the important parameters for a special target within each layer based on the model explanation (Section 4.3.3). After those steps, we construct the balanced graph to identify parameters that are simultaneously important for the remaining targets and determine which parts of the parameters have a critical influence on unlearning target. After those steps, we manipulate critical parameters that have the greatest effect on the unlearned target to achieve the unlearning purpose (Section 4.3.4).

4.3.2 Essential Graph

The core of the model explanation is that model parameters contribute differently to the overall model performance [140]. This means that some parameters are universally important for all targets, while others are selectively important for specific targets. Appropriate pruning of these parameters can affect the performance of the model with respect to the corresponding targets, which will achieve the purpose of unlearning.

However, it is insufficient to unlearn targets by only considering one-layer parameters [9]. Only selecting the most influential parameters within one layer may not meet the goal of effectiveness due to the specific structure of the deep neural network. For example, if only considering the last layer to select parameters that contain information about targets and then pruning that information from the last layer, once the attacker obtains all parameters of the model in a white-box scenario, they can still deduce information about the unlearning target. As shown in Figure 4.3, we construct an easy-to-understand experiment to show this data leakage problem. We cluster the outputs of the last layer and the penultimate layer of the VGG11 model based on the



(a) The Output of Last layer (b) The Output of Penultimate Layer

Figure 4.3: Evaluation of data leakage from the penultimate-layer based on STL10 dataset and VGG11 model.

tSNE algorithm, respectively. From the results, it can be seen that the clustering results are very similar, which indicates that the output of the penultimate layer still contains some information about all data; removing information only at the last layer is not enough. In addition, model parameters in the neighboring layer are usually interactive, considering more than one layer simultaneously can efficiently analyze the impact of parameters on the targets that need to be unlearned.

We assume that the model needs to execute an unlearning process containing L convolutional layers. Each layer's parameters can be represented as $\mathbf{w}_l = \mathscr{R}^{O_l \times I_l \times K_l \times K_l}$, where O_l, I_l and K_l denote the number of output channels, number of input channels, and the kernel size, respectively. We use $A_l \in \mathscr{R}^{I_l \times H_l \times W_l}$ to denote the input of this convolutional layer, where H_l, W_l are the height and width of the input respectively. The output feature maps of this layer is calculated as $F_l = A_l \otimes \mathbf{w}_l$, where $\otimes$ is the convolutional operation, $F_l \in \mathscr{R}^{O_l \times H_{l+1} \times W_{l+1}}$. We use f_l^k to denote the feature maps for the k-th channel in this convolutional layer. The output of this CNN model can be represented as $Y = M(\mathbf{x})$. We denote Y_t as the output probability of target t.

To describe and analyze the relationship of important parameters between multiple layers, we construct one useful graph data structure, named *essential graph*.

Definition 13 (Essential graph)**.** *An essential graph, $G = (\mathscr{V}, \mathscr{E})$, describes the relationship between important parameters. Each node in $\mathscr{V}$ indicates the importance score of each corresponding output channel, while the edges in $\mathscr{E}$ between each node indicate the connection relationships.*

Specifically, assuming that the set of important channels in layer l for target t can be expressed as $\mathscr{T}_l^t$ (The calculation of $\mathscr{T}_l^t$ will be explained later). The essential graph can

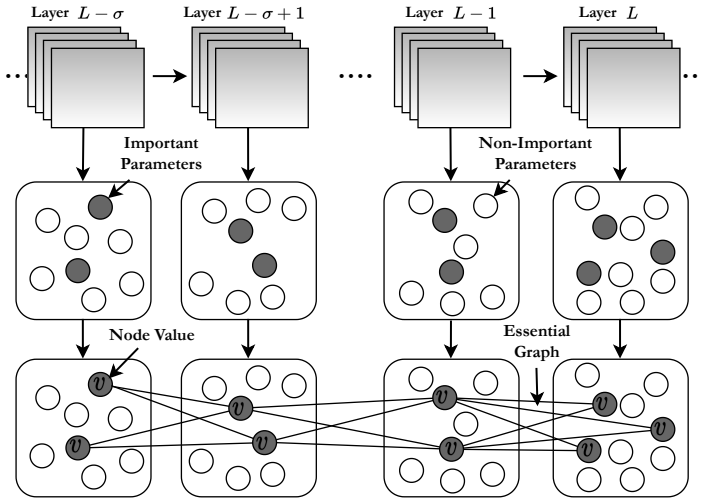


Figure 4.4: The construction of the essential graph.

be denoted as follows:

$$
\begin{aligned}
&G = (\mathcal{V}, \mathcal{E}) \\
&\quad s.t. \mathcal{V} = \left\{ v_{l,k} |\ k \in [0, |\mathcal{T}_l^t|], v_{l,k} = \mathcal{O}(\mathcal{T}_{l,k}^t) \right\} \\
&\quad\quad \mathcal{E} = \left\{ e_{(l,i),(l+1,j)} |\ i \in [0, |\mathcal{T}_l^t|], j \in [0, |\mathcal{T}_{l+1}^t|] \right\} \\
&\quad\quad l \in [L-\sigma, L-1]
\end{aligned}
\tag{4.1}
$$

Each element in $\mathcal{T}_l^t$ is denoted as one channel $\mathcal{T}_{l,k}^t$, where k ranges from 0 to $|\mathcal{T}_l^t|$, the size of the set $\mathcal{T}_l^t$. In subsequent sections, when we refer to parameters, it typically denotes all the parameters within one channel. In the essential graph, $v_{l,k}$ represents the value of each node, which is determined by a function $\mathcal{O}(\cdot)$ using $\mathcal{T}_{l,k}^t$ as the parameter. The function $\mathcal{O}(\cdot)$ can be arbitrary, but it must ensure that the node values for channels of different importance should be different and values for channels of similar importance are similar. The exact definition of $\mathcal{O}(\cdot)$ will be provided later. $e_{(l,i),(l+1,j)}$ mean one connection relationship between node $v_{l,i}$ and $v_{l+1,j}$, where $i \in [0, |\mathcal{T}_l^t|]$ and $j \in [0, |\mathcal{T}_{l+1}^t|]$.

σ is used to control how many layers should be considered. The reason we only consider the last σ layers is that the CNN model usually consists of a low-level feature extractor and a high-level classifier. The high-level classifier is used to distinguish different targets, and only removing the information contained in the high-level classifier will generally not influence model performance for the remaining targets.

The edge between two nodes with larger node values indirectly indicates a strong correlation between the two nodes' parameters. For example, the parameters of a layer

Algorithm 3: Essential Graph Construction

Input: model M, parameter index $\mathcal{T}_l^t$ with $l \in [L-\sigma, L]$, layer selection factor σ
Output: essential graph:G

1 Initialize $\mathcal{V}$ and $\mathcal{E}$ to empty $\emptyset$
2 **for** *each layer* l *in* $[L-\sigma, L-1]$ **do**
3 **for** *each index* $\mathcal{T}_{l,i}^t$ *in* $\mathcal{T}_l^t$ **do**
4 **for** *each index* $\mathcal{T}_{l+1,j}^t$ *in* $\mathcal{T}_{l+1}^t$ **do**
5 $\mathcal{V} \leftarrow$ add $v_{l+1,j}$ with value $\mathcal{O}(\mathcal{T}_{l+1,j}^t)$
6 $\mathcal{E} \leftarrow$ add $e_{(l,i),(l+1,j)}$
7 $\mathcal{V} \leftarrow$ add $v_{l,i}$ with value $\mathcal{O}(\mathcal{T}_{l,i}^t)$
8 **return** $G = (\mathcal{V}, \mathcal{E})$

l for the detection of textures and materials may be jointly linked to a parameter of a particular scene detector in layer $l+1$. All the links between the two layers with larger node values form a route for the transmission of important features of one target.

Algorithm 3 performs the construction of *essential graph* for one instance with target label t, while Figure 4.4 shows the process. Before this algorithm, the model provider indexes all layers that need to be considered to remove information about a target t and select important parameters w.r.t target t. Assume that the results of layer l denoted as $\mathcal{T}_l^t$, and that in layer $l+1$ denoted as $\mathcal{T}_{l+1}^t$. We use $v_{l,k}$ to denote one node in G of corresponding channel k in layer l and use $\mathcal{O}(\mathcal{T}_{l,k}^t)$ to represent the node value. The link between use $v_{l,i}$ and $v_{l+1,j}$ is denoted as $e_{(l,i),(l+1,j)}$. For each layer, model provider sequentially add the node $v_{l+1,j}$ and $v_{l,i}$ to $\mathcal{V}$, and add the edge $e_{(l,i),(l+1,j)}$ to $\mathcal{E}$ to construct the relationship between layer l and $l+1$ (Line 2-7). It is important to note that the above steps can be performed at any time during the inference phase; thus, the time consumption of the unlearning process is not increased.

4.3.3 Parameters Selection

In the above section, we describe how to construct an essential graph based on the important parameters of each layer. In this section, we describe how to find those important parameters within each layer based on model explanation.

There are many works to select important parameters within a model, such as ablation methods [6, 91, 141]. Ablation methods usually investigate the effect of one unit by removing this unit to understand the contribution of this unit to the overall performance. However, there are two main issues when using these methods to find

important parameters for one specific target. Firstly, these methods require indexing all parameters, and the computational cost for complex models with large datasets is high. This will reduce the efficiency of the unlearning process. In addition, these methods analyze the influence of a single parameter at the instance level, which is not sufficient for finding important parameters at the target level. CAM-based methods are another interpretive methods and usually used to highlight the important regions in one image [18, 64, 103, 119, 140]. It did not originally apply to finding importance parameters. Here, we illustrate how it can be used to analyze the importance for a particular target.

Consider a convolutional layer l in a model M. CAM L^t_{CAM} with respect to target t can be described as follows:

$$L^t_{\text{CAM}} = \text{ReLU}\left(\sum_k \alpha^k_l f^k_l\right) \tag{4.2}$$

where k denotes the index of the feature map in layer l. $\alpha^k_l f^k_l$ denotes the linear combination between α^k_l and f^k_l. α^k_l represents the importance of each feature map f^k_l for instance with target t and calculated based on various methods, such as Original CAM [140], Score-CAM [119] and Grad-CAM [103]. For convenience, we calculate α^k_l the based on the method in Grad-CAM [103]:

$$\alpha^k_l = \frac{1}{Z}\sum_i\sum_j \frac{\partial Y_t}{\partial f^k_{l,(i,j)}} \tag{4.3}$$

where $\frac{1}{Z}\sum_i\sum_j$ in Equation 4.3 represents the global average pooling operation, while $\frac{\partial Y_t}{\partial f^k_{l,(i,j)}}$ denotes the gradients of the Y_t with respect to $f^k_{l,(i,j)}$.

Based on the above steps, Grad-CAM will generate a heatmap that can highlight the important regions for a given target in one image. Next, we will simply illustrate that α^k_l also represents the importance of each channel for all instances with the same target. We first give the definition of the influence of channel parameters:

Definition 14. *(Influence of Channel Parameters). Given a model M that takes an instance $\mathbf{x}$ with target t. It will generate the heatmap with the intermediate weights denoted as α^k_l. We define the influence of corresponding channel parameters w^k_l in layer l toward target t as $s_{w^k_l} = \alpha^k_l$.*

Consider a CNN model with L layers, excluding the final linear layers; the model output can be expressed as $Y = R(\mathbf{w}_l \otimes R(\mathbf{w}_{l-1} \otimes \ldots \otimes R(\mathbf{w}_2 \otimes R(\mathbf{w}_1 \otimes x))\ldots))$, where $\mathbf{w}_l$ represents the parameter in layer l, $R(\cdot)$ is the activation function, and $\mathbf{x}$ is the model

Algorithm 4: Importance Parameter Selection

Input: model M, target label t, layer l
Output: index of important parameters: $\mathscr{T}_l^t$

1 Initialize $\mathscr{Z}_l$ to an empty directory $\emptyset$
2 Initialize $\mathscr{T}_l^t$ to an empty directory $\emptyset$
3 Select and feed a instance $\mathbf{x}$ to model $M(\mathbf{x})$
4 **for** *each channel k in layer l* **do**
5 Calculate $\alpha_l^k = \frac{1}{Z}\sum_i \sum_j \frac{\partial Y_t}{\partial f_{l,(i,j)}^k}$
6 Add α_l^k to $\mathscr{Z}_l$
7 **for** *each channel k in layer l* **do**
8 **if** α_l^k *in Top* δ **then**
9 $\mathscr{T}_{l,k}^t$ = True
10 **else**
11 $\mathscr{T}_{l,k}^t$ = False
12 **return** $\mathscr{T}_l^t$

input. We already know that each α_l^k represents the importance of the corresponding feature map f_l^k. We use the function $I(\cdot)$ to denote importance, and denote the relationship mentioned above as $\alpha_l^k \propto I(f_l^k)$.

Consider one layer l within the model, with input denoted as $A_l \in \mathscr{R}^{I_l \times H_l \times W_l}$, where $A_l = R(\mathbf{w}_{l-1} \otimes \ldots \otimes R(\mathbf{w}_2 \otimes R(\mathbf{w}_1 \otimes x))\ldots)$, and parameters as $\mathbf{w}_l$. The output of this layer, that is, feature maps, can be denoted as $F_l = A_l \otimes \mathbf{w}_l$, where $f_l^k = A_l^k \otimes \mathbf{w}_l^k$. When changing one specific channel's parameters w_l^k in layer l to obtain $w_l^{'}$, the output will be changed to $F_l^{'} = A_l \otimes w_l^{'}, F_l \neq F_l^{'}$. This means that when given an instance, the output feature maps of this layer are only determined by the parameters in this layer. We use the notation $\rightarrow$ to denote this relationship and define it as $w_l^k \rightarrow f_l^k$.

Based on the above two inferences, namely $\alpha_l^k \propto I(f_l^k)$ and $w_l^k \rightarrow f_l^k$, we can deduce $\alpha_l^k \propto I(w_l^k) = s_{w_l^k}$ in definition 14. This means α_l^k represents the importance of the output feature map [75, 103, 119, 140], it also represents the importance of the corresponding channel parameters of this layer. In addition, based on the model explanation work in [6, 91, 141], we can conclude that given two different instances with the same target label t, $\mathbf{x}_i^t$ and $\mathbf{x}_j^t$, the distribution $\mathbb{E}(\cdot)$ of influence of channel parameters in layer l for two instances, $\mathbb{E}(s_{w_l^k}, \mathbf{x}_i^t)$ and $\mathbb{E}(s_{w_l^k}, \mathbf{x}_j^t)$, is similar. Based on this, we can select the most important channel parameters for the unlearning target within one layer based on one instance, then use reasonable operations to remove the information in those parameters.

The process of importance parameter selection is described in Algorithm 4. In line 2,

in order to calculate the importance within a layer for a target t, the model provider first selects an instance $\mathbf{x}$ that contains one or a group of targets, one of which's target label is t. Then, this selected instance will be fed to the model M to record all essential data to calculate each channel's α_l^k (lines 3-6). After that, lines 7-11 return the index of selected important parameters, where δ is a hyper-parameter used to control the proportion of important parameters.

4.3.4 Balanced Graph Construction and Unlearning

Given the essential graph generated based on Section 4.3.2 using the important parameters within each layer selected based on Section 4.3.3, it is worth noting that some task-specific parameters in this graph, such as those used to detect textures, are not only important for the unlearning target but may also be used for other detection tasks [120]. For example, the parameters for target t used to detect textures or materials may also be used to detect other targets' textures or materials. Directly removing information on all those parameters will destroy the performance of other targets in the remaining data. To balance the unlearning effectiveness and model performance of remaining targets. We further select the most critical parameters based on balance operations.

Our main idea is to simultaneously consider both unlearning target and remaining targets in the process of constructing graph nodes, and set different node values according to the type of target. After that, summing these node values, and use the final summation to reflect the effect of the parameters on unlearning target. For the calculation of the values $\mathcal{O}(\mathcal{T}_{l,k}^t)$, we define as:

$$\mathcal{O}(\mathcal{T}_{l,k}^t) = \begin{cases} |Y|, & \mathcal{T}_{l,k}^t = True, t \in \mathcal{D}_u \\ -1, & \mathcal{T}_{l,k}^t = True, t \in \mathcal{D}_r \\ 0, & \mathcal{T}_{l,k}^t = False \end{cases} \tag{4.4}$$

where $|Y|$ denotes the number of targets in training data. The above equation indicates that we consider different ways of calculating node values in constructing the essential graph; for unlearning target, we calculate based on $\mathcal{O}(\mathcal{T}_{l,k}^t) = |Y|$ for each selected channel, while for targets from the remaining data, we calculate based on $\mathcal{O}(\mathcal{T}_{l,k}^t) = -1$. For all channels that are not selected as important channels, we calculate based on $\mathcal{O}(\mathcal{T}_{l,k}^t) = 0$. After each target has constructed its corresponding essential graph, we sum the graphs constructed by different targets. If a graph node is important for both data, then the summed node value $v_{l,k}^{sum} = |Y|_{t \in \mathcal{D}_u} + \sum_{t \in \mathcal{D}_r}(-1)$ will tend to the target

Algorithm 5: Balanced Graph Construction

Input: model M, training data $\mathscr{D}$
Output: balanced essential graph: G^*

1 **for** *each target t in* $\mathscr{D}$ **do**
2 **for** *each layer l in* $[L-\sigma, L]$ **do**
3 $\mathscr{T}_l^t \leftarrow$ calculate based on Algorithm 4 (M, t, l)
4 Add $\mathscr{T}_l^t$ to $\mathscr{T}^t$
5 **for** *each target t in* $\mathscr{D}$ **do**
6 $G^t \leftarrow$ construct based on Algorithm 3(M, $\mathscr{T}^t$)
7 **for** *each target t in* $\mathscr{D}$ **do**
8 $v_{l,k}^{sum} \leftarrow$ Accumulate all node value with the same layer l and channel k in each G^t
9 Add $v_{l,k}^{sum}$ with edges to G^*
10 **return** G^*

number of D_u. If the node is important for the unlearning target only, the node value will tend to $v_{l,k}^{sum} = |Y|_{t \in \mathscr{D}_u}$.

Algorithm 5 describes the balancing process of the essential graph. In lines 1-4, the model provider first select important parameters for each target within each layer in $[L-\sigma, L]$. $\mathscr{T}_l^t$ represents the result of target t in layer l and $\mathscr{T}^t$ represents the result of target t in all layers. Then, the model provider constructs each essential graph based on the Algorithm 3, in which the model provider will choose different $\mathscr{O}(\mathscr{T}_{l,k}^t)$ in Equation 4.4 based on the type of target (Lines 5-6). Lines 7-9 accumulate the values of all the corresponding nodes to get the balanced essential graph.

After the balance operation, the operation of target unlearning is simplified to selecting appropriate nodes based on the nodes' values in G^* and manipulating the model parameters corresponding to these selected nodes to remove information about unlearning targets. As shown in Algorithm 6, for each selected layer, we perform the pruning operation for channels where the corresponding node value is equal to $|Y|$.

4.3.5 Performance Analysis

In this section, we analyze the performance of the proposed scheme in terms of computational, and storage overheads.

Algorithm 6: Unlearning Process

Input: Balanced Graph G^*, Model M
Output: unlearned model M_u

```
M_u ← M
for each layer l in [L − σ, L] within M_u do
    for each v_{l,k} in G* do
        if v_{l,k} is equal to |Y| then
            Pruning w_l^k from the model.
return M_u
```

Table 4.1: Experimental settings in evaluating the performance of multi-classification.

Different settings	Original targets	Target that need to be unlearned	δ	σ
I	Bald, Mouth Slightly Open	Bald	0.1	5
II	Bald, Mouth Slightly Open	Mouth Slightly Open	0.1	5
III	Mouth Slightly Open, No Beard, Eyeglasses	Eyeglasses	0.08	5
IV	Smiling, No Beard, Eyeglasses	No Beard,Eyeglasses	0.1	5

4.3.5.1 Computational Overhead

The model training process is the most time-consuming element of machine learning. In the interests of simplicity, we set $f(\cdot)$ to represent the computational cost of the forward propagation, then the computational cost of one-step backpropagation is at most $5f(\cdot)$ [84].

Our computations primarily revolve around identifying important parameters, specifically in the Grad-CAM calculation process. This process will involve a single forward propagation and σ backpropagation for one target. Therefore, when selecting the important parameters of a target in the selected layer, it requires $6f(\cdot)$. Moreover, consider that we need to construct the balanced essential graph, and this process involves all unlearning targets and the remaining targets within the training dataset. Therefore, neglecting basic arithmetic operations, the complexity of our approach is approximately $6f(\cdot) * |Y|$. This value is much smaller than the cost of model training, which will involve multiple epochs and batch sizes, and it's about $\mathscr{N}_{epoch} * \mathscr{N}_{batch} *$ batch size $* 6f(\cdot)$.

4.3.5.2 Storage Overhead

The storage consumed by our target unlearning scheme primarily involves storing the results of Grad-CAM. We use $\mathbf{w}_l = \mathscr{R}^{O_l \times I_l \times K_l \times K_l}$ to denote one layer's parameters, where O_l, I_l and K_l denote the number of output channels, number of input channels, and the

Table 4.2: Target unlearning results for multi-classification (%).

		Original	Retraining	Chundawat et al. [29]	Ayush et al. [112]	Foster et al. [37]	Our
I	Bald	67.55	0.00	0.00	0.00	0.00	0.00
	Mouth Slightly Open	92.50	0.00	0.00	0.00	0.00	91.56
II	Bald	67.55	0.00	0.00	0.00	0.00	66.56
	Mouth Slightly Open	92.50	0.00	0.00	0.00	0.00	0.00
III	Mouth Slightly Open	90.19	0.00	0.00	0.00	0.00	90.07
	No Beard	96.11	0.00	0.00	0.00	0.00	95.78
	Eyeglasses	97.18	0.00	0.00	0.00	0.00	0.00
IV	Smiling	85.56	0.00	0.00	0.00	0.00	83.06
	No Beard	96.73	0.00	0.00	0.00	0.00	0.00
	Eyeglasses	97.43	0.00	0.00	0.00	0.00	0.00

Table 4.3: Computational cost associated with target unlearning for multi-classification (s).

Different settings	Graph Generation Cost	Unlearning Cost
I	34.24	0.03
II	33.21	0.03
III	123.13	0.03
IV	71.35	0.05

kernel size, respectively. Since we consider the importance of different channels in each layer, storing this importance will cost O_l for one target. Due to the need to consider all targets within the dataset and select multiple layers in the construction process of the balanced essential graph, our storage cost is approximately $\sigma * |Y| * O_l$, excluding basic arithmetic operations. Compared to the storage cost of model parameters, which is typically $L * O_l \times I_l \times K_l \times K_l$, our storage requirement is very small.

Summary: From the above analysis, the proposed unlearning schemes are determined to be efficient in terms of computational and storage costs, which demonstrates the practical potential and significant performance improvements obtained by our unlearning schemes.

4.4 Performance Evaluation

4.4.1 Experiment Setup

4.4.1.1 Model and Dataset

We choose three popular models, including AlexNet, VGG and ResNet, and use five widely used public image datasets: MNIST [2], CIFAR-10 and CIFAR-100 [3] and ImageNet ILSVRC2012 [4] and CelebA [5] to evaluate our scheme. The datasets cover different attributes, dimensions, and numbers of categories, allowing us to explore the unlearning utility of the proposed algorithm effectively.

4.4.1.2 Baseline Methods

To demonstrate the effectiveness of our scheme, we consider the following baseline schemes:

- Retraining from scratch [15]: The most straightforward way for machine unlearning involves retraining the model from scratch, after deleting the instances that need to be unlearned from the training dataset. Thus, for comparison, we retrain the model from scratch after deleting all instances that contain targets. Normally, the model retrained from scratch is the optimal unlearned model.

- Knowledge Distillation: Chundawat et al. [29] proposed a novel teacher-student framework with knowledge distillation for unlearning, where both competent and incompetent teachers transfer knowledge to a student model. This method leverages the incompetent teacher to unlearn specific data while retaining general knowledge through the competent teacher.

- Impair and Repair: Ayush et al. [112] presented impair and repair-based unlearning method. They first generated an error-maximizing noise matrix that contained highly influential patterns corresponding to the unlearning class. This matrix is then used to fine-tune model for unlearning. After impairing, further fine-tuning with the remaining data is performed to maintain model performance.

[2] http://yann.lecun.com/exdb/mnist/
[3] https://www.cs.toronto.edu/ kriz/cifar.html
[4] https://www.image-net.org/
[5] https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

- Pruning-Based: Foster et al. [37] proposed a unlearning method called Selective Synaptic Dampening (SSD). First, SSD used the Fisher Information Matrix (FIM) to identify parameters most influenced by the data to be forgotten. Then, SSD induces unlearning by dampening these identified parameters proportionally to minimize their impact, thereby unlearning the specified data while maintaining overall model performance.

For the retraining from scratch baseline mentioned above, we choose to remove all instances containing the target, rather than only removing the target labels or masking parts of the target-related pixels. The reasons are as follows:

- Simply removing the target labels still leaves instances containing information related to those targets. During the retraining process, the model could still learn from these target-related pixels, which does not meet the requirements of machine unlearning.

- Masking parts of the target-related pixels would require a deep model to automatically perform this task, which in turn means we would also need to remove the target-related information from this automatic model. Therefore, we choose the most straightforward way to remove all instances containing unlearning targets.

In Section 4.3.4, we construct a balanced graph to minimize the impact on the remaining data when unlearning targets. To illustrate the effectiveness of this strategy, we also consider the following methods for comparison.

- Unlearning within last-layer: For this setting, we only select and prune the important parameters within the last layer for unlearning targets.

- Unlearning without balance: For this setting, we select and prune the important parameters based on the essential graph without balance operation.

4.4.1.3 Metrics

Training models contain various randomness, especially for complex deep models with enormous training datasets. It is difficult to determine if the unlearning scheme has effectively eliminated the impacts of targets. For evaluating our scheme, we consider two aspects, including performance-based and attack-based metrics.

(a) Before (Person) (b) Before (Bus)

(c) After (Person) (d) After (Bus)

Figure 4.5: Target unlearning results for semantic segmentation.

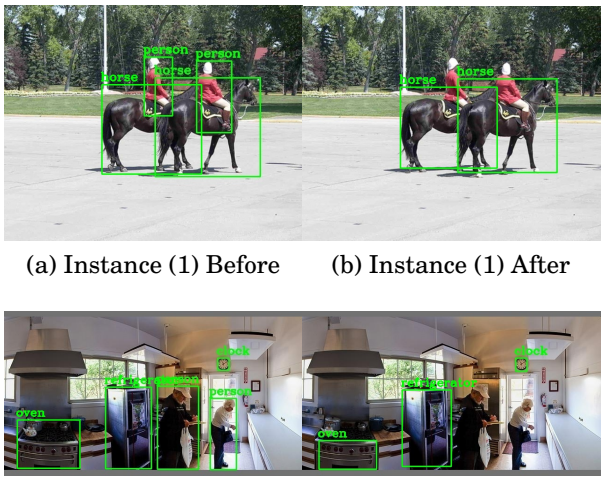
- **Performance-based**: Generally, trained models often exhibit high performance for training data. Therefore, the unlearning process could be verified by the performance of the model. For targets that need to be unlearned, if the performance ideally is the same as that of a model trained without seeing the unlearning targets, it can indicate that the unlearning process satisfies *effectiveness*. If model performance for remaining data is almost kept unchanged after the unlearning process, it means that the unlearning procedure achieves the *model utility* goal.
- **Attack-based**: The basic purpose of unlearning is to reduce the risk of sensitive information leakage. Therefore, certain attack methods can be used to directly and effectively verify the success of unlearning operations. Here, we use model inversion attack [38] and membership inference attacks (MIAs)[133] to evaluate our scheme. For MIAs, we utilize the recall = $\frac{TP}{Tp+FN}$ to represent the results of our evaluation.

4.4.2 Performance Analysis

To evaluate the effectiveness of our unlearning scheme, we do the following experiments:

4.4.2.1 Evaluating Performance of Multi-Classification Task

We first consider a multi-classification task based on CelebA dataset as a scenario containing multiple targets, where each class, such as nose, mouth, and ears, is considered



(a) Instance (1) Before (b) Instance (1) After

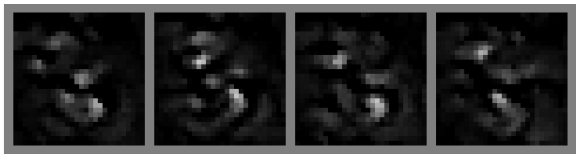
(c) Instance (2) Before (d) Instance (2) After

Figure 4.6: Target unlearning results for object detection.

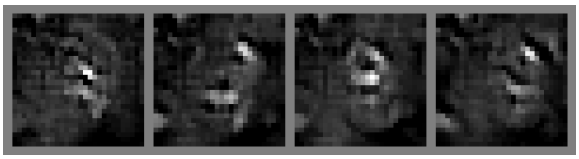
a different target. This configuration enhances the comprehension of the novelty. Each image in CelebA contains all targets, such as the nose and mouth. When unlearning a target, existing unlearning schemes, due to their coarse granularity, can only remove all instances to achieve unlearning goals, thereby affecting the performance of the model after unlearning. Our target unlearning scheme can effectively remove targets, such as mouth information while maintaining the model's ability to recognize other targets.

We choose the ResNet as our model and consider various unlearning settings as shown in Table 4.1, including scenarios involving the unlearning of one and two targets. For all original model training processes, we set epoch = 20, batch size = 128, and learning rate = 5e-06. We also compare our scheme with other four existing schemes, including retraining from scratch, knowledge distillation (Chundawat et al. [29]), impair and repair (Ayush et al. [112]) and pruning-base (Foster et al. [37]). Experimental results are shown in Table 4.2, while the corresponding computational cost is shown in Table 4.3.

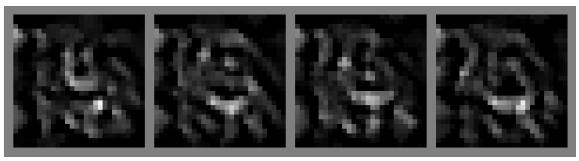
From all the results, our scheme can effectively unlearn all target-related information from the model, making the unlearned model unable to recognize the corresponding targets. For example, as illustrated in setting II, after the unlearning process, the accuracy of if the mouth is slightly open decreased from 92.50% to 0%. As shown in setting IV, our approach also remains effective for simultaneously unlearning two or more targets. However, for retraining from scratch, Knowledge Distillation (Chundawat et al. [29]), Impair and Repair (Ayush et al. [112]) and Pruning-Base (Foster et al. [37]). the unlearning results are quite poor. For example, as shown in setting II, when unlearning *mouth slightly open*, the performance of the remaining data also decreases to 0%. This is



(a) Original Model



(b) Fully Retraining



(c) Target Unlearning

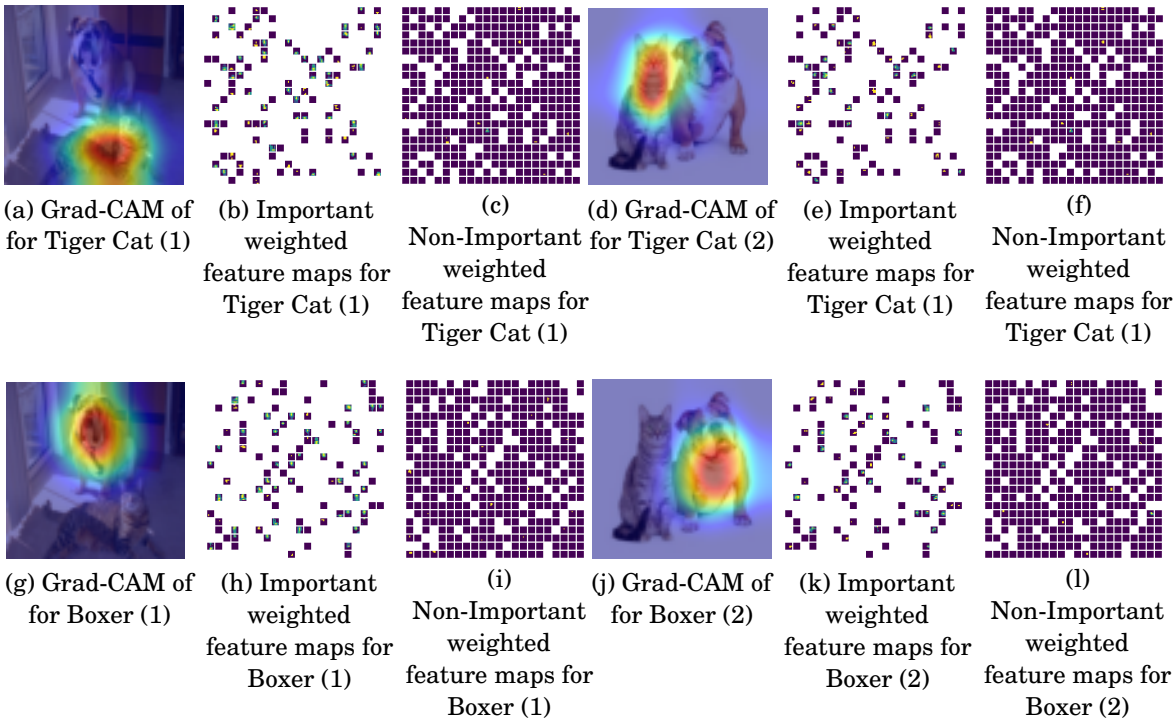
Figure 4.7: The results of model inversion attack.

due to the fact that all the above unlearning schemes lack a more fine-grained unlearning strategy, operating only at the instance level. When all instances contain this unlearning target, information related to other targets within that instance is also removed, leading to a decline in model performance. Take retraining from scratch as an example; when considering the unlearning of one target from the model, such as the information of bald, the only way of retraining from scratch is to remove all the instances containing bald and then retrain the model from scratch. In cases where all instances in the dataset contain the information that needs to be unlearned, such as in the CelebA dataset, this would result in the deletion of all instances, leaving no dataset available for model retraining. Consequently, the final model performance will be 0%.

From Table 4.3, it can be seen that the cost required for our scheme is very short, with all costs being less than 0.1s. Although the construction of the graph consumes more time, it is still much lower compared to retraining from scratch, which takes approximately 4 hours and 30 minutes. Moreover, the graph construction can be performed separately before unlearning, so it will not cost time in the unlearning process.

Table 4.4: The results of the MIAs.

	Resnet20 + CIFAR-10	Resnet20 + CIFAR-100
Original	95.62%	94.91%
Fully Retraining	0.00%	0.00%
Our Unlearning	0.00%	0.00%



(a) Grad-CAM of for Tiger Cat (1) (b) Important weighted feature maps for Tiger Cat (1) (c) Non-Important weighted feature maps for Tiger Cat (1) (d) Grad-CAM of for Tiger Cat (2) (e) Important weighted feature maps for Tiger Cat (1) (f) Non-Important weighted feature maps for Tiger Cat (1)

(g) Grad-CAM of for Boxer (1) (h) Important weighted feature maps for Boxer (1) (i) Non-Important weighted feature maps for Boxer (1) (j) Grad-CAM of for Boxer (2) (k) Important weighted feature maps for Boxer (2) (l) Non-Important weighted feature maps for Boxer (2)

Figure 4.8: The results of Grad-CAM and the distribution of important and non-important feature maps toward different targets in ImageNet ILSVRC2012.

4.4.2.2 Evaluating Performance of Semantic Segmentation and Object Detection Tasks

In addition to multi-classification, we also evaluate our target unlearning scheme for the semantic segmentation scenario, where we use the pre-trained model $deeplabv3\_resnet50$ in PyTorch and set the unlearning target to *person*. We set $\sigma = 26$, and $\delta = 0.09$ to construct the essential graph and don't execute the unlearning process in the last two convolutional layers. We also evaluate our target unlearning scheme in an object detection scenario, where we choose $yolov5s$ as our model, and set $\sigma = 18$, $\delta = 0.05$, unlearning target to *person*. Experimental results are shown in Figure 4.5 and Figure 4.6, respectively.

Results. For semantic segmentation in Figure 4.5, it can be seen that it is difficult for the semantic segmentation model to precisely recognize the person after the unlearning process. On the other hand, for bus, the unlearning operation essentially has no influence. For two results of object detection in Figure 4.6, similar results can be found. The object detection model can accurately detect objects such as horses, ovens, etc., while for person, it does not detect them.

Summary. The above experiments show that our scheme can achieve unlearning targets in different scenarios and can also effectively maintain model performance for all remaining tasks. In addition, it is more efficient since we achieve unlearning purposes based on only pruning the most critical parameters. For example, our scheme for multi-classification tasks based on CelebA only requires a few seconds, while the retraining process, usually takes multiple hours.

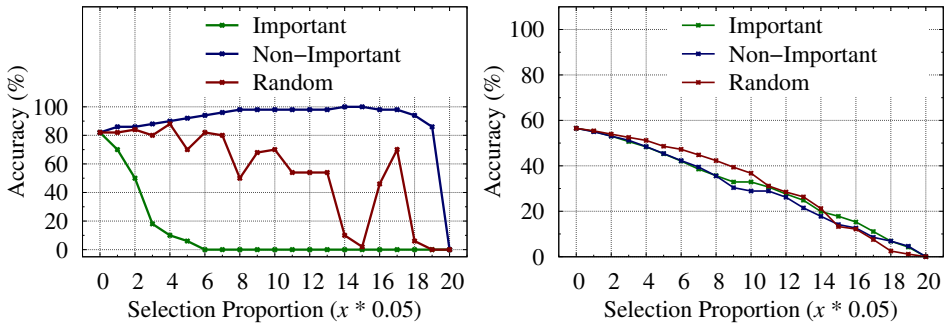
4.4.2.3 Evaluating Unlearning based on Model Inversion

To further analyze the effectiveness of our scheme, we construct the following experiment based on model inversion attack [38] and membership inference attacks [133]. Since those two schemes are typically used for class-level evaluation, we consider a classification model where each class is treated as a distinct target. We implemented the model inversion attack as described in [38]. We first train model ResNet18 based on the MNIST dataset with epoch = 50, batch size = 128 and learning rate = 0.1. We select the unlearning target as class 3 and consider other class data as the remaining data. After the training process, we select the critical parameters for unlearning class 3 and set the $\sigma = 5$, $\delta = 0.6$, then prune those critical parameters, followed by one epoch to recover model performance. Figure 4.7 illustrates the results.

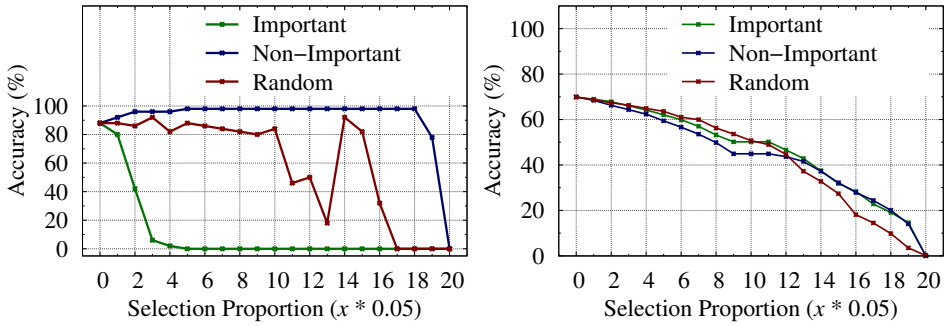
Results. As shown in Figure 4.7, fully retraining and our unlearning scheme produce dark and jumbled images since the model inversion attack has relatively little gradient information to rely on. As expected, pruning the critical parameters and fine-tuning prevent the inference of any meaningful information about the unlearning target. This suggests that our unlearning scheme almost removes the information about the unlearning target and eliminates the potential of inferring useful information via model inversion attacks.

4.4.2.4 Evaluating Unlearning based on Membership Inference Attack

Additionally, we use MIAs in paper [133] to evaluate whether the unlearning instances are still identifiable in the training dataset. We set the number of shadow models as 20



(a) Alexnet (Unlearning Target) (b) Alexnet (Remaining Data)



(c) VGG13 (Unlearning Target) (d) VGG13 (Remaining Data)

Figure 4.9: The model accuracy after pruning different types of parameters.

and the training epoch of the shadow model as 10, batch size = 64. The attack model is a fully connected network with two hidden linear layers of width 256 and 128, respectively, with ReLU activation functions and a sigmoid output layer. We evaluate our unlearning scheme with two settings, CIAFR-10 + ResNet20 and CIFAR-100 + ResNet20, and set unlearning class = 3. In the case of the CIFAR100 dataset, after sorting the model outputs, we choose the top 10 outputs as inputs for the attack model. We equally divide the training dataset into two subsets to generate the dataset based on shadow models and then train the attack model based on the output of those shadow models. After that, we set $\delta = 0.2$, $\sigma = 5$ and $\delta = 0.08$, $\sigma = 5$ to construct balanced essential graphs, respectively, and prune the critical parameters to unlearn target data. We set epoch = 5 to recover the model performance before MIAs. Table 4.4 shows the results.

As shown from Table 4.4, All MIAs have a high success rate for all original models; i.e., they can successfully derive the training dataset containing unlearning targets. However, the success rate of all MIAs is lower for all other approaches, indicating that MIAs cannot determine the existence of the unlearning targets after the unlearning process; this suggests that the influence of the unlearning target has been effectively

removed from the unlearned model.

Summary. The above experiments show that our scheme can effectively obtain critical parameters containing information about the unlearning targets, and reasonable pruning can reduce the probability of an attacker obtaining confidential information.

4.4.3 Feasibility Analysis

Our unlearning solution is based on the fact that the intermediate weights α_l^k in Equation 4.3 represent the importance of the corresponding channel's parameter to a target [64, 119]. To demonstrate the feasibility of our approach, we do the following experiments to verify that (1). given one instance, the distribution of intermediate weights is similar for all instances with the same target data; and (2). given the important parameters in a layer of a target, a reasonable disturbance to those parameters will destroy the performance of this target data. Here, the distribution of intermediate weights means the set of important parameters and non-important parameters for a specific target. Object (1) ensures that all instances with the same target will be influenced by the same important parameters, which guarantees that the subsequent disturbances will influence the same parameters to unlearn the target. Object (2) ensures the feasibility of achieving unlearning based on the selected important parameters.

4.4.3.1 Identity Analysis

To analyze if the distribution of intermediate weight is similar for all instances containing the same targets, we show the distribution of weighted feature maps of the final convolutional layer using the hook technique. In particular, we separately random select two instances from ImageNet ILSVRC2012, and compute the Grad-CAM toward the final layer *layer4.1.conv2* in ResNet18 model using pre-trained weights in PyTorch. During the calculation of GradCAM, we also record the intermediate weights α^k of each feature map, and then calculate the results of $\alpha^k \times f^k$. We divide the top 20% of the results of $\alpha^k \times f^k$ based on the value of α^k as important weighted feature maps and take the remaining 80% as non-important weighted feature maps. We separately plot the above two types of weighted feature maps and maintain the position of each feature map. Ideally, if the index of selected important weighted feature maps of two instances are similar, it means that the distribution of α^k is similar.

Results. Figure 4.8 illustrate the results of our experiment. Figure 4.8a and Figure 4.8d represent the Grad-CAM of *Tiger Cat*, while Figure 4.8g and Figure 4.8j show

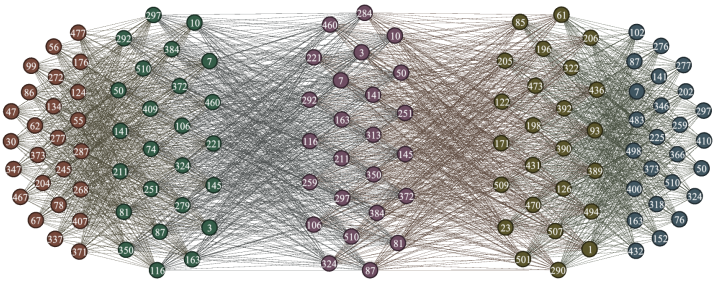
Table 4.5: Experimental settings in evaluating the essential graph with balance.

Different settings	Original targets	Target that need to be unlearned	δ	σ
I	Smiling, No Beard, Eyeglasses	No Beard	0.1	6
II	Mouth Slightly Open, No Beard, Eyeglasses	No Beard	0.25	7
III	Mouth Slightly Open, No Beard, Wearing Hat	Mouth Slightly Open	0.18	7
IV	Smiling, No Beard, Wearing Hat	Smiling	0.3	7

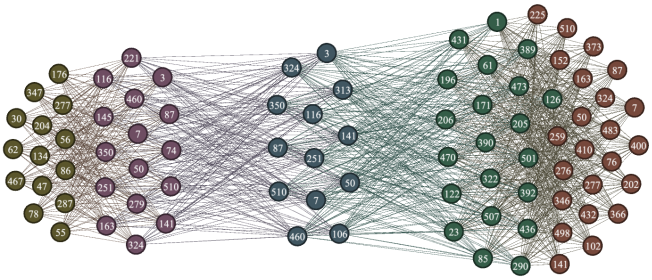
the Grad-CAM of *Boxer*. The other subplots show the index of importance and non-importance weighted feature maps. We can see from the Figure 4.8b and Figure 4.8e that for *Tiger Cat* (1) and *Tiger Cat* (2), the index of the weighted important feature maps is almost same, and from the Figure 4.8c and Figure 4.8f, the index of non-important weighted feature maps is also nearly identical. That is to say, the important parameters within one layer are almost the same for those two different instances with the same target data. This indirectly suggests that the distribution of those two intermediate weights is almost identical. The same results can be derived from Figure 4.8h and Figure 4.8k. In addition, by comparing Figure 4.8b and Figure 4.8h (Figure 4.8e and Figure 4.8k), we can also find that the distribution of intermediate weights generated from two instances with various target labels are different. At the same time, there is some overlap between Figure 4.8b and Figure 4.8h (Figure 4.8e and Figure 4.8k), this suggests that different targets may use the same parameters to analyze instances and give the prediction of those targets. Therefore, it is necessary to consider the model performance of the remaining data when executing the unlearning process.

4.4.3.2 Important Parameter Analysis

To verify if a reasonable disturbance to those important parameters will destroy the performance of target data. We further measure the accuracy of one unlearning target and the remaining data after the unlearning process, respectively. Specifically, we first select two different models with pre-trained weights in PyTorch, including AlexNet, and VGG13, to calculate the intermediate weights based on Grad-CAM in the last layer toward the unlearning target *Tinca* in ImageNet ILSVRC2012. Then, we sort those intermediate weights from large to small and select the top large weights for different proportions. We also compare two other weight selection schemes, including random and non-important selection, where for non-important selection, we select the smallest intermediate weights. After that, we prune model parameters corresponding to the index of these selected weights and record the accuracy. Figure 4.9 illustrates the results of our experiment.



(a) Essential Graph without Balance **Graph**



(b) Essential Graph with Balance (**Graph** (β))

Figure 4.10: Different essential graphs.

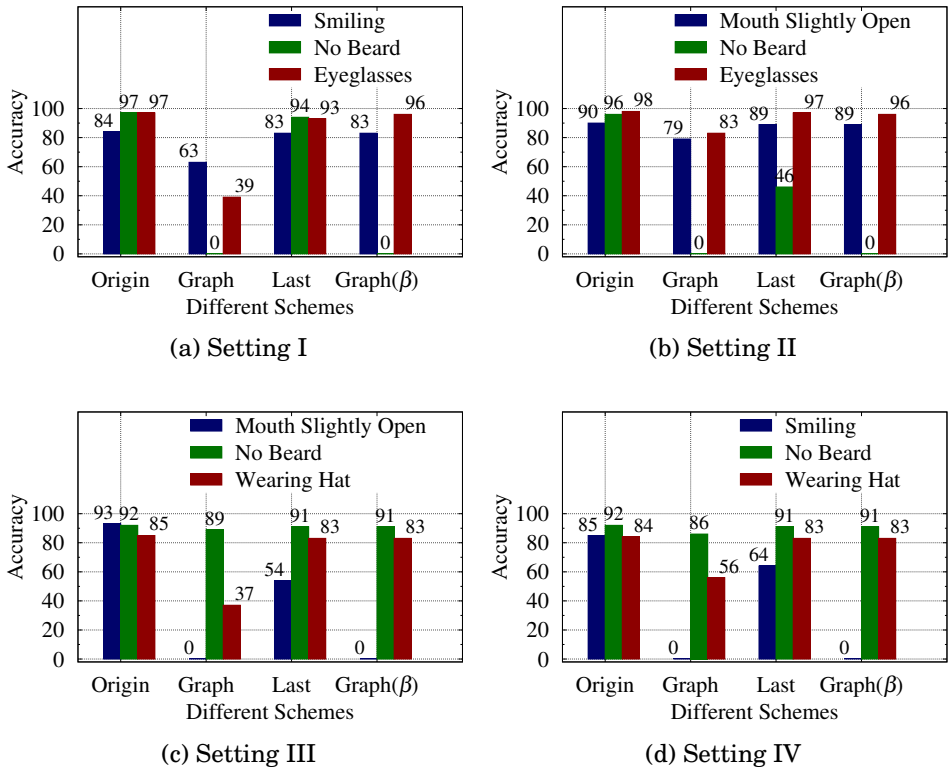
Results. In Figure 4.9, Figure 4.9a and Figure 4.9b show the accuracy of target that need to be unlearned and remaining data of AlexNet, while Figure 4.9c and Figure 4.9d show the results of VGG13. The y-axis denotes the accuracy, while the x-axis represents the different proportions of the selected intermediate weights. The x-axis also indicates how much the proportion of parameters we prune. From all results, we find that as we prune more model parameters, the accuracy of unlearning target and remaining data gradually decreases. This is because as more parameters are pruned, the model contains progressively less information about those data, which will lead to a decrease in accuracy. It is worth noting that in the front part of the x-axis for the important selection scheme, that is $x \in (0:6]$, the accuracy of unlearning target decreases significantly, while in the remaining part of the x-axis, the accuracy remains almost zero. This indicates that some model parameters are only associated with unlearning targets. For the non-important or random selection scheme, the performance for the unlearning target is almost unchanged in the front part of the x-axis, which indicates that these two methods cannot select the parameters that affect the unlearning target.

Summary. The above experiments have shown that the distribution of the most influential parameters for the same target is similar. Reasonable pruning to those selected influential parameters will affect the model performance for those target data without affecting the remaining data, which provides an effective unlearning solution.

4.4.3.3 Essential and Balance Graph Analysis

Some of the parameters that affect the unlearning target may also be important for the remaining data. Directly removing information contained in those parameters that are important for the unlearning target may simultaneously affect the model performance for the remaining data. In Section 4.3.4, we construct the graph to balance unlearning effectiveness and model performance. To evaluate the validity of the balanced essential graph, we construct the following experiments. We use the pre-trained model ResNet18 in PyTorch and set unlearning target label = 0, $\sigma = 5$, and set $\delta = 0.05$ to construct the essential graph based on the unlearning target, which is denoted as **Graph**. We also construct the balanced essential graph while simultaneously considering the remaining data and unlearning data, which we denote as **Graph** (β). Figure 4.10 shows the results of our experiment, where nodes with the same color denote each layer of the model, and nodes in each partition represent the corresponding channels. For example, the rightmost partition in Figure 4.10a represents the last layer in ResNet18, while the node 7 in this partition indicates the corresponding channel 7 in the last layer.

Results. In Figure 4.10, Figure 4.10a shows the essential graph constructed by unlearning target data only, while Figure 4.10b shows the essential graph constructed by balancing the impact of the remaining data. There are two main differences between these two graphs. First, some nodes exist in Figure 4.10a will disappear in the corresponding partitions in Figure 4.10b. For example, the node 297 in the last partition in Figure 4.10a. In the last partition in Figure 4.10b, node 297 doesn't exist. This indicates that parameters in one layer of the model that are important for the unlearning target may also be important for the remaining data. During the construction of the essential graph with balancing the impact of the remaining data, those parameters that are also important for the remaining data will be discharged. Second, the number of nodes owned by each color partition in Figure 4.10a is similar, while in Figure 4.10b, the number of nodes in each partition is different, with fewer nodes on the left. The reason for this is that the lower layers of the model are usually used to extract features, while the parameters at the higher levels of the model are used to distinguish between different targets. Lower-level features always have a higher probability of being used by the remaining data. When constructing the essential graph in Figure 4.10b, since we also consider the performance of the unlearned model for the remaining data, it will select a smaller number of important nodes to avoid degrading the performance of the model.



(a) Setting I (b) Setting II

(c) Setting III (d) Setting IV

Figure 4.11: The results of evaluating balance graph.

4.4.3.4 Evaluating Balance Graph

In this Section, we evaluate whether the balanced essential graph can maintain the performance of the remaining data while ensuring the effectiveness of the unlearning. As shown in Table 4.5, we consider four different settings. For all original model training processes, we set epoch = 20, batch size = 128, and learning rate = 5e-06. We compare three unlearning schemes, including only executing the unlearning process in the last layer (**Last**), unlearning based on the essential graph without balance (**Graph**) and unlearning based on the essential graph with balance (**Graph** (β)). The results are shown in Figure 4.11.

Results. As can be seen in Figure 4.11, when only considering the unlearning process in the last layer (**Last**), the performance of the unlearned model for targets that need to be unlearned hardly decreases to 0, indicating that the model still contains some information about those targets. For the unlearning scheme based on the graph without balance (**Graph**), the accuracy of unlearning target data is completely reduced to 0. However, it does not meet the requirements for the utility of the remaining data. While the parameters associated with the unlearning target are pruned, those parameters

that are also important for other targets in the constructed essential graph are also pruned, which will destroy the performance of the remaining data. For **Graph (β)**, it can maintain the performance of the model after the unlearning while ensuring the effectiveness of the unlearning.

Summary. The above experiments demonstrate the effectiveness of the balanced essential graph. It is able to select the important parameters while controlling the distribution of the selected parameters as well, reducing model performance degradation due to over-selection of important parameters.

4.4.4 The Effect of Hyper-Parameters

Setup. The hyperparameter σ denotes the number of layers that are considered to execute the unlearning process, while the δ determines how many parameters are selected when constructing the graph. To evaluate both hyperparameters, we set the original targets as smiling, no beard, and eyeglasses in CelebA, and we set the unlearning target as smiling. For the original model training processes, we set epoch = 20, batch size = 128, and learning rate = 5e-06. Then, we, respectively, choose different values of σ and δ to construct the essential graph with balance and without balance. Then, we evaluate the accuracy of smiling and eyeglasses after pruning the most critical parameters. Figure 4.12 shows the results.

Results. As seen from all Figures, when keeping δ constant and increasing σ, the number of layers to be pruned will increase, which directly accelerates the unlearning process. Similar results can also be observed from each Figure when setting σ to the same value and increasing δ. It's worth noting that as δ increases, the accuracy of the remaining targets will decrease when constructing graphs without balance. For example, in Figure 4.12c, when setting δ = 0.1 and σ = 5, the model performance for eyeglasses decreases to 81%. On the other hand, even when selecting a larger number of δ and σ, the accuracy of the remaining targets does not decrease for all schemes with balance operations. This is because when constructing **Graph (β)**, we simultaneously consider the performance of the model for the remaining data, which can alleviate the performance decrease for the remaining data. Based on the above results, our unlearning scheme can achieve unlearning effectiveness while ensuring model usability even when both hyperparameters are set too high. Therefore, in general, we can opt for appropriately large hyperparameters to achieve effective unlearning purposes.

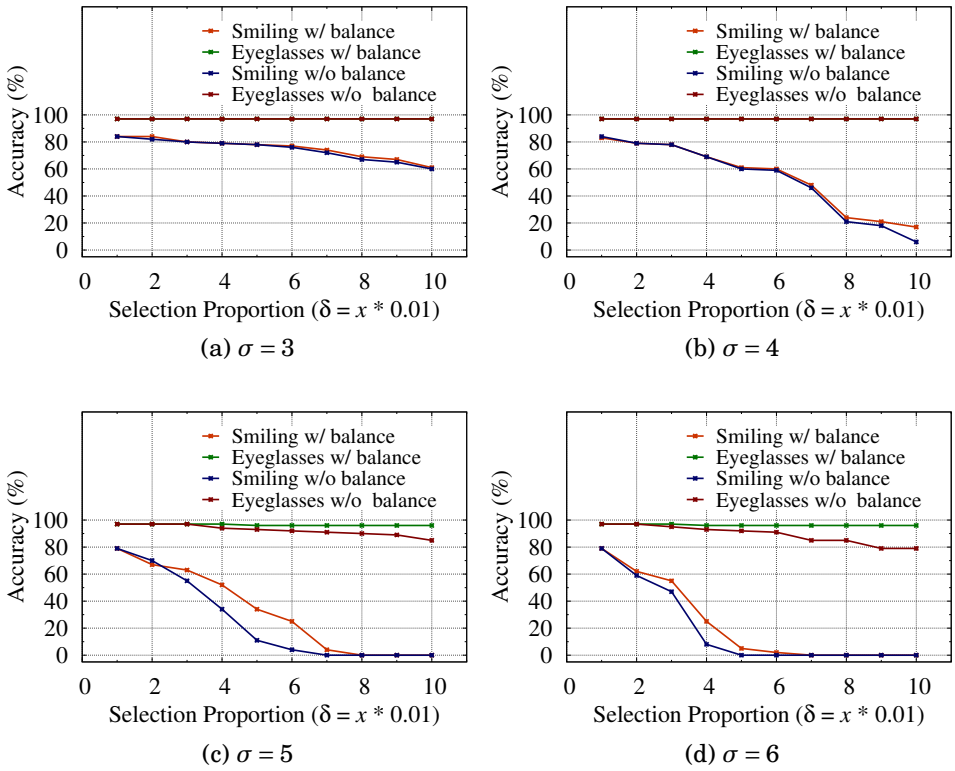


Figure 4.12: The effect of hyper-parameters.

4.5 Summary

This chapter proposed a novel machine unlearning scheme that can selectively remove partial target information from the trained model, named *target unlearning*. As a solution, we have defined the concept of target unlearning and illustrated the challenges of this unlearning problem. We also analyzed the most influential parameters of a model for the given target based on the explainable technique and proposed a pruning-based unlearning method to erase the information about the target. To balance the performance of the unlearned model, we constructed an essential graph to describe the relationship between all important parameters within the model, and simultaneously filter those important parameters that are also important for the remaining data. The experimental results demonstrate that under our scheme, the model can remove the impact of targets and ensure the accuracy of the remaining data in a quick and efficient manner.

This chapter inspires us that we can explore ways to extend and/or modify the current method of evaluating channel important and develop new target unlearning schemes based on this revised evaluation. In addition, we plan to design a more powerful

scheme that supports unlearning requests from Natural Language Processing (NLP) or Generative Adversarial Networks (GAN), combined with other technologies such as information theory.

CHAPTER 5

DON'T FORGET TOO MUCH: TOWARDS MACHINE UNLEARNING ON FEATURE LEVEL

Machine unlearning enables pre-trained models to remove the effect of certain portions of training data. Previous machine unlearning schemes have mainly focused on unlearning a cluster of instances or all instances belonging to a specific class. These types of unlearning might have a significant impact on the model utility; and they may be inadequate for situations where we only need to unlearn features within instances, rather than the whole instances. Due to the different granularity, current unlearning methods can hardly achieve feature-level unlearning. To address the challenges of utility and granularity, we propose a refined granularity unlearning scheme referred to as "feature unlearning" [1]. We first explore two distinct scenarios based on whether the annotation information about the features is given: feature unlearning with known annotations and feature unlearning without annotations. Regarding unlearning with known annotations, we propose an adversarial learning approach to automatically remove effects about features. For unlearning without annotations, we initially enable the output of one model's layer to identify different pattern features using model interpretability techniques. We proceed to filter features from instances based on these outputs with identifying ability. So that we can remove the feature impact based on filtered instances and the fine-tuning process. The effectiveness of our proposed approach is demonstrated through experiments

[1]The main content of this chapter has been published in Heng Xu, Tianqing Zhu, Wanlei Zhou, Wei Zhao. Don't Forget Too Much: Towards Machine Unlearning on Feature Level, IEEE Transactions on Dependable and Secure Computing, 2024.

involving diverse models on various datasets in different scenarios.

5.1 Introduction

Current machine unlearning schemes also mainly focus on removing the effect at either the instance or class level. Instance-level unlearning approaches deal with the request to unlearn information related to an individual or a group of instances [11, 19, 24, 46], while class-level unlearning methods involve eliminating the influences associated with all instances from a particular class [9, 44, 106, 120]. However, many scenarios require unlearning at a feature level, in which the model owner only removes a particular feature from the model. In the context of an image-based trained model, a feature refers to specific patterns or attributes extracted from the input images that are used to understand and differentiate between different visual elements. For instance, consider features, such as age, gender, or skin color, which could be integrated across all instances in one face recognition model. The machine unlearning schemes at the instance level have to unlearn all instances containing those features, which is unpractical [123]. Conversely, feature-level unlearning is urgently needed.

There are very many applications of feature unlearning, such as removing unnecessary image features or removing unwanted adversarial pattern features. Furthermore, the feature unlearning technique can serve as an alternative solution for alleviating model bias. As shown in Figure 5.1, in each sub-figure, elements of different shapes represent different features; and different colors represent different feature annotations. Figure (a) shows that the inconsistent frequency of features (triangles) may lead to unfairness. For example, instances possessing the yellow triangle feature will have a greater likelihood of being classified as class 2, while instances possessing the blue triangle would be classified as 1 [79, 122]. Fairness solutions typically aim to mitigate bias by equalizing the frequency of biased features across different classes, with the goal of transforming these features into normal features. Since these features do not play vital roles for classification, we can eliminate the bias by directly unlearning those features and retaining the remaining features for the model classification (as shown in Figure (b)).

However, compared with instance or class-level unlearning, unlearning feature, especially for unstructured data, is more challenging. We have identified three distinct challenges if achieving feature unlearning.

- Firstly, a single instance may contain multiple tightly embedded features. The

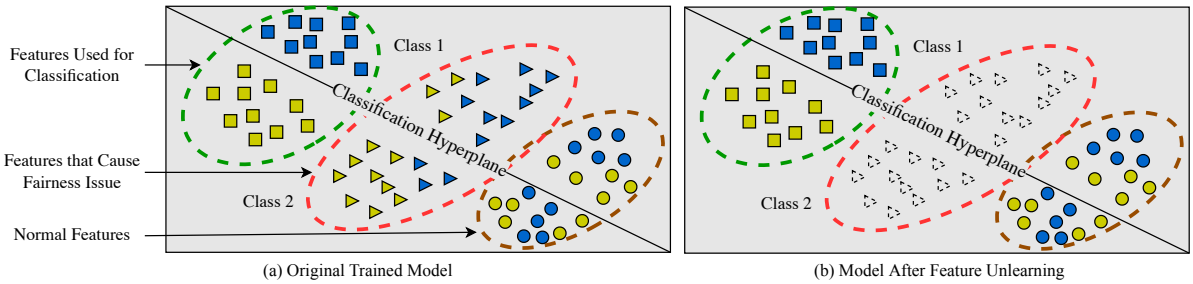


Figure 5.1: Feature unlearning.

interactions between those multiple features and the model's processing across different layers are quite complex, making it difficult to eliminate the influence of a specific feature without negatively impacting other features.

- Second, in many cases, we don't know the value of features, which we define as the *annotation*. For example, an annotation of a skin color feature might be white. The unknown annotation means that the dataset does not provide any information about what these features look like. In a trained gender recognition model, the training dataset may only give information about the gender of each instance, without providing explicit information about their skin color. In this situation, unlearning skin color needs to remove the relevant features without any explicit feedback from the dataset, which is challenging.

- Third, effectively evaluating the results of feature unlearning is a significant challenge. Current instance-level evaluation metrics, such as accuracy-based or membership inference attack-based [30, 49, 137], are unsuitable for assessing feature-level unlearning due to diverse granularity of unlearning targets.

Unlearning without known annotation inherits from the known setting with more practical assumptions and can be used in more realistic scenarios. Taking the fairness problem as an example again, current approaches often rely on adversarial fine-tuning to address fairness [122]. However, in the absence of annotations, this approach becomes ineffective. In addition, current unlearning methods usually rely on data segmentation [11, 24] or influence functions [41, 46] techniques, aiming to effectively partition the dataset to speed up model retraining or compute the effects of individual features, which cannot tackle above challenges.

This chapter introduces two feature unlearning schemes: feature unlearning with known annotations and feature unlearning without annotations. Based on this categorization, we propose two fine-tuning-based methodologies to address the above unlearning

scenarios respectively. The fine-tuning scheme enables the model to quickly unlearn previously learned knowledge when undergoing fine-tuning for a new task [44, 106]. It is more efficient compared to the retraining-based unlearning strategies [11, 24]. In our schemes, feature unlearning with known annotations uses adversarial training, and unlearning without annotations uses filtered instances to fine-tune the model.

To tackle the first challenge, in unlearning with known annotations, where we have annotations associated with the features to be unlearned, we employ adversarial training to fine-tune model. This technique automatically identifies and separates the feature information, allowing us to retain as many task-specific features as possible while removing the unlearning features. In the case of unlearning without annotations, we enable the output of one CNN model's layer to identify and separate distinct pattern features. This is achieved via model interpretability techniques.

To address the second challenge in feature unlearning without annotations, we introduce a novel loss term [105] that encourages one CNN model's layer to identify distinct pattern features. To broaden the identification range, we incorporate the eigengap heuristic, allowing for identifying more features while maintaining the desired effect. By identifying various feature information, we encode the instance to filter out the feature information that needs to be unlearned. Subsequently, we fine-tune the model to achieve the unlearning purpose based on those encoded instances. In contrast, when dealing with unlearning with known annotations, where the annotations are already known, we directly unlearn the features using adversarial learning.

To tackle the third challenge, we evaluate the effectiveness of feature unlearning using the correlation between feature information and model task, as well as the variations of model accuracy. In addition, we qualitatively evaluate our two unlearning schemes by using Guided backpropagation [111] and visualizing whether our model has been fine-tuned on the revoked features in our scheme.

In summary, we make the following contributions:

- We tackle the machine unlearning problem at the feature level and formally define two types of machine unlearning requests: feature unlearning with known annotations and feature unlearning without annotations.

- We introduce adversarial learning techniques to facilitate the feature unlearning with known annotations while keeping useful feature information for remaining model performance.

- We introduce model interpretability to empower the output of one model's layer to decouple and identify various pattern features. These outputs, equipped with identifying capabilities, are then employed to facilitate unlearning without annotations.

- We evaluate our feature unlearning from both quantitative and qualitative perspectives. In addition to evaluating based on accuracy, we also propose two new methods, including variation in accuracy and a gradient visualization-based scheme.

5.2 Preliminaries

In the context of machine unlearning, we define the subset $\mathscr{D}_u \subset \mathscr{D}$ as a portion of the training dataset, whose influence we want to remove from the trained original model $M = \mathscr{A}(\mathscr{D})$, where $\mathscr{A}()$ is the training process. Conversely, the complement $\mathscr{D}_r = \mathscr{D}_u^{\complement} = \mathscr{D}/\mathscr{D}_u$ represents the dataset we intend to preserve those contributions within the model. Other important symbols that appear in this chapter and their corresponding descriptions are listed in Table 3.1.

With these definitions, we give the definition of machine unlearning for instance-level requests.

Definition 15 (Machine Unlearning [15])**.** *Let's define the unlearning process as $\mathscr{U}(M, \mathscr{D}, \mathscr{D}_u)$, which aims to remove a cluster of instances $\mathscr{D}_u$ from the pre-trained model $M = \mathscr{A}(\mathscr{D})$. This unlearning process is a function that takes the pre-trained model M, the training dataset $\mathscr{D}$, and the unlearning dataset $\mathscr{D}_u$ as inputs and outputs a new model M_u. The objective of the unlearning process is to ensure that the unlearned model performs as if it had never seen the instances in the unlearning dataset.*

If $\mathscr{D}_u$ in the above definition denotes all instances within a class, the above definition can be seen as class-level unlearning. However, class-level or instance-level unlearning schemes cannot handle feature-level unlearning requests. Feature unlearning focuses on the unlearning request on the feature level. Bau et al. [8] established a classification of six different types of semantics features within Convolutional Neural Networks (CNNs): objects, parts, scenes, textures, materials, and colors. Specifically, the first two categories can be broadly regarded as patterns related to objects characterized by specific shapes. On the other hand, the remaining four categories can be grouped as attribute patterns lacking distinct contours [139].

In this chapter, we consider unlearning those two types of features, named *pattern feature* and *attribute feature*. For example, in the case of face recognition scenarios, the images typically contain various features, including but not limited to pattern features, such as the shape of the nose and the state of the mouth, and attribute features, such as male and youth. To represent the training dataset, we use $\mathscr{D} = \{(\mathbf{x}_1, y_1, \mathbf{f}_1), (\mathbf{x}_2, y_2, \mathbf{f}_2), \ldots, (\mathbf{x}_n, y_n, \mathbf{f}_n)\} \subseteq \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^k$, where each $\mathbf{x}_i \in \mathscr{X}$ represents an individual instance, $y_i \in \mathscr{Y}$ is the corresponding label, and n represents the size of the dataset $\mathscr{D}$. Unlike the general dataset definition, we define an extra item $\mathbf{f}_i$ in the dataset $\mathscr{D}$. $\mathbf{f}_i$ represents the feature annotations of this instance. Consider one instance $\mathbf{x}_i$ that consists of multiple features. The annotations of those features can be denoted as $\mathbf{f}_i = \{\mathrm{f}_1, \mathrm{f}_2, \ldots, \mathrm{f}_k\}$. For example, in the above face recognition scenarios, if the shape of the nose is described as whether it is pointy, f_{pointy_nose} = True if it is, and f_{pointy_nose} = False if it is not.

Now, we give the definition of feature unlearning.

Definition 16 (Feature Unlearning)**.** *Given one specific feature, the annotation is denoted as* f_i*. We want to remove all effects of this feature from the trained model. Feature unlearning process* $\mathscr{U}_f(M, \mathscr{D}, \mathrm{f}_i)$ *is defined as a function from an already-trained model* $M = \mathscr{A}(\mathscr{D})$*, a training dataset* $\mathscr{D}$*, and a specified feature* f_i *to a model* M_u*, which ensures that the unlearned model* M_u *performs as though it had never seen the unlearning feature* f_i *in all training dataset.*

5.3 Methodology

In this section, we first present an unlearning scheme based on adversarial training techniques. It is simple but effective in simultaneously unlearning multiple features with known annotations and maintaining the model's performance for the original task (Section 5.3.1). Moreover, we address the requests of unlearning features in scenarios where annotations are unknown and propose an unlearning method based on model interpretability and fine-tuning (Section 5.3.2 and Section 5.3.3). These approaches cater to both with known and without annotations cases, with adversarial learning for the former and interpretability-guided for the latter.

5.3.1 Feature unlearning with known annotations

In cases where the annotations of the features that need to be unlearned are known, we employ adversarial learning to fine-tune the original model and remove unlearning features automatically[2]. As shown in Figure 5.2, our scheme mainly consists of two sub-processes that run alternately with each other. *Remover training process* trains the *remover*, whose purpose is to output a mask for filtering unlearning features and remaining useful features for original task. We classify features in each instance into two categories: *target features* and *task features*. Target features denote those features that are required to be unlearned, whereas task features encompass the ones linked to the task of the original model. As an example, in a scene recognition model, the features concerning the scene itself are considered task features, whereas the gender-related features that need to be unlearned are categorized as target features. We can further categorize target features according to whether it is pattern feature or attribute feature. The purpose of *fine-tuning process* is to fine-tune the model based on the masked instances to achieve unlearning purpose.

5.3.1.1 Remover training process

This process contains three main parts: *remover*, *original* and *adversary* models. The upper right portion is the original model M, within which the feature unlearning operation needs to be performed, i.e., we need to remove the specified features from that model. Let $L(M(\mathbf{x}_i), y_i)$ denote the loss of the original model:

$$\begin{aligned} L_M &= \sum L(M(\mathbf{x}_i), y_i) \\ s.t. &(\mathbf{x}_i, y_i) \in \mathscr{D} \end{aligned} \tag{5.1}$$

The adversary C is the lower right portion. It aims to predict whether the output instance $\hat{\mathbf{x}}_i$ from remover based on an input instance $\mathbf{x}_i$ contains target feature information. The adversary's objective is to minimize a loss that quantifies the amount of information about the target feature it can extract from $\hat{\mathbf{x}}_i$:

$$\begin{aligned} L_C &= \sum L(C(\hat{\mathbf{x}}_i), \mathrm{f}_i) \\ s.t. &(\mathbf{x}_i, \mathrm{f}_i) \in \mathscr{D} \end{aligned} \tag{5.2}$$

where the adversary C, takes the outputs $\hat{\mathbf{x}}_i$ from the remover E as its input, alongside the inputs $\mathbf{x}_i$ for remover. f_i represents the annotation for target feature of $\mathbf{x}_i$.

[2]In this chapter, we use the term *unlearning feature* to denote the feature that needs to be unlearned from the model.

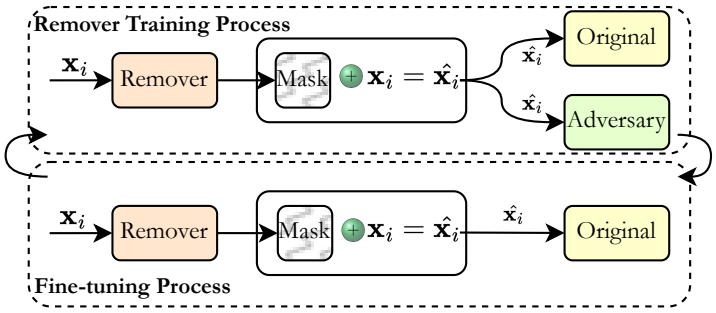


Figure 5.2: Feature unlearning with known annotations.

Remover E, encompasses an encoder and decoder framework designed to generate a mask. The purpose of this mask is to filter target features, while preserving task features as much as possible. In addition to this, to make the before and after filtered instances more similar, we introduce an extra loss term weighted by the parameter β for remover:

$$\begin{aligned} L_E &= \sum_i \left[\beta |\mathbf{x}_i - \hat{\mathbf{x}}_i|_{\ell_1} + L(M(\hat{\mathbf{x}}_i), y_i) - \lambda L_C\right] \\ &s.t. (\mathbf{x}_i, \mathrm{f}_i, y_i) \in \mathscr{D} \end{aligned} \tag{5.3}$$

The primary objective of the first term in Equation 5.3 is to maintain a higher amount of information that already exists in the original instances. This helps to ensure that the instances after masking are as similar as possible to the original instance. The second term focuses on preserving a greater extent of task-specific feature information, which refers to task features. On the other hand, the third term eliminates the intended target features.

It is worth highlighting that we can address the scenario of unlearning multiple features simultaneously within this framework. That is, the number of feature f_i to be unlearned can be arbitrary. When only one feature is considered to be unlearned, the number of f_i can be one. For example, the gender feature information is unlearned in the scene recognition model. We can also unlearn multiple features at the same time. In such cases, we can use the following equation to evaluate the multi features information:

$$\begin{aligned} L_C &= \sum L(C(\hat{\mathbf{x}}_i), (\mathrm{f}_1, \mathrm{f}_2, .., , \mathrm{f}_m)) \\ &s.t. (\mathbf{x}_i, (\mathrm{f}_1, \mathrm{f}_2, .., , \mathrm{f}_m)) \in \mathscr{D} \end{aligned} \tag{5.4}$$

where $(\mathrm{f}_1, \mathrm{f}_2, .., , \mathrm{f}_m)$ represent multi features that need to be unlearned. m is the number of those features.

Algorithm 7: Unlearning with known annotations

Input: The full training set $\mathscr{D}$, original model M, remover E, adversary C, total iteration number T.

Output: model M_u after unlearning process

1 Training adversary C based on the dataset $(\mathbf{x}_i, \mathrm{f}_i) \in \mathscr{D}$
2 **for** $t = 0; t < T; t++$ **do**
3 **if** i % 2 == 0 **then**
4 Setting require_grad = *False* of all parameters from original model M and adversary C.
5 Setting require_grad = *True* of all parameters from remover E.
6 **for** $(\mathbf{x}_i, \mathrm{f}_i, y_i) \in \mathscr{D}$ **do**
7 Input $\mathbf{x}_i$ to remover E and get $\hat{\mathbf{x}}_i$.
8 $L_C = \sum L(C(\hat{\mathbf{x}}_i), \mathrm{f}_i)$.
9 $L_M = \sum L(M(\hat{\mathbf{x}}_i), y_i)$.
10 Updating remover E based on $L_E = \sum_i \left[\beta |\mathbf{x}_i - \hat{\mathbf{x}}_i|_{\ell_1} + L_M - \lambda L_C\right]$.
11 **else**
12 Setting require_grad = *False* of all parameters from remover E and adversary C.
13 Setting require_grad = *True* of all parameters from original model M.
14 **for** $(\mathbf{x}_i, \mathrm{f}_i, y_i) \in \mathscr{D}$ **do**
15 Input $\mathbf{x}_i$ to remover E and get $\hat{\mathbf{x}}_i$.
16 Updating original model M based on $L_M = \sum L(M(\hat{\mathbf{x}}_i), y_i)$.
17 **return** $M_u = M$

5.3.1.2 Fine-tuning Process

After one iteration of the remover training process is performed, we alternately fine-tune the model based on the filtered instance to gradually achieve the unlearning purpose:

$$L_M = \sum L(M(\hat{\mathbf{x}}_i), y_i) \tag{5.5}$$

After several iterations, target features will be unlearned from the original model. Algorithm 7 illustrates the whole unlearning process.

In line 1, we first train the adversary model to give it the ability to detect whether the specific feature information in an instance. Then, in lines 2-10, we train the remover E to remove target features while retaining the relevant task features for the original model task. Specifically, in lines 4 to 5, we fix the parameters of the original model M and adversary C and only update remover E. In lines 6-10, we update the remover based on L_E. Alternately, as indicated in lines 12 to 16, we also fine-tune the original model

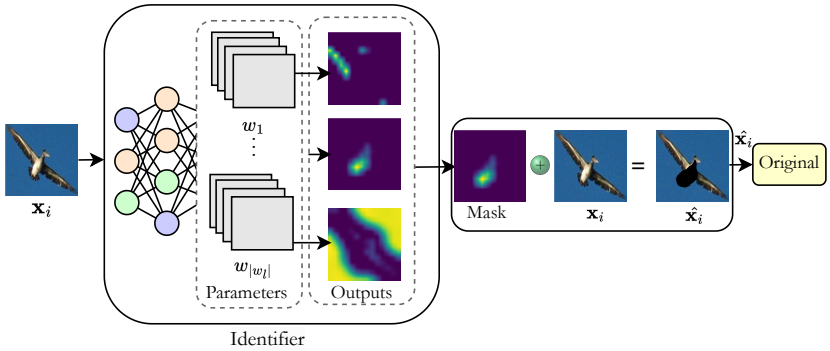


Figure 5.3: Feature unlearning without annotations.

based on instances generated by the remover. This effectively unlearns information about target features due to the presence of catastrophic forgetting [44, 120].

5.3.2 Feature unlearning without annotations

In the above-mentioned section, if we have annotations for the features that need to be unlearned, we can use adversarial training to achieve unlearning purpose. However, the adversarial training method cannot be applied to unlearn features if we lack annotations. This is because we cannot use the adversary model to remove features from instances.

Identifying and unlearning various features without annotations becomes extremely challenging. This task involves three main challenges. Firstly, how do we identify which features are present in the instance and can be unlearned? Secondly, how do we obtain more diverse features to meet different unlearning requirements? Thirdly, how do we unlearn feature information from the model when we know these features? For the purpose of identifying various features, we modify a model so that its output of the middle layer can be used as an encoder to filter feature information. The modified model, we call it *identifier model*. In this section, we illustrate how to achieve unlearning features without annotations based on the output of this identifier model. The construction of the identifier model will be introduced in the next Section.

Our unlearning scheme without annotations is shown in Figure 5.3. We employ the output of the trained identifier model as an encoder. The encoded instance is subsequently provided as input to the original model for fine-tuning. In the encoding process, we first feed one instance $\mathbf{x}_i$ into the trained identifier model and hook the output from a selected layer in *identifier model*. The output of this layer can identify different pattern features, resulting in the output containing diverse feature information. It is worth noting that the identified features only contain different pattern features and no attribute features.

Algorithm 8: Unlearning without annotations

Input: The full training set $\mathscr{D}$, original model M, model with identifying ability $M^{'}_{id}$, feature index id, total iteration number T.
Output: model $M^{'}$ after unlearning process

1 Setting require_grad = *False* of all parameters from $M^{'}_{id}$.
2 Setting require_grad = *True* of all parameters from M.
3 **for** $(\mathbf{x}_i, y_i) \in \mathscr{D}$ **do**
4 Input $\mathbf{x}_i$ to identifier model $M^{'}_{id}$.
5 Hook output O_i in target layer.
6 Resize O_i^{id} to size of $\mathbf{x}_i$.
7 **for** *each pixel* $p \in O_i^{id}$ *and* $q \in \mathbf{x}_i$ **do**
8 **if** p *is not equal to* 0 **then**
9 remove pixel q in $\mathbf{x}_i$
10 $\hat{\mathbf{x}}_i = \mathbf{x}_i$
11 Finetuning M based on $L_M = \sum L(M(\hat{\mathbf{x}}_i), y_i)$.
12 **return** $M^{'} = M$

Therefore, for feature unlearning without annotations, we only consider unlearning the pattern features. Following the above step, we choose a specific output based on the pattern features we intend to unlearn. This selected specific output will be used to encode instance $\mathbf{x}_i$. The algorithm outlining our approach is presented in Algorithm 8.

In Algorithm 8, we first fix the parameters of the identifier model and only update the original model (Lines 1-2). Lines 3-9 involve obtaining the output of the identifier model, and subsequently encoding the original instance $\mathbf{x}_i$ based on these outputs. Specifically, when an output is chosen, it will be expanded to match the original instance's size (line 6). Then, for each pixel point in the expanded output image, if its value is not zero (line 7), the corresponding pixel value in the original instance is removed to eliminate the feature information. This operation indicates that when the pixel value is not equal to one in the expanded output image, it means that this part of the pixel represents the feature that needs to be removed, so the corresponding pixel in the original instance needs to be removed. After that, these modified instances are then used to fine-tune the model further, resulting in feature unlearning without annotations (line 11).

5.3.3 Feature identification without annotations

To tackle the challenges of identifying which features can be unlearned, we empower the output of one model layer with the ability to identify different features. As illustrated in Figure 5.4a, we opt for a specific layer within the model that can be strategically opti-

mized to identify various feature information. Since filters in lower convolutional layers of CNNs typically capture basic texture features, while filters in higher convolutional layers are more inclined to encode object parts or complex pattern features. We focus on training models with automatic detecting features in higher convolutional layers (gray layer in Figure 5.4a), which we call *target layer*. In addition, as shown in Figure 5.4b, since CNNs usually use a set of filters to jointly detect specific image features rather than using a single filter [35], we divide filters $\Omega = 1, 2, \cdots, d$ in the target layer into distinct groups $A_1, A_2, \cdots, A_K$, where $A_1 \cup A_2 \cup \cdots \cup A_K = \Omega$; $A_i \cap A_j = \emptyset$.$\mathbf{A} = \{A_1, A_2, \cdots, A_K\}$ represents the filter partition. Based on this, we need to optimize the selected target layer's parameters to identify various features.

Shen et al. [105] proposed a technique to optimize a convolutional layer's output to identify different feature information with the help of spectral clustering. The group number in the spectral clustering process represents the number of identified features. However, the optimization method in [105] needs to be given the expected number of groups K in advance. It does not consider how to obtain the maximum number of groups and how to ensure that the number of groups is optimal in the clustering process. We should obtain more diverse features in feature unlearning without annotations, enabling us to perform different feature unlearning operations.

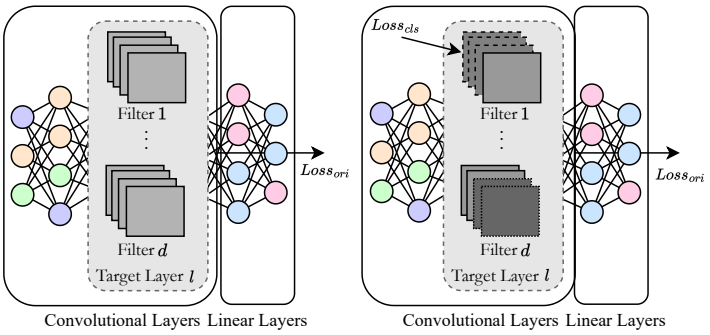
To do that, we introduce the eigengap heuristic, which is based on perturbation theory and spectral graph theory, to determine the number of groups K [117, 134]. The eigengap heuristic indicates that the optimal value for K, representing the number of clusters in spectral clustering, is often identified by maximizing the difference between consecutive eigenvalues. A larger eigengap implies a stronger alignment between the eigenvectors of the desired outcome, resulting in improved performance of spectral clustering algorithms. Therefore, we use the loss in [105] to optimize our model and choose the value of K based on the eigengap heuristic:

$$L(\mathbf{w}, \mathbf{A}) = L_{ori}(\mathscr{D}, \mathbf{w}) + \gamma L_{cls}(\mathscr{D}, \mathbf{w}, \mathbf{A}) \tag{5.6}$$

where $\mathbf{w}$ denotes the parameters of the model. $\mathbf{A}$ represents the parameters of each partition in the target layer. $L_{ori}(\mathscr{D}, \mathbf{w})$ denotes the original model loss on dataset $\mathscr{D}$:

$$L_{ori}(\mathscr{D}, \mathbf{w}) = \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in \mathscr{D}} L(\mathbf{x}_i, y_i, \mathbf{w}) \tag{5.7}$$

γ is a positive weight. $L_{cls}(\mathscr{D}, \mathbf{w}, \mathbf{A})$ is the new loss for identifying the feature information and can be defined as [105]:



(a) Target layer (b) Grouped filters

Figure 5.4: Model structure for feature identification.

$$
\begin{aligned}
L_{cls}(\mathscr{D},\mathbf{w},\mathbf{A}) &= -\sum_{k=1}^{K}\frac{S_k^{\text{within}}}{S_k^{\text{all}}} = -\sum_{k=1}^{K}\frac{\sum_{i,j\in A_k} s_{ij}}{\sum_{i\in A_k,j\in\Omega} s_{ij}} \\
&s.t. \\
S_k^{\text{within}} &= \sum_{i,j\in A_k} s_{ij} = \sum_{i,j\in A_k} \mathscr{K}\left(X_i,X_j\right) \\
S_k^{\text{all}} &= \sum_{i\in A_k,j\in\Omega} s_{ij} = \sum_{i\in A_k,j\in\Omega} \mathscr{K}\left(X_i,X_j\right) \\
s_{ij} &= \mathscr{K}\left(X_i,X_j\right) = \rho_{ij}+1 \geq 0
\end{aligned}
\tag{5.8}
$$

where ρ_{ij} denotes the Pearson's correlation coefficient. X_i denotes the set of output of the i-th filter. For calculating K, we initially calculate the Laplacian matrix using the provided similarity matrixes S_k^{all}. Subsequently, we calculate eigenvectors and eigenvalues from the Laplacian matrix. These eigenvalues are then used to calculate the consecutive difference and arranged in a sorted manner to identify the most optimal value of K. Our algorithm is shown in Algorithm 9. In line 2 in Algorithm 9, we initially compute the eigenvalues and eigenvectors using the similarity values S_k^{all}. Subsequently, we determine the most suitable cluster numbers by identifying the indices associated with larger gaps between the eigenvalues. In this step, each selected cluster number k, satisfies the condition that all eigenvalues $\lambda_1,\ldots,\lambda_k$ are very small, but λ_{k+1} is relatively large. Moving on to lines 4-5, we select the highest value as the number of clusters for filter division. This step ensures that we can achieve the maximum number of clusters while ensuring optimal clustering results. Finally, in lines 6-8, we first train the model using the $L_{ori}(\mathscr{D},\mathbf{w})$ loss function. This initial phase ensures that the model achieves a basic level of classification performance. Subsequently, we proceed to train the identifier

Algorithm 9: Identifying different features

Input: The full training set $\mathscr{D}$, similarity metric $S_k^{\text{all}} = \{s_{i,j} | i, j \in A_k, k \in K\}$, target layer l_{target}, iteration number T_1 and T_2.

Output: model $M_{id}^{'}$ after identifying process

1 Define and initiate one CNN model M_{id}

2 eigenvalues, eigenvectors = $calculateeigen(S_k^{\text{all}})$

3 $Top_5 \leftarrow eigenDecomposition(eigenvalues)$

4 $K \leftarrow max(Top_5)$

5 Divide filters Ω in the target layer l_{target} into distinct groups $A_1, A_2, \cdots, A_K$

6 **for** $t = 0; t < T_1; t++$ **do**

7 $\quad L_{ori}(\mathscr{D}, \mathbf{w}) = \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in \mathscr{D}} \mathrm{L}(\mathbf{x}_i, y_i, \mathbf{w})$

8 $\quad$ Optimizing M_{id} based on $L_{ori}(\mathscr{D}, \mathbf{w})$.

9 **for** $t = 0; t < T_2; t++$ **do**

10 $\quad L_{cls}(\mathscr{D}, \mathbf{w}, \mathbf{A}) = -\sum_{k=1}^{K} \frac{\sum_{i,j \in A_k} s_{ij}}{\sum_{i \in A_k, j \in \Omega} s_{ij}}$.

11 $\quad L_{ori}(\mathscr{D}, \mathbf{w}) = \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in \mathscr{D}} \mathrm{L}(\mathbf{x}_i, y_i, \mathbf{w})$

12 $\quad L(\mathbf{w}, \mathbf{A}) = L_{ori}(\mathscr{D}, \mathbf{w}) + \gamma L_{cls}(\mathscr{D}, \mathbf{w}, \mathbf{A})$

13 $\quad$ Optimizing M_{id} based on $L(\mathbf{w}, \mathbf{A})$.

14 **return** $M_{id}^{'} = M_{id}$

model further using the combined loss function (lines 9-13). This step endows the model with the capacity to identify feature information effectively.

5.4 Experiment

In this section, we first evaluate our scheme from both qualitative and quantitative perspectives (Section 5.4.1 and Section 5.4.2). The feasibility of the fine-tuning-based scheme and validity of introducing the eigengap heuristic technique is evaluated in Section 5.4.3 and Section 5.4.4, respectively. We also test the effect of hyperparameters in Section 5.4.5. Finally, we consider one of the important applications of feature unlearning: as an optional alleviation scheme for model debiasing (Section 5.4.6).

5.4.0.1 Evaluation Metrics

Training deep models with large datasets involves various randomness, which presents a challenge in evaluating the effectiveness of machine unlearning schemes. Existing instance-level evaluation methods primarily focus on model accuracy or attack-based techniques based on model performance [44, 48, 120, 129]. For attack-based methods, Graves et al. [44] and Yu et al. [48] based on the backdoor and membership inference

attacks (MIAs), while Heng et al. [129] based on trigger samples and reaction samples. Users send unlearning requests regarding trigger samples and use reaction samples to verify if the unlearning operation has been successfully carried out. However, when it comes to unlearning specific features, the model itself cannot provide any performance information about those features, which results in the invalid of the above instance-level evaluation schemes. For example, precision information regarding male and female features in scene recognition classification models cannot be obtained. Therefore, these approaches are inadequate for evaluating whether feature information has been effectively unlearned from the model. To evaluate the effectiveness of feature unlearning, we consider three evaluation methods from different dimensions:

- Accuracy from the adversary model: In our unlearning scheme with known annotations, the adversary has the ability to quantify the amount of information about target features. Therefore, we indirectly determine if there is information about the features within the model based on the accuracy of the adversary.

- Variation in accuracy: For feature unlearning without annotations, it is difficult to quantitatively evaluate whether the pattern features are actually unlearned. We adopt a combination approach to determine whether features have been successfully unlearned, relying on the correlation between the target feature pattern and the task pattern, along with the accuracy of the model's task. Suppose we consider unlearning features f_i from any trained original model. The accuracy of the original model was acc_{before} before the feature unlearning operation and became acc_{after} after the unlearning operation. When the correlation between unlearning features and the original model task is low, the value of $acc_{after} - acc_{before}$ should be small. This indicates that when a feature that needs to be unlearned is unrelated to the model's task, unlearning the feature information does not significantly affect model accuracy. Conversely, when this feature is relevant to the model's classification task, unlearning that feature information will significantly decrease the model's accuracy. That is, the value of $acc_{after} - acc_{before}$ should be big.

- Gradient visualization: We introduce guided backpropagation [111] as qualitative evaluation method. This approach primarily focuses on comprehending the features that impact the model's decisions or predictions. Employing this technique enables us to determine the presence or absence of the target feature visually.

5.4.0.2 Baseline methods

As we mentioned before, there is no relevant research on feature unlearning for image classification models. Feature unlearning for tabular data has been proposed in [123], however, this scheme cannot be extended to the image level and cannot be verified under a non-convex model that specific features in an image are really unlearned. Therefore, we consider the following methods as our baselines:

- Instance-level fine-tuning: We remove all instances that contain target feature from the dataset (f_i = True) and use the remaining dataset to fine-tune the model for unlearning (f_i = False). This scheme is used to evaluate whether a model is able to unlearn feature information when it is fine-tuned with instances that do not contain the target features. If our adversarial learning scheme proves effective while the instance-level fine-tuning scheme does not, this further illustrates the validity of our approach.

- Instance-level retraining: The traditional method for machine unlearning typically requires retraining the model from scratch after removing the instances that need to be unlearned. There are also numerous approaches proposed to enhance the retraining process, such as [11, 15, 24]. To illustrate that traditional instance-level methods cannot achieve feature-level unlearning, we choose this as a baseline.

In the scenario of unlearning without annotations, we identify which pattern features within instances can be unlearned by introducing the model interpretability technique. To identify more feature information and achieve optimal training results, we introduce the eigengap heuristic to optimize the identification process. As a comparison, we evaluate our scheme with icCNN proposed in [105].

5.4.1 Performance analysis: visualization results

To evaluate the effectiveness of our scheme, we conduct experiments based on the metric outlined in Section 5.4.0.1. We consider three different settings:

1. Single feature unlearning with annotations: This scenario refers to the fundamental process of feature unlearning, wherein we aim to unlearn information associated with a particular feature from the model. During this unlearning request, we can access corresponding annotation information about the unlearning feature from the dataset.

(a) Results for single feature (b) Results for multi feature (c) Results without annotations

Figure 5.5: Qualitative results based on the guided backpropagation [111].

2. Multi feature unlearning with annotations: As mentioned in Section 5.3.1, our scheme can unlearn multiple feature information simultaneously. This scalability allows us to unlearn multiple features at once efficiently. To validate this capability, we conducted experiments involving the unlearning of multiple features concurrently.

3. Feature unlearning without annotations: This aspect presents one of the most challenging requirements in feature unlearning since we lack knowledge about the annotations of the features that need to be unlearned. Without explicit information regarding the annotations, it becomes significantly more complex to identify and unlearn the features' information effectively.

Setup. For setting 1), we choose ResNet for both the original and adversary models and select U-net for the remover model. In the first step, we respectively train the original model and the adversary model to identify *Bald* and *Mouth Slightly Open* tasks in CelebA. We aim to unlearn the information related to whether the mouth is open, specifically the *Mouth Slightly Open* feature from the original model. For the training process, we set epoch = 10, batch size = 50 and learning rate = 0.000005. For executing the adversarial unlearning process, we set the epoch = 50, batch size = 36, learning rate = 0.000005, $\beta = 5.0$ and $\lambda = 5.0$.

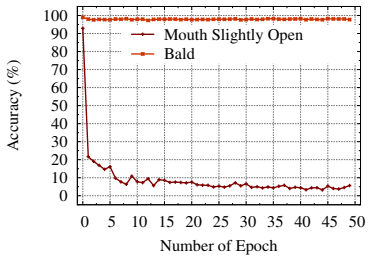
For setting 2), we choose the model structure from setting 1) for this setting. In order to simulate unlearning multiple features, we group the features *Mouth Slightly Open* and *Pointy Nose* together and aim to unlearn them from the original model. Concurrently, we train the original model to identify *Bald* tasks in CelebA. The adversary model is trained to possess the capability of multi-task classification, allowing it to recognize both *Mouth Slightly Open* and *Pointy Nose* in CelebA. For training adversary model, we set epoch=20, batch size =128 and the learning rate=0.000005. For the original model, we set epoch = 10, batch size = 50 and learning rate = 0.000005. For executing the adversarial

unlearning process, we set the epoch=50, batch size=36, learning rate=0.0001, $\beta = 1.0$ and $\lambda = 10.0$.

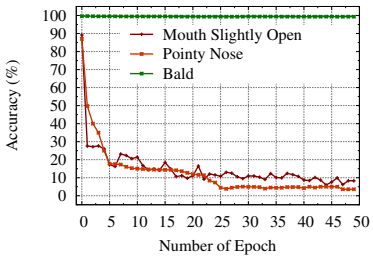
For setting 3), we also use the ResNet model as the architecture of our identifier model and select the CelebA dataset to train this model with the ability to recognize different features. Afterward, we fine-tune the original model using the new encoded images to achieve the purpose of unlearning features without annotations. Specifically, we select the filter that recognizes the mouth in the identifier model and implement the unlearning *Mouth* pattern feature in the *Bald* and *Smiling* classification models based on this filter. For training the identifier model, we first set epoch = 200, batch size = 256 and learning rate = 0.000001 to train it based on the loss $L_{ori}(\mathscr{D}, \mathbf{w})$. By following this, the model will attain a fundamental level of classification performance. After that, we continue to train the identifier model based on $L_{cls}(\mathscr{D}, \mathbf{w}, \mathbf{A}) + L_{ori}(\mathscr{D}, \mathbf{w})$ and set the batch epoch = 2500, size = 128 and learning rate = 0.00001. This step endows the model with the capacity to identify feature information effectively. Other parameters in the training process are the same in [105]. For the fine-tuning unlearning process, we set the learning rate as 0.001, batch size = 128.

During the unlearning process of all the above experimental settings, we show the gradient based on the gradient visualization technique within the original model. The results are shown in Figure 5.5.

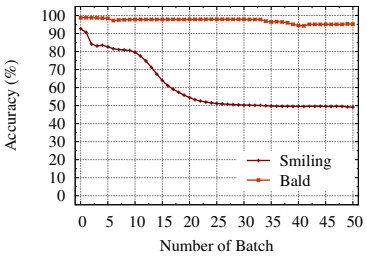
Results. In Figure 5.5, Figure 5.5a shows the results of unlearning a single feature with known feature annotations, while Figure 5.5b illustrates the result of unlearning multiple features. Figure 5.5c shows the result of unlearning feature without annotations. As can be seen from all Figures, the gradient map produced by guided backpropagation does not have information about features that need to be unlearned, which suggests that our scheme does not use feature-related information to fine-tune the model. Examples include the mouth information in Figures 5.5a and 5.5c, and the mouth and nose information in Figure 5.5b. Simultaneously, images unrelated to the unlearning features remain unaffected, indicating that adversarial training or encoding processes can preserve information unrelated to those unlearning features. Considering the impact of catastrophic forgetting, the model will autonomously unlearn information about specific features. In addition to this, it should be emphasized that the results in Figure 5.5c demonstrate the results of feature unlearning without annotations. It can be seen that even without annotations, our unlearning scheme based on interpretability techniques can achieve almost the same results as the known annotations scheme (Figure 5.5a). Both unlearning schemes can remove the gradient information about the mouth feature.



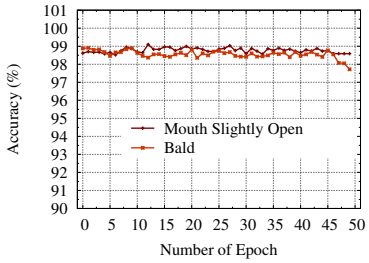
(a) Single Feature Unlearning



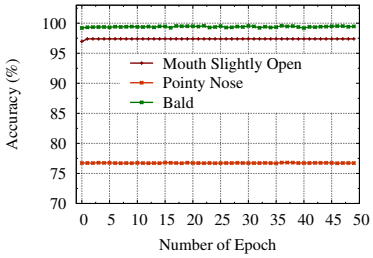
(b) Multi Feature Unlearning



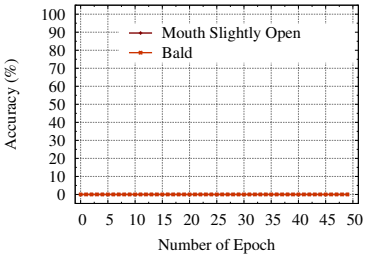
(c) Feature Unlearning without Annotations



(d) Instance-level finetuning (Single)



(e) Instance-level finetuning (Multi)



(f) Instance-level retraining

Figure 5.6: Quantitative results with different configurations.

5.4.2 Performance analysis: quantitative results

In this section, we carry out the experiments utilizing the experimental setup detailed in Section 5.4.1. In addition, as a comparison, we also try to remove feature information from the model based on *instance-level fine-tuning* and *instance-level retraining* schemes mentioned in Section 5.4.0.2:

- For instance-level fine-tuning, we choose the ResNet architecture for the original model and train the original model to identify *Bald* tasks in CelebA. Then, we aim to unlearn the information related to whether the mouth is open, specifically the *Mouth Slightly Open* feature from the original model. We delete all instances that *Mouth Slightly Open* = True from the dataset and use the remaining dataset (*Mouth Slightly Open* = False) to fine-tune the original model for unlearning. In addition, we also consider unlearning *Mouth Slightly Open* and *Pointy Nose* features from the original model. For the training process of the original model, we set epoch = 10, batch size = 50 and lr = 5e-06. For the fine-tuning process, we set epoch = 50, batch size = 50 and lr = 1e-04.

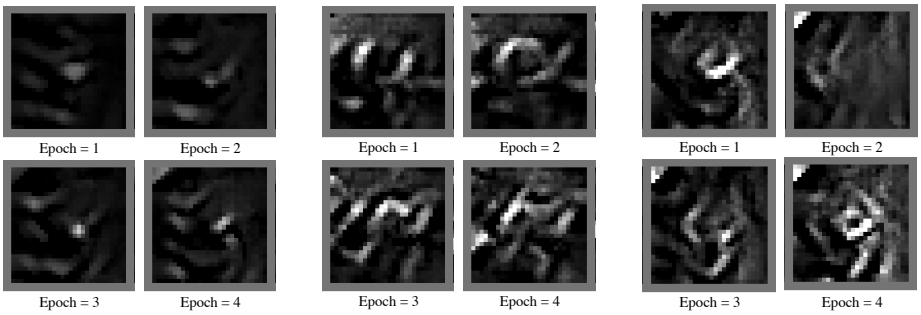
- For instance-level retraining, we also select the ResNet architecture as the original

model and retrain it to recognize *Bald* tasks within CelebA. Our goal is to remove the information associated with the state of the mouth, particularly the "Mouth Slightly Open" feature, from the original model. Therefore, we first remove all instances from the model that illustrate the state of whether the model is opening (Mouth Slightly Open = True and false), and then retrain the model to recognize *Bald* tasks. For the retraining process, we set epoch=50, batch size = 50 and learning rate = 1e-04.

The results are then evaluated using the metrics mentioned in 5.4.0.1 and shown in Figure 5.6.

Results. As illustrated in Figure 5.6, Figure 5.6a and Figure 5.6b show the results of feature unlearning with known annotations, while Figure 5.6c illustrates the result of feature unlearning without annotations. The precision associated with the features that need to be unlearned in Figure 5.6a and Figure 5.6b exhibits a progressive decrease as the unlearning process goes on. Furthermore, the model's original task performance demonstrates marginal alteration, emphasizing the efficacy of feature unlearning in retaining task-related information while removing target feature information from the model. In Figure 5.6c, the accuracy concerning the feature bald remains relatively stable, whereas the accuracy about the feature smiling experiences a substantial reduction. This result is consistent with our expected findings, as the feature smiling exhibits substantial relevance to the features that need to be unlearned, whereas its correlation with the feature bald is less. As a result, the effect on the performance of smiling is more significant. The results demonstrate that our interpretability-driven unlearning approach can achieve nearly identical outcomes as the established annotation-based scheme, even in the absence of annotations.

Figure 5.6d and Figure 5.6e show the results of instance-level fine-tuning with known annotations, while Figure 5.6f illustrates the result of instance-level retraining with known annotations. Contrary to our results in Figure 5.6a and Figure 5.6b, in Figure 5.6d and Figure 5.6e, the accuracy of the features to be unlearned does not decrease. This suggests that even when all instances containing *Mouth Slightly Open* feature are removed (*Mouth Slightly Open* = True), the model can still obtain information from other mouth-related instances (There's a mouth, but it's not opening.). Certainly, it's technically possible to remove all mouth-related instances from the dataset, but this would lead the model's original accuracy to zero, as illustrated in Figure 5.6f. For the experimental result in Figure 5.6f, since it removes all instances where the mouth is opening or closing (both *Mouth Slightly Open* = True and *Mouth Slightly Open* = False),



(a) Results of Original Training Process (b) Results of Retraining Process (c) Results of Finetuning Process

Figure 5.7: The results of model inversion attack.

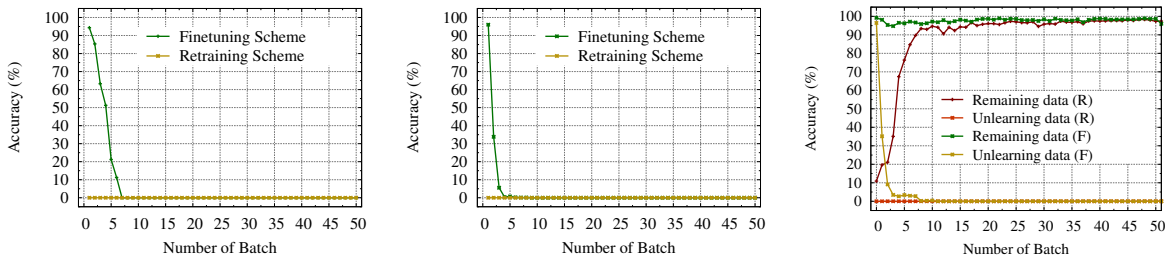
it results in no instances being available for model training during the retraining process, resulting in the accuracy of the model remaining at 0.

Summary: From all the above results, our unlearning schemes can effectively eliminate the information associated with the target features from the model, while all instance-level either fail to remove completely or remove too much. More importantly, our proposed feature unlearning without annotation scheme also effectively removes feature information from the models, which can address fairness issues without annotation, while most existing adversarial fine-tuning-based schemes are impractical [122]. We will discuss this more detail in Section 5.4.6.

5.4.3 Comparing with learning from scratch

In this section, we aim to verify whether the fine-tuning unlearning scheme yields a nearly equivalent outcome while exhibiting more efficiency in comparison to the completely retraining scheme [44]. To accomplish this, we conduct separate evaluations to assess the effectiveness of the fine-tuning and retraining schemes. We employ the currently dominant validation schemes, model inversion attacks and membership inference attacks (MIAs), to assess the results.

Setup. We implement the model inversion attacks as described in [38]. We first train the ResNet model based on the MNIST dataset with epoch = 10, batch size = 128 and learning rate = 0.1. Since these two evaluation schemes only support instance-level unlearning, we select the unlearning data as all instances in class three and consider other class data as the remaining data. To evaluate the performance of the retraining and fine-tuning approaches, we perform unlearning operations on the already trained model using both methods separately. For the retraining and fine-tuning unlearning process, we



(a) MIAs in MNIST Dataset (b) MIAs in CIFAR-10 Dataset (c) Time Consume Evaluation

Figure 5.8: The results of membership inference attacks.

use the same training setting as the original training process. Subsequently, We attack the above three models (original, retrained, and fine-tuned) once per epoch based on model inversion attacks. Figure 5.7 illustrates partial results of model inversion attacks against the original model and the other two models affected by different unlearning methods. Additionally, we use MIAs in paper [133] to evaluate whether the unlearning data are still identifiable in the training dataset. We set the number of shadow models as 10 and the training epoch of the shadow model as 10, batch size = 64. The attack model is a fully connected network with two hidden linear layers of width 256 and 128, respectively, with ReLU activation functions and a sigmoid output layer. We evaluate our unlearning scheme with two settings, MNIST + ResNet and CIFAR-10 + ResNet, and set unlearning class = 3. We equally divide the training dataset into two subsets to generate the training dataset for shadow models and then train the attack model based on the output of those shadow models. Figure 5.8 shows the results of our evaluation.

Results. As shown in Figure 5.7, Figure 5.7a illustrates the results of the original training process. It shows a notable trend: as the training progresses, the model becomes increasingly knowledgeable about the unlearning class 3, leading to a more pronounced exposure of information through model inversion attacks. For the retraining scheme in Figure 5.7b, the attack's effectiveness is almost ineffective since the model lacks knowledge about the unlearning class. For the fine-tuning scheme in Figure 5.7c, the results reveal that after the first round of fine-tuning, the information about the unlearning class becomes considerably challenging to recover. This is because the model inversion attack has relatively little information about the unlearning data to rely on after only one epoch fine-tuning process. As the fine-tuning process progresses, the model retains less and less information about the unlearning class. Consequently, attempts to recover those data through model inversion attacks become increasingly challenging.

For the results of MIAs in Figure 5.8, before the fine-tuning process (points with

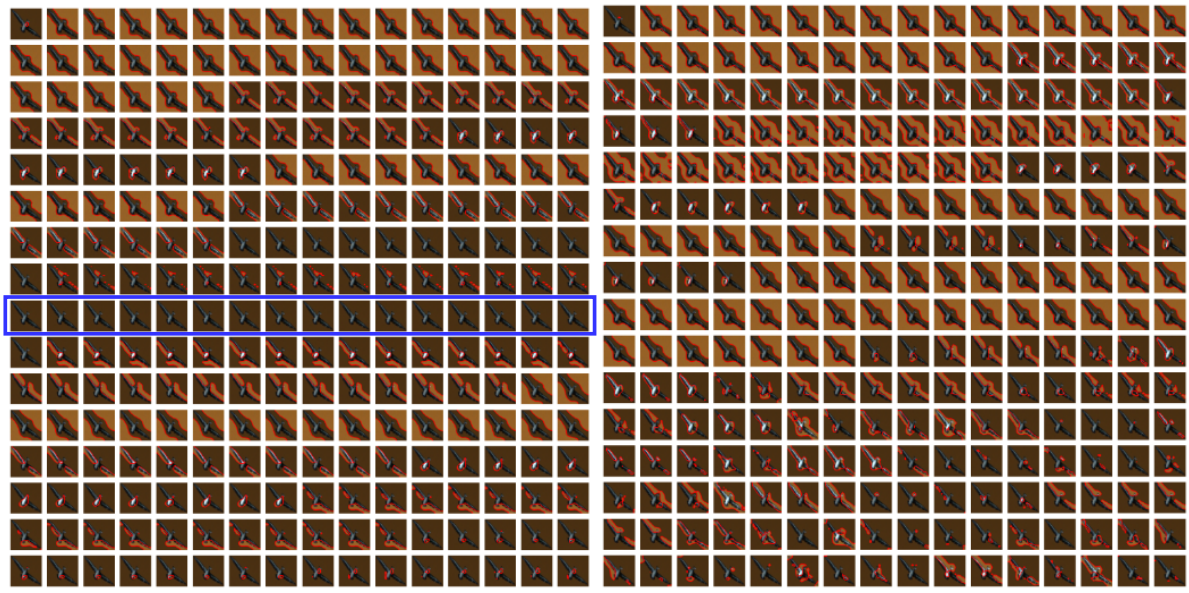
x=0), MIAs have a high success rate for all original models; i.e., it successfully derived the training dataset containing unlearning class. For the retraining process, the success rate of MIAs is zero, indicating that MIAs cannot determine the existence of unlearning class in the training dataset since the unlearning class actually isn't in the training set. For the fine-tuning scheme, as the information contained in the model decreases, the effectiveness of the attack also decreases. This suggests that the fine-tuning scheme can also remove information about unlearning data from the model. In addition, we also conduct measurements on model accuracy for both the unlearning data and the remaining data during both unlearning processes with the MNIST dataset. The results are shown in Figure 5.8c. During the retraining process, the model's accuracy on the unlearning data is 0 because it didn't learn anything from this specific data. On the other hand, for the fine-tuned model, the accuracy of the unlearning data gradually decreases due to the effect of catastrophic forgetting. As for the remaining data, the fine-tuning process shows minimal changes, whereas the retraining process continually learns and improves to eventually achieve a usable state. It is important to highlight that the fine-tuning model achieves much faster unlearning results compared to retraining the model to a usable state. This suggests that the fine-tuning scheme has a better unlearning effect compared to the retraining scheme.

Summary. As expected, similar to the retraining scheme, the fine-tuning scheme hinders the inference of any meaningful information concerning the unlearning data. Moreover, the fine-tuning scheme proves to be more efficient compared to retraining. Therefore, in this chapter, we adopt the fine-tuning scheme for the unlearning of features.

5.4.4 Identifying results

As discussed in Section 5.3.3, when feature annotations are unavailable, we focus on training the model to identify feature information from the image automatically. To accomplish this, we propose applying the eigengap heuristic technique, which optimizes the clustering process and enhances the extraction of additional feature information. In this Section, we provide a comparative analysis between our enhanced scheme and the original scheme, icCNN, in [105].

Setup. We followed experimental settings in Shen et al. [105] to construct our experiment. Specifically, we train the identifier model based on ResNet architecture [52] and choose the bird category in the PASCAL-Part dataset [25] to optimize our model. Just like [105, 139], we add the loss $L_{cls}(\mathscr{D}, \mathbf{w}, \mathbf{A})$ to the first convolutional layer after $layer3$ as our target layer. This is because filters in high convolutional layers tend to



(a) Results from icCNN (b) Resuls from our scheme

Figure 5.9: Visualization of feature maps of icCNN [105] and our scheme.

capture object parts or pattern features rather than fine-grained textures [8]. We first train the original ResNet model based on the classification loss $L_{ori}(\mathscr{D}, \mathbf{w})$. Then, we simultaneously optimize loss $L_{cls}(\mathscr{D}, \mathbf{w}, \mathbf{A})$ and $L_{ori}(\mathscr{D}, \mathbf{w})$ to fine-tune the trained model so that its parameters can recognize different pattern features. In the first step, we set batch size = 256, learning rate = 0.000001 and epoch = 200. In the second step, we set batch size = 16, learning rate = 0.00001 and epoch = 2500. For icCNN scheme in [105], we set the cluster number as 16. For λ in two schemes, we set 1. We followed Zhang et al. [139] to visualize each filter‚Äôs feature map in target layer. The results are shown in Figure 5.9.

Results. In Figure 5.9, Figure 5.9a shows the identification results of the target layer from the icCNN model [105], while Figure 5.9b shows the results from our identifier model. Both visualization results indicate that each filter in different groups can identify various feature information, such as the wings and stomach of the bird, while in the same groups, each filter identifies almost the same feature. This result is the same as that in [105]. In addition to this, compared to Figure 5.9a from [105], our method recognizes more types of feature information and all the filters are involved in the identification operation. For [105], on the other hand, since the number of clusters is randomly determined and does not consider whether the resulting clustering is an optimal solution, it leads to some filters not participating in the identification process (Filters as marked in

the blue box in Figure 5.9a). This comparison illustrates that our scheme can achieve better optimization results and identify more pattern features, which can be used to unlearn various feature information in the unlearning step.

5.4.5 The effect of hyper-parameters

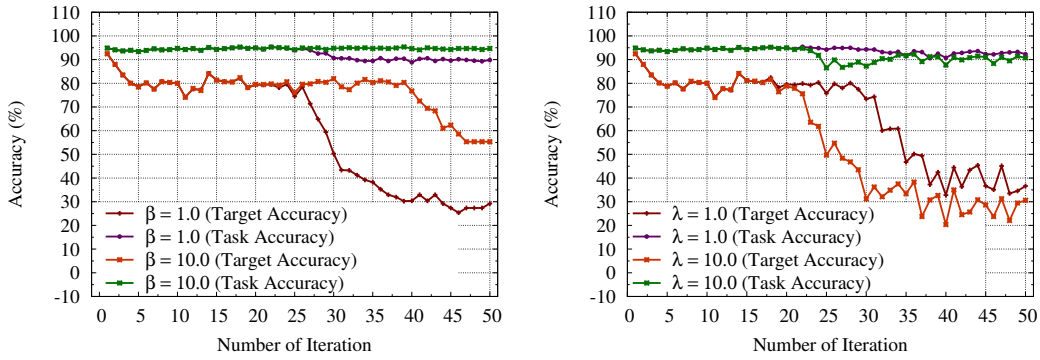
Setup. In our feature unlearning with known annotations, there are two hyperparameters: λ and β. To evaluate the impact of these hyperparameters, we set the following experiments: We opt for the ResNet architecture for both the original and adversary models, while we select the U-Net architecture for the remover model. In the initial step, we separately train the original model and the adversary model to identify the tasks of 'Smiling' and 'Mouth Slightly Open' using the CelebA dataset. Our goal is to unlearn information related to whether the mouth is open, specifically the 'Mouth Slightly Open' feature, from the original model. During the training process, we set the following parameters: epoch = 10, batch size = 50, and learning rate = 0.000005. For the adversarial unlearning process, we set the batch size to 36 and the learning rate = 0.000005. We hold one hyperparameter constant while allowing the other hyperparameter equal to 1.0 and 10.0, respectively. We then record the model accuracy after a single iteration of adversarial unlearning. Figure 5.10 shows the results.

Results. It can be seen from Figure 5.10a, when fixing λ, a smaller β unlearns the target feature quickly but affects the accuracy of the original model. In Figure 5.10b, a larger λ speeds up the feature unlearning of the model, but again reduces the performance of the original model. In summary, larger λ and smaller β will affect the model performance for the original model. In practice, it should choose appropriate values to achieve unlearning while minimizing the impact on the performance of the original model.

5.4.6 Feature unlearning application

As detailed above, the concept of feature unlearning can be regarded as an alternative approach to address fairness issues within models. In this Section, we proceed to evaluate the efficacy of implementing feature unlearning as a strategy to alleviate biases inherent in models.

Setup. We consider three distinct debiasing scenarios that cover different combinations of pattern features and attribute features: the removal of gender bias (specifically, male bias) from a model classifying whether a mouth is slightly open; secondly, the



(a) Evaluation the effects of λ (b) Evaluation the effects of β

Figure 5.10: The Effect of Hyper-Parameters.

Table 5.1: The configuration of training biased model.

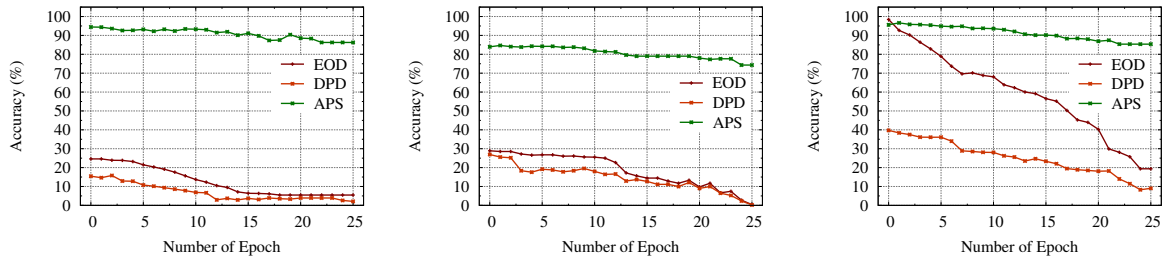
	Mouth (Male)	Young (Smiling)	Bald (Mouth)
Both False	2000	2000	2000
False and True	16000	10000	10000
True and False	16000	10000	10000
Both True	2000	2000	2000

alleviating of smiling bias from a model categorizing whether an individual is young and lastly, the mitigation of bias associated with mouth slightly open in a bald classification model. We reconstruct the dataset to simulate the bias present in the dataset. The configuration of the reconstructed dataset is shown in Table 5.1. We set epoch=10, batch size =50, and the learning rate=0.000005 for training the adversary model. For the original model, we set epoch = 50, batch size = 128 and learning rate = 0.001. The setting for each feature unlearning process is illustrated in Table 5.2. To better show that the bias in the model is indeed alleviated by the unlearning process, we set up a comparison experiment, i.e., we only use the same data to fine-tune the model without the unlearning process (denoted as naive method). If the unlearning scheme succeeds in alleviating bias and the naive method does not alleviate bias, this suggests that feature unlearning can be used as an optional program for removing bias from the model. We use the *equalized odds difference* (EOD) and *demographic parity difference* (DPD) in the *fairlearn.metrics* package to evaluate the model bias and use the *average precision score* (APS) to test the model performance. The results are shown in Figure 5.11.

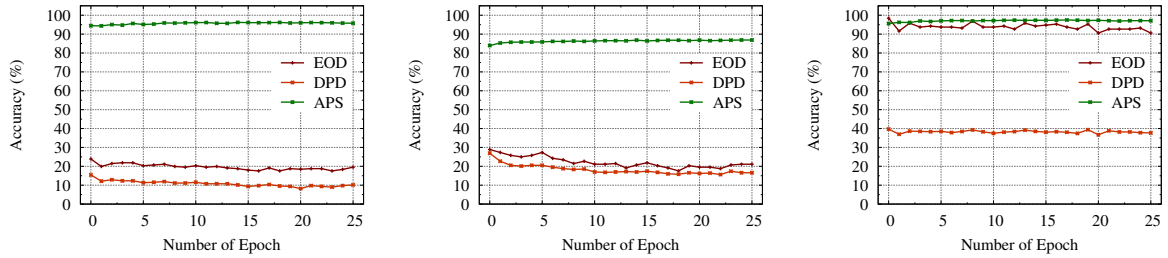
Results. As shown in Figure 5.11, Figure 5.11a to Figure 5.11c illustrate the results of alleviating bias based on feature unlearning, while Figure 5.11d to Figure 5.11f

Table 5.2: The experimental setting for debiasing.

	Mouth (Male)	Young (Smiling)	Bald (Mouth)
Epoch	25	25	25
Batch Size	64	64	64
LR (remover)	0.00001	0.00001	0.000025
LR (fine-tuning)	0.00001	0.000005	0.00001
λ	1.6	1.3	1.6
β	1.0	1.0	1.0



(a) Unlearning result for Mouth (Male) (b) Unlearning result for Young (Smiling) (c) Unlearning result for Bald (Mouth)



(d) Naive result for Mouth (Male) (e) Naive result for Young (Smiling) (f) Naive result for Bald (Mouth)

Figure 5.11: The results of debiasing based on feature unlearning.

show the results from the naive scheme. The values corresponding to *equalized odds difference* (EOD) and *demographic parity difference* (DPD) exhibit a gradual diminution during the unlearning process, signifying progressive alleviation of bias within the model. Conversely, the naive scheme did not reduce bias within the model. Furthermore, the strategy of feature unlearning demonstrates a minimal impact on the model's accuracy, as evidenced by the marginal alteration in the *average precision score* (APS) value. This observation implies that the process of feature unlearning effectively preserves feature information associated with the model task and removes all information about the feature that creates bias. This observation aligns seamlessly with our initial hypothesis that feature unlearning is an elective approach for alleviating model bias.

5.5 Summary

This chapter proposed an innovative machine unlearning approach that enables the selective removal of feature information from a trained model. We consider two types of unlearning requests: feature unlearning with known annotations and feature unlearning without annotations. In the case of unlearning with known annotations, we utilize adversarial learning to eliminate feature-related information from the model. For unlearning without annotations, we design a re-encoded and fine-tuning technique. The experimental results provide evidence that our approach effectively enables the model to eliminate the impact of features while maintaining accuracy in a quick and efficient manner.

This chapter inspires us that we can extend or modify current methods to make it applicable to other complex types of features, such as the texture of the object or the sentiment of the language. In addition, evaluation schemes for feature unlearning also need further research. Furthermore, we have future plans to develop a more comprehensive scheme capable of effectively addressing unlearning requests from Natural Language Processing (NLP) or Generative Adversarial Networks (GANs).

CHAPTER 6

EVALUATING OF MACHINE UNLEARNING: ROBUSTNESS VERIFICATION WITHOUT PRIOR MODIFICATIONS

Machine unlearning, a process enabling pre-trained models to remove the influence of specific training samples, has attracted significant attention in recent years. While extensive research has focused on developing efficient unlearning strategies, the critical aspect of unlearning verification has been largely overlooked. Existing verification methods mainly rely on machine learning attack techniques, such as membership inference attacks (MIAs) or backdoor attacks. However, these methods, not being formally designed for verification purposes, exhibit limitations in robustness and only support a small, predefined subset of samples. Moreover, dependence on prepared sample-level modifications of MIAs or backdoor attacks restricts their applicability in Machine Learning as a Service (MLaaS) environments. To address these limitations, we propose a novel robustness verification scheme without any prior modifications, and can support verification on a much larger set [1]. Our scheme employs an optimization-based method to recover the actual training samples from the model. By comparative analysis of recovered samples extracted pre- and post-unlearning, MLaaS users can verify the unlearning process. This verification scheme, operating exclusively through model parameters, avoids the need for any sample-level modifications prior to model training while supporting verification on a much larger set and maintaining robustness. The effectiveness of our proposed approach is demonstrated through theoretical analysis and experiments involving diverse models

[1]The main content of this chapter has been submitted to IEEE Transactions on Dependable and Secure Computing.

on various datasets in different scenarios.

6.1 Introduction

Despite the success achieved in machine unlearning strategies, The verification of unlearning emerged as a new requirement. Verification means a model provider can prove that the requested sample has been removed from the model, or user can test that his/her samples have been unlearned from the model successfully. However, currently, users have limited ability to monitor the unlearning process and confirm if their data has been truly unlearned from the model [130].

Here are some prior research results on the verification of machine unlearning. Weng et al. [124] proposed using trusted hardware to enforce proof of unlearning, though their method relies on trusted execution environments within Machine Learning as a Service (MLaaS). Cao et al. [15] introduced data pollution attacks to assess if the model's performance was recovered to its original state after the unlearning process. Similarly, Guo et al. [48] evaluated whether the model provider truly performs machine unlearning by analyzing the performance of backdoor attacks. Other verification schemes are based on the distribution of model parameters [84], membership inference attacks (MIAs) [37], model inversion attacks [44], theoretical analysis [46], or model accuracy [12, 40, 42, 120].

However, those unlearning verification schemes have several limitations. Schemes based on model accuracy cannot reliably determine if targeted samples have been truly unlearned, since unlearning partial samples may not significantly affect model performance for those targeted samples [12, 40, 42, 120]. Schemes based on model inversion attacks do not support sample-level verification but rather only enable verification at the class level, as they can only recover class-level representation [44]. Verification schemes, such as those based on MIAs, distribution of model parameters or theoretical analysis, are often ineffective due to unstable performance [63]. Beyond these drawbacks, there are several common limitations:

- Existing unlearning verification schemes always involve modifying samples prior to model training. For example, backdoor-based schemes require altering training samples by adding backdoor triggers [48]. These triggers are then used to verify the unlearning process. Similarly, schemes relying on data pollution use modified, poisoned samples to confirm unlearning. However, if these modifications lose their effect during the learning or unlearning process, those verification schemes become ineffective.

- Most existing unlearning verification methods only support a small, predefined subset of samples. For example, schemes relying on backdoor or data poisoning techniques necessitate the preparation of samples used for verification before model training process. Consequently, those approaches are restricted to verifying only those pre-prepared samples, failing to consider the verification of unlearning for samples that were not modified prior to the training process [48]. This constraint significantly narrows the scope of unlearning verification, potentially leaving critical gaps in the verification process.

- Nearly all existing verification schemes lack *robustness*, which includes both long-term effectiveness and adaptability to varied-term unlearning conditions. To illustrate the concept of long-term effects, consider verification methods such as those based on parameter distribution, MIAs, backdoor, or data poisoning. These methods may initially demonstrate effectiveness in verifying unlearning immediately after the process. However, their reliability can diminish after subsequent model modifications like fine-tuning or pruning [48, 77, 130]. Moreover, in varied-term scenarios, many verification methods are context-dependent, only confirming successful unlearning under specific, predefined conditions. This limited scope increases the risk of false positives - erroneously confirming successful unlearning when the influence of targeted samples may actually persist or even amplify post-unlearning [20, 22, 112, 131]. (see Section 6.4.5).

In this chapter, we propose UnlearnGuard, a novel robustness verification scheme for machine unlearning that operates without requiring any prior modifications to training samples. Our approach is based on the observation that neural networks tend to memorize actual training samples. Therefore, we aim to directly extract these training samples from the model for verification purposes.

1. To address the first limitation, we redefine model training as a maximum margin problem, leveraging insights from implicit bias [62, 87] and data reconstruction [13, 50], we introduce our primary recovery loss component derived from Karush-Kuhn-Tucker (KKT) point conditions. This loss allows us to recover actual training samples from model used for verification without modifying the training samples.

2. To cover as many samples as possible that can be verified for unlearning, we propose an additional loss that allows further recovered samples to exhibit greater similarity to their original counterparts in training data space. This is achieved by

minimizing the negative absolute outputs and applying a projection that constrains those newly recovered samples to remain within a specified range of pre-recovered samples. By enhancing this similarity, more recovered samples become suitable for verification.

3. To tackle the limitation of lacking robustness, all proposed loss components only use model parameters for recovery, without incorporating any other information and condition for verification. We also provide theoretical proof based on implicit bias [62, 87], demonstrating how the differences between samples recovered after successful unlearning and those before unlearning, providing a theoretical foundation for robustness property.

It's worth noting that, in our experiments, we find that some fine-tuning-based machine unlearning schemes does not completely remove the influence of targeted samples. In fact, those schemes appear to deepen the impacts of targeted samples in the model. This result contrasts with previous findings [20, 22, 112, 131]. Comparing with previous verification methods, our proposed verification scheme can further enhance the reliability of machine unlearning.

In summary, we make the following contributions.

- We take the first step in addressing machine unlearning verification problem without prior sample-level modifications, considering both the robustness of the verification scheme and its capacity for more sample verifications.
- We propose an optimization-based method for recovering actual training samples from models, which enables users to verify machine unlearning by comparing samples recovered before and after the unlearning process.
- We provide theoretical proofs and analyses of our scheme based on implicit bias to demonstrate its effectiveness.

6.2 Problem Definition

6.2.1 Existing Verification Schemes

Designing an effective and efficient verification scheme for machine unlearning is difficult. Verification schemes based on accuracy, theoretical analysis and distribution similarity have consistently proven ineffective in [115, 130]. In this Section, we further highlight

the critical limitations of attack-based verification schemes to illustrate the novelty of our scheme. Figure 6.1 illustrates the main ideas of commonly used attack-based verification schemes.

As shown in Figure 6.1, attack-based unlearning verification schemes typically involve three roles, including data provider, model provider and one trusted third party. Data provider first pre-selects a subset of triggers $\mathscr{D}_u$ before model training. For example, in schemes based on data pollution and backdoor attacks, triggers are often generated by a trusted third party [15, 48, 110]. In MIAs-based verification schemes, data providers directly select partial training samples as those triggers [22, 37, 63, 72]. Based on those triggers, the verification process can be summary as the following steps:

- **Trigger Integration**: Triggers $\mathscr{D}_u$, either generated from a trusted third party or selected directly from the data provider,Äôs own dataset, are added to the training dataset. The model is then trained on this combined dataset.

- **Initial Prediction**: The data provider sends the verification request to the trusted third party, and the trusted third party queries the model's prediction for these triggers. The model provider returns the predictions $O(\mathscr{D}_u)$. Existing attack-based verification scheme outputs the verification result $\mathscr{V}(O(\mathscr{D}_u))$ for $\mathscr{D}_u$ based on $O(\mathscr{D}_u)$.

- **Unlearning Request**: The data provider submits an unlearning request for those triggers $\mathscr{D}_u$.

- **Post-Unlearning Prediction**: The data provider and the trusted third party repeat the steps in the initial prediction phase and output the verification result $\mathscr{V}(O(\mathscr{D}_u)')$.

- **Verification**: By comparing the returned predictions before and after unlearning, $\mathscr{V}(O(\mathscr{D}_u))$ and $\mathscr{V}(O(\mathscr{D}_u)')$ respectively, the data provider determines if the model has undergone the unlearning process.

However, those attack-based machine unlearning verification schemes mainly have the following limitations:

- **Sample-Level Modification before Model Training**: Ensuring effective verification often requires incorporating a large number of modified samples, such as backdoored samples, which increases computational costs.

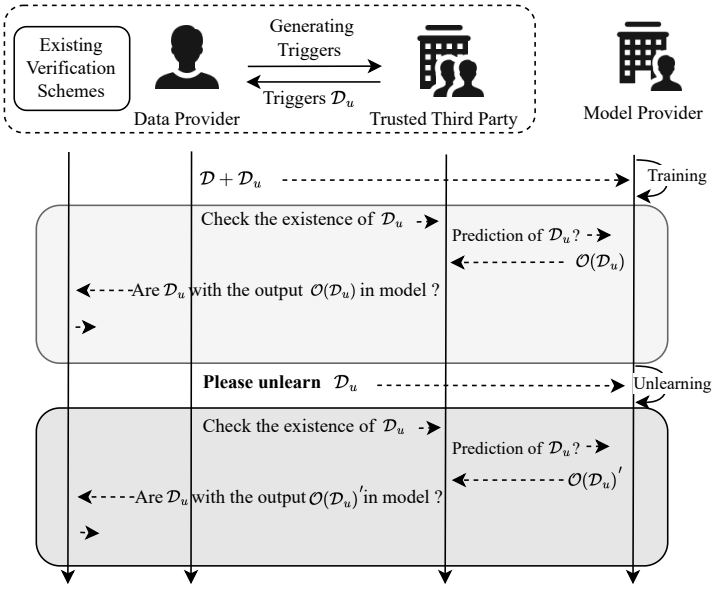


Figure 6.1: Existing verification scheme process.

- **Support Only a Small, Predefined Subset of Samples**: Some schemes use pre-embedded patterns for verification, like backdoor triggers, limiting the process to only those samples. This means the number of verifications must be considered before training, and once these samples are used up, no further verification is possible.

- **Lacking Robustness Verification**: These schemes focus solely on verifying the unlearning process immediately and lack robustness, meaning the model cannot be fine-tuned or pruned for new demands after unlearning.

- **Model Performance Degradation and Security Risks**: Verification schemes based on data pollution and backdoor attacks rely on poisoned samples, inheriting the drawbacks of poisoning methods. This can harm model performance and introduce security risks, limiting their adoption for verification.

6.2.2 Threat Model and Goals

In MLaaS, there are two main roles: data providers and model providers. Data providers also act as verifiers of unlearning, while model providers are responsible for executing it. The data provider shares its dataset with the model provider, who trains a model using a learning algorithm. After training, beyond making regular predictions, the data provider can submit unlearning requests to remove their data from the model. However, model

providers may not always be fully trustworthy in performing unlearning, either because this process is time-consuming, or large-scale unlearning requests could negatively impact model performance [48, 110]. The background and goals of data providers are as follows:

- **Background:** Data providers only have the ability to upload their training data or send prediction, unlearning requests to the model providers.
- **Goal:** After submitting the unlearning request, data providers want to confirm whether their data has been truly unlearned from the model.

Meanwhile, the background and goals of model providers are as follows:

- **Background:** Model provider can collect training data from the data provider and train the model.
- **Goal:** After receiving the unlearning requests from data provider, model providers prefer to avoid executing unlearning as much as possible to protect their own interests.

This chapter proposes a method for data providers to verify whether their samples have been successfully unlearned from the trained model. Specifically, we have four aims related to machine unlearning verification.

- **No Pre-defined Modifications**: Develop a verification scheme that enables data providers to confirm the execution of the unlearning process without depending on any pre-defined sample-level modifications.
- **More Sample Coverage**: Design a scheme supporting unlearning verification for nearly all samples involved in the training process.
- **Robustness Verification**: Address the need for robustness by supporting immediate post-unlearning verification and enabling verification in scenarios where the model undergoes further changes after unlearning (e.g., further fine-tuning or model pruning).
- **Preserving Model Usability**: Ensure that the verification scheme does not negatively impact: model performance, security, and training efficiency.

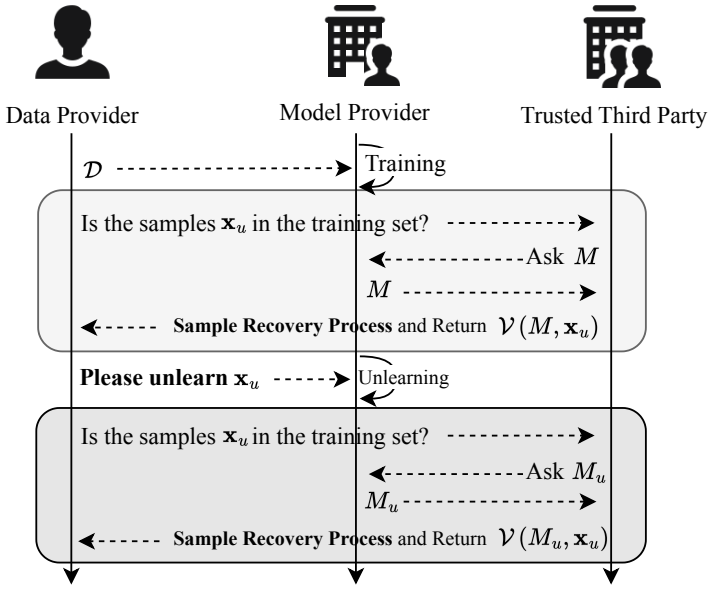


Figure 6.2: Our verification scheme process.

We assume that the data provider conducts verification with the help of a trusted third party. This assumption reflects real-world MLaaS scenarios where data providers often lack the capability to independently verify the unlearning processes [48]. The trusted third party is granted access to the trained model for verification purposes upon receiving unlearning requests. Additionally, we assume that the model provider may carry out further model modifications (e.g., fine-tuning and pruning) after executing the unlearning process.

6.3 Methodology

6.3.1 Overview

Current unlearning verification schemes typically depend on additional information to verify the unlearning process, such as pre-embedded backdoors [48, 110]. However, these schemes become ineffective if this additional information is disrupted during subsequent model fine-tuning or pruning. Therefore, it is crucial to verify the unlearning process only using the model itself and investigate if the information can still be recovered from the model parameters post-unlearning.

In this chapter, we propose a robustness verification scheme without prior modifications, named UnlearnGuard, which verifies machine unlearning based only on the

model and possesses robustness properties. As illustrated in Figure 6.2, UnlearnGuard directly verifies the unlearning process by examining whether the model parameters contain information about the unlearning samples. Before and after unlearning, the data provider sends a verification request about $\mathbf{x}_u$ to trusted third party. The trusted third party ask model from model provider and attempts to recover $\mathbf{x}_u$ from it. Based on two recovery results, the data provider can determine if the data was truly unlearned. Our approach distinguishes UnlearnGuard from the scheme illustrated in Figure 6.1 as UnlearnGuard relies only on the model itself rather than any prior sample-level modifications.

We describe our scheme based on the following two subsections: *unlearning verification process* and *sample recovery process*. In Section 6.3.2, we introduce the main workflow of our entire unlearning verification process, which includes the pre-unlearning process and the post-unlearning process, encompassing all steps shown in Figure 6.2. Next, in Section 6.3.3, we explain how to recover actual unlearning samples from the model, which is the most important part of our scheme and supports the workflow of Section 6.3.2.

6.3.2 Unlearning Verification Process

In this section, we will describe the whole workflow of our unlearning verification process, including three steps.

6.3.2.1 Model Training and Pre-verification

Data providers can submit their datasets to MLaaS for model training, enhancing accessibility and efficiency in machine learning deployment. After training, with the help of a trusted third party, data providers can perform pre-verification to confirm the presence of their data within the trained model. While this verification step enhances the comprehensiveness of our proposed scheme, it is not mandatory for practical implementations.

To conduct this, data providers send sample $\mathbf{x_u}$ to a trusted third party, requesting confirmation of $\mathbf{x_u}$'s presence in the model. The trusted third party then asks the MLaaS model provider for the trained model M and attempts to recover the samples using the scheme described in Section 6.3.3. This verification process outputs a result $\mathcal{V}(M, \mathbf{x_u})$.

Algorithm 10: Unlearning Verification Process

Input: Model M and sample $\mathbf{x}_u$ that need to be unlearned
Output: Whether model provider has unlearned samples $\mathbf{x}_u$

1 **Data provider executes**:
2 Query if $\mathbf{x}_u$ is in the model.
3 $\mathcal{V}(M, \mathbf{x_u}) \leftarrow$ receive results from trust third party
4 Sending the unlearning request regarding samples $\mathcal{D}_u$.
5 Re-query if $\mathbf{x}_u$ is in the model.
6 $\mathcal{V}(M_u, \mathbf{x_u}) \leftarrow$ receive results from trust third party
7 **if** $\mathcal{V}(M, \mathbf{x_u})$ == *True and* $\mathcal{V}(M_u, \mathbf{x_u})$ == *False* **then**
8 **return** *Unlearning process has executed.*
9 **if** $\mathcal{V}(M, \mathbf{x_u})$ == *True and* $\mathcal{V}(M_u, \mathbf{x_u})$ == *True* **then**
10 **return** *Unlearning process has not executed.*
11 **Trusted third party executes**:
12 **Upon** receiving a verification request from data provider
13 Send a request to model provider to query the model.
14 Try to recovering sample $\mathbf{x}_u$ based on Algorithm 11.
15 **return** *recover results* $\mathcal{V}()$.
16 **Model provider executes**:
17 **Upon** receiving a request to query the model:
18 **return** M.

6.3.2.2 Unlearning Request and Execution

To initiate unlearning, the data provider sends a request regarding samples $\mathbf{x}_u$. The model provider locates and removes sample $\mathbf{x}_u$ from the dataset and executes the unlearning process.

6.3.2.3 Re-query and Verification

Following the unlearning request, the data provider again inquires about $\mathbf{x_u}$'s presence in the model. The trusted third party returns the output $\mathcal{V}(M_u, \mathbf{x_u})$, where M_u represents the updated model after unlearning. The data provider then compares $\mathcal{V}(M, \mathbf{x_u})$ and $\mathcal{V}(M_u, \mathbf{x_u})$ to determine whether the model provider has successfully executed the unlearning operation. In Algorithm 10, we provide a detailed process for this process.

In Algorithm 10, lines 2-3, denote the pre-verification process. Specially, the initial result $\mathcal{V}(M, \mathbf{x_u})$ should be True. Line 4 shows the data provider sending an unlearning request, followed by another query for the result related to $\mathbf{x}_u$ (lines 5-6). If $\mathcal{V}(M_u, \mathbf{x_u})$ is False, it confirms that the model provider has truly executed the unlearning operation (lines 7-8); otherwise, it suggests that the model provider has not executed the

unlearning operation (lines 9-10). In our scheme, we define the function $\mathcal{V}$ as:

$$\mathcal{V}(\mathbf{x}_u, \{\mathbf{x}_i\}_{i=1}^m) = \bigvee_{i=1}^{m} \left(\text{SSIM}(\mathbf{x}_u, \mathbf{x}_i) \gg \eta\right) \tag{6.1}$$

where $\vee$ denotes the logical OR operation and m is the number of recovered samples.

6.3.3 Sample Recovery Process

In the previous section, we explained the process of verifying unlearning, mainly based on comparing the recovery results before and after unlearning. This section describes how to recover actual training samples encoded within a trained model, leveraging implicit bias and date reconstruction.

We begin by studying simple models before advancing to the analysis of more complex deep models in Section 6.3.4. Let $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^d \times \{-1, 1\}$ be a binary classification training dataset. Consider a neural network $M(\boldsymbol{\theta}; \cdot)$: $\mathbb{R}^d \to \mathbb{R}$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^p$. For a given loss function $\ell : \mathbb{R} \to \mathbb{R}$, the empirical loss of $M(\boldsymbol{\theta}; \cdot)$ on the dataset D is given by: $\mathscr{L}(\boldsymbol{\theta}) := \sum_{i=1}^n \ell(y_i M(\boldsymbol{\theta}; \mathbf{x}_i))$. Let us consider the logistic loss, also known as binary cross-entropy, which is defined as $\ell(q) = \log(1 + e^{-q})$.

Directly recovering samples from the above-defined model poses significant challenges. To address this, we reformulate model training problem into a maximum margin problem based on the implicit bias theory discussed by Ji et al. [62] and Lyu et al. [87], which simplifies the process of recovering samples from the model.

Theorem 1 (Paraphrased from Ji et al. [62], Lyu et al. [87]): Consider a homogeneous neural network $M(\boldsymbol{\theta}; \cdot)$. When minimizing the logistic loss over a binary classification dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ using gradient flow, and assuming there exists a time t_0 such that $\mathscr{L}(\boldsymbol{\theta}(t_0)) < 1$, then, gradient flow converges in direction to a first-order stationary point of the following maximum-margin problem:

$$\min_{\boldsymbol{\theta}'} \frac{1}{2} \left\|\boldsymbol{\theta}'\right\|^2 \quad \text{s.t.} \quad \forall i \in [n] \quad y_i M\left(\boldsymbol{\theta}'; \mathbf{x}_i\right) \geq 1 \tag{6.2}$$

In this theorem, homogeneous networks are defined with respect to their parameters $\boldsymbol{\theta}$. Specifically, a network M is considered homogeneous if there exists $L > 0$ such that for any $\alpha > 0$, $\boldsymbol{\theta}$ and $\mathbf{x}$, the relationship $M(\alpha\boldsymbol{\theta}; \mathbf{x}) = \alpha^L M(\boldsymbol{\theta}; \mathbf{x})$ holds. This means that scaling the parameters by any factor $\alpha > 0$ results in scaling the outputs by α^L. Gradient flow is said to converge in the direction to $\tilde{\boldsymbol{\theta}}$ if $\lim_{t\to\infty} \frac{\boldsymbol{\theta}(t)}{\|\boldsymbol{\theta}(t)\|} = \frac{\tilde{\boldsymbol{\theta}}}{\|\tilde{\boldsymbol{\theta}}\|}$, where $\boldsymbol{\theta}(t)$ is the parameter vector at time t. $\mathscr{L}(\theta(t_0)) < 1$ means that there exists time t_0 at which the network classifies all the samples correctly.

This theorem describes how optimization algorithms, such as gradient descent, tend to converge to specific solutions that can be formalized as Karush-Kuhn-Tucker (KKT) conditions, enabling data reconstruction from the model using those conditions [50]. The reconstruction loss can be defined as following. Derivation details can be found in Section 6.3.4.

$$\begin{aligned} & L_{\text{reconstruct}}\left(\{\mathbf{x}_i\}_{i=1}^m, \{\lambda_i\}_{i=1}^m\right) \\ & \quad = \alpha_1 L_{\text{stationary}} + \alpha_2 L_\lambda + \alpha_3 L_{\text{additional}} \\ & \text{s.t.} \quad L_{\text{stationary}} = \left\| \boldsymbol{\theta} - \sum_{i=1}^m \lambda_i y_i \nabla_{\boldsymbol{\theta}} M(\boldsymbol{\theta}; \mathbf{x}_i) \right\|_2^2 \\ & \quad L_\lambda = \sum_{i=1}^m \max\{-\lambda_i, 0\} \end{aligned} \tag{6.3}$$

where m denotes the cardinality of the sample set to be reconstructed, typically set to $m \geq 2n$ in their experimental setting. The loss $L_{\text{stationary}}$ represents the stationarity condition satisfied by the parameters at the KKT point, while L_λ represents the dual feasibility condition. $L_{\text{additional}}$ denotes some supplementary constraints predicated on image attributes, such as ensuring pixel values remain between $[0, 1]$.

Using the above $L_{\text{reconstruct}}$ can recover the actual sample from the model, which is our main purpose. However, only using this loss to reconstruct samples exhibits limitations in the context of partial verification. Specifically, it is constrained to reconstructing only those training samples that almost lie on the decision boundary margin. Consequently, when employed to verify the unlearning process, its efficacy is limited to a subset of samples,Äîthose situated on the margin. To expand the scope of reconstruction, we introduce a novel loss term, denoted as the prior information loss, L_{prior}.

The introduction of this new loss term aims to incorporate classification information pertaining to the samples targeted for recovery. Let $\{\mathbf{x}_i\}_{i=1}^m$ represent the samples recovered based on the aforementioned scheme. Our objective is to ensure that the subsequently recovered samples $\{\hat{\mathbf{x}}_i\}_{i=1}^m$ exhibit greater similarity to their counterparts in the training data space:

$$\text{maximize}\left(M_{y_i}(\hat{\mathbf{x}}_i; \boldsymbol{\theta})\right) = -\left|M_{y_i}(\hat{\mathbf{x}}_i; \boldsymbol{\theta})\right| \tag{6.4}$$

Specifically, we aim to maximize $M_{y_i}(\hat{\mathbf{x}}_i; \boldsymbol{\theta})$ assigned to the predicted class for each sample $\hat{\mathbf{x}}_i$ by minimizing its negative absolute value. It is noteworthy that this optimization process does not utilize the original labels. Instead, it optimizes the model's output

Algorithm 11: Sample Recovery Process

Input: The trained model M, iteration number T_1 and T_2.
Output: Recovered Samples $\mathbf{x}^{'}$

1 **for** $t = 0; t < T_1; t++$ **do**
2 $L_{\text{reconstruct}} = \alpha_1 L_{\text{stationary}} + \alpha_2 L_\lambda$
3 Optimizing $\left(\{\mathbf{x}_i\}_{i=1}^m, \{\lambda_i\}_{i=1}^m\right)$ based on $L_{\text{reconstruct}}$.
4 $\{\hat{\mathbf{x}}_i\}_{i=1}^m = \{\mathbf{x}_i\}_{i=1}^m$
5 **for** $t = 0; t < T_2; t++$ **do**
6 $L_{\text{reconstruct}} = \alpha_1 L_{\text{stationary}} + \alpha_2 L_\lambda + \alpha_3 L_{\text{prior}}$
7 Optimizing $\left(\{\hat{\mathbf{x}}_i\}_{i=1}^m, \{\lambda_i\}_{i=1}^m\right)$ based on $L_{\text{reconstruct}}$.
8 Projecting within $[\{\mathbf{x}_i\}_{i=1}^m - \epsilon, \{\mathbf{x}_i\}_{i=1}^m + \epsilon]$
9 **return** $\mathbf{x}^{'} = \hat{\mathbf{x}}$

confidence (logits) for samples $\{\hat{\mathbf{x}}_i\}_{i=1}^m$, aligning with our hypothesis of implementing unlearning verification using only the model parameters.

Finally, we define our recovery loss as:

$$\begin{aligned} L_{\text{reconstruct}} &= \alpha_1 L_{\text{stationary}} + \alpha_2 L_\lambda + \alpha_3 L_{\text{prior}} \\ \text{s.t.} \quad L_{\text{stationary}} &= \left\| \boldsymbol{\theta} - \sum_{i=1}^m \lambda_i y_i \nabla_{\boldsymbol{\theta}} M(\boldsymbol{\theta}; \hat{\mathbf{x}}_i) \right\|_2^2 \\ L_\lambda = \sum_{i=1}^m \max\{-\lambda_i, 0\} \quad & L_{\text{prior}} = -\left| M_{y_i}(\hat{\mathbf{x}}_i; \boldsymbol{\theta}) \right| \end{aligned} \tag{6.5}$$

where we eliminate the $L_{\text{additional}}$ loss term and incorporate L_{prior} to enhance the fidelity of recovered samples.

To mitigate excessive deviation of the newly recovered samples $\{\hat{\mathbf{x}}_i\}_{i=1}^m$ from $\{\mathbf{x}_i\}_{i=1}^m$ in the data space, we introduce a projection function, `project_to_bounds`, applied after each optimization epoch. This function constrains the pixel values of $\{\hat{\mathbf{x}}_i\}_{i=1}^m$ to within the range $[\{\mathbf{x}_i\}_{i=1}^m - \epsilon, \{\mathbf{x}_i\}_{i=1}^m + \epsilon]$. Our recovery algorithm is shown in Algorithm 11.

In lines 1-3, we initially optimize $\{\mathbf{x}_i\}_{i=1}^m$ and $\{\lambda_i\}_{i=1}^m$ utilizing the loss $L_{\text{reconstruct}} = \alpha_1 L_{\text{stationary}} + \alpha_2 L_\lambda$. This preliminary phase ensures that $\{\mathbf{x}_i\}_{i=1}^m$ achieves a basic level of recovery. Subsequently, we proceed to optimize $\{\mathbf{x}_i\}_{i=1}^m$ further to recover samples $\{\hat{\mathbf{x}}_i\}_{i=1}^m$ using the augmented loss function (lines 5-8). At each optimization epoch, we project the recovered samples onto a constrained space (line 8). This ensures they remain within a specified range of the pre-recovered samples while capturing more essential features required for effective unlearning verification.

6.3.4 Theoretical Analysis

Our machine unlearning verification scheme builds upon existing works [13, 50, 62, 87], extending their insights to the context of machine unlearning. We provide a rigorous theoretical basis for our verification method, incorporating concepts from optimization theory and functional analysis.

Let $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^d \times \{-1, 1\}$ be a binary classification training dataset. Consider a neural network $M(\boldsymbol{\theta}; \cdot)$: $\mathbb{R}^d \to \mathbb{R}$ parameterized by $\theta \in \mathbb{R}^p$. The training process with a logistic loss $\ell(q) = \log(1 + e^{-q})$ can be viewed as an implicit margin maximization problem.

As we discussed in Theorem 1, for a homogeneous ReLU neural network $M(\boldsymbol{\theta}; \cdot)$, gradient flow on the logistic loss

$$\mathscr{L}(\boldsymbol{\theta}) := \sum_{i=1}^{n} \ell\left(y_i M\left(\boldsymbol{\theta}; \mathbf{x}_i\right)\right) \tag{6.6}$$

converges in direction to a KKT point of:

$$\min_{\boldsymbol{\theta}'} \frac{1}{2} \left\|\boldsymbol{\theta}'\right\|^2 \tag{6.7}$$

subject to

$$\forall i \in [n] \quad y_i M\left(\boldsymbol{\theta}'; \mathbf{x}_i\right) \geq 1 \tag{6.8}$$

The KKT conditions for this problem yield [62, 87]:

$$\begin{aligned} &\tilde{\boldsymbol{\theta}} = \textstyle\sum_{i=1}^{n} \lambda_i y_i \nabla_{\boldsymbol{\theta}} M\left(\tilde{\boldsymbol{\theta}}; \mathbf{x}_i\right) \\ &\forall i \in [n], y_i M\left(\tilde{\boldsymbol{\theta}}; \mathbf{x}_i\right) \geq 1 \\ &\lambda_1, \ldots, \lambda_n \geq 0 \\ &\forall i \in [n], \quad \lambda_i = 0 \text{ if } y_i M\left(\tilde{\boldsymbol{\theta}}; \mathbf{x}_i\right) \neq 1 \end{aligned} \tag{6.9}$$

Based on these KKT conditions, we can formulate a reconstruction method [13, 50]. The goal is to find a set of $\{\mathbf{x}_i\}_{i=1}^m$ and $\{\lambda_i\}_{i=1}^m$ that satisfy following equation:

$$\min_{\{\mathbf{x}_i\}_{i=1}^m, \{\lambda_i\}_{i=1}^m} L_{reconstruct} = \alpha_1 L_{stationary} + \alpha_2 L_{\lambda} \tag{6.10}$$

where:

$$L_{stationary} = \left\| \tilde{\boldsymbol{\theta}} - \sum_{i=1}^{m} \lambda_i y_i \nabla_{\boldsymbol{\theta}} M\left(\tilde{\boldsymbol{\theta}}; \mathbf{x}_i\right) \right\|^2 \tag{6.11}$$

$$L_\lambda = \sum_{i=1}^{m} \max\{-\lambda_i, 0\} \tag{6.12}$$

Let U be an unlearning operator that unlearns $(\mathbf{x}_i, y_i)$ from a model with parameters $\tilde{\boldsymbol{\theta}}$ to $\boldsymbol{\theta}'$:

$$\boldsymbol{\theta}' = U(M, \mathscr{D}, (x_k, y_k)) \tag{6.13}$$

To verify unlearning, we compare the recovered samples before and after unlearning those samples. Let F be the index set of samples to be unlearned. The parameters of the model after unlearning $\boldsymbol{\theta}'$ should also satisfy the KKT conditions:

$$\begin{aligned} &\boldsymbol{\theta}' = \textstyle\sum_{i \notin F} \lambda_i^{'} y_i \nabla_{\boldsymbol{\theta}} M\left(\boldsymbol{\theta}'; \mathbf{x}_i\right) \\ &\forall i \in [n], y_i M\left(\boldsymbol{\theta}'; \mathbf{x}_i\right) \geq 1 \\ &\lambda_1^{'}, \ldots, \lambda_n^{'} \geq 0 \\ &\forall i \in [n], \quad \lambda_i^{'} = 0 \text{ if } y_i M\left(\boldsymbol{\theta}'; \mathbf{x}_i\right) \neq 1 \end{aligned} \tag{6.14}$$

And the corresponding $L_{stationary}$ is

$$\min_{\{\mathbf{x}_i^{'}\}_{i=1}^m, \{\lambda_i^{'}\}_{i=1}^m} L_{reconstruct} = \alpha_1 L_{stationary} + \alpha_2 L_\lambda \tag{6.15}$$

where:

$$L_{stationary} = \left\| \boldsymbol{\theta}' - \sum_{i=1}^{m} \lambda_i^{'} y_i \nabla_{\boldsymbol{\theta}} M\left(\boldsymbol{\theta}'; \mathbf{x}_i^{'}\right) \right\|^2 \tag{6.16}$$

$$L_\lambda = \sum_{i=1}^{m} \max\{-\lambda_i^{'}, 0\} \tag{6.17}$$

The verification involves:

1. Performing sample recovery on both the original model $\tilde{\boldsymbol{\theta}}$ and the unlearned model $\boldsymbol{\theta}'$, obtaining recovery samples sets $X = \{\mathbf{x}_i\}_{i=1}^m$ and $X' = \{\mathbf{x}_i'\}_{i=1}^m$.

2. Calculating the differences between recovered samples:

$$D_i = \|\mathbf{x}_i - \mathbf{x}_i'\| \tag{6.18}$$

3. Comparing the differences for samples that unlearned and retained samples:

$$E[D_i \mid i \in F] \text{ vs } E[D_i \mid i \notin F] \tag{6.19}$$

If the unlearning process is effective, we expect:

$$E[D_i \mid i \in F] > E[D_i \mid i \notin F] \tag{6.20}$$

This expectation arises because:

1. For $i \notin F$, both $\mathbf{x}_i$ and $\mathbf{x}_i'$ should satisfy the constraints of equations 6.9 and 6.14, leading to small differences.

2. For $i \in F$, $\mathbf{x}_i$ satisfies equation 6.9 but not 6.14, while $\mathbf{x}_i'$ does not satisfy equation 6.9 but satisfies equation 6.14, resulting in larger differences.

Additionally, recent work proposed by Buzaglo et al. [13] has extended the reconstruction method in [50] to multi-class classification scenarios. For a dataset $D_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^d \times [C]$, where C is the number of classes, and a neural network $M(\boldsymbol{\theta};\cdot)$: $\mathbb{R}^d \to \mathbb{R}^C$. Then, the KKT conditions for the multi-class problem yield:

$$\begin{aligned}
&\boldsymbol{\theta} = \textstyle\sum_{i=1}^n \sum_{j \neq y_i} \lambda_{i,j} \nabla_{\boldsymbol{\theta}} \left(\Delta_{y_i,j}(\boldsymbol{\theta})\right) \\
&\Delta_{y_i,j}(\boldsymbol{\theta}) = M_{y_i}(\boldsymbol{\theta};\mathbf{x}_i) - M_j(\boldsymbol{\theta};\mathbf{x}_i) \\
&\forall i \in [n], \forall j \in [C] \setminus \{y_i\} : \Delta_{y_i,j}(\boldsymbol{\theta}) \geq 1 \\
&\forall i \in [n], \forall j \in [C] \setminus \{y_i\} : \lambda_{i,j} \geq 0 \\
&\forall i \in [n], \forall j \in [C] \setminus \{y_i\} : \lambda_{i,j} = 0 \text{ if } \Delta_{y_i,j}(\boldsymbol{\theta}) \neq 1
\end{aligned} \tag{6.21}$$

The corresponding loss $L_{stationary}$ can be formulated as:

$$\left| \boldsymbol{\theta} - \sum_{i=1}^{m} \lambda_i \nabla_{\boldsymbol{\theta}} [M_{y_i}(\boldsymbol{\theta};\mathbf{x}i) - \max_{j \neq y_i} M_j(\boldsymbol{\theta};\mathbf{x}_i)] \right|^2 \tag{6.22}$$

In this case, the above-proof process is also applicable. We will use this loss in our experimental evaluation when dealing with multi-classification tasks.

This theoretical framework utilizes the implicit bias inherent in neural network training and Karush-Kuhn-Tucker (KKT) conditions of margin maximization to evaluate the unlearning process. By analyzing the difference between recovered samples before and after the unlearning process, we can determine if the model provider has successfully removed the targeted samples from the model.

6.4 Performance Evaluation

6.4.1 Experiment Setup

To evaluate our scheme, we utilize four widely-used image datasets: MNIST[2], Fashion MNIST[3], CIFAR-10[4] and SVHN[5].

6.4.1.1 Baseline Methods

We compare our scheme against the following established methods:

- **Membership Inference Attacks-Based Scheme**: We employ the approach proposed in [83] and [44].
- **Backdoor-Based Scheme**: We adopt the scheme proposed by Li et al. [77] as a baseline method.
- **Model Inversion Attack-Based Scheme**: We adopt the scheme proposed by Graves et al. [44].
- **Accuracy-Based Scheme**: We also compare the accuracy of our unlearned model against the original model on both the unlearning and remaining samples [12, 120].

6.4.1.2 Metrics

We consider the following metrics to separately evaluate the baseline scheme and our scheme:

- For membership inference attacks-based scheme [83], we use the success rate of attacking samples as our metric, defined as $INA = \frac{TP}{TP+FN}$ where TP is the number of samples predicted to be in the training set, and $TP + FN$ is the total tested samples. Ideally, INA should be close to 100% before unlearning and approach 0% after unlearning.
- For backdoor-based scheme [77], we also use the attack success rate, ASR, used in [77] to evaluate the unlearning results. Ideally, before unlearning, ASR should be close to 100%. After unlearning, ASR should approach 0%.

[2]http://yann.lecun.com/exdb/mnist/
[3]http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/
[4]https://www.cs.toronto.edu/ kriz/cifar.html
[5]http://ufldl.stanford.edu/housenumbers/

- For model inversion attack-based scheme [44], we directly show the recovered samples. Ideally, before unlearning, those samples should contain discernible information about the class targeted for unlearning. After unlearning, Those samples should appear dark, jumbled, and dissimilar from the unlearning class, indicating the successful unlearning of class-specific information.

- For our scheme, we evaluate it using both qualitative and quantitative perspectives. Qualitative: visual inspection of recovered samples, similar to the model inversion attack-based schemes. Quantitative: we use the SSIM to evaluate the similarity between recovered and original unlearning samples. SSIM values range from -1 to 1, with higher values indicating greater similarity.

6.4.2 Verification Results

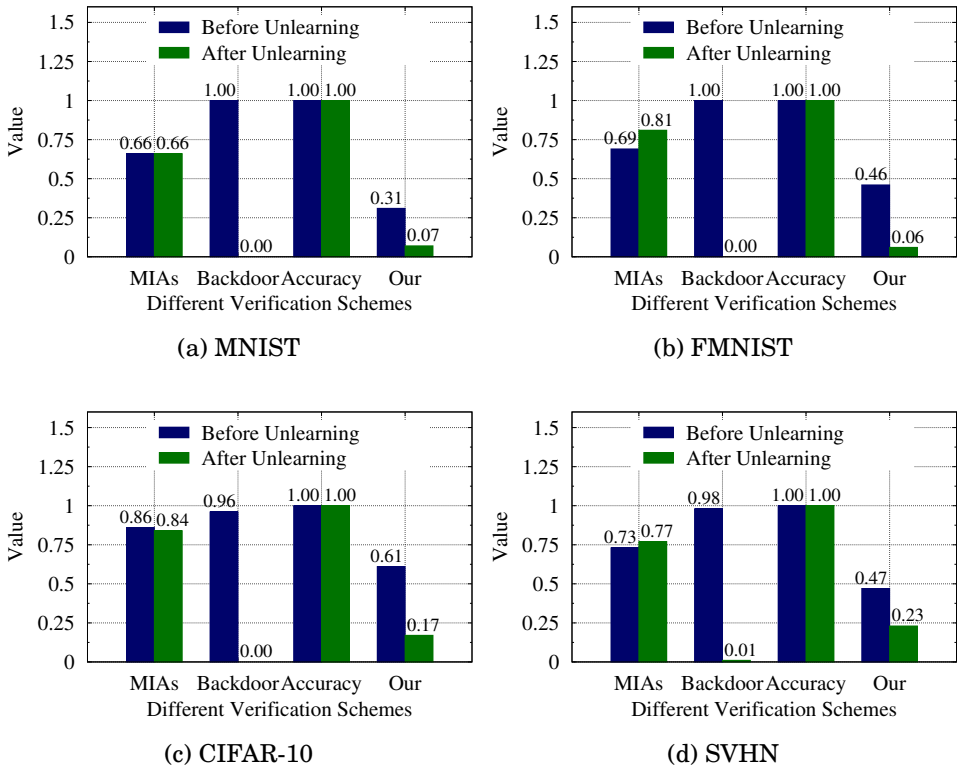
Our evaluation includes sample and class level unlearning requests. To ensure consistency with existing studies [48], we employ retraining from scratch as our unlearning method.

6.4.2.1 Sample-Level Unlearning Verification

Figure 6.3 and 6.4 show our experimental results. Figure 6.3 illustrates the results for unlearning samples before and after the unlearning process, while Figure 6.4 shows results for samples not unlearned. Each sub-figure demonstrates the performance of various verification schemes across different datasets. The Y-axis represents different metrics depending on the scheme: INA for membership inference attacks (MIAs)-based schemes, ASR for backdoor-based schemes, accuracy for accuracy-based schemes, and SSIM for our proposed verification scheme. The conclusion can be summary as follows:

- MIAs-based and Accuracy-based Schemes: Before unlearning, success rates for identifying both unlearning and remaining samples are high, indicating that MIAs effectively identify samples used in model training. However, after unlearning, there's no significant reduction in attack performance for either sample type. This suggests that MIAs cannot reliably distinguish whether training samples have been successfully unlearned. Similar results can be observed from the results of accuracy-based schemes.

- Backdoor-based Scheme: Following backdoor embedding, the accuracy of classifying both unlearning and remaining samples as targets is very high. After unlearn-



(a) MNIST (b) FMNIST

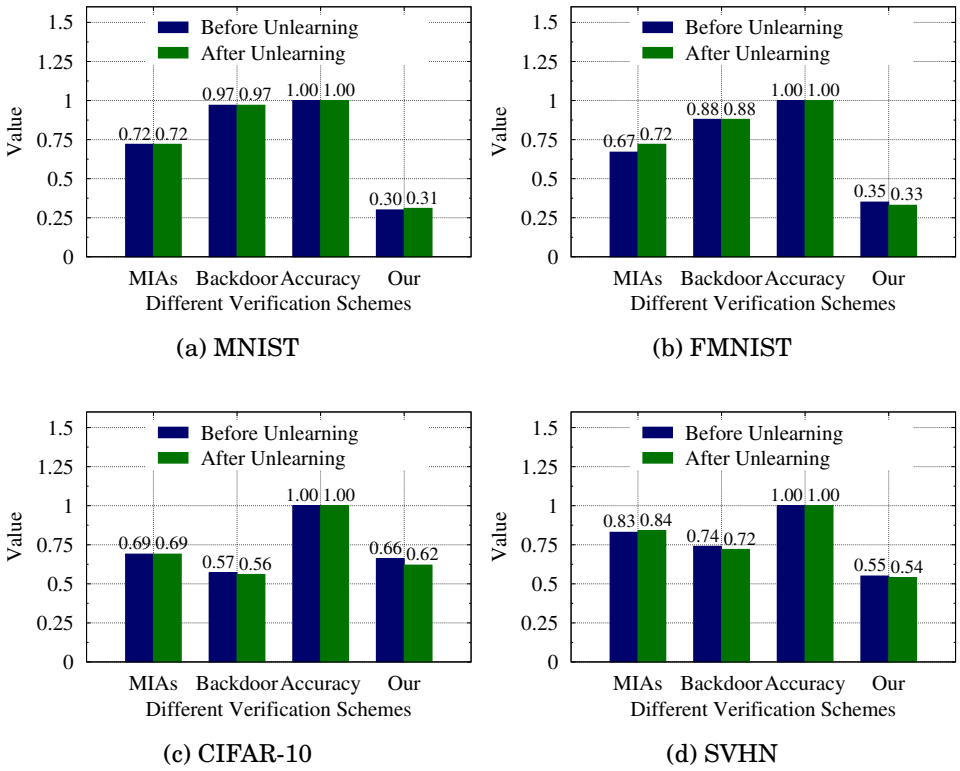
(c) CIFAR-10 (d) SVHN

Figure 6.3: Verification results for unlearning samples under sample-level unlearning requests across different schemes.

ing, the classification accuracy for unlearning samples with the backdoor drops to nearly 0%, while remaining higher for other samples. This indicates that the backdoor-based verification scheme effectively confirms the unlearning process, demonstrating the successful removal of backdoor samples associated with unlearning.

- Our Proposed Scheme: Prior to unlearning, all recovered samples show high similarity based on the SSIM metric. After the unlearning process, SSIM values for unlearning samples decrease significantly, while remaining high for other samples. Those SSIM values before and after unlearning demonstrate that our scheme effectively distinguishes between unlearned and retained samples.

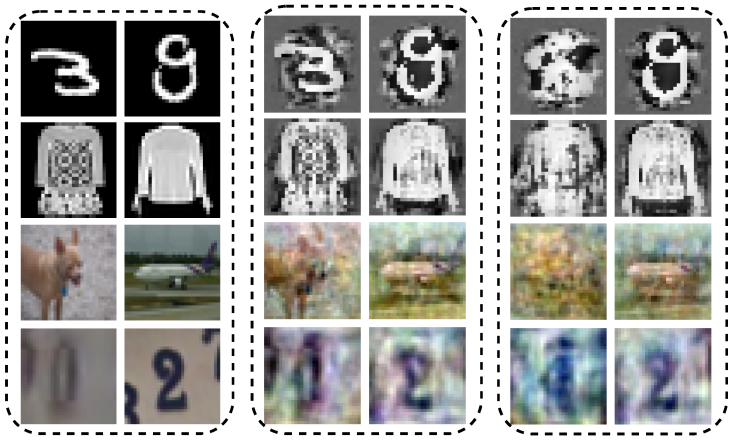
We also show some original training samples and recovered samples extracted using our scheme in Figure 6.5. Each column shows unlearning samples on the left and the remaining samples on the right. Before unlearning, our scheme effectively recovers all samples. After unlearning, the samples that were most similar to those samples targeted



(a) MNIST (b) FMNIST

(c) CIFAR-10 (d) SVHN

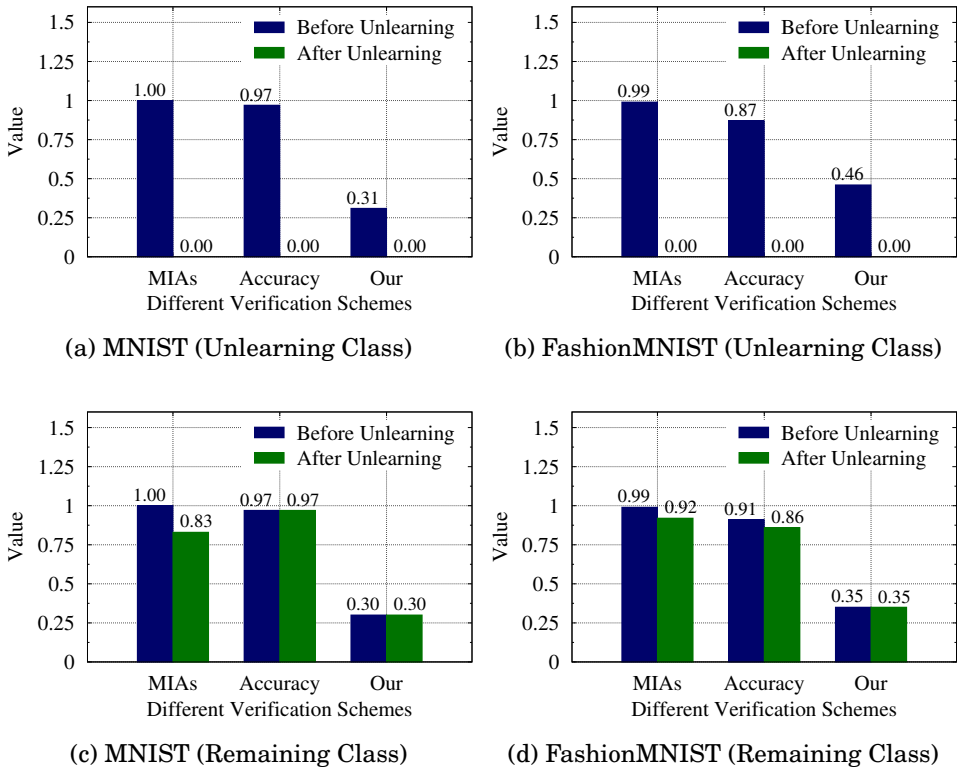
Figure 6.4: Verification results for remaining samples under sample-level unlearning requests across different schemes.



(a) Original Sample (b) Before Unlearning (c) After Unlearning

Figure 6.5: Original and recovered samples under the sample-level unlearning requests.

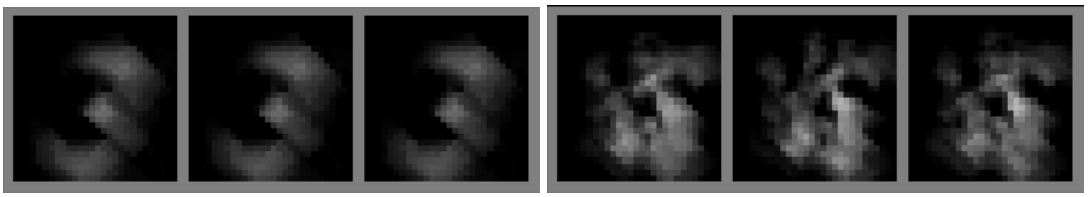
for unlearning no longer contained relevant information, while for the remaining samples, our scheme still recovers images similar to the originals. This demonstrates our scheme's ability to effectively distinguish between unlearned and retained samples, validating it



(a) MNIST (Unlearning Class) (b) FashionMNIST (Unlearning Class)

(c) MNIST (Remaining Class) (d) FashionMNIST (Remaining Class)

Figure 6.6: Verification results under class-level unlearning requests.



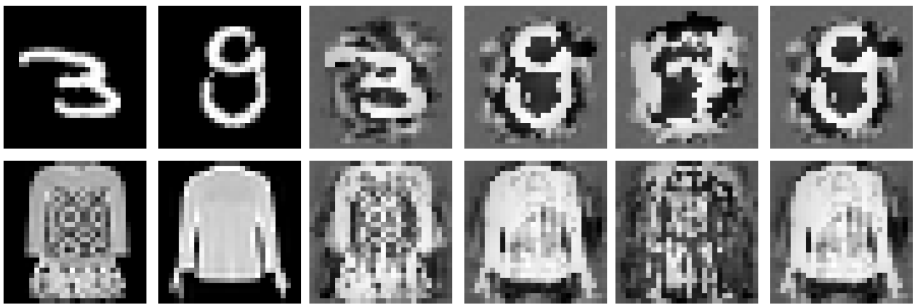
(a) Before Unlearning (b) After Unlearning

Figure 6.7: Reconstructed class representatives based on model inversion attack before and after unlearning in the class-level unlearning setting.

can be used as a verification method.

6.4.2.2 Class-Level Unlearning Verification

We set the remaining class to 9 for the MNIST dataset and to 0 for the FashionMNIST dataset. The unlearning class is selected as 3 for both datasets. Figure 6.6 shows the quantitative results of our evaluation. Specifically, Figure 6.6a and Figure 6.6c show the results for the unlearning class and the remaining class in the MNIST dataset, while Figure 6.6b and Figure 6.6d illustrate the changes in the unlearning class and the



(a) Original Sample (b) Before Unlearning (c) After Unlearning

Figure 6.8: Original and recovered samples based on our scheme under the class-level unlearning request.

remaining class for the FashionMNIST dataset.

Results. For samples in the class that need to be unlearned, MIA-based, accuracy-based, and our proposed methods all show significant changes (see Figures 6.6a and 6.6b). For remaining class, no significant changes are observed across all three approaches (see Figures 6.6c and 6.6d). This indicates that all methods are effective for class-level unlearning verification.

Figure 6.7 and 6.8 show some original and recovered samples extracted using the model inversion attack-based scheme and our scheme. In Figure 6.7, before unlearning, the model inversion attack reconstructs representatives containing information about the unlearning class. After unlearning, it produces dark, jumbled images with almost no information about the unlearning class. Similarly, for our scheme, the recovered sample after unlearning all samples in the unlearning class also becomes dark and jumbled, which illustrates that our scheme can also be used in verifying class-level unlearning requests.

Summary. The above experiments demonstrate that the proposed verification scheme effectively validates both sample-level and class-level unlearning requests. The method consistently performs well across these different granularity levels, showcasing its applicability and reliability in various scenarios.

6.4.3 Robustness Evaluation

As highlighted in Section 6.2.1, the robustness of unlearning verification scheme is crucial for long-term deployment. A robust verification scheme should remain effective even after fine-tuning or pruning the models when the training or unlearning process is finished. In this Section, we evaluate the robustness of our proposed scheme, comparing

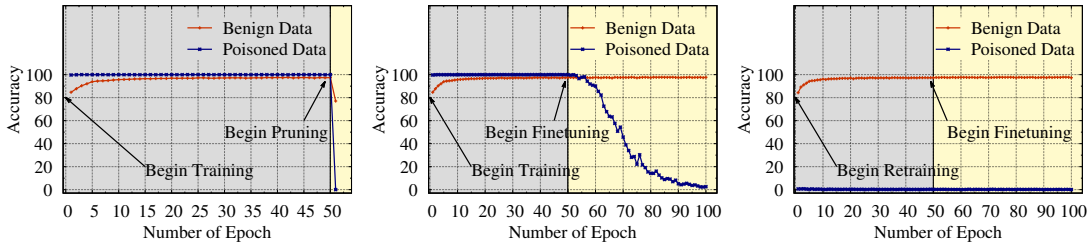
it with the backdoor-based verification scheme [77]. We use the same experimental settings described in Section 6.4.2.1 and re-evaluate these schemes after performing the following operations:

1. Training model → Pruning model
2. Training model → Fine-tuning model
3. Training model → Unlearning → Fine-tuning model

For pruning, we randomly prune 20% of the parameters in each layer. For fine-tuning, we train the model using the original training samples and correct labels, maintaining the initial training configuration. For unlearning, we use retraining from scratch [48]. The results of the backdoor-based scheme are illustrated in Figure 6.9, while Figure 6.10 presents the results of our proposed scheme.

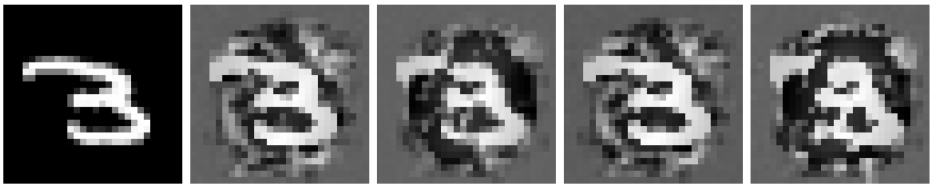
Results: As shown in Figure 6.9a, when the model undergoes pruning after training, the backdoor-based verification scheme becomes ineffective, as evidenced by the near-zero performance on the poison dataset. This indicates that the backdoor-based verification loses its robustness in the face of pruning operations, rendering it unusable. Similarly, in Figure 6.9b, when fine-tuning is performed using the original, unperturbed training samples, the accuracy of the backdoor also decreases, indicating that fine-tuning also compromises the robustness of the backdoor-based verification scheme. Lastly, Figure 6.9c illustrates that when the model is finetuned with previously unlearned samples after unlearning, the backdoor-based verification scheme also fails due to the absence of the backdoor pattern in those samples. In conclusion, the backdoor-based method relies on pre-prepared samples for verification. If the backdoor pattern is not embedded in advance, if those patterns are disrupted after training, or if the backdoor has been previously used, subsequent verification will be infeasible.

Figure 6.10 shows the results obtained from our scheme. Figure 6.10a illustrate the original training sample, while other Figures show samples recovered after various processes: initial training (Figure 6.10b), pruning (Figure 6.10c), fine-tuning (Figure 6.10d), and unlearning followed by fine-tuning (Figure 6.10e). Unlike the backdoor-based scheme, our scheme effectively recovers training samples even after the model has been subjected to these modifications. This consistent performance demonstrates that our method exhibits a higher degree of robustness, maintaining its effectiveness across various post-training adjustments where the backdoor-based scheme fails.



(a) Results of Setting (1) (b) Results of Setting (2) (c) Results of Setting (3)

Figure 6.9: Evaluation results of robustness for backdoor-based verification scheme.



(a) Original (b) Baseline (c) Setting (1) (d) Setting (2) (e) Setting (3)

Figure 6.10: Evaluation results of robustness for our scheme.

Summary. The above experiments demonstrate that, as the backdoor-based verification scheme depends on pre-prepared patterns for verification, if those pre-prepared patterns are not embedded in advance, are disrupted after training, or have been previously used, subsequent verification becomes infeasible. In contrast, our method demonstrates consistent performance and a higher degree of robustness, maintaining effectiveness across various post-training adjustments.

6.4.4 Ablation Study and Analysis of Verification Range

As discussed in Section 6.2.1, the ability to verify more samples is crucial in MLaaS unlearning verification. This section evaluates the number of verifications supported by our proposed scheme compared to the method introduced in [77]. Furthermore, in Section 6.3.3, we add a new constraint to improve the quality of recovered samples and provide more samples used for unlearning verification. In this Section, we also do a comparative analysis between our enhanced scheme and the data reconstruction scheme proposed in [13].

We follow experimental settings used in [13] to construct our comparison experiment. Specifically, we use the experimental setting in 6.4.2.1 and select the MNIST dataset. We first recover samples based on the loss $L_{\text{reconstruct}} = \alpha_1 L_{\text{stationary}} + \alpha_2 L_\lambda$ and record the result as original. Then we add our new loss $\alpha_3 L_{prior}$ to $L_{\text{reconstruct}}$ and continue to

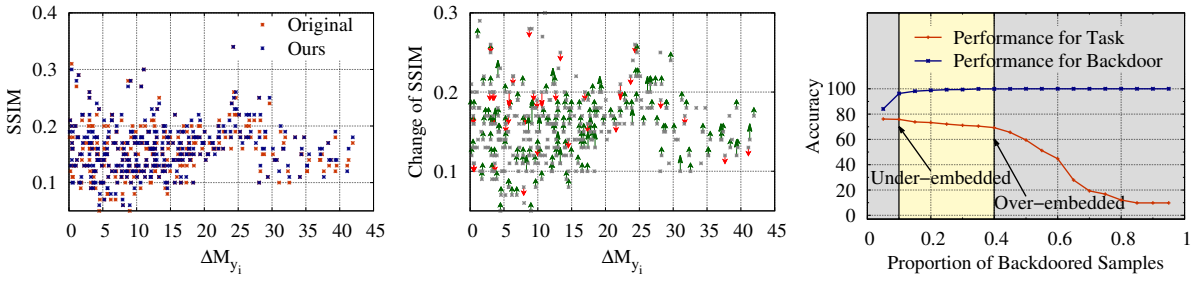
recover samples. We set $\alpha_1 = 1$, $\alpha_1 = 1$ and $\alpha_3 = 1$ and record the recover samples as ours. We adopted the same other hyperparameters provided in [13]. To evaluate the quality of our recovered samples, we employ the same evaluation method in [13]: for each sample in the original training dataset we search for its nearest neighbor in the recovered samples and measure the similarity using SSIM. A higher SSIM value indicates better recovery quality. In Figure 6.11a, we plot the recovery quality (measured by SSIM) against the sample's distance from the decision boundary. This distance is calculated based on:

$$\Delta M_{y_i} = M_{y_i}(\mathbf{x}_i;\boldsymbol{\theta}) - \max_{j \neq y_i} M_j(\mathbf{x}_i;\boldsymbol{\theta})$$

where $M_{y_i}(\mathbf{x}_i;\boldsymbol{\theta})$ is the logit for the true class, and $\max_{j \neq y_i} M_j(\mathbf{x}_i;\boldsymbol{\theta})$ is the maximum logit among all other classes. In Figure 6.11b, we show the change of SSIM for each corresponding original sample. For the backdoor-based scheme [77], we choose the experimental setting in 6.4.2.1 and use the same number of the training dataset in Buzaglo et al. [13]. The corresponding results are shown in Figure 6.11c.

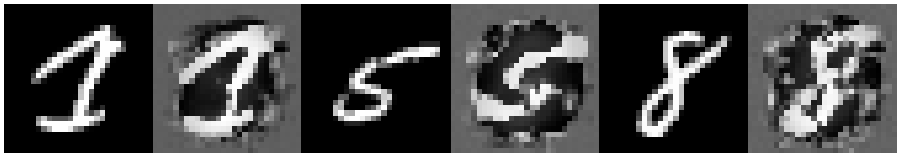
Results. Figures 6.11a demonstrates that both schemes proposed in [13] and ours successfully recover various samples from the model, aligning with the findings reported in [13]. In addition, our scheme shows an improvement, with a larger proportion of recovered samples (blue points) exhibiting greater similarity to the training dataset compared to the original scheme (red points) in Figure 6.11a. Figure 6.11b visually represents this improvement, with green arrows indicating samples where our scheme achieves higher similarity, gray points representing unchanged similarity, and red arrows denoting decreased similarity. This improved similarity between recovered samples and training samples provides more samples for our subsequent verification processes.

Additionally, as shown in Figure 6.11c, when the proportion of backdoored samples used for training is less than 5%, their performance is inadequate for unlearning verification. Effective verification is achieved only when the proportion of backdoored samples reaches 10%. This suggests that, for the MNIST dataset, approximately 10% of the total training samples (500 in total) is necessary for a single verification. Given the total number of samples, it only supports 10 times for verification. Furthermore, as the number of backdoored samples increases, the model,Äôs performance will significantly decrease since the samples that have not been backdoored become less. For example, from Figure 6.11c, when only 60% training samples for original task training, the performance of the trained model will decrease to only 44%. For our scheme, we show some recovered samples in Figure 6.12. It can be seen that when the SSIM of the recovered image equals 0.15, the recovered samples still partially retain information about the



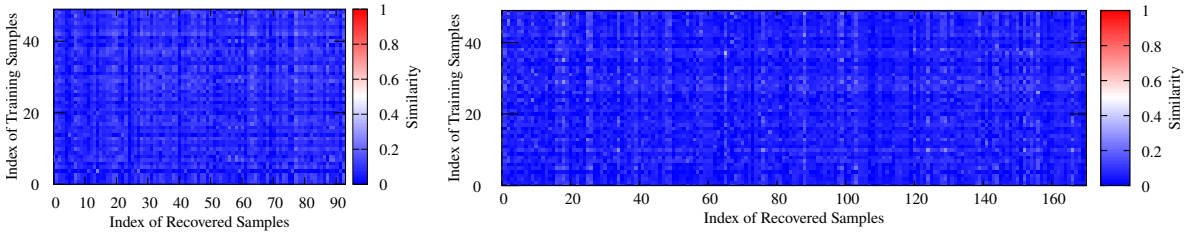
(a) SSIM Distribution (b) SSIM Change (c) Backdoor-based Scheme

Figure 6.11: Evaluation results of the recovery quality (measured by SSIM) against the sample's distance from the decision boundary.



(a) SSIM=0.3 (b) SSIM=0.2 (c) SSIM=0.15

Figure 6.12: Evaluation results of verification range.



(a) Before Fine-tuning (b) After Fine-tuning

Figure 6.13: Evaluation results of unlearning scheme based on relabel-based fine-tuning.

original sample. From Figure 6.11a, we observe that approximately 255 samples have an SSIM greater than 0.15. This suggests that our scheme can support over 40 times the number of verifications compared to backdoor-based methods, without compromising the model's original performance.

Summary. Our enhanced sample recovery method significantly improves the quality of samples available for verification. In addition, it also allows a substantially higher number of verifications compared to existing backdoor-based methods, while maintaining model's performance on its primary task.

(a) Original Training Samples

(b) Recovered after Relabel-based Unlearning

Figure 6.14: Original samples and corresponding recovered samples after random label-based fine-tuning.

6.4.5 Fine-tuning is not a Solution

Currently, many machine unlearning schemes achieve their goals through fine-tuning process [130]. This typically involves manipulating the samples targeted for unlearning, such as assigning random labels, and then fine-tuning the model with those relabeled samples [20, 22, 112, 131]. Experimental evaluations in these works, using membership inference attacks, model inversion attacks, backdoor attack and accuracy-based metrics, have suggested successful unlearning. However, the question remains: Is this truly the case?

Intuitively, any fine-tuning-based unlearning scheme involving unlearning samples should be considered incomplete, as samples targeted for unlearning are still processed by the model during the fine-tuning stage. To verify this hypothesis, we used the experimental setting described in Section 6.4.2.2, focusing on the MNIST dataset and replacing the unlearning method with relabel-based fine-tuning [20, 22]. We use our proposed scheme to recover samples both before and after fine-tuning. For evaluation, we select all recovered samples that can be correctly classified as class targeted for unlearning using the model before unlearning. Then, we calculate the SSIM between each recovered sample and its nearest original sample. Figure 6.13 shows the SSIM results, while Figure 6.14 shows some recovered samples after fine-tuning.

Results. As shown in Figure 6.13a, before performing relabel-based finetuning, the SSIM between some recovered samples and training samples is very high, indicating that the trained model indeed contains some information about the class that needs to be unlearned. Figure 6.13b also reveals similar results, with many recovered samples exhibiting high similarity to the training samples (see both recovered samples in Fig-

ure 6.14). Furthermore, after fine-tuning, the number of recovered samples unexpectedly increases, as the model retains memory not only of the original training samples but also of the newly added fine-tuning samples. All those results suggest that even after unlearning, the model still retains information about the samples that need to be unlearned. Therefore, relabel-based fine-tuning is not an effective unlearning solution.

6.5 Summary

This chapter introduced a novel approach to machine unlearning verification, addressing the challenges of prior sample-level modifications while considering both robustness and supporting verification on a much larger set. Inspired by the existing works in implicit bias and date reconstruction, we propose an optimization-based method for recovering actual training samples from models. This enables verification of unlearning by comparing samples recovered before and after the unlearning process. We provide theoretical analyses of our scheme's effectiveness. Experimental results demonstrate robust verification capabilities while supporting verifying a large number of samples, marking a significant advancement in machine unlearning research. In addition, our machine unlearning verification scheme revealed that relabeling fine-tuning methods do not fully remove, but rather amplify, the influence of targeted samples, challenging previous findings. This suggests that our verification scheme can further enhance the reliability of machine unlearning.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

This thesis proposes multiple unlearning schemes to address challenges across various scenarios. We also consider the need for unlearning verification when interacting with an untrusted model provider. Experimental results demonstrate that our scheme can effectively remove targeted information while maintaining model utility.

To begin with, Chapter 2 reviews basic concepts in machine unlearning. We first introduce the fundamental concepts and diverse targets of machine unlearning. Then, we propose a novel taxonomy that outlines the core principles underlying each approach. Furthermore, we review a wide range of existing works, evaluating their strengths and limitations within the context of our proposed taxonomy. We also highlight the critical role of unlearning verification and review various existing strategies.

Chapter 3 proposes a federated unlearning scheme to remove the influence of specific classes from a trained model. By identifying key channels via ablation analysis, we introduce two methods that fine-tune only influential parameters using perturbation data. Experiments show that our approach efficiently erases target class effects while preserving the accuracy of remaining data.

Chapter 4 proposes a novel machine unlearning scheme-target unlearning-that selectively removes specific information from a trained model. We define the concept of target unlearning, highlight its challenges, and identify influential parameters using explainability techniques. A pruning-based method is introduced to erase target-related

information, while an essential graph is constructed to preserve key parameters important to the remaining data. Experimental results show that our approach effectively removes target influence while maintaining model accuracy and efficiency.

Chapter 5 proposes an unlearning scheme that selectively removes feature-level information from a trained model. We address two scenarios: feature unlearning with known annotations using adversarial learning, and unlearning without annotations via a re-encoding and fine-tuning strategy. Experimental results demonstrate that our method effectively erases feature influence while preserving model accuracy and efficiency.

Chapter 6 takes the first step toward addressing machine unlearning verification in the presence of untrustworthy model providers. We first identify limitations in existing membership inference and backdoor-based verification by presenting two bypass strategies. To overcome these issues, we propose a perturbation-based verification scheme that generates influential sample pairs-trigger samples for unlearning and reaction samples for verification-where the latter's behavior depends on the presence of the former. We provide a theoretical analysis and demonstrate that our method resists bypassing techniques while preserving model utility and ensuring robust verification performance.

7.2 Future Work

As research continues to evolve, we propose the following directions for future work:

1. **The universality of unlearning solutions.** Unlearning schemes with higher compatibility need to be explored. As development progresses, machine unlearning schemes supporting different models and unlearning data types have been proposed in various fields. For example, Zhang et al. [137] provided an unlearning scheme in image retrieval, while Chen et al. [24] considered graph unlearning problem. However, most of the current unlearning schemes are limited to a specific scenario. They are mostly designed to leverage the special characteristics of a particular learning process or training scheme [24, 81, 99]. Although it is feasible to design an appropriate unlearning scheme for every model, this is an inefficient approach that would require many manual interventions [125]. Therefore, universality unlearning schemes should be not only applicable to different model structures and training methods, but also to different types of training datasets, such as graphs, images, text, or audio data. The data pruning-based scheme is an existing and effective approach that could achieve universality unlearning purposes based on ensemble learning techniques [11]. However, this method breaks the correlation

relationships in some scenarios, which is not suitable for models that require correlation information to complete training.

2. **The security of machine unlearning** Unlearning schemes should ensure the security of any data, especially the unlearned dataset. Recently, existing research has shown that the unlearning operation not only does not reduce the risk of user privacy leakage but actually increases this risk [10]. These attack schemes mainly compare the models before and after the unlearning process. Thus, a membership inference attack or a poisoning attack would reveal a significant amount of detailed information about the unlearned samples [23, 89]. In order to counteract such attacks, Neel et al. [94] have proposed a protection method based on Gaussian perturbation in their unlearning scheme. In addition, many previous unlearning schemes rely on the remaining dataset, intermediate cached model's parameters. However, they do not consider the security of this intermediate information and whether an attack would recover any information about the unlearned samples [11, 100]. Therefore, the design of further unlearning schemes needs to consider that any before and after models should not expose any information about the samples that need to be unlearned. Further, the security of the data cached during the unlearning process also needs to be explored.

3. **Information synchronization:** Similar to process synchronization in operating systems, machine unlearning may create information synchronization problems [7, 101]. Since machine unlearning is usually computationally costly, the model provider may not be able to complete the unlearning process immediately. In the interim, how to handle incoming prediction requests deserves careful consideration. Consider that, if predictions continue to be returned prior to the model's update, unlearned data may be revealed. However, if all requests for prediction are rejected until the unlearning process is completed, model utility and service standards will surely suffer. Therefore, how to handle prediction requests within this interval needs comprehensive consideration.

4. **Game-theory-based balance:** Game theory has been a booming field with several representative privacy-preserving techniques coming out in the past decade [74]. There are many schemes involving privacy-preserving solutions based on game theory that trade-off data privacy and utility issues [31, 78]. For a model provider, machine unlearning is also a trade-off between model performance, and user privacy, where an over-unlearning strategy may lead to performance degradation,

while insufficient protection may lead to privacy leaks. Can we formalize the unlearning problem as a game between two players: a model provider and a data provider? If so, we could provide a game model between these two entities and determine a set of strategies and utilities to figure out how to perform unlearning operations that maintain the model's performance to the maximum extent possible. Such an approach could also protect the user's sensitive data from being leaked. These are open issues that need to be explored further.

BIBLIOGRAPHY

[1] *California Consumer Privacy Act (CCPA)*, 2018.
[Online; Retrieved in March 19, 2022 from `https://oag.ca.gov/privacy/ccpa`.

[2] *General Data Protection Regulation (GDPR)*, 2018.
[Online; Retrieved in December 15, 2022 from `https://data.stats.gov.cn`.

[3] *Japan - Data Protection Overview(JDPO)*, 2019.
[Online; Retrieved in March 19, 2022 from `https://www.dataguidance.com/notes/japan-data-protection-overview`.

[4] *Consumer Privacy Protection Act(CPPA)*, 2022.
[Online; Retrieved in March 19, 2022 from `https://blog.didomi.io/en-us/canada-data-privacy-law`.

[5] N. AGARWAL, B. BULLINS, AND E. HAZAN, *Second-order stochastic optimization for machine learning in linear time*, J. Mach. Learn. Res., 18 (2017), pp. 116:1–116:40.

[6] R. A. AMJAD, K. LIU, AND B. C. GEIGER, *Understanding neural networks and individual neuron importance via information-ordered cumulative ablation*, IEEE Trans. Neural Networks Learn. Syst., 33 (2022), pp. 7842–7852.

[7] R. L. BAGRODIA, *Process synchronization: Design and performance evaluation of distributed algorithms*, IEEE Trans. Software Eng., 15 (1989), pp. 1053–1065.

[8] D. BAU, B. ZHOU, A. KHOSLA, A. OLIVA, AND A. TORRALBA, *Network dissection: Quantifying interpretability of deep visual representations*, in 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, IEEE Computer Society, 2017, pp. 3319–3327.

[9] T. BAUMHAUER, P. SCHÖTTLE, AND M. ZEPPELZAUER, *Machine unlearning: linear filtration for logit-based classifiers*, Mach. Learn., 111 (2022), pp. 3203–3226.

[10] S. Z. BÉGUELIN, L. WUTSCHITZ, S. TOPLE, V. RÜHLE, A. PAVERD, O. OHRIMENKO, B. KÖPF, AND M. BROCKSCHMIDT, *Analyzing information leakage of updates to natural language models*, in CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020, J. Ligatti, X. Ou, J. Katz, and G. Vigna, eds., ACM, 2020, pp. 363–375.

[11] L. BOURTOULE, V. CHANDRASEKARAN, C. A. CHOQUETTE-CHOO, H. JIA, A. TRAVERS, B. ZHANG, D. LIE, AND N. PAPERNOT, *Machine unlearning*, in 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021, IEEE, 2021, pp. 141–159.

[12] J. BROPHY AND D. LOWD, *Machine unlearning for random forests*, in Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, M. Meila and T. Zhang, eds., vol. 139 of Proceedings of Machine Learning Research, PMLR, 2021, pp. 1092–1104.

[13] G. BUZAGLO, N. HAIM, G. YEHUDAI, G. VARDI, Y. OZ, Y. NIKANKIN, AND M. IRANI, *Deconstructing data reconstruction: Multiclass, weight decay and general losses*, in NeurIPS, 2023.

[14] J. A. CALANDRINO, A. KILZER, A. NARAYANAN, E. W. FELTEN, AND V. SHMATIKOV, *"you might also like: " privacy risks of collaborative filtering*, in 32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA, IEEE Computer Society, 2011, pp. 231–246.

[15] Y. CAO AND J. YANG, *Towards making systems forget with machine unlearning*, in 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015, IEEE Computer Society, 2015, pp. 463–480.

[16] N. CARLINI, C. LIU, Ú. ERLINGSSON, J. KOS, AND D. SONG, *The secret sharer: Evaluating and testing unintended memorization in neural networks*, in 28th USENIX Security Symposium, Santa Clara, CA, USA, August 14-16, 2019, N. Heninger and P. Traynor, eds., USENIX Association, 2019, pp. 267–284.

[17] N. CARLINI, F. TRAMÈR, E. WALLACE, M. JAGIELSKI, A. HERBERT-VOSS, K. LEE, A. ROBERTS, T. B. BROWN, D. SONG, Ú. ERLINGSSON, A. OPREA,

AND C. RAFFEL, *Extracting training data from large language models*, in 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021, M. Bailey and R. Greenstadt, eds., USENIX Association, 2021, pp. 2633–2650.

[18] A. CHATTOPADHYAY, A. SARKAR, P. HOWLADER, AND V. N. BALASUBRAMANIAN, *Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks*, in 2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018, Lake Tahoe, NV, USA, March 12-15, 2018, IEEE Computer Society, 2018, pp. 839–847.

[19] C. CHEN, F. SUN, M. ZHANG, AND B. DING, *Recommendation unlearning*, in WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022, F. Laforest, R. Troncy, E. Simperl, D. Agarwal, A. Gionis, I. Herman, and L. Médini, eds., ACM, 2022, pp. 2768–2777.

[20] H. CHEN, T. ZHU, X. YU, AND W. ZHOU, *Machine unlearning via null space calibration*, in IJCAI-24, 8 2024.
Main Track.

[21] K. CHEN, Y. HUANG, AND Y. WANG, *Machine unlearning via GAN*, CoRR, abs/2111.11869 (2021).

[22] M. CHEN, W. GAO, G. LIU, K. PENG, AND C. WANG, *Boundary unlearning: Rapid forgetting of deep networks via shifting the decision boundary*, in CVPR, IEEE, 2023.

[23] M. CHEN, Z. ZHANG, T. WANG, M. BACKES, M. HUMBERT, AND Y. ZHANG, *When machine unlearning jeopardizes privacy*, in CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021, Y. Kim, J. Kim, G. Vigna, and E. Shi, eds., ACM, 2021, pp. 896–911.

[24] ——, *Graph unlearning*, in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA, November 7-11, 2022, H. Yin, A. Stavrou, C. Cremers, and E. Shi, eds., ACM, 2022, pp. 499–513.

[25] X. CHEN, R. MOTTAGHI, X. LIU, S. FIDLER, R. URTASUN, AND A. L. YUILLE, *Detect what you can: Detecting and representing objects using holistic models and*

body parts, in 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014, 2014, pp. 1979–1986.

[26] D. CHENG, F. YANG, S. XIANG, AND J. LIU, *Financial time series forecasting with multi-modality graph neural network*, Pattern Recognit., 121 (2022), p. 108218.

[27] H. CHENG, Y. SHI, L. WU, Y. GUO, AND N. XIONG, *An intelligent scheme for big data recovery in internet of things based on multi-attribute assistance and extremely randomized trees*, Inf. Sci., 557 (2021), pp. 66–83.

[28] Y. CHOI, G. FARNADI, B. BABAKI, AND G. V. DEN BROECK, *Learning fair naive bayes classifiers by discovering and eliminating discrimination patterns*, in The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 10077–10084.

[29] V. S. CHUNDAWAT, A. K. TARUN, M. MANDAL, AND M. S. KANKANHALLI, *Can bad teaching induce forgetting? unlearning in deep networks using an incompetent teacher*, in Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023, B. Williams, Y. Chen, and J. Neville, eds., AAAI Press, 2023, pp. 7210–7217.

[30] ——, *Zero-shot machine unlearning*, IEEE Trans. Inf. Forensics Secur., 18 (2023), pp. 2345–2354.

[31] L. CUI, Y. QU, M. R. NOSOUHI, S. YU, J. NIU, AND G. XIE, *Improving data utility through game theory in personalized differential privacy*, J. Comput. Sci. Technol., 34 (2019), pp. 272–286.

[32] Z. DIAO, X. WANG, D. ZHANG, Y. LIU, K. XIE, AND S. HE, *Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting*, in The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI

2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, AAAI Press, 2019, pp. 890–897.

[33] C. DWORK AND V. FELDMAN, *Privacy-preserving prediction*, CoRR, abs/1803.10266 (2018).

[34] D. L. FELPS, A. D. SCHWICKERATH, J. D. WILLIAMS, T. N. VUONG, A. BRIGGS, M. HUNT, E. SAKMAR, D. D. SARANCHAK, AND T. SHUMAKER, *Class clown: Data redaction in machine unlearning at enterprise scale*, in Proceedings of the 10th International Conference on Operations Research and Enterprise Systems, ICORES 2021, Online Streaming, February 4-6, 2021, G. H. Parlier, F. Liberatore, and M. Demange, eds., SCITEPRESS, 2021, pp. 7–14.

[35] R. FONG AND A. VEDALDI, *Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks*, in 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, Computer Vision Foundation / IEEE Computer Society, 2018, pp. 8730–8738.

[36] E. FOSCH-VILLARONGA, P. KIESEBERG, AND T. LI, *Humans forget, machines remember: Artificial intelligence and the right to be forgotten*, Comput. Law Secur. Rev., 34 (2018), pp. 304–313.

[37] J. FOSTER, S. SCHOEPF, AND A. BRINTRUP, *Fast machine unlearning without retraining through selective synaptic dampening*, in Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada, M. J. Wooldridge, J. G. Dy, and S. Natarajan, eds., AAAI Press, 2024, pp. 12043–12051.

[38] M. FREDRIKSON, S. JHA, AND T. RISTENPART, *Model inversion attacks that exploit confidence information and basic countermeasures*, in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015, I. Ray, N. Li, and C. Kruegel, eds., ACM, 2015, pp. 1322–1333.

[39] A. GINART, M. Y. GUAN, G. VALIANT, AND J. ZOU, *Making AI forget you: Data deletion in machine learning*, in Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds., 2019, pp. 3513–3526.

[40] A. GOLATKAR, A. ACHILLE, A. RAVICHANDRAN, M. POLITO, AND S. SOATTO, *Mixed-privacy forgetting in deep networks*, in IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021, Computer Vision Foundation / IEEE, 2021, pp. 792–801.

[41] A. GOLATKAR, A. ACHILLE, AND S. SOATTO, *Eternal sunshine of the spotless net: Selective forgetting in deep networks*, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020, Computer Vision Foundation / IEEE, 2020, pp. 9301–9309.

[42] ——, *Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations*, in Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXIX, A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, eds., vol. 12374 of Lecture Notes in Computer Science, Springer, 2020, pp. 383–398.

[43] C. GRATTON, N. K. D. VENKATEGOWDA, R. ARABLOUEI, AND S. WERNER, *Privacy-preserved distributed learning with zeroth-order optimization*, IEEE Trans. Inf. Forensics Secur., 17 (2022), pp. 265–279.

[44] L. GRAVES, V. NAGISETTY, AND V. GANESH, *Amnesiac machine learning*, in Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, AAAI Press, 2021, pp. 11516–11524.

[45] A. GRØNLUND, L. KAMMA, AND K. G. LARSEN, *Near-tight margin-based generalization bounds for support vector machines*, in Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, vol. 119 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 3779–3788.

[46] C. GUO, T. GOLDSTEIN, A. Y. HANNUN, AND L. VAN DER MAATEN, *Certified data removal from machine learning models*, in Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, vol. 119 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 3832–3842.

[47] S. GUO, Y. WANG, Q. LI, AND J. YAN, *DMCP: differentiable markov channel pruning for neural networks*, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020, Computer Vision Foundation / IEEE, 2020, pp. 1536–1544.

[48] Y. GUO, Y. ZHAO, S. HOU, C. WANG, AND X. JIA, *Verifying in the dark: Verifiable machine unlearning by using invisible backdoor triggers*, IEEE TIFS, 19 (2024), pp. 708–721.

[49] V. GUPTA, C. JUNG, S. NEEL, A. ROTH, S. SHARIFI-MALVAJERDI, AND C. WAITES, *Adaptive machine unlearning*, in Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, eds., 2021, pp. 16319–16330.

[50] N. HAIM, G. VARDI, G. YEHUDAI, O. SHAMIR, AND M. IRANI, *Reconstructing training data from trained neural networks*, in NeurIPS, 2022.

[51] D. HE, Y. WANG, J. CAO, W. DING, S. CHEN, Z. FENG, B. WANG, AND Y. HUANG, *A network embedding-enhanced bayesian model for generalized community detection in complex networks*, Inf. Sci., 575 (2021), pp. 306–322.

[52] K. HE, X. ZHANG, S. REN, AND J. SUN, *Deep residual learning for image recognition*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, IEEE Computer Society, 2016, pp. 770–778.

[53] Y. HE, G. MENG, K. CHEN, J. HE, AND X. HU, *Deepobliviate: A powerful charm for erasing data residual memory in deep neural networks*, CoRR, abs/2105.06209 (2021).

[54] H. HU AND J. PANG, *Membership inference attacks against gans by leveraging over-representation regions*, in CCS '21: 2021 ACM SIGSAC Conference on

Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021, Y. Kim, J. Kim, G. Vigna, and E. Shi, eds., ACM, 2021, pp. 2387–2389.

[55] B. HUANG, X. LI, Z. SONG, AND X. YANG, *FL-NTK: A neural tangent kernel-based framework for federated learning analysis*, in Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, M. Meila and T. Zhang, eds., vol. 139 of Proceedings of Machine Learning Research, PMLR, 2021, pp. 4423–4434.

[56] C. HUANG, H. XU, Y. XU, P. DAI, L. XIA, M. LU, L. BO, H. XING, X. LAI, AND Y. YE, *Knowledge-aware coupled graph neural network for social recommendation*, in Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, AAAI Press, 2021, pp. 4115–4122.

[57] Y. HUANG, S. GUPTA, Z. SONG, K. LI, AND S. ARORA, *Evaluating gradient inversion attacks and defenses in federated learning*, in Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, eds., 2021, pp. 7232–7241.

[58] N. V. HUYNH, D. T. HOANG, D. N. NGUYEN, AND E. DUTKIEWICZ, *Joint coding and scheduling optimization for distributed learning over wireless edge networks*, IEEE J. Sel. Areas Commun., 40 (2022), pp. 484–498.

[59] A. IMTEAJ, U. THAKKER, S. WANG, J. LI, AND M. H. AMINI, *A survey on federated learning for resource-constrained iot devices*, IEEE Internet Things J., 9 (2022), pp. 1–24.

[60] Z. IZZO, M. A. SMART, K. CHAUDHURI, AND J. ZOU, *Approximate data deletion from machine learning models*, in The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event, A. Banerjee and K. Fukumizu, eds., vol. 130 of Proceedings of Machine Learning Research, PMLR, 2021, pp. 2008–2016.

[61] A. JACOT, F. GABRIEL, AND C. HONGLER, *Neural tangent kernel: convergence and generalization in neural networks (invited paper)*, in STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021, S. Khuller and V. V. Williams, eds., ACM, 2021, p. 6.

[62] Z. JI AND M. TELGARSKY, *Directional convergence and alignment in deep learning*, in NeurIPS, 2020.

[63] J. JIA, J. LIU, P. RAM, Y. YAO, G. LIU, Y. LIU, P. SHARMA, AND S. LIU, *Model sparsity can simplify machine unlearning*, in NeurIPS, 2023.

[64] P. JIANG, C. ZHANG, Q. HOU, M. CHENG, AND Y. WEI, *Layercam: Exploring hierarchical class activation maps for localization*, IEEE Trans. Image Process., 30 (2021), pp. 5875–5888.

[65] M. I. JORDAN AND T. M. MITCHELL, *Machine learning: Trends, perspectives, and prospects*, Science, 349 (2015), pp. 255–260.

[66] J. C. X. JUNIOR, A. A. FREITAS, T. B. LUDERMIR, A. F. NETO, AND C. A. S. BARRETO, *An evolutionary algorithm for automated machine learning focusing on classifier ensembles: An improved algorithm and extended results*, Theor. Comput. Sci., 805 (2020), pp. 1–18.

[67] Z. N. KESIMOGLU AND S. BOZDAG, *SUPREME: a cancer subtype prediction methodology integrating multiple biological datatypes using graph convolutional neural networks*, in BCB '21: 12th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, Gainesville, Florida, USA, August 1-4, 2021, H. Jiang, X. Huang, and J. Zhang, eds., ACM, 2021, p. 83:1.

[68] L. U. KHAN, W. SAAD, Z. HAN, E. HOSSAIN, AND C. S. HONG, *Federated learning for internet of things: Recent advances, taxonomy, and open challenges*, IEEE Commun. Surv. Tutorials, 23 (2021), pp. 1759–1799.

[69] M. KHOSRAVY, K. NAKAMURA, Y. HIROSE, N. NITTA, AND N. BABAGUCHI, *Model inversion attack by integration of deep generative models: Privacy-sensitive face generation from a face recognition system*, IEEE Trans. Inf. Forensics Secur., 17 (2022), pp. 357–372.

[70] P. W. KOH AND P. LIANG, *Understanding black-box predictions via influence functions*, in Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, D. Precup and Y. W. Teh, eds., vol. 70 of Proceedings of Machine Learning Research, PMLR, 2017, pp. 1885–1894.

[71] Y. KOIZUMI, S. MURATA, N. HARADA, S. SAITO, AND H. UEMATSU, *SNIPER: few-shot learning for anomaly detection to minimize false-negative rate with ensured true-positive rate*, in IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019, IEEE, 2019, pp. 915–919.

[72] M. KURMANJI, P. TRIANTAFILLOU, J. HAYES, AND E. TRIANTAFILLOU, *Towards unbounded machine unlearning*, in NeurIPS, 2023.

[73] L. C. LAMB, A. S. D'AVILA GARCEZ, M. GORI, M. O. R. PRATES, P. H. C. AVELAR, AND M. Y. VARDI, *Graph neural networks meet neural-symbolic computing: A survey and perspective*, in Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, C. Bessiere, ed., ijcai.org, 2020, pp. 4877–4884.

[74] S. LEONARDOS AND G. PILIOURAS, *Exploration-exploitation in multi-agent learning: Catastrophe theory meets game theory*, Artif. Intell., 304 (2022), p. 103653.

[75] X. LI, C. C. CAO, Y. SHI, W. BAI, H. GAO, L. QIU, C. WANG, Y. GAO, S. ZHANG, X. XUE, AND L. CHEN, *A survey of data-driven and knowledge-aware explainable AI*, IEEE Trans. Knowl. Data Eng., 34 (2022), pp. 29–49.

[76] Y. LI, S. LIN, B. ZHANG, J. LIU, D. S. DOERMANN, Y. WU, F. HUANG, AND R. JI, *Exploiting kernel sparsity and entropy for interpretable CNN compression*, in IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Computer Vision Foundation / IEEE, 2019, pp. 2800–2809.

[77] Y. LI, M. ZHU, X. YANG, Y. JIANG, T. WEI, AND S. XIA, *Black-box dataset ownership verification via backdoor watermarking*, IEEE Trans. Inf. Forensics Secur., 18 (2023), pp. 2318–2332.

[78] X. LIANG, Z. YAN, R. H. DENG, AND Q. ZHENG, *Investigating the adoption of hybrid encrypted cloud data deduplication with game theory*, IEEE Trans. Parallel Distributed Syst., 32 (2021), pp. 587–600.

[79] J. LIM, Y. KIM, B. KIM, C. AHN, J. SHIN, E. YANG, AND S. HAN, *Biasadv: Bias-adversarial augmentation for model debiasing*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2023, pp. 3832–3841.

[80] W. Y. B. LIM, N. C. LUONG, D. T. HOANG, Y. JIAO, Y. LIANG, Q. YANG, D. NIYATO, AND C. MIAO, *Federated learning in mobile edge networks: A comprehensive survey*, IEEE Commun. Surv. Tutorials, 22 (2020), pp. 2031–2063.

[81] G. LIU, X. MA, Y. YANG, C. WANG, AND J. LIU, *Federaser: Enabling efficient client-level data removal from federated learning models*, in 29th IEEE/ACM International Symposium on Quality of Service, IWQOS 2021, Tokyo, Japan, June 25-28, 2021, IEEE, 2021, pp. 1–10.

[82] Y. LIU, Z. MA, Y. YANG, X. LIU, J. MA, AND K. REN, *Revfrf: Enabling cross-domain random forest training with revocable federated learning*, IEEE Trans. Dependable Secur. Comput., 19 (2022), pp. 3671–3685.

[83] Y. LIU, R. WEN, X. HE, A. SALEM, Z. ZHANG, M. BACKES, E. D. CRISTOFARO, M. FRITZ, AND Y. ZHANG, *Ml-doctor: Holistic risk assessment of inference attacks against machine learning models*, in USENIX, 2022.

[84] Y. LIU, L. XU, X. YUAN, C. WANG, AND B. LI, *The right to be forgotten in federated learning: An efficient realization with rapid retraining*, in IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, London, United Kingdom, May 2-5, 2022, IEEE, 2022, pp. 1749–1758.

[85] Z. LIU, Y. JIANG, J. SHEN, M. PENG, K. LAM, X. YUAN, AND X. LIU, *A survey on federated unlearning: Challenges, methods, and future directions*, ACM Comput. Surv., 57 (2025), pp. 2:1–2:38.

[86] S. K. LO, Q. LU, C. WANG, H. PAIK, AND L. ZHU, *A systematic literature review on federated machine learning: From a software engineering perspective*, ACM Comput. Surv., 54 (2021), pp. 95:1–95:39.

[87] K. LYU AND J. LI, *Gradient descent maximizes the margin of homogeneous neural networks*, in ICLR, 2020.

[88] S. MALIAH AND G. SHANI, *Using pomdps for learning cost sensitive decision trees*, Artif. Intell., 292 (2021), p. 103400.

[89] N. G. MARCHANT, B. I. P. RUBINSTEIN, AND S. ALFELD, *Hard to forget: Poisoning attacks on certified machine unlearning*, in Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022, AAAI Press, 2022, pp. 7691–7700.

[90] J. MARTENS, *New insights and perspectives on the natural gradient method*, J. Mach. Learn. Res., 21 (2020), pp. 146:1–146:76.

[91] R. MEYES, M. LU, C. W. DE PUISEAU, AND T. MEISEN, *Ablation studies in artificial neural networks*, CoRR, abs/1901.08644 (2019).

[92] A. MOKHTARI AND A. RIBEIRO, *Global convergence of online limited memory BFGS*, J. Mach. Learn. Res., 16 (2015), pp. 3151–3181.

[93] A. S. MORCOS, D. G. T. BARRETT, N. C. RABINOWITZ, AND M. M. BOTVINICK, *On the importance of single directions for generalization*, in 6th International Conference on Learning Representations, ICLR, OpenReview.net, 2018.

[94] S. NEEL, A. ROTH, AND S. SHARIFI-MALVAJERDI, *Descent-to-delete: Gradient-based methods for machine unlearning*, in Algorithmic Learning Theory, 16-19 March 2021,, V. Feldman, K. Ligett, and S. Sabato, eds., vol. 132 of Proceedings of Machine Learning Research, PMLR, 2021, pp. 931–962.

[95] Q. P. NGUYEN, B. K. H. LOW, AND P. JAILLET, *Variational bayesian unlearning*, in Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds., 2020.

[96] C.-K. PENG, S. V. BULDYREV, S. HAVLIN, M. SIMONS, H. E. STANLEY, AND A. L. GOLDBERGER, *Mosaic organization of dna nucleotides*, Phys. Rev. E, 49 (1994), pp. 1685–1689.

[97] W. QIAN, C. ZHAO, H. SHAO, M. CHEN, F. WANG, AND M. HUAI, *Patient similarity learning with selective forgetting*, in IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2022, Las Vegas, NV, USA, December 6-8, 2022, D. A. Adjeroh, Q. Long, X. M. Shi, F. Guo, X. Hu, S. Aluru,

G. Narasimhan, J. Wang, M. Kang, A. Mondal, and J. Liu, eds., IEEE, 2022, pp. 529–534.

[98] Z. REN, Q. SUN, AND D. WEI, *Multiple kernel clustering with kernel k-means coupled graph tensor learning*, in Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, AAAI Press, 2021, pp. 9411–9418.

[99] S. SCHELTER, *"amnesia" - towards machine learning models that can forget user data very fast*, in 10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings, www.cidrdb.org, 2020.

[100] S. SCHELTER, S. GRAFBERGER, AND T. DUNNING, *Hedgecut: Maintaining randomised trees for low-latency machine unlearning*, in SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, G. Li, Z. Li, S. Idreos, and D. Srivastava, eds., ACM, 2021, pp. 1545–1557.

[101] G. SCHLAGETER, *Process synchronization in database systems*, ACM Trans. Database Syst., 3 (1978), pp. 248–271.

[102] A. SEKHARI, J. ACHARYA, G. KAMATH, AND A. T. SURESH, *Remember what you want to forget: Algorithms for machine unlearning*, in Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, eds., 2021, pp. 18075–18086.

[103] R. R. SELVARAJU, M. COGSWELL, A. DAS, R. VEDANTAM, D. PARIKH, AND D. BATRA, *Grad-cam: Visual explanations from deep networks via gradient-based localization*, in IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017, IEEE Computer Society, 2017, pp. 618–626.

[104] W. SHEN, Y. GUO, Y. WANG, K. ZHAO, B. WANG, AND A. L. YUILLE, *Deep differentiable random forests for age estimation*, IEEE Trans. Pattern Anal. Mach. Intell., 43 (2021), pp. 404–419.

[105] W. SHEN, Z. WEI, S. HUANG, B. ZHANG, J. FAN, P. ZHAO, AND Q. ZHANG, *Interpretable compositional convolutional neural networks*, in Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021, Z. Zhou, ed., ijcai.org, 2021, pp. 2971–2978.

[106] T. SHIBATA, G. IRIE, D. IKAMI, AND Y. MITSUZUMI, *Learning with selective forgetting*, in Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021, Z. Zhou, ed., ijcai.org, 2021, pp. 989–996.

[107] R. SHOKRI, M. STRONATI, C. SONG, AND V. SHMATIKOV, *Membership inference attacks against machine learning models*, in 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017, IEEE Computer Society, 2017, pp. 3–18.

[108] K. SIMONYAN AND A. ZISSERMAN, *Very deep convolutional networks for large-scale image recognition*, in 3rd ICLR, 2015.

[109] R. SINHA, R. K. PAL, AND R. K. DE, *Genseg and mr-genseg: A novel segmentation algorithm and its parallel mapreduce based approach for identifying genomic regions with copy number variations*, IEEE ACM Trans. Comput. Biol. Bioinform., 19 (2022), pp. 443–454.

[110] D. M. SOMMER, L. SONG, S. WAGH, AND P. MITTAL, *Athena: Probabilistic verification of machine unlearning*, Proc. Priv. Enhancing Technol., 2022 (2022), pp. 268–290.

[111] J. T. SPRINGENBERG, A. DOSOVITSKIY, T. BROX, AND M. A. RIEDMILLER, *Striving for simplicity: The all convolutional net*, in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings, Y. Bengio and Y. LeCun, eds., 2015.

[112] A. K. TARUN, V. S. CHUNDAWAT, M. MANDAL, AND M. KANKANHALLI, *Fast yet effective machine unlearning*, IEEE Transactions on Neural Networks and Learning Systems, (2023), pp. 1–10.

[113] A. THAKKAR AND K. CHAUDHARI, *Predicting stock trend using an integrated term frequency-inverse document frequency-based feature weight matrix with neural networks*, Appl. Soft Comput., 96 (2020), p. 106684.

[114] A. THUDI, G. DEZA, V. CHANDRASEKARAN, AND N. PAPERNOT, *Unrolling SGD: understanding factors influencing machine unlearning*, in 7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022, IEEE, 2022, pp. 303–319.

[115] A. THUDI, H. JIA, I. SHUMAILOV, AND N. PAPERNOT, *On the necessity of auditable algorithmic definitions for machine unlearning*, in 31st USENIX Security, Boston, USA, August 10-12, 2022, pp. 4007–4022.

[116] M. VEALE, R. BINNS, AND L. EDWARDS, *Algorithms that remember: Model inversion attacks and data protection law*, CoRR, abs/1807.04644 (2018).

[117] U. VON LUXBURG, *A tutorial on spectral clustering*, Stat. Comput., 17 (2007), pp. 395–416.

[118] D. WANG, M. GABOARDI, A. D. SMITH, AND J. XU, *Empirical risk minimization in the non-interactive local model of differential privacy*, J. Mach. Learn. Res., 21 (2020), pp. 200:1–200:39.

[119] H. WANG, Z. WANG, M. DU, F. YANG, Z. ZHANG, S. DING, P. MARDZIEL, AND X. HU, *Score-cam: Score-weighted visual explanations for convolutional neural networks*, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020, Computer Vision Foundation / IEEE, 2020, pp. 111–119.

[120] J. WANG, S. GUO, X. XIE, AND H. QI, *Federated unlearning via class-discriminative pruning*, in WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022, F. Laforest, R. Troncy, E. Simperl, D. Agarwal, A. Gionis, I. Herman, and L. Médini, eds., ACM, 2022, pp. 622–632.

[121] S. WANG, T. TUOR, T. SALONIDIS, K. K. LEUNG, C. MAKAYA, T. HE, AND K. CHAN, *Adaptive federated learning in resource constrained edge computing systems*, IEEE J. Sel. Areas Commun., 37 (2019), pp. 1205–1221.

[122] T. WANG, J. ZHAO, M. YATSKAR, K. CHANG, AND V. ORDONEZ, *Balanced datasets are not enough: Estimating and mitigating gender bias in deep image representations*, in 2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019, IEEE, 2019, pp. 5309–5318.

[123] A. WARNECKE, L. PIRCH, C. WRESSNEGGER, AND K. RIECK, *Machine unlearning of features and labels*, in 30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023, 2023.

[124] J. WENG, S. YAO, Y. DU, J. HUANG, J. WENG, AND C. WANG, *Proof of unlearning: Definitions and instantiation*, IEEE Trans. Inf. Forensics Secur., 19 (2024), pp. 3309–3323.

[125] G. WU, M. HASHEMI, AND C. SRINIVASA, *PUMA: performance unchanged model augmentation for training data removal*, in Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022, AAAI Press, 2022, pp. 8675–8682.

[126] Y. WU, E. DOBRIBAN, AND S. B. DAVIDSON, *Deltagrad: Rapid retraining of machine learning models*, in Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, vol. 119 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 10355–10366.

[127] Z. WU, S. PAN, F. CHEN, G. LONG, C. ZHANG, AND P. S. YU, *A comprehensive survey on graph neural networks*, IEEE Trans. Neural Networks Learn. Syst., 32 (2021), pp. 4–24.

[128] S. XIA, D. PENG, D. MENG, E. GIEM, W. WEI, AND Z. CHEN, *Ball kk-means: Fast adaptive clustering with no bounds*, IEEE Trans. Pattern Anal. Mach. Intell., 44 (2022), pp. 87–99.

[129] H. XU, T. ZHU, L. ZHANG, AND W. ZHOU, *Really unlearned? verifying machine unlearning via influential sample pairs*, arXiv preprint arXiv:2406.10953, (2024).

[130] H. XU, T. ZHU, L. ZHANG, W. ZHOU, AND P. S. YU, *Machine unlearning: A survey*, ACM Comput. Surv., 56 (2024), pp. 9:1–9:36.

[131] ——, *Update selective parameters: Federated machine unlearning based on model explanation*, CoRR, abs/2406.12516 (2024).

[132] H. YAN, X. LI, Z. GUO, H. LI, F. LI, AND X. LIN, *ARCANE: an efficient architecture for exact machine unlearning*, in Proceedings of the Thirty-First International

Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022, L. D. Raedt, ed., ijcai.org, 2022, pp. 4006–4013.

[133] S. YEOM, I. GIACOMELLI, M. FREDRIKSON, AND S. JHA, *Privacy risk in machine learning: Analyzing the connection to overfitting*, in 31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018, IEEE Computer Society, 2018, pp. 268–282.

[134] L. ZELNIK-MANOR AND P. PERONA, *Self-tuning spectral clustering*, in Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada], 2004, pp. 1601–1608.

[135] H. ZHANG, Y. LI, Y. HUANG, Y. WEN, J. YIN, AND K. GUAN, *Mlmodelci: An automatic cloud platform for efficient mlaas*, in MM '20: The 28th ACM International Conference on Multimedia, Virtual Event / Seattle, WA, USA, October 12-16, 2020, C. W. Chen, R. Cucchiara, X. Hua, G. Qi, E. Ricci, Z. Zhang, and R. Zimmermann, eds., ACM, 2020, pp. 4453–4456.

[136] M. ZHANG, Z. REN, Z. WANG, P. REN, Z. CHEN, P. HU, AND Y. ZHANG, *Membership inference attacks against recommender systems*, in CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021, Y. Kim, J. Kim, G. Vigna, and E. Shi, eds., ACM, 2021, pp. 864–879.

[137] P. ZHANG, G. BAI, Z. HUANG, AND X. XU, *Machine unlearning for image retrieval: A generative scrubbing approach*, in MM '22: The 30th ACM International Conference on Multimedia, Lisboa, Portugal, October 10 - 14, 2022, J. Magalhães, A. D. Bimbo, S. Satoh, N. Sebe, X. Alameda-Pineda, Q. Jin, V. Oria, and L. Toni, eds., ACM, 2022, pp. 237–245.

[138] Q. ZHANG, X. WANG, R. CAO, Y. N. WU, F. SHI, AND S. ZHU, *Extraction of an explanatory graph to interpret a CNN*, IEEE Trans. Pattern Anal. Mach. Intell., 43 (2021), pp. 3863–3877.

[139] Q. ZHANG, X. WANG, Y. N. WU, H. ZHOU, AND S. ZHU, *Interpretable cnns for object classification*, IEEE Trans. Pattern Anal. Mach. Intell., 43 (2021), pp. 3416–3431.

[140] B. ZHOU, A. KHOSLA, À. LAPEDRIZA, A. OLIVA, AND A. TORRALBA, *Learning deep features for discriminative localization*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, IEEE Computer Society, 2016, pp. 2921–2929.

[141] B. ZHOU, Y. SUN, D. BAU, AND A. TORRALBA, *Revisiting the importance of individual units in cnns via ablation*, CoRR, abs/1806.02891 (2018).

[142] T. ZHU, D. YE, W. WANG, W. ZHOU, AND P. S. YU, *More than privacy: Applying differential privacy in key areas of artificial intelligence*, IEEE Trans. Knowl. Data Eng., 34 (2022), pp. 2824–2843.