Prototype Surgery: Tailoring Neural Prototypes via Soft Labels for Efficient Machine Unlearning

Gaoyang Liu
Hubei Key Laboratory of Internet of Intelligence, School of EIC, Huazhong University of Science and Technology
Wuhan, China
liugaoyang@hust.edu.cn

Xijie Wang
Hubei Key Laboratory of Internet of Intelligence, School of EIC, Huazhong University of Science and Technology
Wuhan, China
xjwang@hust.edu.cn

Zixiong Wang
Hubei Key Laboratory of Internet of Intelligence, School of EIC, Huazhong University of Science and Technology
Wuhan, China
zixwang@hust.edu.cn

Chen Wang*
Hubei Key Laboratory of Internet of Intelligence, School of EIC, Huazhong University of Science and Technology
Wuhan, China
chenwang@hust.edu.cn

Ahmed M. Abdelmoniem
School of Electronic Engineering and Computer Science, Queen Mary University of London
London, UK
ahmed.sayed@qmul.ac.uk

Desheng Wang
Hubei Key Laboratory of Internet of Intelligence, School of EIC, Huazhong University of Science and Technology
Wuhan, China
dswang@hust.edu.cn

Abstract

The rapid advancements and widespread application of deep neural networks (DNNs), coupled with their reliance on sensitive and private data, have sparked growing concerns regarding data privacy and the "right to be forgotten". In response, machine unlearning has been proposed to efficiently eliminate the influence of specific data from trained DNNs. However, existing machine unlearning methods often face challenges due to the large number of parameters in trained models, which lead to slow execution and high memory consumption, making them impractical for large-scale applications. To address this issue, we shift our focus to "prototypes", which are defined as the weights of the final classification layer. Our key observation is that prototype would shift before and after unlearning a set of training samples. Building on this observation, we propose two prototype-based unlearning methods to tackle these challenges efficiently. First, we introduce Naive Prototype Surgery (Naive PS), which provides a fast and simplified prototype update process using a closed-form solution to approximate unlearning effects. Second, we propose Prototype Surgery (PS), which further reduces memory consumption by leveraging soft label information to fine-tune prototypes with greater precision. Both approaches directly modify the prototypes of the model by removing the influence of unlearning samples, thereby altering the model's predictions and ultimately achieving machine unlearning. Extensive experiments on four datasets demonstrate that our methods significantly accelerate the unlearning process while achieving comparable results to five existing methods in terms of both unlearning performance and privacy guarantee.

CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Security and privacy** → **Information accountability and usage control**; **Usability in security and privacy**.

Keywords

Machine Unlearning, Deep Neural Networks, Soft Label, Class Prototype

ACM Reference Format:
Gaoyang Liu, Xijie Wang, Zixiong Wang, Chen Wang, Ahmed M. Abdelmoniem, and Desheng Wang. 2025. Prototype Surgery: Tailoring Neural Prototypes via Soft Labels for Efficient Machine Unlearning. In *Proceedings of ACM Conference on Computer and Communications Security (ACM CCS'25).* ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The rapid advancements and widespread application of machine learning models, particularly Deep Neural Networks (DNNs), have achieved near-human performance on various tasks, driven by large-scale training datasets and sophisticated architectures [19]. However, the reliance on collecting large-scale customer data, including sensitive and private information, for training these models has raised significant privacy concerns [11, 40]. Such trained models may inadvertently memorize and expose sensitive information from the training data, resulting in risks of privacy leakage [28, 45]. In response, privacy regulations such as the European Union's General Data Protection Regulation (GDPR) [39] have been enacted, granting individuals the "right to be forgotten". However, legislative measures are not adequate; robust technical solutions are required to implement this right within DNNs. While retraining models from scratch to exclude unlearning data is a theoretically effective approach, it is computationally prohibitive. This challenge has catalyzed growing interest in machine unlearning techniques, which

*Corresponding author.

ACM CCS'25, Taipei, Taiwan

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

aim to efficiently remove the influence of specific data from trained models [33, 44].

Building on the growing interest in machine unlearning techniques, we focus on sample-level unlearning, which specifically aims to remove the influence of individual data samples from a trained model. Current unlearning methods in DNNs primarily adjust model parameters to mitigate the impact of unlearned samples on the model's outputs. Among these, some approaches identify critical parameters for unlearning through matrix computations, such as leveraging the Fisher information matrix [13, 15] or applying optimization techniques to improve computational efficiency [14]. Other methods employ fine-tuning strategies to refine model parameters, while selective fine-tuning combines parameter selection with targeted updates to enhance unlearning efficiency [4, 5]. However, these methods often involve modifying a large number of parameters, resulting in longer unlearning times and substantial memory consumption, thereby limiting their scalability to large-scale models and datasets (c.f. Table 1).

Although such parameter-centric approaches address the unlearning problem, they do so at the cost of computational and memory efficiency due to the extensive adjustments required across the entire model. Importantly, the ultimate goal of unlearning is to ensure that the model's outputs no longer reflect the influence of specific data, which is directly governed by the final classification layer. This insight naturally directs our attention to the prototypes, which represent the weights of this final layer and serve as an abstraction of class-level knowledge. By focusing on prototypes, we hypothesize that it is possible to achieve unlearning with minimal adjustments since prototype shifts encapsulate the effects of unlearning while avoiding the need to propagate changes throughout the network.

To explore this hypothesis, we first observe the behavior of prototypes in DNNs by comparing the prototypes of multiple original models and retrained models, which are normally considered as ideally unlearned models. For clarity, we refer the class containing unlearning data as the target class. As visualized in Figure 1, two critical insights emerge from our analysis. Firstly, the prototypes for the target class undergo significant shifts in the retrained models compared to their counterparts in the original models. These large shifts ensure the effective removal of specific information related to the target class, aligning with the successful unlearning of target data. Secondly, the prototypes for non-target classes remain highly similar between the original and retrained models, maintaining relatively stable positions. Despite this similarity, the prototypes for non-target classes still experience minor perturbations, which correspond to the slight but controlled effects of unlearning on the model's performance for the data not in unlearning set.

In light of the above ideas, we propose two novel prototype-based unlearning methods, Naive Prototype Surgery (Naive PS) and Prototype Surgery (PS), to achieve effective machine unlearning by directly operating on the prototypes. Naive PS employs a closed-form formula to efficiently approximate the update process, selectively removing the influence of the unlearning data. PS introduces a more refined adjustment mechanism, leveraging informative soft-labeled unlearning data to guide prototype updates in a precise and robust manner.

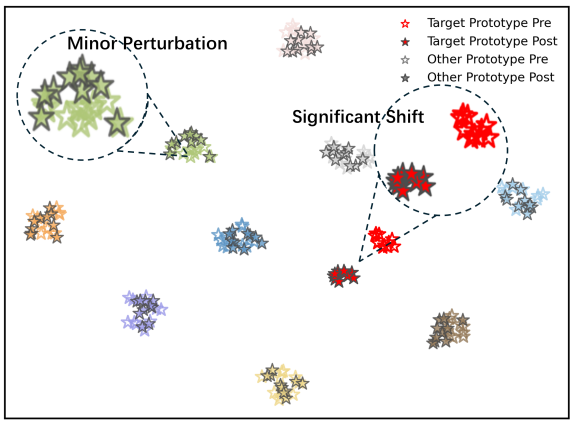


Figure 1: The t-SNE visualization of prototype shifts reveals two key observations. Prototypes before unlearning (pre) and after unlearning (post) are color-coded by class, with the target class highlighted in vibrant red. First, the prototype of the target class undergoes a significant shift from pre to post. Second, prototypes for non-target classes remain largely consistent, exhibiting only minimal perturbations.

Despite the simplicity and elegance of directly operating on prototypes, those approaches encounter two significant challenges. The first challenge lies in the inherent complexity of handling high-dimensional prototype spaces. Unlike low-dimensional visualized spaces where prototype updates are intuitive, updates in high-dimensional spaces are far more complex due to the large number of dimensions and interdependencies. To overcome this, we conduct an in-depth analysis of the prototype update process, simplifying and modeling it mathematically. This modeling allows us to directly handle the high-dimensional prototype updates by approximating the changes required for unlearning through a closed-form formula, as implemented in Naive PS. This solution enables efficient and targeted removal of the influence of unlearning data.

The second challenge arises from the observation that while unlearning primarily affects the target prototype, it also induces subtle but measurable shifts in the non-target prototypes. These minor shifts, if not carefully addressed, can lead to discrepancies between the unlearned model and the retrained model. To address this, PS leverages the rich information embedded in soft labels to guide the adjustments of non-target prototypes, ensuring that the unlearned model aligns more closely with the retrained model. By effectively managing the influence of unlearning data on both target and non-target prototypes, PS achieves precise and robust unlearning.

We summarize our contributions as follows:

- We introduce a novel prototype-centric perspective for machine unlearning in DNNs, offering a distinct shift from the traditional parameter-centric approaches that focus on identifying and modifying large-scale model parameters. By directly operating on prototypes, our approach inherently provides a fresh viewpoint with potential advantage in efficiency. To the best of our knowledge, this is the first work

Table 1: Comparison of Unlearning Methods

Comparison Method	Ratio of Model Parameters	Remaining Training Data	Extra Data	Addition Requirements
SISA [1]	100%	Yes	No	Pre-Storage of Sub-Model Parameters
Parameter Selection [13, 15]	30~90%	√	×	×
Relabel Unlearning [7, 20]	100%	√	×	×
Adversarial Tuning [4, 5]	2~90%	×	√	×
Naive PS (Ours)	0.2~1%	×	×	Embeddings of Training Data
PS (Ours)	0.2~1%	×	×	×

Required: √ Non-required: ×

to explore machine unlearning through a prototype-centric lens.

- We propose Naive PS, a method that efficiently approximates the influence of unlearning data through a closed-form solution. By selectively removing the contributions of unlearning samples from prototypes, Naive PS achieves fast unlearning with minimal computational complexity.
- We propose PS, a method that refines prototypes by leveraging soft-labeled unlearning data. This approach ensures precise adjustments to all prototypes while maintaining memory efficiency. Beyond addressing the primary goal of updating target prototypes, PS also carefully considers the impact on non-target prototypes, effectively capturing subtle shifts across the entire prototype space. By incorporating these secondary factors, PS aligns the unlearned model closely with the retrained model, achieving robust and consistent unlearning.
- We conduct extensive experiments on four popular datasets and four DNN architectures to validate the effectiveness and efficiency of our methods. The results demonstrate that Naive PS and PS achieve comparable unlearning performance to five existing methods, while significantly reducing computational and memory requirements. The code of **PS** is available at: https://anonymous.4open.science/r/PS-Unlearning-3A48.

2 Related Works

In this section, we first provide an overview of unlearning techniques in machine learning models, then focus specifically on DNNs. Existing work in DNNs is categorized into two main approaches: exact unlearning and approximate unlearning.

2.1 Unlearning in Machine Learning Models

Unlearning techniques have been explored in traditional machine learning models to efficiently remove specific data points while minimizing computational overhead. Cao et al. [3] adapt the support vector machine algorithm into a summation form, enabling efficient unlearning by adjusting specific summations. Ginart et al. [12] propose a hierarchical aggregation method for k-means clustering, updating only relevant subsets to avoid full retraining. Similarly, Nguyen et al. [31] leverage variational inference to enable efficient unlearning in Bayesian models by minimizing Kullback-Leibler divergence.

In ensemble models, Brophy et al. [2] introduce the DaRE forest, which updates subtrees with randomness and caching for computational efficiency. Other techniques, such as the Markov chain Monte Carlo-based algorithm by Nguyen et al. [32], approximate retrained models to reduce unlearning overhead. While effective, these methods are model-specific and lack generalizability to DNNs due to their reliance on simpler structures and smaller parameter spaces.

2.2 Unlearning in DNNs

In DNNs, the naive approach of retraining models from scratch to exclude unlearning data is theoretically effective but computationally prohibitive, especially for large-scale applications. To address this, existing research can be categorized into two main types: exact unlearning and approximate unlearning.

2.2.1 Exact Unlearning. Exact unlearning methods aim to accelerate retraining by modifying the training pipeline. Bourtoule et al. [1] propose the SISA framework, which partitions training data into chunks and trains submodels independently, allowing selective retraining. Yan et al. [42] further optimize data partitioning in SISA to improve unlearning efficiency and reduce data dependency issues. For graph neural networks, Chen et al. [6] design GraphEraser, a shard-based partitioning approach to efficiently unlearn specific data. While these methods reduce retraining costs, they fundamentally alter the original training pipeline, which can compromise overall model performance and introduce additional complexity.

Different from SISA framework, DeltaGrad [41] accelerates retraining by caching gradient information, allowing faster updates during unlearning. Similarly, Liu et al. [27] propose a federated unlearning method that combines local data removal with first-order Taylor approximations.

2.2.2 Approximate Unlearning. Approximate unlearning methods aim to remove the influence of unlearning data without full retraining by directly modifying model parameters. For instance, Golatkar et al. [13] utilize the Fisher Information Matrix (FIM) to identify and neutralize parameters heavily influenced by the unlearning data. Similarly, Guo et al. [15] employ influence functions to approximate the impact of specific data points and adjust model parameters accordingly.

To further reduce computational complexity, Suriyakumar et al. [38] propose an online unlearning algorithm that avoids repeated Hessian matrix calculations. Mehta et al. [29] identify a

Markov Blanket of parameters for efficient unlearning, limiting the parameter updates to a small subset. In federated learning contexts, FedEraser [24] and SIFU [10] leverage stored historical updates to unlearn client-specific data efficiently.

Fine-tuning-based methods refine the approximate unlearning process by focusing on specific aspects of the model. Boundary Unlearning [5] modifies decision boundaries rather than the broader parameter space to remove the influence of unlearning data, while adversarial methods [4] use adversarial examples and importance weighting to selectively erase specific instance information. Chundawat et al. [7] propose a teacher-student framework for selective knowledge transfer, introducing the Zero Retrain Forgetting (ZRF) metric to evaluate unlearning performance.

Despite the diversity of methods introduced above, they all share a common underlying thought process: removing the influence of unlearning data by modifying a large number of model parameters. While this parameter-centric perspective has driven significant progress, it also comes with several inherent limitations, particularly in terms of computational efficiency and resource consumption.

First, parameter-centric methods often involve significant computational costs. Even when only partial middle parameters are modified, the inherent update mechanism of DNNs still requires computing gradient information for most of the model, resulting in high memory and computational overhead. This undermines the goal of reducing resource consumption, especially for large-scale models and datasets. Second, these methods often depend on access to the remaining datasets or stored intermediate states, which can be impractical in real-world scenarios due to privacy constraints or data unavailability.

Machine unlearning, by its nature, aims to ensure that the model's outputs are no longer influenced by the unlearning data. This requirement becomes even more critical in Machine Learning as a Service (MLaaS) settings, where the focus is on efficiently updating deployed models to produce correct outputs without incurring significant computational or storage costs. This motivates a shift toward prototype-centric approaches, which operate at a more abstract level and offer the potential to achieve unlearning with significantly reduced resource consumption while meeting the core objectives of unlearning.

3 Preliminaries and Notation

In this section, we establish the notation and foundational concepts related to classification-based machine unlearning. Let us denote the original training set as $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in \mathcal{X}, \mathbf{y}_i \in \mathcal{Y}, 1 \leq i \leq N\}$, where $\mathbf{x}_i \in \mathcal{X}$ represents the input data (e.g., an image), and $\mathbf{y}_i \in \mathcal{Y}$ denotes the corresponding label, with $\mathcal{Y} = \{1, \ldots, C\}$ being the label space, C is the total number of classes and N is the total number of data samples in the dataset. Then, we define the unlearning set $\mathcal{D}_u \subseteq \mathcal{D}$ as a subset of the original training set, which consists of the data points that need to be unlearned. The remaining set is denoted as $\mathcal{D}_r = \mathcal{D} \setminus \mathcal{D}_u$, and $\mathcal{D}_t$ represents the original test set. In this work, we focus on sample-level unlearning, where the unlearning set $\mathcal{D}_u$ is constructed by randomly selecting samples from a particular class.

Given the critical role of prototypes in our approach, we formally define prototypes within DNN classification models. Typically, a DNN classification model $\mathcal{M}$ can be divided into two components: a feature extractor $\mathcal{F}$ and a final classification layer. The feature extractor $\mathcal{F}$ maps an input $\mathbf{x} \in \mathbb{R}^d$ to an embedding vector $\mathbf{e} \in \mathbb{R}^{1\times E}$ as follows:

$$\mathbf{e} = \mathcal{F}(\mathbf{x}), \tag{1}$$

where E denotes the embedding dimensionality.

The final classification layer computes the logits using the following formula:

$$\mathbf{z} = \mathbf{e}\mathbf{W}^T + \mathbf{b}, \tag{2}$$

where $\mathbf{z} \in \mathbb{R}^{1\times C}$ represents the logits for all classes, and $\mathbf{b} \in \mathbb{R}^{1\times C}$ is the bias term. Here, we define the weight matrix $\mathbf{W} \in \mathbb{R}^{C\times E}$ as the prototype matrix, where each row $\mathbf{w}_c \in \mathbb{R}^{1\times E}$ corresponds to the prototype for the c-th class, and C is the total number of classes.

The logits $\mathbf{z}$ are then passed through a softmax function to compute the probability vector $\mathbf{p} \in \mathbb{R}^{1\times C}$, which represents the likelihoods across all classes:

$$p_c = \frac{\exp(z_c)}{\sum_{k=1}^{C} \exp(z_k)}, \quad \forall c \in \{1, 2, \ldots, C\}. \tag{3}$$

However, since the embedding vector $\mathbf{e}$ is typically normalized by a normalization layer, and the optimization usually does not change the magnitude of the prototype, the dot product $\mathbf{e}\mathbf{w}_c^T$ effectively captures the alignment between the embedding vector $\mathbf{e}$ and the prototype $\mathbf{w}_c$. This alignment can be seen as analogous to the cosine similarity between the two vectors due to their stable magnitudes. Through the softmax operation, the logits are transformed into normalized probabilities, allowing higher alignment between $\mathbf{e}$ and a specific prototype $\mathbf{w}_c$ to correspond to a higher probability for the associated class.

Consider the original DNN model $\mathcal{M}$, parameterized by $\theta \in \mathbb{R}^P$, where P is the total number of model parameters, with prototypes denoted as $\mathbf{W}^{\text{ori}}$. The formal goal of unlearning is to adjust the parameters θ such that the modified model approximates the performance of the retrained model with parameters θ^*. Existing unlearning methods involve either modifying the entire parameter set θ or adjusting a subset of important parameters within θ. However, both approaches are often inefficient and computationally expensive. Instead, given the critical role of $\mathbf{W}^{\text{ori}}$ in determining the model's outputs, effective unlearning can be achieved by focusing solely on adjusting the prototypes. Specifically, this involves modifying $\mathbf{W}^{\text{ori}}$ to a new prototype matrix $\mathbf{W}^{\text{un}}$, while keeping the remaining parameters unchanged. This ensures the model's outputs closely match those of the retrained model, achieving the unlearning objective with lower computational cost.

4 Naive Prototype Surgery

4.1 Prototypes in Classification Models

The fundamental idea behind all classification models is rooted in the concept of a one-layer linear classification model. To build a foundation for understanding more complex architectures, we begin with this simple yet powerful model.

In this context, the input $\mathbf{x}$ represents an embedding that serves as the feature representation of the data, which is mapped to C

classes using a set of prototype vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_C$. The linear model can be expressed as:

$$\mathbf{y} = \mathbf{x}\left(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_C\right) + \mathbf{b}, \tag{4}$$

where $\mathbf{y} = [y_1, y_2, \ldots, y_C]$ represents the logits for each class. Intuitively, each logit y_c is computed as the similarity between the embedding $\mathbf{x}$ and the prototype $\mathbf{w}_c$ of class c, making prototypes central to the classification process. These logits are then converted into class probabilities using the softmax function, as defined in Eq. (3).

The training process ensures that each prototype $\mathbf{w}_c$ evolves into a representative embedding for its corresponding class by interacting with individual samples. For the correct class, the prototype is pulled closer to the embeddings of its samples, strengthening alignment. Conversely, prototypes for other classes are pushed farther away, enhancing separation. This mechanism allows prototypes to capture discriminative features that effectively differentiate between classes. It is crucial to understand how this process functions, especially when data distributions shift, to ensure that prototypes adapt appropriately and continue to accurately represent their respective classes.

4.2 Prototypes in DNNs

Now, turning back to the process of optimizing the prototypes for the classification layer in DNNs. We focus exclusively on optimizing the final classification layer, allowing us to treat the embedding vector $\hat{\mathbf{x}}$ as a fixed value extracted by the feature extractor. To quantify the prediction error when inferring the label $\mathbf{y}$ from an input $\mathbf{x}$, a commonly used approach is to minimize the Cross-Entropy (CE) loss function, which is widely regarded as an effective criterion for classification tasks. The CE loss function $\mathcal{L}_{\text{CE}}$ is expressed as:

$$\mathcal{L}_{\text{CE}}(M(\mathbf{x}), \mathbf{y}) = -\sum_{k=1}^{C} y_k \log(p_k), \tag{5}$$

where p_k is the predicted probability for class k, and y_k is the one-hot encoded ground truth label.

Given the non-convex nature of DNNs, solving the optimization problem in a closed form is not feasible. Instead, iterative optimization methods are employed to minimize the loss function. One of the most commonly used optimization strategies is mini-batch stochastic gradient descent (SGD), which updates the parameters iteratively. The update rule for optimizing the prototypes $\mathbf{W}$ through mini-batch SGD is given by:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \left.\frac{\partial L}{\partial \mathbf{W}}\right|_{\mathbf{W}_t, \mathbf{x}_t}, \tag{6}$$

where $\mathbf{W}_t$ represents the prototypes at iteration t, $\mathbf{x}_t$ is the selected sample at step t, and η is the learning rate.

By applying the chain rule and explicitly computing the gradient with respect to the prototypes, the gradient at step t can be expressed as:

$$\left.\frac{\partial L}{\partial \mathbf{W}}\right|_{\mathbf{W}_t, \mathbf{x}_t} = (\mathbf{p} - \mathbf{y})\hat{\mathbf{x}}_t, \tag{7}$$

where $\mathbf{p}$ is the predicted probability vector obtained through the softmax function, and $\mathbf{y}$ is the one-hot encoded ground truth label. Substituting this gradient into the SGD update rule yields the following iterative formula for updating $\mathbf{W}$:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta(\mathbf{p} - \mathbf{y})\hat{\mathbf{x}}_t. \tag{8}$$

Expanding this iterative process over multiple steps, the prototypes $\mathbf{W}_t$ at iteration t are updated according to the following equations:

$$\mathbf{W}_t = \mathbf{W}_0 - \Omega\hat{\mathbf{x}}, \tag{9}$$

$$\Omega = \eta \sum_{j=0}^{t-1} (\mathbf{p}_j - \mathbf{y}_j). \tag{10}$$

Here, $\mathbf{W}_0$ represents the initial prototypes, $\hat{\mathbf{x}}$ denotes the embeddings, and Ω is the cumulative error term, which encapsulates the weight adjustments across all iterations. Specifically, Ω accumulates the gradient contributions from the predicted errors $(\mathbf{p}_j - \mathbf{y}_j)$ over all past steps. These equations explicitly link the embeddings $\hat{\mathbf{x}}$ and the prediction errors to the optimization of the prototypes $\mathbf{W}_t$.

Although DNNs are significantly more complex than one-layer classification models, the core role of prototypes remains fundamentally consistent across both paradigms. While DNNs involve deeper architectures and more intricate optimization processes, the underlying principle persists: prototypes are adjusted to align with positive samples and to distance themselves from negative ones. This shared mechanism highlights the essential role of prototypes in capturing the relationship between data and predictions, regardless of the complexity of the model.

Beyond the mathematical formulation, the cumulative error term Ω provides key insights into the optimization process. It aggregates the influence of all samples seen during optimization, illustrating how each data embedding directly impacts prototype updates. This mechanism ensures that prototypes are iteratively refined to better reflect the overall data distribution. By bridging raw embeddings and classification decisions, prototypes serve as interpretable and adaptable representations that effectively capture the relationship between data and predictions.

4.3 Implementation of Naive PS

Through our observation, we identify the phenomenon of prototype shift before and after unlearning specific data. This shift arises directly from the removal of unlearning data, disrupting the cumulative contributions to the prototype. Since prototypes are constructed as weighted aggregations of data embeddings, this shift can be intuitively corrected by isolating and explicitly removing the influence of the unlearned data.

To implement this unlearning procedure, we first adjust the embedding matrix $\hat{\mathbf{x}}$ by removing the rows corresponding to the unlearned samples. The remaining rows are denoted as $\hat{\mathbf{x}}_r$, representing the embeddings of the retained data. Next, we address the cumulative error term Ω. Though Ω is not directly observable it can be retrospectively computed by leveraging the relationship outlined in the update Eq. (9). Once Ω is determined, we remove the columns corresponding to the unlearned samples, resulting in the adjusted term Ω_r.

The unlearned prototypes are then computed using the following equation:

$$\mathbf{W}_t^{\text{un}} = \mathbf{W}_0 - \Omega_r\hat{\mathbf{x}}_r, \tag{11}$$

where $\mathbf{W}_t^{\text{un}}$ denotes the prototypes after unlearning.

By modeling this shift through the Naive PS, we can effectively capture the observed changes and enable the model to efficiently reach its unlearned state. This method provides a computationally efficient means of aligning the prototypes with their expected behavior after unlearning, ensuring minimal disruption to the remaining data representation.

5 Prototype Surgery

5.1 Designing Soft Label

The Naive PS method effectively reduces storage overhead by relying on pre-saved embeddings. However, this assumption is impractical in real-world scenarios, where training embeddings are typically not deliberately stored, and retrieving the entire dataset to recompute embeddings is neither realistic nor efficient, particularly for large-scale datasets. A closer examination of prototype shift observation also reveals another limitation of the Naive PS method: unlearning specific samples not only causes significant shifts in the target prototype but also influences other prototypes to varying degrees. Despite this, Naive PS primarily adjusts only to the target prototype, neglecting the broader prototype shifts caused by unlearned samples.

To solve these limitations, we conducted experiments to analyze the behavior of unlearned samples. As shown in Figure 2, our observations highlight a key phenomenon: the logits of unlearned samples in the original model naturally align these samples with categories close to their original class. This insight provides a foundation for constructing meaningful soft labels. Soft labels, as established in previous research [30, 43], allow for finer adjustments by incorporating the model's predicted probabilities, thereby enabling prototype updates that better reflect the actual data distribution.

For a given sample to be unlearned, let $\mathbf{z}$ represent its logits vector, and $\mathbf{y}$ denote its one-hot label vector. To compute the soft labels, we first adjust the logits by setting the target class's logit to zero:

$$\mathbf{z}' = \mathbf{z} - (\mathbf{z} \circ \mathbf{y})\mathbf{y}, \tag{12}$$

where $(\mathbf{z} \circ \mathbf{y})$ isolates the logit value of the target class.

The adjusted logits are then normalized to compute the probability vector $\mathbf{p}$. The probability for class c, denoted as p_c, is computed as:

$$p_c = \frac{z'_c}{\sum_{k=1}^{C} z'_k}, \tag{13}$$

where z'_c is the adjusted logit for class c, and the denominator ensures that the probabilities sum to one. This normalization distributes probabilities according to the relative magnitude of the logits, allowing $\mathbf{p}$ to serve as a meaningful soft label that incorporates the model's biases toward other categories. Since the logits for most other classes are typically very small and contribute negligibly to the overall probabilities, they can be safely ignored. Thus, we retain only the top two logits (excluding the target class) and set the rest to zero. This reduction not only improves computational efficiency but also focuses the soft labels on the most relevant competing categories, which are most likely to influence prototype updates.

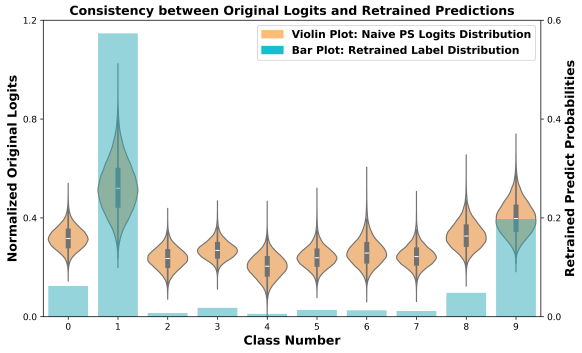


Figure 2: Consistency in Predictions between Original and Retrained Models. The left panel shows a violin plot of the normalized logits distribution for unlearning samples in the original model, and the right panel presents a bar chart of the predicted label probabilities by the retrained model. The alignment between the original logits and the retrained model's predictions highlights the valuable role of the original logits in guiding prototype adjustments, especially for non-target classes.

5.2 Refining Prototypes via Soft Label Surgery

Once the soft labels are computed, we fine-tune the model by updating only the prototype parameters $\mathbf{W}^{\text{ori}}$, freezing all other parameters in the model. The unlearned prototypes $\mathbf{W}^{\text{un}}$ are obtained by minimizing the following objective:

$$\mathbf{W}^{\text{un}} = \arg\min_{\mathbf{W}^{\text{ori}}} \sum_{\mathbf{x}_i \in \mathcal{D}_u, \mathbf{p}_i} \mathcal{L}_{\text{CE}}(\mathbf{x}_i, \mathbf{p}_i, \mathbf{W}^{\text{ori}}). \tag{14}$$

This targeted adjustment ensures that the unlearned model behaves as closely as possible to a retrained model. By leveraging only the unlearning samples and updating the prototypes $\mathbf{W}^{\text{ori}}$, our PS method achieves efficient unlearning without requiring access to additional fine-tuning data. This design significantly reduces computational cost compared to other unlearning methods, since we only need to compute gradient information for a single layer to perform the updates. Additionally, since only the prototypes are updated while the rest of the model remains frozen, the impact on non-unlearned data is minimal, preserving the overall performance of the model. By incorporating the nuanced information encoded in the soft labels, our approach refines the prototypes to better align the classification boundaries with those expected from a fully retrained model. This ensures that the influence of unlearned data is effectively minimized while maintaining the model's robustness and efficiency.

6 Performance Evaluation

6.1 Experimental Settings

We evaluate our method on popular unlearning benchmarks, with settings closely aligning with those used in previous unlearning research [4, 5, 13].

Table 2: Datasets and Corresponding Models

Dataset	Model
MNIST	LeNet5
CIFAR-10	ResNet20
CIFAR-100	ResNet50
TinyImageNet	VGG16

Datasets. We conduct experiments on four widely used benchmark datasets in machine unlearning, namely the MNIST [9], CIFAR-10, CIFAR-100 [21], and TinyImageNet [8] datasets.

Models. We demonstrate the versatility of our methods by applying different model architectures to various datasets (c.f. Table 2). Specifically, for the MNIST dataset, we use the LeNet5 model [23]. For the CIFAR-10 and CIFAR-100 datasets, we employ the ResNet20 and ResNet50 models [17]. For TinyImageNet, we utilize the VGG16 model [37].

Baselines. We implement 6 baseline unlearning methods for comparison:

- `Retrain`: we retrain the initialized model using only the remaining set. Typically, the retrained model is considered the gold standard for unlearning.
- `Fine-Tune`: we fine-tune the original model using the remaining set with a larger learning rate and an increased number of epochs. Continuing to train on the remaining set will induce catastrophic forgetting in the unlearning set.
- `Random Label` [16]: we fine-tune the original model on the modified unlearning set to remove the knowledge. Specifically, we randomly assign labels for each image. Since the unlearning set is randomly labeled, the model will be misguided and perform worse on unlearning set .
- `Negative Gradient` [13]: we fine-tune the original model on the full training set . If a sample belongs to the unlearning set, we use the negative loss value to update the parameters. This approach directly degrades the model's performance on the unlearning set.
- `Fisher Forgetting` [13]: the Fisher Forgetting method first uses gradient information to identify the most relevant parameters, and then adds noise to interfere with the performance.
- `L2UL` [4]: the Learning to Unlearn method uses an importance metric to identify the important parameters, and then leverages the unlearning set to generate adversarial samples that help the model unlearn at a representative level.

Implementations. Our methods and other baselines are implemented in Python 3.8 using the PyTorch library [34] to build all DNN models, with all experiments conducted on a workstation equipped with two NVIDIA GeForce RTX 4080 GPUs. We use the SGD optimizer with a weight decay of 10^{-4}, a momentum of 0.9, and a batch size of 256. The learning rates for MNIST, CIFAR-10, CIFAR-100, and TinyImageNet are 10^{-3}, 0.2, 0.01, and 0.02, respectively. Each experiment is repeated 10 times to ensure the robustness of our results, with the reported outcomes being the average of these repetitions. Unless otherwise mentioned, these configurations are used by default. For unlearning methods based on retraining or fine-tuning, we employ the SGD optimizer with varying learning rates ranging from 5×10^{-6} to 0.3 and epochs ranging from 5 to 60. For the Fisher and L2UL methods, we adopt the hyperparameters from the original papers.

Specifically, we repeat each method 10 times within a single program, then record the average and standard deviation of the accuracy curve to evaluate the stability of the unlearning method. An ideal unlearning method should not only be efficient but also exhibit high stability, ensuring consistent performance while effectively achieving the intended unlearning of specific data. Note that we focus on unlearning at the sample level; thus, we randomly select images from one class to construct the unlearning set. In the main experiment table, similar to other unlearning settings [22, 26], we randomly choose 10% of the data from one certain class as the unlearning set, while the remaining samples in the training data forms the remaining set.

Metrics. We evaluate the proposed unlearning methods along four key dimensions: Unlearning Performance, Privacy Guarantee, Unlearning Efficiency, and Prototype Similarity.

Specifically, for **Unlearning Performance**, we evaluate four metrics: accuracy on the unlearning set, $\mathcal{D}_u$, the remaining set, $\mathcal{D}_r$, the original test set, $\mathcal{D}_t$, and the average accuracy gap between the unlearned and retrained models. In most cases, the retrained model serves as the gold standard for all unlearning methods. Besides, we also track the accuracy curve throughout the unlearning process to ensure that rapid convergence does not come at the cost of stability. For **Unlearning Efficiency**, we measure the total time required for each unlearning method.

For **Privacy Guarantee**, we adopt the framework established in prior research, including the works of [5] and [13]. Specifically, we perform a widely used Membership Inference Attack (MIA), as introduced by [36], on the unlearned model by analyzing both predicted labels and probability outputs. The goal of MIA is to determine whether a particular data record was included in the training set, with the optimal outcome being that the unlearned model's behavior is indistinguishable from that of a retrained model [18, 25, 35]. Additionally, we define Entropy Distance (ED)[13] as a metric to evaluate the effectiveness of our unlearning methods by comparing the logits distributions of the retrained and unlearned models. Let P_r be the probability distribution derived from the logits of the retrained model and P_u be the distribution from the logits of the unlearned model. The ED is calculated using the Jensen-Shannon (JS) divergence:

$$\mathrm{ED}(P_r \parallel P_u) = \frac{1}{2}\left(\mathrm{KL}(P_r \parallel \frac{P_r + P_u}{2}) + \mathrm{KL}(P_u \parallel \frac{P_r + P_u}{2})\right), \quad (15)$$

where the Kullback-Leibler divergence (KL) is:

$$\mathrm{KL}(P \parallel Q) = \sum P(x) \log \frac{P(x)}{Q(x)}. \quad (16)$$

A lower ED indicates that the output distribution of the unlearned model is more similar to that of the retrained model, suggesting effective unlearning.

Finally, we define **Prototype Similarity Score** (PSC) as a metric to assess whether our unlearning methods are effectively aligning with our intended outcomes by observing shifts in prototype vectors. For a given class, let $\mathbf{v}_r$ be the prototype vector from the retrained model and $\mathbf{v}_u$ be the corresponding prototype vector from

Table 3: Average accuracy (%) of various unlearning methods. The performance gap relative to the Retrain method is indicated in parentheses. Both Original and Retrain methods are not directly comparable in terms of performance gap, hence denoted by a "-" symbol. Experiments are conducted across four datasets. The best results among the unlearning methods are highlighted in bold. The Avg. Gap column represents the average performance gap compared to the Retrain method for each dataset.

Dataset	Metric	Original	Retrain	Fine-Tune	Random Label	Negative Gradient	Fisher	L2UL	Naive PS (Ours)	PS (Ours)
MNIST	Acc on $\mathcal{D}_u$	99.06	91.29	96.30 (5.01)	96.35 (5.06)	0 (91.29)	98.90 (7.61)	88.80 (2.49)	97.68 (6.39)	**91.19 (0.10)**
	Acc on $\mathcal{D}_r$	98.10	97.43	99.29 (1.86)	98.97 (1.54)	95.64 (1.79)	98.13 (0.70)	93.94 (3.49)	98.15 (0.72)	**97.96 (0.53)**
	Acc on $\mathcal{D}_t$	98.20	97.15	98.42 (1.27)	98.45 (1.30)	85.83 (11.32)	98.17 (1.02)	93.73 (3.42)	98.08 (0.93)	**97.48 (0.33)**
	Avg. Gap	-	-	2.71	2.63	34.80	3.11	3.13	2.68	**0.32**
CIFAR-10	Acc on $\mathcal{D}_u$	88.27	63.06	70.16 (7.10)	63.64 (0.58)	0 (63.06)	87.59 (24.53)	27.58 (35.48)	70.39 (7.33)	**63.43 (0.37)**
	Acc on $\mathcal{D}_r$	83.35	88.74	94.49 (5.75)	93.04 (4.30)	82.66 (6.08)	78.51 (10.23)	75.66 (13.08)	78.17 (10.57)	**84.52 (4.22)**
	Acc on $\mathcal{D}_t$	80.60	72.44	78.63 (6.19)	82.58 (10.14)	69.97 (2.47)	76.19 (3.75)	69.81 (2.63)	**74.43 (1.99)**	76.87 (4.43)
	Avg. Gap	-	-	6.35	5.01	23.87	12.84	17.06	6.63	**3.01**
CIFAR-100	Acc on $\mathcal{D}_u$	75.92	4.87	19.15 (14.28)	**4.12 (0.75)**	0 (4.87)	74.49 (69.62)	5.67 (0.80)	53.28 (48.41)	3.67 (1.20)
	Acc on $\mathcal{D}_r$	83.26	85.18	**83.93 (1.25)**	74.27 (10.91)	71.25 (13.93)	76.47 (9.71)	83.26 (1.92)	81.74 (3.44)	82.73 (2.45)
	Acc on $\mathcal{D}_t$	56.98	45.60	49.17 (3.57)	46.85 (1.25)	**43.85 (1.75)**	55.33 (9.73)	48.26 (2.66)	54.25 (8.65)	47.48 (1.88)
	Avg. Gap	-	-	6.37	4.30	6.85	29.69	**1.79**	20.17	1.84
TinyImageNet	Acc on $\mathcal{D}_u$	100	21.06	12.81 (8.25)	30.43 (9.37)	2.55 (18.51)	100 (78.93)	21.78 (0.71)	98.70 (77.63)	**22.27 (1.21)**
	Acc on $\mathcal{D}_r$	98.93	99.26	100 (0.74)	98.30 (0.96)	99.83 (0.57)	97.98 (1.28)	93.55 (5.71)	**98.87 (0.39)**	99.70 (0.44)
	Acc on $\mathcal{D}_t$	54.01	49.40	44.57 (4.83)	44.92 (4.48)	51.29 (1.89)	53.57 (4.17)	46.64 (2.76)	52.78 (3.38)	**48.80 (1.4)**
	Avg. Gap	-	-	4.61	4.94	6.99	28.13	3.06	27.13	**1.02**

an unlearned model. Formally, the PSC is calculated using cosine similarity, defined as:

$$\mathrm{PSC}(\mathbf{v}_r, \mathbf{v}_u) = \frac{\mathbf{v}_r \cdot \mathbf{v}_u}{|\mathbf{v}_r||\mathbf{v}_u|}, \tag{17}$$

where $\cdot$ denotes the dot product, and $|\mathbf{v}|$ represents the Euclidean norm of vector $\mathbf{v}$. A high PSC indicates that the prototype vectors for the corresponding class are similar, while a low PSC reflects greater dissimilarity.

6.2 Unlearning Performance

The primary objective of machine unlearning is to effectively remove the impact of the unlearning dataset $\mathcal{D}_u$ while maintaining model performance on the retained dataset $\mathcal{D}_r$ and unseen test data $\mathcal{D}_t$. To assess the efficacy of various unlearning methods, including our proposed approaches, we present a comparative analysis in Table 3. This table provides a comparative analysis of our methods alongside several existing techniques, showing that our approaches achieve competitive results in most cases.

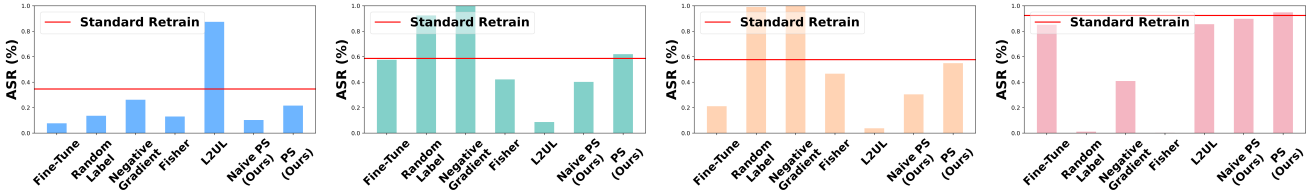
The TinyImageNet dataset presents the most challenging scenario in our experiments. Here, our PS method achieves an exceptionally low average gap of 1.02%, representing a significant improvement compared to the latest L2UL method, which records a gap of 3.73%. The L2UL method utilizes high-level features from $\mathcal{D}_u$ to guide fine-tuning directions, resulting in a small gap of 0.71% on $\mathcal{D}_u$. However, it struggles to effectively constrain the model as a whole, resulting in a significant 5.71% gap on the $\mathcal{D}_r$ and a 2.76% gap on the $\mathcal{D}_t$. By contrast, our PS method fine-tunes only a small subset of parameters with precision, focusing on prototype updates. This targeted approach minimizes disruptions to the model's structure, preserving robust performance on $\mathcal{D}_r$ (accuracy of 99.70% with a gap of 0.44%) and $\mathcal{D}_t$ (accuracy of 48.80% with a 1.4% gap). Compared to simpler methods such as Random Label, which performs well on $\mathcal{D}_u$ but struggles with a higher average gap of 4.94%. This improvement can be attributed to the additional information within soft labels. By incorporating soft label information and the inherent characteristics of the data, our method provides more precise guidance for the model to align with the retrained version, ensuring retention of overall model performance while achieving unlearning on the target data.

Next, we discuss our proposed Naive PS, which shares a strong resemblance to Fisher unlearning. Both approaches avoid additional fine-tuning by directly computing the necessary parameter adjustments for unlearning, making them computationally efficient. On simpler datasets such as MNIST, Naive PS demonstrates superior performance with a smaller average gap of 2.68% compared to Fisher's gap of 3.11%. As noted in the original Fisher paper, its method is effective for basic architectures like LeNet5 but struggles as dataset complexity increases. Naive PS, on the other hand, extends its effectiveness to CIFAR-10, consistently outperforming Fisher and demonstrating stronger robustness in handling unlearning tasks. However, the limitations of Naive PS become apparent as the complexity of the dataset increases. On more challenging datasets like CIFAR-100 and TinyImageNet, its performance declines. This can be attributed to the breakdown of the simplification assumptions in its parameter adjustment process when faced with higher-dimensional data and greater class diversity. Despite this, Naive PS still consistently outperforms Fisher across all datasets, highlighting its robustness and practicality.

Overall, our proposed methods consistently deliver competitive results across all datasets, demonstrating robust unlearning performance. Even on challenging datasets like TinyImageNet, our PS method achieves a significantly lower average gap than existing approaches, validating its effectiveness. While Naive PS shows some limitations on complex datasets, it consistently outperforms Fisher unlearning. These results highlight that our methods are compatible with five existing approaches, confirming their adaptability and reliability in various unlearning scenarios.

6.3 Privacy Guarantee

Our approaches focus on modifying only the final layer of the model, which inherently means that some information about the target unlearned data may still exist in the intermediate layers.



(a) MNIST (b) CIFAR-10 (c) CIFAR-100 (d) TinyImageNet

Figure 3: Membership inference attack results on unlearned models. The red line represents the MIA results for the retrained model. A good unlearning method should have results that are close to the red line.

Table 4: ED between the retrained model and other unlearned models, calculated based on the output distributions' entropy. A smaller ED indicates a closer similarity between the output distributions of the retrained model and the unlearned models. The best results are highlighted in bold.

Dataset	Metric	Original	Fine-tune	Random Label	Negative Gradient	Fisher	L2UL	Naive PS (Ours)	PS (Ours)
MNIST	Dist. on $\mathcal{D}_u \downarrow$	0.611	0.567	**0.049**	0.647	0.596	0.669	0.532	0.100
	Dist. on $\mathcal{D}_r \downarrow$	0.019	0.296	0.493	0.621	0.014	0.135	**0.011**	0.013
	Dist. on $\mathcal{D}_t \downarrow$	0.033	0.269	0.406	0.575	0.027	0.151	0.021	**0.013**
	Avg. $\downarrow$	0.221	0.377	0.316	0.614	0.212	0.318	0.188	**0.042**
CIFAR-10	Dist. on $\mathcal{D}_u \downarrow$	0.053	0.015	0.011	0.111	0.087	0.183	0.007	**0.005**
	Dist. on $\mathcal{D}_r \downarrow$	0.039	0.046	0.092	0.043	**0.021**	0.035	0.038	0.036
	Dist. on $\mathcal{D}_t \downarrow$	0.015	0.019	0.060	0.041	**0.006**	0.016	0.020	0.017
	Avg. $\downarrow$	0.036	0.027	0.054	0.065	0.038	0.078	0.022	**0.019**
CIFAR-100	Dist. on $\mathcal{D}_u \downarrow$	0.167	0.025	0.113	0.180	0.202	0.035	0.016	**0.010**
	Dist. on $\mathcal{D}_r \downarrow$	**0.032**	0.304	0.066	0.050	0.037	0.102	0.059	0.061
	Dist. on $\mathcal{D}_t \downarrow$	0.006	0.038	0.095	0.096	0.038	0.016	0.002	**0.001**
	Avg. $\downarrow$	0.068	0.122	0.091	0.109	0.092	0.051	0.026	**0.007**
TinyImageNet	Dist. on $\mathcal{D}_u \downarrow$	0.665	0.048	0.284	0.061	0.645	0.021	0.136	**0.034**
	Dist. on $\mathcal{D}_r \downarrow$	**0.054**	0.307	0.421	0.356	0.066	0.446	0.483	0.478
	Dist. on $\mathcal{D}_t \downarrow$	0.032	0.038	0.128	0.065	0.026	0.032	0.034	**0.025**
	Avg. $\downarrow$	0.250	**0.131**	0.278	0.161	0.246	0.166	0.218	0.179

However, this is not a critical issue in modern MLaaS scenarios, where interactions are typically limited to black-box access, and the internal states of the model remain inaccessible. In such cases, the primary requirement is to prevent any information about the target unlearned data from being leaked through the model's output—a condition that our approaches effectively satisfy.

To confirm that our proposed methods do not inadvertently leak information about the unlearned data, we plot the Attack Success Rate (ASR) for all unlearning methods across four datasets. The results are shown in Figure 3. The red line in Figure 3 represents the ASR of the retrained model, and unlearning methods should ideally align with this line. The ASR results demonstrate that our proposed PS method achieves the closest performance to the retrained model on the two more challenging datasets, CIFAR-100 and TinyImageNet. On the two simpler datasets, MNIST and CIFAR-10, although our method is not the closest to the retrained model, the difference is minimal. This suggests that our approach effectively prevents the unlearned model from leaking private information about the forgotten samples.

Specifically, for the CIFAR-10 dataset, Fine-Tune yields the best result in terms of ASR, but it generally performs less consistently across other datasets. We believe this is because the Fine-Tune method mainly relies on the catastrophic forgetting effect to achieve unlearning. However, this effect is not easily controlled and can lead to significant variation in performance across different datasets. Interestingly, the L2UL method shows poor performance on smaller datasets, which we attribute to the model's reliance on high-level features from the $\mathcal{D}_u$. Guided by high-level features may have a substantial impact on the model, allowing the unlearned model to retain detailed knowledge of the unlearned data, which leads to leakage of information during the MIA.

Next, we present the results of ED, which is the JS divergence between the entropy distribution of the retrained model and another unlearned model. A smaller ED indicates a closer match between the distributions. The results are summarized in Table 4, as you can see, they align with the MIA results discussed earlier. Our PS method performs best on the unlearning set, with the lowest ED value, such as 0.005, 0.010, and 0.034 for CIFAR-10, CIFAR-100, and TinyImageNet, respectively. However, we were surprised to find that all unlearning methods show unusually high ED values on the $\mathcal{D}_r$ of TinyImageNet. We believe this high ED value in TinyImageNet is due to its intrinsic complexity. Both the retrained model and all the unlearned models seem to "force" themselves to memorize the examples in the remaining set, rather than achieving

Table 5: Total time required for different unlearning methods (in seconds).

Dataset	Retrain	Fine-tune	Random Label	Negative Gradient	Fisher	L2UL	Naive PS (Ours)	PS (Ours)
MNIST	860.10	1138.40	8.90	8.70	160.22	17.25	0.12	1.25
CIFAR-10	1514.10	2478.00	39.10	45.10	500.61	33.50	0.12	0.98
CIFAR-100	5913.00	2993.50	127.35	140.70	1370.29	4.60	0.38	2.50
TinyImageNet	7823.50	4659.60	1438.15	208.15	1370.29	2.65	0.60	0.85

a unified, generalized solution. In simpler datasets, both retrained and unlearned models can converge to similar global minima on the remaining set, resulting in a smaller ED. However, for more complex datasets like TinyImageNet, all models end up stuck in different local minima on the remaining set, which leads to a larger ED between models.

Nevertheless, the above unusually high distance values do not affect the overall performance of our methods. Our methods still achieve the best performance in terms of ASR and ED, demonstrating their effectiveness in preserving privacy.

6.4 Unlearning Efficiency

Unlearning efficiency is particularly critical in applications where quick updates are necessary, such as in privacy-sensitive contexts or fast-evolving data environments. Table 5 compares the total time required for different unlearning methods. As shown in the table, both of our proposed methods demonstrate significant efficiency advantages over other unlearning techniques. For example, on the CIFAR-100 and TinyImageNet datasets, our PS method is 2,360 times and 9,176 times faster than traditional retraining, respectively. Furthermore, compared to the latest L2UL method, PS method achieves a speedup of 1.84 times on CIFAR-100 and 3.11 times on TinyImageNet. The remarkable efficiency of our approach is driven by two key factors. First, by focusing updates solely on the data in the unlearning set, we drastically reduce the number of samples processed in each batch, thereby accelerating the update process. Second, our method updates only the prototypes located in the final layer, which allows gradient computations to be limited to a single layer rather than propagating through the entire network. Unlike methods like L2UL, while also updating a subset of parameters, still require computing gradients across multiple layers to derive the necessary updates, besides, they further rely on adversarial samples, which add to the computational cost. In contrast, our prototype-based approach avoids such overhead by leveraging the unique position of prototypes in the final classification layer, enabling efficient updates with minimal computational effort.

Next, we discuss two distinct sets of data from the Table 5: Fisher and Naive PS method. Both methods do not require additional training. From the table, it is evident that our Naive PS method significantly outperforms Fisher unlearning in terms of time efficiency. This performance gap can be attributed to the fundamental differences in the computation processes of the two methods. Our approach simply deletes selected data from a pre-stored matrix and uses basic matrix multiplication to update the model parameters. In contrast, Fisher unlearning requires the estimation of the Fisher information matrix using all the data, a process that is computationally intensive and time-consuming. Therefore, the substantial time advantage of our method is a direct result of these computational differences. Overall, our proposed Naive PS method demonstrates remarkable efficiency in comparison to Fisher unlearning, making it a more viable solution for time-sensitive applications.

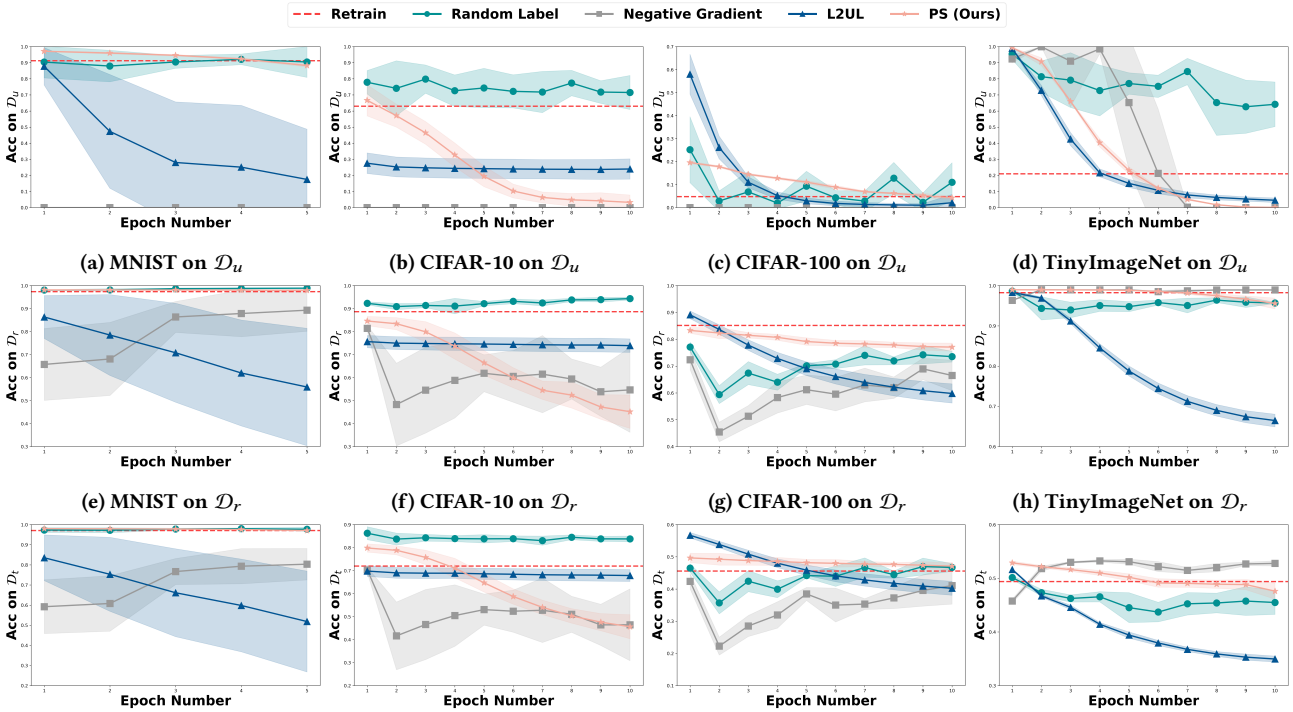
After exploring the efficiency of different unlearning methods, we next examine their stability, which is critical for ensuring consistent model performance. A good unlearning method should maintain stability across repeated experiments, meaning that the model's behavior should remain consistent with the same parameters. Figure 4 shows the accuracy variation curves for different unlearning methods across four datasets. The horizontal red line represents the accuracy of the retrained model, and the closer an unlearning method's accuracy curve is to this line, the more successful it is in unlearning the target data. Note that we did not include Fine-Tune results in the figure, as it requires an excessive number of epochs for updates, making it difficult to clearly show its performance and changes within the figure.

Overall, our proposed PS method demonstrates a relatively small variance in accuracy, quickly converging toward the retrained model's performance. Specifically, our method shows minimal variance across all datasets, indicating that the well-designed relabeled samples can bring about stable and fine-grained adjustments to the model. On the TinyImageNet dataset, our method's unlearning speed is second only to the L2UL method, which benefits from the use of adversarial samples. However, this advantage comes with trade-offs, as the L2UL method exhibits greater discrepancies between the remaining set and the test set. In contrast, our approach minimizes the impact on unrelated data, ensuring more stable performance.

6.5 Prototype Similarity

In order to evaluate the similarity between prototypes of the unlearned model and the retrained model, we focus on analyzing prototype shifts. Our method is grounded in observing prototype shifts, so it's crucial to ensure that the unlearning process maintains this consistency. By comparing the prototype similarities after unlearning, we can validate that our approach is functioning as intended. To visualize this similarity, we employ radar charts, as shown in Figure 5, where each point represents the PSC between the unlearned model and the retrained model for a specific prototype. Ideally, a high degree of similarity would indicate that the unlearned model closely aligns with the retrained model at the prototype level.

Examining the radar charts, our proposed methods demonstrate superior performance across multiple datasets, including MNIST, CIFAR-100, and TinyImageNet. On the CIFAR-10 dataset, while our

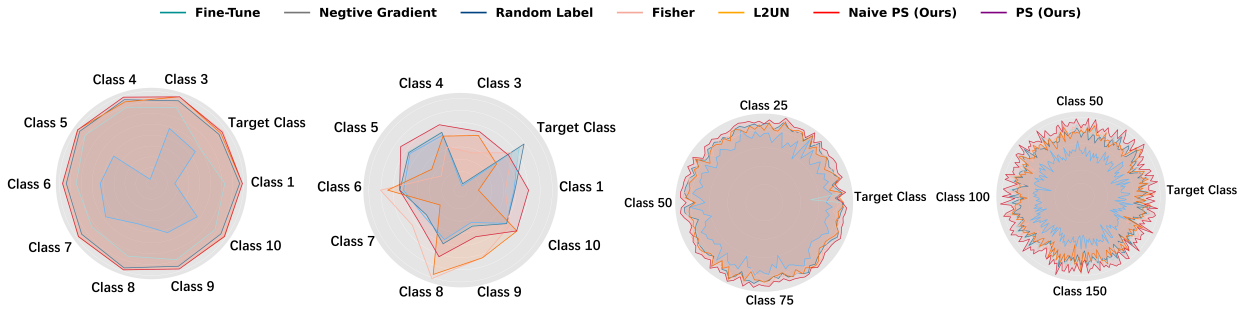


(a) MNIST on $\mathcal{D}_u$ (b) CIFAR-10 on $\mathcal{D}_u$ (c) CIFAR-100 on $\mathcal{D}_u$ (d) TinyImageNet on $\mathcal{D}_u$

(e) MNIST on $\mathcal{D}_r$ (f) CIFAR-10 on $\mathcal{D}_r$ (g) CIFAR-100 on $\mathcal{D}_r$ (h) TinyImageNet on $\mathcal{D}_r$

(i) MNIST on $\mathcal{D}_t$ (j) CIFAR-10 on $\mathcal{D}_t$ (k) CIFAR-100 on $\mathcal{D}_t$ (l) TinyImageNet on $\mathcal{D}_t$

Figure 4: Unlearning accuracy curves for different methods across datasets. Each column represents a dataset (MNIST, CIFAR-10, CIFAR-100, TinyImageNet) with rows depicting accuracy on $\mathcal{D}_u$, $\mathcal{D}_r$, and $\mathcal{D}_t$, respectively. The red dashed line indicates the performance of the retrained model, with closer proximity indicating better unlearning performance.



(a) MNIST (b) CIFAR-10 (c) CIFAR-100 (d) TinyImageNet

Figure 5: Prototype similarities between retrained model and unlearned models

methods do not achieve the highest PSC for certain classes, the overall PSC surpasses the second-best L2UL method by 0.076.

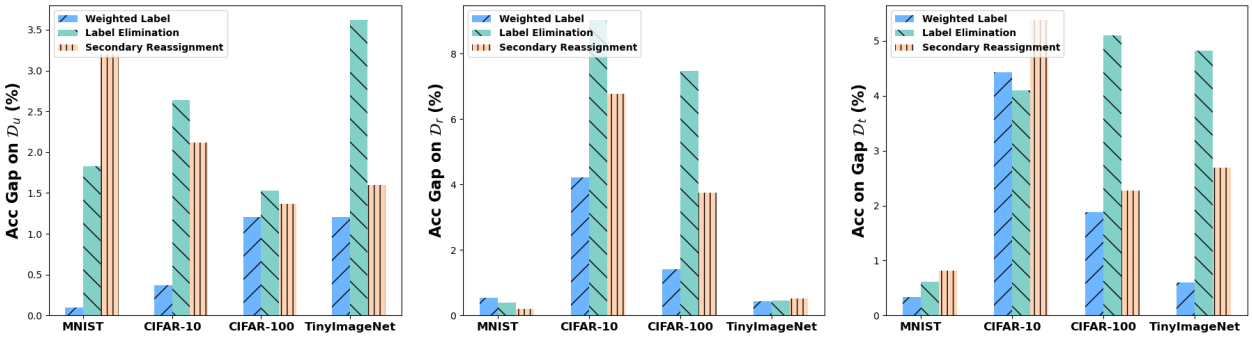
It is noteworthy that the PSC for the prototype of the target class in both Naive PS and PS is consistently lower than the average PSC across all four datasets. This observation aligns well with our expectation of more pronounced prototype shifts in the target class, further corroborating the effectiveness of our approach in managing prototype dynamics during the unlearning process.

6.6 Impact of Unlearn Ratios

We investigate the impact of varying the unlearn ratio on our methods. While our main results use an unlearn ratio of 10% for samples from one target class, we also evaluate performance at unlearn ratios of 5% and 15%. As shown in Table 6, our methods demonstrate a high degree of consistency across different unlearn ratios.

Table 6: Ablation study with varying unlearn ratios from 5% to 15%. The table presents accuracy (%), with the performance gap relative to the retrained model indicated in parentheses.

Dataset	Unlearn Ratio	Naive PS			PS		
		Acc on $\mathcal{D}_u$	Acc on $\mathcal{D}_r$	Acc on $\mathcal{D}_t$	Acc on $\mathcal{D}_u$	Acc on $\mathcal{D}_r$	Acc on $\mathcal{D}_t$
MNIST	5%	96.80 (10.00)	98.10 (0.05)	97.99 (0.81)	85.78 (1.01)	97.80 (0.35)	96.79 (0.38)
	10%	97.68 (6.39)	98.15 (0.71)	98.08 (0.92)	91.20 (0.01)	97.96 (0.53)	97.48 (0.33)
	15%	98.62 (2.75)	98.15 (0.10)	98.15 (0.15)	91.01 (4.86)	97.94 (0.10)	97.37 (0.63)
CIFAR-10	5%	72.64 (5.56)	82.08 (5.46)	78.10 (5.36)	61.10 (5.97)	80.54 (7.00)	75.85 (3.12)
	10%	70.39 (7.33)	78.17 (10.56)	74.44 (1.99)	63.43 (0.37)	84.52 (4.22)	76.87 (4.44)
	15%	89.08 (43.38)	82.58 (3.26)	80.23 (7.12)	48.80 (3.09)	78.79 (7.05)	75.35 (2.24)
CIFAR-100	5%	48.66 (42.37)	72.28 (9.94)	52.22 (4.75)	3.71 (2.57)	80.52 (8.70)	51.89 (4.42)
	10%	53.28 (48.41)	81.74 (3.44)	54.25 (8.65)	3.67 (1.20)	82.73 (2.45)	47.48 (1.88)
	15%	56.61 (55.49)	77.47 (10.86)	55.90 (7.24)	8.03 (6.91)	78.45 (9.88)	53.77 (5.11)
TinyImageNet	5%	98.40 (73.95)	99.97 (0.03)	52.73 (6.42)	31.73 (14.57)	99.37 (0.62)	49.81 (3.51)
	10%	98.70 (77.63)	98.87 (0.39)	52.78 (3.38)	22.27 (1.21)	99.70 (0.44)	48.80 (1.4)
	15%	97.37 (76.72)	99.98 (0.01)	53.48 (6.52)	14.65 (6.00)	99.83 (0.16)	50.77 (3.81)



(a) Results on $\mathcal{D}_u$ **(b) Results on $\mathcal{D}_r$** **(c) Results on $\mathcal{D}_t$**

Figure 6: Ablation study of relabel strategies across different datasets: The primary method used is Weighted Label, with Label Elimination and Secondary Reassignment explored as alternative strategies. The values presented in the figures represent the gap, where lower values indicate better performance.

The Naive PS method performs well on smaller datasets like MNIST and CIFAR-10, but its performance varies significantly under certain conditions. For instance, at the 15% unlearn ratio on CIFAR-10, there is a notable accuracy drop on the unlearning set, which highlights the limitations of its simplified approach. This behavior suggests that Naive PS struggles to maintain unlearning effectiveness as the target ratio increases, likely due to the insufficient parameter update strategy when dealing with higher unlearning ratios. Similarly, on CIFAR-100 and TinyImageNet, Naive PS achieves reasonable performance but exhibits more pronounced performance gaps, especially on $\mathcal{D}_u$, as the unlearn ratio grows.

In contrast, the improved PS method consistently performs well across all conditions. For example, on TinyImageNet, the PS method maintains minimal accuracy gaps of 14.57%, 1.21%, and 6.00% on the $\mathcal{D}_u$ at 5%, 10%, and 15% unlearn ratios, respectively. Furthermore, the PS method preserves strong accuracy on $\mathcal{D}_r$ and $\mathcal{D}_t$ across all datasets and unlearn ratios, demonstrating its robustness and adaptability. These results confirm that the PS method remains reliable and stable even under higher unlearn ratios, making it a robust choice for diverse unlearning scenarios.

6.7 Ablation Study on Relabeling Methods

The experiment focuses on an ablation study of our relabeling method. We evaluate three different strategies: our Weighted Relabel method, and two alternative approaches. The first alternative involves deleting the highest probability class after obtaining initial predictions and using the remaining probabilities to assign new label. The second alternative assigns the second-highest predicted class as the new label directly. For example, if a sample's prediction probabilities are [0.6, 0.3, 0.1] for three classes, the second strategy would remove the 0.6 probability and the new soft label would be [0, 0.3, 0.1], while the third strategy would simply take the class corresponding to 0.3 as the new hard label.

Our results for different strategies are summarized in Figure 6, indicating that the Weighted Relabel method achieves the closest performance to the retrained model across all datasets. This is especially evident in the accuracy gap on the unlearning set, where the Weighted Relabel method consistently outperforms the other two strategies.

We attribute this success to the additional information provided by the soft labels in the weighted version, which better guides the model's prototype updates. In contrast, the deletion strategy often leaves very small probabilities for the remaining classes, leading to poor guidance for the model. Similarly, assigning the second-highest predicted class as the label disregards valuable information from the original prediction, resulting in less effective unlearning.

7 Conclusion

In this paper, we have introduced a prototype-based framework for machine unlearning, comprising two methods: Naive PS and PS. Naive PS utilizes a closed-form solution to approximate the unlearning process efficiently, while PS refines this approach by leveraging soft-labeled data to guide prototype updates with greater precision. Both methods focus exclusively on modifying the prototypes in the final classification layer, significantly reducing computational overhead and memory consumption while keeping the rest of the model unchanged. Experimental results on four benchmark datasets and four neural network architectures demonstrate that our methods achieve unlearning in a significantly faster manner while maintaining performance on par with existing approaches. Furthermore, by directly addressing the core unlearning task without extensive retraining or large-scale parameter updates, our methods drastically reduce the time and memory requirements compared to traditional unlearning techniques. These properties make Naive PS and PS particularly suitable for scenarios requiring efficient and scalable machine unlearning.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 62272183, 62071192 and 62171189; by the National Key R&D Program of China under Grant 2024YFE0103800; by the UKRI EPSRC Grant EP/X035085/1; and by the Key R&D Program of Hubei Province under Grants 2024BAA008, 2024BAA011, 2024BAB016, 2024BAB031 and 2023BAB074.

References

[1] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine unlearning. In *Proceedings of IEEE S&P*. 141–159.

[2] Jonathan Brophy and Daniel Lowd. 2021. Machine unlearning for random forests. In *Proceedings of ICML*. 1092–1104.

[3] Yinzhi Cao and Junfeng Yang. 2015. Towards making systems forget with machine unlearning. In *Proceedings of IEEE S&P*. 463–480.

[4] Sungmin Cha, Sungjun Cho, Dasol Hwang, Honglak Lee, Taesup Moon, and Moontae Lee. 2024. Learning to unlearn: Instance-wise unlearning for pre-trained classifiers. In *Proceedings of AAAI*. 11186–11194.

[5] Min Chen, Weizhuo Gao, Gaoyang Liu, Kai Peng, and Chen Wang. 2023. Boundary unlearning: Rapid forgetting of deep networks via shifting the decision boundary. In *Proceedings of IEEE CVPR*. 7766–7775.

[6] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. 2022. Graph unlearning. In *Proceedings of ACM SIGSAC*. 499–513.

[7] Vikram S Chundawat, Ayush K Tarun, Murari Mandal, and Mohan Kankanhalli. 2023. Can bad teaching induce forgetting? unlearning in deep networks using an incompetent teacher. In *Proceedings of AAAI*. 7210–7217.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of IEEE CVPR*. 248–255.

[9] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE signal processing magazine* 29, 6 (2012), 141–142.

[10] Yann Fraboni, Martin Van Waerebeke, Kevin Scaman, Richard Vidal, Laetitia Kameni, and Marco Lorenzi. 2024. SIFU: Sequential Informed Federated Unlearning for Efficient and Provable Client Unlearning in Federated Optimization. In *Proceedings of AISTATS*. 3457–3465.

[11] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of ACM CCS*. 1322–1333.

[12] Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. 2019. Making ai forget you: Data deletion in machine learning. In *Proceedings of NeurIPS*, Vol. 32.

[13] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. 2020. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of IEEE CVPR*. 9304–9312.

[14] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. 2020. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. In *Proceedings of ECCV*. 383–398.

[15] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. 2020. Certified data removal from machine learning models. In *Proceedings of ICML*. 3832–3842.

[16] Tomohiro Hayase, Suguru Yasutomi, and Takashi Katoh. 2020. Selective forgetting of deep networks at a finer level than samples. *CoRR, arXiv:2012.11849* (2020).

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of IEEEF CVPR*. 770–778.

[18] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S Yu, and Xuyun Zhang. 2022. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)* 54, 11s (2022), 1–37.

[19] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR, arXiv:2001.08361* (2020).

[20] Hyunjune Kim, Sangyong Lee, and Simon S Woo. 2024. Layer Attack Unlearning: Fast and Accurate Machine Unlearning via Layer Level Attack and Knowledge Distillation. In *Proceedings of AAAI*. 21241–21248.

[21] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report. University of Toronto, Technical Report.

[22] Meghdad Kurmanji, Peter Triantafillou, Jamie Hayes, and Eleni Triantafillou. 2024. Towards unbounded machine unlearning. In *Proceedings of NeurIPS*.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*. 2278–2324.

[24] Gaoyang Liu, Xiaoqiang Ma, Yang Yang, Chen Wang, and Jiangchuan Liu. 2021. Federaser: Enabling efficient client-level data removal from federated learning models. In *Proceedings of IEEE IWQOS*. 1–10.

[25] Gaoyang Liu, Tianlong Xu, Xiaoqiang Ma, and Chen Wang. 2022. Your model trains on my data? Protecting intellectual property of training data via membership fingerprint authentication. *IEEE Transactions on Information Forensics and Security* 17 (2022), 1024–1037.

[26] Jiancheng Liu, Parikshit Ram, Yuguang Yao, Gaowen Liu, Yang Liu, PRANAY SHARMA, Sijia Liu, et al. 2024. Model sparsity can simplify machine unlearning. In *Proceedings of NeurIPS*, Vol. 36.

[27] Yi Liu, Lei Xu, Xingliang Yuan, Cong Wang, and Bo Li. 2022. The right to be forgotten in federated learning: An efficient realization with rapid retraining. In *Proceedings of IEEE INFOCOM*. 1749–1758.

[28] Nils Lukas, Ahmed Salem, Robert Sim, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. 2023. Analyzing leakage of personally identifiable information in language models. In *Proceedings of IEEE S&P*. IEEE, 346–363.

[29] Ronak Mehta, Sourav Pal, Vikas Singh, and Sathya N Ravi. 2022. Deep unlearning via randomized conditionally independent hessians. In *Proceedings of IEEE CVPR*. 10422–10431.

[30] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. 2019. When does label smoothing help?. In *Proceedings of IEEE S&P*.

[31] Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillet. 2020. Variational Bayesian Unlearning. In *Proceedings of NeurIPS*, Vol. 33. 16025–16036.

[32] Quoc Phong Nguyen, Ryutaro Oikawa, Dinil Mon Divakaran, Mun Choon Chan, and Bryan Kian Hsiang Low. 2022. Markov chain monte carlo-based machine unlearning: Unlearning what needs to be forgotten. In *Proceedings of ACM ASIA CCS*. 351–363.

[33] Thanh Tam Nguyen, Thanh Trung Huynh, Zhao Ren, Phi Le Nguyen, Alan Wee-Chung Liew, Hongzhi Yin, and Quoc Viet Hung Nguyen. 2022. A survey of machine unlearning. *CoRR, arXiv:2209.02299* (2022).

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of NeurIPS*.

[35] Shahbaz Rezaei and Xin Liu. 2021. On the difficulty of membership inference attacks. In *Proceedings of IEEE CVPR*. 7892–7900.

[36] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *Proceedings of IEEE S&P*. 3–18.

[37] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR, arXiv:1409.1556* (2014).

[38] Vinith Suriyakumar and Ashia C Wilson. 2022. Algorithms that approximate data removal: New results and limitations. In *Proceedings of NeurIPS*. 18892–18903.

[39] Paul Voigt and Axel Von dem Bussche. 2017. The EU general data protection regulation (GDPR). *A Practical Guide* (2017).
[40] Xiaodong Wu, Ran Duan, and Jianbing Ni. 2024. Unveiling security, privacy, and ethical concerns of ChatGPT. *Journal of Information and Intelligence* (2024).
[41] Yinjun Wu, Edgar Dobriban, and Susan Davidson. 2020. Deltagrad: Rapid retraining of machine learning models. In *Proceedings of ICML*. 10355–10366.
[42] Haonan Yan, Xiaoguang Li, Ziyao Guo, Hui Li, Fenghua Li, and Xiaodong Lin. 2022. ARCANE: An Efficient Architecture for Exact Machine Unlearning. In *Proceedings of IJCAI*. 19.
[43] Chang-Bin Zhang, Peng-Tao Jiang, Qibin Hou, Yunchao Wei, Qi Han, Zhen Li, and Ming-Ming Cheng. 2021. Delving deep into label smoothing. *IEEE Transactions on Image Processing* 30 (2021), 5984–5996.
[44] Haibo Zhang, Toru Nakamura, Takamasa Isohara, and Kouichi Sakurai. 2023. A review on machine unlearning. *SN Computer Science* 4, 4 (2023), 337.
[45] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. 2020. The secret revealer: Generative model-inversion attacks against deep neural networks. In *Proceedings of IEEE CVPR*. 253–261.